Technical Report

IRI-TR-18-01

Implementing a ROS Interface for HTC Vive tracker

Developing ROS software for HTC Vive

Laia Freixas

17 October 2018

Institut de Robòtica i Informàtica Industrial

Abstract

SteamVR is a virtual reality system that uses the HTC Vive headset, controllers and trackers, and is able to calculate their position with great accuracy. We used this system to develop a ROS node which allowed to locate the tracker with respect to the base stations. We then calibrated the system so that it would be possible to know the position of the tracker with respect to another reference, such as a robot base. Finally, we developed a system which allowed the WAM arm to follow the tracker.

Institut de Robòtica i Informàtica Industrial (IRI)
Consejo Superior de Investigaciones Científicas (CSIC)
Universitat Politècnica de Catalunya (UPC)
Llorens i Artigas 4-6, 08028, Barcelona, Spain
Tel (fax): +34 93 401 5750 (5751)
http://www.iri.upc.edu

Corresponding author:
Laia Freixas
tel: +34 93 4054490
lfreixas@iri.upc.edu
http://www.iri.upc.edu/staff/lfreixas

1 Introduction

Valve Corporation developed a virtual reality system called SteamVR, which uses the HTC Vive headset, two controllers (shown in Figure 1a), and also includes the possibility of using trackers (shown in Figure 1b). The advantage of this system is that it very accurately measures the position and orientation of trackers and controllers.

These accurate measures are performed with two base stations (shown in Figure 1c) which are located at the corners of the delimited playing area. These base stations scan the room, alternating between horizontal and vertical sweeps. The maximum playing area is of 4.5m x 4.5m.

In our case, we used the trackers without the headset. The communication between the devices to be tracked and the computer was done through a USB dongle. The goal was to have the WAM arm follow the position in which the tracker was located.

(a) HTC Vive controller

(b) HTC Vive tracker

(c) One of the two base stations used by the system.

Figure 1: Different devices used by HTC Vive system

2 System Requirements

SteamVR is a program that uses many resources, which is why it's necessary to test if the system we will install it on will be ready for it. The recommended computer specs are the following :

Component	Recommended
Processor	Intel Core i5-4590 or AMD FX 8350, equivalent or better
Graphics Card	NVIDIA GeForce GTX 1060 or AMD Radeon RX 480, equivalent or better
Memory	4 GB RAM or more
Video output	1x HDMI 1.4 port, or DisplayPort 1.2 or newer
USB	1x USB 2.0 port or newer

For more information and downloadable software to test the computer, check the following link : Vive Ready

3 Installation

3.1 Installing SteamVR

Steam is a video game platform developed by Valve. HTC Vive works with SteamVR, so it is necessary to install Steam in order to use HTC Vive.

The first step is to create a Steam account, needed to access all the games and tools provided by the platform. This can be done in the sign-up page.

Next, use the following command to install the dependencies for Steam platform

```
$ sudo apt install python-apt steam-launcher
```

Next, follow the instructions for downloading to obtain a .deb package for Ubuntu 16. Install the downloaded package as follows:

```
$ sudo dpkg -i steam_latest.deb
```

Once Steam is installed, we need to install SteamVR. To do so, we must open the Steam app, and login with the account we created previously. Then, go to LIBRARY, which will show a drop-down menu and select TOOLS. There, look for SteamVR and right click on it to install it. You should see something similar to Figure 2.

The next step is to opt-in to SteamVR beta. To do this, once SteamVR is completely installed, right clicking on SteamVR will open a drop-down menu in which we will select Properties. This opens a menu with different tabs: click on the BETAS tab, and select the SteamVR beta update.

The final step is to modify two configuration files. This is done in order to be able to use SteamVR without needing to plug in the headset, which will allow to work with the trackers only.

- `~/.steam/steam/steamapps/common/SteamVR/resources/settings/default.vrsettings`
  - Search for the **requireHmd** key under *steamvr*, set the value of this key to **false**.
  - Search for the **forcedDriver** key under *steamvr*, set the value of this key to **null**.
  - Search for the **activateMultipleDrivers** key under *steamvr*, set the value of this key to **true**.
- `~/.steam/steam/steamapps/common/SteamVR/drivers/null/resources/settings/default.vrsettings`
  - Search for the **enable** key under *driver_null*, set the value of this key to **true**.

3.2 Install dependencies

The dependencies needed are OpenVR and Vulkan. To install it, open a terminal and run the following commands.

```
$ sudo apt install libvulkan-dev
$ git clone https://github.com/ValveSoftware/openvr.git
$ cd openvr
$ mkdir build
$ cd build
$ cmake .. -DBUILD_SHARED=ON
$ make
$ sudo make install
```

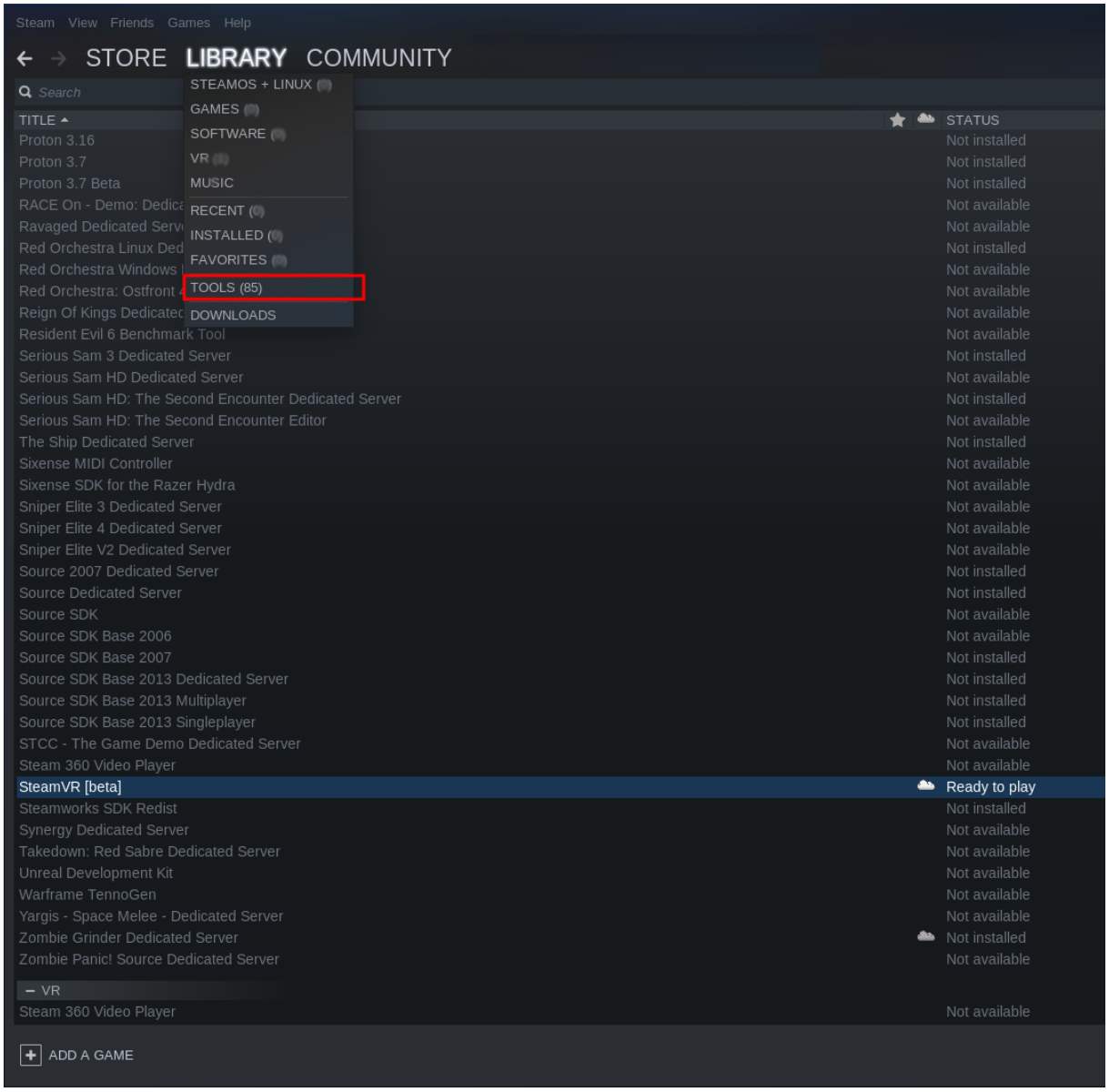


Figure 2: Installation instructions

Additionally, when installing, we can modify the CMakeLists.txt to install the headers automatically. To do so, edit `openvr/src/CMakeLists.txt` and add the following line at the end

```
install(FILES ${CMAKE_SOURCE_DIR}/headers/openvr.h ${CMAKE_SOURCE_DIR
    }/headers/openvr_capi.h DESTINATION include)
```

3.3 Installing C++ library

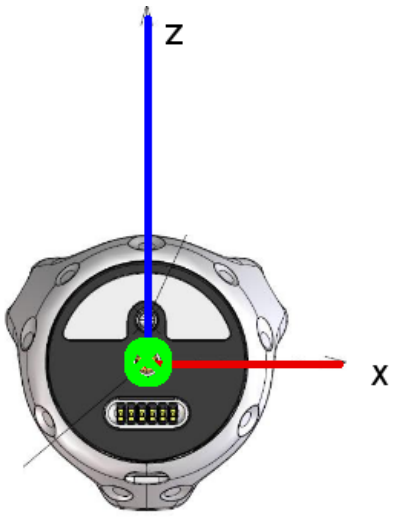
To install the C++ library, obtain it from the htc_vive_tracker github by executing the following commands.

```
$ git clone ssh://git@gitlab.iri.upc.edu:2202/labrobotica/drivers/
    htc_vive_tracker.git
$ cd htc_vive_tracker
```

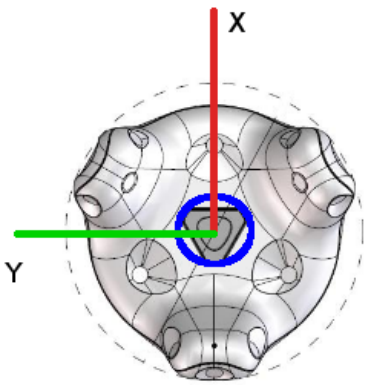
```
$ mkdir build
$ cd build
$ cmake ..
$ make
$ sudo make install
```

4 Coordinate systems

The original coordinate systems of the HTC Vive tracker is shown in Figure 3a. However, we decided that the coordinate system shown in Figure 3b would be more appropriate for the applications we need.



(a) Original coordinate system



(b) Final coordinate system

Figure 3: Different coordinate systems

5 ROS Node

This ROS node is in charge of publishing the position as a tf tree, publishing the odometry for the Kalman Filter and publishing the pose of the device so that it can be followed by a robot.

5.1 Installation

To download and install the ROS node execute the following commands.

```
$ sudo apt-get install ros-kinetic-robot-localization
$ roscd
$ cd ../src
$ git clone ssh://git@gitlab.iri.upc.edu:2202/labrobotica/ros/
    sensors/iri_htc_vive_tracker.git
$ roscd
$ cd ..
$ catkin_make --only-pkg-with-deps iri_htc_vive_tracker
```

5.2 Publishers

5.2.1 vo (navigation_msgs/Odometry)

Visual odometry of the device that we are interested in. This is used by the robot_localization_node to do the Kalman Filter.

5.2.2 new_pose (geometry_msgs/PoseStamped)

Pose of the device that we want the WAM arm to follow.

5.3 Subscribers

5.3.1 filtered_odometry (navigation_msgs/Odometry)

Filtered odometry that is received from the robot_localization_node after performing the Kalman Filter. This value is used to obtain the current Pose of the device.

5.4 Services

5.4.1 trigger_pulse (iri_htc_vive_tracker/TriggerHapticPulse.srv)

When calling this service with the device name, a haptic pulse will be triggered on the device if possible. The duration of this pulse is a configurable parameter.

5.4.2 button_pressed (iri_htc_vive_tracker/GetButtonPressed.srv)

When calling this service with the device name, we will obtain the value of the last button that was pressed on the device.

5.5 tf transforms

This node will publish the position of all detected devices, as well as the position of the link boxes, in the form of a tf tree. The source frame is called the **chaperone**, and is defined when performing the room setup on Steam. The tf tree is shown in Figure 4

5.6 Launch files

There are three launch files included in this node, which are the following.

- **htc_vive.launch**: Basic launch file which allows to view the tf of the htc vive system.
- **wam_follow_device.launch**: Extended launch file which also launches the robot localization node, and remaps the *new_pose* topic so that the wam arm follows the device. By default, it will follow the *tracker_1*, and it can be changed, for example to follow a controller, by adding *device:=controller_1* in the command line.
- **publish_wam_chaperone_link.launch**: File that publishes the static transform between the link base of the WAM and the chaperone frame.

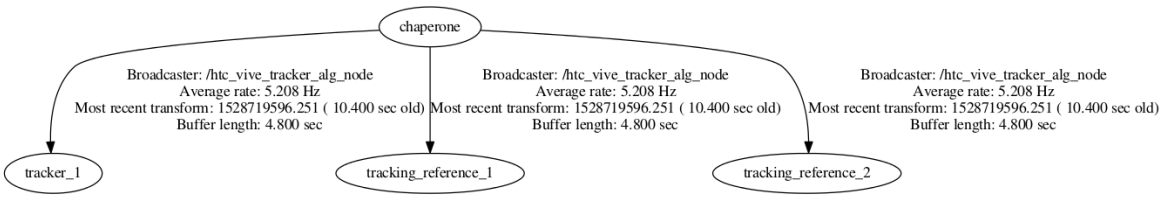


Figure 4: tf tree obtained by the ROS node

In order to have everything working properly, we need to find the relationship between the world in HTC vive and other reference systems (such as the WAM arm or the TIAGo robot). To do this, we will perform a Hand-Eye calibration.

6 Hand Eye Calibration

The first step for hand-eye calibration is to record a rosbag of movements. We need to secure the HTC Vive tracker on the robot, and move it to different positions. When moving the robot, make sure that each of the positions it is moved to is different from the rest, so that the system can fully calculate the transform between frames. Figure 5 shows all the transforms in a hand-eye calibration system. In blue we have the known transforms, and in red the unknown transforms which we want to calculate.

Before recording the bag file, make sure that the computers used for this are synchronized, because a slight offset of just a few seconds will prevent from obtaining a good calibration.

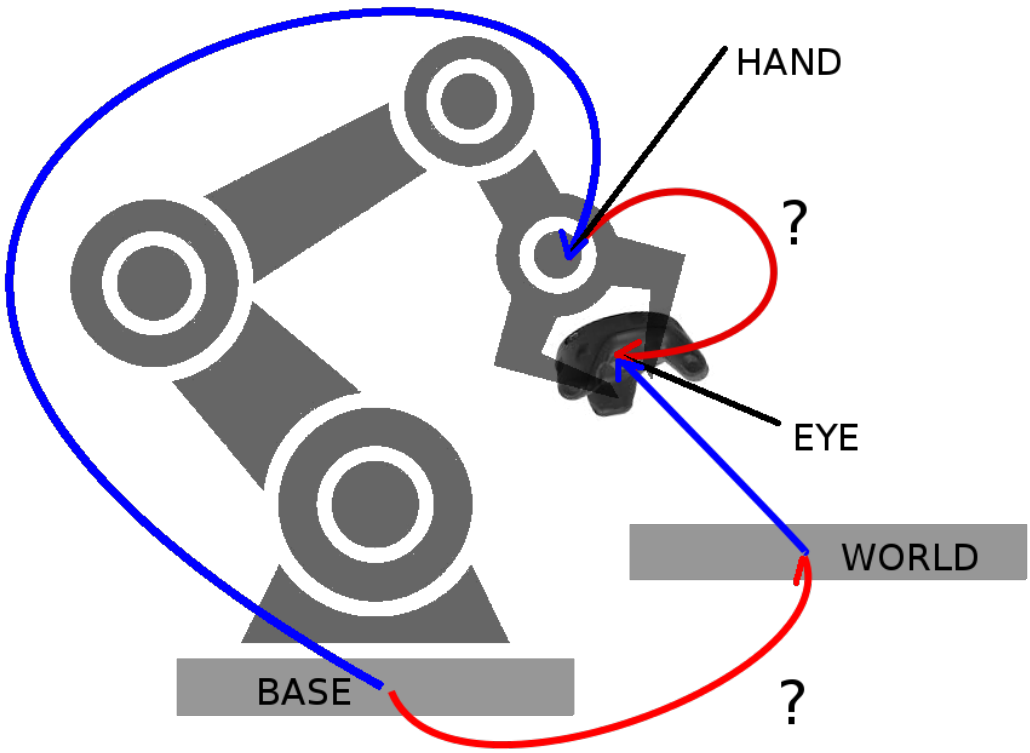


Figure 5: Transformations (known and unknown) in hand-eye calibration

The goal is to obtain the transform from BASE to WORLD, which are the two fixed frames in our environment. Once the rosbag is recorded, we use MATLAB to find the unknown transformations.

The gitlab repository is called HandEyeCalibrationMatlab. We download and run this code in matlab as follows:

```
[baseWorld,handEye] = completeCalibration("rosbag_file.bag", "
    robot_base_link","robot_hand_link","world_link","eye_link");
```

In the case of HTC Vive and WAM arm, the instruction would be as follows:

```
[baseWorld,handEye] = completeCalibration("rosbag_file.bag", "
    iri_wam_link_base","iri_wam_link_7","chaperone","tracker_1");
```

This will return a translation and quaternion, which defines the transform from the robot base link to the world link. We will take this values and publish them as a static transform.

7 Following the tracker

One application for this system is to have the WAM arm follow the HTC Vive tracker. To do this, the steps are the following.

1. Perform the hand eye calibration between the WAM and the chaperone. Check Section 6 on how to do it.
2. Modify the **publish_wam_chaperone_link.launch** so that it publishes the transform obtained from hand eye calibration. Recall that in MATLAB the order of quaternions is $\langle q_w, q_x, q_y, q_z \rangle$. However, in ROS, when publishing the static transform, the order must be $\langle q_x, q_y, q_z, q_w \rangle$
3. On the system that is connected to the WAM, run

   ```
   roslaunch iri_wam_bringup iri_wam_bringup.launch
   roslaunch iri_wam_dmp_tracker iri_wam_dmp_tracker.launch
   ```

4. On the system in which SteamVR is installed and running, run the following:

   ```
   roslaunch iri_htc_vive_tracker wam_follow_device.launch
   ```

The WAM will start following the tracker.

8 Conclusions

The software presented in this report has been tested with the WAM arm. In order to check that the robustness of the result, the tracking example was tested in two different locations, performing the hand-eye calibration in each of those.

It was observed that when performing the tracking of the device there was a slight offset in the position received by the tracker. The possible reasons for this are the following: first, the fact that when performing the hand eye calibration, the base of the WAM arm wasn't completely secured: a slight jitter may have caused an error on the transform from the base link to the chaperone, and this translates to an error on the tracker position. Secondly, the cables in the WAM robot weren't correctly tightened, and finally, when the tracker is moving, there is less accuracy when measuring its position. Further work could include analyzing if the calibration is more accurate when recording only the tf of the system when it is not moving, and checking that the robot cables are correctly tightened and its base is fix.

Besides the calibration error, the system works in a robust way and it can be easily adapted to interact with other robots.

A Useful resources for more information

If you want to learn more about the HTC Vive Tracker and its possible usages, check out the developer guidelines. There's also the steam community which has several forums where people share their issues with SteamVR and SteamVR developer hardware.

References

Acknowledgements

IRI reports

This report is in the series of IRI technical reports.
All IRI technical reports are available for download at the IRI website
`http://www.iri.upc.edu`.