ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

RSL Robotic Systems Lab
Prof. Dr. Marco Hutter

Semester Thesis

Handheld Augmented Reality for Robotic Excavators

Autumn Term 2023

Supervised by:
Dominic Jud
Ryan Luke Johns

Author:
Lasse Fierz

Contents

Chapter 1

Preface

The aspiration for this project was to contribute to the process of making use of new technology in order to assist humankind. The field of robotics has tremendous potential in this regard and I feel fortunate to be partaking in this endeavour.

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. A second aspiration was to explore the usage of less conventional technologies for robotics such as Unreal Engine and Azure Spatial Anchors. The main goal was to provide the necessary building blocks for a handheld remote control to fullfil the task of feeding an input to an excavator. The main requirements are a way of sharing information between the two devices and solving the colocalization problem between the reference frames. After failed previous attempts of establishing the data transfer using a ROS connection between the two devices the focus in this work lied on making use of the already implemented excavator setup.

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 between the device and the machine still persists. The visual requirements 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 excavator[1] 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 whole structure of Unreal Engine multiplayer connections is a fundamental building block for this project. Especially cross platform multiplayer connections were essential for this work. Secondly the Azure Spatial Anchors process is a pipeline that this work relied on.

Chapter 5

Method

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

- Data transfer between the excavator and the handheld device
  - Making use of the already existing UE interface on the excavator to store system information in game components
  - Creating an Augmented Reality Unreal Engine (UE) game on the handheld device
  - Using an Unreal Engine local multiplayer connection in order to transfer data between the two UE instances
- Colocalization between the two frames of reference
  - 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

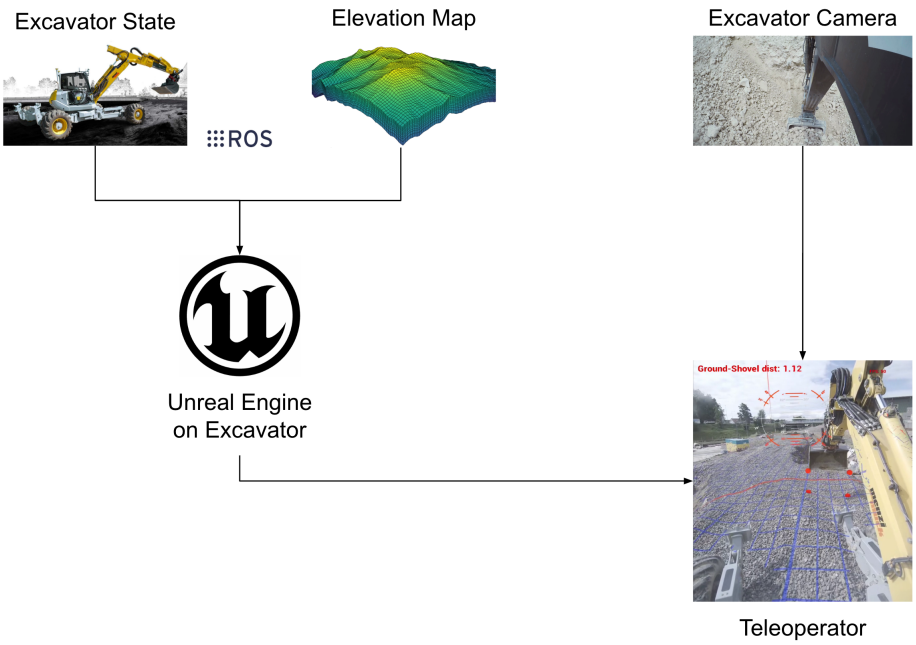


Figure 6.1: High level excavator setup prior to this project

The excavator has an Unreal Engine instance running on it. In the UE instance there is a model of the excavator which is updated continuously using a built in ROS subscriber in UE.
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 calculate the parts in the camera input image which are occluded by the excavator. This feature allows for the creation of a mask to cut the excavator out of the elevation map projection.
In the end the Unreal Engine instance outputs this tailored elevation map projection which can be applied on top of the input image which is fed to the teleoperator.
In the handheld setup we can make use of this pipeline to fullfill our specific requirements and down the road enable high level remote control inputs.

Chapter 7

Data Transfer

7.1 Multiplayer Connection

As mentioned above the approach to fullfil the data sharing requirement was to use an Unreal Engine multiplayer Connection.
There are different possibilites 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. Unreal Engine offers different types of multiplayer connections. 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 the 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 arises 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.
It turns out that on a low level a successfull UE multiplayer connection requires identical checksums in the two systems. The checksum is a number generated from 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 this checksum 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.
At a later point in this project when working on the colocalization part 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. 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 than the UE default one has to be used.

Generated Checksum	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 the same 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 the pose of the detected anchor.

8.1 Azure Spatial Anchors

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.
In practice a mapping of the environment is created with the camera's starting position being the coordinate origin. As soon as enough visual inputs from the environment are gathered a location can be sent. In this project the location 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.

8.2 Colocalization Testing

For the testing of the proposed pipeline without the need of using the actual excavator a Realsense t-265 tracking camera was used which represented the camera view from the excavator. The procedure consisted of the steps described above:

- Scanning de environment
- Creating an anchor at the world origin
- Transmitting the anchor to the handheld UE instance using the Azure Cloud
- Extract the relative transform T_{EH} from the excavator to the handheld device

To test for robustness regarding the construction environment two area types were tested on:

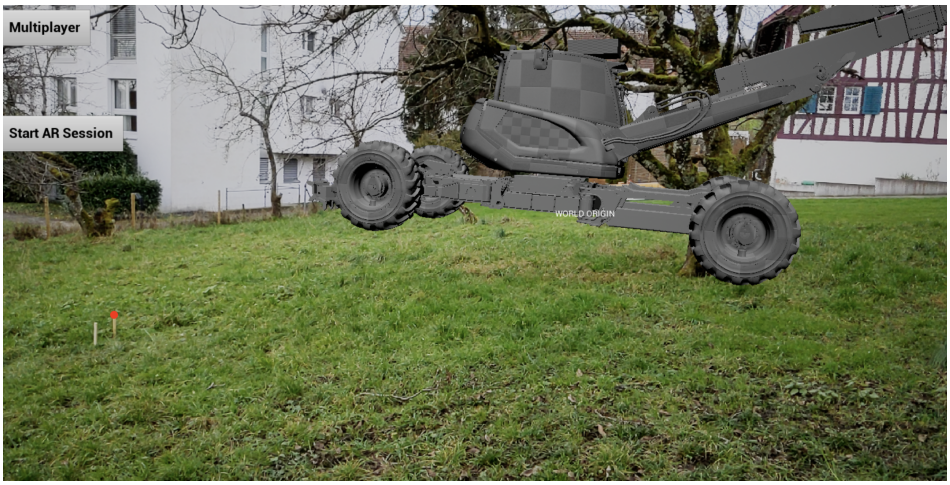
(a) Meadow

(b) Parking Lot

Figure 8.1: Testing Environments

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

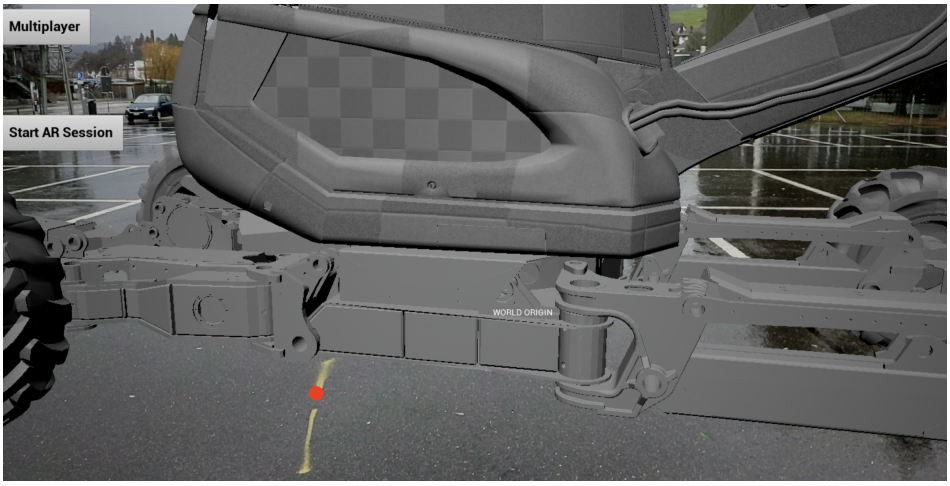
Figure 8.3: Meadow Colocalization

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 excvator assumes. This can be tracked 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. "Correct" relocalization to a faulty 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 to the change in quality. For instance the anchor points for the parking lot environment were detected at a somewhat dry period of the day while the anchors were detected rather easily when the ground was really wet. Looking at **??** we can also see robust results for worse lighting conditions in the evening.

The second kind robustness that was tested in this setup was the repeatability of this process from different starting poses. The results thereof can be seen in **??**. The same conclusion can be drawn yet again: we have a very consistent behaviour (considering the fact that the anchor created is still 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.

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 spatial anchor location 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.

Chapter 9

Conclusion

To come back to the initial problem setting in this project it was attempted 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 two following options come to mind:

- Reverse engineering the checksum in order to sequentally change update it and ideally 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 Steam[1] 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 server[2] on the excavator instead of the client - server model.

[1] Steam Online Subsystem `https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Online/Steam/`

[2] as described in section 7.1

9.2 Colocalization

There is of course a multitude of possibilities to accomplish colocalization between two camera origins. The Azure Spatial Anchor set up allows for quite an easy implementation as the colocalization process is handled completely by this module given the successfull 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.

9.3 Outlook

With the proof of concept of the state transfer in this manner it was possible to share information between the two systems. With this connection working any other piece of data can be converted into an Unreal Engine game component and shared in the same way. This very same connection can be used to transfer inputs from the handheld device up to the excavator given that the UE connection stays on a client-server basis.
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 working modules for the key requirements handheld remote control of autonomous excavators.

Bibliography

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