


AR Objects Relocation

Report Project Course
UniTn 2022/2023



EVELYN TURRI

 evelynturri/AR-Objects-Relocation

Introduction

AR Laboratory Tour is an AR project in which it is possible to see holograms in an AR environment through the use of Lenovo ThinkRealityA3 smart glasses [3]. The aim of the project is to relocate objects in an indoor space, which have already been saved by the user in previous sessions. The relocation of the objects has been implemented through the use of Snapdragon Spaces SDK [5], which can save and upload AR anchors in an offline mode without using any online dataset tools.

Overview

As it has been said in the introduction, the app aims to augment the real world by loading holograms saved in previous sessions. Snapdragon Spaces SDK has been crucial to make this work, in particular, it's spatial anchors' feature.

The main point of the project is the use of AR Anchors, thanks to which we save the desired object in a certain point, and at the next use of the glasses, the app is able to track the AR anchors and so relocate the objects. In the following subsections I will explain in more detail what is an AR Anchor, the Snapdragon Spaces SDK and the algorithm at the base of the project.

AR Anchor

Anchors are key elements for AR applications. The user can see both the real world and AR elements, for this it can be necessary to locate an object in a specific position with respect to the real world. Different happens in Virtual Reality (VR), where the user does not have any idea of the real world, and the objects can be placed at any point in the virtual world. So AR anchors are a very important bridge between the real and virtual world.

There are different types of AR Anchors and different SDKs that used this element, each of them has a different way of loading and tracking the points in the real world. Some of them take advantage of Internet connection to save the coordinates of the anchors and some others use tracking algorithms, as with the SDK used in this project.

The main point is that anchors have to be tracked, and to do so software able to track and detect the world is needed. One of the main algorithms that has been developed with this aim is the SLAM algorithm and consequently the vSLAM.

Snapdragon Spaces SDK

Snapdragon Spaces SDK provides various features to help the users to create better XR experiences. This SDK makes use of OpenXR [2], which is an open, royalty-free standard for access to virtual reality and augmented reality platforms and devices.

As said before the main feature used in this project is the **Local Spatial Anchors** feature [4]. Local anchors enable XR content to be placed in spatial space at a given location, position, and orientation on the map generated through Positional Tracking and persist them locally over the session.

This feature is based on AR Foundation's AR Anchor Manager [7], but what makes this SDK so innovative for the spatial anchors is the ability to track these points and save them locally, without the use of any dataset or the use of Internet connection.

This is possible thanks to the use of algorithms such SLAM and Visual SLAM.

The anchor points are saved in a local store, and in the loading phase the smart glasses, thanks to the help of its stereo camera, are able to look at the environment and track the saved points.

Visual SLAM algorithm, stereo camera and deep learning techniques make the Snapdragon Spaces SDK perfect for dealing with AR anchors, and so the perfect tool for the goal of the project.

Before going in deep with the explanation of the project, there is a short theoretical overview of the Visual SLAM algorithm [1].

vSLAM

Simultaneous Localization and Mapping (SLAM) is an algorithm used to reconstruct a map of the real world and localize the object in that map at the same time. This method is widely used in robotics and nowadays also in AR applications, thanks to the help of computer vision and deep learning frameworks. For this last application is very important to have also visual feedback, it is for this reason that vSLAM has been introduced, which stands for Visual SLAM.

The SLAM system performs two tasks simultaneously: mapping and localization. The first framework for robot localization was filter-based, because it was based on Extended Kalman Filter (EKF). The concept of mapping and localization at the same time was introduced in 1992. The innovative point of vSLAM algorithm is the fact to detect and track feature points taken from the image environment over time.

Over the years, both SLAM and vSLAM have been improved by the use of deep learning, in particular for vSLAM by the use of scene segmentation and object detection. Even though nowadays the vSLAM systems are based on deep learning and computer vision techniques, the base of SLAM and vSLAM methods is the same.

The initial vSLAM architecture is based on three different modules: the initialization, the localization and the mapping. These are interconnected to achieve different objectives at the same time for the total final goal of the model. Localization and mapping correspond respectively to the estimation of the current pose of the camera in the environment and the partial observation of it, both are simultaneously performed and integrated together into a single model. Then, other two extra modules were added to the initial ones: the relocalization and the global map optimization. These have the objective to improve the accuracy and robustness of the process. Relocalization means that we do not have knowledge about the current pose of the camera. Due to this lack of information, the process has to be restarted through two different modalities: by using features points and landmarks detection or by closed-loop detection process. In particular, the feature points from the tracking step are used to define the keyframes, including frames with feature points different from the previous keyframes, while the overlapping frames are rejected, in order to decrease the number of keyframes and so the computational load too. The loop closure detection, instead, uses landmarks of the current frame and compares them with the previous keyframes, and this allows to have a geometrically consistent map. Finally, the global optimization module removes the accumulated estimation errors.

Design

Nowadays vSLAM algorithms have very good performances and follow these steps, to which some deep learning techniques can be added to have even better results:

1. Frame acquisition
2. Initialization : First mapping
3. Visual Odometry (VO) : Estimation the 3D camera pose and the 3D landmark position into the environment between consecutive frames.
4. State estimation : A global refinement of estimated poses from VO and loop closure detection.

5. Relocalization : The current location in case of tracking failure.
6. Global optimization : Preservation of the geometric consistency of the whole map.
7. Loop closure detection : Check if any places have been already mapped.
8. Mapping : Integration of the partial observations of the environment into a coherent model to propose a possible navigation path.

These steps are grouped in 4 parts: initialization, front-end, back-end and mapping. In the next paragraphs is explained the subdivision.

Initialization The initialization is needed to build the first map. In this phase, the estimation of the camera motion and the 3D structure of the environment are achieved too, in particular they are achieved without taking into account any global considerations and in a discontinuous manner.

Front-end The front-end part is composed of VO, where the feature points are extracted from successive frames. The feature points allow considering only the camera motion without taking into account the whole map, which means that it is only a local estimation that does not integrate the global vision of the environment. For this reason, there is a lack of global consistency in the mapping.

Back-end The back-end step improves the estimation through either relocalization, global optimization, or loop closure detection. In details:

- Relocalization : in case of tracking failure or in the presence of a place already mapped.
- Global optimization : Improves the mapping quality by adding new landmarks to the updated map or by deleting incorrect map points.
- Loop closure detection : the process of detecting whether a point has been already visited or not. It requires precise feature points to efficiently select keyframes.

Mapping The mapping step consists in managing keyframes and feature points to build a consistent global map automatically updated.

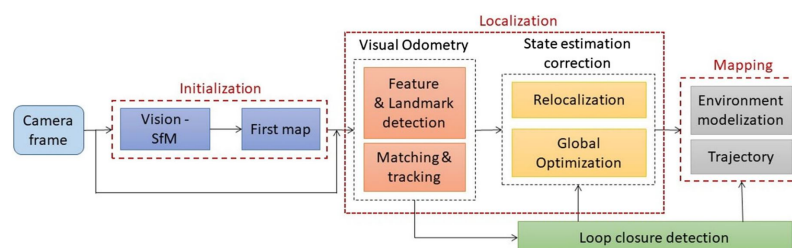


Figure 1: Diagram of the architecture of visual SLAM systems. [1]

Stereo camera and Visual SLAM

In this project it has been used the ThinkRealityA3 smart glass, which provides stereo vision, thanks to its double cameras. The stereo camera plays a key role in vSLAM algorithm, because the algorithm can be performed both with a monocular and a stereo camera, but with the first one, the results will be not as good as in the stereo case, because the depth can not be accurately estimated.

Project

The general project's idea is to be able to wear smart glasses to augment the real world and see objects, such as different types of holograms or notes, in the augmented world. In order to build such idea, the solution coming to my mind was to use AR anchors in order to track the main points in the world and so relocalize them, so that the objects will be loaded in the same places as the previous sessions. Each anchor has connected multiple AR objects, decided by the user, in order to have a cloud of objects around the anchor, as shown in Figure 2. To define this idea I used smart glasses and a specific AR app for the phone.

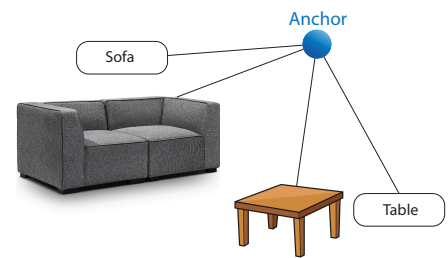


Figure 2: Example of an anchor with some objects connected to it.

Organization

The application is structured in two modalities: the Admin mode and the Guest mode. As soon as the user enters the application, he/she can see a panel where to choose its modality and so continue the experience in the application, as shown in Fig. 3.

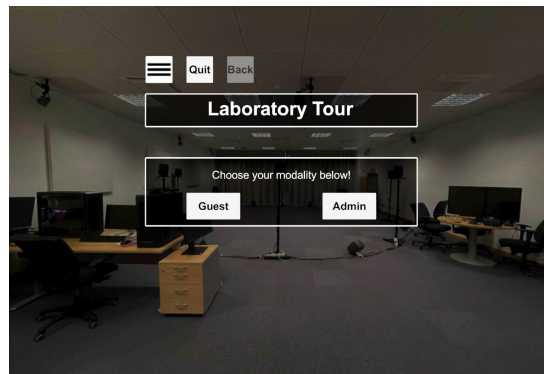


Figure 3: Graphical explanation of the 2 modalities.

In addition to the button of each mode's panel, there are three additional buttons (Fig.4), which are consistent in each scene. The **Quit** button has the function to close the application in the right way, saving all the data necessary for the next session; the **Back** button permits to pass to the previous scene (note that if the user is in the first scene, the **Back** button is disabled) and the hamburger menu button, which has the function to hide the first two buttons.



Figure 4

Admin

The admin mode is structured for the admin user, who decides how to augment the real world. The admin can create and save objects or completely clear the store and delete all the objects. The admin is able to enter in the Guest mode too, in order to see if everything is fine.

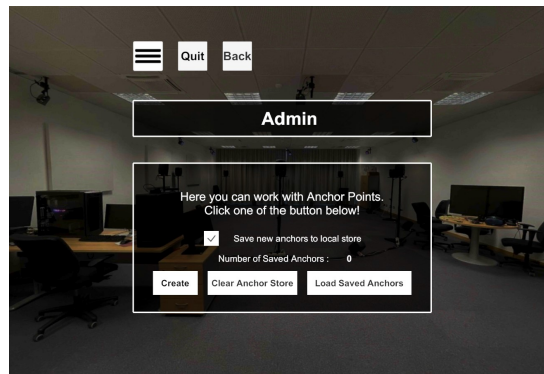


Figure 5: Admin mode

In the admin panel there are 3 buttons, as shown in Fig. 5: Create, Clear Anchor Store and Load Saved Anchors.

Create The create button brings the user in the Creation Mode, where the admin is able to create objects and descriptions, and where the application is able to add local anchors in order to save the objects and then relocalize them in the next sessions. Both objects and descriptions are considered as objects, and the application treats both of them in the same way, with some little differences:

- Holograms: By triggering the button Add `hologram`, the application will add the desired object in the specific position.
- Descriptions: By triggering the button Add `note`, the keyboard will be opened on the phone, and the user will write the specific note to add. If the input text of the note is not null, then a little panel is added at the specific point, to show the desired text.

In the creation mode, the user visualizes not only the gaze but also a transparent gizmo, which is more or less in the same position as the gaze. It is important because every object will be placed in that precise position, when the specific button is triggered.

It is significant to note that the admin has the possibility to add new objects also in different sessions, so he/she can enter and exit multiple times the application, without losing the work, and keep going to add new elements in other moments.

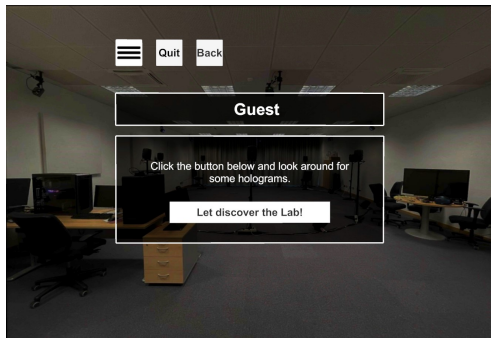
Clear Anchor Store With the button Clear `Anchor Store`, the admin can clear the entire store and begin to save new holograms and notes again.

Load Saved Anchors With the button Load `Saved Anchors`, the admin can enter the Load modality and actually see what the guest can see.

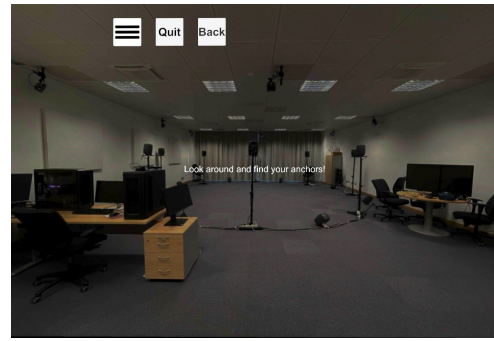
Guest

In the Guest mode, it is only possible to load all the objects created by the admin and see them in the augmented world. First, the panel guest user can see the panel in Fig. 6a, then by triggering the button on the panel, the load scene in Fig. 6b is loaded.

It is recommended to look around very well in the room, in order to make the glasses map the world and find the features to track the local anchors saved by the user. In this way, only once the smart glasses tracked the points, the holograms can be loaded to the user.



(a) Guest scene



(b) Loading scene

Figure 6: Guest mode.

Interaction

The interaction is completely managed by the gaze, indeed the application is almost entirely hands-free. It is possible to move between scenes, select buttons or move panels only with the use of the gaze. It is mandatory to use the hands with the phone only in the case of the descriptions' creation, where to add notes the keyboard is opened on the phone, and the user has to look at it and interact with it. For the guest mode instead, the application is perfectly working without the use of the hands. This fact is particularly useful in cases where it is not possible to use the hands or the phone.

Implementation

The project has been developed in `Unity` and all the code is written in `C#`. In the next subsections I will explain some implementation choices, made due to some problems or for better usage of the application.

Scenes' management

The project is structured in different scenes, in particular, there is a global scene where there are all the common tools to use over all the application, and then the other scenes are loaded as Additive scenes to the main one, through the command `LoadSceneMode.Additive` [9].

When the application starts, there is the Global Scene as the main scene of the application, which stays consistent over all the time, and then the Start Scene as additive scene. From now on, the other scenes are loaded one at a time, after the current additive scene has been unloaded.

In the project, I used different scenes instead of a single one. This choice has been made because it was the best idea to deal with two different modalities: guest and admin. While the reason why I used additive scenes, instead of only different single scenes at a time, was because some Snapdragon's features could be used only with additive mode, otherwise they were having problems, such as the Anchor Store being a different one for each scene. It is for this reason that the common features, which are consistent for each scene, they are added in the Global scene, such as: `AnchorManager`, `AnchorStore` by Snapdragon and the gaze for the interaction.

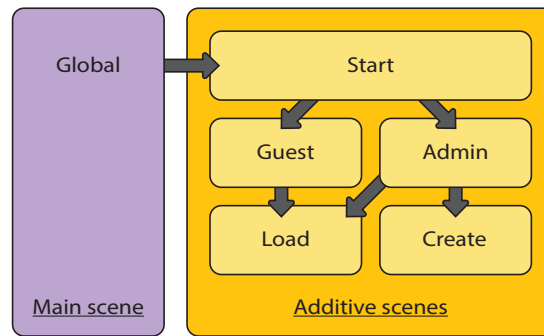


Figure 7: Diagram of the scenes' management.

As described in the previous section, there is also a Back button, which permits to go back to the previous scene. The current additive scene will be replaced with the desired scene to load, in this case the previous one.

Local Spatial Anchors

A Local Spatial Anchor is an AR Anchor, which is a pose in the physical environment, tracked by an XR device. The local anchors cannot be seen by the user, unless a GameObject is attached to the anchor.

In this project, the anchors are not used with a single prefab, but they are used with multiple objects, in order to visualize different holograms for a single local anchor, as shown in Fig. 2. To define this idea, it has been created a C# class, named `Anchor`, to save in a smart way all the data. The class has the following attributes:

Anchor	
◇ Name	Name of the anchor saved. The name is taken from the Anchors' store, once the AR Anchor is saved.
◇ Position	Position of the anchor, which will be updated at the start of the new session.
◇ Holograms	List of objects of type <code>Hologram</code> attached to the anchor.
◇ Descriptions	List of objects of type <code>Description</code> attached to the anchor.
◇ Tracked	Bool variable to say if an anchor has been tracked or not.
Transform	Transform attach to the anchor. At each session, once the corresponding transform is tracked the transform will be updated. It is defined when the AR Anchor is created for the first time, and it will be updated every time the anchor is tracked.

In the following Tables are described the objects of type `Hologram` and `Description`:

Hologram	
◇ Name	Name of the type of hologram will be instantiated.
◇ Position	Local position w.r.t. the anchor is attached.
◇ Rotation	Local rotation w.r.t. the anchor is attached.
Transform	Transform attach to the hologram, at each session, once the corresponding anchor is tracked, the transform will be updated.
HologramType	GameObject of type <code>Name</code> . It is defined when the Hologram is instantiated for the first time or when the application is launched in the next sessions.

Description	
◇ Text	Text of the description.
◇ Position	Local position w.r.t. the anchor is attached.
◇ Rotation	Local rotation w.r.t. the anchor is attached.

(The elements with \diamond are the variables saved for the next sessions, in subsection *Saving information* it is better explained how data are saved).

The local anchors are treated in a transparent way, this means that the application chooses where to create the anchor points. The user decides where to add a hologram or a description, while the application understands if in that position it has to create a new anchor or attach the object to an already existing anchor. The possibilities can be the following:

- If the store is empty, it means that there are no anchors and no objects saved, and so the first object created, either a hologram or a description, will create also a local anchor, which will be added in the store. The object created will be added to the list of the anchor.
- If in the store there are already some objects, this means that there are already some anchors created, and so, the desired object will be instantiated in that point. In this case, the application checks some constraints and decides what to do:
 1. If the object is far away less than a certain distance d to a specific anchor, then the object will be attached to that anchor;
 2. If the object is distant more than d to all the already existing anchors, then it will create a new anchor, and so it will be connected to this new anchor;
 3. If we are in case 2, but the already existing anchors are more than a certain number, decided a priori by the programmer, then it will be connected to the closest existing anchor, even if its distance is bigger than d .

Some important clarifications:

- d is the minimum distance between different anchors, it is possible to set it in the Unity project. In the actual project is set to 1 meter.
- As written in point 3. the application can save only a maximum number of anchors. This number can be set in the Unity project by the programmer. At the moment is set to 4. This choice has been made to lighten the application in the moment of tracking. The number can be set before the build of the application, and it can be modified depending on the size of the room, where the objects are visualized. An important advice is to not add too many local anchors, and so keep the max number low.

Once the anchor is created, all the other objects instantiated are attached to a specific anchor's transform, in particular they are instantiated in a specific position and with a specific rotation as children of the anchor's transform.

The local anchors are managed by a function named `OnAnchorChanged`, as described in the documentation [7]. In the case of this project, the function is used to update the anchors and their transform, but at the same time, whenever an anchor is updated, the gameobjects attached to its transform as children, are removed and instantiated again through the function `LoadObjects` (in the scripts `AnchorPointsCreate` and `AnchorPointsLoad`). Note that the objects of an anchor are loaded only once the anchor is tracked for the first time, otherwise the user can not visualize them. To do so, the variable `Tracked` in the class `Anchor` has been used, indeed this boolean variable is updated in the following way:

- Once the anchor is created the variable is set to `true`.
- When the app changes scene, for each anchor the variable is set to `false`.
- When the anchors are loaded in a scene, until the `OnAnchorChanged` function can not track the anchor, the variable is kept to `false`, once the anchor has been tracked, then the variable is set to `true`, and at that moment the objects related to the anchor are loaded for the first time.
- When the `Quit` button is triggered, for each anchor the variable is set to `false`.

Object's position and rotation

One of the main objectives of the project was to be able to see holograms and descriptions as the user created them. To reach this result it was fundamental to understand how to save and how to load the position and the rotation of the object.

For this point is crucial to take into consideration that every time the application is launched, the origin of the AR session is different, and for this reason, it is important to save the right position and rotation.

To solve this problem the position and the rotation saved were respectively the local position and rotation w.r.t. the anchor attached to the object. In this way the application, once it tracks the local anchors in the world, can load the objects in the local position w.r.t. their anchor's position, and with the right rotation. This was possible thanks to 4 important commands:

- `AnchorTransform.InverseTransformPoint(GlobalPosition)`
It computes the relative position of the object w.r.t. the anchor, which is the local position of the object in the coordinate space of the anchor;
- `Quaternion.Inverse(AnchorRotation) * GlobalRotation`
The first part of the multiplication computes the inverse rotation of the anchor, which is essentially a rotation that, when combined with the original rotation, results in no rotation or the identity quaternion (0, 0, 0, 1); while the second part of the multiplication calculates the world space rotation of the object. By multiplying them together, we obtain the relative rotation of the object with respect to the anchor, which is the local rotation of the object in the coordinate space of the anchor;
- `AnchorTransform.TransformPoint(LocalPosition)`
It computes the global position of the object, starting from the coordinates space of the anchor;
- `AnchorTransform.rotation * LocalRotation`
By multiplying the global rotation of the anchor with the local rotation of the object w.r.t. to the anchor, we obtain the global rotation of the anchor.

Saving information

The idea of the project was also to be able to save different objects for a single local anchor, in order to have multiple holograms in different locations attached to a single anchor point. In this way the application becomes much lighter and easier to use.

To do so, a smart way to save all the data necessary for the next sessions was needed, so I chose to transform the information into a JSON file [8].

All the data to save are grouped into two global variables:

- `_anchorsNames` : list of all the anchors' names.
- `_anchors` : list of all the `Anchor GameObjects` of the project.

When the application quits, both variables will be saved in the `PlayerPrefs` [10], which works as a dictionary. The application saves the variables in this way:

- `_anchorsNames` : it sets as key the string "AnchorsNames" and as value the variable transformed in JSON.
- `_anchor` : for each anchor in the variable sets as key the anchor's name, while as value the `Anchor's` structure transformed into JSON.

When the application is launched, it reads the `PlayerPrefs` and rebuilds the two variables, `_anchorsNames` and `_anchors`, by reading the JSON and transforming them into lists and C# classes.

In details, the variables `_anchorsNames` and `_anchors` will have the following structure in JSON form:

- List of all anchors' names: the list is simply transformed in JSON.
- Anchor: each object of type Anchor of variable `_anchors` is transformed in JSON, so it will be a dictionary of type

```
{ "Name": "",
  "Position": {
    "x": float,
    "y": float,
    "z": float
  },
  "Holograms": [],
  "Descriptions": [],
  "Tracked": bool
}
```

where Holograms and Descriptions are lists, which will be transformed in JSON, and each element of the list will have the following structure:

```
Hologram
{ "Name": "",
  "Position": {
    "x": float,
    "y": float,
    "z": float
  },
  "Quaternion": {
    "x": float,
    "y": float,
    "z": float,
    "w": float
  }
}
```

```
Description
{ "Text": "",
  "Position": {
    "x": float,
    "y": float,
    "z": float
  },
  "Quaternion": {
    "x": float,
    "y": float,
    "z": float,
    "w": float
  }
}
```

Holograms organization

The holograms in the application are added as 3D models and then for each of them a Prefab is created, in order to add it to the application. In the following steps, it is explained how to add them to the Unity project and where they are managed.

Import 3D Models

1. Find your desired 3D Model in one of the supported formats by Unity
2. Enter the Unity project
3. Through the Project tab, navigate into Assets > 3D Models
4. Create a subfolder named as your 3D Model
5. Import your model by the menu Assets > Import new Asset

Set 3D Models

1. Select the 3D Model

2. Go in the Inspector Tab of the model
3. Set the Materials and the Textures of the model
4. If an animation of the model is present, then set also the Animation
5. Go into the Material settings and set its shader as: `Unlit > Texture` (for most of the object which already have a material)

Create Prefab

1. Through the Project tab, navigate into `Assets > Prefab > Anchors`
2. Create a new empty Prefab with the name of the 3D model
3. Import the desired 3D Model in the Prefab

Add Prefab to the list

1. Through the Project tab, navigate into `Assets > Prefab > Anchors`
2. Select the file `AnchorsSettings`
3. In the Inspector tab:
 - (a) Add a new element to the list
 - (b) Drag the prefab just created in the new input field of the list

UI Holograms In order to be able to use the new hologram, then go in the Create scene, and edit the UI as necessary.

- If the new hologram is added to the list:
 1. Add a new button
 2. Name the button as the Prefab
 3. Go in the Inspector tab and add in the `OnClick()` tab, the functions as in Fig. 8
 4. In the `ObjectActivation.Activation` function set the same name of the desired Prefab in the input field

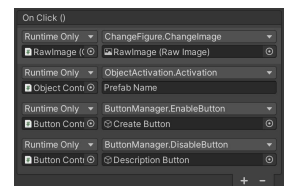


Figure 8

Usage

Necessary Tools

- Lenovo ThinkRealityA3 Smart Glasses [3]
- Motorola edge 30 pro or Motorola edge+ phones
- Snapdragon Spaces SDK 0.11 [5]

NOTE! Any device working with Snapdragon Spaces SDK can be used with this project and application.

Getting started

1. Install *Unity* version 2020.3.40 (A different version is not guaranteed to work properly!)
2. Clone the repository at the link Github in the preface

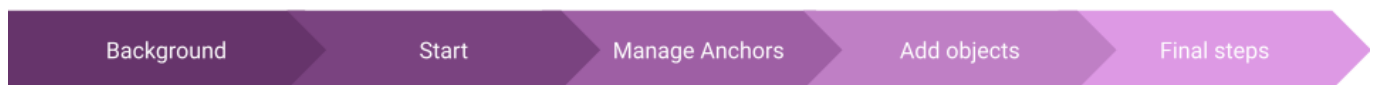
3. Open the project via Unity
4. Set up the Unity project as described in the website [6] of the Snapdragon Spaces SDK documentation

Usage

For the users, who only want to try the application is sufficient to install the Release app via adb command.

For the users who want to see the code or the Unity project, is sufficient to open the project as described in the *Getting started* bullet points.

Workflow



Background

- Read documentation
- Try ThinkRealityA3 smart glasses
- Try SDK'S samples on the glasses
- Set the goals of the project
- Create a theoretical idea of the project

Start

- Understand how to use the Snapdragon Spaces SDK
- Start to set the graphic of the project
- Set the diagram of the scene
- Add the possibility to interact in the app, to move between scenes

Manage Anchors

- Create the first anchors and load them with a simple prefab
- Add the constraint on the distance between local anchors
- Understand how the anchors are updated when the application is running
- Add a max number of anchors

Add Objects

- Understand how to connect multiple object to a single anchor
- Understand how to save them (JSON)
- Understand what is the exact position and rotation to save, in order to visualize the object in different sessions
- Load the objects in different sessions
- Add the possibility to create notes by opening the keyboard on the phone and save the input text

Final steps

- Manage the anchors in a transparent way
- Add different type of 3D Models, more complex and with animations

Final considerations

Finally with this project I could understand and learn how to deal with local anchors in an AR application. Furthermore, I could develop a way to attach multiple objects to a single anchor, in order to have only few anchor points in the world, with more objects, such as holograms or little notes.

The applications of this project can be many, even though there are still some problems to work on and different possible further works to improve the app.

Use cases

The app can be used in different situations for the use of anchors but also given the fact that it is quite all hand-free. In the following bullet points there are some possible use cases.

- **Work**: The app can be used in a work environment where the worker has to relocalize objects to have more information about the real world. For example in the case of a worker in a warehouse, who has to find specific products. With this application, the worker can have more information about the products in certain locations. Moreover, the possibility to have the app hand-free in the guest mode gives the chance to the worker to wear the glasses without the tie to use their hands while using the app.
- **Disabled person**: Another use case can be with disabled people. Thanks to the fact that works hand-free, the app is very useful both for people who does not have the capability to move their hands, but also for people who can not use them, such as people in the wheelchair, who have to use hands for their movement and they can not actually use them for the app.
- **Exhibition**: Another possible use of the project can be for exhibitions, such as in a museum or in a company. With the use of the glasses and the feature of objects relocalization, the app can be used to show extra information in addition to what there is already in the real world.

Problems

There are still some problems with the app, which are caused especially by the instability of Snapdragon Spaces SDK.

Three are the main problems:

- In case of a crash of the application, to go back to normal functionality it is better to follow the next steps:
 1. Force the quit of the application in Settings > App > ApplicationName > Force Stop
 2. Delete the data of the application and its cache
 3. Restart the phone
- It can happen that the application is not able to save at the first try the holograms, especially when it has to create the local anchor. This is usually due to the fact that the illumination is not the best or the environment is not so clear, and this causes difficulty for the application in saving the local anchors because it can not find the right features to track the point and save them. If the application can not save the object, the object will be removed. So it is important to note that, in the case of the disappearance of the object, it means that the object has not been saved in the store. This

problem usually happens at the creation of the first local anchor. The solution is to try again to add an object in the AR world, until the object is saved.

- This type of anchor is very sensitive to changes in the environment, such as objects' motion or light changes, so pay attention to keep the real world as similar as possible to when the objects are instantiated.

NOTE! Pay attention to close the application in the right way by triggering the button `Quit`. Do not remove the glasses until you see the phone's graphic on the glasses. This step is crucial for saving the data in the right way.

Further works

There could be different ways to improve the project, such as:

- Keep updating the Snapdragon Spaces SDK version.
- Add the possibility to use hands for users who can actually use their in the app, especially for the Admin mode.
- Insert additional features, in order to have a better experience, such as Hit testing when adding a new object.
- Add a possible AR marker-based feature, such as the use of a QR Code, especially in those cases where the environment is dynamic and can change over time.

References

- [1] Ayman Beghdadi and Malik Mallem. "A comprehensive overview of dynamic visual SLAM and deep learning: concepts, methods and challenges". In: *Machine Vision and Applications* 33.4 (May 2022), p. 54. ISSN: 1432-1769. DOI: 10.1007/s00138-022-01306-w. URL: <https://doi.org/10.1007/s00138-022-01306-w>.
- [2] The Khronos® Group. *OpenXR*. URL: <https://www.khronos.org/openxr/>.
- [3] Lenovo. *ThinkRealityA3*. URL: <https://www.lenovo.com/it/it/thinkrealitya3?orgRef=https%5C%253A%5C%252F%5C%252Fwww.google.com%5C%252F>.
- [4] Qualcomm Technologies. *Local Spatial Anchors*. URL: <https://docs.spaces.qualcomm.com/common/features/GeneralFeatures.html#local-spatial-anchors>.
- [5] Qualcomm Technologies. *Snapdragon Spaces™ Augmented Reality SDK*. URL: <https://spaces.qualcomm.com/sdk/>.
- [6] Qualcomm Technologies. *Unity Setup Guide*. URL: <https://docs.spaces.qualcomm.com/unity/setup/SetupGuideUnity.html>.
- [7] Unity Technologies. *AR anchor manager*. URL: <https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.2/manual/anchor-manager.html>.
- [8] Unity Technologies. *JSON Serialization*. URL: <https://docs.unity3d.com/2020.1/Documentation/Manual/JSONSerialization.html>.
- [9] Unity Technologies. *LoadSceneMode.Additive*. URL: <https://docs.unity3d.com/ScriptReference/SceneManager.LoadSceneMode.Additive.html>.
- [10] Unity Technologies. *PlayerPrefs*. URL: <https://docs.unity3d.com/ScriptReference/PlayerPrefs.html>.