Project Report
Master in Mobile Computing

AR Kart: Towards an Adaptable Solution For Virtual Elements and Real World Interaction in Mobile Applications

Dian Wang

Leiria, *March 2019*

Project Report
Master in Mobile Computing

AR Kart: Towards an Adaptable Solution for Virtual Elements and Real World Interaction in Mobile Applications

Dian Wang

Report developed under the supervision of Doctor *Gustavo Reis*, professor at the School of Technology and Management of the Polytechnic Institute of Leiria.

Leiria, *March 2019*

Acknowledgements

I would like to thank Professor Gustavo for his supervision during this project.

Abstract

Mobile devices entered its booming era in augmented reality with the launch of Apple's ARKit (September, 2017) and Google's ARCore (Preview in August, 2017). Both frameworks provide developers with solution, which doesn't need external devices for AR experience. Common mobile devices are able to deliver AR effects with built-in back and/or front facing camera. For this master project report, the possibility of creating an interactive AR experience will be discussed. Approaches including successes and failures will be examined and the elected solution will be presented.

The first step will consist of hardware designing and related software approaches. For this project the goal is to take advantage of common objects in everyday life without evolving any complicate customised hardware and make the project able to be duplicated by anyone who is interested in it. Due to the fact low cost Arduino and toy car chassis was selected as potential candidates.

The core of the project is to use real world objects to interact with virtual object created in augmented reality while the theme of the project is car racing. In this case the user controlled toy car is expected to interact with virtual game objects such as goal, checkpoints and / or specific game characters. To facilitate the interaction the car is supposed to be recognised and tracked by the game. In other words the user controlled car is expected to be recognised as a part of the AR Kart game instead of being separated from the game like other irrelevant objects in user's real world environment.

Once the tracking reaches satisfactory performance a demo game scene will be created in the application. The game scene plays the role of stage for demonstrating interactive AR gaming experience. Further usage of such solution will also be discussed.

Keywords: Augmented Reality, AR interaction, ARKit, ArUco

List of figures

List of tables

List of acronyms

2D	Two Dimensions
3D	Three Dimensions
API	Application Programming Interface
AR	Augmented Reality
BLE	Bluetooth Low Energy
CNN	Convolutional Neural Network
FPS	Frames per second
IDE	Integrated Development Environment
IoT	Internet of Things
OpenCV	Open Source Computer Vision Library
OS	Operating System
SDK	Software Development Kit
SLAM	Simultaneous Localization and Mapping
VIO	Visual Inertial Odometry
VR	Virtual Reality
WWDC	Worldwide Developers Conference
YOLO	You Only Look Once

Table of Contents

1. Introduction

In the year of 2017 with the launch of ARKit (iOS) and ARCore (Android), two major mobile devices' platforms have been provided with their dedicated augmented reality (AR) SDKs. Creating AR enabled applications, including games, is officially supported by the industry's leaders. Together with the new SDKs, Apple's Xcode with iOS 11 and Google's Android Studio with Android bridges the AR features in a smooth manner, which gives developers the ability to build immersive objects with plane detection [1], dynamic lightening and physical collision systems. The created virtual objects are inserted directly into the device's camera feed.

Checking out different popular AR apps and games available on App Store and Google Play, it appears that the majority of the works follow the 'Screen - Camera feed' pattern that allows users to interact with AR objects / game scenes on the device's screen. This means that despite offering an apparent immersion in an augmented reality world, there is still a strong separation between the AR world and real world. The application / game has no ability to recognise collision or other forms of interaction between virtual objects and real world or real objects. In short, a player's touch on the AR objects he sees on the screen will result in nothing. This project's goal is to challenge this limitation by creating a way of interaction between real world objects and virtual objects.

Previous to ARKit and ARCore, there have been solutions developed and open sourced such as ARToolKit [2], ArUco [3], as well as commercial solutions such as Vuforia [4] and Wikitude [5]. Similar to ARKit and ARCore, these solutions provide AR plane detection, object / scene rendering, environment effects simulating (eg. Light estimating and particle system) and some levels of object / image tracking. Compared to virtual reality technologies, AR SDKs and solutions don't commonly offer developers direct interaction with the virtual objects and / or scenes created by the AR enabled applications. Table 1 shown below provides a clear comparison between popular AR SDKs.

	Vuforia	Wikitude	EasyAR	Kudan	ARToolKit	Maxst	Apple ARKit
Licence	Free, Commercial	Commercial	Free, Commercial	Free, Commercial	Free Open Source	Free, Commercial	Free
Supported platforms	Android, iOS, UWP	Android, iOS	Android, iOS, UWP, macOS	Android, iOS	Android, iOS, Linux, Windows, macOS	Android, iOS, Windows, macOS	iOS
Smart glasses support	+	+	-	-	+	+	+
Unity support	+	+	+	+	+	+	+
Cloud recognition	+	+	+	-	-	-	+
3D recognition	+	+	+	+	-	+	+
Geolocation	+	+	-	-	+	-	+
SLAM	-	+	+	+	-	+	+

Table 1: Mainstream AR SDKs
Source: [57]

This table reveals some common features of popular AR SDKs that are SLAM (Simultaneous Localization and Mapping, "a technology which understands the physical world through feature points" [6]) supported, Unity supported and Smart glasses supported. One noticeable fact is that ARKit is the only free SDK that supports all the features. As expected, mainstream AR SDKs are focusing on displaying virtual contents rather than providing first person interaction with the virtual environment. None of the SDK offers direct solution to interaction between real world objects and virtual world objects. Another noticeable fact is the trend of enabling object tracking, starting from Apple's launch of ARKit 2. 3D object recognition and image tracking has been added as new features to this SDK [7]. The addition of object tracking / recognition indicates that a company in a leading position in the field started making effort in enhancing the interaction between users and AR environment. Indeed, the tracking function is the first step towards greater immersive experience, which will eventually lead to more interaction between real and virtual.

This project, as discussed above, aims at 'creating a way of interaction between real world objects and virtual objects, which has the potential to contribute to the next level of augmented reality applications' development. This is where the project starts and develops a possible prototype with simple components and public-released SDK. In this project, the Arduino smart car 'AR Kart' is the real world object in the game app while the player's goal is to drive the car to gain points by hitting virtual cubes in a limited time. AR objects including waypoints and special object (explosive) play the role of interactive virtual objects.

According to the skill and specialty of the developer, ARKit was selected as the candidate SDK for the project, which is an iOS only SDK. Unfortunately ARKit alone is not able to support the necessary fast moving object tracking (Tracking of the smart car in this case), based on its current development. This problem aside, the strong point of ARKit is its robust environment understanding ability, which includes vertical and horizontal plane detection, swift anchor placing and mature particle, physics systems by utilising Apple's SceneKit alongside. Because of this, using ARKit remained the best solution, but it became necessary to find a solution to fit the necessity of fast tracking. The project's prototype being a mobile device application, ArUco code, which is based on OpenCV, was eventually selected as the assistance method. A smart car that carries a printed ArUco code was built in order to maintain an acceptable object tracking performance. After this stage the smart car is 'covered' by an AR node that follows the car's movements. Based on a well-tracked smart car, the game scene is constructed by ARKit and reacts to the interaction between the car and virtual objects. As a result, a user interactive AR app solution is created. To reach the final solution briefly introduced above, the project has gone through multiple stages with several attempts that involve different technologies. Details of each stage and the development of the final solution will be discussed in the following chapters. Discussion on the further usage of the project's final solution as well as the implication for future research on this topic will be carried out at the end of this report.

2. Background

This chapter presents the key elements related to this project. As expressed in the introduction, this work is based on Augmented Reality technology. Therefore, we will first explore the concept of Augmented Reality. Then we will specifically introduce the more specific topics of Marker-based Augmented Reality and ARKit. In a last part, we will present the various technologies and tools used to build the smart car (Arduino, Bluetooth low energy…).

2.1. Augmented Reality

Azuma stated, "*Augmented Reality* is a variation of *Virtual Reality*" [8]. While Virtual Reality allows users to immerse themselves in a fully virtual environment, Augmented Reality offers the possibility to integrate virtual elements in a real life environment. Furthermore, the Gartner IT glossary more specifically defined AR as "the real-time use of information in the form of text, graphics, audio and other virtual enhancements integrated with real-world objects" [9].
Still according to Azuma's definition, AR not only introduces virtual elements in a real environment, it is also interactive and three-dimensional. In this case, AR is not solely limited to head-mounted structures, but can also be applied to any device that offers the opportunity to provide an immersive experience that combines real and virtual [8].
The term "Augmented Reality" was first coined in 1990 by Thomas Caudell and David Mizzel [10], but L. Frank Baum writings back in 1901 are the earliest reference to a type of Augmented Reality technology. Indeed, the author of the famous book The Wizard of Oz' introduced his 'character marker': a pair of spectacles allowing the user to see data overlaid on reality - here letters places on each person's forehead [11] which perfectly fits Gartner's definition.
Harvard Professor Ian Sutherland developed the first head mounted display in 1968, but History shows that Augmented Reality really kicked off in the 1990s. Indeed, "Louis Rosenberg at the U.S. Air Force's Armstrong Laboratories […] developed the first immersive AR system" [12]. This system, called Virtual Fixtures, allowed a person to remotely effectuate tasks by using a helmet that provided guidance through digital overlays and controllers connected to robotic arms [13]. This system provided better 3D perception of the remote environment, therefore achieving better accuracy.
In the 2000s, AR started to be more prominent in the civilian scene. Quake AR, the first Augmented Reality game, was launched in 2000 [14]. The problem with early Augmented Reality technology was its lack of mobility: the first mobile solutions dating back from 1997, required to travel around with a large backpack style computer, which was not very convenient [15]. However, in 2005, Augmented Reality was brought to mobile devices: AR Tennis, a duo player game was built for Nokia's S60 series mobile phones [16]. Suddenly, Augmented Reality technologies became available in the palm of the hand. The following year, Nintendo introduced the WII, a console using a motion-sensitive controller that gave players the most immersive experience in video games at the time [16].
Since then, AR technologies have been widely adapted to different industries, from gaming and entertainment to medicine and security. In fact, spending in AR and VR are expected to pass the 20 billion dollars mark worldwide this year, a clear indication of the popularity of this technology [17]. The application of AR to games has been researched since the end of the 1990s. For example, Magerkurth et al. created AR KnightMage, an

adventure AR game using rear projection to place the AR environment on a horizontal surface. The dices and avatars interact with the AR world [18,19]. This is of course only one example of the multiple articles that have been written on he topic.

The development of AR applications in the 2000s was also made easier by the release of the ARToolKit, the first open-source framework providing engineers and developers with the necessary tools to build real-time AR applications with the use of markers [20].

2.2. AR Frameworks and Libraries

As stated earlier, first 'mobile' AR devices were in fact extremely bulky and inconvenient to move around. However, with the arrival of the smartphone, AR found a new vehicle, and technology had to be adapted to fit such small devices.

2.2.a ARToolKit

The ARToolKit is celebrating its 20th anniversary this year, as it was created in 1999 by Hirokazo Kato. In 2005, it became the first AR SDK to be ported on a mobile OS - Symbian OS - in order to achieve adequate performance in further gaming development [15]. Nokia'a AR Tennis took advantage of ARToolKit and S60 mobile operation system to deliver a brand new experience for AR gaming [15].
Indeed, Anders Henrysson developed a "highly-optimized version of the ARToolKit library for the Symbian platform", which he then used to build the AR Tennis game [21]. In 2008 it was running on the iPhone 3GS and in 2010 on Android. Nowadays it is not only available on mobile (iOS, Android) but also on Mac OS, Linux and Windows. Furthermore it is integrated in Unity as a plugin, and it is still widely used by developers nowadays. However, its limitations in terms of tracking are such that new frameworks and libraries are now offering better options for more mobile work.
Nowadays however, while ARToolKit remains a pioneer in the field of marker based AR technology, other frameworks are being developed to fit developers needs.

2.2.b ARKit

Apple released the first version of its ARKit API on June 5, 2017. It was conceived as a tool for developers dealing with augmented reality and virtual reality in the development of native applications. When it was introduced, the demonstration showed that "ARKit had the potential to place complex scenes on a surface within the view of an iPad, and that Apple had solved some of the tougher problems developers had working within AR" [22]. Until the introduction of the ARKit, iOS developers lacked a native tool to implement augmented reality function in their app. With this toolkit, developers now can rely on a technology that does a lot for them, including the analysis and mapping of the environment, such as suitable surfaces and lighting. "ARKit lets developers place digital objects in the real world by blending the camera on the screen with virtual objects, allowing users to interact with these objects in a real space" [23]. ARKit technology was compatible with "iOS device that runs on Apple's A9, A10 or A11 Bionic processors. That includes the iPhone 6s and 6s Plus, iPhone 7 and iPhone 7 Plus, all iPad Pro models, the

9.7-inch iPad released in 2017, the iPhone 8, 8 Plus and iPhone X" [24].

ARKit allows both "world tracking" and "face tracking". World tracking "create AR experiences that allow a user to explore virtual content in the world around them with a device's back-facing camera" while face tracking "uses the front-facing TrueDepth camera to provide real-time information about the pose and expression of the user's face for you to use in rendering virtual content" [25].

The first version of ARKit focuses mainly on horizontal surfaces such as floors or table, to place 3D virtual objects into the real world. Thanks to its environment texturing tools, the virtual objects can be better integrated in the real world through good analysis of the lighting for instance. ARKit can also recognize 3D objects: once the object has been analyzed and its featured recorded, ARKit will have the ability to recognize it when mapping the environment [25].

Since Apple first introduced the ARKit in June 2017, there as been two major updates, ARKit 1.5 and ARKit 2.0:
ARKit 1.5 brought a new big feature, which is using vertical surfaces to help map the environment. Furthermore, it improved the ability to recognize objects of weird shapes and 2D objects (posters, for example), with better tracking speed and accuracy. ARKit 1.5 also provides images of the world with a better resolution (1080p rather than 720p) [26].

With iOS 12 being revealed this year, ARKit also gets a second version. Three main changes can be identified. The main one is the possibility to for several devices to interact in the same AR app. This means that two players can evolve in a same AR environment, just like a multiplayer mode. At the WWDC on June 4, 2018, this function was demonstrated by LEGO's director of innovation, Martin Sanders, through an original app allowing up to 4 people to join in the same AR environment. The players' actions were visible by the others, allowing them to interact both among themselves and with the environment [27].
Another update brings persistent tracking, which means that once a 3D object has in placed in the AR environment, it will stay there. The player or other players will be able to see it and interact with it at a later time as long as they enter the environment [28].
ARKit 2 comes with an app called Measure, which will allow greater accuracy in measuring and detecting real objects. Measure uses the camera to determine the size of objects with great accuracy, all the user has to do is place points to delimitate the object [29]. Compared to the first version of ARKit, this "lets the framework know when it's looking at something that's got more than two sides" [28].
For ARKit 2, Apple worked alongside Pixar to create a new file format, USDZ, which will allow sharing AR objects among the platforms. "This could greatly expand the reach of AR experiences; image saving and sharing something you created over Messages, then mailing it to a friend who can view it and start manipulating it through their phone" [28].

The functionalities and performance of ARKit are being completed and improved by Apple constantly. Considering the importance of AR and the current trend, the future of ARKit and the technology itself can be predicted as promising according to the market response and the passion from users' choices.
For this project however, ARKit revealed many drawbacks, hence it needed to be complemented with a library called OpenCV, and more specifically ArUco, to achieve the tracking and marking necessary.

2.2.c OpenCV

OpenCV means Open Source Computer Vision Library. Created in 2000 in the Intel Labs, it has grown since its birth to become the “de facto library for computer vision development” [30], and indeed many large companies are relying on it to build their software, including big names like Intel, IBM, Google, Microsoft [31]…
It was “built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products” [31]. The library offers thousands of algorithms that can be used by developers working in Computer Vision, which therefore can be applied to augmented reality, in particular because “OpenCV was designed for computational efficiency and with a strong focus on real-time applications” [32], which is crucial for a real-time immersive augmented reality application. Algorithms available in the OpenCV library “can be used to […] identify objects, […] track camera movements, track moving objects, […] recognize scenery and establish markers to overlay it with augmented reality” [31]. It is compatible with most platforms (Linux, Microsoft, iOS…) and available in several languages (C++, Python…).

2.2.d ArUco

ArUco, according to its official page on the University of Cordoba’s website, is a “minimal library for Augmented Reality applications based on OpenCV” [3]. Thanks to the ArUco library, it is not only possible to detect square markers but also to estimate the camera pose, which allows greater efficiency in implementing augmented reality elements. “ArUco is very robust and can detect markers in a very wide range of situations” [33].

2.3. Hardware

We talked about the background for the software part, and now we will quickly introduce the main hardware elements that went into building the smart car, for future reference.

First, the car functions with an Arduino Uno REV3. Arduino boards are microcontroller boards that are able to receive instructions and be used for many DIY projects. “Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping” [34]. By adding different modules and tools, developers can use it for a large variety of functions, from robotics to smart objects [34].
The Arduino Uno is the most commonly used board, adaptable to many creative ideas and therefore a good place to start for any project [35].
Another essential element is the Bluetooth Low Energy board HC-08. Bluetooth Low Energy is “a wireless technology which […] is intended to provide considerably reduced power consumption and lower cost [compared to Bluetooth], while maintaining a similar communication range with the devices” [36]. BLE can run for years with extremely small energy consumption, making it particularly useful for IoT, robotics or other smart devices projects [37].

3. Related Works

There are two large research areas that are related to this project. They are computer vision and virtual reality. There are works from both fields which inspired and assisted the development of the project. This chapter is going to discuss the impact of related works on project AR Kart.

3.1. Adrian Rosebrock's Ball Tracking Project

Computer vision isn't a fresh field. Good amount of works and contributions have been done and many of them are shared with the public through open source on the network. During the development process of the project, computer vision related works have not only provided inspiration to the developer but also directly assisted the compilation of the project. OpenCV, a real-time computer vision library originally developed by Intel, who decided making it open source with BSD license [38] demonstrated a robust real-time object-tracking feature. Among the showcases of OpenCV's efficiency are projects such as Adrian Rosebrock's ball tracking project, which is a particularly good example of object tracking. It is a simple look OS X application that tracks a green ball in the developer's hand and draws red movement tracks on the screen [39]. The project uses OpenCV's colour tracking function that captures the RGB colour range changes in the video feed. In this case, the developer sets the upper and lower boundaries of the colour green in the HSV colour space thus detect the location of the green ball in the video file. The project showcased the robust performance of OpenCV's object tracking and the stability of the tracking. Compared to the previous solutions including ARKit's native image tracking or neural network assisted tracking, this project shows that continuous object tracking is possible by integrating OpenCV library.

The inspiration from ball tracking project leads to the final solution of the AR Kart project which is using ArUco with ARKit which will be discussed in later chapters. Adrian's ball tracking project didn't only showcased OpenCV library's ability in high performance object tracking it also brought further questions to the project.

- Weather it is possible to use such library to track the smart car itself as a complicated object?
- If it is impossible to track the whole smart car then will it be any options to perform the tracking feature?
- Will the performance be satisfactory or acceptable for players to experience an AR game on minimum hardware configuration according to Apple's guidance for ARKit compatible devices (iPhone 6s/SE with A9 chip) [40].

The questions laid the foundation and starting point to introduce ArUco solution to the project and it was eventually selected to be the solution for AR Kart's iOS application development.

3.2. VR Mario Kart

New York based researcher/founder at Onirux Labs Asier Arranz's VR Mario Kart prototype project is another important inspiration [41]. This project built a prototype of VR Mario Kart experience by using real hoverboard, Lenovo Mirage headset and Android phone. The goal of the game is to capture as many coins as possible while avoiding obstacles such as the 'bombs' and other vehicles. The player himself sits on a battery-powered hoverboard that is the real world version of 'Mario Kart'. All other objects players can find in Mario Kart will be projected by the VR headset, which also plays the role of screen.

Figure 1 shows the virtual game scene viewed from the headset and the real room scene.

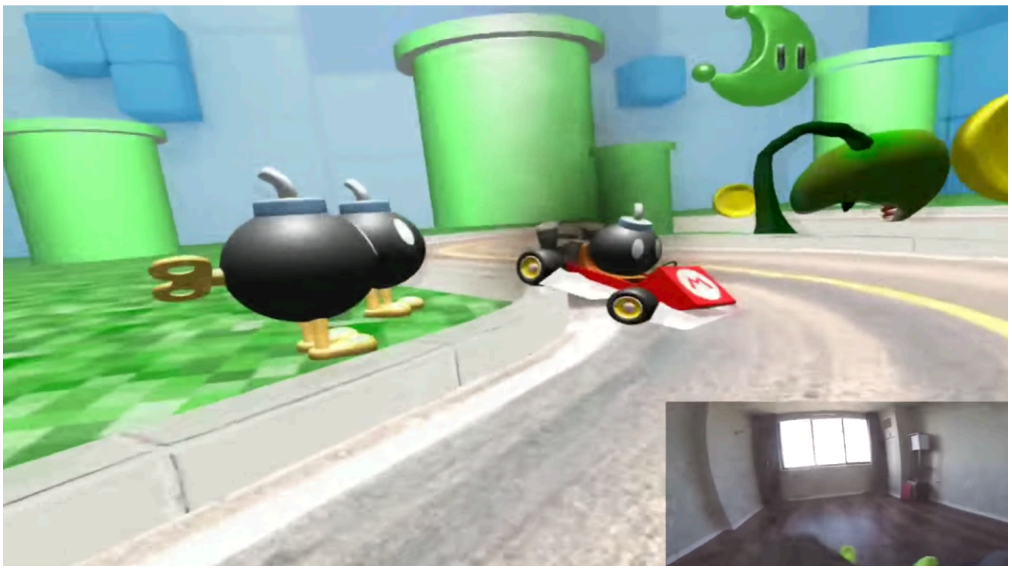

Figure 1: Mario Kart VR Prototype
Source: [61]

Mario Kart VR demonstrates the basic theory of indoor virtual reality game. The developer Asier Arranz made good use of the immersive gaming environment created by VR technology. The player himself is the camera, and together with the virtual scenes, objects around, it brings a truly immersive experience similar to first person shooting games. This design takes into consideration the fact that the player will be 'driving'. First person view is no doubt the best way to simulate the real car driving experience and brings the original Mario Kart's item-capturing feature to the game in a natural way. Our project AR Kart adopted Mario Kart VR's game theory and a similar idea of creating unique scene as a 'map' for the player.

In contrast to the related work, project AR Kart has its own set of available resources, which are different to the OpenCV ball tracking and the Mario Kart VR. Compared to the traditional OpenCV tracking, AR Kart has the support from ARKit, which allows the developer to create lifelike 3D models with real-time light estimation and different rendering effects. Rather than having a simple green ball the AR Kart is able to have any creative 3D contents on the top or surrounding the smart car. Powered by ARKit the background/game scene of the project can be rendered in a professional manner, which isn't available to traditional OpenCV based computer vision works. In the meantime the project's goal is to create an interactive AR experience with inexpensive hardware, which excludes costly equipment such as VR headsets and controllers.

4. Approaches

The project AR Kart has experienced a series of attempts, which include successful ones and failed ones, which led us to finally decide on the final result. This chapter will discuss solutions that have been tried in order to reach the final goal and analyse each of their pros and cons.

4.1. Attempt #1: Image Detection

At the very beginning of the project, the first attempted solution was to use ARKit's image detection feature to realise the tracking of the smart car, thus bringing the smart car into the AR game scene and interact with other virtual objects. In detail, this solution tried to use smart car's picture for image detection. Once the image of the smart car has been recognised by the ARKit, a customised 3D model will be placed over the smart car. The desired outcome will be an effect similar to what Figure 2 illustrates. Once an AR node has been attached to the car, other related game objects and logics can thus be applied to the game and interact with the smart car. The idea behind this is to make use of the physical collisions and effects provided by Scenekit, which are often used together with ARKit. Actual collisions happen between the node to the car and all other game objects. The player is driving the smart car and sees the 'car' performs real world alike physical actions such as 'hitting an object', 'running over objects' or 'crashing into objects', etc. which generates AR effects visible on screen.

Figure 2: Desired Effect of Real World Toy Car With AR Effect Attached To It
Source: [58]

The idea was promising but when it was executed during development process several problems appeared immediately.

Apple has provided exact explanation of its ARKit's image detection feature as an 'add-on' feature for world-tracking AR sessions [42]. It uses known 2D images in the user's environment to enhance a world-tracking session [25]. 'Known 2D images' refers to the images of objects from the real world that the developer intends to detect. Image detection/recognition is a sub feature of world tracking. The goal of image detection is to place (visualise) the developer desired virtual content above or around the target image. The first attempt tried to use this feature. After taking pictures of the smart car, a test

application was built using the car's pictures as reference image and placing a 3D star model on the car's ARImageAnchor which on the screen will result in a virtual 3D star hovering above the smart car. While this solution worked fine when the car was static, the problem occurred once it started moving. Indeed, the 3D model was placed at the car's first detected position but it didn't follow the car at all. In this case it is impossible for the car to activate any effects with other virtual objects in the game scene. At the time of this first attempt, ARKit was at its fresh launch stage. Looking into Apple's documents there are the following elements that limited this feature's application to still images only.

I. The image is detected only once during the application's life circle. This means the ARAnchor that tells the ARKit where to place virtual object is generated once from the first detection of the image till the termination of the application.
II. The reference can only be images on flat surfaces otherwise the image might not be recognised or might create an image anchor at wrong location [42]. In this project's case, the smart car is the detection target. Being a 3D object in real world has limited the ability for the smart car to be detected by ARKit although pictures of the car were used as reference pictures.

It is clear from the document image that ARKit detecting feature may not fit the project requirement. However, as an attempt to solve this issue, the image detection function was forced to run on every camera frame. The result was negative. Test application crashed after exceeding the device's memory capacity.

Pros of the attempt:

I. The speed of image detection is satisfactory. According to the test app's result the average detection time was around 1.5 seconds in good lightning condition. The performance went down to approximately 4 to 5 seconds or even longer when the lighting condition wasn't good (For example, dimmed light, uneven surface lighting and sudden lighted up spots in the scene involving the target)
II. The angle of image detection is wide. From 40 degrees to the best angle 90 degrees towards the surface the reference image can be detected according to the test.

Cons of the attempt:

I. The ARAnchor was misplaced due to the fact our detection target is a 3D real object instead of 2D image.
II. AR 3D model didn't follow the smart car's movement at all. As mentioned above, the 3D model was projected around the smart car but stayed at the first detection position without following the car.
III. The success of image detection relies heavily on the lighting condition, contrast, details of the reference picture(s) and the angle of how the image is placed.

4.2. Attempt #2: 2D Image Tracking

Together with the launch of iOS 12, ARKit received a second version update in June 2018. ARKit 2.0 brought image tracking to developers' toolset. Similar to Vuforia's marker tracking feature, ARKit is able to track certain patterns/images. A distinct difference from traditional marker tracking is that ARKit 2 tracks any user input images which satisfies certain requirements [43]. The image-tracking feature of ARKit 2.0 provides can be described as below:

- Once the ARKit detects the reference image(s), an AR anchor is attached on the surface of the image and follows the movement of reference image.
- Developer can set up a 3D model or any other types of media, for example a video player. The additional media content will follow the AR anchor that is attached to the reference image.
- There is certain degree of motion tracking feature, which will enable the projected media content to follow the oblique of the reference image. If the reference image's oblique degree exceeded ARKit's tracking performance limit (+/- 40 degree towards the plane, data acquired during the project's test runs) the image-tracking feature will automatically stop functioning, which results in the disappearing of the projected media.

The attempt of using image tracking to track the smart car is described as follow:

Step 1. Searching and printing an image that fits the requirements of ARKit's image tracking feature. According to the WWDC's presentation, the picture used as reference picture should meet three requirements that are: 'Distinct features', 'Well textured' and 'Good contrast'. As a result Figure 3 was selected as the reference picture.

Figure 3: The AR Reference Image

Step 2. Attaching the printed reference image on the top of the smart car. The image was attached on the top of the car in order to achieve a better tracking angle and adequate lighting.

Step 3. Loading the image to the application and putting a virtual 3D cube on the top of reference image. As a tracking performance indicator the virtual cube can directly demonstrate the tracking performance in real time.

Step 4. Using an iPhone (6s and greater. In this case the test device is an iPhone 6s) with the application installed to test the tracking performance.

The result shows a robust tracking performance when the car with picture attached is moving in a well-lighted room on low speed (around 0.1 m/sec). The chance of losing tracking increases dramatically with the increasing of the car's moving speed.

Lighting is the second element affecting the tracking performance. The smart car has to move in a room with even lighting. Reflections from metallic objects around and sudden white balance change cause immediate loss of tracking.
The distance between the iPhone and smart car is the third element affecting the tracking: the maximum distance is around 1 meter from the smart car. The player who holds the iPhone has to keep bowing while trying to track the car. ARKit achieves the best tracking performance when the player holds the iPhone from approximately 50 centimeters above the smart car.

As a result ARKit 2.0's image tracking feature isn't suitable for this project due to the following reasons:

- Tracking performance requires strict steady lighting, flat ground and low object moving speed. These requirements can't meet the project's features, which are fast moving object, unsteady lighting and unsteady camera angles.
- The distance limit makes the player/user bow down in order to get better tracking. The user experience is affected by the specific posture.
- Due to the fact that the image tracking feature has no support from machine learning modules when it is used alone, the tracking has low tolerance to differences between the captured camera frames and the reference picture(s). This means that any kind of background changing, lighting changing and integrity of the captured camera frame will cause tracking loss.

Pros of the attempt:

I. Tracking is robust in limited conditions
II. No specific tags/images are required
III. Wide range of media contents can be 'put' on the tracked image (Video, image, 3D model)

Cons of the attempt:

I. The limitations of the environment are strict
II. Tracking algorithm has low tolerance to differences between the reference picture and camera frames
III. As a combination the player loses tracking of the smart car constantly due to the limits and requirements

4.3. Attempt #3: Machine Learning

The previous attempts can be summarised as using purely image processing functions and features from ARKit. The methods and features are fixed without any further learning ability and the only input to the application is the camera frame, which results in picky tracking requirements. The failures that resulted from the two previous attempts indicates the need for a hybrid solution, which still relies on ARKit's virtual objects creating ability but integrating some machine-learning feature, which allows the application to adapt to various changes in both environment and player's posture.
This attempt includes two different approaches: the first hybrid solution used Apple's CoreML plus ARKit while the second solution opted for TinyYOLO and ARKit.

Apple launched its own machine-learning framework CoreML during the 2017 WWDC (Worldwide Developer Conference). The framework simplifies the process of integrating machine learning models into iOS applications [44]. CoreML also creates easy to use interface for integrated model that allows the developer to use the model like a common class. Apart from integrating trained models, CoreML also provides its own tool (Playground from Xcode) for model training. During the first solution development two sets of pictures were used for model training. One set was named 'CarModel' while the other set of pictures was named 'TestImage'. This pattern follows the 80% training data set and 20% test data set rule. The amount of training set has the amount 200 pictures of the smart car from different angles, lighting conditions and backgrounds. 40 test pictures were taken from different scenarios with the smart car inside of the pictures. The model was then trained by using Playground and integrated into the application. This version of the test app passes each camera frame to the CoreML model. The model then analyses camera frames and determines whether the current camera frame contains the smart car. The camera frame will then be forwarded to the ARKit so that the 3D model can be added. The 3D model is supposed to 'follow' the smart car if the frame is classified as containing the car. If the current camera frame wasn't classified as containing the car then the process will be paused until the next positive result is reached.

Combined with the machine-learning model, the ARKit was supposed to have satisfactory rate of detecting the tracking target (smart car) and track it with smoother performance.

The solution didn't make it to the actual testing stage due as it was later found out the playground trained CoreML model doesn't provide location information such as bounding boxes and/or coordinates of the tracking target. The only function it provides is to determine whether the current camera frame contains the training result. In this case the tracking feature isn't available from this solution.

At this stage of the research it was clear that ARKit has a major flaw, which is its inability to track real world objects. It has been introduced by Apple in its developer documents that ARKit 2.0 supports 2D image detection, tracking and 3D objects detection [25]. Figure 4's code block below demonstrates an important and common mechanism for both 2D/3D object detection.

```
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
    if let objectAnchor = anchor as? ARObjectAnchor{
        node.addChildNode(self.model)
    }
}
```

Figure 4: Anchor Placement Method For 3D/2D Objects Detected
Source: [59]

It is clear from the code that each time an AR anchor is placed on the tracked object, it is detected once and such function isn't executed every camera frame, which only makes it able to detect and track still or slowly moving objects/images. This feature also determines that ARKit is unable to place virtual object and keep updating the virtual object's position according to the movement of the tracked real world object.

The realisation of ARKit's robust 3D object tracking would be composed with the following components:

- Fast object recognition
- Frequently updated object location data (eg. Coordinates or bounding boxes) from camera frames
- Fast 3D model rendering

ARKit has fulfilled the last requirement with its swift virtual object rendering, but needs help with the two other components. Tiny YOLO is a lightweight machine-learning framework that has the potential to meet the other two requirements thus it was chosen as the solution after the previous failure with CoreML. Tiny YOLO is an optimised version of its parent work the YOLO neural network.

YOLO refers to 'You only look once.' Which is a short definition of the neural network developed at University of Washington by Joseph Redmon and etc. The developers presented it as "a single neural network [that] predicts bounding boxes and class probabilities directly from full images in one evaluation" [45]. In a simple way, YOLO neural network analyses the image at one time only and pipelines the workflow. Based on the YOLO's single pass network structure, it can be optimised end-to-end to achieve higher performance.

The main theory of YOLO is to predict a (7, 7, 30) tensor by building a Convolutional Neural Network (CNN). The spatial dimension is then reduced to 7 X 7 with 1024 output channels at each location with the help of CNN. There are two fully connected layers in YOLO for performing linear regression. As a result 7 X 7 X 2 sized boundary box predictions are generated. The final prediction/result is made based on the scoring of the box confidence. The ones with greater than 0.25 confidence scores will be kept and used as final predictions.

Figure 5 illustrates YOLO's workflow:

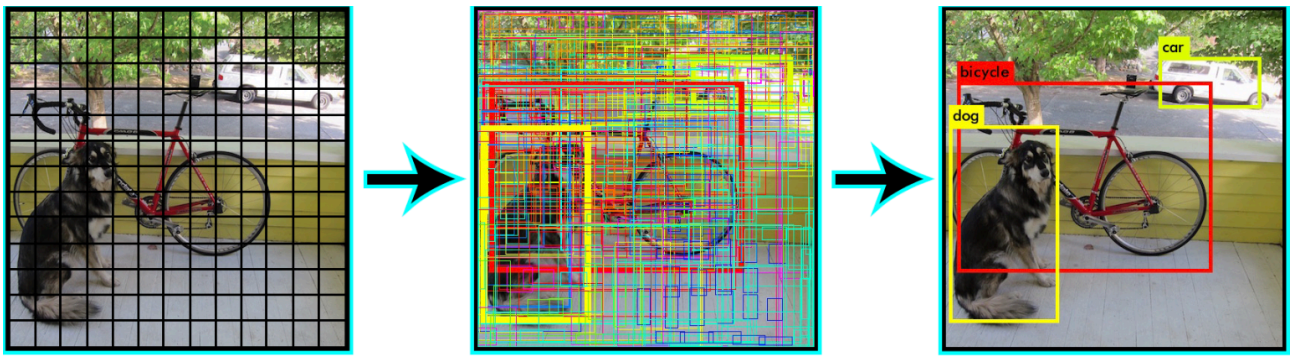

Figure 5: YOLO's Workflow
Source: [60]

Features that make Tiny YOLO stand out from other machine learning options are straightforward: lightweight structure (three times less layers than full YOLO v2) allows models running smoothly on mobile devices with limited computing power and storage; extra support from the A12 Bionic chip which includes Apple's next-gen Neural Engine when deployed to the latest iPhone models (XS and XS Max) Tiny YOLO.

Integrating Tiny YOLO to the project's mobile application went through four stages:

Stage 1: Taking pictures of the target (smart car) from different angles, lighting conditions and backgrounds. 300 pictures were taken of the car together with 100 pictures in randomly switched background and 100 pictures that are not relevant to the project. This setting tries to balance the outcome (model) between being over naive or over negative by feeding the training data set with the same object in complex background and irrelevant 'noise'. Test data set consists of 150 smart car pictures taken from completely different environment and location.

Stage 2: The training data set was annotated manually by using an open source tool 'Simple image annotator' from Github [46]. The smart car was labelled as 'car' in each training data picture as shown below:

Figure 6: Labeled Smart Car From Training Data Set

Stage 3: The labelled training data was then trained using python and Apple's custom machine learning model-training toolkit 'Turi Create' by following the instructions on its developer page [47]. The weight for this model was left using default settings due to the hardware limitation and time limitation for this project.

Stage 4: The trained model was then plugged into the mobile application by using CoreML as a bridge. The application was then able to give predictions from each camera frame and output bounding box location data.

The result of the test application has accurate object detection but its performance on test device, iPhone 6s and iPhone XS was at average 17 FPS (iPhone 6s) and 28 FPS (iPhone XS). For a game application it affects the player experience when the FPS is lower than 30 FPS. Apart from the low performance on real test devices, the solution has no potential of being used as a generic solution to ARKit object interaction. The nature of model training makes models for recognising/tracking specific object unique while popular universal models such as Place205-GoogLeNet, ResNet50 detects wide ranges of objects but there is no guarantee these models can detect customised objects like the the smart car. There is also no way to determine what class label the model will give to certain customised object once it is detected. As a summery, developers have to train different models for each specific task which is extremely time consuming and resource demanding. This solution was then eliminated from the candidate list due to its low performance, high development cost and lack of potential for future universal usage.

Pros of the attempt:

I. Relatively high accuracy in customised object detection/tracking

II. Able to detect and track customised object with noise from background (lighting, other objects and view angles)
III. Significantly reduced computing power requirements compared to traditional full YOLO neural network. Mobile devices are expected to run it smoothly

Cons of the attempt:

I. Tiny YOLO is designed for mobile devices and promises a swift performance but the performance for this particular project wasn't satisfactory. Further development and adjustment for the model will be necessary but it is out of this project's budget in both time and cost
II. The trained model isn't universal for other projects while project AR Kart aims at creating a universally adaptable solution

5. Design

Project AR Kart's goal as mentioned in the introduction is to create a way for virtual objects from augmented reality world to interact with real object from the real world. Breaking it down to the scenario of using ARKit as SDK, this project's priority goal is to fill the gap of ARKit's incomplete object tracking feature. Prior to ARKit 2.0 object tracking was a blank field for ARKit. Apple launched 2.0 version of the SDK but it only covered 2D object, precisely speaking, image tracking. 3D object tracking function is currently left blank with a slightly relevant function which is '3D object detection' presented by Apple in the 2.0 version of ARKit [7]. As it has been discussed in previous chapters, a developer can theoretically force the application to run 3D detection and anchor relocating at every camera frame in order to simulate object tracking visually but the limitation of memory usage and heavy computing power will cause the application to crash. Indeed, iOS memory managing mechanism will detect the extra memory usage and eventually force the application to stop. Clearly it is impossible to reach a universal solution for 3D object tracking purely relying on ARKit's native functions or other SDKs, tools from Apple (CoreML, Turi Create etc. as discussed).

As it is mentioned in the project's title and also in many parts of the report, this project is aiming at creating solution for augmented reality applications/games. The development of the project focuses on the iOS platform with ARKit and native Swift as programming language. This choice was made to fit the developer's own skills and experience, as he is specialised in iOS development with strong knowledge of swift, as well as Objective-C and C++. Using a familiar platform is cost efficient and provides better understanding and analysis, which allows going more in depth in the problem.

In this case the project design can be split into two sections: The first section contains the design of an iOS application, which functions as the game itself and also tracks the smart car. The second section contains the hardware design of the smart car and tracking module attached to it.

5.1. Software Design

ARKit's advantages and disadvantages have been discussed through the report in multiple chapters. To summarise, below are the pros and cons of ARKit without any other external SDK assistance.

Advantages:

- The world tracking performance is satisfactory in terms of plane detecting speed, speed of re-detecting plane after losing tracking and anchors tracking precision. In short, ARKit does a good job at detecting, re-detecting plane and keeping every AR anchor in its exact position when the user moves the camera around.
- Precise object motion tracking. Thanks to the VIO (Visual Inertial Odometry) technology used in iPhone 6s (and later) with M9 and above core motion coprocessor, ARKit has access to processed camera and motion data from VIO, which helps the application developed using ARKit understand how the device is moving in space relative to landmarks it sees [48].

- Assistant features such as feature points, world origin, add desirable functions for developers to get a better grasp of the application's current working condition. Feature points are the raw feature point cloud and they are used to track objects in the real world. The feature points are positioned on detected solid surface of both objects and planes. With the assistance of feature points the developer has a clear view of the 3D environment and how well ARKit is working in it. When world origin debug option is enabled, the developer is able to determine whether objects are placed in expected location relevant to the ground.
- Similar model rendering quality compared to certain paid framework like Wikitude.

Disadvantages:

- Strict lighting and surface condition requirements. Ideally ARKit enabled applications work well in a room with bright and evenly distributed light source in comparison to bright but centralised light source. Any types of light reflection can interrupt the AR experience. The surface suitable for ARKit to work on should be flattened and filled with certain features such as marble or natural wood.
- ARKit alone lacks ability to track moving 3D objects as it has been analysed previously. Image tracking is the feature closest to 3D objects tracking. Even though its performance relies heavily on the environment condition and object's motion changes.

For this project to succeed, it became evident that the priority was to find a suitable object-tracking framework that would be efficient but also co-op with ArKit in the same mobile application in a smooth manner.

Luckily thanks to the inspiration from the related work 'Ball tracking with OpenCV' the path of development was set; the first step towards a successful solution therefore required seeking and testing several available object-tracking frameworks. Marker and non-marker tracking frameworks are both included in the candidate pool. Each candidate framework was tested by building up a sample application from its own website with official guidance and code in order to reflect the base performance of the framework. To be straightforward a table comparison is shown below with the test results.

Table 2 illustrates the candidates and relevant elements in decision-making:

Name	Tracking Quality	Cost	Marker	Open Source	iOS support
ArUco 3	S	Free	Single	Y	N
OpenCV ArUco	A	Free	Single	Y	N
EasyAR	S	Free	Multiple	N	Y
Vuforia	S	$499	Multiple	N	Y
ARToolKit 5	A	Free	Multiple	Y	Y
ARToolKit 6	S	Free	Multiple	Y	Y

Table 2: Framework Candidates

* 'S' refers to 'Satisfactory performance'. 'A' refers to 'Average performance'

Initial selection of the SDKs was based on the availability and cost. The project intends to be built with the lowest cost possible with both software and hardware components available to the public. In this case only SDKs that are popular and public released are selected and tested. Apart from the cost factor, tracking quality is the most import factor in the decision making process. It is expected to have at least acceptable (average 25 FPS) smooth tracking performance without frequent glitches. Marker tracking ability is taken into consideration because the variety of makers accepted has potential of boosting the tracking performance and/or potential to create innovative tracking methods. Open source affects the selection because open sourced SDK allows the developer to customise its code, which is an essential feature when integrated with ARKit. The last factor 'iOS support' indicates whether the SDK provides direct sockets, interfacing methods to be integrated to iOS applications.

Vuforia had excellent performance during the test with the build of its official sample application for iOS [49]. Unfortunately due to the strict budget, the cost to afford a Vuforia developer account makes it unsuitable for this project.
ARToolKit 5 [2] had tracking performance at 20 FPS or lower in some scenes and frequent glitches when the tracked object started moving. The framework was not considered due to the performance issue.
The rest of frameworks, ARToolKit 6, EasyAR [50], ArUco 3 and OpenCV ArUco met both performance and cost requirements. At early stage of the development, a problem arose: both ARKit and candidate frameworks require exclusive camera access. Fortunately the access to raw image data of the last frame is available to multiple frameworks at the same time, which makes it possible to attach another framework to ARKit and work together with it. In this case the frameworks included in this project's application has to follow a frame-by-frame architecture. Easy AR and ARToolKit 6 were crossed out from the list due to the fact that neither of them supports frame-by-frame input. The option left is ArUco. ArUco's tracking performance isn't the best among all candidates but its features suit the project needs the best. ArUco supports frame-by-frame input and it is an open sourced framework, which means it is possible for the developer to implement and customise its functionalities according to the project. The issue with ArUco lies in its software architecture. It is available in X64 architecture without any mobile end arm 64 support. The language used to build ArUco is C++ which makes it unusable directly in Swift built iOS application. Because of that, the framework's build configuration files were modified. As a result, an iOS compatible ArUco framework is ready for the project although the implementation of the new framework requires using the mixture of Objective-C and C++, which was named 'Objective C++'.

At the time of building the first ArUco plus ARKit solution, ArUco 3 was also discovered and tested with satisfactory results. Compared with the original OpenCV ArUco, ArUco 3 offers a visually better performance, with an average of 30 FPS on iPhone 6s test device. With the promising tracking performance and standalone clear structure (compared to the old version's OpenCV plugin structure) ArUco 3 was selected as the co-op framework to assist ARKit.

The final product is expected to be user friendly and offers the ability to be further developed as an adaptable solution to the field of AR and real world object interaction. The user/player won't need specific knowledge in using AR applications to enjoy the

interactive AR experience. To realise the expectation, the app is designed to have initial screen instructions with point-and-play feature which the user only needs to follow and point the device camera at desired tracking target to start the game/interaction.

5.2. Hardware Design

The hardware part of the project consists of an Arduino Uno powered smart remote control car and a printed 3D ArUco cube.

Although the project has the expectation of being a universal solution and should be adaptable to work with any common models of remote controlled toy car, the reason for using and building a smart robot car is mainly for future development of the project. With the Arduino Uno development board installed on the car, it is possible to expand the car's functionality at any time. For example, the car's speed and obstacles around can be detected by accelerometer and ultra-sound sensor that can be soldered to the Arduino board. With the assistance of a bluetooth module the smart car will be able to communicate with mobile application. This feature adds various development ideas and potentials to the project.

The smart car's main body is a plastic chassis with two layers structure that host motors, motor control board (Model: L298N). On the top layer, Arduino Uno board and an Arduino extension board were mounted. The battery pack contains two rechargeable batteries and is located at the rare part of the top layer. Additionally at the front of the top layer board there is a slot for ultra-sound sensor installation but it is left empty due to the fact that at the current stage of development of the project it is pointless.

Figure 7 presents the lower layer board of the smart car with motors and motor control board, L298N.

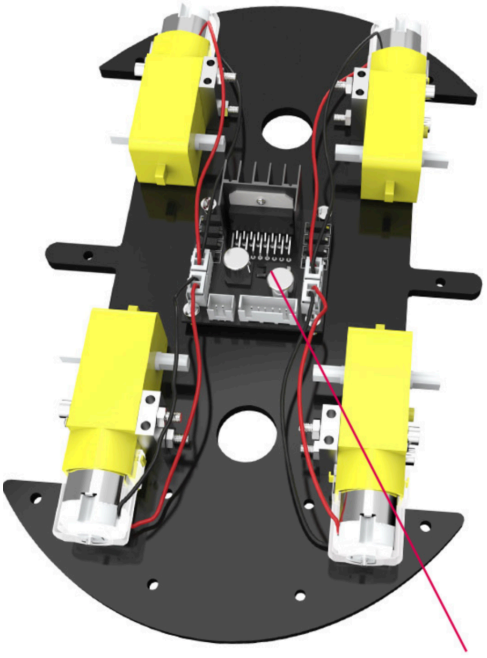

Figure 7: Smart Car Motor Base

As one can see, each wheel (not seen here but located in the 4 slots on the side of the layer) is equipped with a motor, which is then connected to the motor board.

Figure 8 shows the upper layer with Arduino Uno board, Extension board and battery pack.

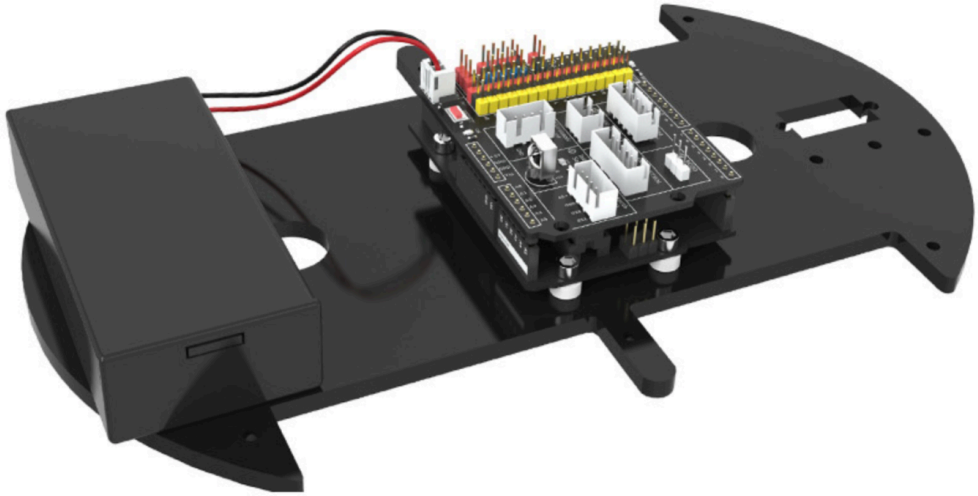

Figure 8: Smart Car With Arduino Extension Board And Power Pack

The upper layer is the 'brain' of the smart car; it feeds the motors with energy and instructions. The Arduino board is safely mounted on the upper layer of the car and is therefore visible.

Figure 9 shows the fully assembled smart car with the ArUco tracking cube mounted on the top of it.

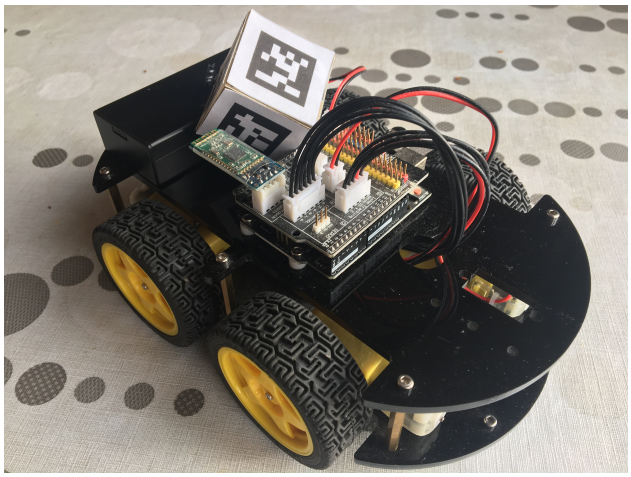

Figure 9: Fully Assembled Smart Car With ArUco Tracking Cube

As said previously, here one can see the Arduino board clearly visible on the upper layer, while the motors and motor board are hidden in the middle. The ArUco cube is placed between the battery pack and the Arduino board.

The smart car is connected to an Android device via a Bluetooth low energy board (HC-08). Player can easily control the car with one hand by holding the android device using gesture or virtual joystick. Although in the initial design the car was supposed to be

controlled by hand gesture with the support of accelerometer, extra Arduino and Bluetooth parts attached to the player's hand, the attempt was given up for this prototype because of time and cost considerations. Considering the main focus of this project is not to build a motion controlled car, this feature was considered secondary and therefore the use of an android device as the remote was elected as a temporary solution, in order to remain focused on the real topic here, which is AR and real world interaction. The development of the motion control remains a future objective, but not as a part of this project.

A noticeable feature of the smart car is the tracking cube mounted on the top of it. ArUco is very good at tracking markers' motion and location but inevitably it loses tracking of the marker when any part of the marker moves out of the camera view or the tracking angle is at around 30 degrees to the ground (Data gathered from this project's tests). Furthermore, the virtual cover (a transparent 3D model) around the smart car is also supposed to follow the car without any affection of tracking angles, yet the first design of the smart car with ArUco marker uses a flattened printed marker, which causes the miss-placement of the smart car's virtual cover.

Figure 10 presents the first design with 2D marker.

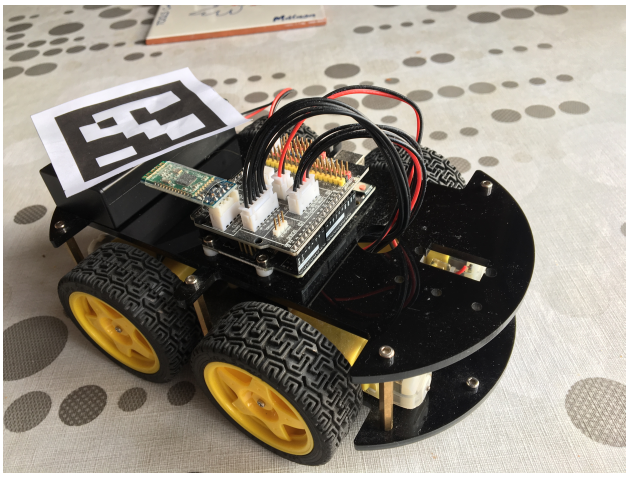

Figure 10: Smart Car With 2D Marker

Here the marker is a simple flat square fixed on the battery pack. Because of its 2D structure, the tracking was complicated, with a specific angle required for good performance and higher risk of tracking loss.

The problem with such design as discussed above is placing the 3D model (the transparent cover for the car in this case) in undesired positions while the car is moving. ArUco is likely to project the 3D model according to the marker position relative to the camera. In the game scenario if there is a target on the right of the car while the car's 3D cover is projected on the left, the player will find his smart car does nothing when it collides with the target on the right. By using a cube with 6 surfaces it is ensured at least one marker is detected at any camera angle. When more than one marker is detected, the application will perform calculation and output a middle point for anchor placing thus the smart car's 3D cover can always be accurately placed on top of it. The theory of ArUco tracking cube will be discussed in depth in the next chapter.

For the sake of lowering cost and fast prototyping, the tracking cube is currently made with cardboard and printed ArUco code markers that are glued to each surface of the cube.

6. Implementation

In Figure 11, the car can be seen evolving in the virtual environment. Once it reaches the location of the small boxes, the car “hits” them and the player gains point. Furthermore, the red bar at the top represents the player’s “life”. If the physical car touches a virtual wall, the life decreases. These are the effects this project sought to create, and this chapter will explain how it was done.

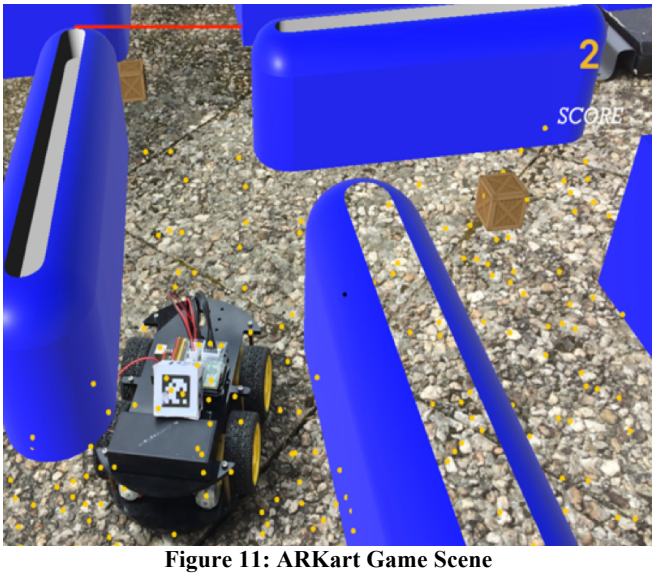

Figure 11: ARKart Game Scene

In this chapter the details of the project’s implementation will be discussed including the following topics:

- The implementation of ARKit
- The implementation of ArUco
- Integration of ARKit and ArUco
- Implementation of tracking box

6.1. ARKit

The project’s mobile app was created using Swift language with Apple’s Xcode as IDE. Starting point of the application was the boilerplate ‘single view app’ template provided by Xcode. The advantage of using empty template is the freedom of designing the UI with AR scene view in the background. When building the app there were two choices: using Metal with ARKit or SceneKit with ARKit. Metal is a low-level development kit that gives the developer more control over the hardware while the price is the higher complexity of the finished product. SceneKit is at higher level, which means it provides more ready to used 3D rendering functions and simpler software structure. As a result SceneKit was selected for the sake of clear structure and taking into consideration the developer’s previous experience. As previously mentioned, Swift was used as development language. At the time of development the Swift version was 4.0. The app will require upgrade when the later Swift version is launched in order to support future iOS.

Figure 12 presents an overview of the ARKit framework.

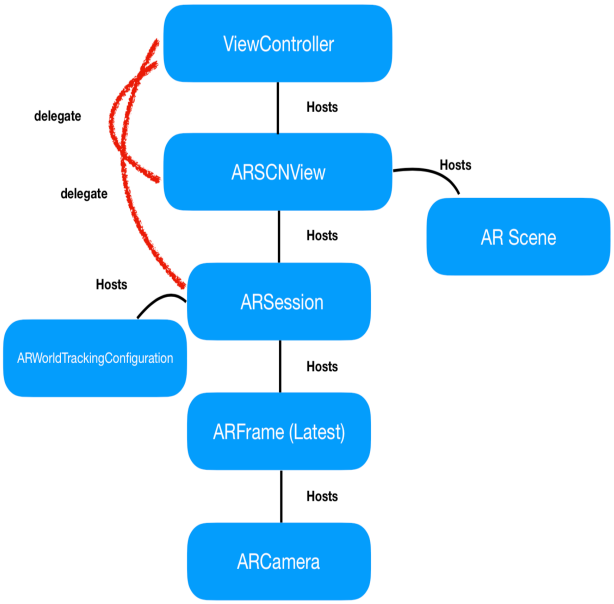

Figure 12: ARKit Framework Overview

ARSession is the most important class in term of accessing raw camera frames. Besides that, the ARSession class also has members of two tracking configurations, which are ARWorldTrackingConfiguration and ARImageTrackingConfiguration. In this project ARWorldTrackingConfiguration was chosen because of the need for plane detection.
AR Scene is a class for creating AR scene that usually contains the AR contents displayed on the screen. In this project the game scene has been pre-built and stored as an object of ARScene named 'gameSCN.scn'. In order to place the game scene exactly on the ground or any other user desired solid location the app runs plane detection first and presents the player with a 'tracker' to indicate a suitable plane has been detected and the game is ready to start as shown by Figure 13.

Figure 13: Tracker For Plane Detection

The screen clearly displays the Tap to Start button, which shows when solid plane has been found.

Thanks to the up-mentioned plane detection feature from ARKit the player can avoid putting virtual object in the middle of the air if the application performs plane detection prior to placing any nodes.

At the bottom of Figure 12 sits the key class ARFrame, which has access to each camera frame processed by ARKit while the SDK also allows other framework to access the raw

camera frame by using this class instance. Any AR objects requiring real-time frame refreshing will also need the access through this class.

ARKit's mission in this project can be described as following:

- Detecting plane for game scene placing
- Rendering the AR 3D game scene which contains all the game objects
- Detecting object collision

Majority of the ARKit's tasks are carried out in the app's ViewController. In Figure 14 presented bellow, it shows the initialisation of the game scene and ARKit. This snippet doesn't include all the initialisation. For the sake of easy maintenance the tracking configuration setting is placed on the top of the file while the plane detection setting is placed in ViewWillAppear section.

```
    override func viewDidLoad() {
        super.viewDidLoad()
//Creating a new game scene
        let scene = CarScene(named: "art.scnassets/gameSCN.scn")!
        scene.markerBox = MarkerBox()
        self.arSceneView.pointOfView?.addChildNode(scene.markerBox)

        self.pluginManager = PluginManager(scene: scene)
        self.pluginManager.delegate = self
        self.arSceneView.session.delegate = self.pluginManager.arManager
        self.arSceneView.delegate = self

//Some ARKit scene settings
        arSceneView.debugOptions = [.showFeaturePoints]
        arSceneView.autoenablesDefaultLighting = true
        arSceneView.pointOfView?.name = "iDevice Camera"
//Set the scene to the view
        arSceneView.scene = scene

        runTimer()
    }
```

Figure 14: Initialization of the Game Scene

The key to the project is also included in this part of the code. The interaction between the smart car and the virtual objects is realised by covering the smart car with an appropriate size of AR node, which can also be understood as a 'mask' in this case. Similar to the idea of using neural network for object segmentation, the interaction actually happens between the 'mask' of the real object and virtual object. The difference is that neural network segmentation relies on using trained model to identify the position of desired tracking object in real-time thus covering the tracked object with a virtual node while this project's solution is using ArUco to perform the identification and tracking process and ARKit to place the virtual node.
The code here shows that the markerBox object is created and added to the scene. The markerBox is a class created separately to calculate the position of the tracking box on the top of the smart car, which also ensures that the ArUco markers can always be tracked by the device thus providing updated location data of the cube for ARKit to place the AR node/Mask over the smart car. Combined with the every last frame refreshing function, the player will have the impression that a real world smart car is hitting virtual objects.

In order to keep the future development and maintenance ability, ARKit's implementation has been separated into several classes including the ViewController class introduced above.

1. ViewController.swift: Controller of the game storyboard and initialisation, settings of the ARKit. The controller also combines every other parts of the project together.
2. CarScene.swift: The class creates the node/mask for the smart car. In this class the developer can customise the node/mask if there is any creative ideas. For example if the developer wishes to put a 3D cartoon model on top of the smart car in the game, only step to do is changing the geometry to any customised model. In common cases the node is left transparent while in debug mode the developer can assign a colour to it in order to monitor the tracking performance.
3. ARManager.swift: This is a conjunction of ARKit, ArUco and OpenCV. Every last frame processed by ARKit is also fed to OpenCVWrapper class thus enabling the marker to be tracked and update its location after each frame has been processed. The SceneKit session function below shows the mechanism:

```
func session(_ session: ARSession, didUpdate frame: ARFrame) {
    self.opencvWrapper.findMarker(frame.capturedImage, withCameraIntrinsics: frame.camera.intrinsics, cameraSize:
        frame.camera.imageResolution)
}
```

Figure 15: Cube Tracking

The ‘didUpdate’ parameter defines the rate of the frame refreshing which is ‘every last frame’ while the processed ARFrame is then fed to OpenCV wrapper in order to update the marker position.

ARManager is also a delegate that supports the location update of the smart car’s covering node in ViewController where most of the functions are executed.

4. Plugin and PluginManager classes: These two classes are responsible for the smart car’s future game feature development. Plugin is a protocol that every instance of the CarScene has to follow, while PluginManager allows the developer to expand the AR features of the game. For example, the covering node (in this case the CarScene) can have the feature of ‘dragging’ virtual objects around. In the game the car will have new feature such as ‘attaching a fire circle around it’ like in the Mario Kart version. Developers will only have to create relevant ARKit functions and register it in the PluginManager.

This application is designed to follow modular structure. It tries to assign one functionality to one class. As a result the project is expected to have easy to upgrade features, which allows the game to have more exciting features and ways to interact with.

6.2. SceneKit

Apple defines SceneKit as a framework that combines a high-performance rendering engine with a descriptive API for import, manipulation, and rendering of 3D assets [51]. The engine itself also provides physics simulation which makes it suitable to work with ARKit in this project.

The main goal of the game AR Kart is to ‘collect’ as many cubes as possible in limited time. To build this game, the smart car has to be able to collide and ‘collect’ the cube (referred to as the ‘target node’ below) objects in the game scene. As it has been

discussed in Section 6.1, the actual interaction happens between the transparent node covering the smart car (referred to as the 'car node' below) and the virtual object. In this case the goal of development can be transferred to creating object collision with SceneKit. In addition to the collision, it is a good idea to have certain effect when the collision happens rather than simply remove the target node from the scene.

The collision feature is realised by setting up the target nodes' physics body to dynamic and change the value of bit masks as shown in Figure 16 below:

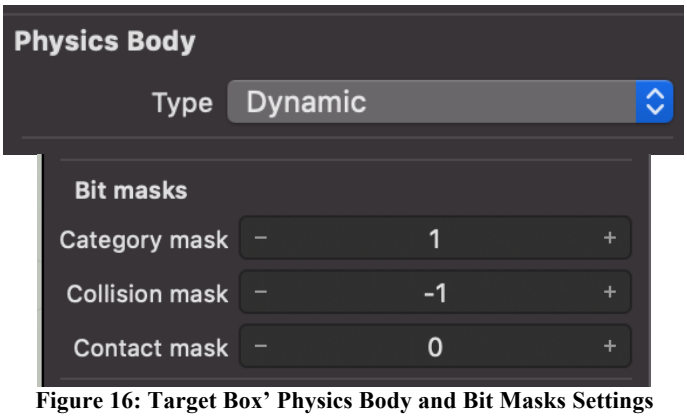

Figure 16: Target Box' Physics Body and Bit Masks Settings

SceneKit supports different physics bodies such as the Dynamic (as shown above) and Static as an option. Dynamic physics body means the object can be moved by the physics simulation [52], which is suitable for the target node in the game scenario (the target should be able to move in a small range in order to simulate the effect of 'being hit').

The collision is detected by objects with different contact masks, thus the car node's contact mask was set to 1 as shown by Figure 16 above. Below, Figure 17 shows the implementation of collision and its related particle system effect.

```
//PhysicsWorld functions for collision control
    func physicsWorld(_ world: SCNPhysicsWorld, didBegin contact: SCNPhysicsContact) {
        let targetNode = contact.nodeA.physicsBody?.contactTestBitMask == 0 ? contact.nodeA : contact.nodeB
        let explosion = SCNParticleSystem(named: "Explosion.scnp", inDirectory: nil)!
        let explosionNode = SCNNode()

        explosionNode.position = targetNode.presentation.position
        arSceneView.scene.rootNode.addChildNode(explosionNode)
        explosionNode.addParticleSystem(explosion)
        targetNode.removeFromParentNode()

        //Scoring system in test
        if (contact.nodeA.name == "block" || contact.nodeB.name! == "block") {
            score += 1
            self.scoreLbl.text = String(score)
        }

        let wall = self.arSceneView.scene.rootNode.childNode(withName: "pacmaze", recursively: true)
        if(contact.nodeA.name! == "pacmaze" || contact.nodeB.name! == "pacmaze") {
           wall?.geometry?.materials.first?.diffuse.contents = UIColor.red
        }
    }
```

Figure 17: Collision and Scoring

The collision is monitored by 'physicsWorld', one of the SceneKit's physics delegate function at real-time when any two objects collide. Code block below shows the virtual model that covers the car.

```
37      func setupCarPosPoint() {
38          self.carPosPoint.geometry = SCNSphere(radius: 0.2)
39          self.carPosPoint.name = "CarPos"
40          self.carPosPoint.geometry?.materials.first?.diffuse.contents = UIColor.green .withAlphaComponent(0.5)
41
42          //Car tracking cover node's physics settings
43          carPosPoint.physicsBody = SCNPhysicsBody(type: .dynamic, shape: nil)
44          carPosPoint.physicsBody?.categoryBitMask = 3
45          carPosPoint.physicsBody?.contactTestBitMask = 1
46
47
48          self.rootNode.addChildNode(self.carPosPoint)
49          self.rootNode.addChildNode(self.carTraceNodes)
50      }
```

Figure 18: Car Cover Configuration

The physicsWorld delegate function can be separated into three parts:
Part 1: Checking if any other node with different contact mask number is colliding with the ones with contact mask 0 regardless of which one is colliding with which one.

Part 2: Executing a particle system that has explosion effect when the nodes collide and then remove the target node. In this case the player will visually see the cube 'explodes' and disappears with the explosion when the smart car approaches it.

Part 3: Creating a node that refers to the walls in the game scene and then apply the same mechanism to it. The wall will be recognised as a 'node' and it flashes red when the car node collide with it. In this case it simulates the visual effect of crashing the car on the wall when the player accidentally drives the car into the wall.
The player will gain score each time when him/her 'collects' a cube.

Apart from hosting the object collision part, SceneKit also undertakes the task of hosting particle system effects. Once any two of the objects in the game scene collide the particle system will be triggered. As a part of the implementation of SceneKit the particle system effect was attached to the node named 'explosionNode' in the code snippet above and executed at the position where the target node was removed.

6.3. ArUco

The implementation of ArUco was straightforward. As we opted to use ArUco 3 in this project, there was no need to build the OpenCV plus ArUco plugin library bundle. First of all, OpenCV framework and standalone ArUco framework were added to the project folder and directly followed up by creating OpenCV wrapper class which is used to link Objective C and C++ written OpenCV library due to the fact Swift is incompatible with C++. Xcode will automatically generate bridging files to connect Objective C classes with Swift. After these steps, relevant functions from ArUco and OpenCV library can be directly used in other parts of the swift code. Finally ArUco is used in marker motion data translation and tracking functions that are implemented in MarkerBox and ARManager classes in order to provide the car node with the real-time tracking feature.

The starting point of ArUco's implementation is at OpenCVWrapper class. Strictly speaking it is an Objective C++ class.
The class that is the most relevant to this project is the 'aruco::MarkerDetector' class. It performs marker detection function. Parameters required by the class include camera parameters, which could be supplied partly from ARCamera's 'intrinsics' property. In the same wrapper class, the marker dictionary was set to 'ARUCO_MIP_36h12', which

defines the types of markers that are going to be recognised by the application. The markers share the same design pattern, which is built with 6 X 6 black or white squares and surrounded by a black line. The squares are not assembled in symmetrical way in order to avoid calculation errors [53].

Marker detection process can be described as below:
The detector (instance of the MarkerDetector class) is fed with an image by calling the function detector.detect(image, markers, camera, Parameters, markerSize). The next step involves feeding cameraParameters data (as shown below) and the markerSize data to the calculation.

```
aruco::CameraParameters camParams = aruco::CameraParameters(cameraMatrix, distCoeffs, cv::Size(cameraSize.width,
    cameraSize.height));
```

Figure 19: Camera Parameters

```
/// markerSize is the size of the printed marker in real world in meters.
float markerSize = 0.0258;
```

Figure 20: markerSize Parameter

Camera parameters define the camera's position relative to the marker. Marker size defines the physical size of the printed markers. These two parameters relate to the real world objects closely, which helps the ArUco to determine any changes of the marker's position and motions in relation to the camera.

Finally once the results of calculation are known, the detected marker IDs and their data of three-dimensional relative position to the camera are available to the developer.

An ambiguous but vital point is the source of 'image' parameter, which has been mentioned multiple times in the above discussion about marker detection and its calculation process. Image origins from the device's raw camera frame, is then processed by the ARKit and then output as a member of the ARFrame. Being one of the ARKit and ArUco conjunction points, the point where image can be pulled from ARKit to ArUco is from ARSessionDelegate's delegate function session(_: didUpdate :) which is located in ARManager class. The code below presents the exact implementation.

```
func session(_ session: ARSession, didUpdate frame: ARFrame) {
    self.opencvWrapper.findMarker(frame.capturedImage, withCameraIntrinsics: frame.camera.intrinsics, cameraSize:
        frame.camera.imageResolution)
}
```

Figure 21: Feeding ArUco With Image Updates From ARKit

The function is called after every new camera frame has been captured, thus updating itself almost at real-time speed. The captured frame image was at CVPixelBufferRef format which needs to be converted to ArUco's input format, cv::Mat.

There was another problem standing in the way after transferring the image format: ARKit encodes each pixel as an unsigned 8-bit integer with one channel while such encoding isn't compatible with ArUco. To solve this issue a helper function was created to specify the exact pixel encoding with an argument CV_8UC1 which code implementation is available below:

```
+(cv::Mat)convertPixelBufferToOpenCV:(CVPixelBufferRef)pixelBuffer {
    CVPixelBufferLockBaseAddress(pixelBuffer, 0);
    void *baseaddress = CVPixelBufferGetBaseAddressOfPlane(pixelBuffer, 0);

    CGFloat width = CVPixelBufferGetWidth(pixelBuffer);
    CGFloat height = CVPixelBufferGetHeight(pixelBuffer);

    cv::Mat mat(height, width, CV_8UC1, baseaddress, 0);
    CVPixelBufferUnlockBaseAddress(pixelBuffer, 0);
    return mat;
}
```

Figure 22: Pixel Interpreter

This function can be named 'Pixel interpreter', which describes its functionality. Noticeable point is that the cv::Mat initializer doesn't keep any copies of the image: instead it uses the same reference to the image. The advantage of such structure is speeding up the process.

With the effort above, the app is able to get latest update of image frame from ARKit and feed it to the OpenCV ArUco, thus the continuous detection of markers becomes a ready to use feature and serves the needs of smart car tracking.

Marker detection, format transferring and pixel interpreting can be computing power consuming tasks and have the chance of causing glitches. In order to create a smooth user experience it was decided to use multi-threading to increase the app's performance. To achieve this goal the marker detection process is set to run in a different thread. At the start of the OpenCVWrapper the queue was initialised as NSOperationQueue, which makes threading easier. Apart from separating the queues there is only one detection process allowed to run at once thus ensuring that the app won't be forced shutting down with over limitation resource usage.

6.4. Conjunction of ARKit and ArUco

In order to make it possible for ARKit and ArUco to work in a manner of fusion, the wrapper class OpenCVWrapper was built to bridge the C++ written library and the Objective C class which Swift can work with directly. The wrapper class requires initialisation before using. The class ensures any class can receive result from calculation in a defined manner by setting up OpenCVWrapperDelegate protocol, which is mandatory to conform to. Code block below shows the definition of the wrapper class.

```
@protocol OpenCVWrapperDelegate
-(void)markerTranslation:(NSArray<NSValue*>*)translation rotation:(NSArray<NSValue*>*)rotation ids:(NSArray<NSNumber*>*)ids;
-(void)noMarkerFound;
@end

@interface OpenCVWrapper : NSObject

@property id<OpenCVWrapperDelegate> delegate;
-(void)findMarker:(CVPixelBufferRef)pixelBuffer withCameraIntrinsics:(matrix_float3x3)intrinsics  cameraSize:
    (CGSize)cameraSize;
@end
```

Figure 23: OpenCV Wrapper Class Definition and Protocol Part

ARKit and ArUco's relationship can be metaphorically described as the constructor and the engineer. ARKit has the ability of rendering scenes, objects, while it needs to know where and how to place these objects. ArUco provides analysed information, in this case the processed location vectors of markers. It supplies ARKit with updated data of 'where' to place the objects. Working together, the two frameworks create an interactive AR experience.
Such fusion/conjunction point exists in ARManager where ARKit updates its ARFrame and feeds it to ArUco in order to update the marker location at the rate of each camera frame. However the marker position vectors supplied by the OpenCVWrapper were relative to the device's camera, which moves along in the room. In this case it is necessary to create a connection between the positions of the markers and the camera position. Adding camera to an SCNNode, which can adopt other child nodes, solved the solution to this issue. With this design, the position, rotation and scale can be applied to each child node. As it has been mentioned, ARKit is specialised in rendering work, the car node (set to green colour for debugging purpose) can then be attached to the camera node as a child node. In this case the smart car's cover (car node) is placed in a correct position.

Milestone

The tracking of the ArUco cube has been accomplished in this chapter. In order to create a tracked smart car that can visually interact with other AR objects, the cube was simply attached on the top of the smart car and the size of the car node was adjusted to cover the smart car. Finally set the colour of the car node to clear, thus creating a smart car that has the ability to interact with virtual objects.

6.5. Tracking Box

AR Kart is an augmented reality game with the theme of car racing. Based on this theme it is necessary to consider the player's movement around the smart car. The player needs to discover the positions of the 'cubes' in the game scene while navigate the car to approach the targets. Different to many AR applications for commercial products display or museum exhibition, AR Kart requires tracking from 360 degree around the smart car due to the nature of the game.

As mentioned in Chapter 5.2, a 2D printed ArUco marker was initially used to test the tracking performance. The tracking got lost frequently when the player started tracking the car with the 2D marker on top from different angles. Although there is no official document specifying what are the limitations in terms of tracking angles and camera moving speed, the test showed that the best tracking performance was reached when the player pointed the camera at approximately 90 degrees to the marker. The tracking would keep working until the camera exceeds 30 degrees towards the marker. Taking these elements into consideration it is important to keep the player's camera position in the best performance tracking angles. Having to remain exactly above the car or at a 30° angle would dramatically lower the quality of the experience, therefore a solution had to be found.
After inspecting some of the 3D models it was decided using a cube as the model for robust marker tracking. The reasons are:

1. Cube has six surfaces, which are all same size squares. ArUco marker is composed of 6 X 6 black and white squares that match the shape of the cube surface.
2. In different positions there is always at least one side fully displayed in the player's phone camera view. ArUco marker requires full marker shown when being tracked. Any corners missing from the camera view causes loss of tracking
3. It is relatively easy to print and make cube with ArUco markers on its surface. Common 2D printer is capable for the task. No other special material is needed. Paper or cardboard are both fit for the work.
4. The installing of the box is easy. Simply glue or put the box steadily on the top of the smart car

Figure 24 shows the finished prototype of the tracking box.

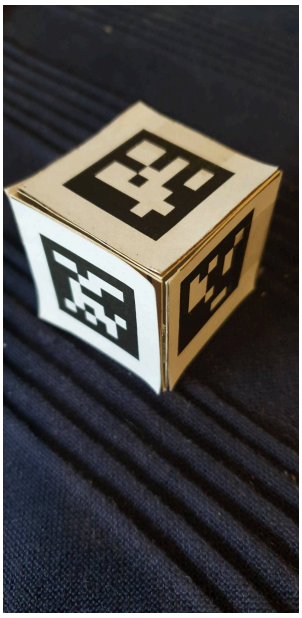

Figure 24: The Tracking Box

During the gameplay there will always be at least one marker detected by the app, which drastically lowers the risk of losing tracking randomly, while also providing a more comfortable user experience.
Once the makerBox class initialises, there is an array to save the detected markers in order to calculate the position of the box. As a result the car node of the smart car will have a low chance of loss tracking or miss placed somewhere else in the game scene.

Milestone:

Finding and creating the methods to track the markers and putting an AR object on the tracked marker is the first step towards building the game experience while building a tracking device with simple design and providing all-time tracking feature fit for the game's requirements.

6.6. The Smart Car

The smart car is an Arduino Uno-powered 4WD remote controlled car. As it has been introduced in the previous chapter, the design and actual build of the smart car all follow the purposes of future development, game performance and low cost.

Game performance is one of the reasons for choosing building an Arduino smart car over any other common RC toy car. At the beginning of the development, RC toy car was a candidate but its average speed made it unfit for the project. Majority of the RC cars in the market have selling points of 'high speed' and 'fast turning'. The average high travelling speed makes it impossible to keep the game in most of the indoor areas. Same issue makes the game require massive size of AR rendered game scene, which leads to significantly low performance on mobile devices with limited computing power. The last issue is the player experience. During the initial development stage it was found with the handheld device that the best size for the game plane is around 12 square meters. Bigger game plane will result in the player running and consuming too much energy thus losing the focus on the game itself. Common RC car can easily run out of a game plane around 12 square meters while on the contrary an Arduino smart car with lower speed keeps the game in a controllable space. The smart car also runs with a different turning mechanism compared to toy RC car, as it is using the speed difference in two sides' wheels. When the right side wheels are spinning at higher speed the car turns to the left and vice versa. Such turning mechanism isn't convenient in the scenario of RC car racing but suitable for AR Kart due to the fact that the mechanism adds on the difficulty for the player to 'collect' the cubes in the game. ARKit and its co-op ArUco also benefit from the lower car speed in term of providing better AR and tracking performance.

The smart car connects to its controller by using Bluetooth low energy (BLE) technology. HC-08 was chosen as the Bluetooth transmitter/receiver because it is low cost and easy to use. HC-8 has a good compatibility with both iOS and Android devices, which is also why it was chosen for the smart car. The future development of the project is likely to expand the functionalities and make it controllable by more devices.

At the moment of development, the smart car is controlled via Bluetooth by an android application as mentioned. This isn't an ideal solution but it leaves space for future development. Solutions such as accelerometer gesture controller or muscle flex sensor controller are likely to increase the immersive experience of the game and should be further investigated in the future. For the current design settings however, this android phone - smart car solution fits in the low cost and easy to use guidance, although it requires an extra device. While it is clear that this solution can only be used in the prototyping process, electing it was necessary to allocate both time and money resources to research the question of virtual-real world interaction; therefore it was a rational decision.

The double layer design of the smart car as shown in Figures 7 to 9 provides practical space for housing hardware boards such as the Arduino Uno board, the power pack and leaves spots for further hardware development.

Figure 25 presents an overview of the smart car's hardware structure.

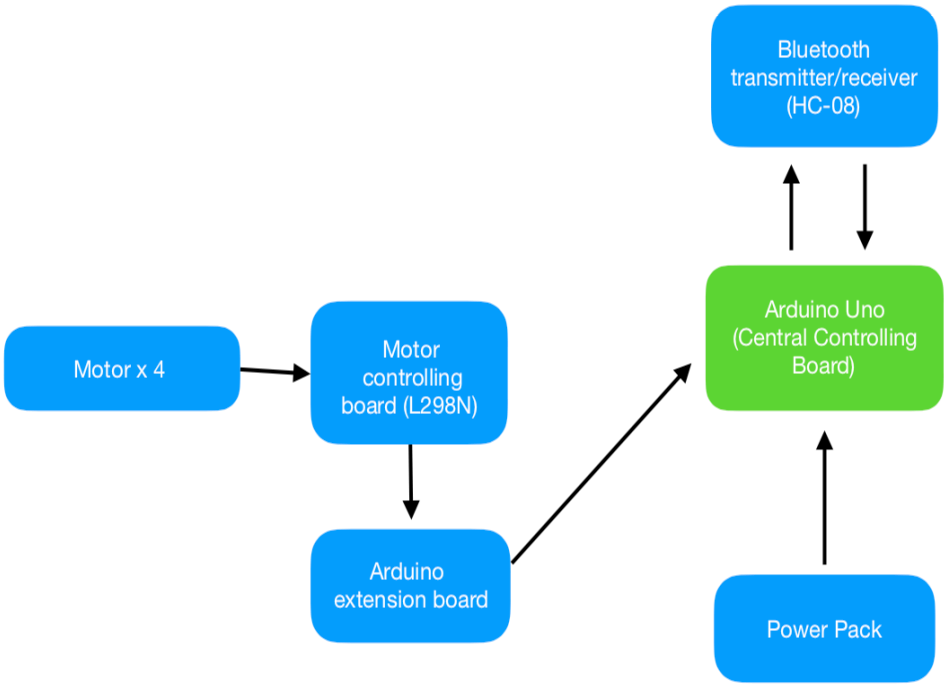

Figure 25: Smart Car Hardware Overview

As shown, the hardware structure of the smart car is very straightforward, with only the strict minimum amount of elements. This provides greater clarity in building the car, without extra features that aren't necessary, and also leaves place for customization, improvements and future adds-on.

The smart car was conceived to be easy to use and easy to reproduce, which is why simplicity was an important factor in the choice of elements and the fabrication process. The code was also built to be as straightforward as possible, in order to provide possibilities of further development and added functions. This project should therefore serve as a base for more work in the future.

7. Evaluation

As an AR powered mobile game, AR Kart has two pieces of required hardware which are smart car and an iPhone 6s or later. Considering the hardware cost the player can make his own setup with different smart car chassis, Arduino boards and other onboard hardware as long as the car is able to support Bluetooth low energy transmitter/receiver HC-08. The iPhone can be any country version starting from 6S to the latest XS Max. The game environment requires at least 10 square meters indoor or outdoor space with flat ground and good lighting condition (refer to ARKit's lighting condition requirement which has been discussed previously).

In order to evaluate actual player feedback two groups (Separated as Group A and Group B) of players were invited to test the project game. Each group consists of 4 players. Two groups of players come from different backgrounds. Group A players all have jobs in software development field while Group B players all have jobs in human resources. The reason for choosing this sampling is based on the need to gather feedback from both 'professionals (Group A, developers)' and 'amateurs (Group B, HRs)'. The average age of both groups is 32 years old. In this case the result of evaluation will have the lowest bias possible.

The two groups of players were given same models of iPhones (6s) in order to avoid performance difference among iPhone models. Some of the devices are owned by the players while others were borrowed. Players used the same model of the smart car and the same tracking box attached to the car. The test continued for two days due to the fact that some players couldn't make it to the test on the first day.

The procedure of the test is listed below:

- Inviting one player at a time to the test room (15 square-meters)
- Providing the player with short but complete instructions of how to setup the game
- Observe player's actions and record the time player uses to setup the game
- Observe player's reaction during the game and record the result (score of the player).
- Let the player finish the questionnaire

Figure 26 shows the short questionnaire for the game evaluation.

Evaluation AR Kart

Gender: ( ) Female ( ) Male

Age:

Knowledge of Augmented Reality

( ) None ( ) Some ( ) Plenty

How do you rate the game experience in general?

( ) Poor. It isn't a game

( ) Below average. Looks like a game but boring

( ) Average. AR game with some exciting features

( ) Interesting! Brand new experience but the game have space to improve

( ) Good. Will play more rounds

( ) Great. Will download it if it is available on App Store

What it is like to crash some virtual objects with a real toy car?

( ) Boring

( ) It isn't new to me

( ) Interesting

( ) Amazing! Can't believe my eyes

How easy it is to play the game?

( ) Hard. Have no idea how to get it running

( ) Average. It works after doing some ‘homework’

( ) Easy. Almost like playing with a toy RC car

How do you like the control of your car?

( ) Hard. It doesn’t follow my orders

( ) Average. No problem to run it but loses control sometime

( ) Good. Work as charm

How do you feel the screen size of the iPhone 6s for this game?

( ) Too small

( ) Perfect

( ) Too large

Any comments?

Thank you for your time!

Figure 26: Two Pages Questionnaire

By analysing the results of the eight questionnaires it can be summarised as following:

Five out of eight of the participants believe this game is average but has exciting features. Among the five participants three are from group A (developers) while the other two are from group B (HRs). Above average number of the players think this game's feature is exciting while one out of the eight players think it is cool and will play it for more rounds. This player is from the HR group who doesn't have much knowledge of the AR and mobile apps industry. An interesting point is this player is also a fan of Mario Kart. This discovery hints that players with professional knowledge of mobile apps and/or AR technologies have higher passion for interactive AR games like the project while player with similar game preference is likely to welcome such game as well. When AR games have better or new interactive methods they have higher chance of being popular.

Three out of eight players think crashing virtual objects with the smart car is amazing and really cool while the other five players agree on the interaction was interesting. This question reached 100% positive answers. It reveals that the player/user interaction method created in the project AR Kart is valued by the player. On the other hand it also shows that available AR games are likely to have less satisfactory/interesting player interaction methods. Based on the comments section, two of the players from group A mentioned such interaction method should be promoted and adopted to other AR games.

The players' feedback regarding the easy level of playing the game is average with seven players voted for average while one voted for hard. The main problem and available comments show the car is harder to setup since the prototype doesn't have user friendly instructions on the hardware itself and it also lacks instructions when the car loses tracking. Future development is expected to improve this prototype and put more focus on user experience.

The worst evaluation is about the smart car's control part. Six out of eight of the players believe the car does run according to their orders but it loses control sometime. Possible reason for this is at the start of the game the app requires a short while to detect and then start tracking the car while the UI doesn't have any instructions asking the player to wait until the tracking is established. In some rare chances the app loses tracking because the player moves the device in a high speed. This problem can be avoided by creating in-game warning system, which detects and warns the player of unsupported fast movements. A solution for enhancing the tracking box and app performance is necessary for the future development.

As for the screen size, five of the players think 4.7 inches (iPhone 6s model) is too small for an interactive AR game like the project. In the comments players suggest using 9.7 inches iPad Pro is the best choice for this kind of AR games.

8. Conclusion

The AR Kart project created a new experience for the player/user to interact with virtual objects in the AR game. Currently in the app markets (both App Store and Google Play), augmented reality isn't a new technology used in games and apps but the method for users/players to interact with the apps is limited to screen interaction meaning the player/user can't actually have any interaction directly with what they are looking at through the screen in real space. This is kind of pity for the AR technology nowadays and also a significant gap between the AR and VR experience. As one of human's senses and instinct, touching is a reaction common people will perform when they are interested in new objects around. Although the technology today can't simulate muscle sensation in AR technology at least not in a low cost solution, still it is a step forward in term of enabling collision between real object and virtual object.

8.1. Summary

AR Kart's approach relies on low cost available resources, which makes it possible for other developers to further explore the solution's potentials with a relatively low entrance bar. According to the works and history timelines, it is not hard to find both technology started in an early time (Augmented reality: First implemented in 1957 [54], Computer vision: First implemented in 1966 [55]) while both of the technologies later focused on military and/or scientific studies. Until recent years with the open sourced SDKs introduced in the report, ARKit and computer vision started being used by civilians. Unfortunately good quality and complete SDKs like ARCore and ARKit were launched in only two years ago, which shows that the interest in these technologies at a civilian level is still very new. This is likely to be the cause of why there hasn't been a trend of using both technologies as a fusion. With the strong points of the two technologies, robust object tracking from computer vision (OpenCV ArUco) and excellent AR object rendering from ARKit, AR Kart is expected to be the example of an innovative solution for the augmented reality field.

The meaning of the project is more important than the game play. In fact the game takes reference to Nintendo's Mario Kart because of its intense collision among the characters and objects in the scene, which can give the project a suitable stage to demonstrate the interaction between real object and virtual objects. Neural network was a candidate solution for the project and as a matter of fact a well trained model with vast amount of training data would have chance to achieve better performance. The problem with such solution is its extreme high cost in time and budget. Apart from these external problem as it has been discussed, the development of mobile friendly neural networks lack the ability of continuous studying or self-learning. IBM Watson is an exception [56] but with the high development cost it doesn't fit for non-commercial research project. The solution of using neural network will solve unique problems instead of being a potential universal solution.

Using project AR Kart as a starting point for possibilities of immersive user experience similar to the VR applications can be added to existing works such as AR games and/or apps, and hopefully inspire other developers to work on this topic. Indeed, it is a cheap solution developers can easily adapt the core technology of the project, the tracking box

into different scenarios. A good example will be AR products display. Sellers can use different tracking boxes to represent each product's 3D model. When such AR app is enhanced by tracking box, the potential consumers can actually put the object on them by using their own hand and view the products with better tracking performance. A shoe promotion AR app can be modified to be a watch promotion app in very short time without rebuilding its core structure, however if the app uses neural network to detect and track human foot then in this case the model has to be re-trained to recognise hands and wrists. With the enhancement of tracking box the part that needs to be modified is the 3D model.

The project as a game has reached its prototype level of completion. From the evaluation part of the report eight players from real life have given opinions on improving the game's control mechanism, UI notification system and the game play. There is a distance between the prototype stage of the project and a completed App Store shipment-ready game work. Enriching the game play by including well designed racing maps and new enemies, objects is a necessity but achievable mission based on the prototype.

8.2. Future Work

AR Kart has its potential to be a completed exciting game and also the starting point of other related projects. There have been suggestions to the game itself and ideas that are expected to be realised in future work.

For the game part, the current version of the AR Kart is a simple Mario Kart alike game with single target object and single game map in AR. To achieve a completion and exciting game, AR Kart will need to have its own game story or be modified to be a complete multiple player game. A set of customised 3D models are necessary for the game in which case it can help building an immersive game environment in the player's living room, for example. New AR game maps will be created to match the game story or theme. As an upgrade, the future work for game development is expected to put these aspects as 'blood' and 'flesh' on the top of the prototype (the 'skeleton').

During the development of the project until the end there has always been an idea for reusing the tracking part of the project on other related work or further develop it to be a generic solution to the topic of this project. The current version of the tracking box has its problem of being fragile because of its cardboard base. Redesign the tracking box to make it solid and enhance its reusable ability is one of the future work goals. By redesign the tracking box, it is expected to be 3D printed using solid material such as plastic. Therefore the user can print out his alternative ArUco markers and stick it on the surfaces in which case the tracking box can 'display' different AR models by having new stickers on the surfaces.

The project's idea can be further applied to some scenarios that requires privacy. It was decided at the end of the project development that a related personal project 'Private AR switch' will be developed. The idea of Private AR switch includes an IoT idea powered lock system and ARKit powered iOS app. Locking something with a traditional key, whether it's an office door or a safe, can cause many safety issues, for example because of the risk of duplicated key, lost keys or stolen keys. The Private AR switch aims at using a similar idea from AR Kart, which is to use a tracking box as the key. Virtual object can

only be displayed and tracked by the app. The user will need to have both matching markers with the correct unlocking sequence to collide with the AR pattern shown by the app to unlock a door or safe. In this case the markers attached to the tracking box can be generated at any time with different ArUco codes while the AR pattern can also be regenerated for safety reason.

As we can see, this project offers the possibility of many applications, not only in mobile gaming, but also in a multiplicity of other styles of apps. With the results obtained from this prototype, it is clear that this solution is robust and could be easily adapted to many different projects.

References

[1] Apple. (n/a, n/a) Developer Apple. [Online]. https://developer.apple.com/documentation/arkit/arworldtrackingconfiguration/2923548-planedetection

[2] ARToolKit. (n/a, n/a) Github. [Online]. https://github.com/artoolkit

[3] Universidad de Córdoba. (n/a, n/a) Aplicaciones de la Visión Artificial. [Online]. http://www.uco.es/investiga/grupos/ava/node/26

[4] Vuforia. (n/a, n/a) Vuforia. [Online]. https://www.vuforia.com/content/vuforia/en/augmented-reality-app-development-developer.html

[5] Wikitude. (n/a, n/a) Wikitude. [Online]. https://www.wikitude.com

[6] Wikitude. (n/a, n/a) Wikitude. [Online]. https://www.wikitude.com/wikitude-slam/?fbclid=IwAR2XDE6A9UGfCykvdQBpGnRoaM0a-Cm87tCDiKXOTuozBi66eon-csVtmuY

[8] Ronald T Azuma, "A Survey of Augmented Reality," *Presence*, vol. 6, no. 4, pp. 355-385, August 1997.

[7] Apple. (n/a, n/a) Developer Apple. [Online]. https://developer.apple.com/arkit/

[9] Gartner. (n/a, n/a) Gartner IT Glossary. [Online]. https://www.gartner.com/it-glossary/augmented-reality-ar/

[10] Donna R Berryman, "Augmented Reality: a review," *Medical Reference Services Quarterly*, vol. 31, no. 2, pp. 212-218, 2012.

[11] David Grover. (2014, February) Macquarie University. [Online]. https://wiki.mq.edu.au/display/ar/Augmented+reality+history%2C+background+and+philosophy

[12] Chris Curran. (2016, April) PWC. [Online]. http://usblogs.pwc.com/emerging-technology/the-road-ahead-for-augmented-reality/

[13] Louis B Rosenberg, "Virtual fixtures as tools to enhance operator performance in telepresence environments," in *SPIE*, vol. 2057, Boston, 1993, pp. 10-21.

[14] B Thomas et al., "ARQuake: an outdoor/indoor augmented reality first person application," in *ISWC2000 - 4th International Symposium on Wearable Computers*, Atlanta, 2000.

[15] Anders Henrysson, Mark Ollila, and Mark Billinghurst, "Mobile Phone Based Augmented Reality," in *Mobile Computing, Concepts, Methodologies, Tools and Applications*. Hershey, United States of America: Information Science Reference (an imprint of IGI Global), 2009, vol. 1, pp. 984-997.

[16] Bendik Stang, *The Book of Games*. Ottawa, Canada: GameXplore N.A. Inc, 2006, vol. 1.

[18] C Magerkurth, M Memisoglu, T Engelke, and N Streitz, "Towards the next generation of tabletop gaming experiences," in *Proc. Graphics Interface 2004*, London, Ontario, Canada, 2004, pp. 73-80.

[17] Marcus Torchia and Michael Shirer. (2018, December) IDC. [Online]. https://www.idc.com/getdoc.jsp?containerId=prUS44511118

[19] C Magerkurth, R Stenzel, N Streitz, and E Neuhold, "A Multimodal Interraction Framework for Pervasive Game Applications," in *Artificial Intelligence in Mobile System*, Seattle, US, 2003, pp. 1-8.

[20] HITLab. (n/a, n/a) ARToolKit. [Online]. http://www.hitl.washington.edu/artoolkit/

[21] Jon Peddie, *Augmented Reality: Where We Will All Live*. Cham, Switzerland: Springer

Nature, 2017.

[22] Malcom Owen. (2017, December) Apple Insider. [Online]. https://appleinsider.com/articles/17/12/30/apple-2017-year-in-review-the-realities-of-ar-and-vr-and-apples-arkit

[23] Cory Bohon. (2017, September) Tech Republic. [Online]. https://www.techrepublic.com/article/apples-arkit-everything-the-pros-need-to-know/

[24] Caitlin McGarry. (2018, January) Tom's Guide. [Online]. What Is Apple's ARKit? Everything You Need to Know

[25] Apple. (2017) Apple. [Online]. https://developer.apple.com/documentation/arkit

[26] Tech 2 News Staff. (2018, March) First Post - Tech 2. [Online]. https://www.firstpost.com/tech/news-analysis/apple-releases-ios-11-3-bringing-in-arkit-1-5-iphone-battery-health-new-animojis-and-other-bug-fixes-4411805.html

[28] Michelle Fitzsimmons. (2018, June) What is ARKit 2? Here's what you need to know about Apple's latest AR update. [Online]. https://www.techradar.com/news/what-is-arkit-2-heres-what-you-need-to-know-about-apples-latest-ar-update

[27] (2018, June) The Brothersbrick. [Online]. https://www.brothers-brick.com/2018/06/06/lego-showcases-augmented-reality-play-with-apples-arkit-2-at-wwdc-news/

[29] Andrew O'Hara. (2018, June) Apple Insider. [Online]. https://appleinsider.com/articles/18/06/11/hands-on-with-the-new-arkit-measure-app-in-ios-12

[30] Sajjad Taheri, Alexeandru Nicolau, Alexeander Vedienbaum, Ningxin Hu, and Mohammad Reza Haghighat. (2018, May) EE Times. [Online]. https://www.eetimes.com/document.asp?doc_id=1333336

[31] OpenCV. (n/a, n/a) OpenCV. [Online]. https://opencv.org/about.html

[32] NVIDIA. (n/a, n/a) NVIDIA. [Online]. https://developer.nvidia.com/opencv

[33] Muñoz Rafael Salinas, ArUco: An efficient library for detection of planar markers and camera pose estimation, April 16, 2018.

[34] Arduino. (n/a, n/a) Arduino. [Online]. https://www.arduino.cc/en/guide/introduction

[35] (n/a, n/a) Arduino. [Online]. https://store.arduino.cc/arduino-uno-rev3

[36] Samsung. (2018, March) Samsung. [Online]. https://www.samsung.com/in/support/mobile-devices/what-is-bluetooth-low-energy-ble-technology/

[38] OpenCV. (n/a, n/a) OpenCV. [Online]. https://opencv.org

[37] Cicicom. (n/a, n/a) Cicicom. [Online]. https://www.cicicom.gr/pages/ble/

[39] Adrian Rosebrock. (2015, September) PyImageSearch. [Online]. https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/

[40] Apple. (2017, October) Developer Apple Documentation Archive. [Online]. https://developer.apple.com/library/archive/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/DeviceCompatibilityMatrix/DeviceCompatibilityMatrix.html

[41] Asier Arranz. (2018, June) Medium. [Online]. https://arvrjourney.com/mario-kart-in-a-real-vehicle-with-vr-4536bddd52d9

[42] Apple. (n/a, n/a) Developer Apple. [Online]. https://developer.apple.com/documentation/arkit/recognizing_images_in_an_ar_experience

[43] Apple. (n/a, n/a) Developer Apple. [Online]. https://developer.apple.com/documentation/arkit/arworldtrackingconfiguration/2941063-detectionimages

[44] Apple. (n/a, n/a) Developer Apple. [Online]. https://developer.apple.com/documentation/coreml

[45] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. (2016, June) Cornell University. [Online]. https://arxiv.org/abs/1506.02640

[46] Sebastian G Perez. (2018) Github. [Online]. https://github.com/sgp715/simple_image_annotator

[48] Huai Zheng and Guoquan Huang. (2018, May) Cornell University. [Online]. https://arxiv.org/abs/1805.04031

[47] Apple. (2018) Github. [Online]. https://github.com/apple/turicreate/tree/master/userguide/object_detection

[49] Vuforia. (n/a, n/a) Vuforia Engine Developer Portal. [Online]. https://developer.vuforia.com/downloads/samples

[50] EasyAR. (2018) EasyAR. [Online]. https://www.easyar.com/view/download.html

[51] Apple. (n/a) Developer Applie. [Online]. https://developer.apple.com/documentation/scenekit

[52] Apple. (n/a, n/a) Developer Apple. [Online]. https://developer.apple.com/documentation/spritekit/skphysicsbody/1520132-dynamic?language=objc

[53] S Garrido-Jurado, R Muñoz-Salinas, F J Madrid-Cuevas, and R Medina-Carnicer, "Generation of fiducial marker dictionaries using Mixed Integer Linear Programming," *Pattern Recognition*, vol. 51, pp. 481-491, March 2016.

[54] (2018, September) Interaction Design Foundation. [Online]. https://www.interaction-design.org/literature/article/augmented-reality-the-past-the-present-and-the-future

[55] Seymour A Papert, The Summer Vision Project, July 7, 1966.

[56] IMB. (2019, February) IBM Watson Studio. [Online]. https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/ml-continuous-learning.html

[58] Wikitude. (2018, September) Youtube. [Online]. https://www.youtube.com/watch?v=Zlh8DkMRu6Y

[57] V. Mykola and B. Gleb. (2017, June) RubyGarage. [Online]. https://rubygarage.org/blog/best-tools-for-building-augmented-reality-mobile-apps

[59] Apple. (n/a, n/a) Developer Apple. [Online]. https://developer.apple.com/documentation/arkit/scanning_and_detecting_3d_objects

[60] Jonathan Hui. (2018, March) Medium. [Online]. https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

[61] Asier Arranz. (2018, June) Youtube. [Online]. https://www.youtube.com/watch?v=jbqJM-JmMak