Right Move
Augmented Reality Project

Department of Electronics, Telecommunications, and Informatics
University of Aveiro
Supervisor: Professor Joaquim Madeira

Beatriz Rodrigues (93023) bea.rodrigues@ua.pt,
Catarina Barroqueiro (103895) cbarroqueiro@ua.pt,
Ricardo Covelo(102668) ricardocovelo11@ua.pt,
Rui Campos (103709) ruigabriel2@ua.pt,
Telmo Sauce (104428) telmobelasauce@ua.pt

June 2023

Abstract

Our Project in Computer Engineering and Informatics has as its main objective to develop an application that makes use of Augmented Reality and helps the user to carry out a sequence of instructions related to the assembly of Legos. The user will have several boxes available. Each box is identified with a QR Code and will contain a type of LEGO block, with different colors and sizes available. The user's purpose is to carry out a given assembly and the application will guide him in order to track both the collection of LEGO Blocks (verifying shape, size, and color) and the position where they were subsequently placed. For this we use the 'Google Glasses Enterprise Edition 2', kindly provided by Huf Portuguesa, which has been supporting us in the development of this project.

Resumo

O nosso Projeto em Engenharia de Computadores e Informática tem como principal objetivo desenvolver uma aplicação que faça uso da Realidade Aumentada e auxilie o utilizador a realizar uma sequência de instruções relacionadas com a montagem de Legos, com vista a aplicar no futuro a mesma metodologia a montagens do foro industrial. O utilizador terá disponíveis várias caixas. Cada caixa é indentificada com um QR Code e conterá um tipo de peça, estando disponíveis diversas cores e tamanhos. A finalidade do utilizador é realizar uma dada montagem e a aplicação irá orientá-lo de modo a rastrear tanto a recolha das peças(verificando forma, tamanho e cor), como a posição onde estas foram posteriormente colocadas. Para isso, além dos dispositivos mais comuns (tablets e smartphones) temos também uma versão da app mais imersiva desenvolvida especificamente para os 'Google Glasses Enterprise Edition 2', gentilmente cedidos pela Huf Portuguesa, que nos tem vindo a apoiar no desenvolvimento deste projeto. Todo o material de suporte ao projeto está disponivel em RightMoveWebsite

Acknowledgements

We are grateful to Huf Portuguesa for making available the Google Glasses used in our project.

We also thank the availability, support and valuable suggestions of Engineer David Pinheiro, Engineer Jorge Silva, and their colleagues at Huf Portuguesa.

Contents

Glossary

API Application Programming Interface

AR Augmented Reality

HTTP Hypertext Transport or Transfer Protocol

IL Immersive Learning

MR Mixed Reality

SDK Software Development Kit

SOAP System Object Access Protocol

VR Virtual Reality

XR Extended Reality

Chapter 1

Introduction

In this chapter, we delve into the motivation, objectives, anticipated results, document structure, and the team involved in this project. By providing a comprehensive overview, we set the stage for a thorough exploration of the project's scope and significance.

1.1 Motivation

With the evolution of industry 4.0, every day new technologies extend our reality, with **AR**, **VR**, and **MR**. This **XR** mechanisms allow us to optimize production on many levels: **Immersive Learning** and **AR**-based guidance in product assembly and repair.

Our motivation is to develop a solution for any user that wants to do assembly work, like an operator. Although industrial processes are increasingly automated, components are assembled in factories, such as Huf Portuguesa, which produces automotive components, where a large part of the assembly lines are manual, as we saw when we visited the factory. To prevent significant production breaks, worker training in industrial environments needs to be fast and effective, so carrying out the training of workers autonomously and independently of supervisors would bring added value to the efficiency of production in any industry. Autonomous training enables more efficient integration of operators into the assembly line, minimizing production delays caused by insufficient instruction. It also enables the simultaneous training of multiple workers.

Then, using LEGO we recreated an environment that helps the untrained personnel better perform their tasks and recognize their errors without supervision.

Augmented Reality is getting better by the day and is increasingly being more and more used in applications and this project will help us understand better the potential of **XR**-based technologies, while we are developing the application. Although it is aimed at the assembly of LEGOs, we have in mind possible future applications at an industrial level, which could have an impact on the efficiency of manual assembly processes.

Our project was proposed by **Huf Portuguesa**, which required us to use **Google Glasses** as our primary device for implementing our solution. However, we faced several challenges during the development process due to the limitations of the device.

Firstly, we encountered issues with the version of Android used on Google Glass, which was outdated and lacked support for some of the latest features and technologies. This meant that we had to modify our solutions to work within the constraints of the outdated Android version.

Additionally, the camera resolution on the Google Glass was **limited**, and the hardware itself was **not very powerful**. This posed a significant challenge when it came to tasks such as reading QR codes, as the codes were often only visible at very close distances, making them difficult to capture and process accurately.

Despite these challenges, we were able to adapt our solutions to work effectively with Google Glass. We explored alternative approaches, such as using external devices and integrating with other software systems, to overcome the limitations of the device.

1.2 Goals

Our **primary objective** is to create an application that assists users during LEGO assembly, providing step-by-step guidance and notifying them whenever a step is not executed correctly. In order to **train** the user to perform the assembly each time faster and more independently. With a future goal of transferring to the **assembly of parts** in an **industrial environment**.

In the initial phase, our objective is to enable users to scan the intended **assembly** directly from the **catalog** available on the website. This allows users to access the **list of blocks** they need to collect. To track their progress, users will **scan** the QR code on each box of the collected block. The QR code contains information such as the block's ID, color, and dimensions. The application will then verify whether the scanned block is the correct one to scan **in the designated order**.

In the second phase, the user must **position** the LEGO block correctly at the

indicated coordinates, and later the application will verify the position of the piece and **recognize** all the pins on the green board, which will work as a basis for all constructions.

Initially, we aimed, in a **controlled scenario**, based on **Aruco Markers**, to define the position of the green board where we positioned the various LEGO Blocks necessary for the selected assembly, to identify all the pins on the board in order to track the color present in each of these pins. However, we have implemented a **different algorithm** discussed in detail in Section "Detecting using HoughCircles". This decision was made because the Aruco Markers algorithm would introduce calculation errors in the location of the pins when we later attempt to build assemblies vertically.

In the third phase, we introduced a functionality where the application the user has placed the correct piece by **analyzing its RGB color** and checking its orientation. Upon successful verification of color and orientation, the user is **permitted to proceed** with the assembly process.

In the Fourth phase, in order to give the user more freedom, we built an **online assembly system**, from which it is possible to extract a QR code containing a JSON that can be scanned with the app where the user can assemble the construction he wants to train and thus build his own catalog.

In the final phase, our goal is to **optimize the project** by implementing various improvements. This includes enhancing the overall **speed** and **performance** of the application, refining the user interface to ensure **better usability**, and incorporating design changes to enhance the overall user experience. By focusing on these optimizations, we aim to create a more efficient and user-friendly application.

1.3 Results

The project has yielded promising results, with the creation of **three different prototypes** designed to enhance the LEGO assembly process.

The **first prototype** involves detecting coordinates using **Aruco markers**. This prototype uses a sophisticated computer vision system to identify the location of the LEGO blocks. By detecting the Aruco markers on each block, the system can quickly and accurately assemble the blocks into the desired structure.

The **second prototype** uses the **Hough Circles** function to detect coordinates, instead of using **Arucos markers**. This method relies on detecting circular shapes in the LEGO blocks to determine their location.

Finally, the **third prototype** is **designed specifically** for use with **Google Glasses** or for small displays. In this version, our focus was on improving the user interface to enhance the overall usability of the application. Specifically, we made modifications to ensure that both the text and QR codes are clearly visible on the small display of the glasses.

In addition to the three prototypes developed by the RightMove project, our RightMove **RightMoveWebsite** offers a range of additional features to enhance the LEGO building experience. One notable feature is our catalog of pre-designed LEGO constructions, which provides users with a diverse selection of assemblies to build using our technology.

We also adapted a feature called **Brick Builder**, which allows users to create their own custom Lego constructions. This feature provides a range of tools and resources that enable users to design and build their own unique creations. With Brick Builder, users can unleash their creativity and build anything they can imagine.

1.4 Project Team

RightMove is composed of 5 elements.

Name	Role	Function
Beatriz Rodrigues	WebSite Developer	Develop the Website
Catarina Barroqueiro	Team Manager/ Developer	Responsible for setting targets. Designed and Implemented the product software program Develop the QR Code Detection App and the Scenario
Ricardo Covelo	Software Architect Developer	Designed and Implemented the product software program. Develop the Coordinates Detection App, HoughCircles function
Rui Campos	Developer/ Writer	Assisting with functions and any issues the team may need. Do the write reports and documentation.
Telmo Sauce	Software Architect Developer	Designed and Implemented the product software program Develop the Color Detection App

1.5 Document Structure

In our report, we have structured the content into **four main sections**: the state of the art, system requirements, system architecture, and implementation.

The **state of the art** section provides an overview of the current state of technology and research related to our project. This includes a review of existing solutions, methodologies, and tools in the relevant domain.

The **system requirements** section outlines the functional and non-functional requirements of our system. These requirements describe what the system should be able to do, as well as any constraints or limitations on its performance, security, usability, and other characteristics.

The **system architecture** section describes the high-level design of our system, including its components, interfaces, and interactions. This section provides a conceptual framework for understanding how the system works, and how its various parts fit together.

Finally, the **implementation** section describes the actual development of our system, including the technologies, programming languages, and tools used to build it. This section provides a detailed account of the steps taken to translate the system requirements and architecture into a functional system.

Chapter 2

State of the Art

The state of the art analysis is a crucial component of our project, as it provides a comprehensive understanding of the current advancements and existing research in the field. By examining the state of the art, we can identify the gaps, challenges, and opportunities that will guide our project's direction and contribute to its overall significance.

2.1 Introduction

From the beginning of the development until the present moment, the project has undergone several changes. Initially, we did not have any type of information about the material that would later become available to us. Consequently, we began **to research** in order to explore which tools have been used to develop projects **where AR is used in industrial environments**. This research was then divided into **two moments**, an initial one in which we did not know what materials we would have available, and a second in which we began to be accompanied by members of **Huf Portuguesa** who gave us a more precise idea of where we should start and the objectives to be fulfilled taking into account our project.

In the overall phases of the research, we **consulted several articles, papers**, and we also attended the International Conference on Graphics and Interaction – **ICGI'2022**. Given all this research, we found quite relevant articles that deepened our understanding of Augmented Reality but also identified the tools necessary to effectively work with this technology. We also came across an article [SAP22], which emphasized the **need to update operator training** due to the recurring nature of innovation and new product development. This need was further reinforced during our visit to the Huf factory. In order to obtain more immediate results, submitting people to training using resources such as **AR** is a strategy that is widely adopted

today since immersive teaching has **better results** when compared to **traditional teaching** methods (as exposed in the article [CRURCC22]). We researched many papers and almost all of them used expensive technology that was not available to us. So another part of the challenge was to develop a low-cost solution, by focusing on cost-effectiveness, we aimed to create a solution that could be easily adopted by HUF Portuguesa in their operations on a larger scale.

2.2 AR Environment

Augmented reality (AR) is the integration of digital information with the user's environment in **real-time**. Unlike virtual reality (VR), which creates a totally artificial environment, AR users experience a real-world environment with generated perceptual information overlaid on top of it.

In our project, we utilized **'Google Glasses'** to create the AR environment. This technology allowed us to superimpose computer-generated visuals onto the user's field of vision, enhancing their perception and interaction with the surrounding real-world environment.

All of this becomes advantageous for the evolution of the current world when applied to time-consuming processes in order to optimize them, for several tools are used (as explained in the article [DPN+22]) such as Unity3D (this being one of the most frequently used in the analyzed papers), the Vuforia, among others now explained.

During our research, we explored **Unity 3D** and **the Vuforia Engine** to understand their use in AR development. However, we found that the existing libraries in Android Studio **provided similar functionalities**, making Unity 3D and Vuforia unnecessary for our project. The Android Studio libraries offered tools for developing AR environments and facilitating user-application communication. Therefore, we utilized these libraries to achieve our project goals without relying on external applications.

2.3 Google Glasses

Google Glasses were designed to be lightweight and are equipped with a camera and sensors, that help in reading QR codes and detecting what the user is seeing. Transforming the data into something helpful for the user. It also comes with a small

screen in the top right corner where the user can see the output intended. These glasses need to use a google library. with multiple functions to manage all the devices available by the glasses.

Although the use of AR Glasses is more frequent (as for example in the project described in [MZV17]), Google Glasses have the advantage that, even accompanying the user, they do not provide a fully immersive environment that makes the user feel abstract from reality in the present moment. Therefore, it is a more suitable tool for projects where the objective is to perform a task in real-time and not simulate an immersive environment.

To understand how we could start to develop our project, we visualized some projects with a structure similar to ours. Ultimately, we discovered a project where users utilized LEGO blocks on a green board **to create a U-shaped design** ([SJL22]).

2.4 Related Projects

When it comes to developing a project, one of the most important aspects is the ability to read and interpret project documents. Whether it's a blueprint, a schematic, or a technical drawing, being able to understand the details and specifications of a project is crucial for its success. It's important to take the time to study and analyze these documents thoroughly in order to gain a clear understanding of what needs to be done.

As part of our research process, we maintained a detailed record of all the research we did in Excel files in Search Log where we highlighted the key points of each article consulted, such as the architecture used, programming languages, Software Packages, devices, evaluation and success of the article.

By recording all this information relating to various projects within our area of interest in tables, we were able to highlight the most relevant points of each one and extract some useful information for the further development of our project. Of all the articles and projects studied, we highlight 5 below:

Title	Goals	Devices	Software Packages
Augmented Reality Application to Develop a Learning Tool for Students: Transforming Cellphones into Flashcards	Making learning with flashcards more efficient and intuitive	Iphones Android Mobile Devices	Unity3D
Augmented reality application to support remote maintenance as a service in the Robotics industry	Allows a non-specialized technician to repair an assembly line machine with the help of an expert and AR 3d models.	Smartphones/Tablets AR googles	Unity 3D
A Novel Augmented Reality Mobile-Based Application for Biomechanical Measurement	Develop an application capable of evaluating posture and range of human movements with the help of markers	Smartphones	OpenGL
Comparative study of AR software development kits (SDK's)	Know the difference between augmented reality and virtual reality; know the working of AR system; Related work done using several AR SDK's; Augmented reality SDK's along with their features and comparison of AR SDK's based on different criteria	Android/IOS	Unity 3D/Windows
WebPoseEstimator: A Fundamental and Flexible Pose Estimation Framework for Mobile Web AR	Based on a camera and some sensors, it aims, based on APIs developed in accessible languages, to track movements and consequently show them in a virtualized environment.	Camera/Smartphone/ IMU Sensor	MSF C++ Source Code

We have chosen to highlight five specific articles. These articles **closely align with our objectives**, considering the available materials and our recently defined goals. They share similarities with our desired development, focusing on optimizing learning, reducing task completion time, and integrating Augmented Reality.

In addition, we aim to eventually implement our project in an industrial environment, assisting workers in their tasks. After consulting these articles, we discovered that similar technologies are already being adopted by the industry and that it may one day be possible to **complete our objectives**.

Waking App

Waking App is a virtual and augmented reality software company. Its goals are to allow Autodesk Revit and Fusion 360 users to transform their 3D projects into augmented reality and virtual reality, through a smartphone or tablet. So, using either a smartphone or a tablet and, through an application, engineers can enter their projects and receive a more real and tangible experience of their creations. This project is related to ours due to the use of virtual and augmented reality which will also be a very used theme in our project, mainly due to the use of Google Glasses.

WebPoseEstimator

WebPoseEstimator uses the fundamental and flexible pose estimation framework for mobile web AR. Thus, the objective of this project is, based on a camera and some sensors and based on APIs developed in accessible languages, to track movements and consequently show them in a virtualized environment. It uses a camera, a smartphone, and an IMU sensor. After being evaluated, the tool developed had excellent results, it should be noted that it was developed adapting easily accessible languages, making the tracking of movements/poses more accessible to anyone.

This project is related to ours due to the use of Augmented Reality.

Automatic Navigation and Landing of an Indoor AR

Automatic Navigation and Landing of an Indoor AR use drone quadrotor using Aruco marker and inertial sensors. Its objectives are to ground drones without human intervention in a specific location inside buildings. It uses Arucos, low-cost drones, an IMU sensor, and finally a vision sensor. This method has acceptable results, especially considering that the drone is low-cost. This project is related to ours mainly due to the use of Arucos to make a limit in terms of space. In our project, it is the space for placing the legos, whereas in the Automatic Navigation and Landing of an Indoor AR project, the objective is to land the drone in the space covered by the Arucos.

Brick Builder

The Brick Builder tool available at BrickBuilderWebSite is a fantastic resource for anyone interested in creating unique and customizable LEGO constructions. This tool allows users to easily and intuitively design their own creations, selecting from a wide variety of different LEGO blocks, colors, and sizes to create something truly unique.

What makes the BrickBuilder particularly appealing for our project is the fact that it allows users to assemble their LEGO creations directly within the tool itself. This means that users can see exactly how their creations will look and how they will fit together, without having to physically build and disassemble the model multiple times. This saves time and effort and allows users to experiment and explore different design options with ease.

Overall, the BrickBuilder is a valuable addition to our project, as it offers an intuitive and user-friendly way to create and design LEGO constructions. By allowing

users to build and experiment within the tool itself, it streamlines the process of creating and testing different designs and makes it easier than ever to bring your LEGO creations to life.

2.5 Software Frameworks

Android Studio

Android Studio is the official integrated development environment (IDE) for Android application development. It is based on IntelliJ IDEA, a Java-integrated development environment for software, and incorporates its code editing and developer tools.

To support application development within the Android operating system, Android Studio uses a Gradle-based build system, Android Emulator, code templates, and GitHub integration. Every project in Android Studio has one or more modalities with source code and resource files. These modalities include Android app modules, Library modules, and Google App Engine modules.

Android Studio uses an Apply Changes feature to push code and resource changes to a running application. A code editor assists the developer with writing code and offering code completion, refraction, and analysis. Applications built in Android Studio are then compiled into the APK format for submission to the Google Play Store.

Throughout our project, we **heavily relied** on these frameworks to support us in developing the app, both in terms of designing the layout and implementing the necessary logic. These frameworks provided us with essential tools and resources that facilitated the development process and enabled us to create a robust and functional application.

OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products.

One of the most important libraries in OpenCV is the Core Library, which provides basic data structures and functions for image processing, including reading and writing images, manipulating pixels, and transforming images.

Another important library is the **High-level GUI Library**, which provides a simple interface for displaying images and capturing input from cameras or video files.

The Image Processing Library is another important part of OpenCV, providing a wide range of functions for image processing, including filtering, segmentation, feature detection, and object recognition.

The Video Analysis Library is another important library in OpenCV, providing tools for analyzing video streams and performing tasks such as motion detection and tracking.

OpenCV played a crucial role in our project, particularly in tasks such as color detection and pin identification in both Aruco markers and Hough circles.

2.6 Conclusion

In conclusion, the State of Art phase played a pivotal role in the progress of the project, considering its exploratory nature and the lack of knowledge among team members regarding the current practices employed by factories in the application of AR and XR to these environments. Through extensive reading and research on successful previous projects and tools utilized for this purpose, we acquired a wealth of knowledge and resources at our disposal. However, our primary constraint throughout the process was the limitation imposed by the use of Google Glasses. Despite this restriction, the State of Art phase proved to be instrumental in equipping us with the necessary insights and understanding to forge ahead in our project.

Chapter 3

System Requirements

The system requirements analysis is a fundamental step in our project, as it lays the groundwork for defining the necessary features, functionalities, and constraints of the system we aim to develop. By thoroughly examining the requirements, we can ensure that our project aligns with the desired objectives and meets the expectations of its intended users. In this chapter, we delve into the specific requirements that our LEGO system must fulfill.

3.1 Requirements Elicitation

As we observed during the factory visit, when operators are on the assembly line, they retrieve the LEGO Blocks and place them in the correct location following a specific order. In our project, the same process occurs, but with the use of **LEGO blocks**, which simplifies the process to some extent. Possible scenarios with some resemblance to the factory scenario were discussed within the operations carried out on the assembly line, but in the end, this seemed the most appropriate.

3.2 Actors

Common users who have the hobby to assemble LEGO, or companies that manufacture them, will act as **actors** in the developed project. On one hand, customers with the objective of **completing a construction project** strive to enhance their **efficiency** by utilizing the application to assemble the components, following the provided instructions. On the other hand, companies manufacturing such products can utilize the application to **enhance the customer experience** by creating as-

sembly instructions for users to later utilize. This entire process can be applied to different situations in industrial settings, for example, on a factory floor, where much of the work is done manually. Its implementation would provide considerable value by assisting operators in performing tasks with enhanced monitoring and facilitating the **automation of their movements and steps**, consequently increasing overall efficiency and speed.

It is a very versatile application that can evolve and be adapted to various scenarios, where someone has to be instructed step by step to carry out a given task, thus having a wide range of possible actors.

Our project is a **proof of concept**, that is, it serves to show that someone can do what a company wants in their day-to-day work. In our case, that company is **'Huf Portuguesa'**.

If the user accesses our platform through our website, he will have the ability to create his own **unique** assembly that can then execute using our application. This allows for a more personalized experience and enables you to bring user ideas to life.

On the other hand, if the user accesses our application directly, he will have access to our vast **catalog** of pre-existing designs. This means that the user can select from a wide range of options and build one of these structures with ease.

3.3 Use Cases

In this project, we aim to explore and implement use cases that demonstrate the practical applications and benefits of our solution. These use cases serve as real-world examples that highlight the value and potential of our project, providing tangible evidence of its capabilities and advantages.

- **LEGO assembly:** The system correctly guides the user through the assembly of the LEGO blocks. The system waits for the user to collect the appropriate LEGO block to advance and then waits for the user to place that LEGO block in the correct place to advance to the next LEGO block.

- **Assembly creation:** The system correctly guides the user to make his own assembly on the website.

3.4 Functional Requirements

Functional requirements are the backbone of any project, defining the specific features, capabilities, and behaviors that the system or solution must possess to fulfill its intended purpose. These requirements serve as a blueprint, outlining the essential functionalities that our project will deliver to its users.

- The App must **show** the initial menu with 3 initial options to choose;
- The App must **show** a picture of the Lego block scanned by the user.
- The App must **inform** the user in what box the next Lego block should be taken
- The App must **recognize** when the wrong piece was scanned
- The App must **inform** the user the location and direction of the Lego block
- The App must **recognize** if the Lego block is well placed
- In the WebSite: the BrickBuilder must **allow** the user to make their own assembly

3.5 Non-functional Requirements

In addition to the functional requirements that define what a system or solution should do, non-functional requirements play a crucial role in shaping the overall quality, performance, and user experience.

- **Usability:** The app should be intuitive and easy to use.
- **Resilience:** In case of crashes, whether related to software or hardware, it should have the capability to quickly recover and resume operation without significant downtime.
- **Efficiency:** The recognition of the Lego LEGO block should be quick and efficient. The recognition and recovery of the QR codes must be done in an adequate time frame.
- **Maintainability:** The system must be constructed with its potential future growth in mind.

Chapter 4

System Architecture

The system architecture serves as the foundation for our project, providing a **structured** and **organized** framework that defines how different components, modules, and subsystems will interact and work together to achieve our goals. It establishes the overall design and structure of the system, ensuring its **scalability**, **reliability**, and **efficiency**.

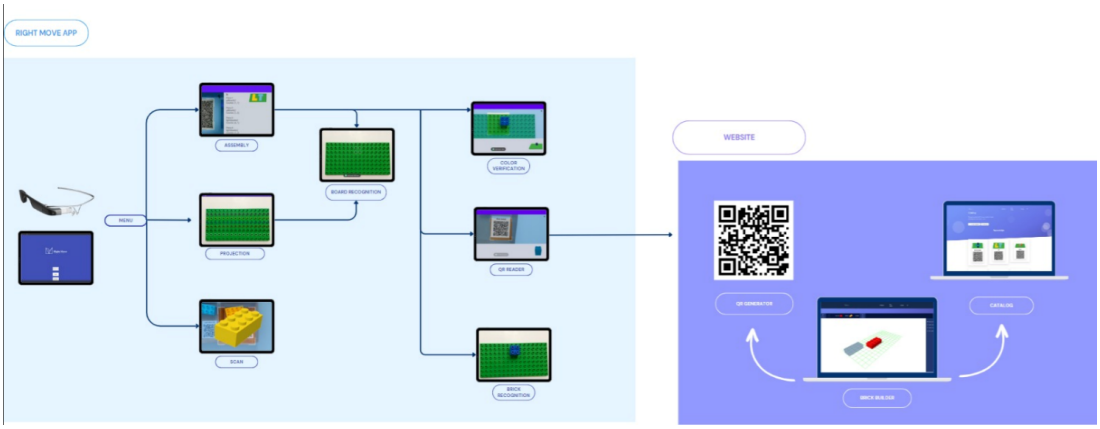

Fig.2: Architecture

4.1 Right Move Google Glass application (API)

The System's API works as a connector between all of the application entities and components to its presentation and data storage.

In this case, Right Move's API is a Java SOAP Web Server, which means that the request to the API is an HTTP POST request with an **XML** request body. This provides an important advantage when in comparison to regular **REST APIs**: It defines its own **security** and it is highly **secure**.

As mentioned, it has to be developed in Android Studio and use Android SDK. The application will handle the information gathered by Google Glass, and using this SDK will be able to explore its features with no limitations.

4.2 Information Storage

To have some characteristics that differentiate the LEGO blocks from each other, taking into account that we are using **official LEGO bricks**, we used the brand's official nomenclature, so each model of the LEGO block already has an **id**. This official code only identifies the **model** of the LEGO block and does not account for the color. To address this limitation, we have modified the code to include the initial letter of the desired color at the end of each ID. For instance, if the original LEGO block ID is 11111, our internal ID for a white block would be 11111W, if the LEGO block is white(W).

Data relating to both LEGO Blocks and assemblies are currently stored locally in **Json files**, and more legibly organized in an Excel file so that all elements of the group can consult it in a more practical way. We have two classes: **LEGO blocks** and **assemblies**. LEGO blocks are made up of **id**, **color**, and **dimensions**. Assemblies consist of an id, four steps, each one with one LEGO block, and a name. We have entered 17 types of LEGO blocks, 3 types of assemblies consisting of 4 LEGO blocks each, and 4 steps for each assembly into the Excel spreadsheet we discussed earlier. This is just a test to ensure that we are following the correct processes. All the relevant information regarding LEGO blocks and assemblies is stored within QR Codes.

4.3 Scenario Creation

To test what we were developing in our project, we created a **scenario** that would serve to verify that everything was correct. This scenario is made up of a green LEGO board where we assembled our LEGO blocks.

Bearing in mind that we initially developed an application based on position markers (**Arucos**), 3 **scenarios** were developed that accompanied the evolution of the application.

An initial scenario with a black background, consisting only in a green board of LEGO block, where the **Arucos** and some boxes where the LEGO blocks were stored

next to the QR Codes that identified each type of LEGO block were positioned in order to test the operation of the application in an initial phase.

In addition, an initial construction **catalog** was also drawn up, which only included assemblies with only 1 level in height. This catalog had several assemblies that the user could choose to assemble, with the LEGO blocks that were part of that construction and that the user would have to collect.

Following that, we developed the **final scenario**, which consists of a two-sided setup. The backside features a white background made of a wooden board, with the Arucos where the initial version of the **RightMove Application** can be demonstrated. It also features an assembly catalog and boxes containing both the LEGO Blocks and their corresponding QR codes, similar to the provisional scenario described earlier.

On the front side, there is a completely blank surface. This front side is dedicated to showcasing the final version of the application, where the recognition of board pins is independent of any placeholders. This setup allows for a comprehensive demonstration of the application's capabilities and advancements.

4.4 Assembly Maker

As previously indicated, two of the essential questions for the development of the app were: How could we detect the LEGO blocks and the whole scenario? How could we detect the color of the LEGO blocks placed? We can conclude that each assembly present in the catalog is divided into steps, currently 4, so the verification of the position and color of each placed **LEGO block** is easily executed between each step.

From the previously mentioned catalog, the user will select the desired assembly by scanning its respective **QR Code**, which contains an object of the Assembly type, identified by an ID. This ID present in the JSON object of the QR code will serve as a guide to finding in the local JSON files an assembly with the same ID so that each step can be correctly tracked.

Using a catalog with QR Codes for each construction, users can select their desired assembly by scanning the corresponding QR Code. Upon scanning, they receive the assembly's ID and the LEGO blocks needed for that construction. Users then **place** the LEGO blocks on a green board, following the prescribed order. The construction is considered **complete** when the final LEGO block is accurately positioned.

Chapter 5

Implementation

5.1 Mobile Application

In our mobile application, we offer users multiple options upon starting the app. They can choose to either **create an assembly**, **scan** a QR code of a LEGO block for its projection, or observe the pin detention mechanism in action by selecting the **projection** feature.

If the user decides to **create an assembly**, they will be prompted to gather the required LEGO blocks for that specific assembly. As the user collects each LEGO block, they can place it in its designated position on the green board. With every successful placement of a LEGO block, the app will **provide notifications** to acknowledge the user's progress and **verify** various aspects such as color, orientation, and position. Once all the LEGO blocks are correctly placed, the user will be notified of the assembly's completion.

If the user selects the **scan** option, they can scan the QR Code of a LEGO block, which will then **display** an image of that specific LEGO block. This functionality is particularly useful in a factory environment, where it enables users to **view the contents** of closed boxes or verify that items are **correctly placed** in their respective boxes identified by the QR codes.

By selecting the **projection** option, users can witness **the pin recognition feature** in action. It assesses factors such as lighting, camera positioning, and the detection of all 128 pins in their environment, ensuring the **optimal functionality** of our application.

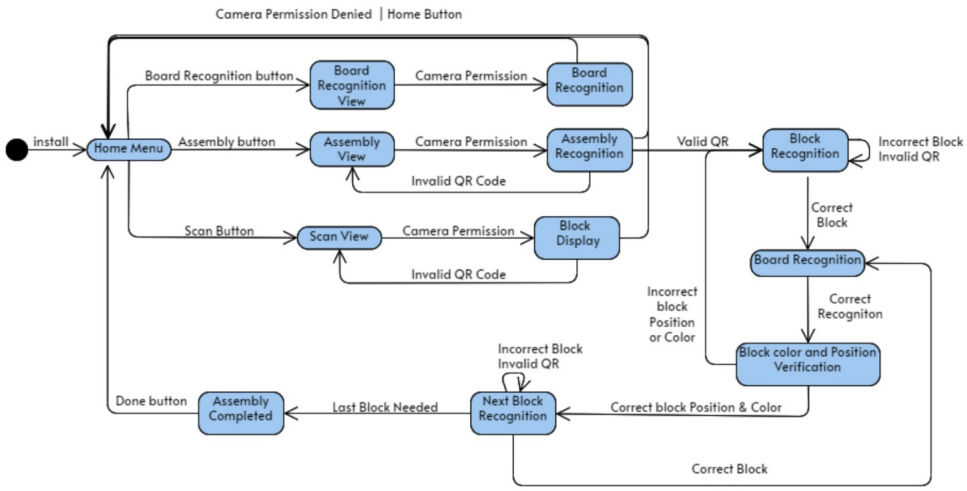

Fig.1: Machine State Diagram

5.2 QR Code Detection

During the planning stage of our application, we encountered three critical questions that would shape the rest of the app's development: How could we **track** the process of collecting and placing LEGO Blocks for assembly? How could we **detect** the LEGO blocks and the entire scenario? And how could we detect the color of the LEGO Blocks placed?

To address the first question, we developed a system for **reading** and **generating** QR codes. This section of our app is dedicated to the QR detection module and how it functions.

The QR code system allows us to track the **movement** of LEGO Blocks throughout the assembly process. Each block has a unique QR code that is scanned when it is collected and again when it is placed in its appropriate location.

This allows us to ensure that all necessary LEGO Blocks have been collected and placed correctly, and if not, we can quickly identify the issue and make corrections.

In addition to tracking the pieces, the QR codes also **enable** us to detect the scenario in which the assembly is taking place. By scanning a specific code at the beginning of the process, we can **identify** the specific assembly scenario and adjust the app accordingly.

Finally, to **detect** the color of the Blocks, we use **image recognition** technology. The app captures an image of the LEGO Block, analyzes it, and **determines** its

color. This information is then used to ensure that the pieces are placed correctly and in the right order.

So, we created a **QR Detection App** to detect QR Codes. Each LEGO block has an associated QR Code. When a QR Code is scanned, it contains a JSON file that serves as an identifier for the corresponding LEGO block. Utilizing this identifier, the app searches through local files to match the LEGO block and subsequently displays its corresponding image.

It is used when the user wants to collect a LEGO block to do the assembly and wants to know if he collected the correct LEGO block.

In conclusion, the development of the QR detection module was crucial in addressing the first of our three essential questions and ensuring the success of our assembly application.

LEGO Block Scan

Initially, it was decided that identification in our project would be done using QR codes. As a result, the first phase of implementing this system involved identifying each of the LEGO blocks used. Each LEGO block is assigned a unique QR code that contains a JSON file. The JSON file contains a JSON object that includes parameters such as: **(ID, color, and coordinates)**.

The ID parameter helps us to **identify** the specific LEGO part that is being used. This is crucial as it allows us to ensure that each LEGO block is in the right place and that the assembly is completed correctly.

The color parameter enables us to **differentiate** between different LEGO Blocks that may look similar but are, in fact, different. This ensures that the right LEGO Blocks are used and that the assembly process goes smoothly.

Finally, **the coordinates parameter** helps us to **locate** the position of each LEGO block within the assembly process. By knowing the location of each LEGO block, we can ensure that the assembly is completed correctly and that no LEGO Blocks **misplaced**.

This phase was merely transitional. Initially, the idea was only to use this system when building a specific assembly. However, we realized that it would be useful to use this same functionality in **parallel**, particularly when dealing with **opaque boxes** where the contents are not visible and need to be quickly consulted. As it happens on a factory floor where all the components of a given part are stored by type of component in a given box, we decided to develop this functionality that will allow users to understand, in opaque boxes for example, what their content is and thus

locate the part/component faster and more easily. When scanning this QR Code, that contains all the **data** described, an image of the element contained in the box will be shown.

In conclusion, by using QR codes and JSON files, we have been able to develop a **highly effective** identification system that ensures that the assembly process is completed smoothly and accurately. The identification of each LEGO block through QR codes and the associated JSON files has proved to be a critical component of our project.

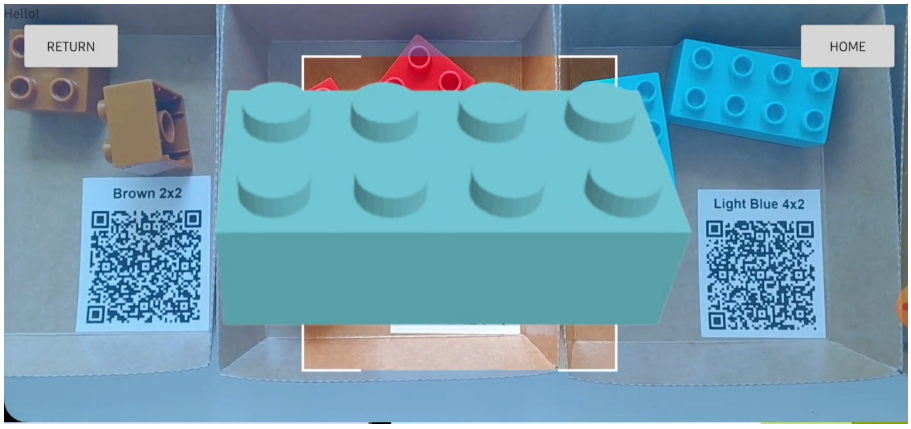

Fig.3: LEGO Block Scan seen from the mobile phone

Assembly Scan

Through the available online catalog, users can access a variety of pre-created assemblies. By scanning the QR Code of their desired assembly, users can enjoy an immersive experience while carrying out their own construction. In this option, the app guides users by indicating the required LEGO Blocks for the chosen assembly. It then verifies both the collection and placement of the blocks based on color and dimensions, ensuring that users have gathered the correct LEGO Blocks.

The verification of the placement of the LEGO block will be possible through the **'Coordinates Detection App'** function, which will use the Aruco markers or HoughCircles, depending on the version, to determine if users are correctly positioning the LEGO Blocks.

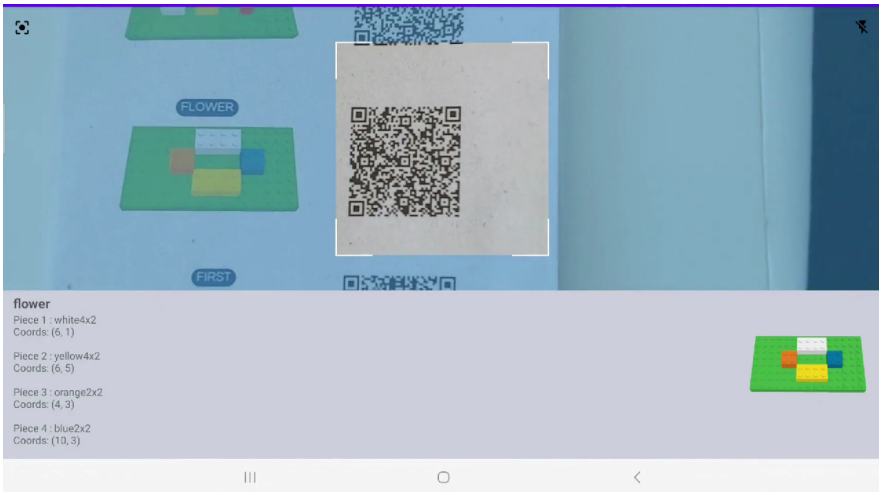

Fig.4: Assembly Scan done in the mobile phone

5.3 Board Recognition

Our board recognition must be the **first feature** used by the user. It is necessary to see if the environment of the user is **correctly** set up for the optimal operation of our application, since the LEGO blocks are made of reflective material. if the conditions such as **smooth light** and **plain background** are not met the app might not be able to recognize the LEGO blocks correctly. Once the right conditions are met the app will show the user all the **128 pins** identified divided by rows distinguished by colors as shown below. If the app fails to display the pins or if any of the pin colors are incorrect, it indicates that the environment has not been set up correctly.

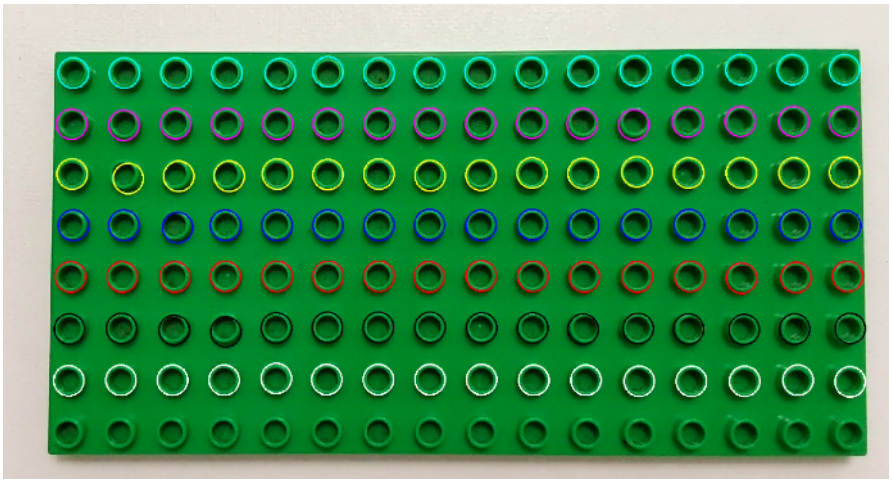

Fig.5: Board Recognition with a favorable environment

5.4 Coordinates Detection

5.4.1 Aruco Markers

Our initial method involved using Aruco Markers, which are easily identifiable tags that software can use to determine **location**. We placed these markers at the four corners of a green board to figure out their coordinates. Then, we measured the **distance** between pins on the board to calculate their positions based on these four corners. However, we found a problem with this approach: it only worked accurately if the user was **directly above the board** and **facing the same direction**. So, we had to come up with a better way. In our improved algorithm, we started by computing the line equations between the two left corners and the two right ones. Then, following a similar process to our initial method, we located the **8 pins** farthest to the left and right. We used these pins to calculate another line equation that goes through the highest pin from each group of 8. We repeated this process for all 8 pins on the far left and right. With this new algorithm, combined with our original one, we were able to accurately figure out the **coordinates** of all the pins, even when the LEGO block is not in line with the user's view. However, we ultimately chose not to utilize this solution because it proved to be highly inaccurate when dealing with tall constructions, as shown below.

as demonstrated in the examples below

This application is written in **Java** and uses **OpenCV** libraries. OpenCV helps us capture a live image from the camera and identify and estimate the position of the **Aruco Markers**.

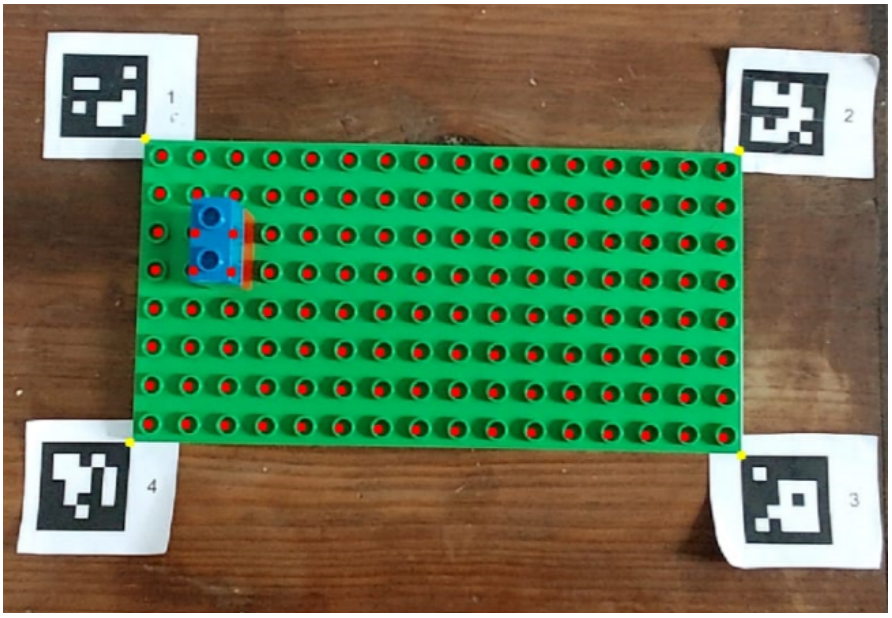

Fig.5: Coordinates Detection using Aruco Markers

5.4.2 Detecting using HoughCircles

To detect the coordinates, as we previously mentioned, we initially used Aruco markers. However, since the pins were not detected, we couldn't have a **second level** of height in the constructions, as the discrepancy between the calculation and the actual position of the Legos would be too large. Therefore, we tried several ways to obtain these coordinates:

Firstly, we tried to calculate the new coordinate of the pins based on their 3D position and then convert it to 2D using the **camera's projection matrix** to obtain the pixel coordinate and try to read the color of the pixel to know the pin's color. This approach didn't work because the glasses have only one camera and at least 2 cameras are required to make these calculations accurately.

Then, we looked for new solutions in which we used 3D projection to create a 3D model of the construction and compared that model with the real model. However, this option is slightly beyond our working capabilities, and we were not able to implement it. Finally, we tried to use **EdgeDetection**, which gave us the edges of the pins, edges of the borders, and the rest. However, we were unable to detect all of these edges because some pins were overlapping and not visible.

Therefore, we ended up using **HoughCircles**, which is an **OpenCV** function that detects basic shapes such as triangles, squares, or circles. In our case, the important thing is to detect **circles**. It identifies and provides a list of the coordinates of the

center positions of all the circles it detects on the phone screen. With that, we can detect all the **128 pins** of the green board.

This is the most recommended option to Google Glasses, since the camera has less resolution, turns out to be easier to recognize the 128 pins.

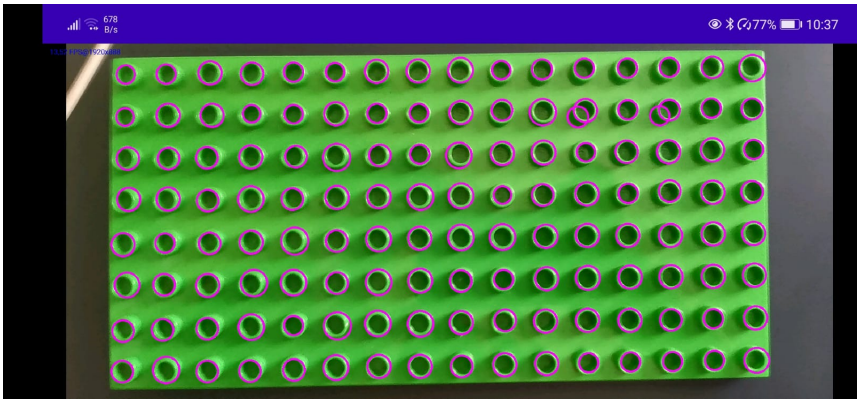

Fig.7: Coordinates Detection using HoughCircles

5.5 Color Detection

In our project, it is essential to incorporate a color detection function to enable distinguishing LEGO blocks based not only on their **shape** but also on their **color**. This allows us to **differentiate** LEGO blocks with **identical dimensions** using the aspect of color.

The camera is activated and the app collects information about a given pixel, using **RGBA**, it calculates the Euclidean between the RGBA of the pixel and RBG of all the possible colors of the Bricks and returns the name of the one with less distance.

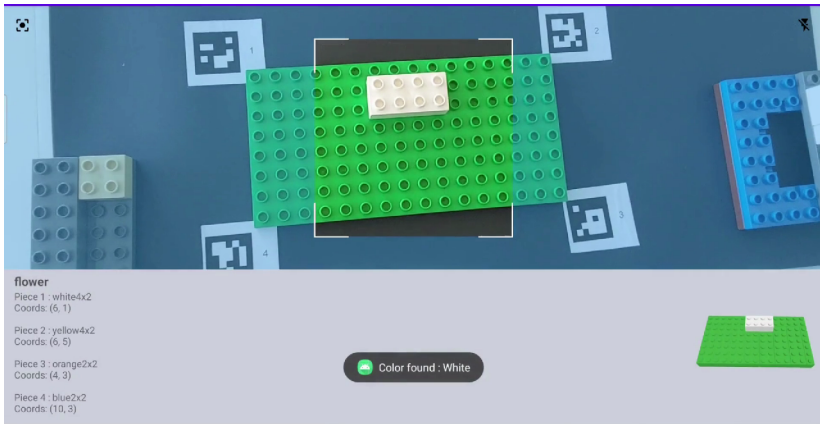

Fig.8: Color Detection of a White LEGO Block

5.6 Website

Our website offers a variety of useful features to help you with your building projects. With the **Brick Builder** tool, the user can easily create and design their own constructions, choosing from a wide range of materials and colors to create the perfect assembly. The **Brick Builder** is not our work, but a project that we discovered during the **requirements gathering phase** in which we investigated a lot about all existing projects that could have some similarity and even be suitable for our project, always with the adaptations that we knew we have to do. Thus, upon realizing the mechanism of this **Brick Builder**, we decided to adapt it to our needs, taking advantage of something that was previously developed. The **Brick Builder** is intuitive and easy to use, making it accessible to both beginners and experienced builders.

Our website also offers a comprehensive catalog of all possible building assemblies, allowing the user to browse and explore the many different assembly options available to them.

In addition to the Brick Builder functionality, our project also offers users the ability to access and view our group's work plan directly on the website. This feature ensures that users are well-informed about our project's progress and provides them with concise information regarding our path. By offering transparency and clarity on our project's roadmap, users can easily understand our journey and stay updated on the latest developments.

Finally, we have a button that takes the user directly to our project's GitHub and makes it possible for the user to see the codes used.

Overall, our website is a valuable resource for anyone involved in the building industry. With its user-friendly features and extensive catalog, it makes the process of planning and executing building projects easier and more efficient than ever before.

5.7 Project Evolution

In our project, we started by using a few **LEGO blocks** to test the functionality of our project. Each of these LEGO blocks was assigned a QR code to facilitate identification and tracking.

Once we had created the initial LEGO blocks, we created a testing environment that would allow us to test the code of our project. This environment is a scenario that includes the green board and that allows the placement of LEGO blocks in the appropriate places.

After the testing environment was in place, we started to develop **functions** to help us achieve our goals, and that included functions that would detect the coordinates and colors of the LEGO blocks, which is essential for any LEGO project to work as intended. Subsequently, we initiated the **creation of constructions**, each assigned with a unique QR code. These constructions were **initially** limited to a **single level** of height, but we **gradually** added more LEGO blocks and shapes to increase the number of possible constructions that could be created with our application.

As we continued to develop our project, we created a **board recognition** feature that would allow users to determine whether they were in an environment where the assembly could be performed accurately. This feature is particularly useful, as it helps users avoid the frustration of attempting to build a LEGO structure in an environment that is not suitable.

Finally, we managed to achieve a **significant milestone** in our project when we were able to add two levels of construction using the **HoughCircles** function. This is a noteworthy achievement, as it significantly **increases** the number of possible constructions that can be created with our application.

As a result of implementing **the HoughCircles function**, we now have two versions of our project. The first version allows for the creation of constructions with **two building levels** and is independent of any specific scenario. In contrast, the second version incorporates **Arucos** and supports constructions with **only one level**, depending on the scenario that has been established.

Overall, by starting small and gradually building up, we have been able to create a robust and functional application that offers a wide range of possibilities for users. We are **proud** of our progress so far and look forward to continuing to develop our project further.

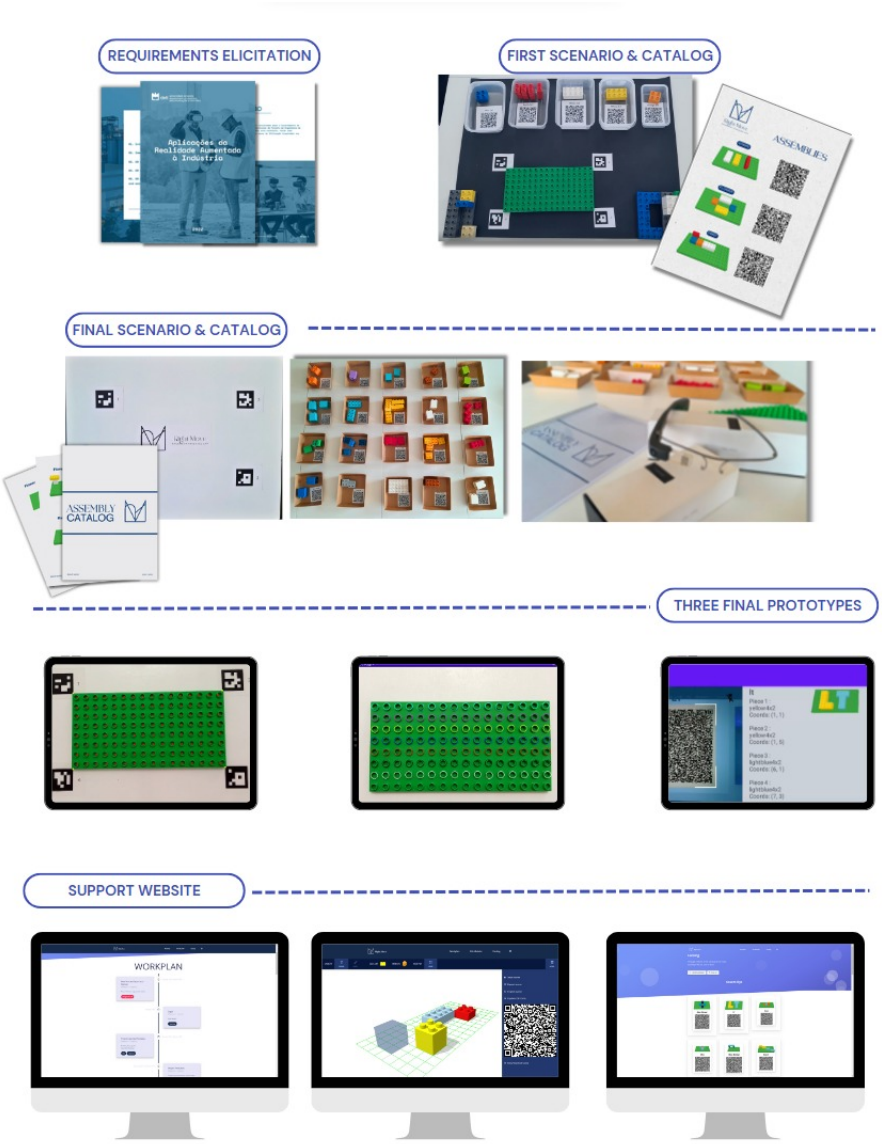

Fig.9: Project Progress

Chapter 6

Conclusion and Future Work

Since our project is an **exploratory** project, we can conclude that most of the work done so far has been essentially research work and requirements gathering. Initially, the fact that we had no direct contact with **Huf Portuguesa**, or the material made it difficult to start the project.

This LEGO project was a **fun** and a **educational experience**. Through this project, we were able to **improve** our creativity and problem-solving skills while building and designing with LEGO bricks. We learned how to **work collaboratively** as a team, communicate effectively, and think critically to **overcome challenges**.

Additionally, we developed a greater appreciation for the versatility and endless possibilities of LEGO bricks. We were able to create unique and imaginative structures, from simple designs to complex models.

Overall, the LEGO project not only provided an **enjoyable experience** but also helped us develop **important skills** that can be applied in various aspects of our lives. We look forward to future LEGO projects and continuing to explore the world of LEGO bricks.

As we look towards the **future** of our platform, the construction industry aims to simplify and streamline its processes by integrating a comprehensive **database system**. This innovative approach will **eliminate** the reliance on QR codes for tracking and managing various construction elements. By **storing** relevant information in a centralized **database**, such as project details, material specifications, and quality control parameters, construction professionals will have instant access to critical data without the need for physical QR codes. This advancement will **significantly enhance efficiency** and minimize errors in construction operations.

Another exciting development on the horizon is the advancement of **color recog-**

nition algorithms in construction. Traditionally, color recognition has played a crucial role in differentiating between construction materials and components. However, the future brings improved algorithms capable of accurately and swiftly identifying colors even in complex environments. By harnessing machine learning and computer vision techniques, these algorithms will enhance the speed and accuracy of material selection, inventory management, and quality control processes, ultimately resulting in substantial time and cost savings.

As we continue to develop our platform, the **block recognition** module, which is currently utilized in various construction applications, is set to undergo significant modifications to cater specifically to industrial settings. **Industrial environments** often present unique challenges, such as complex machinery, hazardous conditions, and diverse materials. Future advancements will focus on enhancing the module's capabilities to robustly **identify** and **analyze** different types of blocks commonly found in industrial settings. This enhanced recognition will allow for more precise automation, efficient resource allocation, and improved safety protocols, ultimately leading to optimized industrial construction processes.

For those interested in learning more about the technical aspects of our platform, all the information necessary for continued development is available on our GitHub: RightMoveGitHub.

We are excited about the potential of our platform and how it can benefit industries. By making it easy for employees to construct the necessary LEGO blocks and assemblies. Our platform can help increase **efficiency** and **productivity** in the workplace. We are grateful for the support and feedback from our partners at **'Huf Portuguesa'**, particularly David Pinheiro and Jorge Silva, who have been instrumental in shaping our platform.

Chapter 7

Appendices

7.1 Usage Session

Scan

To view the contents of a box that is previously identified with a QR that identifies the contents of that box, by scanning this QR Code the user will be able to see the 3D model of the LEGO Block contained in the box

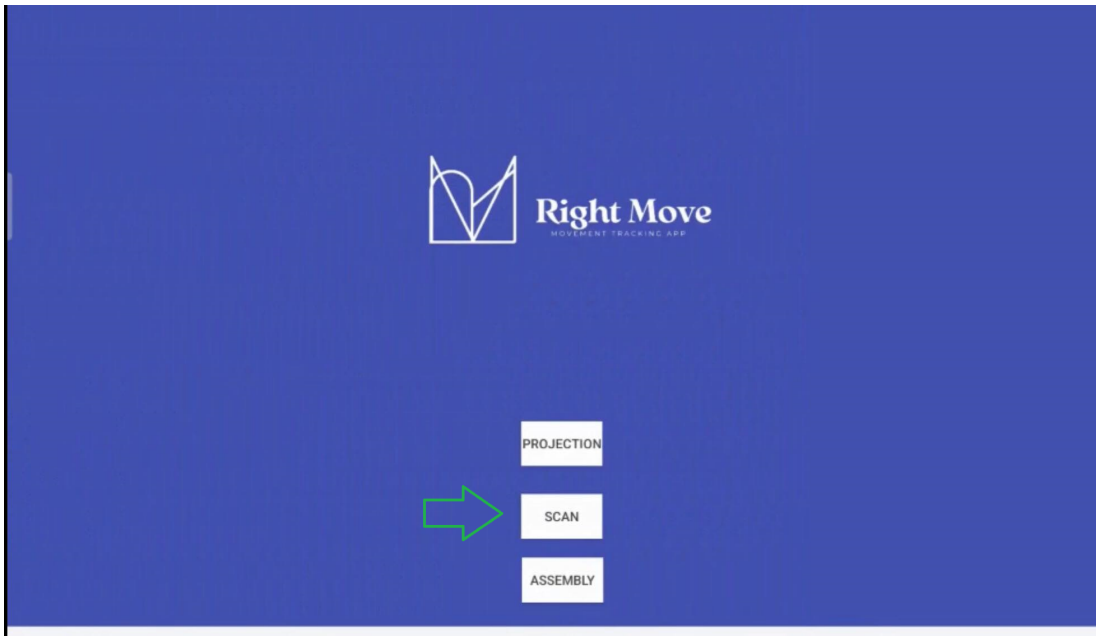

Fig.10: Menu

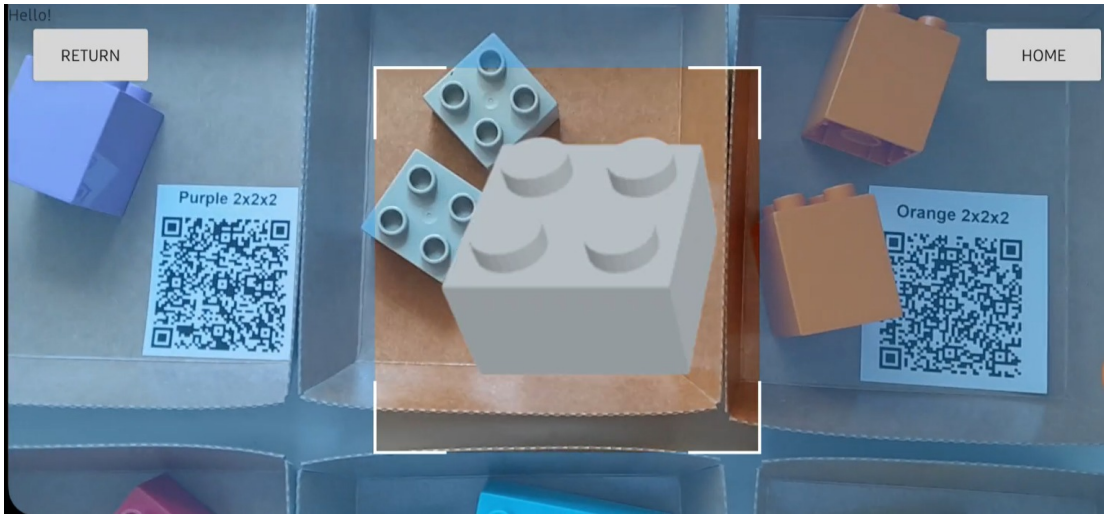

Fig.11: LEGO Block scan

Board Recognition

The user wants to make a Board Recognition to know if he is in a good environment to do an assembly. He will need to select the projection button in the main menu. Then, if it is the Aruco Markers version, the user will be able to test the environment as shown in figure 13 . Using the Aruco Markers version, it must be possible for the camera to see the 4 Aruco markers in full. If it is the final version, the user can test the environment as shown in figure 14. Using the Independent version, it must be possible for the camera to see the 128 pins in full and the edges of the board:

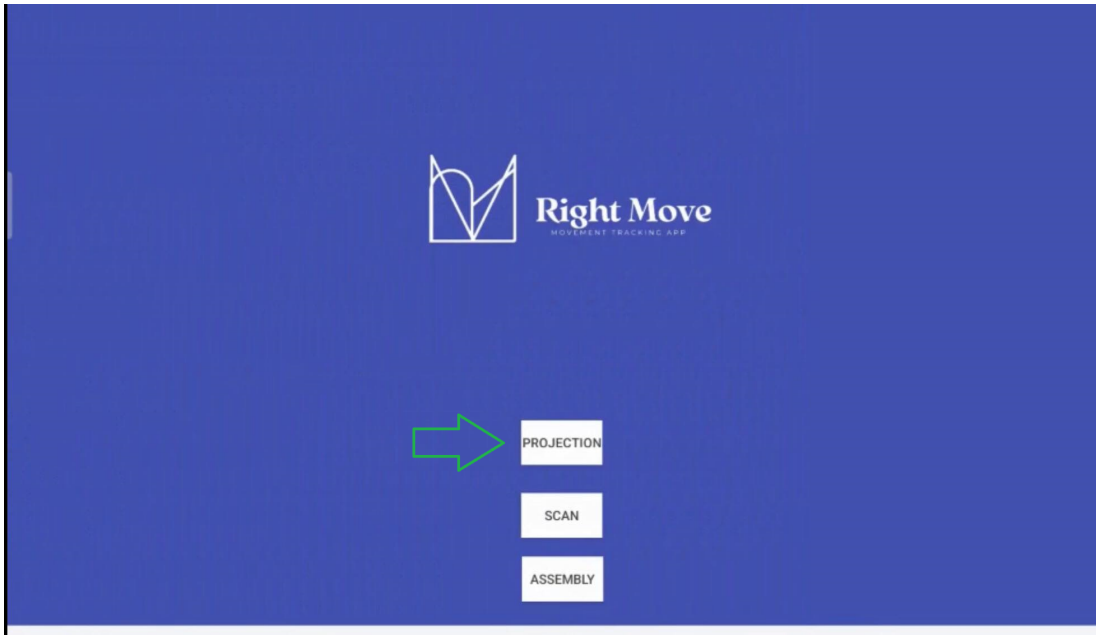

Fig.12: Menu

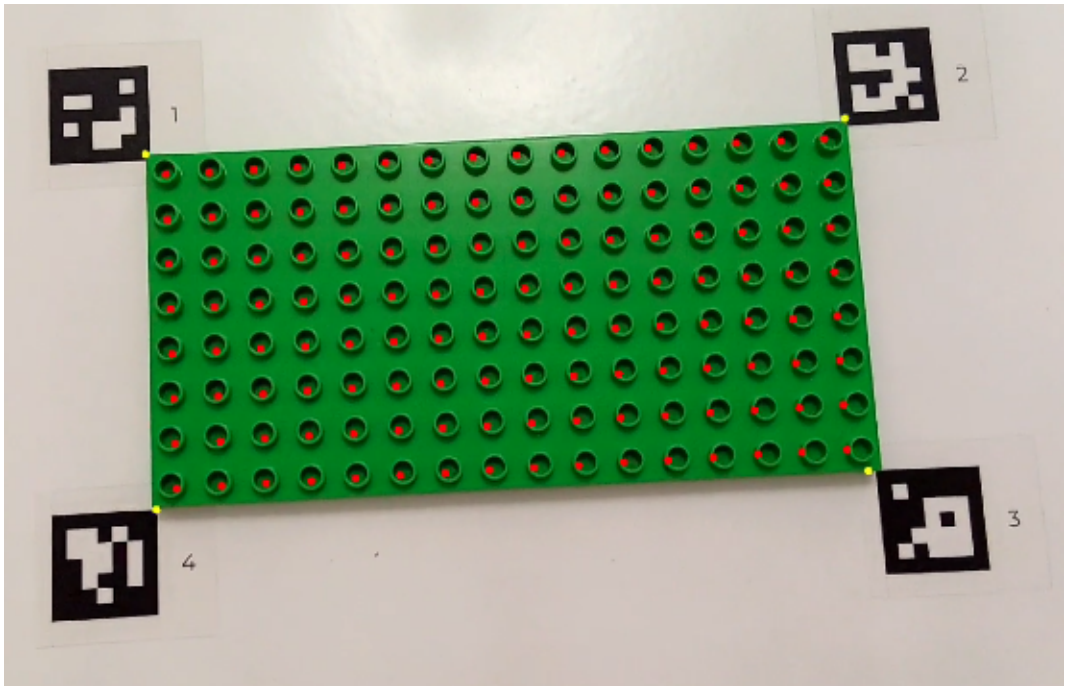

Fig.13: Aruco Markers version to test the environment

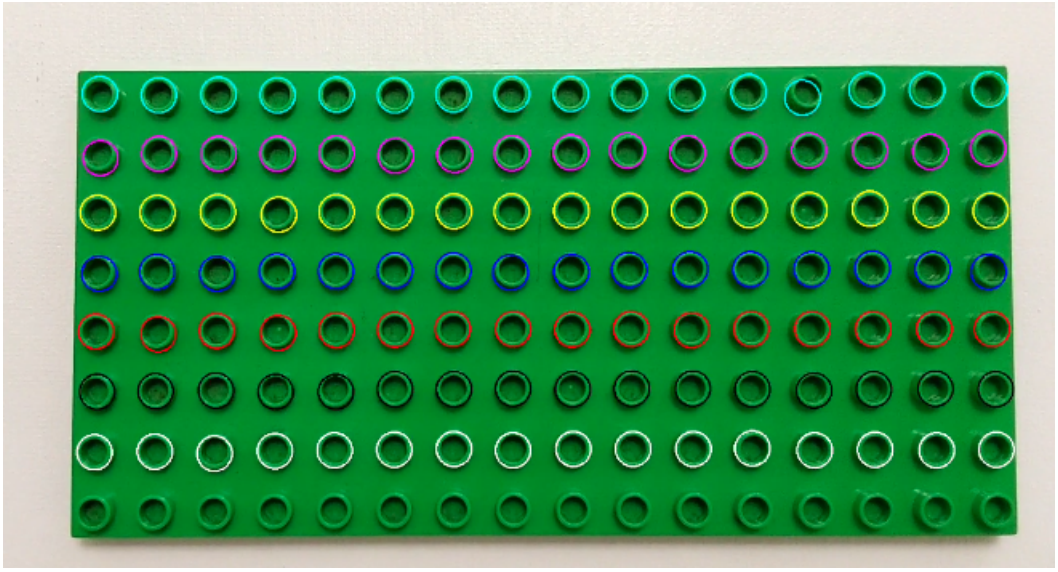

Fig.14: Independent version to test the environment

Mobile Phone Assembly

The user wants to do an assembly with his mobile phone. He will need to select the assembly button in the main menu. Next, the user will have to scan a QR Code of a construction, from this scan the user will have access to the ordered list of LEGO Blocks of the assembly. Then each step corresponds to the collection and placement of a LEGO Block, this collection is done by scanning a QR so that he can then collect and place it in the correct place, piece by piece. If the camera is misplaced, the app will not move on to the next part and wait for the correct placement of the piece.

The user can also generate an assembly on the Website, as explained further on **WebSite** subsection below.

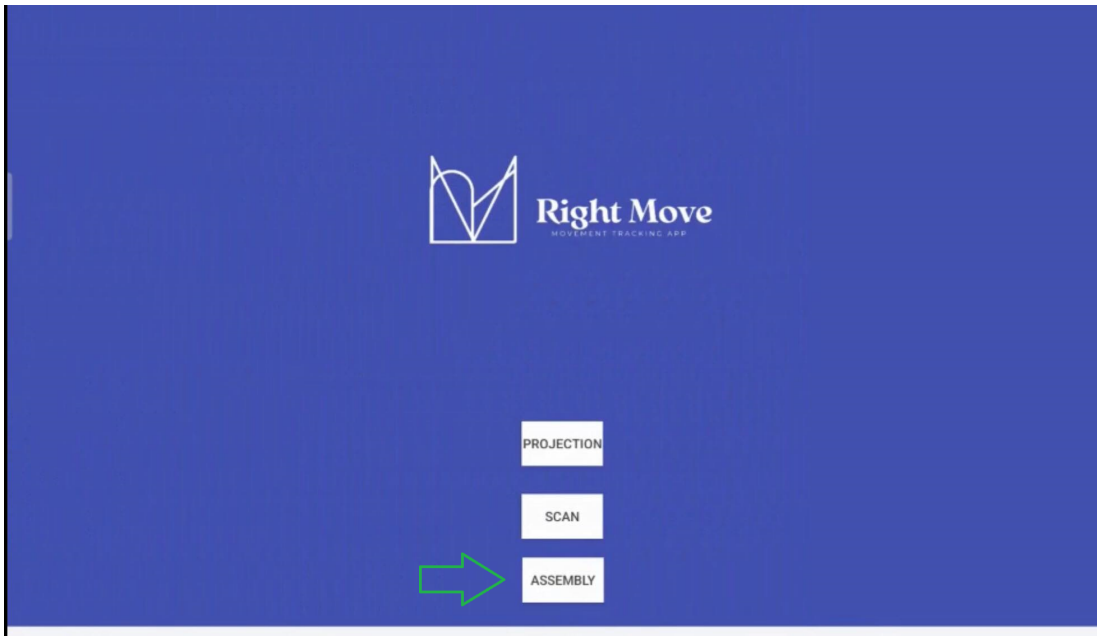

Fig.15: Menu

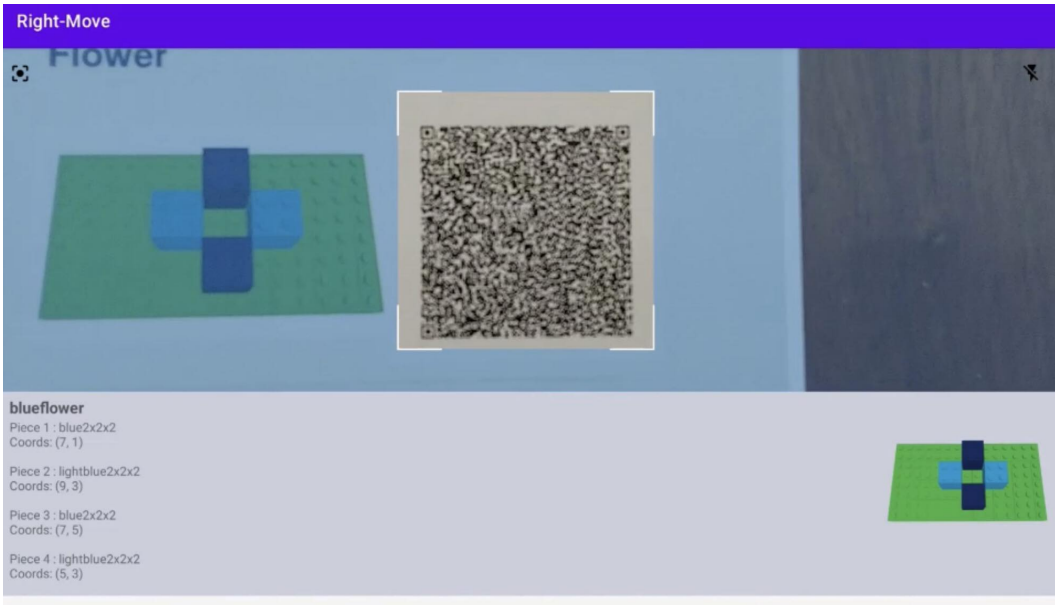

Fig.16: Scan of an Assembly QR Code

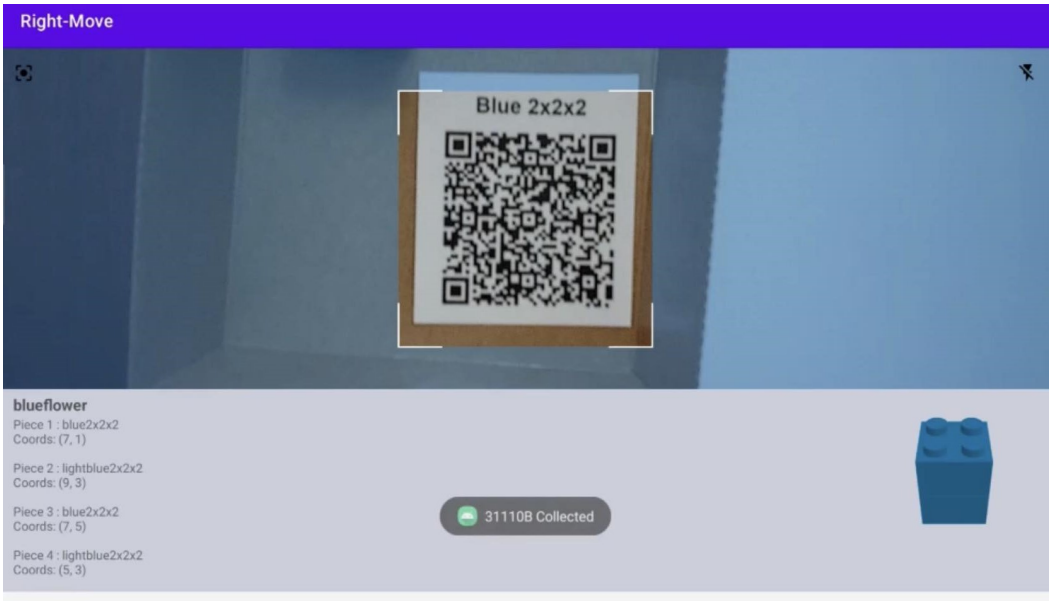

Fig.17: Collect of a LEGO Block

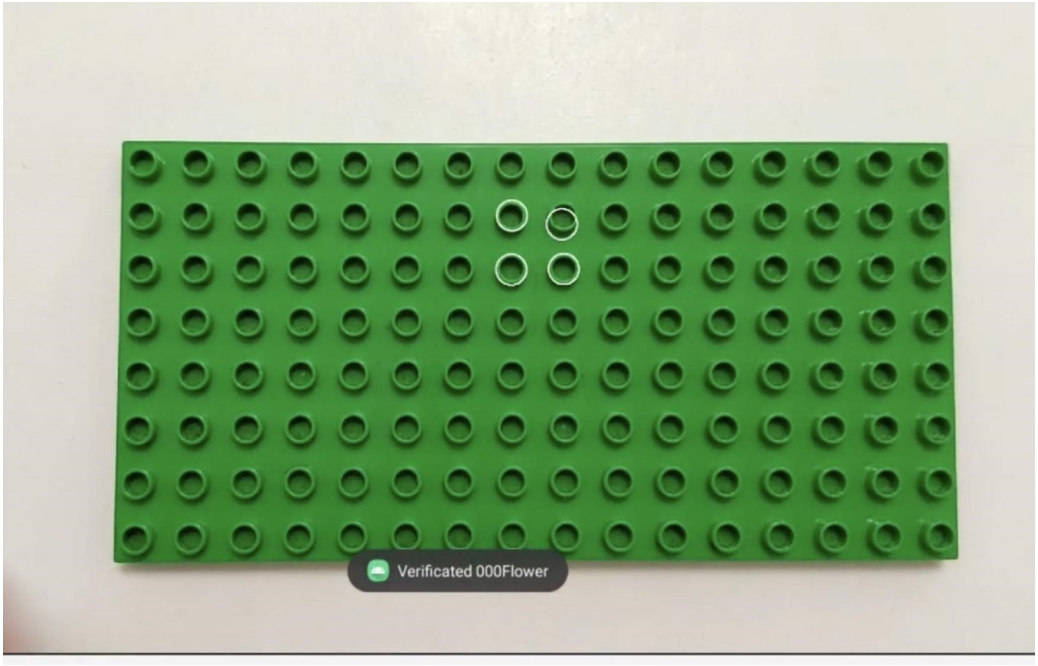

Fig.18: Place to put the LEGO Block

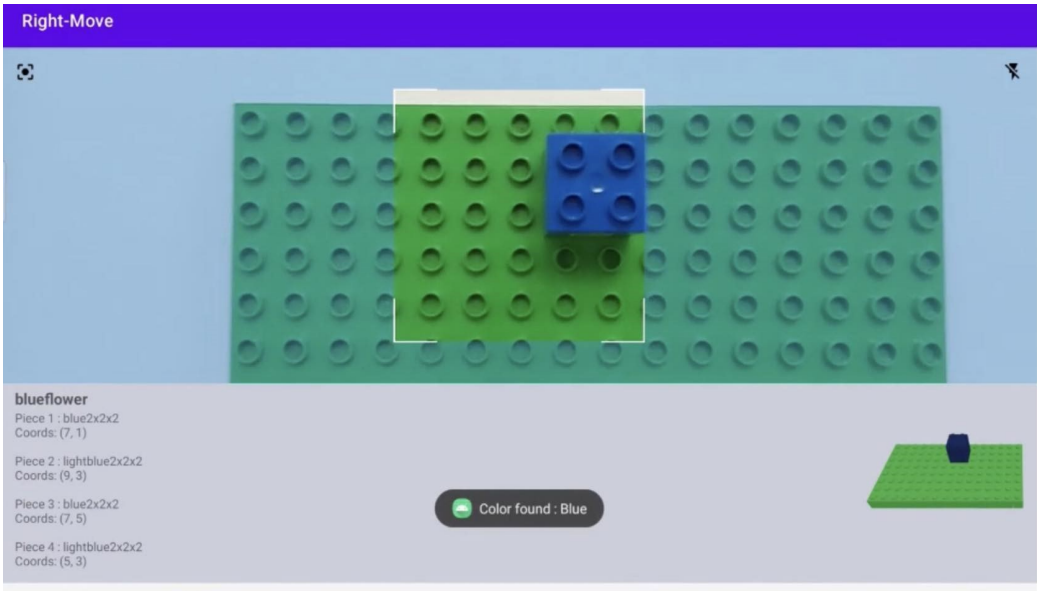

Fig.19: LEGO Block placed correctly

Google Glasses Assembly

The user wants to do an assembly with Google Glasses. The same functions are available in the Google Glasses version but with a different User Interface since the UX would also be different. The user interface needs to be different because the

Google Glasses display is very small and caused difficulties for the user.

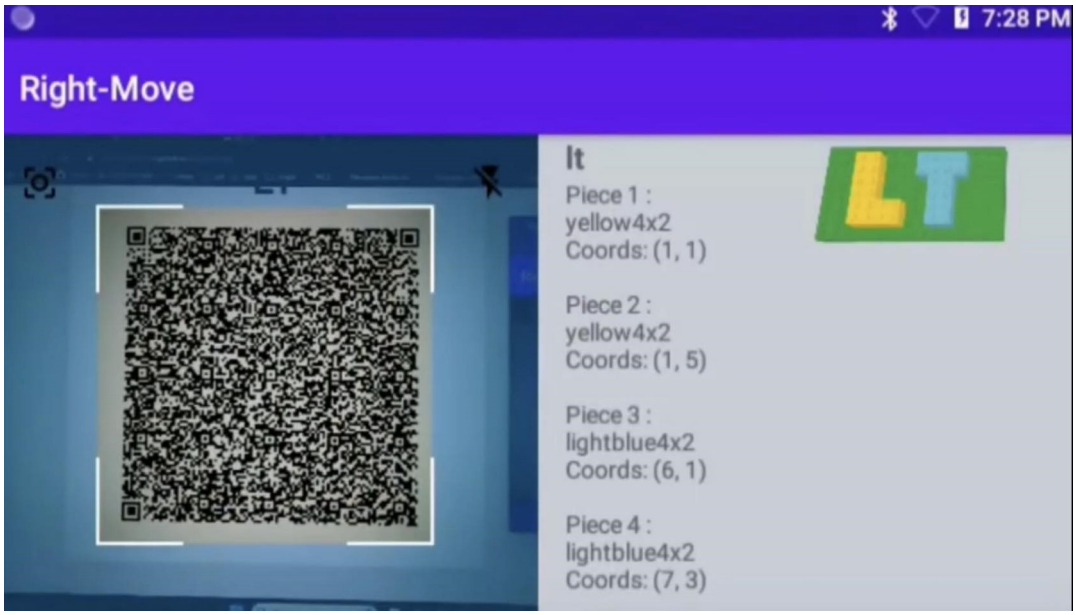

Fig.20: Scan of an Assembly QR Code in GoogleGlasses

WebSite

On our website, the user can make his own assembly using the BrickBuilder which is not pre-built, but a project that we discovered and used to allow the user to make his own assemblies. After finishing the construction, a QR Code will be generated that the user can scan and thus carry out the assembly. The user can also see our Project Workplan and our Catalog assemblies.

If the user wants to make an assembly in our website :

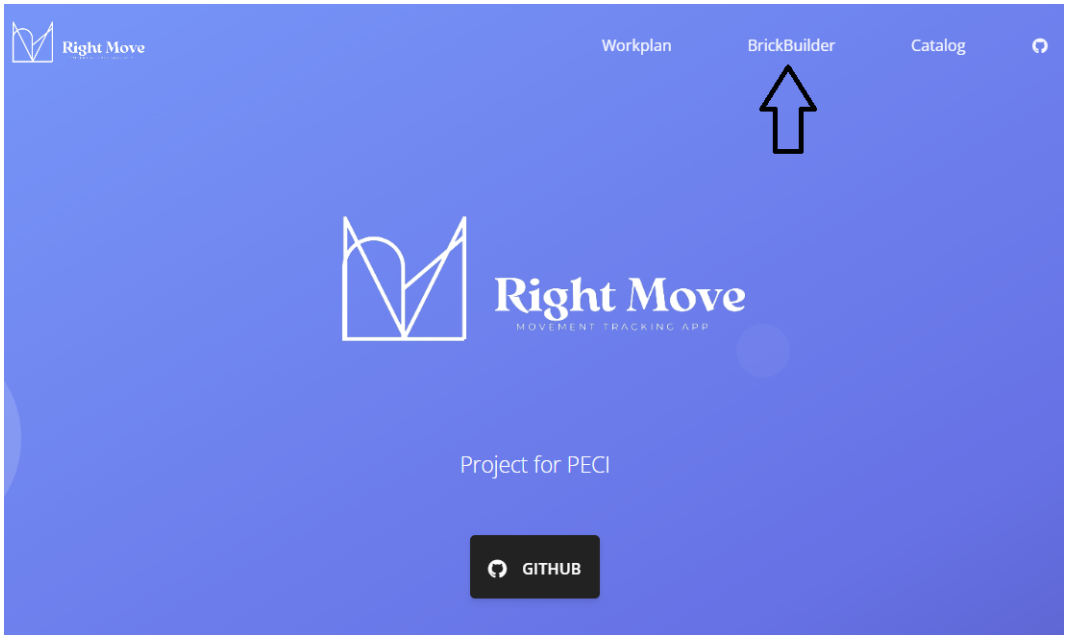

Fig.21: Our Website Initial Page

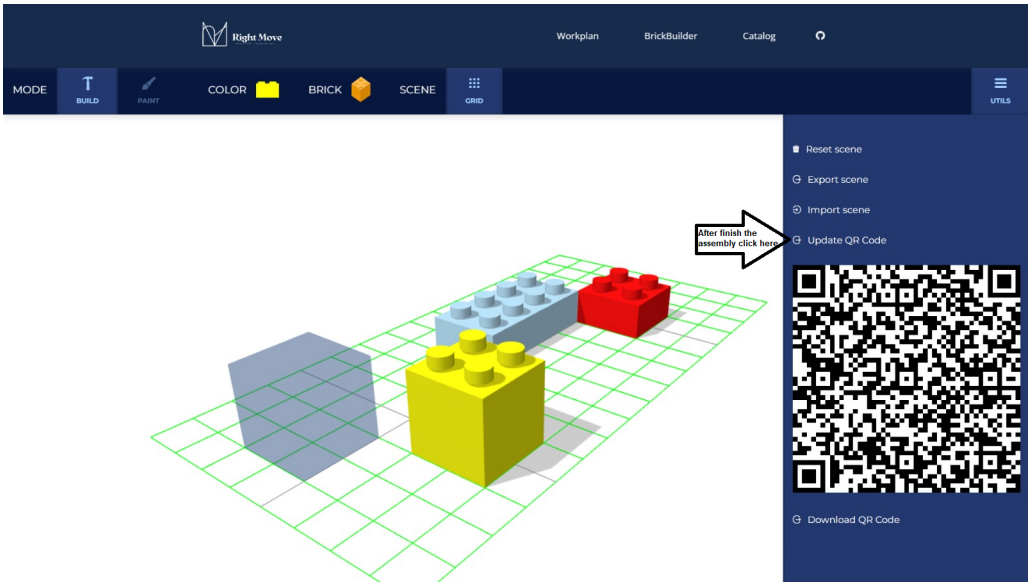

Fig.22: BrickBuilder in our website

If the user wants our project worplan:

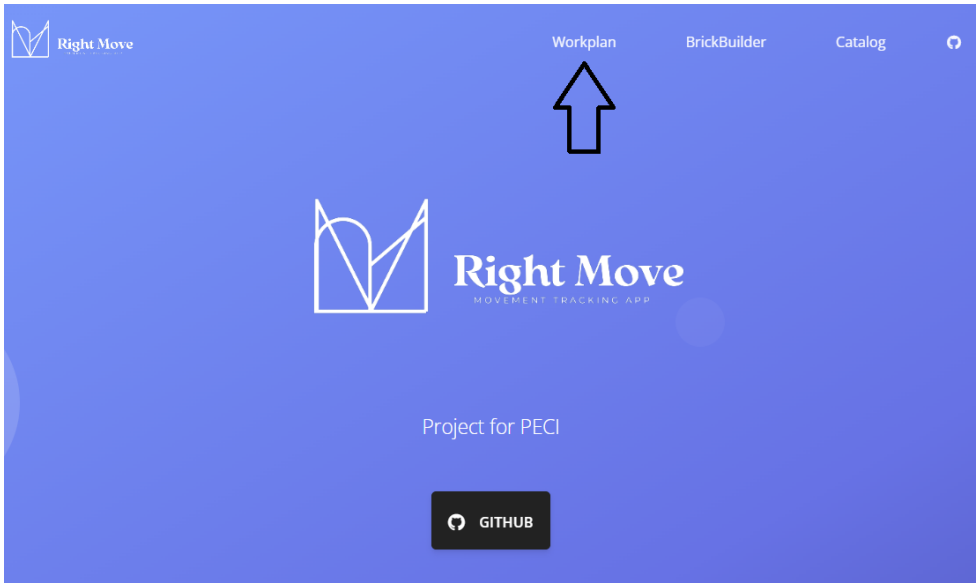

Fig.23: Our Website Initial Page

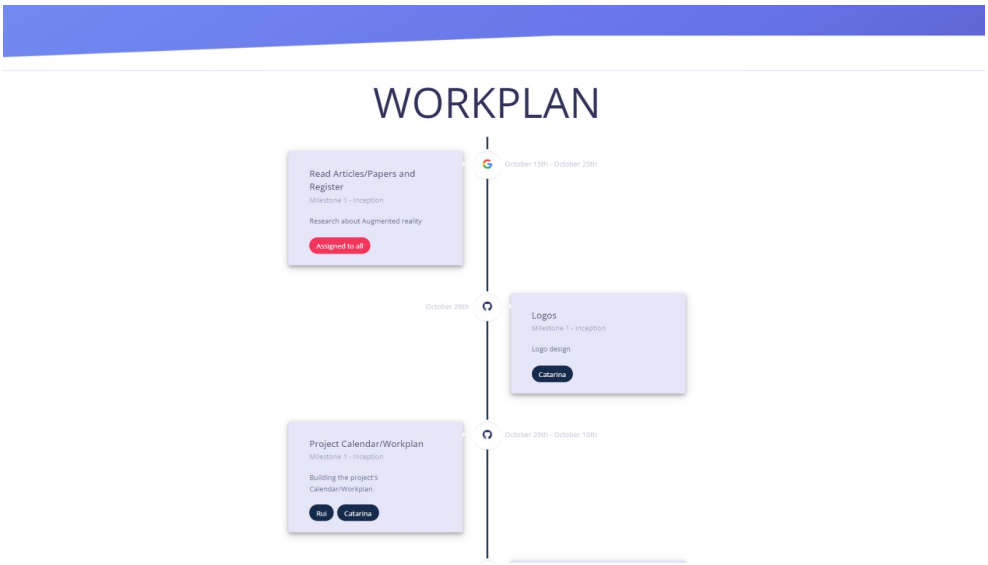

Fig.24: Workplan in our website

If the user wants to see our catalog to make an assembly:

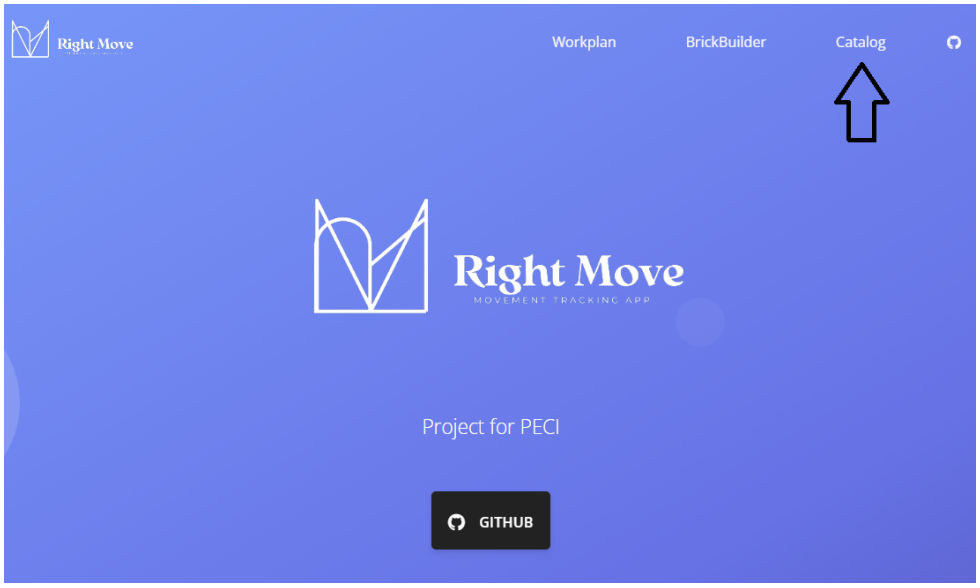

Fig.25: Our Website Initial Page

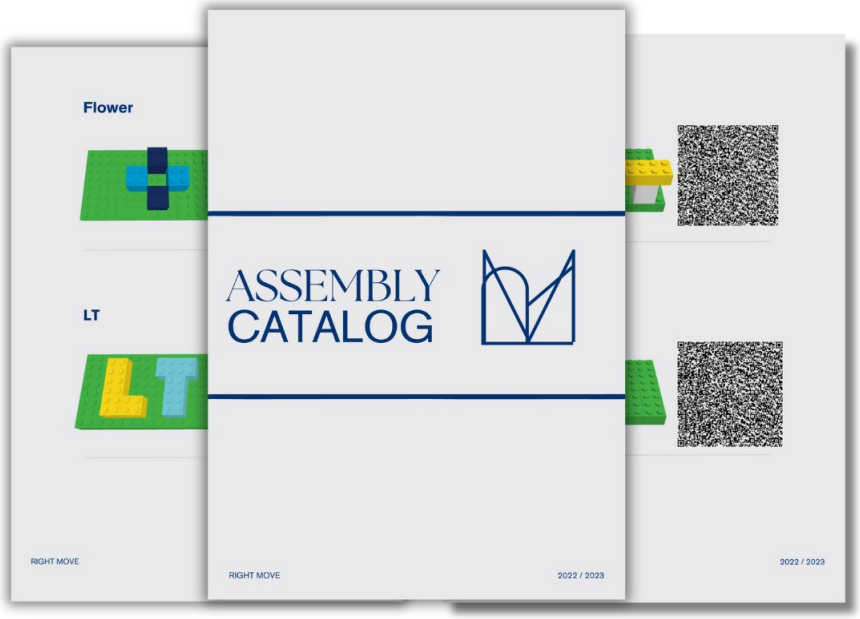

Fig.26: Catalog in our website

7.2 Project Documentation

Files Available

The available files are:

1. ***assembly.kt***: This file encompasses the entire application, except for the license board recognition and color detection. This crucial file comprises the core functionalities and features that drive the application's overall behavior and performance. It serves as the backbone, orchestrating various components and modules to ensure seamless execution. This file contains the *onRequestPermissionsResult* function, which shows the user a pop-up to allow him to use the camera; the *firstcodeScanner* function, which contains all construction steps, such as picking up LEGO Blocks; the *verificationFirst* function that verifies that the part was placed in the correct place on the board;

```
private fun verificationFirst(id: String, index: Int, assembly: String) {
    val instruction = instructionsList.find { it.assembly == assembly }
    if (instruction != null) {
        Toast.makeText(this@assembly, "Verificated ${instruction.assembly}", Toast.LENGTH_SHORT).show()
        Log.e("MYAPP", "Before")
        switchActivity(instruction.steps[index].coordinates.x, instruction.steps[index].coordinates.y,
            object : ColorSelectedListener {
                override fun onColorSelected(color: String) {
                    Log.e("MYAPP", "After")
                    if (instruction.assembly == assembly && instruction.steps[index].idPiece == id && instruction.steps[index]
                        Log.e("MYAPP", color)
                        val resourceId = resources.getIdentifier(instruction.steps[index].idStep, "drawable", packageName)
                        image_view.setImageResource(resourceId)
                        image_view.visibility = View.VISIBLE
                        return
                    }
                }
            }
        )
        Log.e("MYAPP", "After Switch")
    }
}
```

Fig.27: *assembly.kt*

Then, this file has a Kotlin function named *firstcodeScanner* that sets up and handles the functionality of a QR code scanner. The function initializes a *CodeScanner* object by passing the current activity (this) and a *scannerview* as parameters.

The CodeScanner object's properties are configured: the camera is set to use the back camera; formats are set to scan all QR code formats; autoFocusMode is set to *AutoFocusMode*; SAFE to enable safe auto-focus mode; *scanMode* is set to ScanMode.CONTINUOUS to continuously scan for QR codes. *isAutoFocusEnabled* is set to true to enable auto-focus. *isFlashEnabled* is set to false to disable the flash. The *decodeCallback* property is set to a callback function that is invoked when a QR

code is successfully decoded. Inside the callback function, the following actions are performed:

The decoded QR code result is deserialized using Json into an object of type *AssemblyData*.

```
decodeCallback = DecodeCallback { result ->
    runOnUiThread {
        try {
            val qrData = Gson().fromJson(result.text, AssemblyData::class.java)
            codeAssembly = qrData.assemblyId
            val pieces = arrayOf(
                "1st Piece: ${qrData.piece1}",
                "2nd Piece: ${qrData.piece2}",
                "3rd Piece: ${qrData.piece3}",
                "4th Piece: ${qrData.piece4}"
            )
```

Fig.28: *QRCode decoded*

Various pieces of information from the deserialized object are extracted and stored in local variables. The extracted information is used to update views on the User Interface, such as setting text in *qrInfoTitle* and *qrInfoDetails* TextViews, showing the *qrInfoLayout*, and displaying an image based on the QR code's name. A *current-Piece* variable is set to -1, which will be used to keep track of the currently collected piece. After a delay of 5 seconds, the codeScanner's *decodeCallback* is updated to a new callback function. Inside this new callback, the following actions are performed: The QR code result is deserialized using Json into an object of type *QRCodeData*; The ID of the second QR code is compared with the ID of the expected piece and if the IDs match, a "Piece Collected" toast notification is shown, and the *currentPiece* counter is incremented. The *verificationFirst* function is called with appropriate parameters to perform some verification process. A log message is printed, and if the *currentPiece* value reaches 3, a pop-up dialog is displayed to notify the user that all pieces were collected. If the IDs don't match, a "Wrong Piece" toast notification is shown. Views and visibility are updated accordingly. The first scanner's preview is started again using *codeScanner.startPreview*. The scanning variable is set to false to indicate that scanning has finished. An error callback is set to handle any errors that occur during the camera initialization process. If an error occurs, an error message is logged.

The $scanner_v iewissetwithaclicklistenertostartthepreviewofthecodescannerwhenclicked.$

2. ***assemblyData.kt***: This file contains the defined class related to each Assembly, such as the ID, the required LEGO Blocks, and the name.

```
data class AssemblyData(
    val assemblyId: String,
    val piece1: String,
    val piece2: String,
    val piece3: String,
    val piece4: String,
    val name : String
)
```

Fig.29: *assemblyData.kt*

3. ***ChecklistAdapter.kt***: This file adapts the user's view and provides necessary functionalities.

4. ***colorVerification.java***: This file has a function that verifies the color of a specific LEGO Block using OpenCV.

The provided code is a method called onCameraFrame that processes frames from a camera in real time. It utilizes the OpenCV library to perform various computer vision tasks such as detecting markers, extracting corner points, and analyzing pixel colors. The method begins by obtaining the RGB and gray scale versions of the input camera frame (rgba and gray).

Main version: The class declares some private variables, including *mOpenCvCameraView*, *mIsJavaCamera*, and *mItemSwitchCamera*.

It initializes an empty ArrayList called *"coloridos"* to store instances of the *Pino* class. The *mLoaderCallback* is a callback object that handles the initialization of OpenCV; When OpenCV is successfully loaded, it enables the camera view; The constructor of the *colorVerification* class logs a message when an instance of the class is created; The static block loads the OpenCV library. The *onCameraFrame* method is called when a new camera frame is available. It processes the frame to detect circles using the Hough transform. It applies median blur to the grayscale image, performs the Hough transform, and stores the detected circles in the *circles* matrix. The method then processes the detected circles by converting their coordinates and radius to integers. It filters out smaller circles contained within larger ones. If the number of detected circles is 128, it creates a *Placa* object using the detected circle centers. The *Placa* class seems to represent a board. The method extracts the coordinates of the pins from the intent extras. It retrieves the color of the pins

from the intent extras. It calculates the closest color name for two pins using the *guessColor* method, which takes an RGB image and a point as input and returns the closest color name based on the RGB values of the pixel at that point. Finally, it logs the color of the first pin and compares it with the specified color. If both colors match, it calls a method called *setResultColor*.

Aruco Markers Version: It initializes a list called *corners* to store the corner points of detected markers and creates a matrix called ids to hold the detected marker IDs; an Aruco dictionary is defined; an ArucoDetector object is created with the defined dictionary and parameters, and it detects Aruco markers on the grayscale frame; then creates a HashMap called *dictCantos* to store the detected Aruco IDs along with their corresponding corner points; inside a loop, it iterates through the detected markers, retrieves the ID and corner points, and adds them to *dictCantos*; if the size of *dictCantos* is equal to 4, indicating all four markers have been detected, the code proceeds with further operations; it retrieves the corner points from *dictCantos* for markers 1, 2, 3, and 4, respectively; the coordinates of the corner points are printed and marked on the rgba frame using red dots; a *Placa* object is instantiated with the corner point coordinates to further process and analyze; a matrix called *matrix* is obtained from placa using the *getMatrix* method; the code enters a loop to determine the position of a "piece" using the *getPieceX* and *getPieceY* methods until valid values for *pinoX* and *pinoY* are obtained; the x and y coordinates are retrieved from the matrix using *pinoX* and *pinoY* as indices.

5. ***Menu.kt***: This file primarily initializes the scanning and construction functions and only has 3 buttons.

6. ***MainActivity.kt***: The file is an Android application code written in Kotlin. The code demonstrates the use of a QR code scanner using the CodeScanner library.

7. ***Pino.java***: These files have a class that provides the coordinates of each pin on the board.

```java
public class Pino {
    1 usage
    String cor;
    3 usages
    double x;
    3 usages
    double y;
    sauce
    public Pino(double x, double y, String cor ) {
        this.x= x;
        this.y =y;
        this.cor= cor;
    }

    sauce
    public double getX() { return x; }

    sauce
    public double getY() { return y; }
```

Fig.30: *Pino.java*

8. ***Placa.java***: This file contains a function called *Placa* that stores the coordinates of each pin and another function to obtain the coordinates of the LEGO pieces.

9. ***Point.java***: This file holds the coordinates of a specific point.

```java
public class Point {
    public double x;
    public double y;

    sauce
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
```

Fig.31: *Point.java*

10. ***QRCodeData.kt***: This file contains the defined classes about each QR Code.

```kotlin
data class QRCodeData(
    val id: String,
    val color: String,
    val dimensions: Dimensions,
    val name : String
)

Sauce
data class Dimensions(
    val x: Int,
    val y: Int
)

Sauce
data class Step(
    val idPiece: String,
    val idStep: String,
    val color: String,
    val coordinates: Dimensions
)

Sauce
data class Instruction(
    val assembly: String,
    val steps : Array<Step>,
)
```

Fig.32: *QRCodeData.kt*

11. ***Reta.java***:

This file collectively supports the functionalities and operations of the application.

```java
public  class Reta{
    1 usage
    double m;
    1 usage
    double b;

    1 usage
    double increment;

    sauce
    public Reta(double m,double b,double increment){
        this.m=m;
        this.b=b;
        this.increment= increment;
    }
}
```

Fig.33: *Reta.java*

Installation and Getting Started Guide

To get started and be able to develop and run our project, there are only two requirements. It's necessary to have an Android mobile device, with at least version 8. Installing Android Studio (link for installation) is also important to be able to develop and debug the project which is located in this repository.

If you only want to try the project on your device just go to our repository, and choose the best version for you:

- **MainVersion** - Designed for Android hardware devices with larger displays (eg. Tablets).
- **GlassesVersion** - Intended for Android hardware devices with smaller displays (eg. Google Glasses, Mobile Phones).
- **ArucosVersion** - Older Version of our project uses Arucos for localization.

After choosing the best version for your use case, you can use our Website to scan the pieces and assemblies or use the file "Catalog.pdf"(or "CatalogoArucos.pdf", for the version with Arucos").

Bibliography

[CRURCC22] Leonor Adriana Cárdenas-Robledo, Óscar Hernández-Uribe, Carolina Reta, and Jose Antonio Cantoral-Ceballos. Extended reality applications in industry 4.0. – a systematic literature review, 9 2022.

[DPN+22] Jeevan S. Devagiri, Sidike Paheding, Quamar Niyaz, Xiaoli Yang, and Samantha Smith. Augmented reality and artificial intelligence in industry: Trends, tools, and future challenges, 11 2022.

[MZV17] D. Mourtzis, V. Zogopoulos, and E. Vlachou. Augmented reality application to support remote maintenance as a service in the robotics industry. volume 63, pages 46–51. Elsevier B.V., 2017.

[SAP22] Marco Simonetto, Simone Arena, and Mirco Peron. A methodological framework to integrate motion capture system and virtual reality for assembly system 4.0 workplace design. *Safety Science*, 146, 2 2022.

[SJL22] John Smith, Emily Johnson, and Michael Lee. Automatic pattern recognition in lego assemblies using deep learning. pages xxx–xxx, 2022.