

Semester Thesis

Handheld Augmented Reality for Robotic Excavators

Autumn Term 2023

Contents

1 Preface	ii
2 Abstract	iii
3 Introduction	1
4 Related Work	2
5 Method	3
6 Excavator Setup	4
7 Data Transfer	5
7.1 Multiplayer Connection	5
7.1.1 Connection Challenge	5
7.2 State Sharing	6
8 Colocalization	7
8.1 Azure Spatial Anchors	7
8.2 Colocalization Testing	7
8.2.1 Testing: Meadow Environment	9
8.2.2 Testing: Parking Lot Environment	11
8.2.3 Robustness	12
8.2.4 Testing Conclusion	13
8.3 Alternative Location Sharing	13
9 Conclusion	14
9.1 Data Transfer	14
9.1.1 Checksum Inequality	14
9.1.2 Unreliable Local Connection	14
9.2 Colocalization	15
9.3 Outlook	15
Bibliography	16
A Additional Images	17
A.1 Colocalization Testing	17
A.1.1 Ambient Changes	17
A.1.2 Repeatability	19

Chapter 1

Preface

Working with robotic systems is both very challenging and rewarding especially when tapping into a new direction. Personally it is a challenge I quite enjoy to tackle however it becomes meaningful only when the underlying goal has the capability of invoking positive change. I believe an autonomous excavator contains such a potential and I feel very fortunate to contribute a small part in its development.

Chapter 2

Abstract

This project aimed at paving the way for the creation of a high level remote control for autonomous excavators using a handheld augmented reality device.

The two key requirements for such an endeavour are first of all to be able to send data as well as inputs back and forth between the two devices. Secondly there are two camera views in this setup the AR view of the handheld device and the image input that the excavator receives. In order to precisely send geometric inputs from one device to the other colocalization is required.

The idea was to connect the handheld device making use of Unreal Engine game instances. For the excavator's side this was already setup. For the colocalization requirement the approach was to use Microsoft's Azure Spatial Anchors for a frictionless implementation between the two camera origins. Thus an additional goal was to further understand the capabilities of these less conventional methods for robotic applications.

Chapter 3

Introduction

When operating an excavator in the conventional fashion two requirements are imperative:

- The view onto the construction site
- The handles necessary to give inputs to the machine

In the case of an autonomous excavator the low level actions to perform are determined by the machine itself, given a high level action input. So in this case the second requirement becomes the possibility to give such a high level input to the system.

With the desired handheld remote control setup the necessity of sharing information and even inputs between the device and the machine still persist. The requirements of the visual component change however. From the handheld camera we now receive the required view of the surroundings on the construction site but in order to successfully share a geometric location with the excavator the colocalization problem has to be solved for the two entities. Having an AR remote control integrated in a handheld device also provides the possibility of introducing further useful features such as displaying a preview of an action of choice.

In this project I attempted to overcome the key challenges constituting the requirements mentioned above.

The plan was to solve the data transfer requirement utilizing the already implemented Unreal Engine setup of the autonomous excavator¹ from the Robotic Systems Laboratory. To account for the colocalization problem an approach using Microsoft's Azure Spatial Anchors was used.

¹ HEAP - The autonomous walking excavator[1]

Chapter 4

Related Work

Certainly the most important related work and the reason this project can even take place is the autonomous excavator heap[1] itself.

Overall the methods chosen in this project are very applied and unconventional which is why there is little related work to be cited. It has to be mentioned however that the entire structure of Unreal Engine multiplayer connections¹ is a fundamental building block for this project. Especially cross platform multiplayer connections were essential for this work. Secondly the Azure Spatial Anchor² process is a pipeline that this work relied on.

¹Networking in Unreal Engine

<https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Networking/Overview/>

²<https://azure.microsoft.com/de-de/products/spatial-anchors/>

Chapter 5

Method

As mentioned above the approach in this project can be divided into two stages:

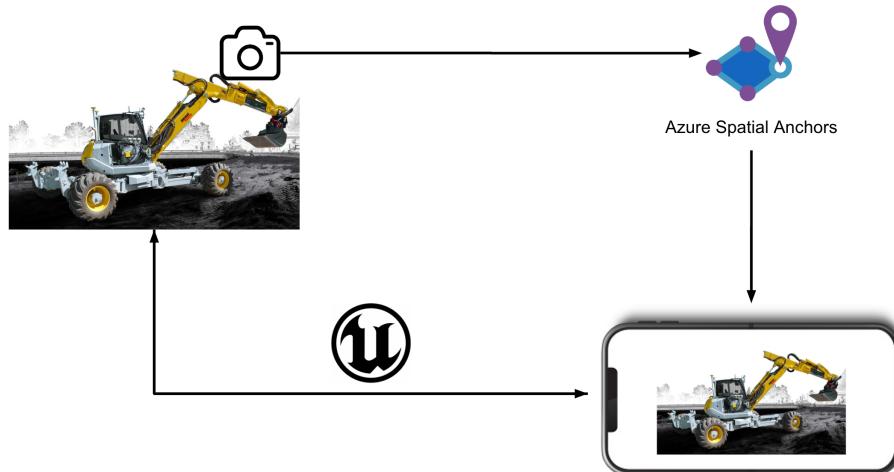


Figure 5.1: High level handheld remote setup

- Data transfer between the excavator and the handheld device (left arrow connection)
 - Creating an Augmented Reality Unreal Engine (UE) game on the handheld device
 - Making use of the already existing UE interface on the excavator to store system information in game components
 - Using an Unreal Engine local multiplayer connection in order to transfer data between the two UE instances
- Colocalization between the two frames of reference (right arrow connection)
 - Extracting a spatial anchor from the excavator's camera view using a ROS wrapper
 - Uploading this visual anchor in the form of an Azure Spatial Anchor to the Azure cloud
 - Retrieving the stored spatial anchor in the handheld UE instance
 - Using the spatial anchor to relocate the UE world origin

Chapter 6

Excavator Setup

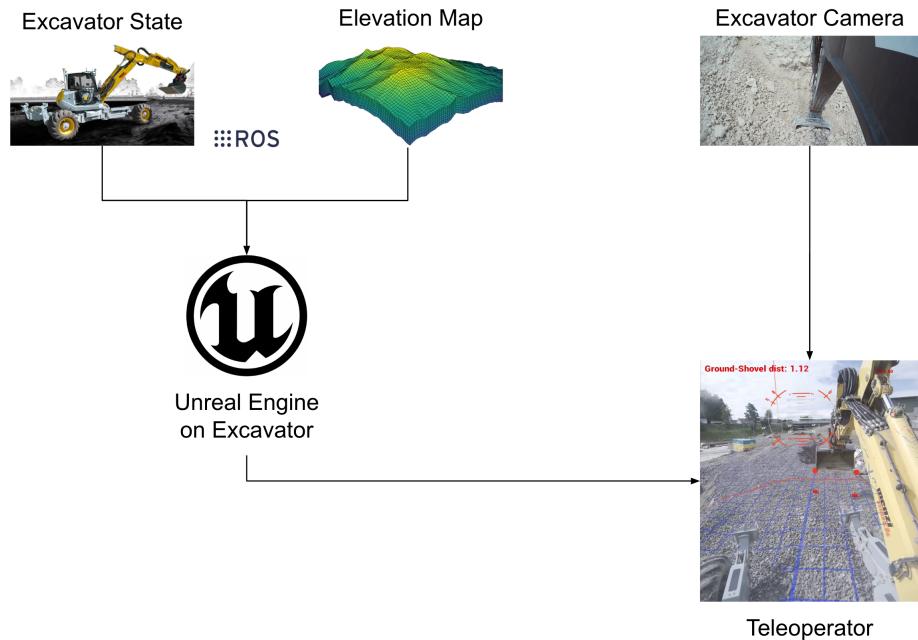


Figure 6.1: High level excavator setup prior to this project

The excavator has an Unreal Engine instance running on it. In the UE instance there is a model of the excavator itself which is updated continuously using a built in ROS subscriber game component.

Another crucial input to the excavator is an elevation map which is generated from the visual input of the environment.

Through the current state of the excavator at any time it is possible to determine the parts of the camera input image which are covered by the excavator. When overlaying the elevation map onto the camera input this feature allows for the creation of a mask to cut the excavator out of the elevation map projection. This combined image input is then fed to the teleoperation setup. This is just one use case of having a real time updated simulation of the system at hand.

In the handheld setup we can make use of this pipeline to fulfill our specific requirements and further down the road enable high level remote control inputs.

Chapter 7

Data Transfer

7.1 Multiplayer Connection

As mentioned above the approach to fulfill the data sharing requirement was to use an Unreal Engine multiplayer Connection.

There are different possibilities of online subsystems to use. As the plan is to use the handheld device on the construction site, the local network connection provided by Unreal Engine was the method of choice as opposed to an online connection. Unreal Engine offers different types of multiplayer implementations. Mainly two options could be relevant:

- Client - Server model
- Dedicated Server model

The first option allows both the server and the client to actively interact with the game. This is useful in case we want to perform any UE action on the excavator. In this case the excavator's UE instance would be playing the part of the server and the handheld device would be the client.

The second option lets an Unreal Engine instance function as a real server without the possibility of interacting actively with the game. With this option the Unreal Engine instance on the excavator would serve as the dedicated server. In general this setup provides a more stable multiplayer connection however the possibility of interacting actively with the running UE instance would no longer be possible on the excavator.

In this project I used the Client - Server approach in case the necessity of input based interaction with the excavator UE instance should arise in the future.

7.1.1 Connection Challenge

Normally when connecting two Unreal Engine games through a multiplayer connection it is the same game simply played from different devices. In this case however we have one UE instance running on the excavator using Linux both for development (ROS) and for the platform it is played on (ROS Subscriber). The other game runs on Android and is an AR game. So in this setting we have a connection between two games which have fundamentally different components as well as different platforms that they run on.

On a low level a successful UE multiplayer connection comes down to the requirement of identical checksums in the two instances. The checksum is a number generated by Unreal Engine which depends on the project name and certain components in the game level amongst other factors.

It turns out that neither the platform the game runs on nor special components like the AR implementation interfere with the checksum creation as long as the necessary components from the server (excavator) are also existing in the client (handheld) instance. Thus establishing a local connection between the two instances was possible. After successfully creating a local section on one instance, connecting to it with the other a game component could be successfully replicated to the second instance. Given that this game component contains the information of interest this solves the data transfer problem.

At a later point in this project when working on the colocalization part as described in chapter 8 with Azure Spatial Anchors it was necessary to switch to the handheld android app development on Windows. This is due to the fact that the Azure Spatial Anchor Unreal Engine plugin is only available on windows.

When developing on Windows the checksum of the project changed however prohibiting the two UE instances to establish a successful connection. So in order to use the Azure Spatial Anchor colocalization approach from this project the checksum has to be manually overwritten if possible or otherwise another Online Subsystem has to be used than the UE default implementation.

Generated Checksums	Ubuntu Game	Android Game
Linux Development	96120701	96120701
Windows Development	-	2769037734

Table 7.1: Checksum Comparison

7.2 State Sharing

To establish all data transfer connections between the two devices would have been too wide of a scope for this project which is why I focused on the transmission of the excavator's current state. In general if the transmission of any information succeeds using this setup then all further information can be packaged into game components accordingly and shared in a very similar fashion.

As seen in fig. 6.1 the excavator's state is fed to the Unreal Engine model directly through a ROS state subscriber node which is embedded in the game.

To solve the data transmission problem for the handheld device the idea was to intercept this connection. The state input is thus not only sent to the excavator model but also into a state storage game component. This game component would then be updated continuously and replicated in the handheld UE instance with the new values. Simultaneously if an excavator model is referenced by this state storage actor the state is fed into the model at each game tick to also update the handheld excavator model's state.

This state sharing pipeline worked really robustly and without added delay when testing on one Unreal Engine instance with a rosbag file representing the excavator movement.

Chapter 8

Colocalization

When creating a point in the Unreal Engine instance on the handheld device then it's coordinates are considered in reference to the world coordinate system of the Unreal Engine game. In order for the excavator to be able to use the set geometric location it has to be converted into a representation with respect to the excavator's coordinate system.

As previously mentioned the approach used in this work was to extract Azure Spatial Anchors from the excavator's camera view using a ROS wrapper and after uploading them to Azure Cloud retrieve them using an Unreal Engine plugin on the mobile device. In the end this allows us to find the relative transformation from the UE world coordinate origin to the excavator's coordinate origin which is represented by the pose of the detected anchor.

8.1 Azure Spatial Anchors

An Azure Spatial Anchor works in the following way:

- By scanning the environment visual reference points are detected.
- When enough visual information is received the initial pose of the scan can be stored by creating an anchor at that position.
- This anchor can then be uploaded to the Azure cloud to be retrieved whenever a device detects the visual surrounding encoded in the anchor.
- Note: When scanning the environment an anchor can also be placed at a set location with respect to the origin as opposed to the origin itself.

In this project the pose to track and to store in the Azure Cloud was the origin of the camera device itself. This made it possible to find the relative transformation between the coordinate systems.

The advantage of using Azure Spatial Anchors is that the colocalization is completely taken care of by creating and detecting an anchor due to the visual information of the environment that is encoded within it.

8.2 Colocalization Testing

For the testing of the proposed pipeline a Realsense t-265 tracking camera was used which represented the camera view from the excavator.

Two different environments were used to perform the colocalization tests. A meadow and a mostly empty parking lot:



Figure 8.1: Testing Environments

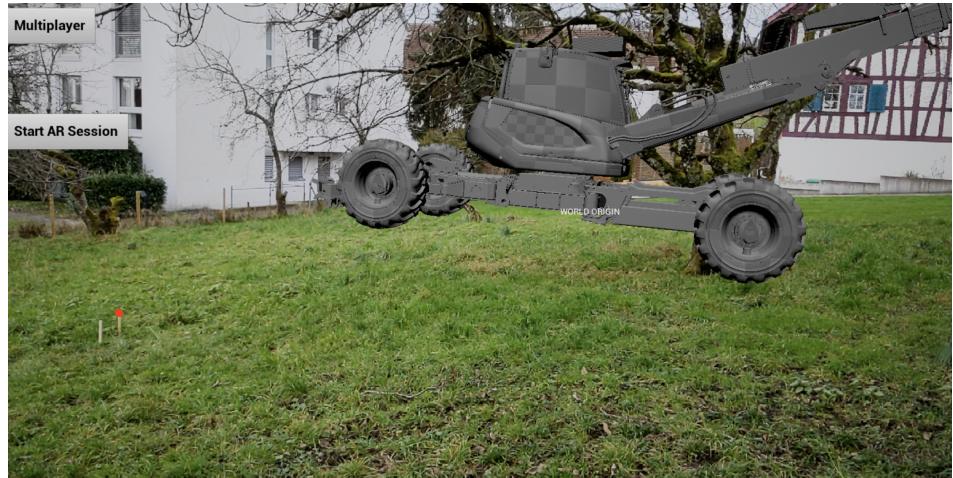
For the following tests the setup was always the same:

- The t-265 camera was started at a prominent point and the environment was scanned.
- An anchor was created (supposedly at the starting position) using the visual information registered.
- The handheld UE app was started at a random position spawning an AR excavator model at that pose.
- The anchor was detected by the handheld UE instance and using the relative transform the excavator model was shifted to that pose.

8.2.1 Testing: Meadow Environment



Figure 8.2: Camera origin representing the location where to place an anchor



(a) Excavator pose at initialized UE world origin



(b) Excavator pose after colocalization

Figure 8.3: Meadow Colocalization

As can be seen in fig. 8.3 after the colocalization the excavator together with the world origin is relocated to a new pose. There is however a shift between the supposed anchor location (indicated with an afterwards inserted red dot) and the new location the excavator assumes. This can be traced back to a shift in the anchor creation.

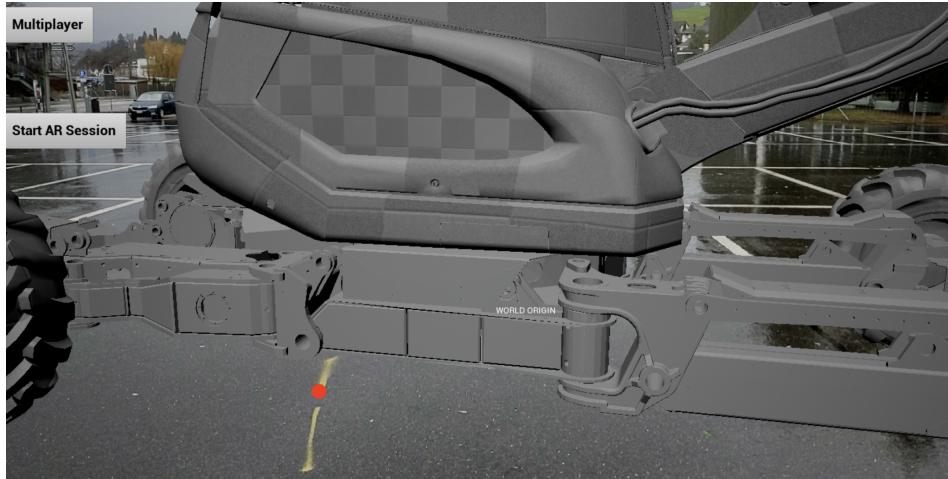
8.2.2 Testing: Parking Lot Environment



Figure 8.4: Camera origin representing the location where to place an anchor



(a) Excavator pose at initialized UE world origin



(b) Excavator pose after colocalization

Figure 8.5: Parking Lot Colocalization

Here we have a similar performance as with the meadow environment. A "correct" relocalization to a shifted initial anchor.

8.2.3 Robustness

When doing the same test with different ambient circumstances like variations in lighting or weather we can see quite robust results. For instance the anchor point for the parking lot environment was created at a somewhat dry period of the day while the anchors were detected when the ground was really wet. Looking at appendix A.1.1 we can also see robust results for worse lighting conditions in the evening.

The second kind of robustness that was tested in this setup was the repeatability of this process from different starting poses of the handheld device. The results thereof can be seen in appendix A.1.2. The same conclusion can be drawn yet again: we have a very consistent behavior considering the fact that the anchor created is off.

8.2.4 Testing Conclusion

We find that we have a persisting error when creating the visual anchors with the t-265 Tracked Camera. Given these shifted anchors however the colocalization process is very accurate and robust regarding changes in illumination, weather and starting handheld position.

Also the different environments didn't seem to have a significant influence on the colocalization performance.

8.3 Alternative Location Sharing

Taking a step back the initial reason to establish colocalization was to send a high level spatial input from the handheld device to the excavator.

Note that the transmission of any location in the form of a spatial anchor could work. In the case of the origin being transmitted we get a transformation between the coordinate systems which can be used for any further point created in the handheld UE. However, only as long as the excavator does not move.

An alternative possibility to send a touch location from the handheld input to the excavator and receive it in that frame of reference would be to use Azure Spatial Anchors directly. In that case a point would be created in UE using the intersection of the touch input's virtual line and the first obstacle hit (usual touch input point generation). Instead of transmitting this points location in the world coordinate system through UE, an anchor would be created encoding that location and sent up to the Cloud. It would be using the same Azure Spatial Anchor connection as before but in the other direction. An advantage from this approach would be that there is no dependence on a correct and persistent transformation to the excavator position. At any given time an anchor could be created and transferred without worrying about the relative transformation between the two systems.

Chapter 9

Conclusion

The initial problem setting in this project was to pave the way for the creation of an AR handheld remote control for an autonomous excavator. The following two requirements were tackled:

9.1 Data Transfer

The proposed method in this work was to use an Unreal Engine multiplayer connection for the information sharing process. This seems to be a working solution with a few persisting obstacles which have to be worked around before implementing it in the pipeline.

9.1.1 Checksum Inequality

When the colocalization requirement is tackled using the Azure Spatial Anchor approach as described in chapter 8 then the problem of unequal checksums arises. This has to be worked around. Currently the following options come to mind:

- Sequentially changing the handheld game structure in order to reach the desired initial checksum again. (Due to limited documentation of the checksum generation this is mostly educated guesswork)
- Finding a way of manually overriding the generated checksum in the handheld UE instance
- Making use of a different subsystem as for instance Steam¹ provides. This option provides a robust solution with the drawback that one more piece of software is necessary on the excavator which naturally is undesirable.

9.1.2 Unreliable Local Connection

Throughout this project many different network settings were tested and even with sufficient network speed the multiplayer connection between the UE instances was not reliable. To draw a definite conclusion here more tests would be necessary however a possibility might be to use a dedicated server² on the excavator instead of the client - server model.

¹Steam Online Subsystem
<https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Online/Steam/>
²as described in section 7.1

9.2 Colocalization

The Azure Spatial Anchor set up allows for quite an easy implementation as the colocalization process is handled completely by this module given the successful creation and detection of such a spatial anchor.

As seen in section 8.2 the anchor creation tends to have a shift in its location. Apart from this the colocalization pipeline seems very reliable with robustness regarding both ambient changes as well as repeatability. As it is not apparent where this shift of the anchor location is introduced it would be best to perform tests with the actual excavator setup directly.

9.3 Outlook

With the state transfer connection working between the two UE instances any other piece of data can be converted into an Unreal Engine game component and shared in the same way. A more robust connection between the devices would be beneficial however.

This very same connection can be used to give orders from the handheld device to the excavator given that the UE connection remains a client-server relationship.

As the inclusion of the Azure Spatial Anchor plugin renders the connection unviable for the time being this has to be looked into further as indicated in section 9.1. Alternatively a different method for colocalization would be necessary.

So all in all this project provides two individual modules that solve the two respective key requirements of handheld remote control for autonomous excavators. However for now there are still complications prohibiting this joint setup.

Bibliography

- [1] D. Jud, S. Kerscher, M. Wermelinger, E. Jelavic, P. Egli, P. Leemann, G. Hot-tiger, and M. Hutter, “Heap - the autonomous walking excavator,” 2021.

Appendix A

Additional Images

A.1 Colocalization Testing

A.1.1 Ambient Changes



(a) Dummy ball component pose at initialized UE world origin



(b) Dummy ball pose after colocalization

Figure A.1: Parking Lot with Evening Lighting



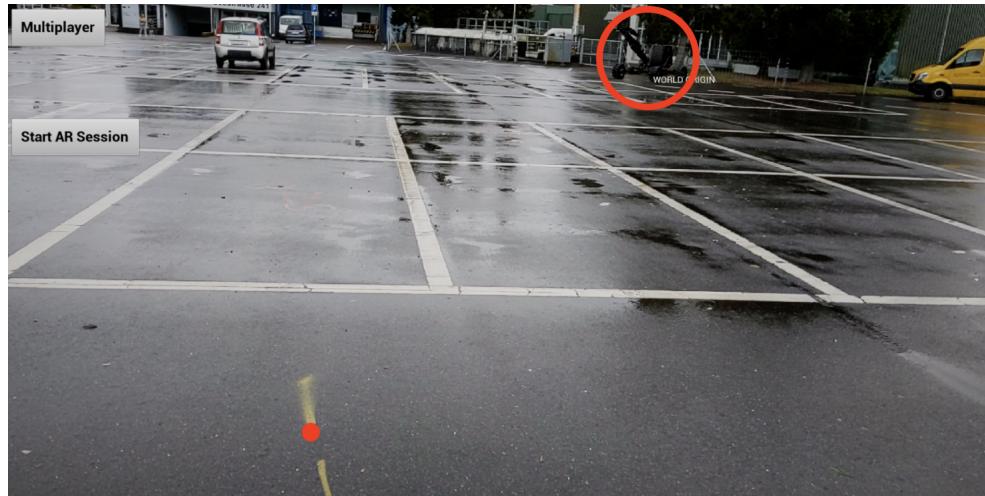
(a) Excavator pose at initialized UE world origin



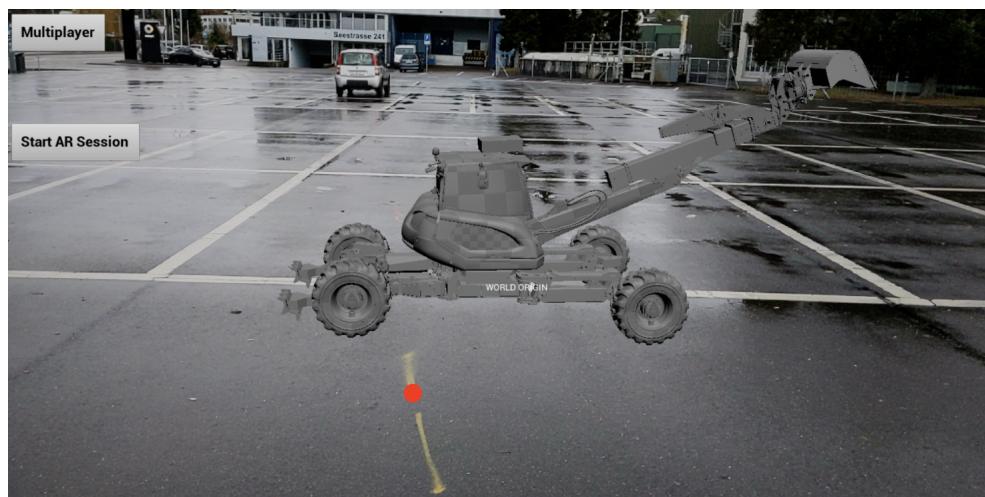
(b) Excavator pose after colocalization

Figure A.2: Meadow with Evening Lighting

A.1.2 Repeatability



(a) Excavator pose at initialized UE world origin location 1

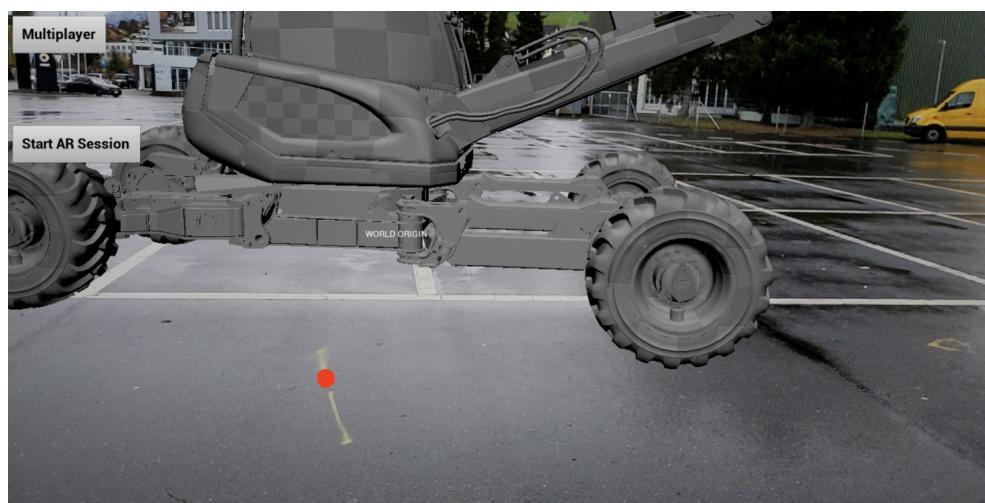


(b) Excavator pose after colocalization 1

Figure A.3: Colocalization Location 1



(a) Excavator pose at initialized UE world origin location 2

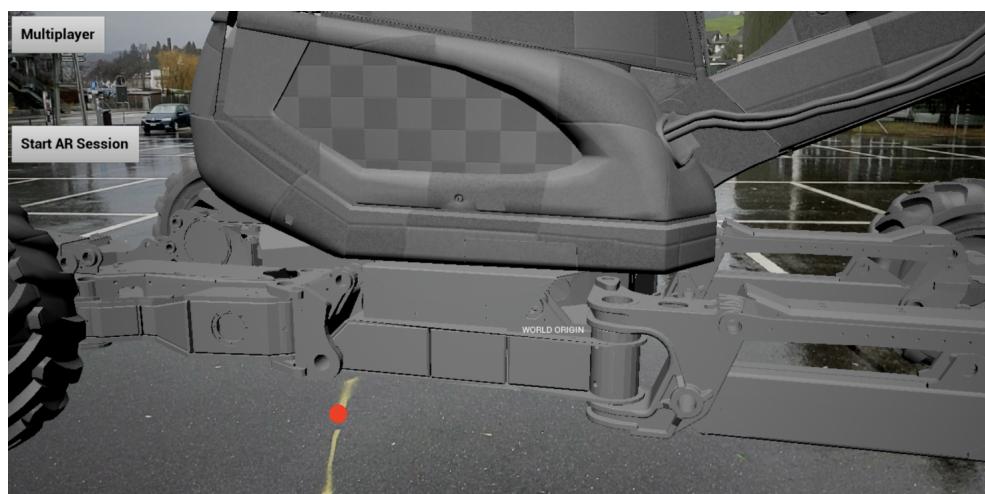


(b) Excavator pose after colocalization 2

Figure A.4: Colocalization Location 2



(a) Excavator pose at initialized UE world origin location 3

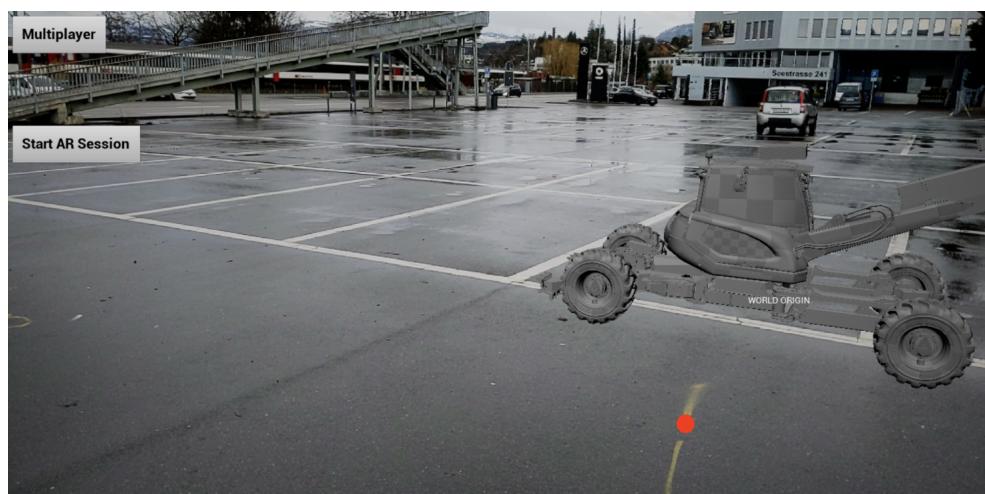


(b) Excavator pose after colocalization 3

Figure A.5: Colocalization Location 3



(a) Excavator pose at initialized UE world origin location 4



(b) Excavator pose after colocalization 4

Figure A.6: Colocalization Location 4