Documented By: BRANDON WINATA NG SHAO QI (190501P)

NYP Diploma in Aeronautical and Aerospace Technology

CFM56-7B AR Scanner application

Conceptualization Stage

A picture containing text, clipart

Description automatically generated

# Abstract

This project aims to further expedite, as well as improve the standards of the Inspection Process carried out by the engineers. By integrating this operation with technology known as Augmented Reality (AR), we can achieve the best solution that increases the efficiency of the whole Inspection Process.

# Acknowledgements

I would like to thank my School Internship Mentor (SIM), Mr Chnioh, for giving me such a great opportunity to be working on such an amazing project. This made me realise how much I was into computing, and hopefully this passion continues to stay ablaze.

I would also like to thank my Industrial Internship Mentor (IIM) Mr Li Guo Hao, for his continuous guidance on this project. Without all these resources that he has provided, we would not have gotten anywhere near where we stand on this project.

Contents

[Abstract 1](#_Toc70953366)

[Acknowledgements 1](#_Toc70953367)

[Introduction 3](#_Toc70953368)

[Objectives 3](#_Toc70953369)

[Progress Timeline 3](#_Toc70953370)

[Proof of Concept 4](#_Toc70953371)

[Image Target 4](#_Toc70953372)

[Ground Plane Recognition 5](#_Toc70953373)

[Standard Model Target Database 6](#_Toc70953374)

[Advanced Model Target Database 6](#_Toc70953375)

[Key Features 6](#_Toc70953376)

[User Interface 7](#_Toc70953377)

[Data Input Fields 8](#_Toc70953378)

[Counter System 9](#_Toc70953379)

[Sequential System 9](#_Toc70953380)

[Wireframe for Engine CAD Model 9](#_Toc70953381)

[Multiple Display System 9](#_Toc70953382)

[Implementation 10](#_Toc70953383)

[Software Development 10](#_Toc70953384)

[C# Scripts 10](#_Toc70953385)

[User Interface 11](#_Toc70953386)

[CAD Model for CFM56-7B 11](#_Toc70953387)

[Multiple Display System 11](#_Toc70953388)

[Hardware Development 11](#_Toc70953389)

[Integration and Testing 12](#_Toc70953390)

[Key Technical Issues and Solutions 12](#_Toc70953391)

[Conclusion 12](#_Toc70953392)

# Introduction

After receiving the engine from the outside manufacturers, the engineers would have to undergo a tedious process of inspection to ensure the best quality is delivered to the customers. This application aims to expedite this process while ensuring that the room for error is kept to a minimum.

# Objectives

This application would be created to superimpose a holographic clone of the engine onto the real-life engine itself. The hologram would clearly display the inspection points in sequence, allowing users to quickly pinpoint the locations of the inspection points, as well as keep track of their progress.

# Progress Timeline

The entire project is split into 3 phases: Research phase, ideation phase and the implementation phase.

Before starting on the project, we exposed ourselves to an 18-hour long Vuforia Course in order to gain sufficient knowledge of it to work on the project.

After research phase, we moved on to think of how the app should be, what features the app should have, etc. We made a list of features that could be done, some of which were chipped in by our supervisor and the boss, Mr Vincent. After we gained a comprehensive list to consider, we ruled out a few features that were unrealistic/impossible to derive.

Diagram

Description automatically generatedSoon after ideation, we proceeded to work on the features. The process was tedious, as we had to continuously test it out on the real-scale engine to see if it works. If it did not, we had to find a solution to work around it.

# Proof of Concept

Before going deeper into the design process, a basic prototype was created to test whether an augmented model will be able to be superimposed based on a few known methods. These methods are:

* Image Target
* Ground Plane Recognition
* Standard Model Target Database
* Advanced Model Target Database

## Image Target

This process, known as Image Targeting, is when you want the AR camera to superimpose the given 3D model when it has recognised the image provided in the database. The AR camera can detect feature points that are present in the image itself. The consistency of the results depends on how many features the image has, and how clear the image is. The orientation of the 3D model also depends on what orientation the image is being recognised from.

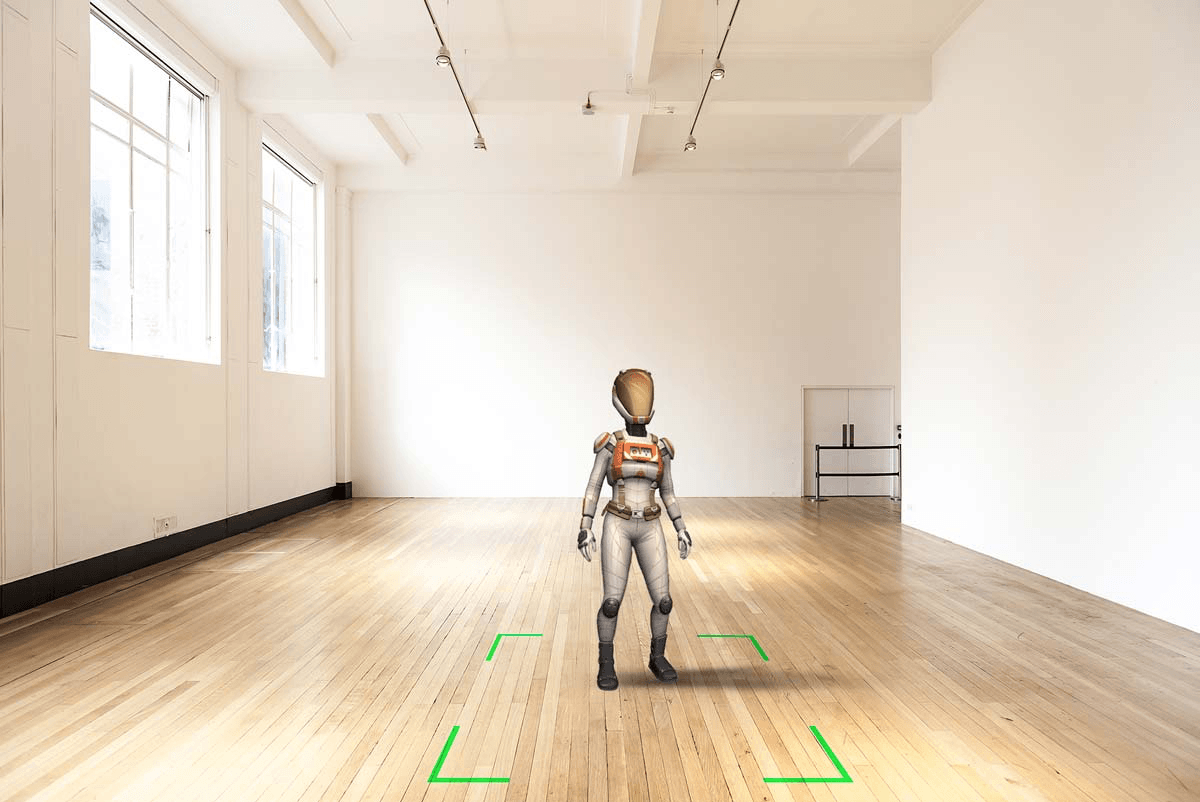


*Fig. 1 Image Targeting*

After experimenting with this form of AR, it does not seem to be effective in this project. A major flaw that exists is that the instant the Image Target goes out of sight, any change in orientation made with respect to the user’s device would not be tolerated, causing the augmented 3D model to “drift”. Knowing that during the inspection process, since the user would most likely be too busy with the inspection to keep track of whether the Image Target was still in sight, we decided to explore other means to superimpose the 3D model.

## Ground Plane Recognition

Ground Plane Recognition is an AR mode that does not rely on Markers. Instead, the user can choose any flat, horizontal surface to superimpose the 3D model on. That is known as Ground Plane Recognition.



*Fig. 2 Ground Plane Recognition*

Although this mode of AR is flexible as it allows us to place the 3D model freely, we need a constraint to fix the AR model in only 1 spot. This mode does not work as it would be simply ineffective to pinpoint an exact location of where to place the model in order to align it to the actual model.

## Standard Model Target Database

Instead of recognising an image like Image Targeting, the AR camera detects features from a 3D object and superimposes a 3D Model provided by the database. This process is called Model Targeting.

In Guided View Model Targeting, we use a software (Model Target Generator) to generate a Guide View for the model. The guide view is rendered to help the user align the camera to the actual object. The position of the outline generally has some tolerance on how exact you need to match the outline. After matching the outline, the 3D model will be superimposed.

From our experience, the 3D model does not seem to be able to continue being tracked after moving to a certain orientation. Moreover, after losing sight of the actual model, you will need to go back to the same orientation to match the outline again, which is not convenient as the actual model is very large, and it will take some effort to match the outline again.



*Fig. 3 Model Target Outline*

## Advanced Model Target Database

Although it is somewhat similar the Standard Model Target Database, this form of AR does not utilise the outline to match the real-life object. Instead, using a deep learning process, the Model Target is trained such that it supports recognition up to 360°.

Again, from our experience, it seems that this mode of AR is quite suitable. Even if the actual model was taken out of sight momentarily, the augmented model appears in the same location after detecting the actual model again. Also, it can continuously track the actual model from every angle.

# Key Features

With the idea of Advanced Model Target Database in mind, we proceeded to brainstorm on what features should be included in the app itself. We came up with the following, inclusive of ideas/feedback given by our supervisors:

* User Interface
* Data Input Fields
* Counter System
* Sequential System
* Wireframe for Engine CAD Model
* Multiple Display System

With these considerations in mind, we created a prototype app that covers all these features, which I will be explaining in further detail.

## User Interface

User interface consists of the login screen, the instruction panel and the inspection panels.

Login Screen consists of the Start and Exit buttons. The “start” button changes the scene into the Main AR Scene where the scanning process happens. The “exit” button quits the app.

A picture containing chart

Description automatically generated

*Fig.4 Login Screen*

Instruction Panel consists of instructions for the user to read, and the continue button. Pressing “I understand!” will close the instruction panel and start the inspection process, opening the Main Scene.

Graphical user interface, text

Description automatically generated

*Fig. 5 Instruction Panel*

Inspection Panels consist of component name/number, inspection procedures, data input fields and save buttons. Upon pressing “save”, inspection panel will be closed, and next button will appear.

Text

Description automatically generated with low confidence

*Fig. 6 Inspection Panel 1*

## Data Input Fields

As seen from Figure 6, the data inputs are as such. Many more input fields can be created, depending on the actual procedures required to be carried out. Currently, there are no databases (e.g., mySQL) installed into the program, so all these data are not being stored anywhere.

## Counter System

Throughout the whole process, the inspector may miss one or more inspection points due to negligence or carelessness. Thus, in order to prevent that, we implemented a Counter System to keep count of the number of inspection points that are done, and the total number of inspection points present. Every time an inspection point has been completed; the counter system goes up by one. The counter system is coloured in red on the Main Scene Display.

## Sequential System

In another effort to reduce any risks of skipping an inspection point, a sequential system has been implemented. This system basically allocates every inspection point to a number, and the inspector must complete all the points in ascending order. If Inspection Point 1 is not complete, the button for Inspection Point 2 will not show. This way, we can be sure that every inspection point will be worked on.

## Wireframe for Engine CAD Model

Originally, the augmented 3D Model was a solid, opaque object. However, the feedback we received was that engineers from other displays (using the Multi-Display System) would not be able to join in the inspection process as they were not able to see the real-life object on the screen. Taking that into consideration, the augmented model must then be translucent while still having the details being outlined such that it is still see-through. This was easily achieved, just by altering the textures of the augmented model itself.

A picture containing web, outdoor object

Description automatically generated

*Fig. 7 Wireframe of CFM56-7B*

## Multiple Display System

The idea is for other engineers, in a separate location, to join in the inspection process using another display. While the engineer on-ground is doing the inspection, the rest looking at the display can make doodles onto the screen, marking out any spots that the on-ground engineer may have missed.

# Implementation

After spending a few weeks researching and ideating, we spent the next few weeks developing the very first prototype. The tasks ahead can be categorised into the following.

## Software Development

### C# Scripts

Coding is a very tedious process. After visualising what I had in mind, I needed to convert those ideas, into lines of digital code.

Unity Vuforia utilises a programming language known as C#. When we first started, I only had learned Python prior to the start of the project. Nevertheless, I picked up C# as I went along and managed to get everything working.

In this section, I will be explaining briefly on what each code does. It would be useful to have prior knowledge of C# coding before trying to do the API Scripting in Unity.

#### **ChangeScene.cs**

This script is responsible for switching up the scenes in the app, depending on which method gets called. For example, in the LoginScreen Scene, after clicking on “Start”, the method *LoadPrototype1CFMScene()* gets called, which executes *SceneManager.LoadScene("Prototype1usingCFM567B").* This basically changes the scene to the scene with the name “*Prototype1usingCFM567B”*. The rest of the code serve similar functions, just with different scene names.

#### **CloseInstructions.cs**

This script is responsible for closing the Instruction Panel that pops up after pressing start. In unity editor, this panel is named “AtStartPanel”.

When *closePanel()* gets executed, it disables the Instruction Panel, and enables the first button B1. At the start, you will notice that all the buttons on the Augmented Model (B1-B6) are disabled. This is part of the sequential system.

Once B1 is enabled, user can continue with the inspection process.

#### **WebsiteButton.cs**

This script is responsible for allowing gameObjects to behave like buttons. After creating this script, you must allocate it to the desired gameObject you want to behave as a button and define it as the “definedButton”. After that, give it whatever program you want it to run OnClick().

In this instance, once the gameObjects are clicked on, they will execute the *sequence()* method in ***OpenInspPanel.cs****.*

#### **OpenInspPanel.cs**

This script works hand in hand with CloseInspPanel.cs to make up the brains of the Sequential System. It assigns itself a private variable “sequencescore”. This serves as a loop counter and remembers how many times the method has been called.

The first time it is called is when the user presses B1. *sequence()* will be executed with the *sequencescore* being *0*. This will execute the first “if statement”, which will enable Button1Popup (also known as Inspection Panel 1). The loop counter will then add one to itself.

The second time it is called is when the user presses B2. Now, the method called is with the *sequencescore* carrying the value of *1*, which executes the second “if statement” to enable button2Popup (Inspection Panel 2).

This repeats until all Inspection Panels have been done.

#### **CloseInspPanel.cs**

This script works hand in hand with CloseInspPanel.cs to make up the brains of the Sequential System, as previously mentioned. It also has a self-assigned private variable “*sequence2score*”, serving the same purpose of acting as a counter.

The first time it is called is when the user presses “Save” in the Inspection Panel. *sequence2()* will be called with the *sequencescore* being *0*. This will execute the first “if statement”, which will disable button B1 and the current Inspection Panel while also enabling button B2. The loop counter also adds 1 to itself.

The second time it is called is when the user presses “Save” again in the Inspection Panel. Now, the method called is with the *sequencescore* being *1*, which executes the second “if statement” to enable the next block of code, respectively.

This repeats until all Inspection Panels have been done.

#### **CounterManager.cs**

This script is solely responsible for the inner workings of the Counter System. It also has a self-assigned private variable “*sequence3score*”, serving the same purpose.

After the user presses the “Save” button in the Inspection Panel, *sequence3()* will be executed. *sequence3score* adds itself by 1. Then, the following code will change the Counter Display’s value (in Unity, this entity is a text field named “counter”) into the current *sequence3score* value.

This repeats for all values of sequence3score as more Inspection Points are completed.

### User Interface

Creating the user interface was simple. The general purpose of a good user interface is to allow the user to navigate through the app with ease. As seen from the previous figures, the user interface is effective, yet simple enough to get the user to navigate.

### CAD Model for CFM56-7B

The original CAD Model was supplied to me by Ting Rui, which I used to import into the Unity Project. However, the asset file format was not compatible, a problem I easily solved by downloading a file format converter online.

Then, since the model texture desired was a wireframe, I explored all the textures available in Unity until I found the desired texture.

### Multiple Display System

Due to the overwhelming rendering time and lag issues seen in the testing phase, we decided not to include this feature inside the app for now. This feature could be further investigated once the app has been developed such that lag is no longer an issue.

## Hardware Development

In terms of hardware, there was not anything much to “develop”. However, the camera quality was a big issue. When we first built the app into the Microsoft Surface Go, we could not get any response from the scanner. On the contrary, when using my mobile device (OPPO Reno 3 Pro), the AR model was able to be superimposed onto the real-scale engine. More about this at the “Key Technical Issues and Solutions” part.

## Integration and Testing

After completing my Prototype app, we went ahead to test it out on a real-scale engine. Before that, there was one existing problem:

* When scanning the smaller-scale model, the drifting effect seems to take place very often. In lay man terms, the augmented model was unstable.

However, this problem ceased to exist when we got to the larger model. That was because with respect to the large-scale model, the camera device was smaller. Hence, there was a lesser tendency to scan different feature that we are not currently facing, which causes the random jittering in the augmented model.

Of course, scanning a larger model not only fixed our problem, but presented itself with new ones:

* Even after directing the camera towards the engine, it takes an absurdly longer time to load.
* Even when it loads, there is a chance that the model is superimposed in an incorrect position.

## Key Technical Issues and Solutions

We racked out brains trying to figure out the reason behind why the AR Scanner might be facing that issue. There were a few reasons that surfaced:

* The camera quality of the device is not good enough, hence it has trouble detecting the feature points of the engine model itself.
* The real-life engine in the workshop looks different from the CAD Model given, thus it is harder for the camera to detect the features that it was trained to detect.
* The size of the CAD model is so big, when trying to render a real-scale model, it takes a longer time to load.

First, we tried getting a better device to install the application onto in hopes of rectifying Reason (1). So, we installed the App into a Microsoft Surface Pro 7, which has better specifications than the Surface Go. However, this improvement did not rectify the problem.

Second, we tried installing a webcam to the Surface Pro 7 to see if it takes a good camera to rectify this problem. The camera we used was the Intel(R) RealSense™ Depth Camera 455 RGB. However, later I learned that Vuforia apps were not compatible to use with Webcams, they can only use the device’s inbuilt camera.

Currently, we are trying to use Guided View instead of Advanced view. Personally, I do not think this will work as we are just changing the way the model is being recognised, but since we had nothing at that point, it would not hurt to try.

# Conclusion

## Future Development Considerations

There are many other features to be explored in the future, so do consider the following:

* A database to communicate with. This will be the most important part of the software, so this is feature is of the highest priority to be researched and developed on.
* The Multiple Display System.
* A feature that allows technicians on ground to submit information instantly to the engineers. Currently, whenever there is a defect/discrepancy on the engine, they will have to page the engineers to look, pausing the inspection process altogether. Using this feature, it will allow work progression to continue without needing to wait for engineers to respond.

In terms of using this app itself, there are many other operations that could use this, along with minor modifications if necessary. Operations include:

* Dispatch Check
* Other Inspection Processes

If we want to talk about developing a totally different app for a different operation, there are still many areas available for AR development, like:

* Maintenance Repair Overhaul (MRO)
* Wire Harnessing
* Personnel Training
* Many other possible applications

## Documenter’s words

With this new idea, it is just the start. I have laid out the foundation that future interns can work from. There are many improvements that can be made, that is for sure. But with the technological advancements that humankind is creating throughout the years, I believe we can greatly improve this software in the future.

For now, it may seem like nothing. There are many ideas that are just waiting to be put into life. Other improvements like Dead Reckoning would be essential in this project. Maybe there is another way to get a camera with great specifications to work on Vuforia. The potential that this technology has is overflowing, and it is up to the future interns to grasp it.

This document was created with the intention of passing on the torch to future interns. I hope that this document will hopefully give you a clearer picture of what exactly is going on in the project itself. This is just the beginning of what could be a formidable software that can drastically improve the efficiency of many operations in ST Engineering. Never let your knowledge limit your creativity!

## Appendix

Fig. 1 <https://library.vuforia.com/articles/Solution/Native-Image-Target-Sample.html>

Fig. 2 https://blogs.unity3d.com/2018/01/15/vuforia-in-unity-build-cross-platform-ar-apps/

Fig. 3 <https://library.vuforia.com/articles/Solution/model-target-guide-view.html>

Google Slides for Concept Presentation: <https://docs.google.com/presentation/d/12xvGFr7X-3dYpD9hHmKASJ-X2IyqWrvLWqAP2yusYJU/edit?usp=sharing>