**AR Assembly System for Reconfigurable Multicopter Drone Delivery System**

## FAST (Functional Analysis System Technique)

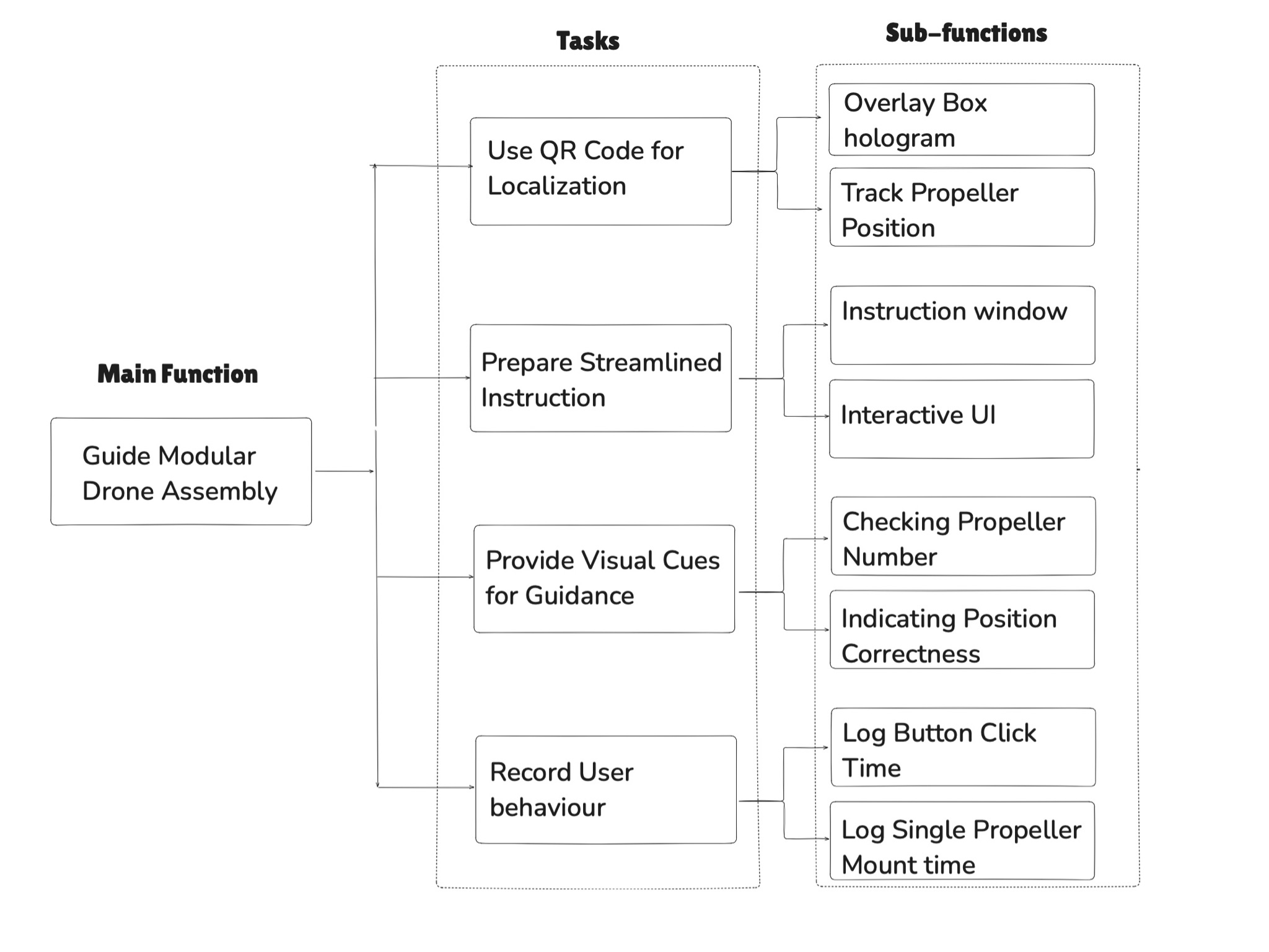
****

Fig. 1. FAST diagram.

Thanks to the FAST, we were able to identify a number of sub-functions and tasks supported by these sub-functions in order for the system to achieve its primary function (see Fig. 1). These are as follows:

1. **Use QR codes for part localization**: The system uses multiple QR codes to track the location of different drone parts. A large QR code is centrally placed on the drone's main body and acts as the anchor for all tracking capabilities. By tracking the position of this central QR code, the AR interface can overlay digital CAD onto the drone's structure. Furthermore, the desired positions of the propellers are also displayed relative to this main QR code. Each such propeller has a small QR code attached, allowing the system to perform two key functions: (1) identify and verify the propeller number, and (2) track the propeller's position to ensure it is mounted correctly.
2. **Prepare Streamlined Instruction**: A guidance window shows and explains the steps for drone reconfiguration (copter attachment).
3. **Provide Visual Cues for Guidance**: Implement means (arrows) that indicate the direction to slide the propeller. When the user looks for a specific number index of the propeller, based on the small QR code, the system should show an aid (a cross) to indicate if this is the correct part. When the user is mounting the propeller, another visual aid (a red ball) will be shown above the QR code, and if inserted at the correct position, the aid will change colour (turn green).
4. **Record User behaviour**: Log the time when the user presses every button, and use this information to calculate how long it takes to mount one propeller.

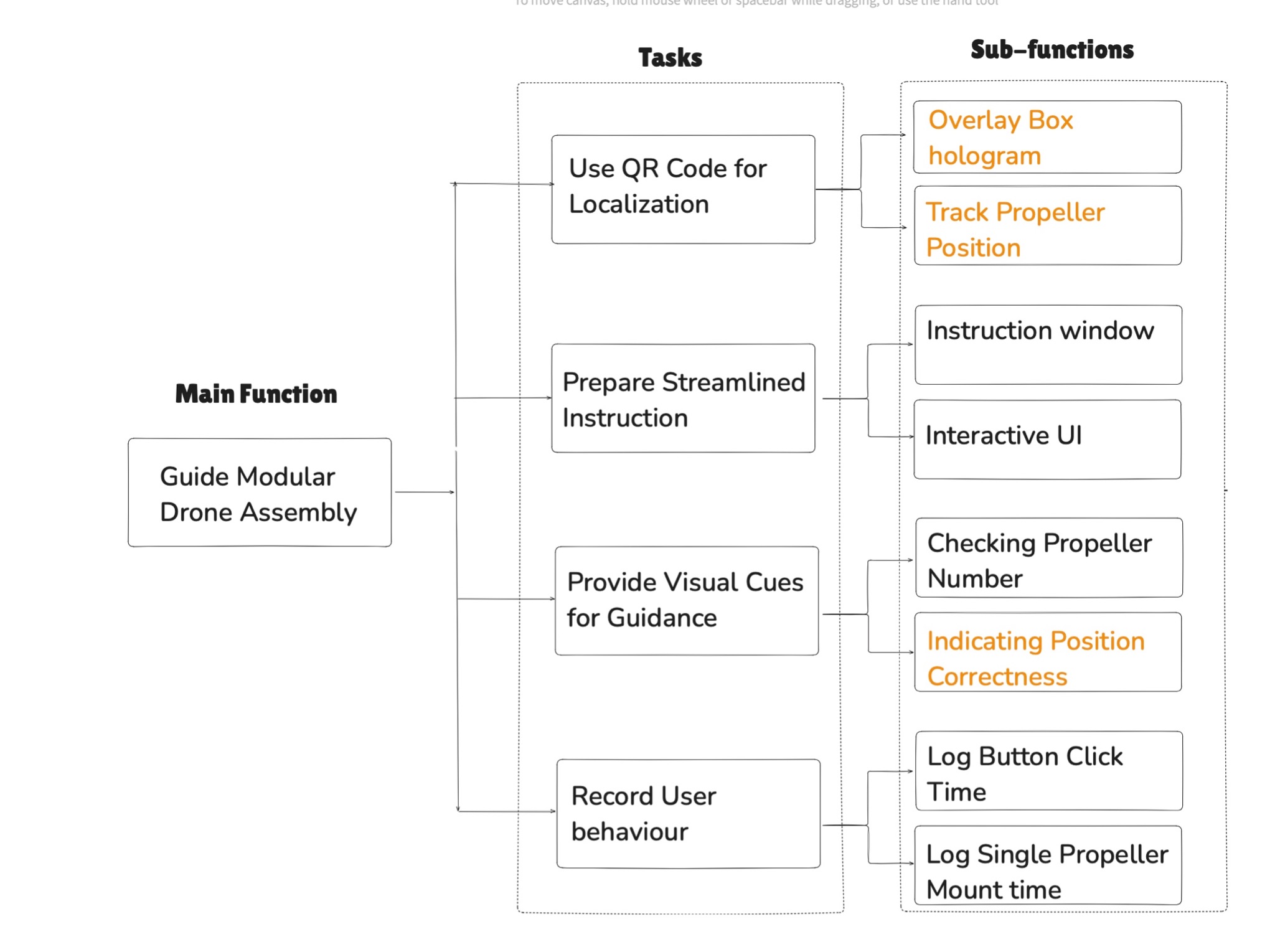


Fig. 2. FAST diagram with critical sub-functions denoted in orange.

## Critical Function Analysis

Among all the sub-functions identified through FAST (see Fig. 2), we were able to select three that can be denoted as critical, as their malfunction might cause the whole system to fail. These are:

1. **Overlay hologram:** We track the large QR code and obtain its coordinate (world centre coordinate). Due to hardware limitations, the accuracy of this process is limited. Here we used the Vuforia engines, which causes a certain offset between the QR code and the overlay in the direction between the camera and the code that needs to be manually compensated for. After compensation, the error should fall within 1 cm. Furthermore, to prevent the “shaking” of the overlay caused by frequent head movements, we introduced the **Freeze** function so the algorithm can stop scanning the QR code when the accurate hologram (overlay) position is found.
2. **Track Propeller Position:** We track the propeller position of using a small QR code attached to each such part. However, as the QR code is very small, the offset in the depth direction is even greater than in the case of a large QR code. Therefore, we need to transform the absolute coordinates to the coordinate system relative to the camera, so we can neglect the offset in the depth direction.
3. **Indicating Position Correctness:** Following the tracking of the small QR code, we are showing a visual aid (i.e., a small sphere) to tell the user whether the propeller is within the 2-3 cm tolerance.

## Failure Mode and Effects Analysis (FMEA)

1. **Software and Hardware Prolonged Usage**: As Vuforia is a third-party package, it places additional workloads on the headset's built-in CPU of HoloLens. As a result, the system's response time is longer than usual, and the hardware (i.e., the headset) can heat up and no longer be usable until cooled down.
2. **Battery Depletion:** When HoloLens is used for an extended period, its display dims, which can only be resolved by rebooting the system.
3. **Simulation Sickness:** The use of immersive interfaces can lead to discomfort and dizziness. This effect, called “simulation sickness,” is an ongoing area of research. While it is more prevalent in virtual reality, it can also be present when using HoloLens for an extended period.
4. **Colour Vision Deficiency:** When deployed in certain spaces, colour-coding of the physical background (wall paint or carpet colour) may interfere with the correct identification of system’s components by its users.
5. **Interaction Insensitiveness:** For a number of new users, the act of clicking the virtual button within the HoloLens can be insensitive to their hand movement.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. | Task steps | Failure modes | Failure causes | Probability | Failure effects | Actions |
| 1 | During App running | App becomes slow and sudden shutdown occurs | The app is putting too many workloads on CPU | High | Hololesn becomes too hot, app is too slow to work | Not using recording.  Prevent using app for too long |
| 2 | During App running | Display got dark | Battery depletion | Low | User cannot see holograms clearly | A restart could help  Charge Hololens timely |
| 3 | During App running | User feel sick | Bad synchronization and visual display | Moderate | Reduced training effectiveness. | Put hologram in the scene, avoid having hologram following user’s view |
| 4 | During App running | User cannot view the real world clearly | The hologram being too bright | Moderate | Less accuracy, longer operating time | Flip Hololens glasses  Increase the transparency of the hologram |

**AR Development**

**1.0 Project setup**

Prerequisite:

Unity Hub

Unity editor(2021.3.21f1, you can use later version)

Visual Studio 2022(with required component installed)

Windows 10 or 11PC(or use Mac with Virtual Machine)

Mix Reality Feature Tool

<https://learn.microsoft.com/en-us/training/modules/learn-mrtk-tutorials/>

Check this website for detailed tutorial for installation

**Software:**

MRTK2

<https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05>

Vuforia

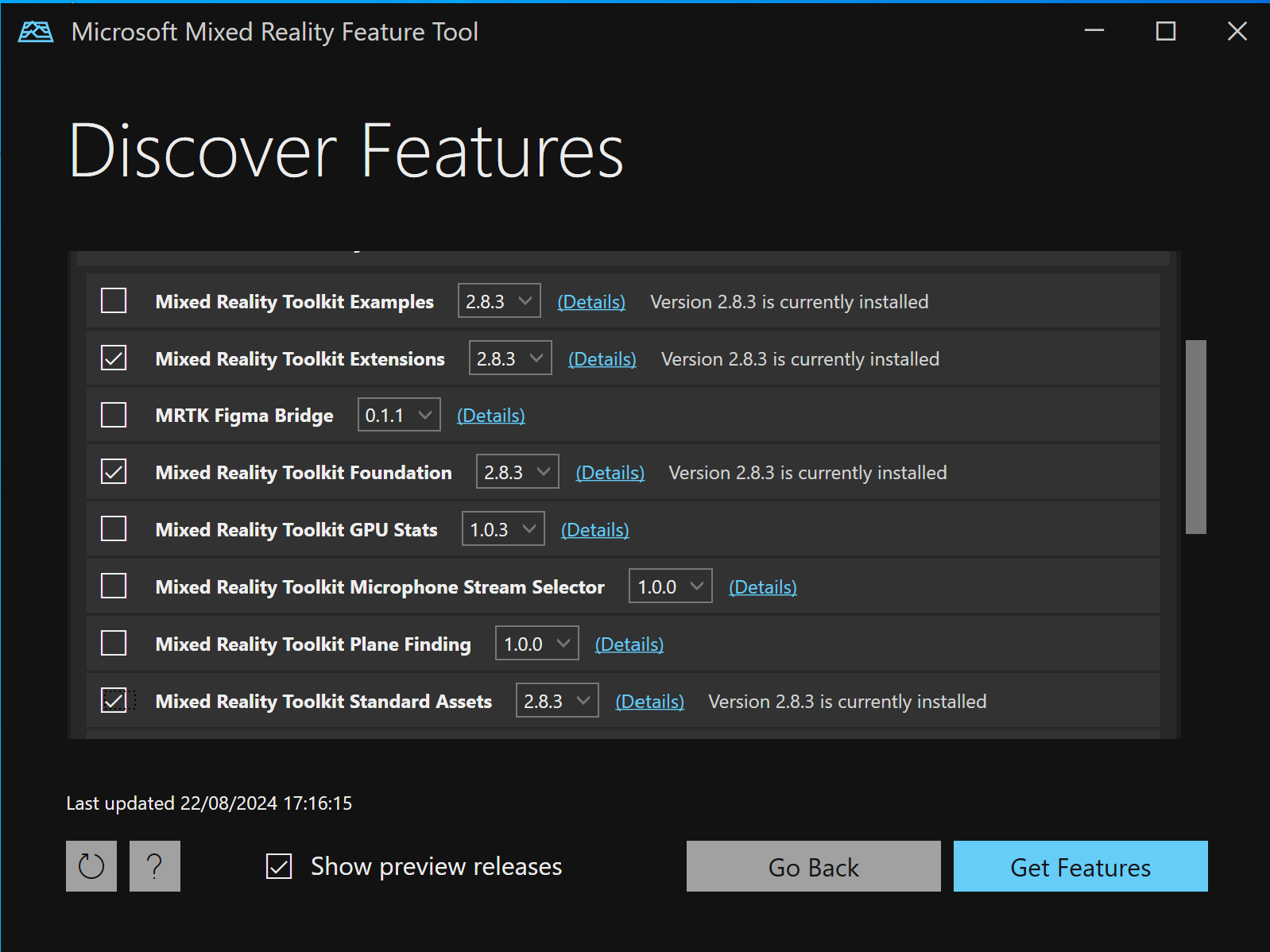
<https://developer.vuforia.com/home>

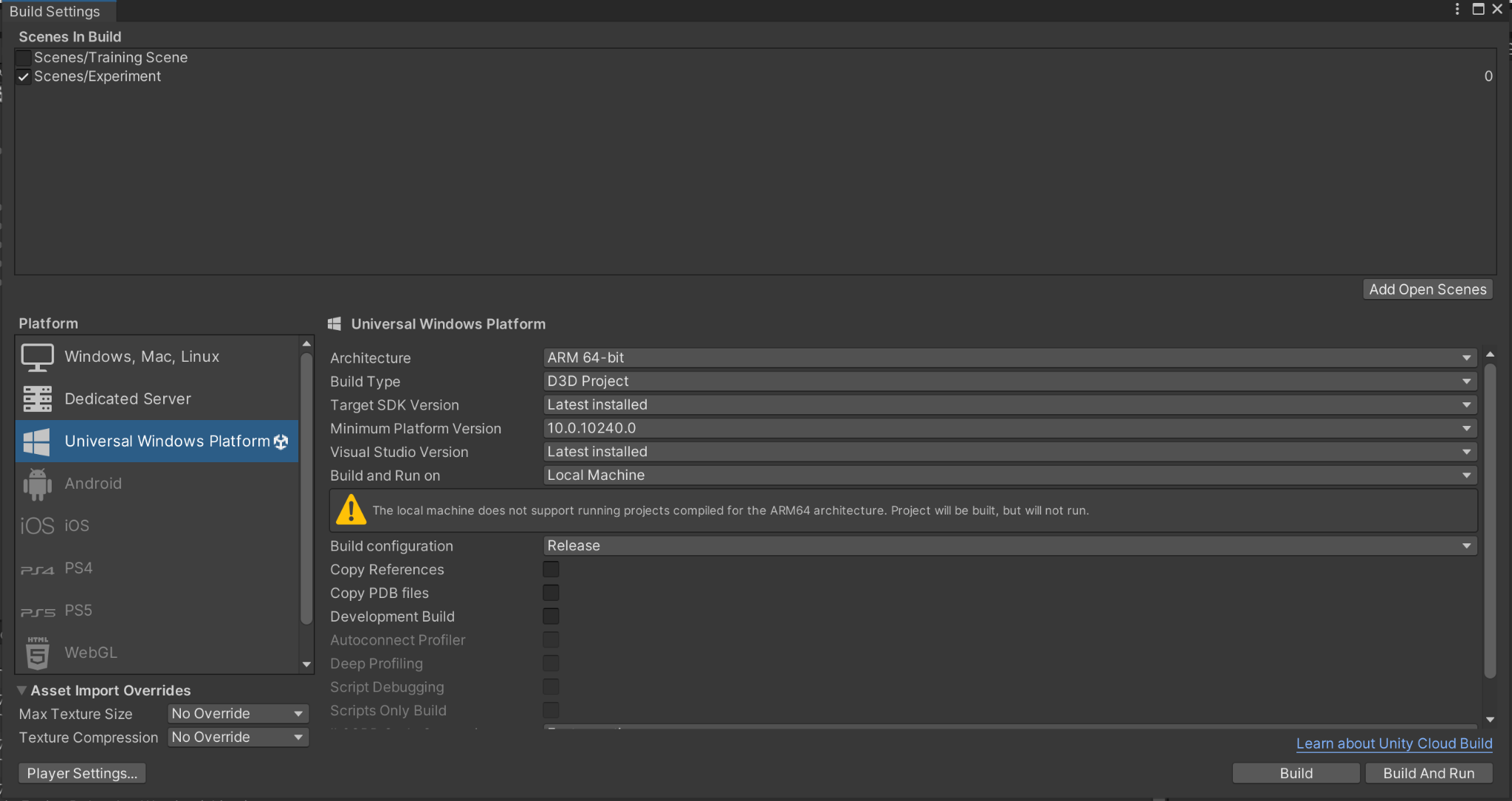
**Hardware:**

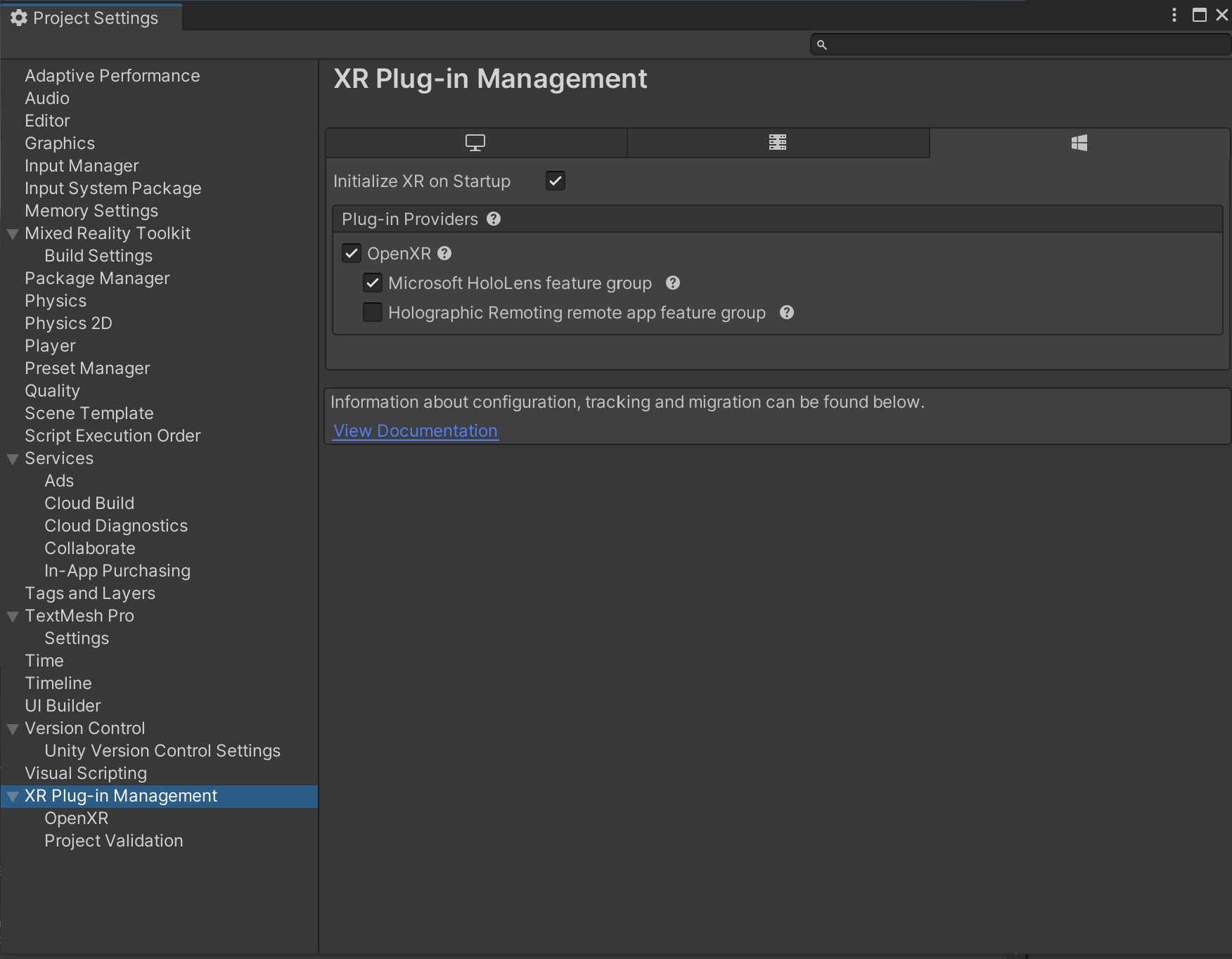
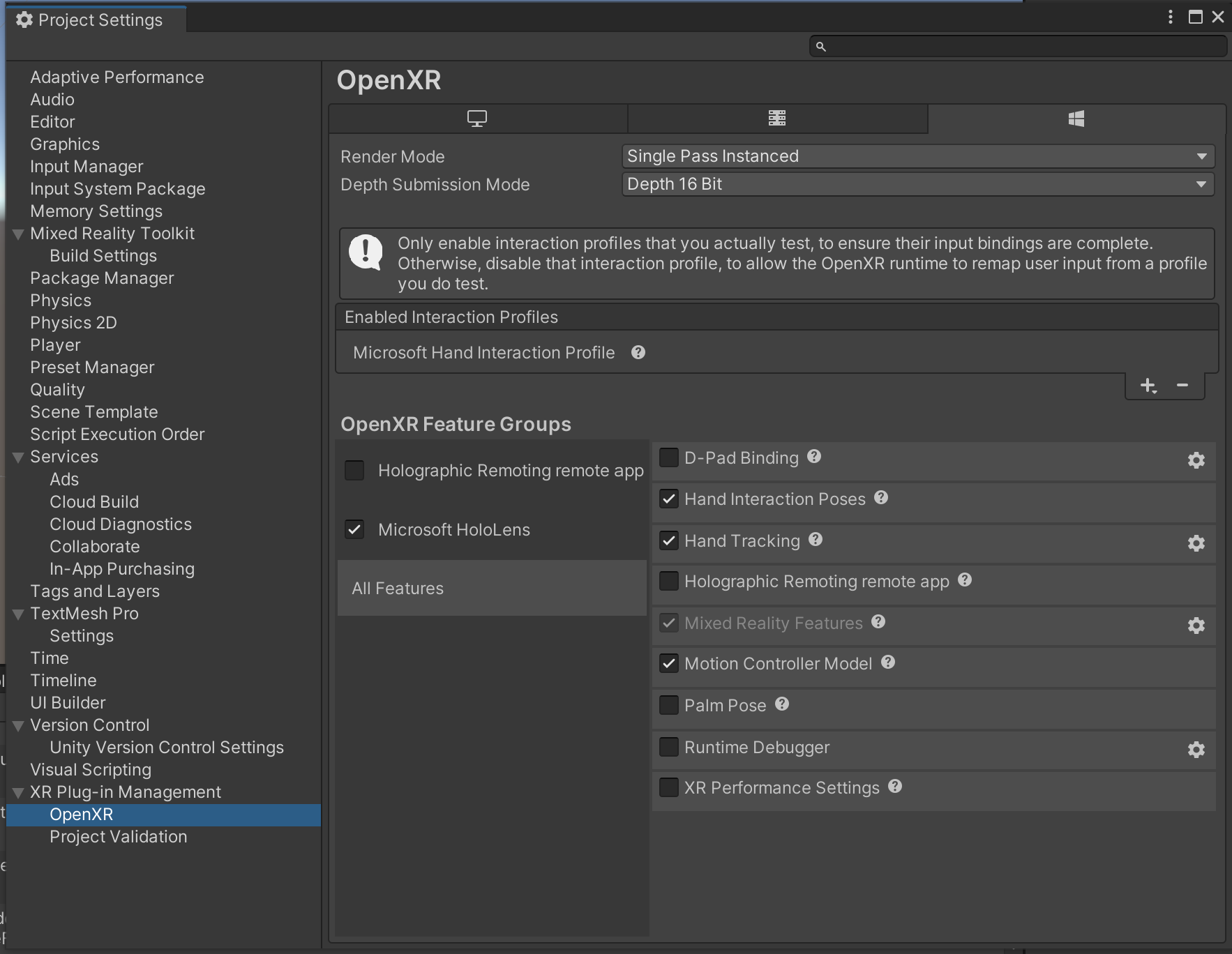
Hololens 2

**How to set up the environment**

1.Open Unity Hub, create a new project, using template Universal 3D

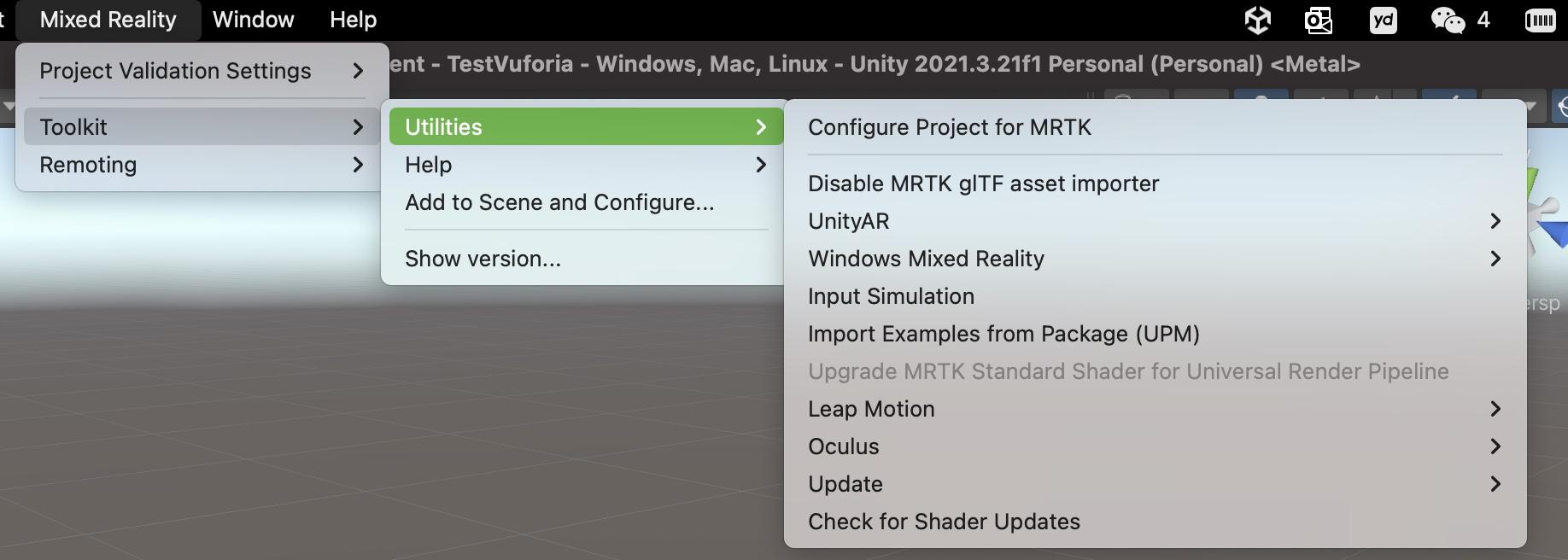
2.Using Mix Reality Feature Tool, import MRTK2 to this unity project****

**3.**Open the project, in Build settings, switch to UWP platform, and change the settings as following

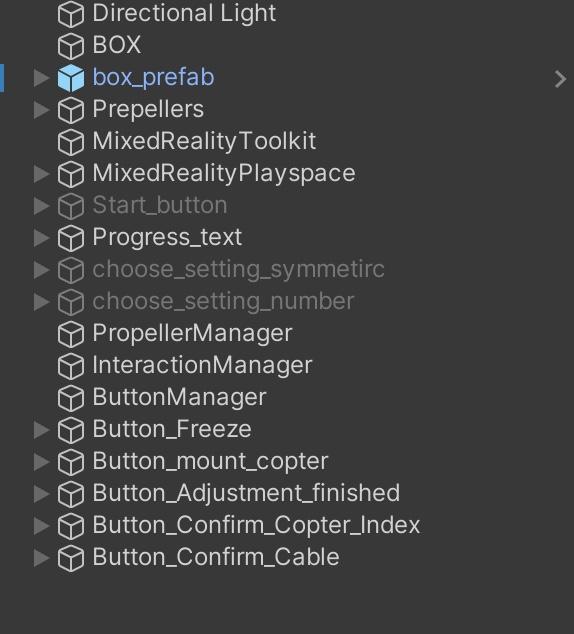
**4.**Go to project settings, in XR plugin management, change the settings as following. Note we are using the Microsoft Hand Interaction Profile, and make sure to tick Hand tracking.

5. After importing the MRTK2, you can find Mix Reality in the top bar. First click add to scene and configure. Then in Utilities, click configure project for MRTK. Follow the step to configure.

Note in MRTK toolkit in the hierarchy, you can edit the profiles. Remember to disable the Spatial awareness and Diagnostic.

****

6.The Hierarchy is lay out as following.

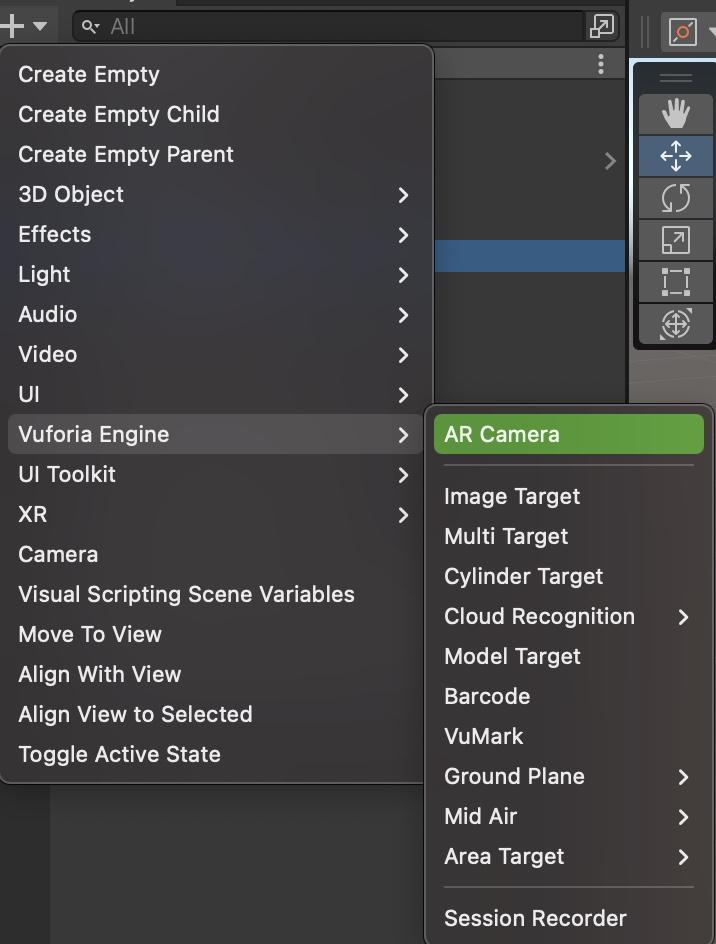


**How to use Vuforia**

1.Import Vuforia Engine

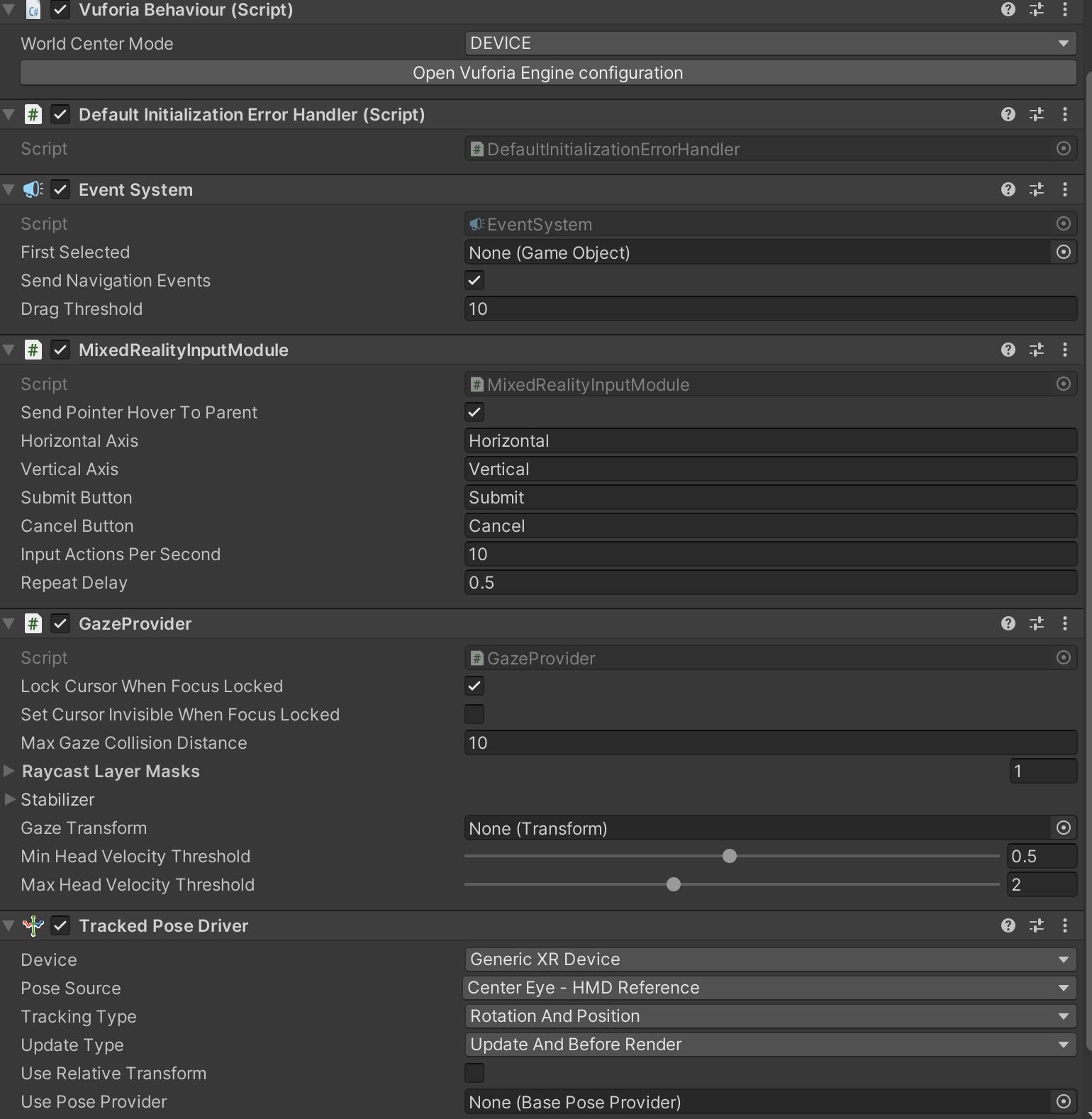
<https://www.youtube.com/watch?v=RMOMTyfECTk&list=PLhHZ6CAe8dA3JiG62bwWb3NmCROVkoZ2S>

Follow this 4 min video to import.

2.Place the AR camera under MRTK playspace

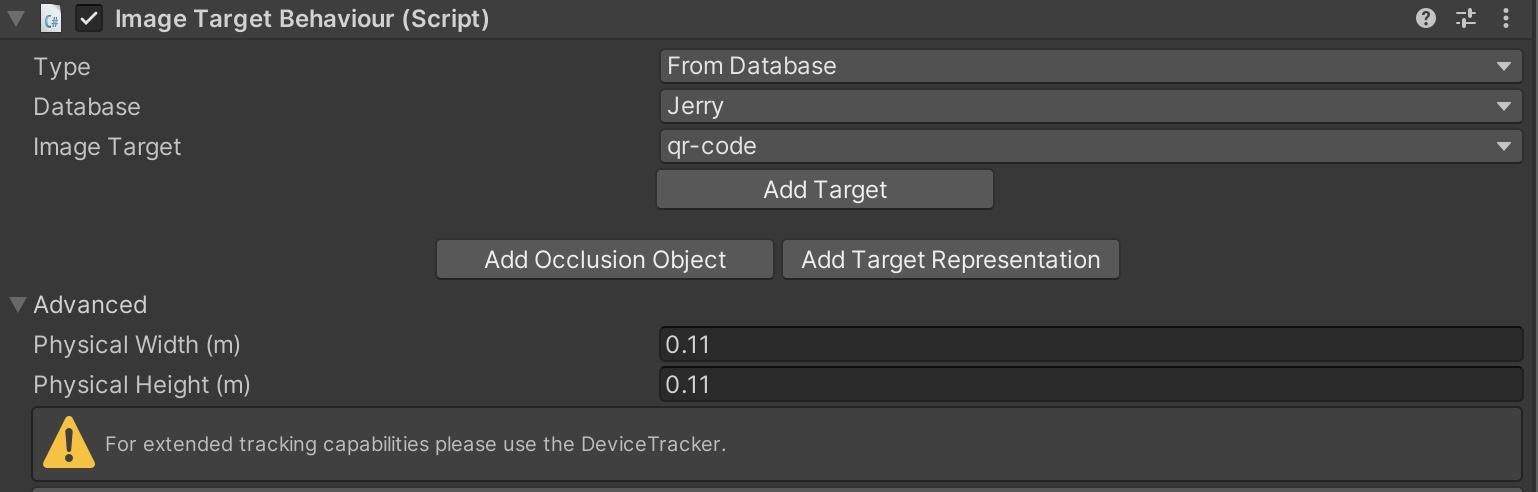


The inspector for AR camera

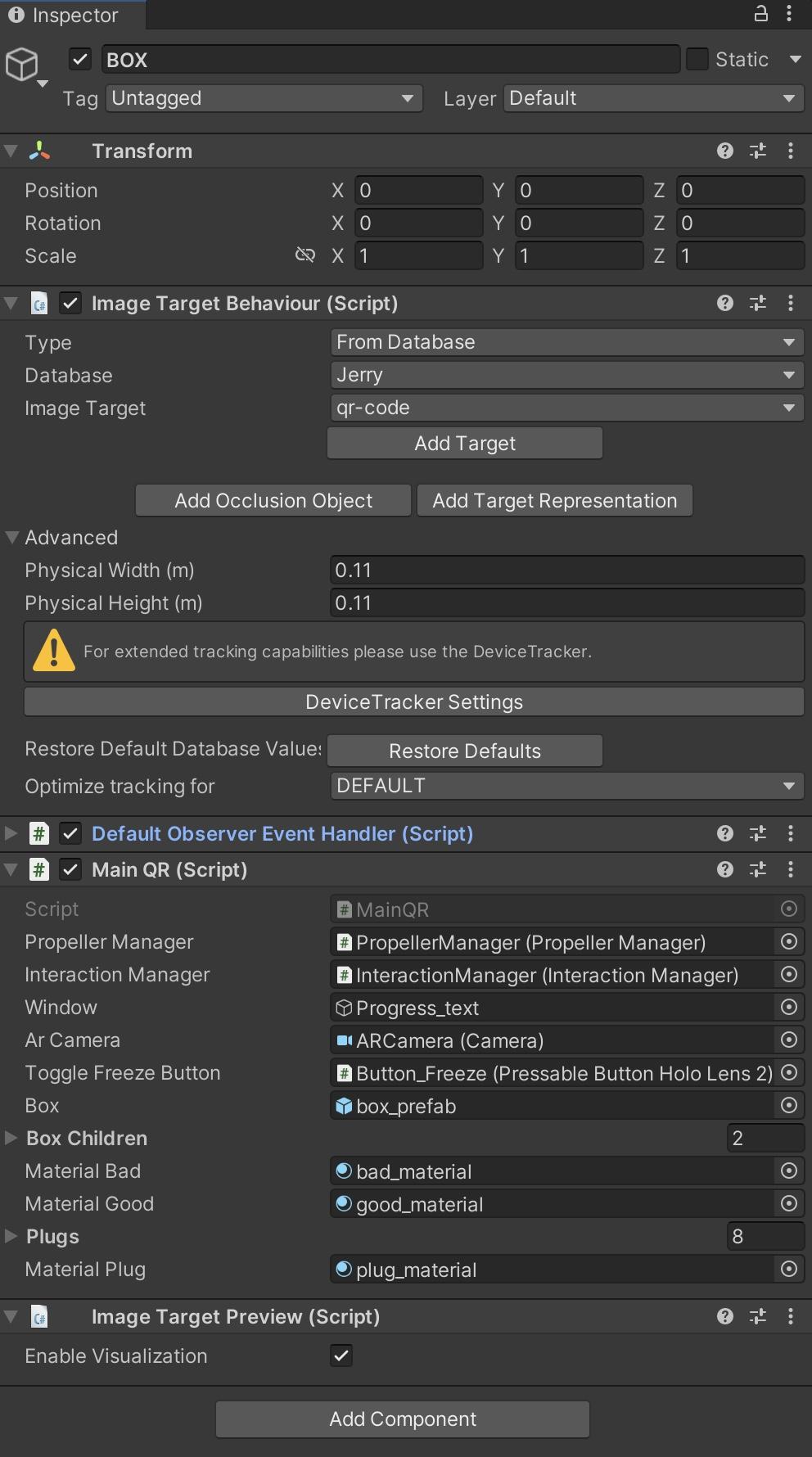


3.Use Vuforia’s image target

<https://www.youtube.com/watch?v=-bF0oxgtt6A&list=PLhHZ6CAe8dA3JiG62bwWb3NmCROVkoZ2S&index=2>

The image target is configured under BOX in the hierarchy.  
You can create your own database for the big QR code on the box and for the propellers' small QR code.

Important: the physical width and height can greatly affect the performance of tracking.



The custom script attached to BOX is MainQR.cs

You should drag the corresponding gameObject or prefabs to the window.

PropellerManager, InteractionManager, ARCamera, Button\_Freeze and box\_prefab(box’s CAD model) are already in the scene.

For Window, you should drag the TextMeshPro under Progress\_text.

The used materials are under folder CAD in the Assests.

*//this class is for handling the big/main QR code on the box. We use it to track the position of the real box. Around this coordinate,*

*//we place the hologram of the propellers. Due to the fact that when people move their head the hologram of the box will move, we can to freeze the box's hologram. This is done by the freeze button in the scene.*

*//Use can either use unfreeze the hologram all the time, so it will keep tracking the QR code, but if not tracked, no hologram will be shown. Or*

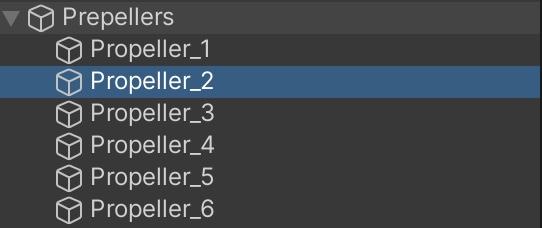
*// they can freeze the hologram, so that the hologram will stay in the last position when the QR code is tracked.*

*//Due to Vuforia's capability, we might manually adjust the hologram's position in the code.*

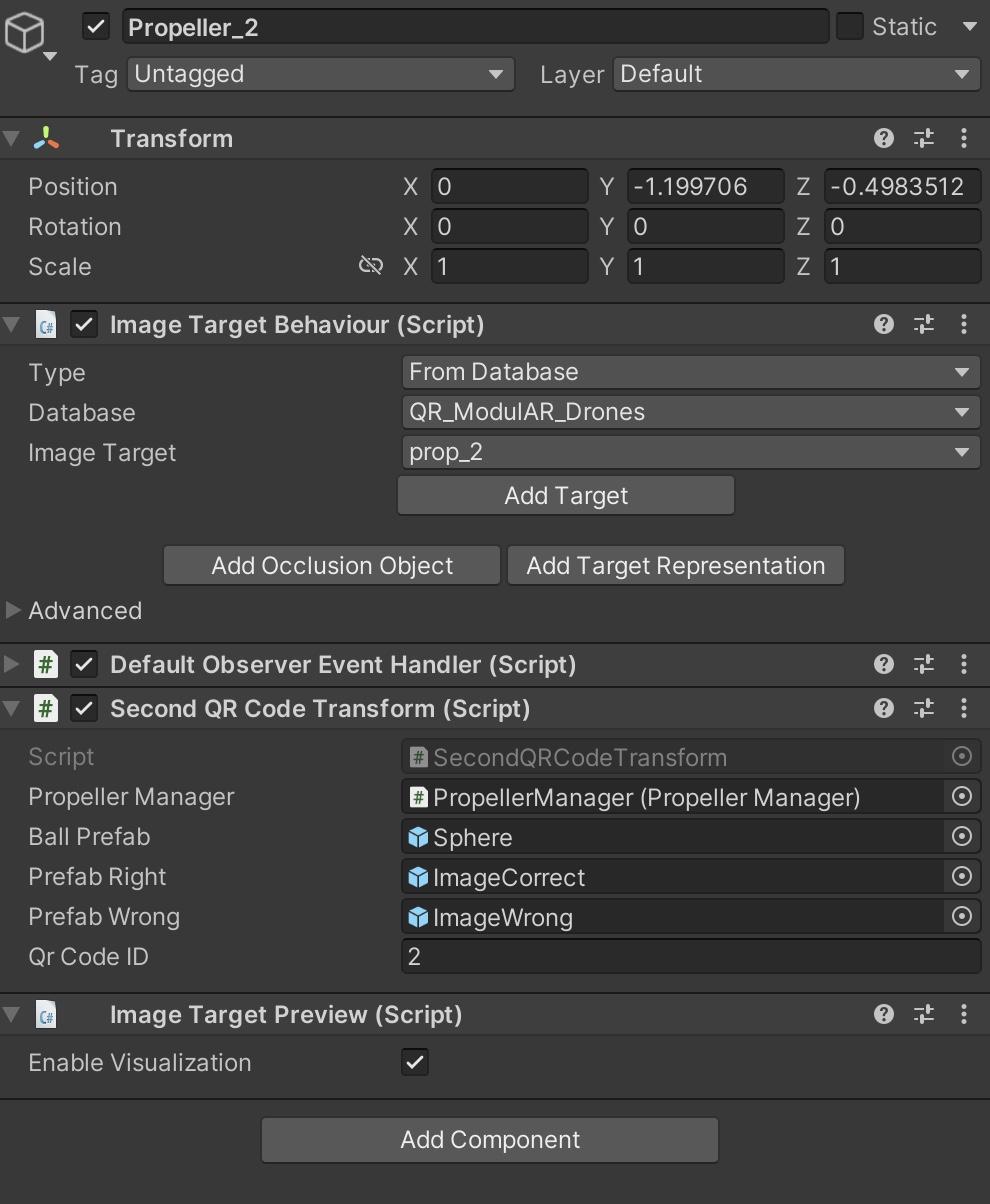
*//We also use this code to manage the color of box's hologram. When the box's hologram is frozen, the box will turn green. If not frozen, turn purple*

*//We also change the plug's color when needed.(to show user which plug to use)*

4. Apply image target to the 6 propeller game objects. Each of them should have their own image target



For each of these propellers, we attach the SecondQRCodeTransform.cs script as a component.



Reference to propellerManager.

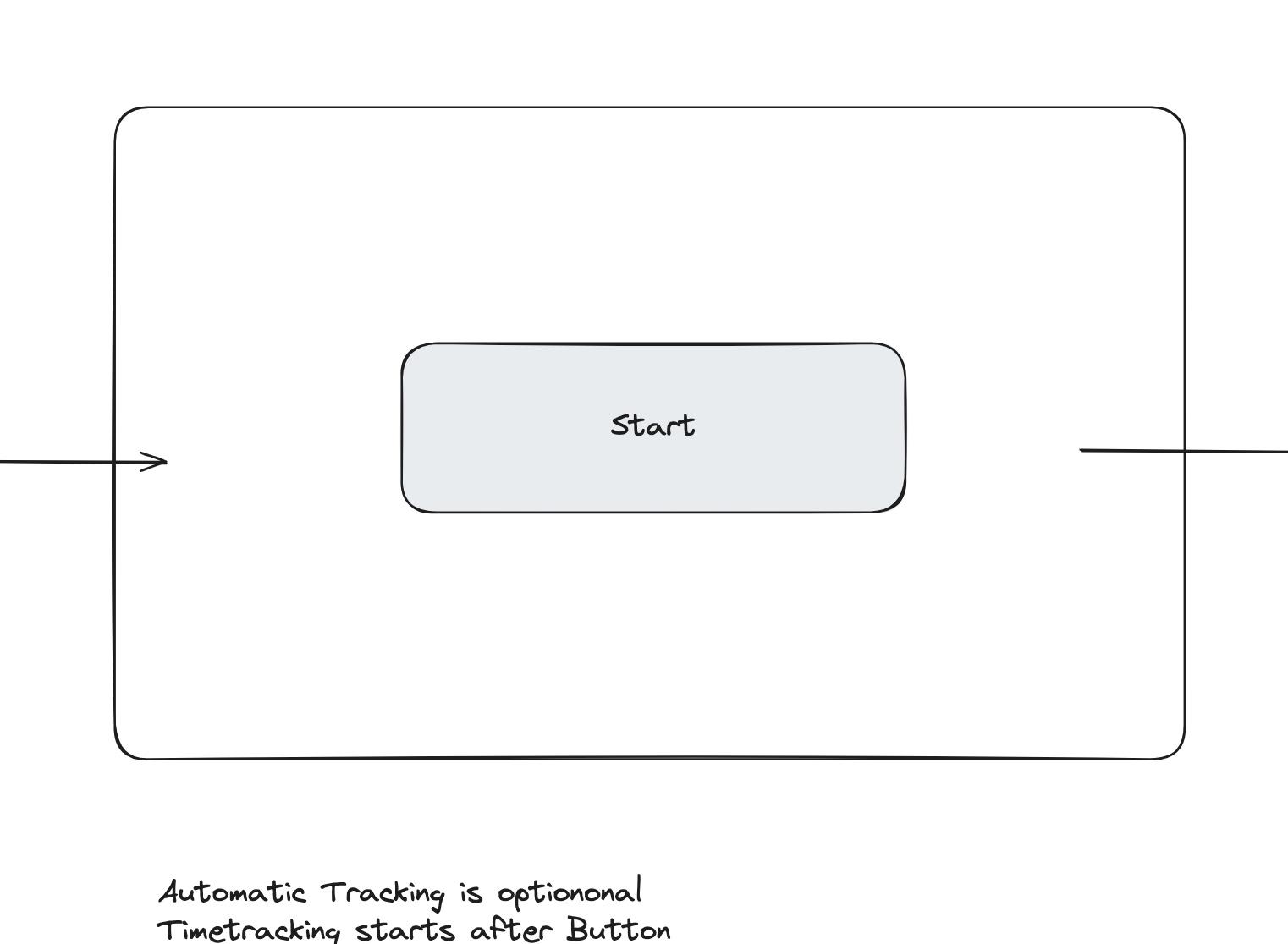
The sphere is in the CAD folder under Asset, which is the ball for indicating if the propeller is aligned with the propeller hologram’s position.

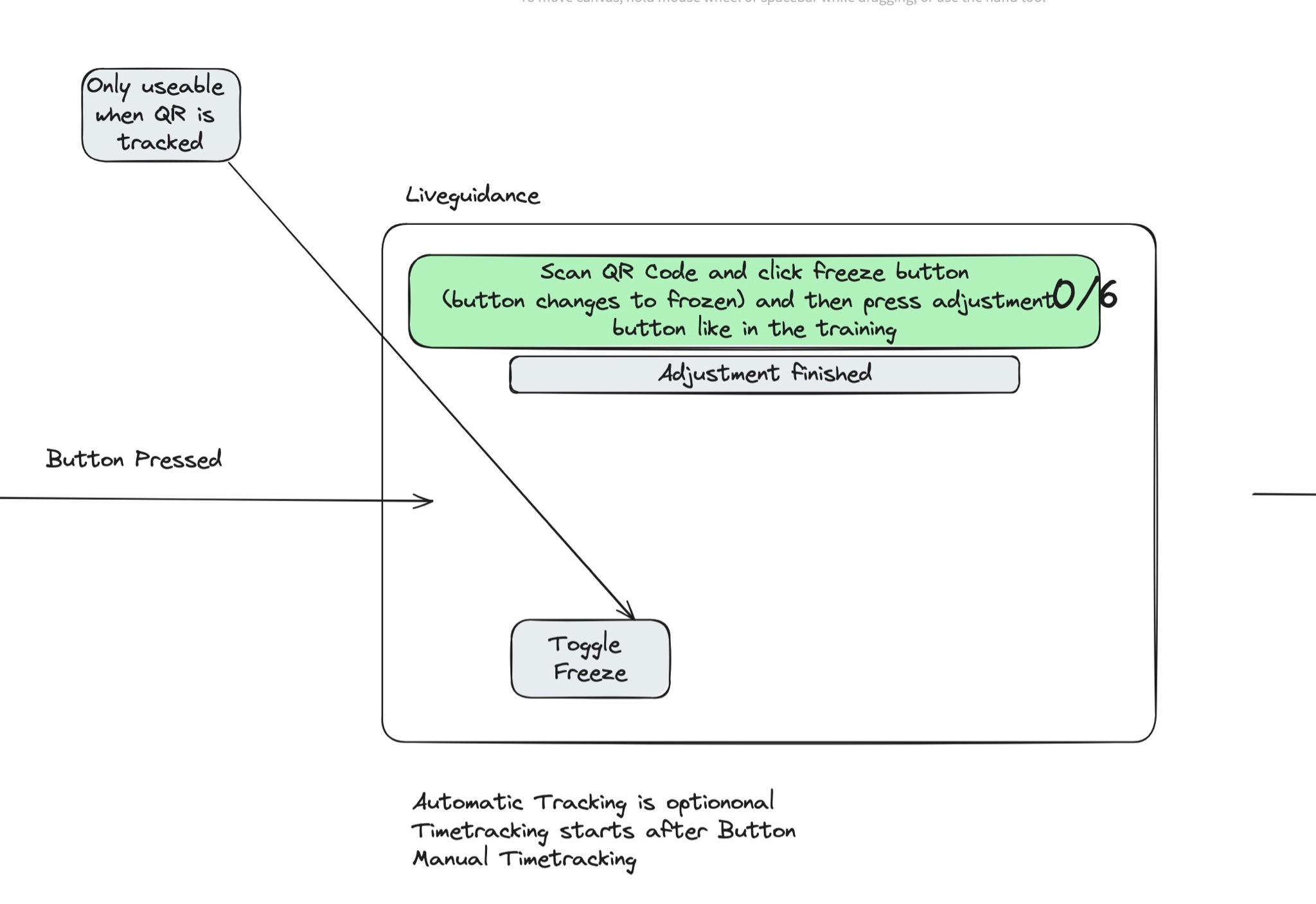
The Image Correct and Wrong can be found in the Images folder under Assets. They will be shown when users are looking for a certain number index of the propeller.

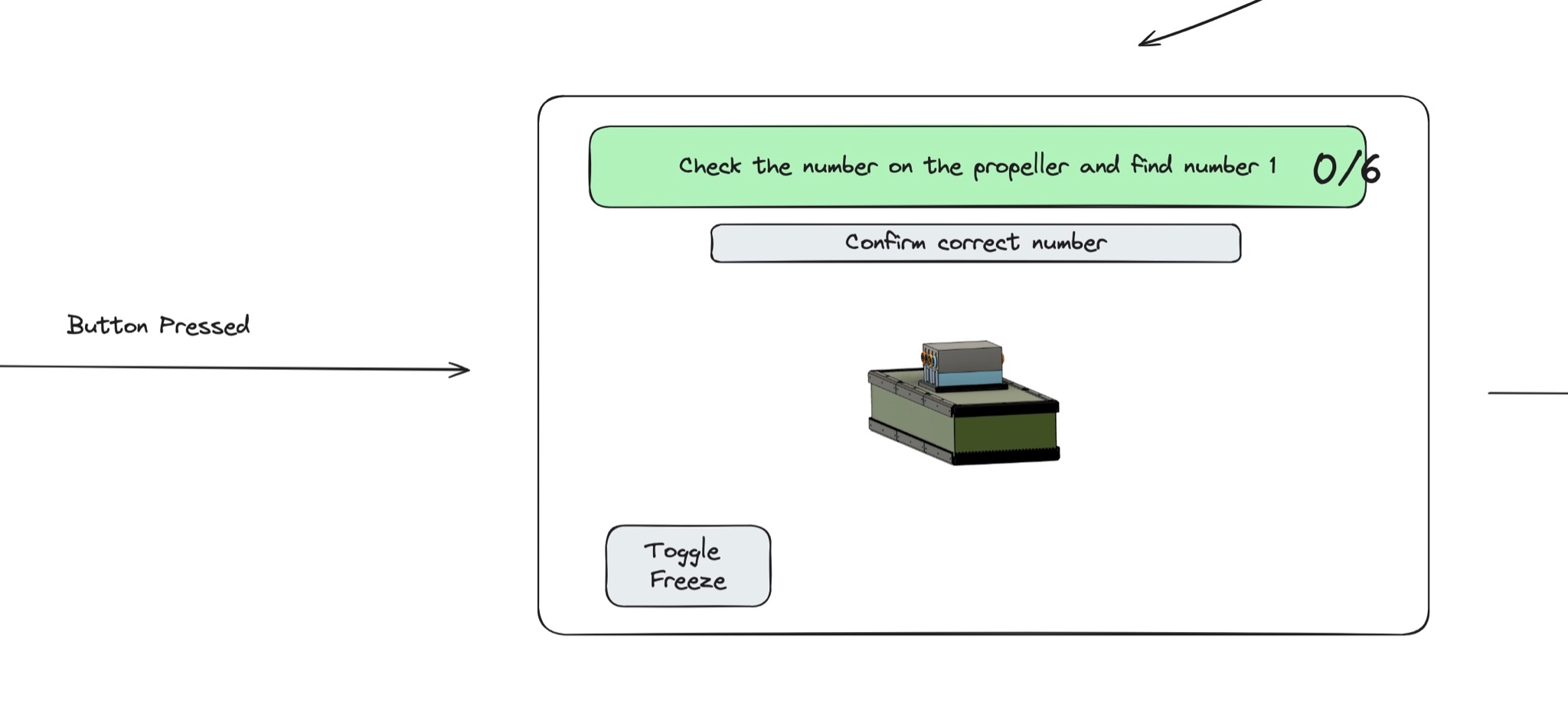
QR Code ID is for labelling different propellers

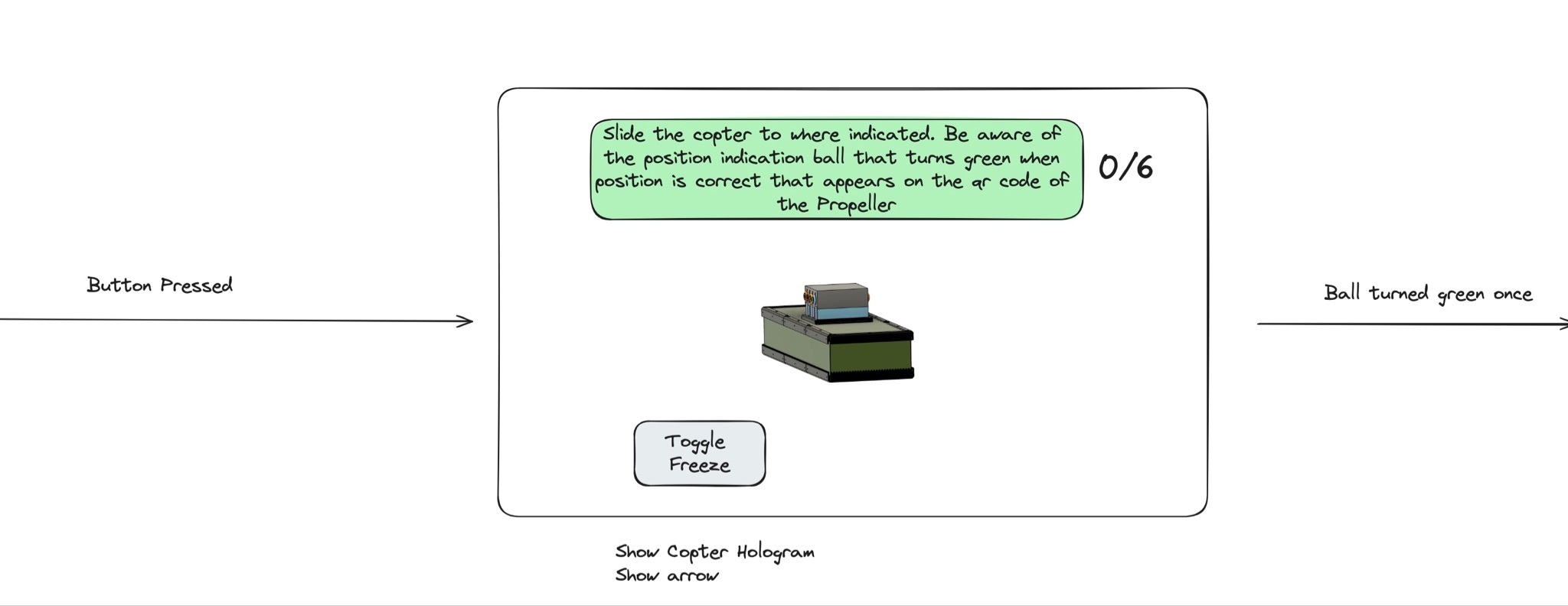
**UX (User experience)**

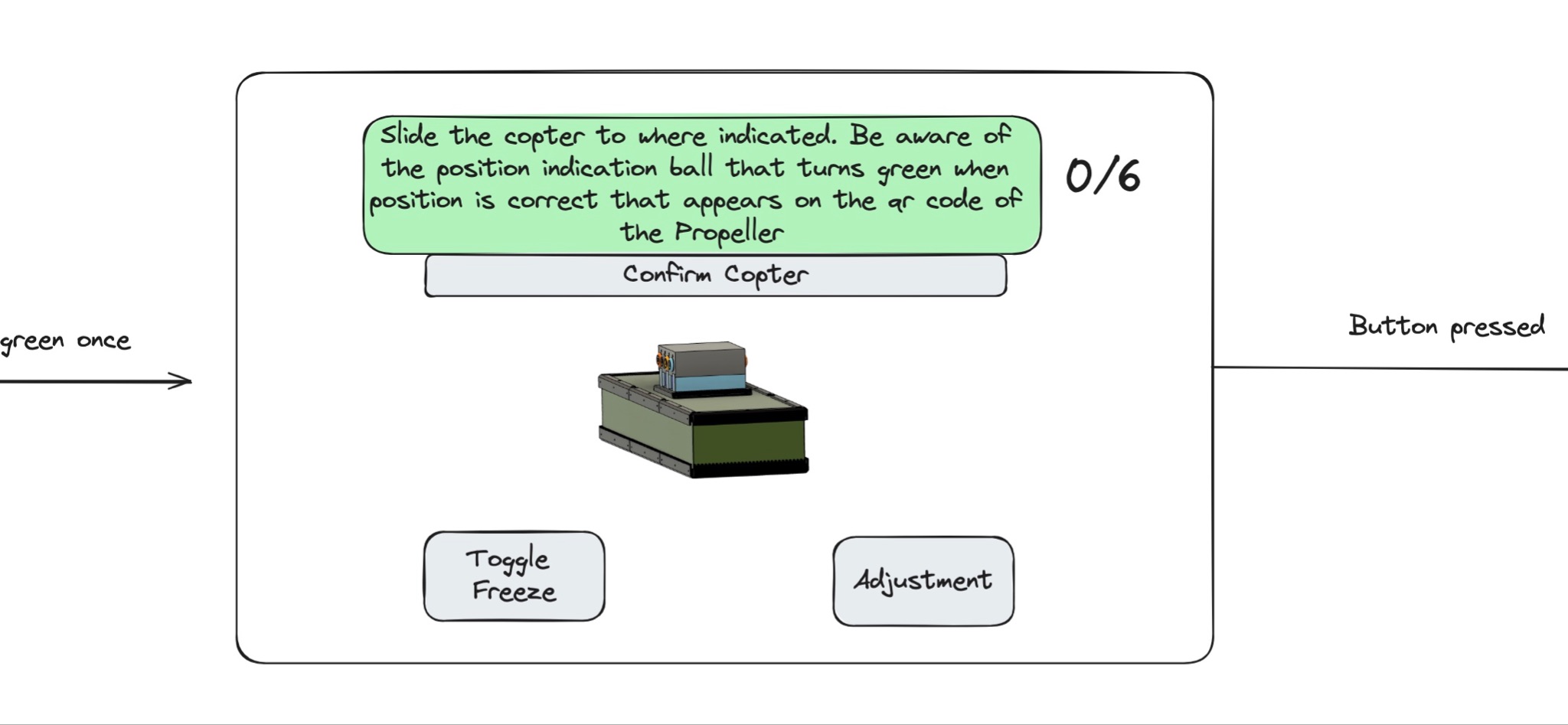
First before the user wears hololens, you should use the dialog to choose the configuration. And hand it to user.

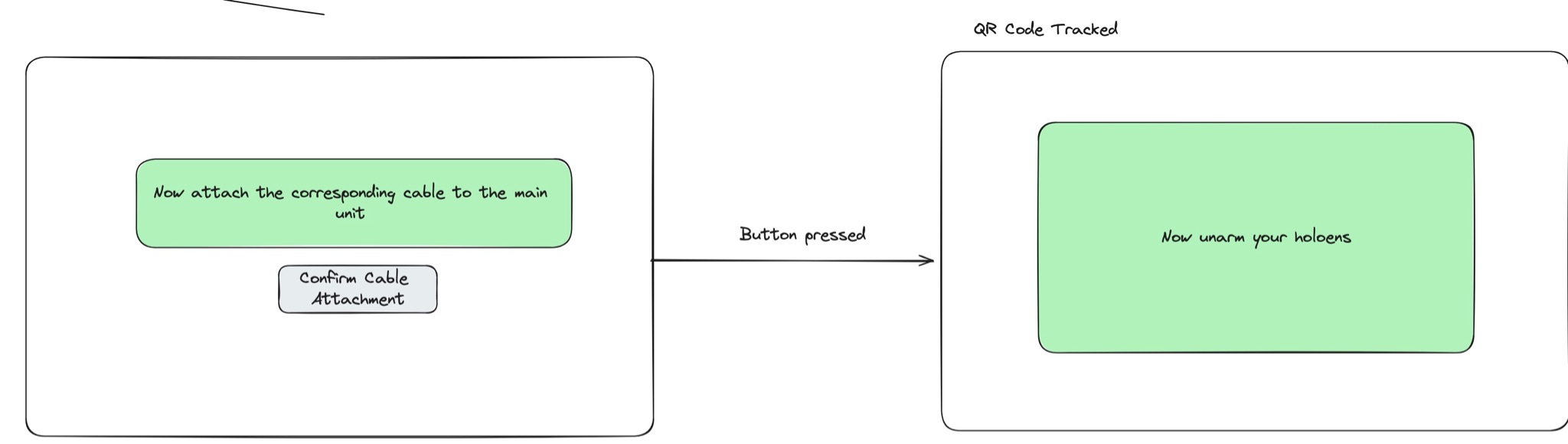






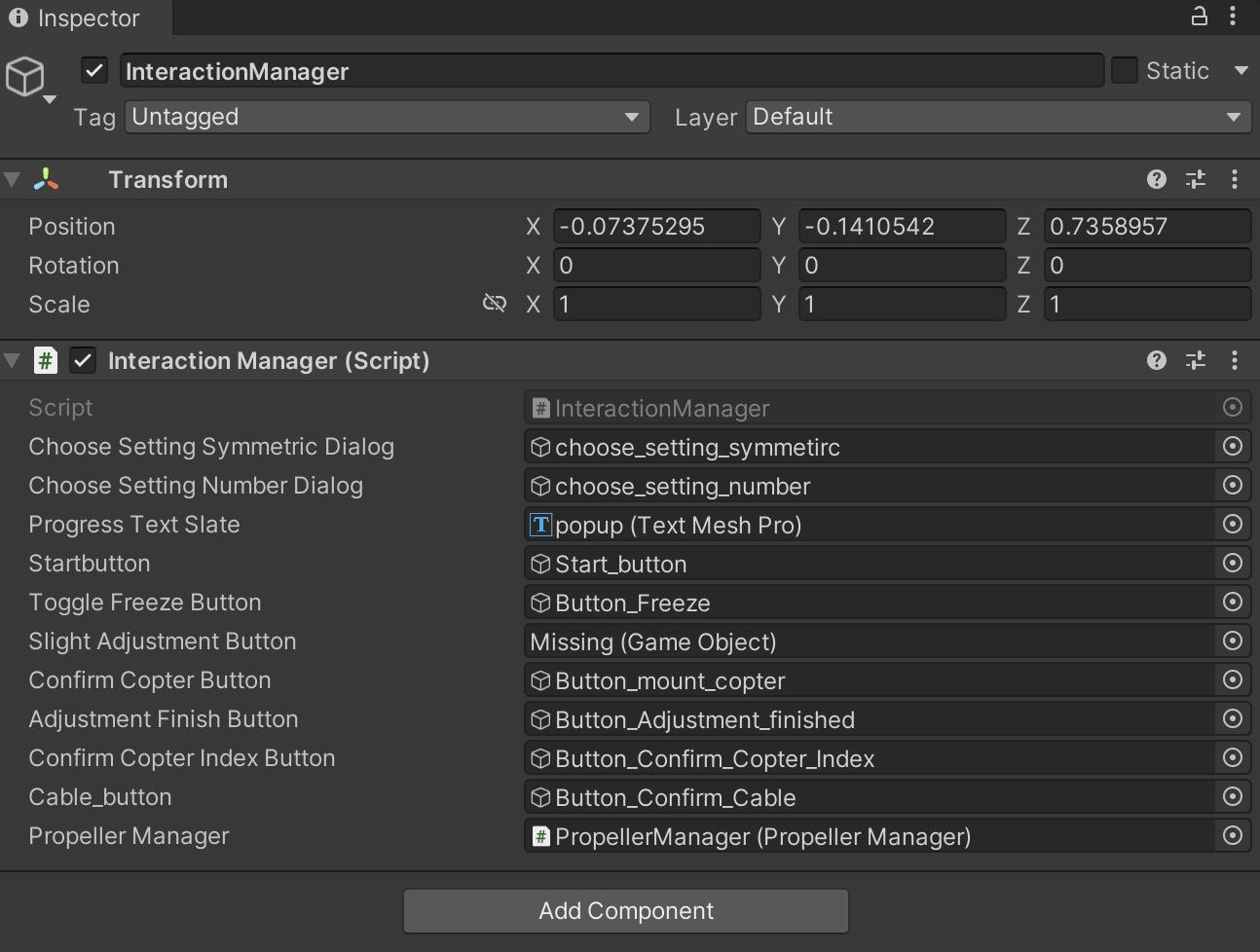






This logic is handled by the InteractionManager in the hierarchy.

An InteractionManger.cs script is attached to it.



For the placement of propeller hologram and checking mechanism, they are all handled by Propeller Manger in the hierarchy with **PropellerManager.cs** as its component.

The introduction of its methods is as follows.

### **1. Initialization and Configuration**

* **Initialization:** It initialises propeller and arrow holograms based on predefined configurations.
* **Configurations:** There are four configurations for propeller placements: symmetric and asymmetric setups for 4 and 6 propellers. These configurations determine the positions and rotations of the propellers relative to a main QR code.

### **2. Setting Up Propellers**

* **Set Configuration:** Based on user input, it sets up the propeller configuration as either symmetric or asymmetric and determines the number of propellers to be displayed.
* **Instantiating Propellers:** Depending on the chosen configuration, it instantiates the propeller holograms and prepares them for display.

### **3. Updating and Displaying Propellers**

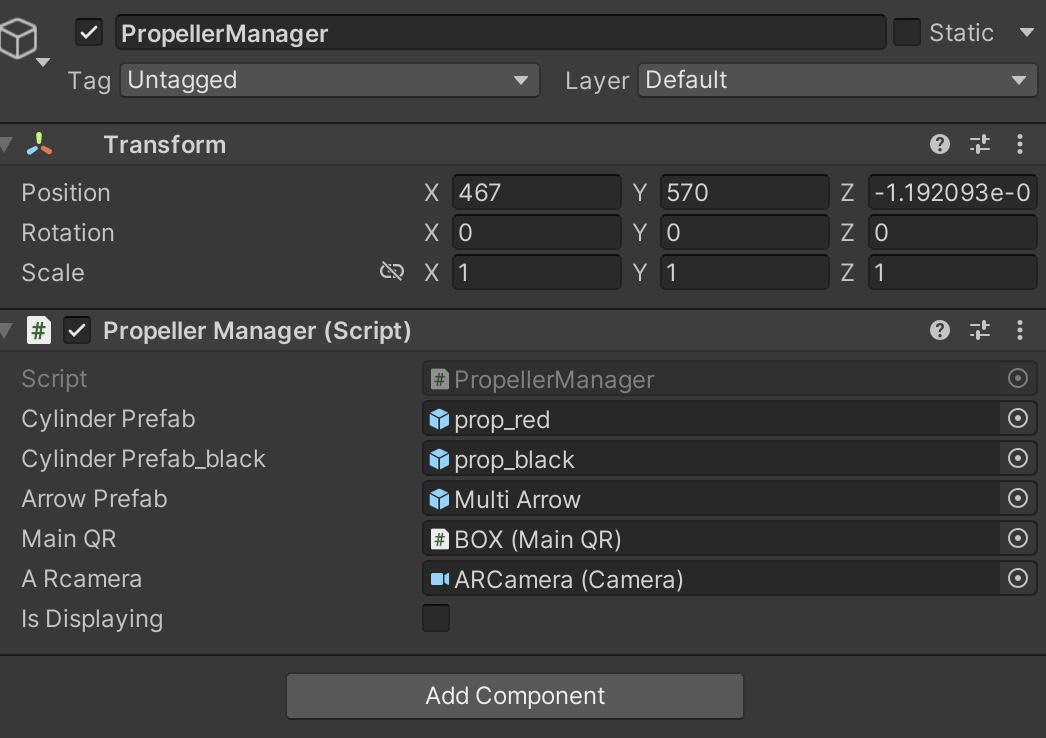
* **Update Cylinder Positions:** This method updates the position and rotation of the propellers based on the main QR code's position and orientation. It handles whether the propellers should be active (visible) or inactive (invisible).
* **Activate Next Copter:** It activates the next propeller hologram to be mounted, updates its position, and shows an arrow to help users locate it.

### **4. Handling User Interaction**

* **Finish Mounting Step:** Once a propeller is mounted, this method deactivates the mounting state and updates the display to show a tick or cross based on the correct QR code being scanned.
* **Check Second QR Code Position:** It checks whether a small QR code (associated with a specific propeller) is correctly positioned relative to the hologram. It provides visual feedback (color change) to indicate the accuracy of the position.

### **5. Visual Aids**

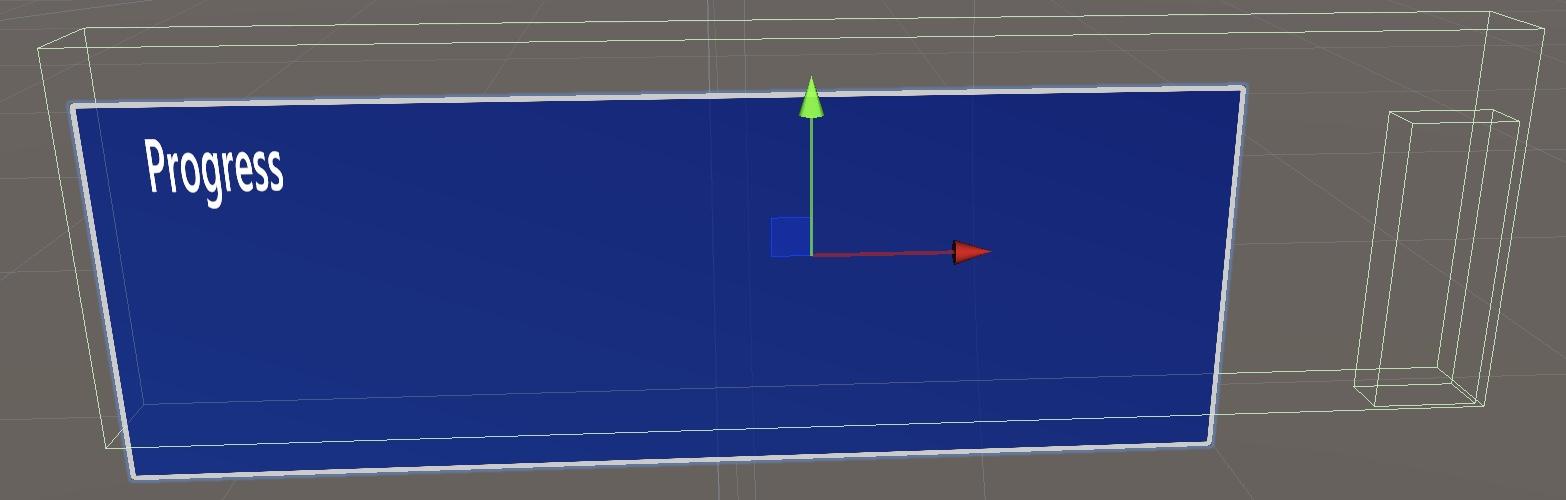
* **Update Arrow Transform:** An arrow is displayed to help users find the correct position to slide in the propeller. The arrow's position and rotation are adjusted based on the location of the propeller relative to the main QR code.
* **Change Plug Color:** It updates the color of the plug (in the hologram) to indicate the correct location for inserting a propeller.



You can find prop\_red and black under CAD folder. MultiArrow prefab under Animation. It also references to the BOX.

**UI**

Progress text

The progress window is designed to locate 20cm above the QR code, and it will rotate to face the user

Its text will be updated in the InteractionManager.cs script, and its location will be handled by MainQR.cs.

To be specific,

void UpdateWindowPosition()

{

*// Set the window's position 20cm above the QR code's original position*

Vector3 windowPosition = lastKnownWindowPosition + new Vector3(0, 0.2f, 0);

window.transform.position = windowPosition;

*// Make the window face the user*

Vector3 directionToCamera = arCamera.transform.position - window.transform.position;

directionToCamera.y = 0; *// Lock the rotation in the y-axis to avoid tilting*

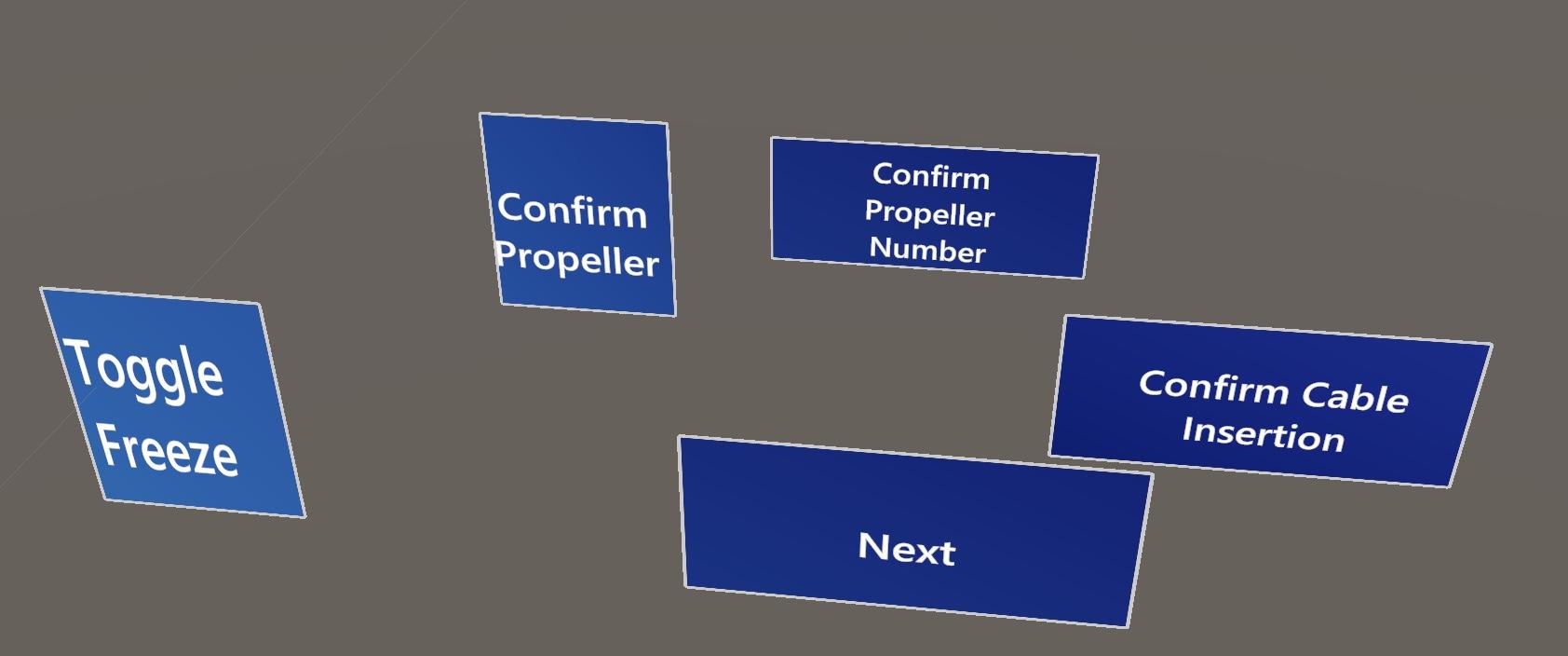
Quaternion targetRotation = Quaternion.LookRotation(-directionToCamera);

window.transform.rotation = targetRotation;

}

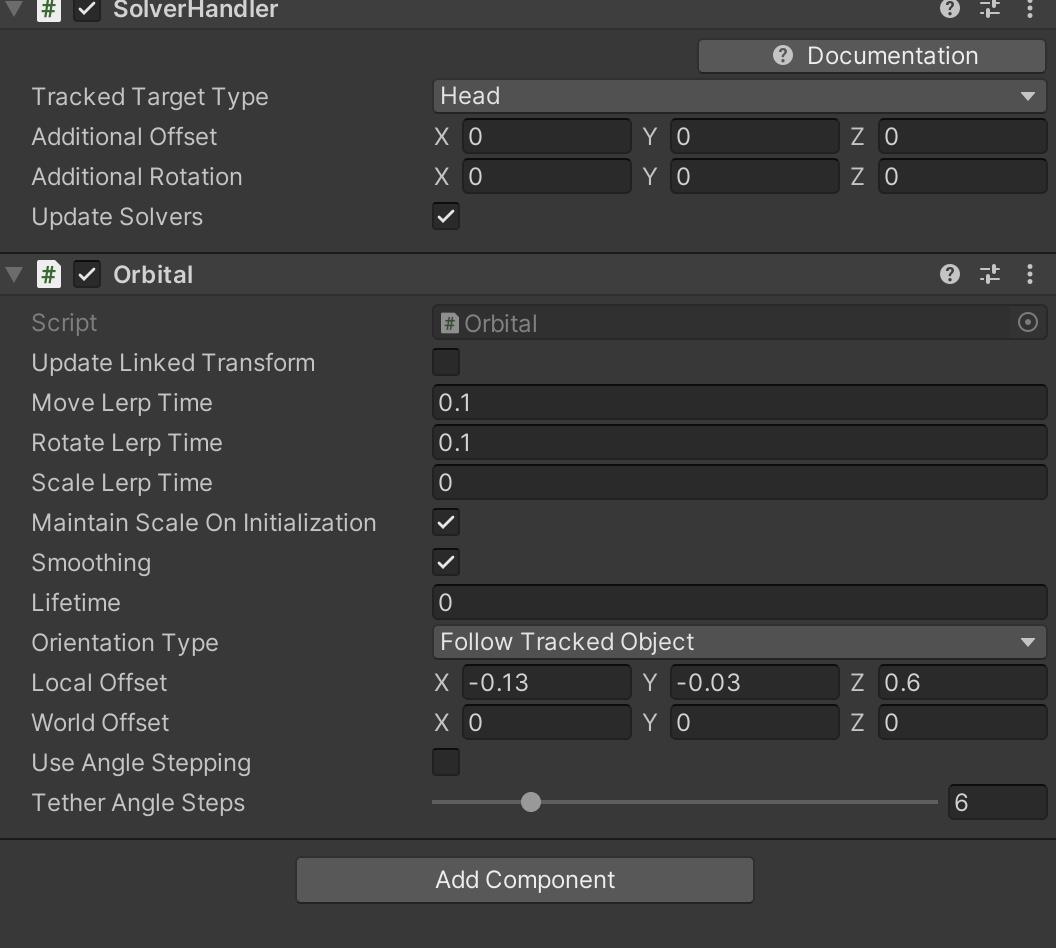
This function will make sure the progress window is always facing the user.

Buttons



Original prefab: Hololens2PressableButton from the MRTK2 package

The freeze button will be always in the user's view, and all the rest of the buttons’ appearance and disappearance is handled by InterationManager.



Solver Handler and Orbital are used so that they will be fixed to the user’s view in fixed distance.

Button\_freeze’s event is handled by MainQR.cs(to control to freeze the hologram) and ButtonManager(to update its text)

For the other buttons, their onClick event are handled by InteractionManager.cs. For example, for Button\_mount\_copter’s inspector

OnMountCopterButtonClicked will be called when clicking the button

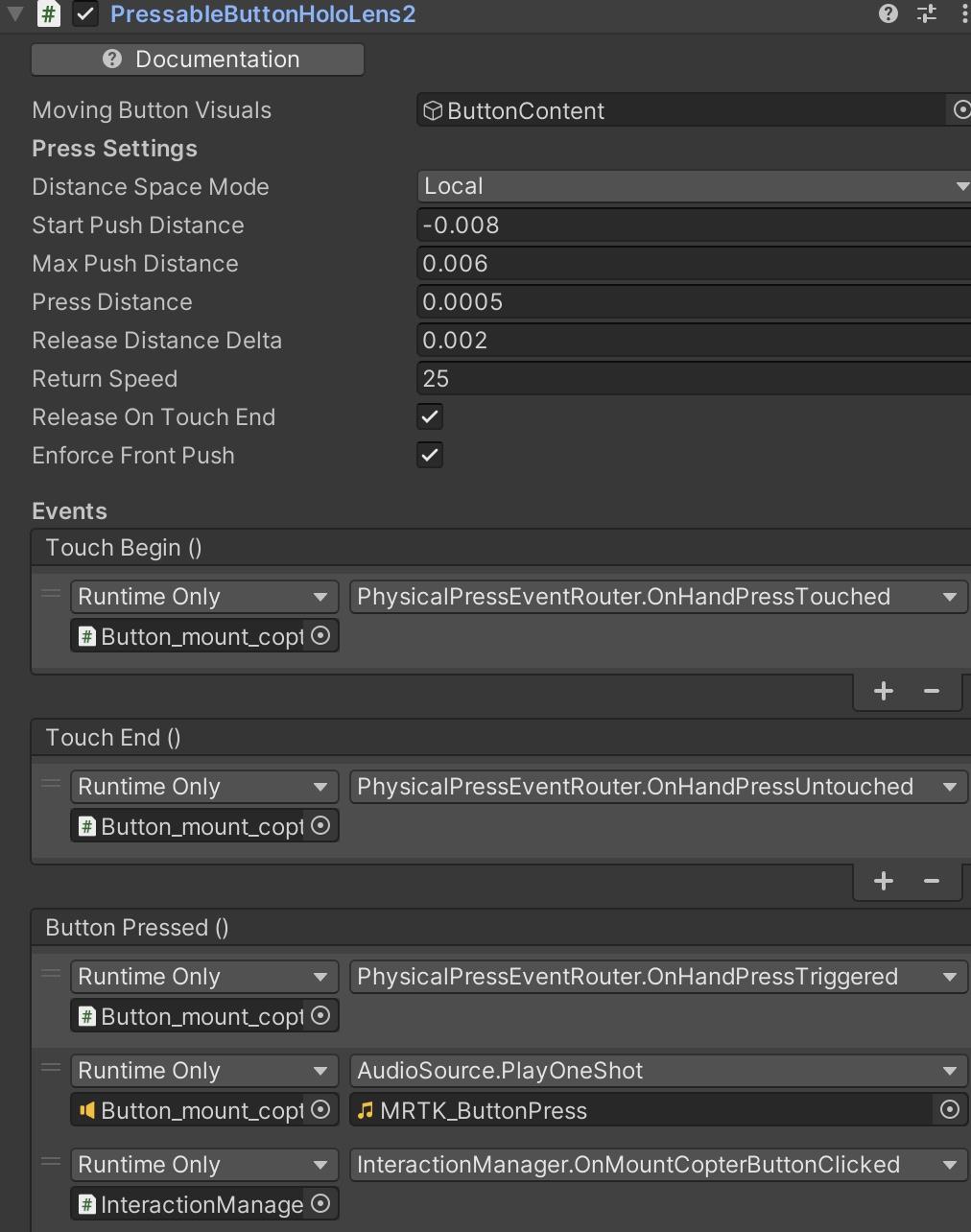


Table for function and button relationship

|  |  |
| --- | --- |
| Button\_mount\_copter | InteractionManger.OnMountCopterButtonClicked |
| Button\_adjustement\_finished | InteractionManger.OnAdjustmentConfirmed |
| Button\_confirm\_copter\_index | InteractionManger.OnConfirmNumber |
| Button\_confirm\_cable | InteractionManger.OnCableMounted |

**Further explanation on codes.**

**For MainQR.cs**

* Vuforia Integration: The class uses Vuforia’s ImageTargetBehaviour to track the main QR code on the box. This tracking information is used to position the holographic box and propellers in the AR environment.
* Position Adjustment: The QR code's position is adjusted slightly to account for tracking offsets, ensuring the hologram is displayed correctly relative to the physical box.
* Freeze/Unfreeze Mechanism: The class allows the user to freeze or unfreeze the hologram's position. When frozen, the hologram remains fixed in its last tracked position, and when unfrozen, it continues to update according to the QR code's position.

User Interaction:

* Toggle Freeze: The user can toggle the freeze state of the hologram using a button. This affects whether the hologram follows the QR code or stays fixed in place.
* Window Positioning: The class manages a guidance window (a UI element or message) that appears above the QR code and always faces the user. This window is positioned and rotated to stay visible and informative as the user moves around the box.

Visual Feedback:

* Color Change: The holographic box changes color based on its freeze state—green when frozen (indicating a stable position) and purple when unfrozen (indicating active tracking).
* Plug Color Management: As the user progresses through the mounting process of the propellers, the corresponding plugs on the box change color to indicate where the next propeller should be connected. The previous plug's color is reset to the default when the next one is highlighted.

Communication with Other Components:

* PropellerManager Interaction: The class interacts with the PropellerManager to update the positions of the propellers around the box whenever the box’s position is updated. It ensures that the propellers are correctly positioned relative to the frozen or unfrozen state of the hologram.
* InteractionManager Reference: Although not explicitly used in the provided code, a reference to InteractionManager suggests potential interactions or coordination between user actions and the box's hologram behavior.

For **InteractionManger.cs**(the **time log** is also here)

1. Initialization:
   * Logging Setup: The app creates a log file to record timestamps and user actions throughout the interaction. The log file is saved with a unique timestamp in its name.
   * Dialog Initialization: The app starts by displaying a dialog where the user chooses between symmetric and asymmetric configurations.
2. User Interaction Workflow:
   * Symmetry and Number Selection: The user is prompted to select whether the propeller configuration is symmetric or asymmetric, and the number of propellers to be mounted. Based on these choices, the PropellerManager is configured.
   * Start Process: The user begins the process by pressing a start button. They are instructed to freeze/unfreeze the box's hologram to align it with the physical box. The app starts timing the overall process here.
   * Adjustment Confirmation: Once the box is aligned, the user confirms the adjustment and moves on to the propeller mounting steps.
3. Propeller Mounting Process:
   * Ask to Grab Propeller: The app prompts the user to find and grab the correct propeller based on its number. A timer starts to log how long the user takes to mount each propeller.
   * Confirm Propeller Selection: After selecting the correct propeller, the user confirms their choice. The app then activates the corresponding hologram to guide the user in mounting the propeller.
   * Mounting Confirmation: The user confirms that the propeller has been mounted, which logs the time taken for that propeller and advances to the next step. If all propellers are mounted, the process ends, and the overall time is logged.
4. Logging and Time Management:
   * Action Logging: Every significant user action (e.g., selecting a configuration, starting the process, mounting a propeller) is logged with a timestamp in UTC format.
   * Time Tracking: The app tracks both the overall time taken to complete the process and the time taken to mount each individual propeller. These times are logged to a file for later analysis.
5. Interaction with Other Components:
   * PropellerManager Communication: The InteractionManager interacts closely with the PropellerManager, which handles the display and positioning of the propeller holograms.
   * UI Elements: The app uses various UI elements, such as buttons and text prompts, to guide the user through the process and record their actions.

**How to publish the app**

Normally you should be able to build the app to HoloLens through cable or WIFI.

(remember to install the USB connectivity component for Visual Studio)

An easier way is through side load.

<http://lucyestela.com/dev/unity/sideloading-onto-your-hololens-2-through-the-device-portal/>