QUALISYS
Motion Capture Systems

ZENUITY

How To Guide - AR Engineering Demo

Gustaf Sjösten, Simon Jacobsson

June 2019

Document metadata

This document aims to describe how to set up and use a mixed reality demo application with Oculus Rift using ZEDmini and Qualisys motion tracking. The chapters I-III are intended for developers, while chapter IV is intended for users.

The project was an 8 week long joint venture between Zenuity and Qualisys AB.

Contents

I Setup **3**
- I.1 Qualisys Track Manager . . . 3
- I.2 ZEDmini . . . 3
- I.3 RealSense . . . 3
- I.4 Unity . . . 3
  - I.4.1 Unity plugins . . . 3
- I.5 Our GitHub Repository . . . 3
- I.6 Hardware . . . 4
- I.7 QTM 6DOF body . . . 5

II Calibration **6**
- II.1 AR Engineering demo - OR-QTM calibration Unity application . . . 6
- II.2 calibrate.py . . . 6
- II.3 AR Engineering demo - 6DOF Rotation calibration Unity application . . . 7

III Outlook **8**
- III.1 Better tracking . . . 8
- III.2 Ability to run application without having an oculus sensor plugged in . . . 8
- III.3 Automate 6DOF body rotation calibration . . . 8
- III.4 Better GUI:s . . . 8

IV Quick Guide to Depth Mapping in Virtual Reality **9**
- IV.1 Steps for running the application . . . 9

I. *Setup*

I.1 Qualisys Track Manager

QTM can be found at Qualisys' website `https://www.qualisys.com/`.

GitHub repo for Unity SDK: `https://github.com/qualisys/Qualisys-Unity-SDK`.

I.2 ZEDmini

GitHub repo for Unity SDK: `https://github.com/stereolabs/zed-unity`.

I.3 RealSense

RealSense SDK: `https://github.com/IntelRealSense/librealsense/releases/tag/v2.23.0`. The file `Intel.RealSense.Viewer.exe` is an interface for the RealSense camera, handy for checking if the camera and depth mapping works separately from the Unity project.

I.4 Unity

From `https://unity3d.com/get-unity/download?_ga=2.3117872.1092136400.1560262861-1948595714.1560159396`, Unity Hub can be dowloaded. In Unity Hub, download version 2019.1 and follow this `https://docs.unity3d.com/Manual/ManualActivationGuide.html` guide to activate the license.

I.4.1 Unity plugins

For this project we used

- Qualisys Unity SDK Package 7
- ZED Unity Plugin v2.8.0 (`https://www.stereolabs.com/docs/unity/`)
- Oculus Integration Unity plugin (`https://developer.oculus.com/downloads/package/unity-integration/`)

I.5 Our GitHub Repository

`https://github.com/gussjos/qz_summer_2019`.

The scripts contained in our github repo require Python3 with numpy and matplotlib.

I.6 Hardware

We used the following hardware:

- Oculus Rift
- ZED Mini Stereo Camera
- 4 Qualisys Miqus cameras
- Intel RealSense Depth Camera D435i

Attaching the ZEDmini to the Oculus Rift: `https://www.stereolabs.com/zed-mini/setup/rift/`.

Figure I.1 shows how the MoCap markers were attach to our Oculus Rift.

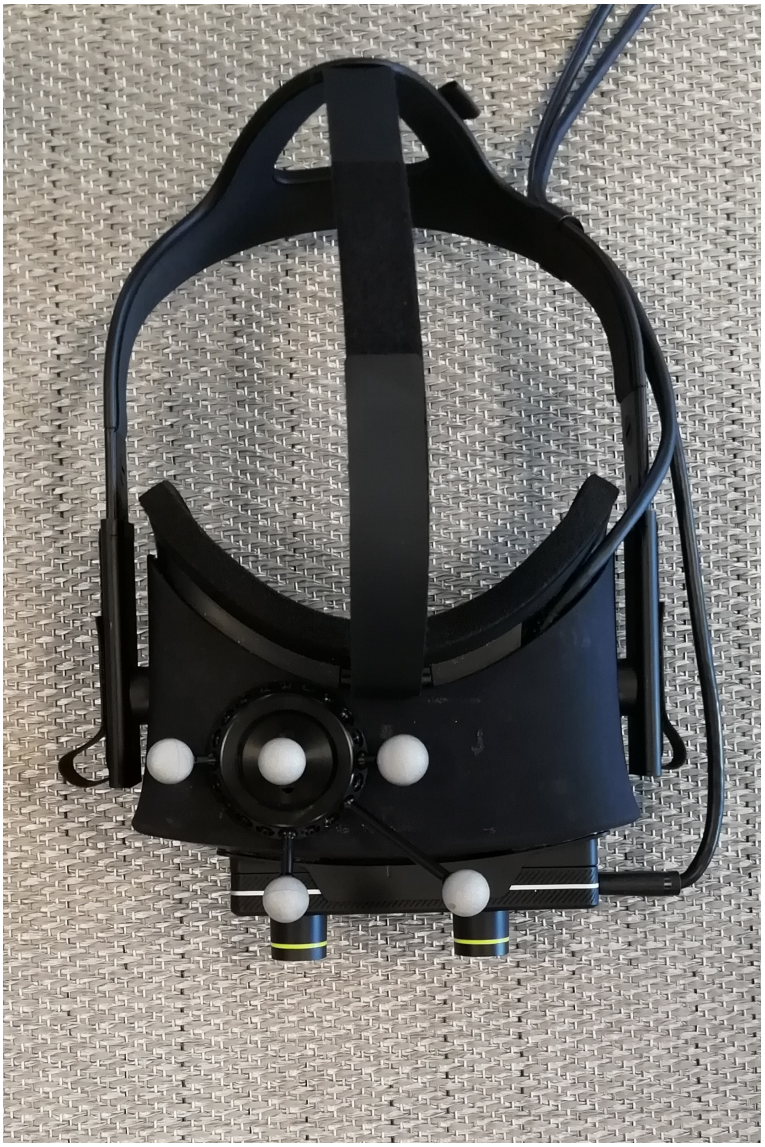

Figure I.1: Our experience is that such an elaborate setup is a bit overkill. Attaching the markers directly on the Oculus would likely be enough even with few MoCap cameras.

Figure I.2: Intel Realsense with a mounted 6DOF-body for MoCap-tracking.

I.7 QTM 6DOF body

After attaching the MoCap markers, define a 6DOF body in QTM named "oculus_rift" and save it as an .xml file. In Our Github Repo, the python script `calibrate.py` relies on having access to this file. To add its filepath, edit the `filepath_xml` string in `calibrate.py`.

II. *Calibration*

II.1 AR Engineering demo - OR-QTM calibration Unity application

The purpose of the Unity application "AR Engineering demo - OR-QTM calibration" is to track the OR HMD using both QTM and OR's own sensors so that `calibrate.py` can place the QTM 6DOF body origin at the OR reference node (HEAD).

To use this script, make sure the OR is tracked both by QTM and its own sensors (the external sensors). The HMD tracking is gathered as soon as the scene plays and until it is stopped by pressing esc. During the runtime of the application you will also need to locate the github repo from I.5, specifically, you want to select the qz_summer_2019 folder.

II.2 calibrate.py

The transformation between the QTM and OR coordinate system origins is described by

$$\vec{r}_{\mathrm{OR}} = R\big(\vec{r}_{\mathrm{QTM}} + \vec{s}\big) + \vec{t} \tag{II.1}$$

where $\vec{s}$, the *local translation*, is constant in the QTM 6DOF body's local coordinate system while $\vec{t}$, the *the global translation*, is constant in the global coordinate system. `calibrate.py` uses this model to calculate $\vec{s}$ through optimization. $\vec{s}$ is then used to identify the QTM 6DOF body origin with the OR reference node.

To use `calibrate.py`, run it with python3. It runs an optimization algorithm to solve (II.1) with respect to R and $\vec{s}$ and thus may take a couple of minutes to execute. The script will show you a comparison between the estimated right and left hand side of (II.1) that looks something like figure II.1. What you want to see in the figure is a good overlap between transformed QTM and OR trajectories. The script also prints the RMS error, which should be less than 0.05 mm for a good fit.

After you close the figure window, the script queries you about saving the data. If you chose to save the data, and locate the .xml file containing the 6DOF body "oculus_rift" (the one created in section I.7), the script automatically edits the 6DOF so that its origin matches the Rift's reference node. In order for this edit to take effect, the oculus_rift body then needs to be loaded again from its .xml file from QTM→Project Options→6DOF Tracking→Load Bodies. A protip is to run `calibrate.py` from a cmd terminal, so that you may view its output after the script finishes. Another protip is to not run this script several times since the you'll be translating the 6DOF body's origin further than appropriate.

If `calibrate.py` returns a bad fit, the problem is most likely a bad initial guess. To tweak this, open the script and edit the two variables `s_guess` and `quaternion_guess`. To make the optimization object function more concave, you may want to start with a short and uneventful trajectory and then take those results as initial guesses for a more complicated trajectory where you make sure all 6 parameters in 6DOF space are being varied.

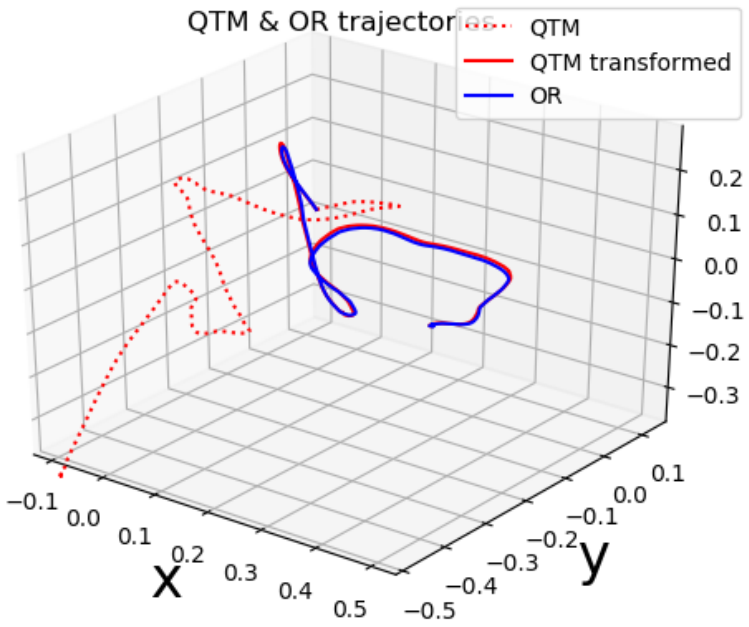

Figure II.1: In this plot, it appears that the transformation we've found produces a good fit between the data sets since the blue trajectory almost completely overlaps with the red trajectory.

If you want `calibrate.py` to run faster, you want the recording from the ZED-OR_Calibration to be shorter. This can be tweaked without rerecording by editing the `start_index` and `end_index` in the `read_unity_data.py` script.

II.3 AR Engineering demo - 6DOF Rotation calibration Unity application

This application is for calibrating the oculus_rift 6DOF body rotation in QTM. What the application does is basically just the let you see tracked MoCap markers in passthrough so that you can rotate your 6DOF body in order to align the virtual markers with the real. In QTM, the negative x-axis corresponds to forwards in Unity. You can start by guessing the forward direction from the 6DOF body. Since the Rift tracks yaw and roll by itself, the only value you'll need to tweak is the 6DOF rotation around its Z-axis.

Similarly, for calibrating the 6DOF body attached to the Realsense, you can just align the point cloud in the AR Engineering demo - RealSense AR Unity application with reality.

III. *Outlook*

This chapter provides some ideas for what someone continuing to build on this application might want to implement.

III.1 Better tracking

While running the application, the user might experience that the point cloud drags a bit, especially when making head movements, but it is also apparent when only moving the depth flashlight. There are at least two ways to adress this problem:

- Implement and improve the script `PIDController.cs` to fully utilize the rotational tracking of the Rift's IMU in combination with the rotation data gathered from QTM
- Improve the script `RecenterRotation.cs`. This is the script used currently for making smooth rotations but there might be room for improvement.

more details?

III.2 Ability to run application without having an oculus sensor plugged in

Right now, at least one Rift sensor must be connected to the computer running the application. The sensor is not used for tracking, but if disconnected, a critical error warning window will appear. The window can be closed by pressing 'OK' on the warning box, but pressing the button is difficult with a Rift controller when the sensors are disconnected..

It should be possible to get rid of this critical error and remove the need for a connected sensor.

III.3 Automate 6DOF body rotation calibration

The "OR-QTM calibration" application in Unity automates the calibration of the 6DOF body's origin position by comparing Rift and QTM tracking position data, but the rotation is also tracked by both the Rift and QTM. Assuming the Rift has a good sense of what is forward, the QTM 6DOF body could be rotated to match this.

III.4 Better GUI:s

For example, at the moment the different scenes each get their own .exe-file. It would probably be more user friendly if there was just one file with a GUI from which you could load the different scenes. It would also be nice with some confirmation from the calibration scenes that you had managed to successfully save the data.

IV. *Quick Guide to Depth Mapping in Virtual Reality*

This project uses an Intel Realsense Depth Camera D435i together with an Oculus Rift to create a depth mapping flashlight in virtual reality. The project relies on motion tracking with Qualisys QTM to track both the Realsense camera and the oculus rift for an outside-in-tracking that relates ones head position with the virtual flashlight.

To get started, please see section I.1, I.5, and I.6 for the required software and hardware.

IV.1 Steps for running the application

Follow the steps below to run the depth mapping application.

1. Mount the MoCap markers on the Oculus Rift and on the Intel Realsense as shown in figures I.1 and on I.2.
2. In QTM `Project Settings` load the predefined 6DOF bodies from `predefined_6dof_for_realsense_demo.xml` (section I.5). These bodies have predefined origins that coincide with the hardware.
3. Connect the Oculus Rift, ZED Mini, and RealSense Camera to your computer. At least one Rift sensor must be connected for the application to run.
4. Run one of the applications `AR Engineering demo - RealSense VR/AR.exe`. Press esc to quit the application.
5. The rotation of the 6DOF bodies in QTM are likely slightly off. For aligning the oculus_rift and realsense 6DOF bodies, see section II.3.
6. You're now up and running! Have fun :)