Game of Drones

An Augmented Reality Multiplayer "Pong" game for smartphones using real drones

Iby and Aladar Fleischman Faculty of Engineering Tel Aviv University

הפקולטה להנדסה ע"ש איבי ואלדר פליישמן אוניברסיטת תל-אביב

Autonomous Drones Lab

Nir Idisis, M.EE student

Under supervision of Yonatan Mendel

https://github.com/tau-adl/GOD

Contents

Abstract 3
About the Game “Pong” 3
About the Unity IDE 4
About the Vuforia AR Engine 4
About Augmented Reality 5
How It Works 7
Hardware 7
Image Stabilization Mechanism 7
Object Detection and Tracking 7
Localization and Mapping 8
About the Drone 10
WIFI Mode 10
Technical Specifications 10
Game Design 11
Drone Detection and Tracking 11
Using Vuforia Advanced Model Target 360 11
Using Vuforia Image Target 13
Using Drone Height Measurements 15
Using Mission-Pad Detection 16
Final Decision 16
Playground Placement and Augmentation 17
Using an Anchor on the Ground 17
Using a User Chosen Anchor 17
Real-Time Multiplayer Communication 18
Player Discovery 19
GOD Messaging Channel 19
User Guide 21
Game Settings 21
Drone Settings 22
Single Player 23
Multi-Player 23
References 27

Abstract

The purpose of this project is to build a 3D AR (Augmented Reality) “Pong” game in which real drones serve as the paddles.

The game will be designed to run on recent smartphones, running the Android operating system, and featuring built-in AR support. It will be developed by using the **Unity**™ game engine, together with the **Vuforia**™ AR framework. This should allow the game to be compiled also for iOS devices without doing any platform-specific modifications.

In this book will discuss the challenges that had to be solved in order to achieve a working playable game with drones. We will be describing different solutions for each challenge and explain our choices to use one solution over the other.

About the Game “Pong”

The game “Pong” is a 2D tennis-table video game originally released by Atari in 1972.
The game consists of two players, each controlling a paddle moving vertically across the left or the right wall. The players need to protect the wall behind them from the moving ball by using their paddles to hit the ball when it gets near.

When the ball hits a player’s wall, the opponent gets a point.

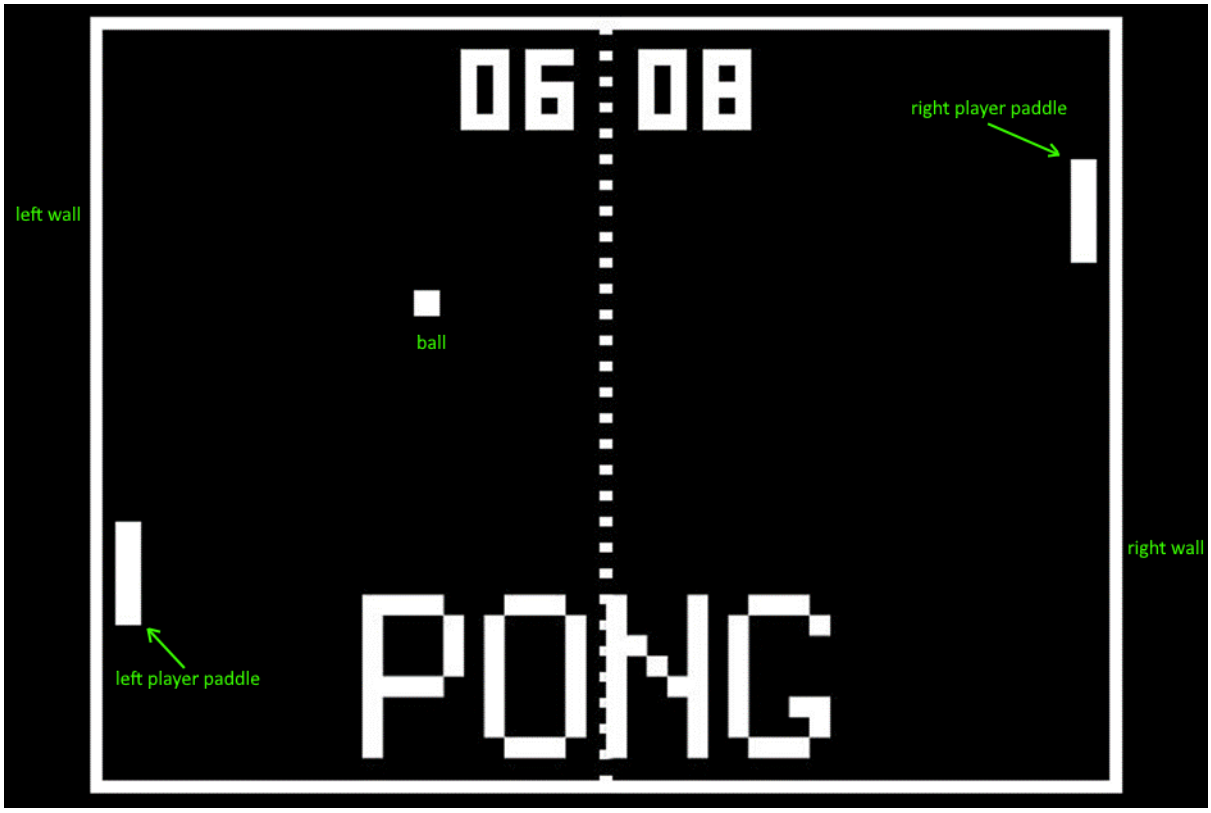

Figure 1 - The original Pong game

About the Unity IDE

Unity is a cross-platform game development environment. Unity games can be compiled for Android, iOS, WMP, without messing with platform-specific details. The IDE provides 3D graphics and physics engine that allows developers to build games using a rich graphical user interface and concentrating primarily on game-logic, leaving graphical transformations, geometry, and physics calculations to Unity.

The code in Unity is written in C# scripts, which are translated by the Unity engine to platform-native executables. The C# code is first compiled into ordinary IL assemblies, which are then converted automatically to C++ code for the chosen platform. Finally, the C++ code is compiled by using the platform-specific compiler and libraries. The whole process is performed by Unity's using its IL2CPP engine.

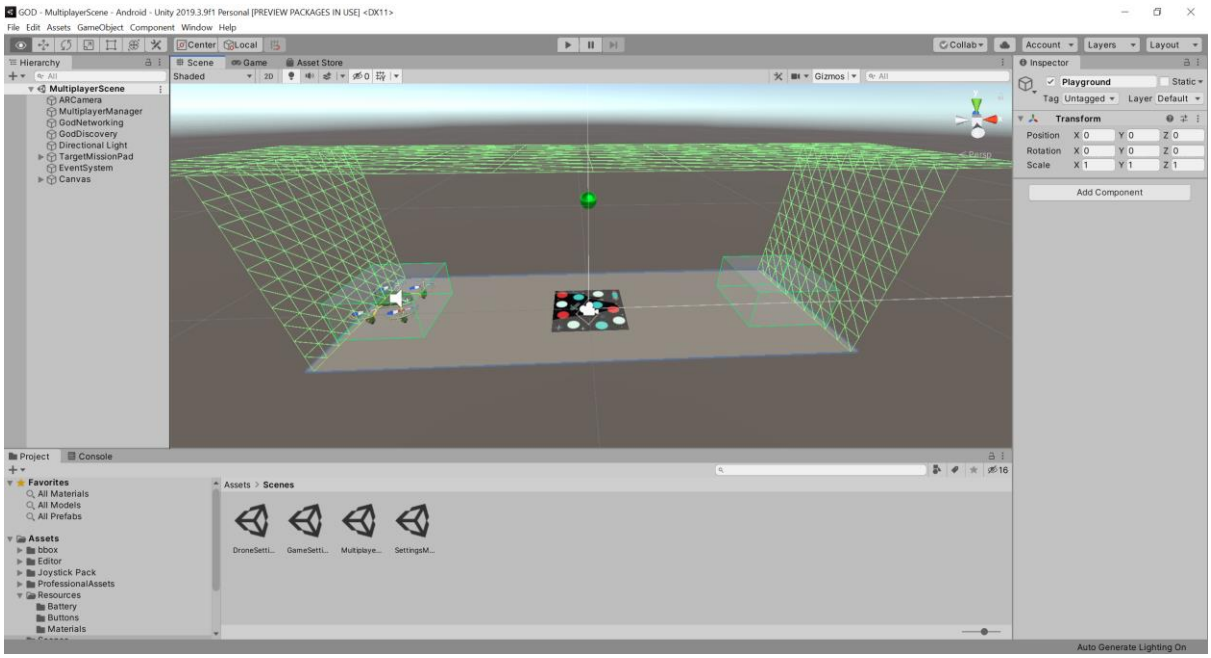

Figure2 - The Unity IDE

About the Vuforia AR Engine

The Vuforia AR Engine provide high-level API for implementing complicated AR-related logic and features. Like Unity, it's cross-platform and harnesses the power of ARkit on iOS devices and ARcore on Android devices.

Vuforia integrates easily with Unity and provides drag-and-drop objects to be used inside the Unity designer.

Among its features are:

1. Image Detection and Tracking
2. 3D Model Detection and Tracking
3. Simultaneous Localization and Mapping (SLAM)
4. Plane Detection

About Augmented Reality

Augmented Reality features the ability to place (augment) virtual object in a real-life scene.
The most extensive usage of AR today is to augment graphics on people's faces in IM and VC applications (Snapchat, Zoom, etc.).

One of the first examples was the ability to augment a virtual mustache on people's faces within an arbitrary image.

Figure3 - Mustache augmentation

In order to do that, the machine must detect a face in the given image and separate its components: eyes, ears, nose, mouth, eyebrows, chicks, chin, forehead, and hair.
In a video application it's also necessary to track these components as they move around from frame to frame and update their corresponding transform accordingly.

Another application for AR is to augment instructions and schematics onto real machines in order to help technicians to service them.

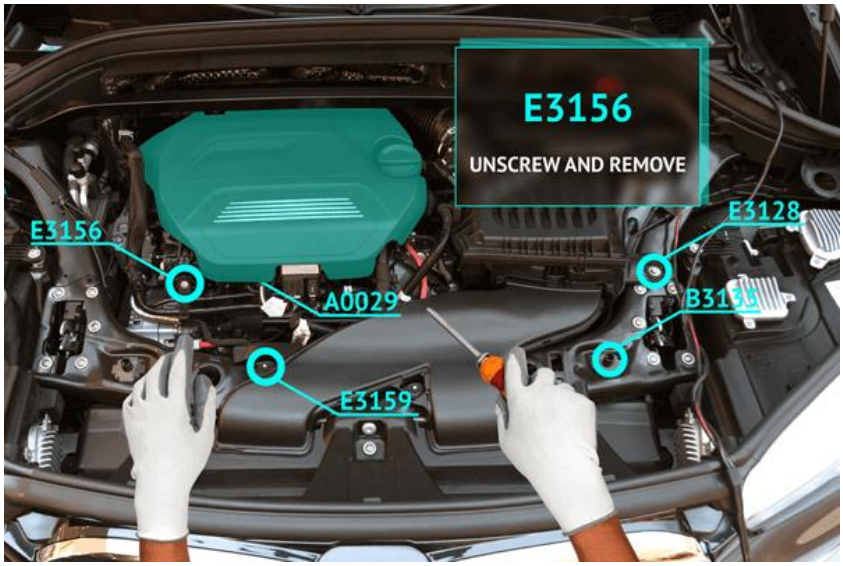

Figure 4 - An AR application for car mechanics

A more advanced example of AR is for home decoration.
In these applications, virtual models of real furniture can be placed inside a room to see how they fit.

Figure5 - AR for home decoration

This use-case is much more challenging due to the absence of a pre-loaded detectable model.
The machine must detect the floor and the walls on its own (plane detection) and estimate the size of the room so that the furniture can be placed with the correct proportions.
Furthermore, since decorating means that the user moves inside the room, there comes the problem of occlusion, in which feature-points are constantly moving in and out of the frame.
The machine has to build and maintain a virtual map of the room, while only "seeing" some part of it at any given time.

How It Works

Hardware

Today's mobile phones incorporate built-in hardware and software especially designed for VR and AR applications.

Modern Apple iPhones are equipped with an A13 Bionic CPU which has a built-in ISP (image signal processor) and even support hardware-accelerated algorithms for image stabilization, lens distortion correction, and even face-recognition. Apple also provides the ARkit framework for developing advanced AR application for their devices.

As for the Android market, all the major manufacturers of CPUs for mobile devices integrate VR and AR capabilities into their chips. Furthermore, the Android operating system comes with the ARcore framework, which is similar to Apple's ARkit.

In order to achieve a good AR experience, a modern smartphone will incorporate a 60-fps capable camera. The reason for this high frame-rate requirement will be described in the following sections.

Image Stabilization Mechanism

Today's smartphones incorporate a built-in hardware accelerated image stabilization mechanism that makes sure that the image received by the software is sharp and motionless. The mechanism handles two problems: the first is the motion blur caused during the image sampling process; the second is vibrations in video clips taken by a hand-held device.
The mechanism first estimate how the camera was moved from one frame to the next. This process is called **global motion estimation** and is usually implemented by performing phase-correlation between subsequent frames. Other common techniques apply some kind of a transform on each frame and then compute the differences between the resulted features. The motion is then estimated by optimizing an argument on the feature differences. These include, for example, the Lucas-Kandade and the Buxton-Buxton methods. Once the motion matrix is estimated, the image stabilization mechanism applies the reverse motion transform on the image to remove any motion blur. The image is then cropped in way that further minimizes the difference between the current frame and its predecessor. This eliminates some of the vibrations that occur during video footage. The resolution of the image is kept constant by using an oversized image sensor with higher resolution. The higher resolution provides the required margins for the cropping stage.

Object Detection and Tracking

In the most basic AR application, one needs to detect an object of interest in an image frame. Although object-detection is an extensively studied problem, most of the research has been focusing on 2D detection only; i.e. detecting an object in a 2D image and estimating a 2D bounding-box around it. This is suitable only for AR applications in which the object of interest takes a major part of the frame and is facing directly at the camera (its pose is straight forward).

Once we know the bounding-box of the object within the frame, it's quite easy to augment some pre-loaded graphics on top of it. However, in case the object's pose changes, the detection becomes harder and the augmentation of the preloaded graphics becomes less convincing (due to the lack of pose information in the transform).

Another problem is that object detection, even in the 2D case, is a very CPU intensive task. Instead of detecting the object in each and every frame, the object is detected only once and then **tracked** in subsequent frames.

Tracking algorithm basically use the a-priori knowledge of where the object was in previous frames to try and estimate where it would be in the current frame. Simple tacking algorithms just apply a detection algorithm on a limited part of the frame, based on the object's previous location. More sophisticated algorithms try to evaluate the trajectory of the tracked object in order to predict more accurately where the object should be in the next frame. Some tracking algorithms can also provide limited pose information which can then be applied on the augmentation to make it more convincing.

Object tracking works well when subsequent frames are very similar. If the current frame differs too much from its predecessor(s), the tracking algorithm can fail. Sometimes, the only way to recover from a tracking failure would be to run the detection algorithm once again, which requires a lot of resources and can be quite slow. As the frame rate goes higher, the more similar are subsequent frames. Tracking performance gets better and thus the overall AR experience.

For mobile phone applications the tracking becomes even more difficult. Changes in frames occur not only because of object motion and static noise from the camera light-sensor, but also because the camera is hand-held and moving. Furthermore, object movement will usually affect only some pixels in the frame, while came movement will always affect all the pixels in the frame. These rapid camera movements can easily cause subsequent frames to become very different from one another.

The camera motion problem is solved primarily by the **image stabilization mechanism** built-in to modern phones, by minimizing the difference between two subsequent frames. As in the tracking problem, the effectiveness of the stabilization mechanism increases with the frame rate.

This is the reason why AR enabled devices are equipped with a camera that allows a 60-fps frame rate or higher.

Localization and Mapping

In some AR applications detecting and tracking an object isn't enough.
Home decoration AR applications build a virtual map of the environment and position virtual objects inside it. In addition, as the user is moving in the physical domain, the application needs to estimate her location on the virtual map. Once a virtual object has been placed, the application must remember where it was, even when the user moves, and the virtual object goes out of frame. This process is called **S**imultaneous **L**ocalization **A**nd **M**apping, or **SLAM** for short. In the AR case, the application actually estimates the location of the camera, rather than the location of the user.

Although SLAM algorithms can utilize inputs from GPS and IMU sensors to improve their accuracy, the vast majority of implementations for smartphones use camera data only. That includes Google's ARCore and Apple's ARkit. The reason for choosing to rely solely on the camera sensor is that GPS data is not available when the user is indoors and IMUs suffer from significant drifts and needs to be calibrated frequently.

SLAM works by choosing segments of the frame that have distinct optical features. These segments are referred to as "feature-points" and are usually points on edges found in the frame. These points are stored in a 3D graph that combines their relative positions inside the frame. This graph is called the "point-cloud". The pose and position of the camera is then determined by tracking the movement of the feature points from one frame to the other. The feature points are all assumed to be static within the scene. When several feature-point are detected in a new frame, the algorithm compares their distances from one another to the distances stored in the point-cloud. The differences are then used to estimate the camera pose and location that would have caused the feature-points to move in such a way. Since SLAM requires all feature-points to be static, each point

receives a score of how reliable it is, or in how many frames it was detected in the position where it was expected to be. Feature-points with low scores are assumed to belong to moving objects within the scene and are therefore dropped from the point-cloud.

In addition to feature-points, some SLAM algorithms use key-frames. These are whole frames that are stored in the point-cloud together with their feature points. This allows the algorithm to look for the entire frame, match the map to the physical world and verify its estimations.

The resulting point-cloud spans a virtual coordinate-system on which virtual objects can be placed with respect to real-world feature-points.

ARCore and ARkit also use the concept of “anchors”, which allows a human to choose some of the feature-points for the algorithm. Anchors are treated by the algorithm as highly reliable feature-points that can be used to re-localize itself. In addition, it tells the algorithm that this part of the map must be kept when the point-cloud gets too big and algorithm decides the purge some of it.

About the Drone

This project was developed using the **Tello EDU** drone by RYZE Robotics (DJI).
This is a small and inexpensive drone that features a reasonable flight time and a very well-performing position control-loop, making it easy to control. In addition, the Tello EDU has an **open SDK** that allows simple interaction from any software running on a WIFI-enabled device.

WIFI Mode

Most drones create their own WIFI network and serve as the access-point, requiring the controlling device to connect to it. When the controlling device connects to the drone's network, it usually loses connection to the outside world and cannot communicate with any other device.

The Tello EDU has the pretty unique ability to connect to an existing WIFI network, rather than creating its own. This feature, known as "swarm mode", enables interaction not only with the drone, but also with other devices connected to the network (and also the Internet).

This feature makes it the perfect candidate for a multiplayer drone game.

Figure 6 - DJI Tello EDU Drone

Technical Specifications

Weight: 80g (with battery)
Dimensions: 98x92.5x41 mm
Propeller: 3 inches
Max Speed: 8 m/s
Max Flight Time: 13 minutes
Max Flight Height: 30m
Battery: 1.1Ah/3.8V
Camera: 5MP (2592x1936 pixels)
FOV: 82.6°
Video: HD720P30
Processor: Intel® Movidius™ Myriad™ 2 VPU
Recommended Price: 99 USD

Game Design

The development of the game featured several difficult challenges that will be discussed in this section. The challenges were:

1. Real-Time Drone Detection and Tracking
2. Playground Placement and Augmentation
3. Real-Time Multiplayer Communication

Many of the above challenges could be solved by running the game-logic on an external system to which the drones and smartphones will connect. The external system can also be equipped with a camera array to track the drones inside a predefined stage. But the desire to be able to run the game just on a smartphone, without any complicated setup, motivated us to try other means.

Drone Detection and Tracking

The first and most critical design challenge of the project was to find a way to determine the location of the drone in the scene. A significant amount of time was spent on testing and evaluating different techniques for drone localization. All of these techniques must perform in real-time on a smartphone with limited CPU and GPU resources.

Using Vuforia Advanced Model Target 360

The Vuforia engine provides an object tracking feature called “Advanced Model Target”.
This feature allows developers to upload a CAD model of the desired object to the Vuforia cloud, where it is trained by a deep-learning process to produce an Advanced Model Database.
Unfortunately, there is no much documentation on the inner workings of the Advanced Model feature, being a proprietary technology.

The Advanced Model Target 360 feature has two major limitations:

1. Object detection requires the object to be entirely static.
2. The object should take at least 30% of the screen in order to be first detected.

At this stage of the project we tried to use the Parrot Bebop 2 drone, which is 32.8x32.2 cm.
Using a large drone should allow easier detection and tracking.

Figure 7 - The Parrot Bebop 2 Drone

Indeed, during our tests we found that once the drone is detected, Vuforia manages to track it up to around 3 meters. However, tracking worked only if the drone was moving very slowly. Much too slow to make a pong game. Furthermore, once tracking has failed, the drone had to be static in order to detect it again. Finally, the attempt to augment a virtual ping-pong bat on top of the drone

resulted in a very jittery experience even when the drone was not moving at all. At this stage we decided to abandon this method and look for other means for drone localization.

Figure 8 – An Augmented Ping-Pong Bat on the Bebop 2

Figure 9 - An Augmented Ping-Pong Bat on the Bebop 2

Using Vuforia Image Target

The image tracking feature provided by Vuforia is called “Image Target”.
It is much more mature and reliable than the Advanced Model Target feature for object tracking and it is primarily based on image-tracking facilities from the ARCore and ARkit libraries.

The detection and tracking performance of the Image Target feature highly depends on the image of interest. In order to optimize tracking performance, Vuforia published best practices for designing a trackable image:

1. The image should be rich in detail
2. The image should be with good contrast
3. In order to avoid ambiguity during detection and tracking, the image should not contain too many repetitive patterns, avoid symmetry and large featureless areas.

Following the Vuforia best-practices, designing a good image was not a real challenge. Our first attempt was with a QR-code, as QR-codes are specifically designed for detection.
As can be seen in the image below, our QR-code received a 5-star rating. It has plenty of features, perfect contrast, no repetitive patterns and no symmetry.

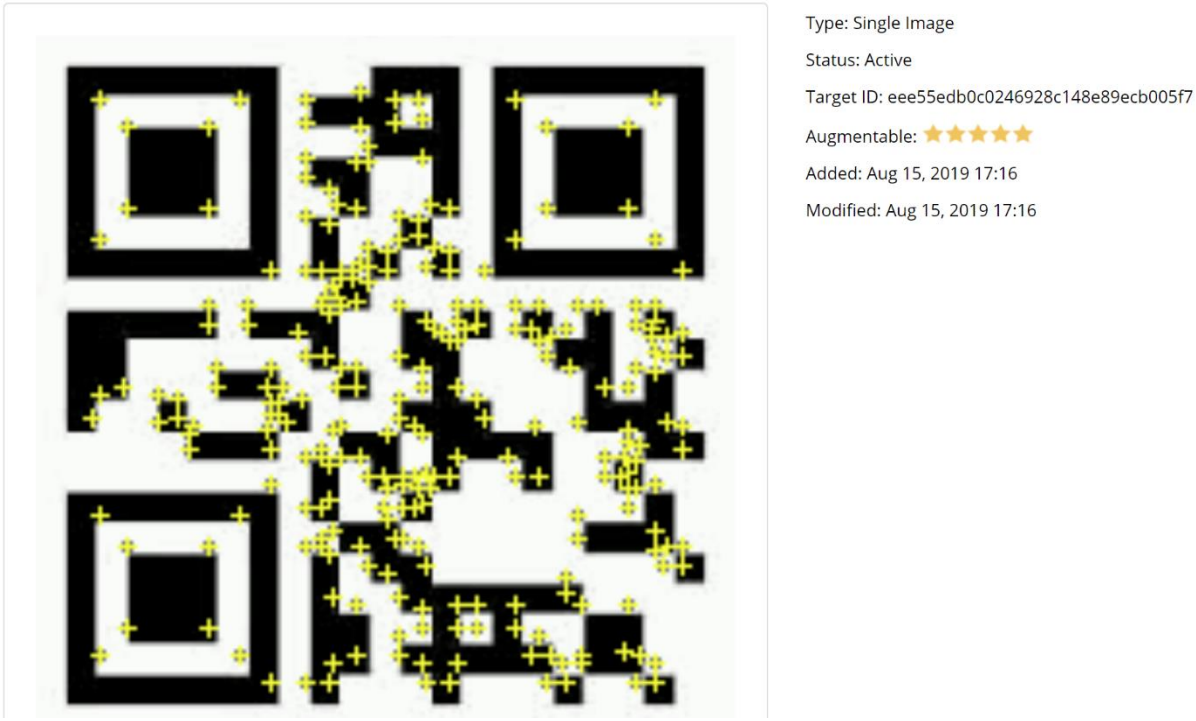

Figure 10 - Drone QR Code

The QR-code provided very good results. An QR-code of size 20x20 cm was successfully detected from up to 1 meter and tracked up to 3 meters.

However, image had to be mounted on top of the drone and the length of the Tello EDU body is just 6.5cm, front to back. Using an image so small requires the drone to be very close to the camera for the image to be detected - no more than 30cm. Furthermore, tracking now worked for only up to 1m with perfect lightning conditions.

In order to achieve reasonable tracking performance, the size of the image had to be increased. Furthermore, the image had to be placed on a rigid surface in order to remain flat while the drone is airborne.

A simple solution was to use a standard plastic card measuring 54x84mm as the image surface. The image can then be printed in a regular printer and glued to the plastic card.

A new rectangular image was designed to be both trackable and a bit more playful than a boring barcode. As can be seen below, the resulting image-target receives a perfect score from the Vuforia target manager.

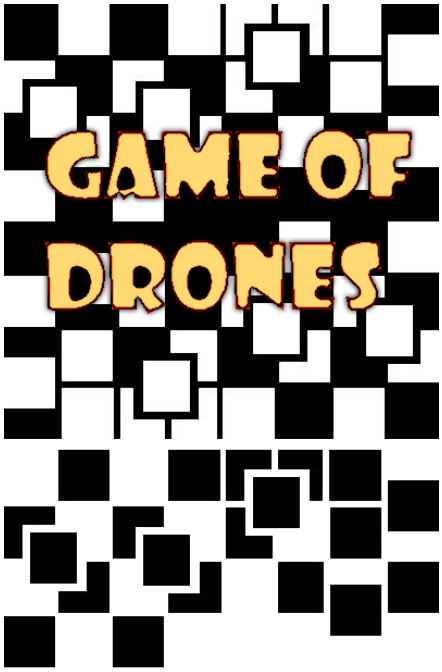

Figure 11 - The Final "Game of Drones" Image Target

Type: Single Image

Status: Active

Target ID: d148e4c6ea04457c82531ea7739065f0

Augmentable: ★★★★★

Added: Apr 22, 2020 12:17

Modified: Apr 22, 2020 12:17

Figure 12 - Game of Drones Image Target Ranking

Using Drone Height Measurements

The Tello EDU drone measures its height above the ground using a laser sensor on its bottom. These measurements are reported as part of the drone's telemetry at a rate of 100Hz. The idea was to use the height readings from the drone to determine its location on the vertical axis relative to a trackable image that is placed on the floor. Since the image is static, tracking becomes much easier and it can also be used as an anchor for the virtual world coordinate system. Assuming the drone will be moving only on the vertical axis above the image target, the height measurements should provide a very accurate location.

Testing out this idea revealed two problems:

1. The height readings were reported at a rate of 100Hz but were accurate only to 10cm. This resulted in the drone augmentation to move spasmodically. This problem was partially solved by using a 1-tap low-pass filter to interpolate the drone height points between the readings and make the augmentation to move more smoothly. Using just a single tap prevented an observable delay and provided a very good result.
2. Once the drone took off, it became very hard to get it inside the frame together with the image on the ground. In addition, the angle between the camera and the floor was close to 90°, resulting in a very poor image tracking performance.

Our conclusion was that these problems rendered the idea impractical for our game.

Using Mission-Pad Detection

The Tello EDU drone has an on-board capability to detect “mission pads” placed on the ground and report their relative position on its telemetry channel.

The idea was to train Vuforia to detect a mission pad and use it as an anchor. The drone position can then be determined relatively to the mission pad from the drone’s readings.

However, as in the Drone Height Measurements idea, this idea also suffered from the fact that tracking an image on the ground becomes very hard when the drone is airborne, and the phone’s camera is perpendicular to the ground.

Another problem was that as the drone moved up and down it started to drift away from the mission pad until it could no longer detect it. Finally, the mission pad location estimation from the drone was extremely noisy and inaccurate.

Eventually, we had to rule out also this idea.

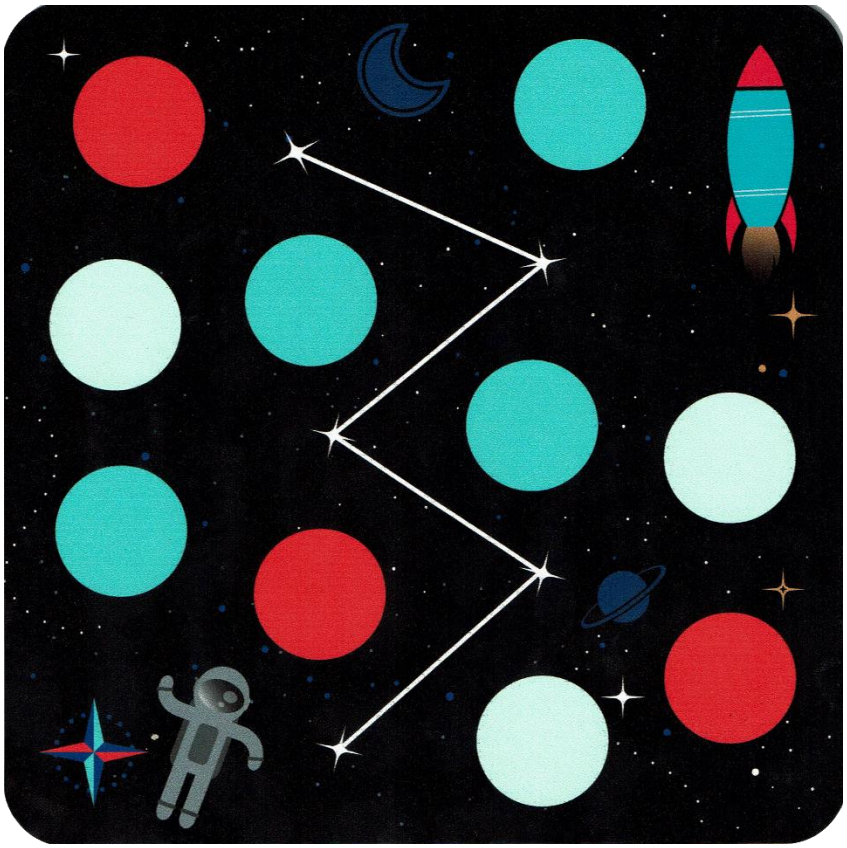

Figure 13 - A Tello EDU Mission-Pad

Final Decision

All our ideas for estimating the drone’s position in the scene were eventually ruled-out after testing, except for using an **Image Target**. In addition, the Image Target solution proved to provide reasonable tracking performance for the game.

Playground Placement and Augmentation

Drones are the paddles in the Pong game, but what about the walls and the ball? The paddles should be moving relative to the wall they are protecting, and the ball should be moving on the plane between them. So how do we place the playground?

Using an Anchor on the Ground

One of the simplest and best performing ways to augment the playground was to use an Image Target on the ground as an anchor. Since the image is static, the AR engine can always use it to re-calibrate itself and validate its point-cloud. Furthermore, when tracking fails, reacquisition becomes very simple once the anchor goes back into the frame. In addition, the anchor can be used to match the existing point-cloud more accurately with the physical world.

Below is a screenshot from an early version of the game demonstrating augmentation of a virtual drone on a virtual mission pad (on the right) next to a real mission pad (on the left).

Figure 14 - Mission Pad Augmentation

The idea was that each player will place her drone on a mission pad that will be detected by the application. Each drone will fly up and down only above the mission pad on the floor below it. The ball will be moving horizontally between the two mission pads, hitting the left of right walls defined by the mission-pad edges. The virtual mission pad will always be placed on the floor, in a constant predefined position from the real mission pad.

While the mission pad detection and augmentation worked very well, there was the problem of keeping it inside the frame together with the drone while the drone is airborne. Unfortunately, we couldn't provide a good gaming experience while forcing the user to keep the mission-pad in sight and we had to abandon this idea.

Using a User Chosen Anchor

The final solution was to ask the user for help. When the application starts the user is requested to point the camera forward and click on a static point in the frame. This causes Vuforia to use this point and several points in its environment as anchors for its point-cloud. This is when SLAM is initiated, and a virtual map of the scene is starting to form. The playground can now be augmented in the virtual environment.

This worked quite well, but there was still one more challenge that had to be addressed.
The problem was that the drone could not keep a constant position while changing its altitude. Instead, it drifted slightly on the horizontal axes until it eventually went out of the playground entirely. Keeping the drone inside the playground while playing turned out to be extremely challenging, resulting in a poor gaming experience.

If the drone can't be forced to stay within the playground, we can force the playground to track the drone. Since the paddle in a Pong game only moves on a single axis, the other two axes can be controlled by the game rather than by the player. Updating the X and Z coordinates of the playground according to the X and Z coordinates of the drone made sure that the drone will always keep its location inside the playground, except for its height.

Real-Time Multiplayer Communication

A two-player game requires that the two smartphones will agree on three things:

1. The position of each player at any given time
2. The position of the ball at any given time
3. The final score

The challenge is that the latency must be kept to a minimum in order to provide a reasonable game experience. Too much of a delay can cause inconsistencies and disagreements between the two players – did the ball hit the drone or the wall? Did it hit the front of the drone, and thus be moving backwards, or it hit the top of the drone and thus be moving upwards?

Furthermore, if a message from one smartphone doesn't reach the other smartphone on time it has no value and must be discarded. The conclusion is that the communication channel between the two smartphones should be as **fast** as possible and **unreliable**. Retransmissions are not allowed since a retransmitted message will not be relevant by the time it gets to the other side.

All the above considerations made the choice of UDP quite easy for our use-case. UDP is an unreliable connectionless protocol that takes minimal resources from the smartphone and provides minimal latency for delivered datagrams.

Unfortunately, our tests showed that smartphones impose very high latency even when UDP is being used. A reasonable gaming experience could only be achieved when the two smartphones are on the same WIFI network and communicating directly with each other. Even in this setup the latency can still go as high as 60ms on a Samsung Galaxy S8. Furthermore, our tests revealed that if one of the smartphones is serving as a WIFI hotspot for the other, the latency can go skyrocketing up to 250ms. Therefore, a custom communication protocol was developed for the game as will be described in the following sections.

For simplicity, the GOD protocol uses plain-text ASCII messages sent in a single datagram.
In addition, to avoid fragmentation can cause latency, the length of all GOD messages was limited to 508 bytes, which is the maximal UDP payload size that is guaranteed by the IPv4 standard not to be fragmented.

Player Discovery

Before the players can communicate, they need to find one another. This was implemented by sending broadcast messages over the subnet on UDP port 5555 every 1 second. When the application receives a broadcast message from another smartphone, a quick handshake is performed and a new unicast UDP channel is established.

A discovery datagram begins with the magic sequence "GOD HELLO" followed by the company name (tau), the product name (GameOfDrones) and the current version. In addition, the datagram contains a unique instance identifier that is used to identify the current session.

Following is an example GOD discovery payload:

```
GOD HELLO tau.GameOfDrones 1.0
instance: 334
```

When a datagram is received on port 5555, the application first looks for the magic sequence. If there is no match, the datagram is simply discarded.

Once the magic sequence is validated, the IP address, source UDP port number, and the given instance number of the remote user are registered. They will be used to separate the discovered partner from other GOD broadcasts in the subnet.

At this stage a new unicast UDP channel is established on port 5556, that will be used for GOD messages between the two parties.

GOD Messaging Channel

When a GOD messaging channel is established between two sides, the parties agree which will act as the master and which will act as the slave. The rule is that the party with the smaller IP address is the master.

The GOD messaging channel carries only a single type of message that is called "GOD Update". This message encapsulates the entire state of the game from its sender's perspective.

Message fields are as follows:

1. Message sequence number.
   This field can be used to ensure that old messages received after newer messages are not processed after newer messages.
2. Game Status (encoded as an integer bit-field):
   a. Is the drone connected?
   b. Is the drone airborne?
   c. Is the drone detected and being tracked by Vuforia?
   d. Did the player indicated that she is ready to start the game?
3. Ball Position (x,y,z):
   The ball position is given with respect to the playground position and therefore invariant to the placement of the playground in the scene.
4. Ball Speed Vector (x,y,z)
5. Drone Position (x,y,z):
   The drone position is given with respect to the playground position and therefore invariant to the placement of the playground in the scene.
6. Game Score (my score, your score)

Following is an example GOD update payload:

```
GOD-Update 123 ball-p:1:2:3 ball-s:4:5:6 drone-p:7:8:9 score:10:11 status:6
```

When either side receives an update message, it will store it and use it on the next frame update to update the position of the other player's drone. Only the most recent GOD-Update message is stored at any given moment. In case several GOD-Updates arrive between two frame updates, only the most recent one will be processed.

When the slave party receives an update message, it will also update the score, the ball position, and velocity vector. This will make sure that the parties agree on the score.

In case the latency is too high, the ball movement on the slave side will not appear smooth. In this case it might be better to disable ball updates. If the ball is not moving too fast the ball position will be very similar even without the updates and the overall game experience will be much better.

User Guide

This section explains how to use the game and what kind of setup is required.

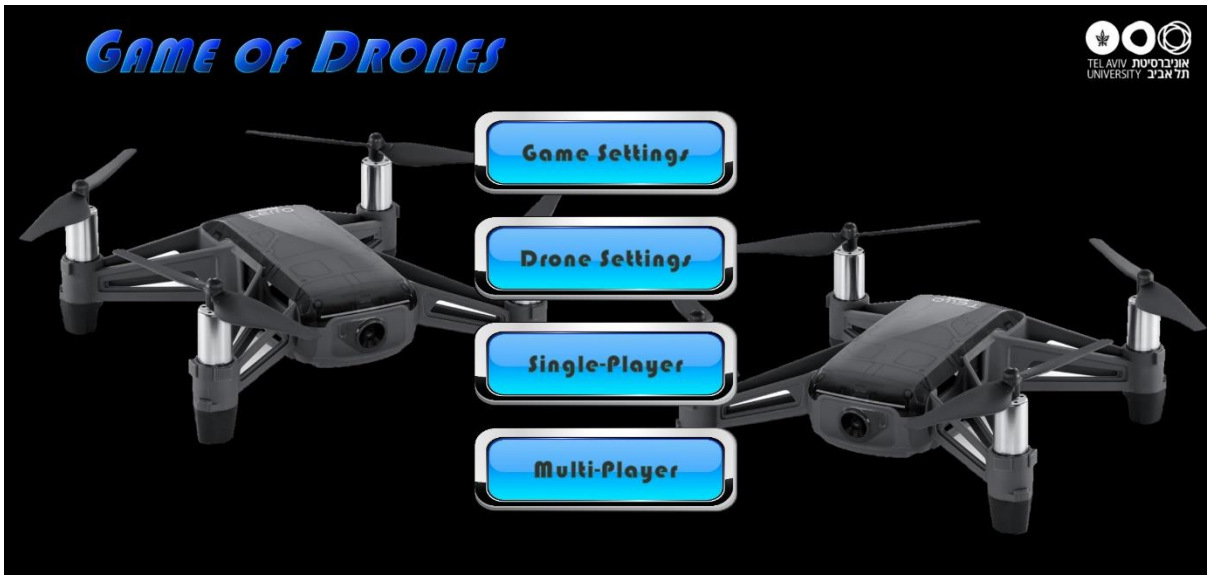

Figure 15 - Main Menu

Game Settings

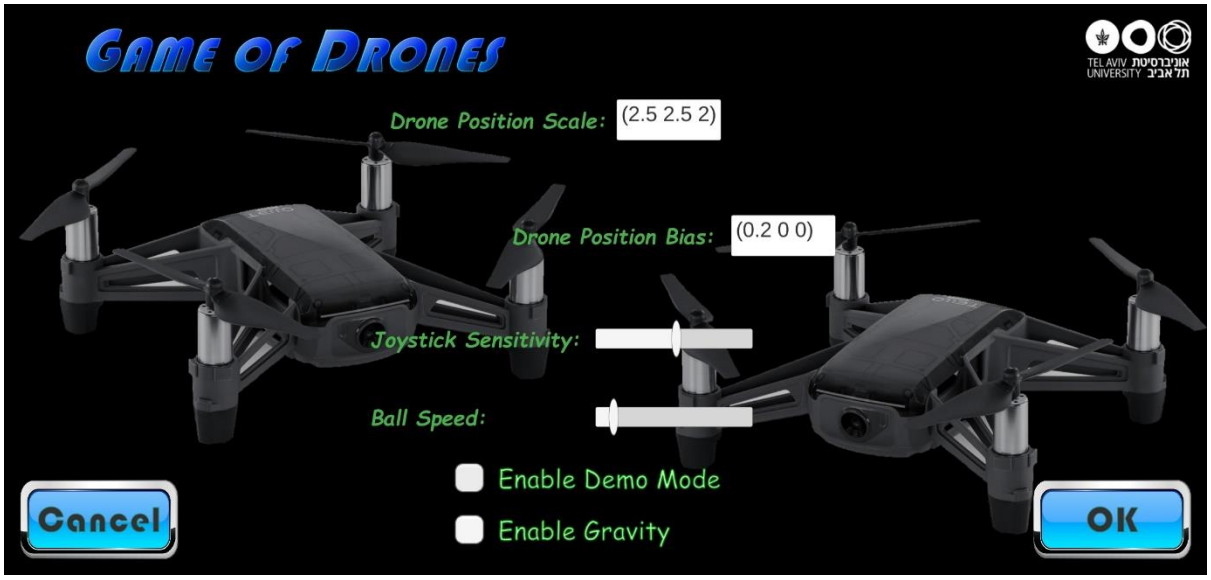

Figure 16 - Game Settings Menu

The game settings menu provides control on different features of the game.

1. Drone Position Scale:
   This is a vector that scales the drone position estimate acquired from Vuforia.
   This is a calibration field that needs to be setup differently for each smartphone model.
2. Drone Position Bias:
   This is a bias added to the drone position estimate acquired from Vuforia.
   This is a calibration field that needs to be setup differently for each smartphone model.

3. Joystick Sensitivity:
   This field controls how fast the drone will respond the joystick commands.
   Lower sensitivity will ease the control over the drone but will also reduce its speed.
4. Ball Speed:
   This field specifies how fast the ball should be moving.
   The faster the ball moves the harder the game becomes.
5. Enable Demo Mode:
   This feature enables testing the Multiplayer Mode functionality without connecting to another player. The drone that belongs to the opponent will be positioned according to the drone of the current player.
6. Enable Gravity:
   This feature controls whether the ball is affected by gravity.
   If gravity is disabled, the ball moves in a constant direction until it hits something, just like in the original Pong game; Otherwise, the ball will drop and bounce off the augmented virtual floor.

Drone Settings

Figure 17 - Drone Settings Menu

This menu specifies the communication methods to the drone.

As mentioned before, the Tello EDU drone can act either as a WIFI access-point (default) or as a WIFI station.

In the WIFI access-point mode, the drone creates its own network to which the smartphone has to connect in order to communicate with the drone.
In this mode, the IP address of the drone is constant and is set to 192.168.10.1.
Multiplayer cannot be played in this mode, as the two smartphones will not be able to communicate with each other since they are on different networks.

In the WIFI station mode, the drone will connect to the given WIFI network and obtain an IP address automatically using the DHCP protocol. The IP address of the drone can be obtained from the WIFI access-point. You should look for the dynamic address allocations table. This IP address must be configured in the Drone Settings menu for it to work.

Important note: any changes made in the Drone Settings menu will be committed to the current IP address specified in the "Drone IP Address" field. Make sure to change the IP address **only after** modifying and **committing** all the other settings to the drone by pressing the OK button.

Remember that you might need to switch between WIFI network while configuring drone settings.

Single Player

This mode allows a player to play against a (very) simple bot.
See the Multiplayer section for more details.

Multi-Player

This mode allows two players with two smartphones and two drones to play against each other.

Before the game can start, each player must choose an anchor for the Vuforia SLAM.
This is done simply by pointing the phone to a static scene and pressing the green dot on the screen.
Note that the two players don't have to be in the same location and don't have to see the same scene in their cameras.

For the best AR experience, make sure to hold your phone perpendicular to the floor while pressing the green dot.

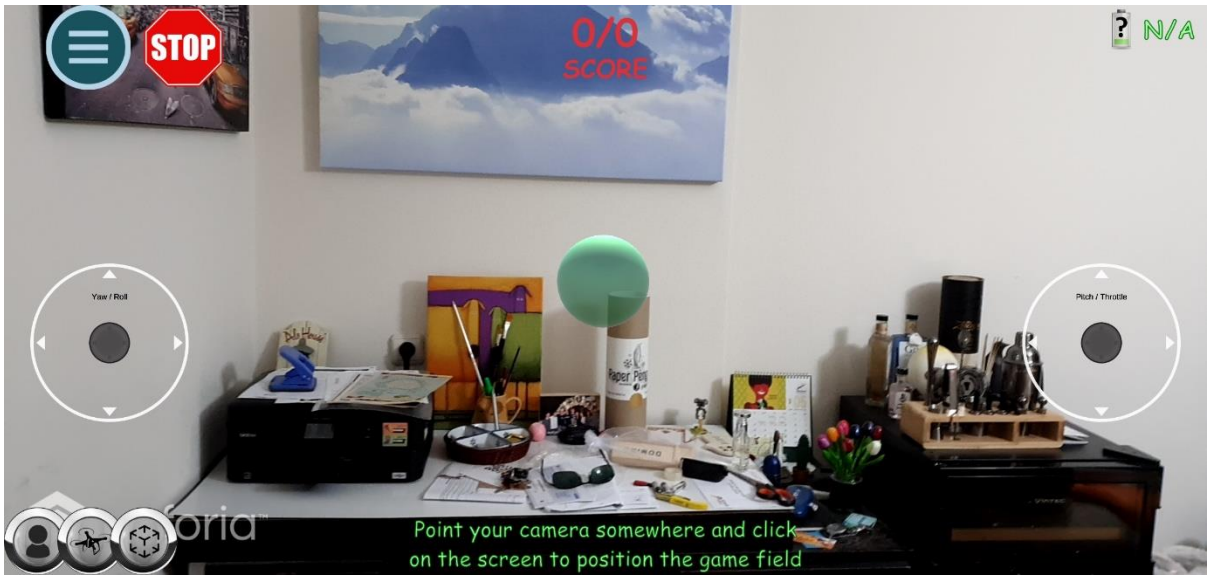

Figure 18 - Anchor Selection (SLAM Initialization)

On the top left of the screen there are two buttons.
The right button is the **Stop** button, which stops the game and sends a **land** command to the drone.
This button can be pressed in any situation to try and tell the drone to land.
The left button is the **Main Menu** button. Pressing it will stop the current game and return to the main menu.

On the bottom left of the screen there are the three status indicators.
The leftmost indicator indicates the status of the communication with the other player.
A gray background means that no other player has been discovered;
An orange background means that communication with the other player has been established;
A green background means that the other player has indicated that she is ready to play.

The middle indicator indicates the status of the communication with the drone.
A gray background means that the application cannot communicate with the drone;
An orange background means that communication with the drone has been established and that its motors are currently off;
A green background means that the that communication with the drone has been established and the drone is airborne.
If drone settings were configured properly and the smartphone is connected to the same WIFI network, the application will connect automatically to the drone.

The rightmost indicator indicates whether the drone image target was detected in the scene.
A gray background means that the image target wasn't detected or that tracking has failed;
A green background means that the image target was detected and is being tracked.

Note: during a game all three indicators must always be green.

The battery indicator on the top right of the screen indicates the remaining battery **in the drone**, not the smartphone.

The target button on its left will allow re-initialization of the SLAM by specifying a new anchor.

On the left and right middle edges of the screen there are two joysticks.
The left joystick controls the drone's yaw and roll, while the right joystick controls the drone's throttle and pitch.

Finally, the ready button in the center of the screen signals to the other player that you are ready to play.

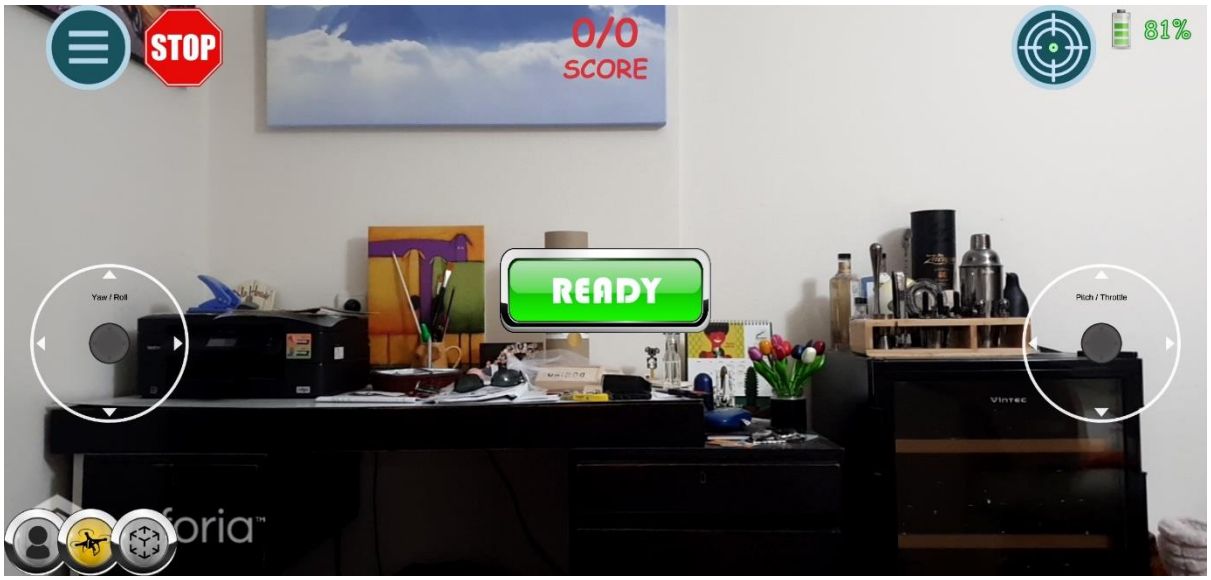

Figure 19 - Multiplayer Idle

Once the ready button is pressed, the application makes sure that all preconditions for starting the game apply:

1. Drone image (sticker) was detected and is being tracked for both players
2. Both applications managed to establish a connection with their drones
3. Communication was established with the other player (except for demo mode)

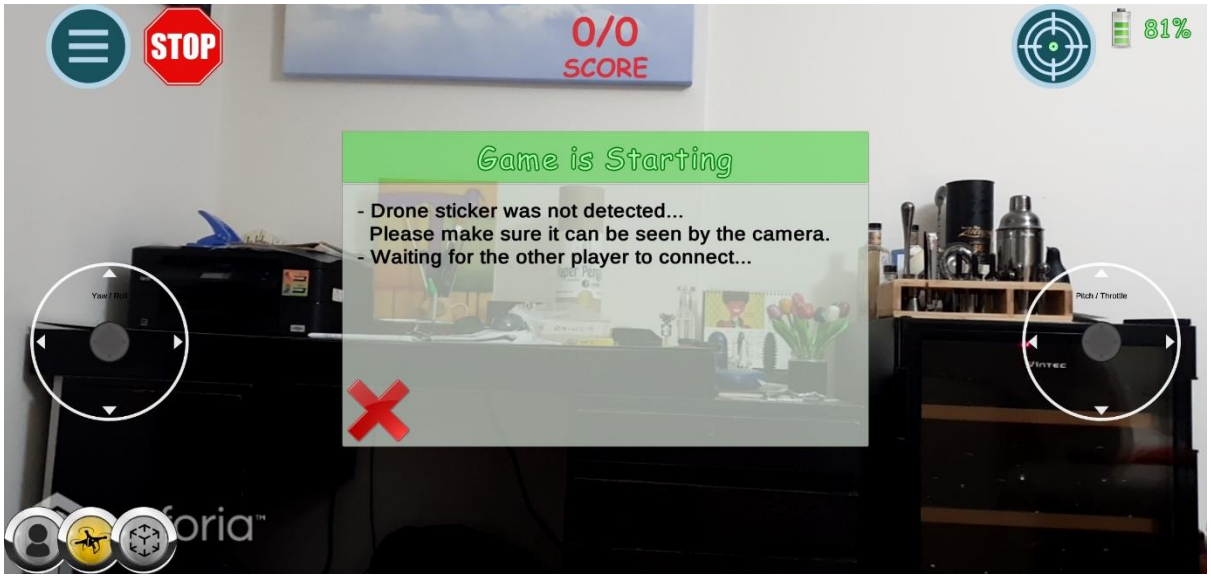

Figure 20 – Ready Button Pressed - Game is Starting

Figure 21 - Drone Sticker Detected

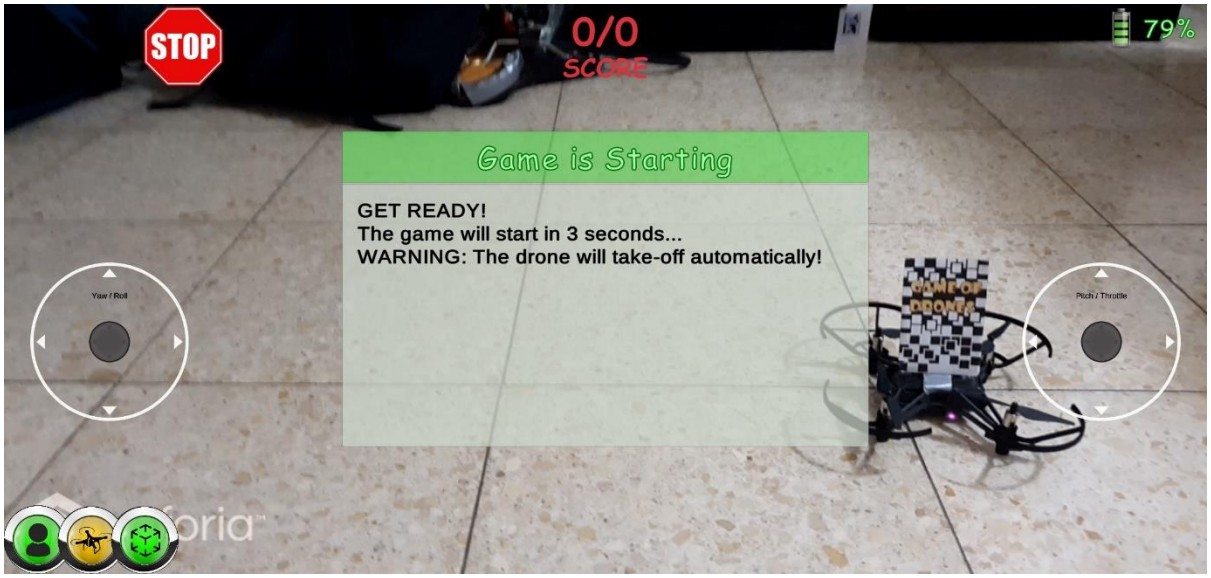

Figure 22 - Everything is Set

Figure 23 - Game in Progress

References

1. K. Sartipi, R. C. DuToit, C. B. Cobar and S. I. Roumeliotis, "Decentralized Visual-Inertial Localization and Mapping on Mobile Devices for Augmented Reality," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China, 2019, pp. 2145-2152, doi: 10.1109/IROS40897.2019.8967804.

2. Danping ZOU, Ping TAN, Wenxian YU. Collaborative visual SLAM for multiple agents: A brief survey. Virtual Reality & Intelligent Hardware, 2019, 1(5): 461—482
DOI: 10.1016/j.vrih.2019.09.002

3. R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," in *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147-1163, Oct. 2015, doi: 10.1109/TRO.2015.2463671.

4. J. R. Puigvert, T. Krempel and A. Fuhrmann, "Localization Service Using Sparse Visual Information Based on Recent Augmented Reality Platforms," 2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct), Munich, Germany, 2018, pp. 415-416, doi: 10.1109/ISMAR-Adjunct.2018.00123.

5. Tello SDK 2.0 User Guide
Ryze Tech, 2018.11

6. Vuforia Developer Portal:
developer.vuforia.com

7. ARCore Developer Portal:
developers.google.com/ar/develop

8. ARkit Developer Portal:
developer.apple.com/augmented-reality

9. Unity Developer Portal:
unity3d.com

10. TelloPilots Forum – Reversing the Tello native protocol
tellopilots.com/forums/tello-development