MPSE HoloLens - Quick Guide

Tobias Berthold, Peter Enenkel, Kin Liu, Mario Miosga

February 28, 2017

Contents

1 Project Definition

This chapter will introduce the project, its initial motivation, the specific goals and a tentative roadmap for future development.

1.1 Motivation

The overall goal of the project is to explore the possibilities of using the Microsoft HoloLens in the context of theater stage craft. To that effect multiple scenarios have been considered, of which the two predominate ones will be presented here.

One potential application is virtual stage design, that is a virtual stage scenery that, anchored on the (empty) physical stage, would allow the director (or stage designer) to walk, view and experience the set under realistic conditions. To the best of our knowledge the usual design process is still mostly an artisanal process involving craftsmen and physical miniature models. While sometimes virtual computer models are used, they are usually not the first choice as they cannot fully replace walking the physical set and inspecting it from different angles, also the established stage designers are apparently largely unfamiliar with such a medium. The introduction of, presumably easy to use, augmented reality could thus promote the more widespread use of such virtual models.

An other possibility lies in the area of stage machinery control. Modern theaters utilize a multitude of mechanical machinery to support the performance, the operators of said machinery often have a fairly limited direct view of the stage elements they control. This is mostly due to the fact that their position is fixed and constricted by the auditorium. Furthermore those elements may be occluded either by scene elements or other machinery. As the movement of potentially heavy loads requires the use of heavy-duty machinery the involved forces are often considerable, which in turn poses a significant risk of injury to the performers. Augmented reality would allow displaying helpful information to the operator without hindering his view of the elements he controls and without requiring him to divert his attention to some monitoring display.

From this later scenario the goals for the first phase of the project were derived. To simplify matters the scenario was reduced to a single stage element, a singular load bar. Figure 1.1 visualizes the milestones of phase I and

shows that the specific goal definition was found rater late in the course of the semester.

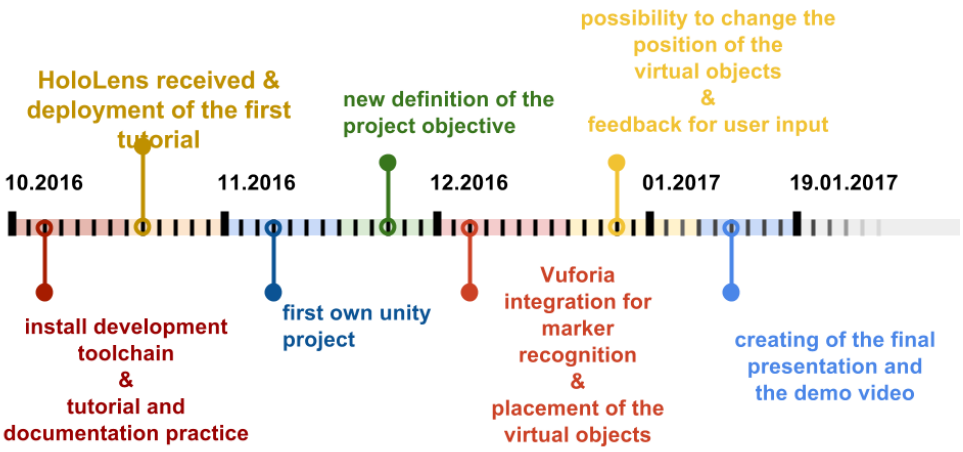

Figure 1.1: Milestones of phase I.

1.2 Goals of Phase I

The primary goal is to display a virtual load bar at the position of the physical load bar. In a real world scenario this should simplify the control of the stage machinery because the operator is still able to see all load bars, even if they are occluded by other objects.

In addition to that, some meta-information about the load bar is to be displayed next to said virtual load bar. For example a unique ID, the current position and the velocity are useful information for the operator that may increase the overall safety.

Furthermore it a virtual scenery shall be attached to the load bar. This corresponds with the desire for the possibility of virtual stage design as it has been described in section 1.1.

1.3 Roadmap

This section contains some tentative projections on how the project might develop in the upcoming semesters/phases.

- Phase I (until January 2017)
  - Superimpose a virtual load bar over the physical one

  - Display information near said virtual load bar
  - Attach a virtual scenery (i.e. poster) to the virtual load bar
- Phase II (until June 2017)
  - Receive information over the network
  - Position the virtual objects according to the received information
- Phase III (vision)
  - Multiple virtual objects/the whole stage
  - Integration with real stage machinery, both with a physical (miniature) stage and with stage control software

1.4 Organization

After some experimentation with H-DA internal recourses (Perforce and Moodle) we settled on GitLab as our primary source control platform. We also use it as a document management system by adding session protocols and documentation. This is manly due to the fact that it allows for easy, self-governing repository management and reliable, configuration free access, particularly from outside the university network. Additional features like hosting of a project website have not yet been used, but should perhaps be considered in the future, especially for documentation purposes.

2 Unity and Developing for HoloLens

Unity is a cross-platform game engine developed by Unity Technologies. It has been widely known for its ease of use and easy portability to different platforms. Even though Unity's primary focus is the creation of 2D and 3D games, it has also been known to be versatile in the creation of other realtime applications. In cooperation with Microsoft, Unity now includes support for HoloLens, Microsoft's latest mixed-reality device.

HoloLens Apps can be created in a variety of ways which include implementing them as Universal Windows Apps, DirectX projects or Unity Applications. Unity was chosen for this Project because of it's ease of use and intuitive workflow.

2.1 Unity Overview

A Unity Project contains four main elements: Assets, Scenes, GameObjects and Components. These elements interact with each other in different ways to create the final application.

Assets

Assets are the main content of any Unity Project. These can include 3D-models, textures, audio files software-libraries etc. Usually, most of the Assets are created using external software and then imported into the project.

Scenes

A Scene is a self-contained 3D space where all interaction happens. Every Unity Project needs to have at least one Scene. Scenes are the 3D space where Assets can exist. Most projects will have multiple Scenes. The most common use of Scenes in game development is to create different levels inside a game, however this will largely depend on the project specifications and design. Although a Unity Project can may have as many Scenes as necessary, it is important to state that only one Scene is active and running at a given time. Unity manages Assets in a Scene in form of GameObjects.

GameObjects

A GameObject is Unity's base entity. For an Asset to exist in a Scene, it has to be a GameObject. A GameObject by itself does not have any sort of behavior, other than that it exists inside a Scene. To give GameObjects meaningful behavior, so called Components have to be attached to a GameObject.

Components

A Component is responsible for assigning roles, properties and/or behaviors to GameObjects. Unity provides a wide variety of components such as lighting, colliders and physics bodies etc. which can be used to change the behavior of GameObjects. Custom behavior through scripting is also added to GameObjects through Components.

2.2 Working with Unity

Unity's primary focus is that of an game-engine, thus most Assets will be created in separate applications. The typical Unity workflow includes creating Assets, importing and configuring them inside Unity and adding logic through scripting.

Unity Editor

Unity manages its user interface through different windows. The default user interface contains the most used windows, which include:

Project Window

This view acts as a file browser where all Assets that belong to the project can be viewed. Assets can be organized by grouping them in folders. The project window also has search function to quickly find certain Assets.

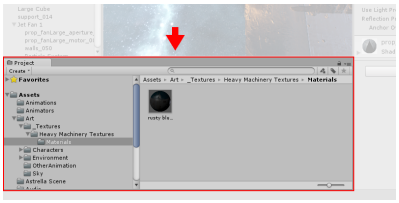

Figure 2.1: Project Window

Scene View

The Scene View is the interactive view into the world that is created. The Scene View is used to select and position scenery, objects, cameras, lights, and all other types of Game Objects.

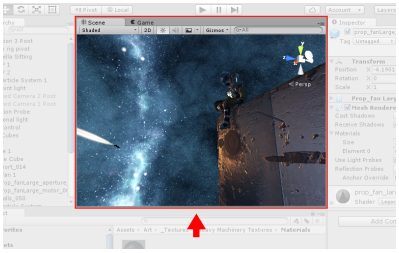

Figure 2.2: Scene View

Game view

The Game View is rendered from the Camera(s) in the Scene. It is representative of the final, published application. One or more cameras are needed to control what the user actually sees when they are using the application.

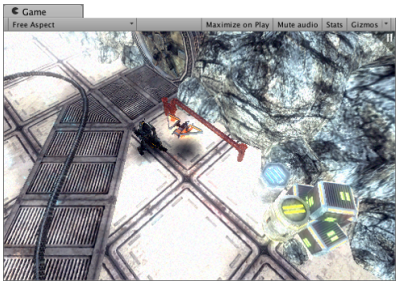

Figure 2.3: Game View

Hierarchy window

The Hierarchy window contains a list of every Game Object in the current Scene. Some of these are direct instances of Asset files (like 3D models), and others are instances of Prefabs, which are custom objects that make up most of your game. As objects are added and removed in the Scene, they will appear and disappear from the Hierarchy as well. By default, objects are listed in the Hierarchy window in the order they are made. Objects can be re-ordered by dragging them up or down, or by making them child or parent objects.

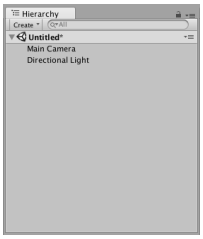

Figure 2.4: Hierarchy Window

Inspector window

The Inspector is used to view and edit the properties and settings of Game Objects, Assets, and other preferences and settings in the Editor. When a GameObject is selected in the Hierarchy or Scene View, the Inspector will show the Properties of all Components and Materials on that object which can be edited. The image below shows the inspector with the default 3D camera GameObject selected. In addition to the objects position, rotation and scale values, all the properties of the camera are available to edit.

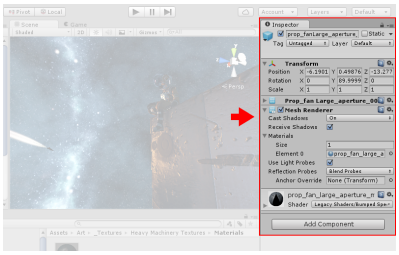

Figure 2.5: Inspector Window

A more in-depth overview of how to use Unity can be found in the official Unity manual[1]

2.3 Asset Workflow

When creating applications in Unity, the typical workflow involves importing Assets, configuring certain settings, adding components to GameObjects and adding custom behavior to those GameObjects through scripting.

[1]https://docs.unity3d.com/Manual/UnityOverview.html, last visited 27th February 2017

Importing/Adding Assets

Unity supports a wide variety of file types to be used as Assets, some common file types include:

- Image files
- 3D Model files
- Meshes & Animations
- Audio files

Each file type has it's own import settings, which show up in the inspector when a file is imported, and can be configured to suit the specific needs like quality, compression etc. Assets can also be imported from Asset-Packages, exported from other projects or from the Unity Asset Store. Being primarily a 3D game engine, Unity can work with 3D models of any shape that can be created with modeling software. However, there are also a number of primitive object types that can be created directly within Unity, which include

- Cube
- Sphere
- Plane
- Capsule

These objects can be added through the **GameObject >3D Object >Cube/ /Sphere/Plane/Capsule** panel. These basic primitives can act as placeholder objects which can be replaced for more complex 3D models later on.

Scripting

Unity provides its own API to manipulate GameObjects. This is known as Scripting. Scripts are code files which can be added to GameObjects as Components to achieve the desired behavior. Scripting can be done in two languages: C# or UnityScript (usually called JavaScript). UnityScript is Unity's own scripting language with a syntax similar to JavaScript, yet the two languages are not the same. Usage of C# in Unity is basically as one would expect. Even though both languages can be used in the same project, it is not recommended to do so for obvious reasons. Generally, C# is considered to be the better choice for scripting.

Creating and Using Scripts

Scripts are usually created within Unity directly. A new script can be created from the Create menu at the top left of the Project panel or by selecting **Assets > Create > C# Script (or JavaScript)** from the main menu. The new script will be created in whichever folder is selected in the Project panel. The new script files name will be selected, prompting to enter a new name.

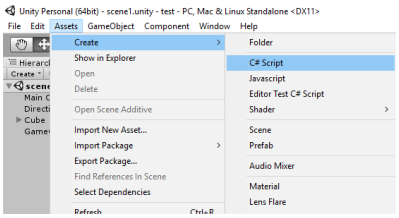

Figure 2.6: Creating a Script

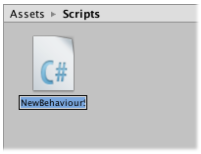

Figure 2.7: Script as it appears in the Project Window after creation

Editing Scripts

By default, the initial contents of a newly added Script will be the following:

```
using UnityEngine;
using System.Collections;

public class CustomScript : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

Unity's default editor for Scripts is MonoDevelop, though this can be changed to Visual Studio or any other editor of choice under the External Tools panel in the Preferences menu.

MonoBehaviour

A script makes its connection with the internal workings of Unity by implementing a class which derives from the built-in class **MonoBehavior**. Creating a new Script is basically creating a new Component-type that can be attached to a GameObject. Each time a Script Component is attached to a GameObject, it creates a new instance of the object defined by the Script. The name of the class is taken from the name supplied when the file was created. The class name and file name must be the same to enable the script component to be attached to a GameObject. Scripts can be added to GameObjects either by dragging them onto GameObjects via the Unity Editor or In-Code by using the *AddComponent* <>()-function derived from MonoBehavior:

```
void Start () {

    this.gameObject.AddComponent<ComponentToAdd>();
}
```

In Unity scripting, there are a number of event functions that get executed in a predetermined order over the lifecycle of the script. When creating a new script, the function bodies for *Start()* and *Update()* are generated automatically. To perform other actions, the corresponding MonoBehaviour functions can be overwritten. The following chart illustrates the available event functions:

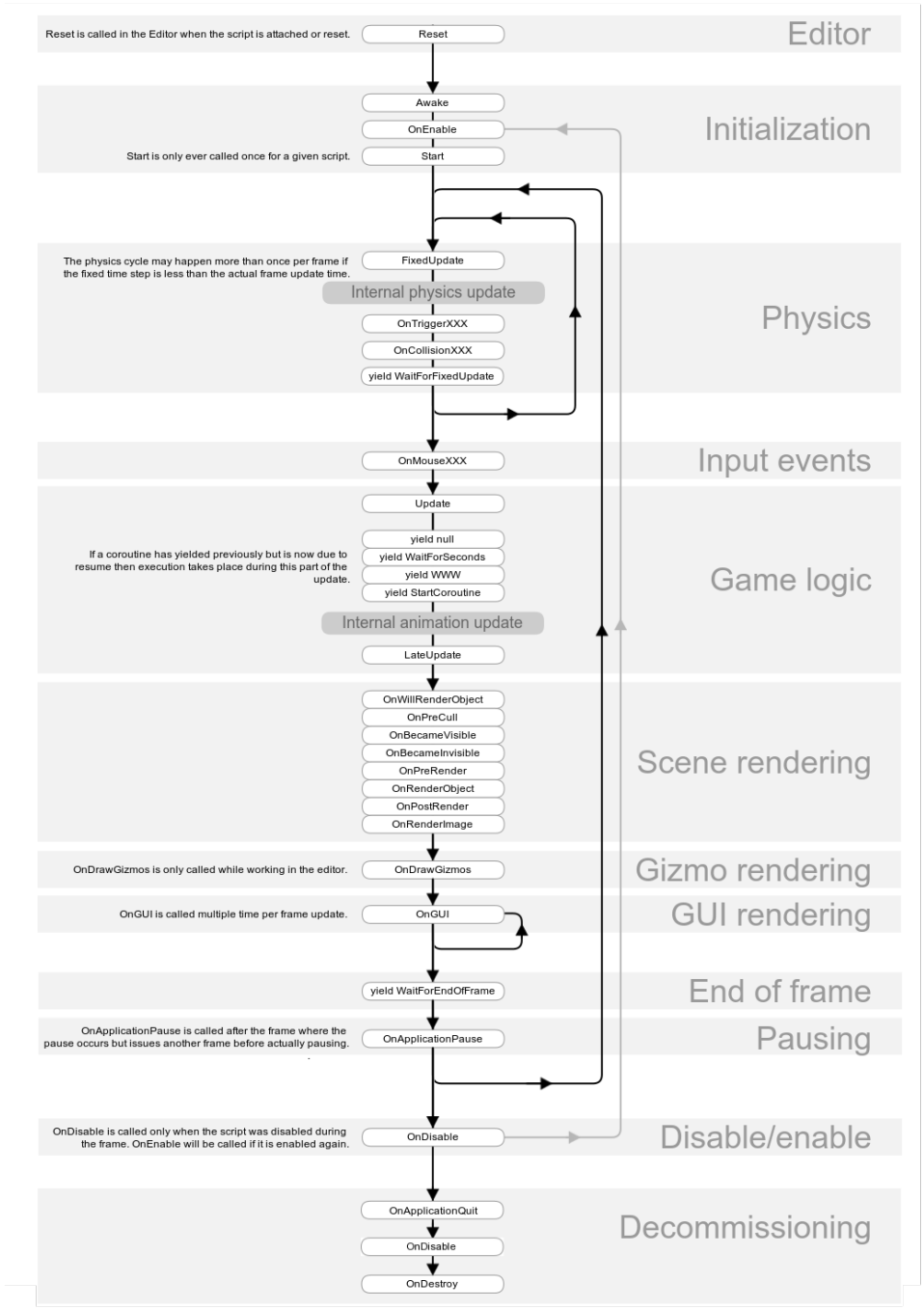

Figure 2.8: MonoBehavior lifecycle

2.4 Deploying an App to HoloLens

The build process of a Unity-based HoloLens-Apps differs from the build process of conventional Unity-Apps. Normally, a Unity Application can be run and debugged inside Unity itself, by using the toolbar:

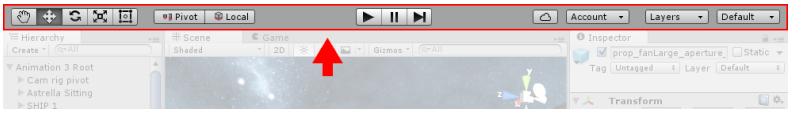

Figure 2.9: Unity toolbar

Conventional Unity applications can be run and debugged by pressing the *Play*-button from the toolbar. HoloLens-applications require an extra step in order to be deployed to HoloLens, which involves generating a VisualStudio-Project which is then used to deploy to HoloLens. A detailed description on the standard deployment process can be found in the *Holograms 100* tutorial[2]

Holographic emulation

Holographic Emulation allows HoloLens applications to be run and debugged inside of Unity, which shortens deployment time significantly. During the development of this project, Holographic Emulation was still in beta and not available for usage. The current project runs with Unity 5.4, whereas Holographic Emulation is scheduled to be released with Unity 5.5.

HoloToolkit

The HoloToolkit is a collection of scripts and components intended to accelerate development of holographic applications targeting Windows Holographic. HoloToolkit version 1.5.4.0 was integrated into this project. During development, version 1.5.5.0 has been released, but not yet integrated into the project. HoloToolkit contains scripts supporting the following topics:

- Input
- Sharing
- Spatial Mapping
- Spatial Sound
- Utilities

[2]https://developer.microsoft.com/de-DE/windows/holographic/holograms_100, last visited 27th February 2017

HoloToolkit also provides an extension to the Unity user interface, featuring its own Build Window, allowing for a more streamlined deployment process.

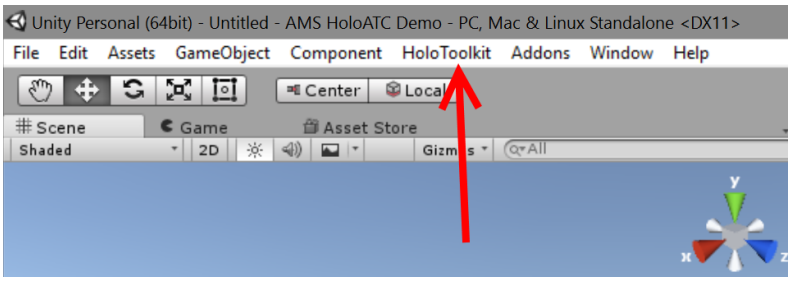

Figure 2.10: HoloToolkit extension

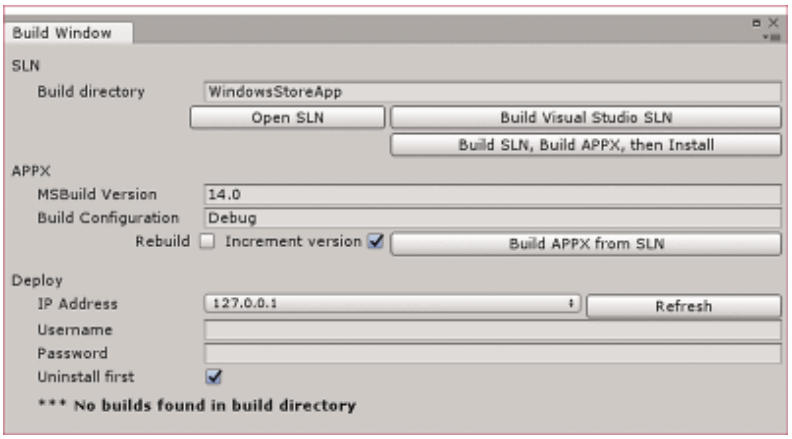

Figure 2.11: HoloToolkit build window

3 Implementation

In this chapter the current state of the HoloLens application developed by this project is presented. As stated in section 1 the considered scenario is a stage environment which is to be enriched with virtual overlays.

3.1 Concept

Because the project revolves around augmented reality both virtual and physical aspects must be considered. For the time being the stage is reduced to a single load bar, and is represented by a (rolling) clothing rack. The bottom plate of which is to be interpreted as the stage floor and the top bar as a (static) load bar. The corresponding virtual components have been kept similarly simple, with a box of approximate dimensions representing the load bar.

The application under development consists of a single Scene (see section 2 for a definition). The stage is represented by a corresponding GameObject. While that GameObject does not have any visual representation in the application, it allows us to use the Unity feature of nested positioning to place any stage element relative to the (virtual) stage. Said relative position can be assumed to be known. In the case of our clothing rack it was simply determined by measuring.

To map the virtual reality onto the real world it is now only necessary to synchronize the virtual stage with the physical one. To effect that a defined point on the virtual stage must be aligned with the corresponding point on the physical stage. This must be done in a way that also allows for the orientation to be aligned. In our application that is done via a QR marker utilizing the Vuforia framework.

All other virtual components that are to be positioned relative to the load bar (e.g. the information panel and the poster/virtual scenery) can now be automatically placed by including them as children of the virtual load bar.

3.2 Features

In this section we present all features implemented so far. This includes those directly required by the given goals, those merely convenient for development work as well as those enhancing the user experience.

1. Detecting a marker (QR-Code) in the real world.
2. Place a virtual load bar relative to the detected marker.
3. Three steps positioning of the virtual load bar.
4. Display information about position and velocity of the load bar.
5. Interaction via voice commands.
6. Feedback via TextToSpeech.
7. Debug log panel when debug mode is enabled.
8. Show and hide the spatial mesh.

3.3 Voice Commands

A big part of any immersive experience is the interaction with the user. Due to the fact that the HoloLens presentations feature gesture recognition so prominently we started out using that for our interaction requirements. As it turns out gesture recognition in not a particularly flexible mode of interaction, mostly because the number of available gestures is severely limited and thus context has to be provided by focusing scene elements which is not particularly comfortable for the user. A much better (and coincidently easier to implement) method is voice recognition, this allows implementing any desired action simply by adding an other key phrase. As long as one remembers to use sufficiently dissimilar phrases and an English pronunciation this works surprisingly well. Here is an overview of the currently implemented voice triggers:

Show Mesh: Displays the spatial mesh.

Hide Mesh: Hides the spatial mesh.

Show Log: Shows the debug log panel. Only available in debug mode. Displays all messages output via the Debug.Log() method. This was included because otherwise the debug log is only available while the HoloLens stays connected to the deploying PC which is slow and cumbersome.

Hide Log: Hide the debug log.

Calibrate: Start the detection of the marker and thus the placement of the virtual objects.

Stop calibration : Stops the calibration (normally not needed because after a successful marker detection the calibration mode is automatically terminated).

Move Up: Moves the virtual load bar down in steps of 0.5m.

Move Down: Moves the virtual load bar up in steps of 0.5m.

3.4 Audio Feedback

All user *inputs* are accompanied by a feedback mechanism, so that the user can be certain that his command was recognized correctly. This is done by acoustical feedback via buildin text to speech components. In addition to those confirmations there is also a feedback if the highest or lowest position of the virtual load bar has been reached.

4 Experiences and Tips

In this Chapter we are going to share our experiences, known problems and some tips that should help reducing errors and frustration while working with the HoloLens.

4.1 Requirements

Accounts and Logins

To contribute to this project the following logins are required. The accounts have been specifically created for this project. Access credentials should be provided by the previous/existing project team.

- Microsoft account
- Unity account
- Vuforia account
- Access to the GitLab group
- Device Portal login information
- Project computer login information

Software requirements

The following requirements are necessary for setting up the development toolchain and to run the emulator:

- Windows 10 64-bit (required for Emulator and Unity (when building UWP/HoloLens applications))
- Hyper-V (required for Emulator)
- Visual Studio 2015 update 3
- Windows 10 UWP SDK 10.0 (UWP=UAP) (can also be installed via the Visual Studio installer)
- Unity for HoloLens (special build based on Unity 5.4.0f3)

The Emulator is not mandatory. We tried to set it up but due to the high hardware requirements and the limitations (e.g. lack of a camera) we ended up not using it.[3]

4.2 Pairing and first deployment

The first time you deploy an app from Visual Studio to your HoloLens, you will be prompted for a PIN. On the HoloLens, generate a PIN by launching the Settings app, go to Update > For Developers and tap on Pair. A PIN will be displayed on your HoloLens. Paste this PIN in the popped up dialog in Visual Studio. After pairing is complete, tap Done on your HoloLens to dismiss the dialog. The PC is now paired with the HoloLens and you will be able to deploy applications. Repeat these steps for every PC that is used to deploy to your HoloLens.

There are two possible ways to deploy an application on the HoloLens. The deployment over Wi-Fi is more comfortable. especially if you want to debug your application. But we do not recommend it. One reason for that is, that we had a lot of issues due to a mis-configured subnet. If the two devices (PC from which you want to deploy and the HoloLens) are not in the same subnet, it won't work. If this happens there should be an error message like this : Visual Studio failed deploy to HoloLens: Error DEP6957 : Failed to connect to device. So if you have problems deploying an application over Wi-Fi or got this error, check the IP-configuration. A second reason is that the connection between the devices is fairly slow. As the size of data which must be transfered is about 100MB, it takes quite some time to deploy an application. In the end we mostly deployed via USB connection. Which is not all that comfortable especially when you want to debug your application (the HoloLens has to stay connected for that), but it is much faster and causes not as many problems.

First deployment via Wi-Fi:

1. Using the top toolbar in Visual Studio, change the target from Debug to Release and from ARM to X86.

2. Click on the drop-down arrow next to the Device button, and select Remote Device.

3. Set the Address to the name or IP address of your HoloLens. If you do not know your device IP address, look in Settings > Network & Internet > Advanced Options or ask Cortana "Hey Cortana, What's

[3] `https://developer.microsoft.com/de-DE/windows/holographic/install_the_tools`, last visited 27th February 2017

my IP address? (do not use Cortana, it gives wrong (i.e external) IP address!!)

4. Leave the Authentication Mode set to Universal.
5. Click Select
6. Click Debug > Start Without debugging or press Ctrl + F5. If this is the first time deploying to your device, you will need to Pair the device now.

4.3 Vuforia

Vuforia is used to detect the marker in the real world. The current marker is a QR-Code which contains the Homepage of Bosch Rexroth[4]. To use Vuforia in a new application first thing you need to do is go to the Vuforia Homepage and log in with the provided account[5]. Go to Dev Portal and Log in. Then go to Downloads and download Vuforia for Unity. Integrate the downloaded content in your Unity Project. Next you have to include the license key. To get this key you have to go to Develop and choose the license key in the license Manager. The last thing to do is to download the database in the Target Manager tab with all needed marker. To add a new marker you have to uploaded the picture here. Be sure that the size is correct and matches the size of your printout! Then download the database again and include it to your Unity project.

4.4 Device Portal

The Device Portal is very useful. Read the Using the Windows Device Portal[6] guide to learn how to set it up and what you can do with it. The login information should be provided. We made the experience that the functionality, especially the live view, depends on the web browser. The best results were achieved with Google Chrome. But the live view still does not work perfectly. The view is delayed about 5 seconds and it crashes very often. Also while streaming or recording, the quality of the holograms on the HoloLens decreases significantly and the power drain increases dramatically.

[4] `https://www.boschrexroth.com/de/de/`, last visited 27th February 2017

[5] `https://www.vuforia.com`, last visited 27th February 2017

[6] `https://developer.microsoft.com/de-de/windows/holographic/using_the_windows_device_portal`, last visited 27th February 2017

5 Useful Links

Microsoft HoloLens: `https://www.microsoft.com/microsoft-hololens/en-us` (zuletzt abgerufen am: 28.02.2017)

Unity for HoloLens: `https://unity3d.com/de/partners/microsoft/hololens` (zuletzt abgerufen am: 28.02.2017)

Developing Vuforia Apps for HoloLens: `https://library.vuforia.com/articles/Training/Developing-Vuforia-Apps-for-HoloLens` (zuletzt abgerufen am: 28.02.2017)

Microsoft HoloLens Academy: `https://developer.microsoft.com/en-us/windows/holographic/academy` (zuletzt abgerufen am: 28.02.2017)

Microsoft HoloLens Toolkit for Unity: `https://github.com/Microsoft/HoloToolkit-Unity` (zuletzt abgerufen am: 28.02.2017)

Using the Windows Device Portal: `https://developer.microsoft.com/en-us/windows/holographic/using_the_windows_device_portal` (zuletzt abgerufen am: 28.02.2017)