QUALISYS
Motion Capture Systems

ZENUITY

How To Guide - AR Engineering Demo

Gustaf Sjösten, Simon Jacobsson

Summer 2019

Document metadata

This document aims to describe how to set up and use a mixed reality demo application with Oculus Rift (OR) using ZEDmini, Qualisys motion tracking and Intel RealSense Depth Camera or a Velodyne LiDAR. The chapters I-III are intended for developers, while chapter IV - V is intended for users.

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

Contents

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.

Our main Unity project is called Rift_QTM_ZED_RealSense, which has two scenes, rift_qtm_realsense and rift_qtm_zed_realsense. There is also a project called OR-QTM_Calibration from which we built the calibration application.

I.4.1 Unity plugins

For this project we used

- Qualisys Unity SDK Package 7 (`https://github.com/qualisys/Qualisys-Unity-SDK/releases`)
- ZED Unity Plugin v2.8.0 (`https://www.stereolabs.com/docs/unity/`)
- Oculus Integration Unity plugin (`https://developer.oculus.com/downloads/package/unity-integration/`)
- Unity RealSense Wrapper at `https://github.com/IntelRealSense/librealsense/tree/master/wrappers/unity`.

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.

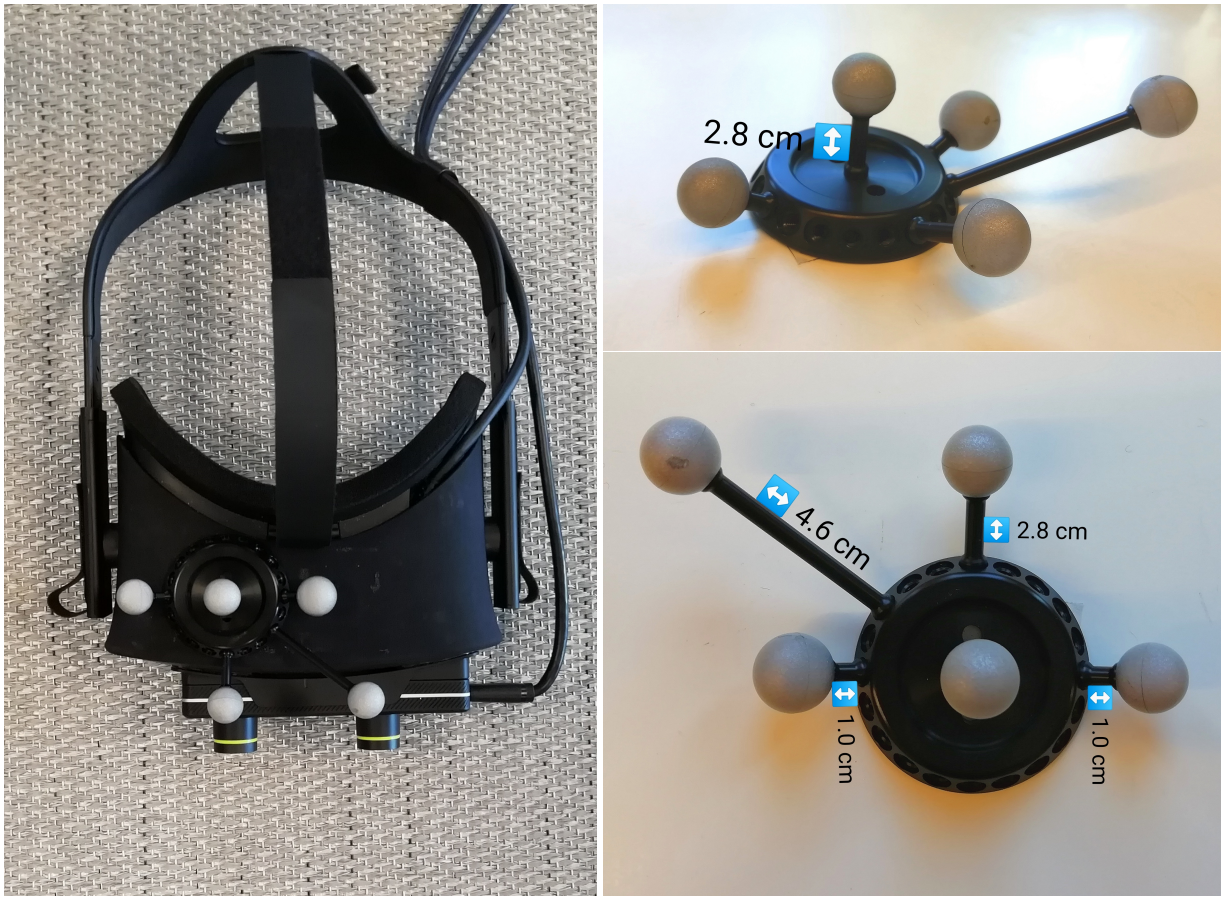
I.6 Hardware

We used the following hardware:

- Oculus Rift
- ZED Mini Stereo Camera
- Qualisys Oqus/Miqus cameras
- Intel RealSense Depth Camera D435i
- VLP-32c Velodyne LiDAR

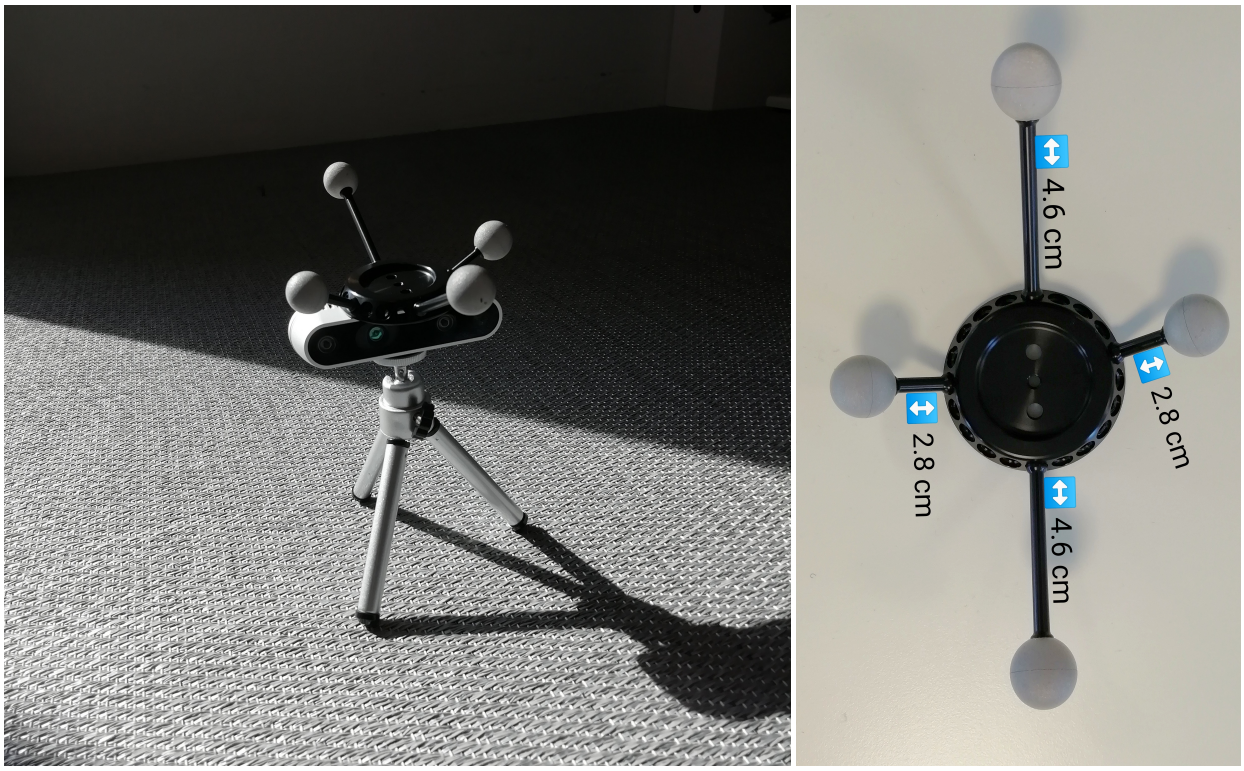
Attaching the ZEDmini to the Oculus Rift: `https://www.stereolabs.com/zed-mini/setup/rift/`.

Figure I.1a shows how the MoCap markers were attached to our Oculus Rift, and figure I.2a how the markers were attached to the RealSense.



(a) (b)

Figure I.1: This is the 6DOF rigid body contained in our GitHub repo. If you use our predefined 6DOF bodies, the application expects the rigid body to be attached to the rift in this way.



(a) (b)

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

I.7 QTM 6DOF body

In our GitHub repo, we provide a demo 6DOF rigid body which is contained in `predefined_6dof_for_realsense_demo.xml`.

If you want to use your own 6DOF rigid bodies, you can define one in QTM and save it as an .xml file. The 6DOF rigid body attached to the Rift should be named "oculus_rift" and the one attached the RealSense "realsense". Our application looks for these specific names. Also, in Our GitHub Repo, the python script `calibrate.py` relies on having access to this .xml-file.

II. *Calibration*

If you don't use the predefined 6DOF bodies contained in our GitHub repo, you'll want to accurately calibrate the position and rotation of the coordinate system attached to the 6DOF rigid body. The workflow of the positional calibration is

1. Measure the moving Rift headmount with both QTM and Rift's own tracking using the "AR Engineering demo - OR-QTM calibration" Unity application
2. Find the best transformation between the two data sets using the `calibrate.py` python script
3. Edit the QTM 6DOF rigid body according to this transformation

The workflow of the rotational calibration is more straight forward.

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 Rift using both QTM and its own sensors so that `calibrate.py` can place the QTM 6DOF rigid body origin at the Rift reference node (HEAD).

To use this application, make sure the Rift is tracked both by QTM and its own sensors (the external sensors shown in figure II.1). It first queries you for the path to the qz_summer_2019 folder from out GitHub repo, once you select this folder the HMD tracking data is gathered. You stop gathering by pressing esc. It is usually appropriate to gather data for ∼10 seconds.

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 Rift 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.2. What you want to see in the figure is a good overlap between transformed QTM and Rift trajectories. The script also prints the RMS deviation between the curves, 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 choose to save the data in the .xml file containing the 6DOF rigid body "oculus_rift" (the one created

Figure II.1: The Oculus Rift sensors.

in section I.7), the script automatically edits the 6DOF rigid body 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 save the data several times since the you'll be translating the 6DOF rigid 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.

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 rigid body rotation in QTM. What the application does is basically just to let you see tracked MoCap markers in passthrough so that you can rotate your 6DOF rigid 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 rigid body attached to the RealSense, you can just

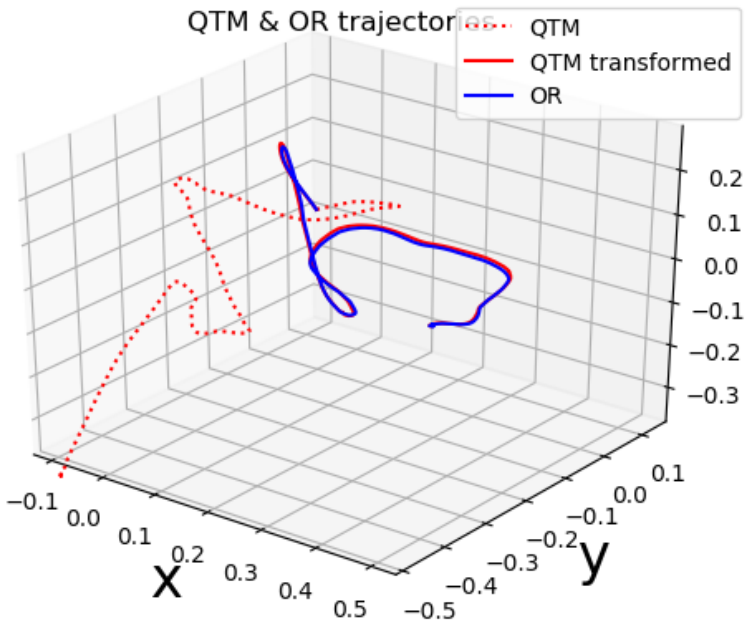


Figure II.2: In this plot, it appears that the transformation we've found produces a good fit between the data sets since the blue trajectory almost completely overlaps with the red trajectory.

align the point cloud in the "AR Engineering demo - Visualizing RealSense point cloud" Unity application with reality using passthrough.

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. The script `PIDController.cs` and `PIDEventNotifier.cs` are located the rift_qtm_zed-but_with_homebrewn_tracking scene in the Rift_QTM_ZED_RealSense project.
- Improve the script `RecenterRotation.cs`. This is the script used currently for making smooth rotations but there might be room for improvement. Currently the script seems to overcompensate a bit when making rapid head movements.

III.2 Performance analysis on calibration.py

The script calibrate.py takes a long time to run. There are likely ways to speed it up through vectorization of for loops and such.

III.3 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.4 Automate 6DOF rigid 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.5 Better GUI's

GUI's are always nice to have, specifically maybe a scene selector or a RealSense feature slider would be appropriate.

III.6 More features

- Currently, in the AR application, the pointcloud just renders on top of the canvas. There is actually functionality in the ZEDMini to sense depth and apply depth occlusion, in the Rift_QTM_ZED_RealSense Unity project: ZEDManager → Rendering → Depth Occlusion.
- A persistant point cloud, so that a user can trace out the 3D environment around them with the RealSense.

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

This project uses an Intel RealSense Depth Camera D435i together with an Oculus Rift and ZED Mini to create a depth mapping flashlight in mixed reality. The project relies on motion tracking with Qualisys Track Manager 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.1a and on I.2a.
2. Start tracking the MoCap markers in QTM. 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 the application `AR Engineering demo - Visualizing RealSense point cloud.exe`.
5. The rotation of the 6DOF rigid bodies in QTM are likely slightly off. For aligning the oculus_rift and realsense 6DOF rigid bodies, see section II.3.
6. You're now up and running! Have fun :)

IV.2 Keybinds

- Esc - quits the application
- 1 - toggles ZED Mini (AR/VR)

IV.3 QTM streaming

If you wish to stream QTM data from some other device than the one you're running on, there is a config.txt file next to the application (or you could create it). This .txt file contains just one line, with the IP adress of the streaming device. If there is no such .txt file, the application defaults to localhost (127.0.0.1).

V. *LiDAR application*