**Visualizing the point cloud**

To complete the steps described in this document you will need to have a (virtual) machine installed with Ubuntu 18.04 and ROS Melodic. An image for Ubuntu 18.04 can be found here:

<https://releases.ubuntu.com/18.04/>

And a tutorial on how to install ROS here:

<http://wiki.ros.org/melodic/Installation/Ubuntu>

From the Mechatronica team we have received a ROSbag which contains data that can be turned into a point cloud. To do this we also received a file called stereo\_image\_proc.launch. This file performs the duties of image\_proc for both cameras, undistorting and colorizing the raw images. Note that for properly calibrated stereo cameras, undistortion is actually combined with rectification, transforming the images so that their scanlines line up for fast stereo processing. In this case we will use it to determine the positions of the points of our point cloud.

In order to do this we need to make a catkin workspace in our Ubuntu environment. A catkin workspace is a folder where you modify, build, and install catkin packages. A catkin workspace can contain up to four different spaces which each serve a different role in the software development process. The **source space** contains the source code of catkin packages. This is where you can extract/checkout/clone source code for the packages you want to build. Each folder within the source space contains one or more catkin packages. The **build space** is where CMake is invoked to build the catkin packages in the source space. CMake and catkin keep their cache information and other intermediate files here. The build space does not have to be contained within the workspace nor does it have to be outside of the source space, but this is recommended. The **development space** (or devel space) is where built targets are placed prior to being installed. The way targets are organized in the devel space is the same as their layout when they are installed. This provides a useful testing and development environment which does not require invoking the installation step. When ever referring to a folder which can either be a development space or an install space the generic term **result space** is used. We can make a catkin workspace by simply running the following commands:

$ mkdir -p ~/catkin\_ws/src

$ cd ~/catkin\_ws/

$ catkin\_make

Additionally, if you look in your current directory you should now have a 'build' and 'devel' folder. Inside the 'devel' folder you can see that there are now several setup.\*sh files. Sourcing any of these files will overlay this workspace on top of your environment.

$ source devel/setup.bash

To make sure your workspace is properly overlayed by the setup script, make sure the ROS\_PACKAGE\_PATH environment variable includes the directory you're in.

$ echo $ROS\_PACKAGE\_PATH

/home/youruser/catkin\_ws/src:/opt/ros/melodic/share

Next up, to use this workspace we need to create a package. For a package to be considered a catkin package it must meet a few requirements:

* The package must contain a [catkin compliant package.xml](http://wiki.ros.org/catkin/package.xml) file.
  + That package.xml file provides meta information about the package.
* The package must contain a [CMakeLists.txt which uses catkin](http://wiki.ros.org/catkin/CMakeLists.txt).
  + If it is a [catkin metapackage](http://wiki.ros.org/catkin/package.xml#Metapackages) it must have the relevant boilerplate CMakeLists.txt file.
* Each package must have its own folder
  + This means no nested packages nor multiple packages sharing the same directory.

The simplest possible package might have a structure which looks like this:

my\_package/

CMakeLists.txt

package.xml

To create a new package, first we need to change to the source space directory of the catkin workspace:

$ cd ~/catkin\_ws/src

Now use the catkin\_create\_pkg script to create a new package, you can also add dependencies to the package at this step. We will add the stereo\_image\_proc dependency. This command looks like this:

$ catkin\_create\_pkg pointcloud\_visualizer std\_msgs rospy roscpp stereo\_image\_proc

Now that we have created the package we cd back to the workspace root folder so that we can remake the workspace with the package:

$ cd ~/catkin\_ws

$ catkin\_make

We recommend creating a new folder inside the package directory called launch, in this folder we will deposit the ROSbag file as well as the stereo\_image\_proc.launch file. Now we are ready to visualize the point cloud from the ROSbag.

We need to start with running ROScore by running the following command:

$ roscore

This step is not necessary when the ROSbridge has already been launched.

Then, open RViz by running the following command:

$ rosrun rviz rviz

Then, in a separate terminal run the stereo\_image\_proc from the package. You can do this as such:

$ cd ~/catkin\_ws/src/pointcloud\_visualizer

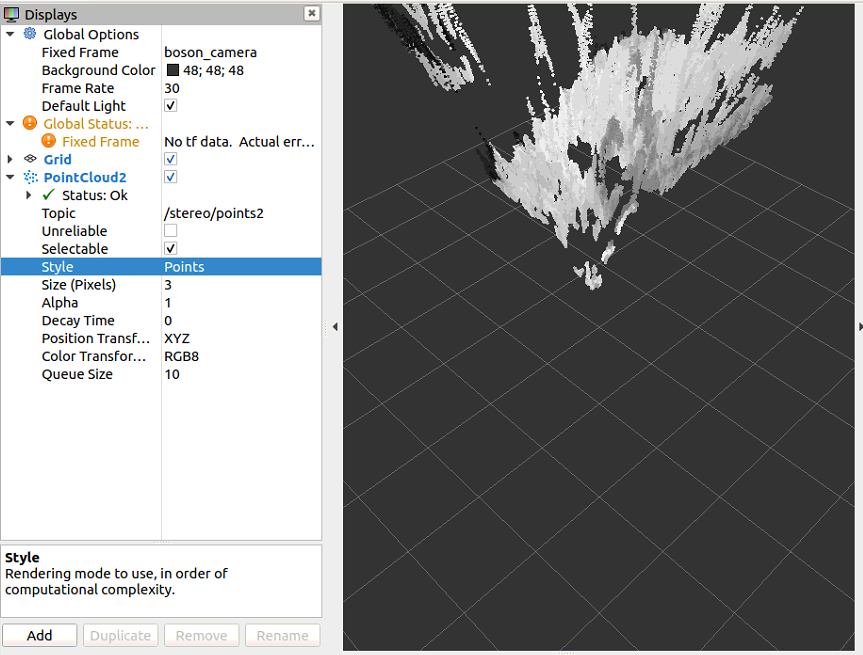
$ roslaunch pointcloud\_visualizer stereo\_image\_proc.launch

Be sure that you sourced the setup.bash of your workspace, otherwise you will not be able to use the package or the launch file.

And lastly, we can run the ROSbag. We need to do this from a different terminal as well:

$ rosbag play [name of rosbag]

Now we need to open up RViz and click the “add” button on the lower left hand corner of the screen. Click on the tab “by topic” and scroll down to the topic stereo/points2, and add “PointCloud2”. Now, the last thing we need to do is change the “Fixed Frame”, which you can find in the upper left hand corner of the screen. In this example we need to change it to “boson\_camera”, but this can change depending on what ROSbag you are using. At this point RViz should visualize the point cloud, which looks like this:



Sources:

<http://wiki.ros.org/stereo_image_proc>

<http://wiki.ros.org/catkin/workspaces>

<http://wiki.ros.org/catkin/Tutorials/create_a_workspace>

<http://wiki.ros.org/ROS/Tutorials/CreatingPackage>

**Streaming Data**

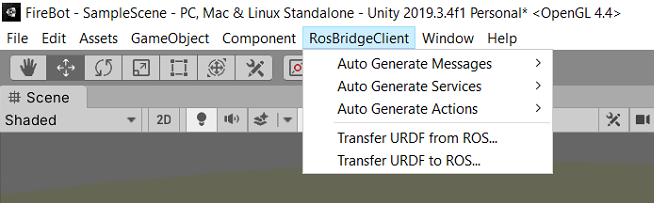
To start streaming data from ROS to Unity, we will need a connection between the two. For this we will use ROS#. ROS# is a set of open source software libraries and tools in C# for communicating with ROS from .NET applications. ROS# has an extensive tutorial on how to install all the necessary software needed to use ROS# that can be found on their repository page. However, for what we want to use ROS# for we don’t need to do all of these tutorials. Instead we can limit ourselves to only a couple of steps.

First of all, create a new Unity project and copy the “RosSharp” folder from the ROS# GitHub repository (ros-sharp/Unity3D/Assets/RosSharp) into your Unity project’s assets folder. To subscribe to and render point clouds, we are also going to make use of a GitHub repository called “unity\_assets”. This repository contains scripts that expand on ROS# and allows us to subscribe to and render a point cloud. So also add the PointCloudStreaming folder from this repository (unity\_assets/PointCloudStreaming) to your Unity assets folder.

Make sure that Unity is using .NET Framework 4.x, since it is required by RosBridgeClient. To do this:

* In the Unity menu, go to Edit > Project Settings > Player.
* In the Inspector pane, look under Other Settings > Configuration.
* Set Scripting Runtime Version\* to .Net 4.x Equivalent.

Now RosBridgeClient and UrdfImporter are included in your Unity project. Once the plugins have been loaded, the following new menu items will show up:



Next up, we need to set up our Unity objects. To see the point cloud in Unity, we will need the following objects:

* RosConnector:

This is an empty object, inside this object we will add 2 scripts:

* Ros Connector:

This is a ROS# script that will connect the Unity project to the ROSbridge. Make sure that the server URL is inputted correctly.

* Point Cloud Subscriber:

This script subscribes our Unity project to a point cloud topic. Make sure the topic is the correct topic you want to subscribe to.

* Renderer:

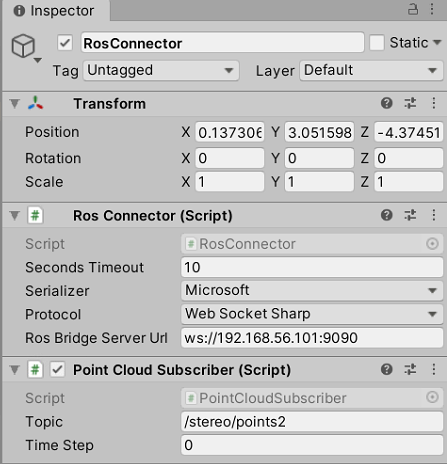
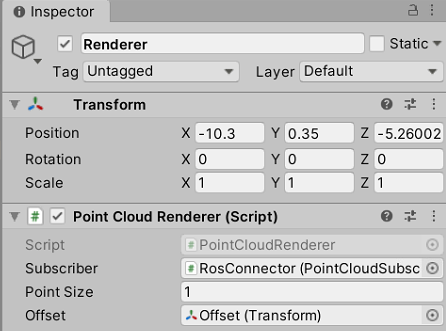
This is also an empty object with just 1 script:

* Point Cloud Renderer:

This script renders the point cloud once we receive data from the ROSbridge. It will spawn the point cloud in the position of the last object. This script is also linked to the PointCloudSubscriber inside of the ROS Connector.

* Offset:

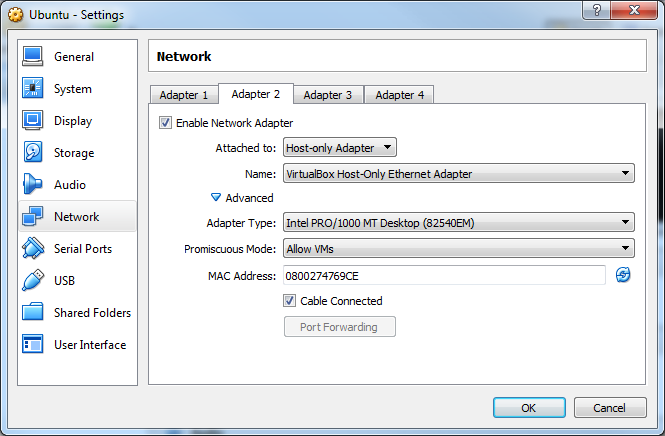
This item is linked to the renderer and will act as the base for the point cloud once it’s rendered.



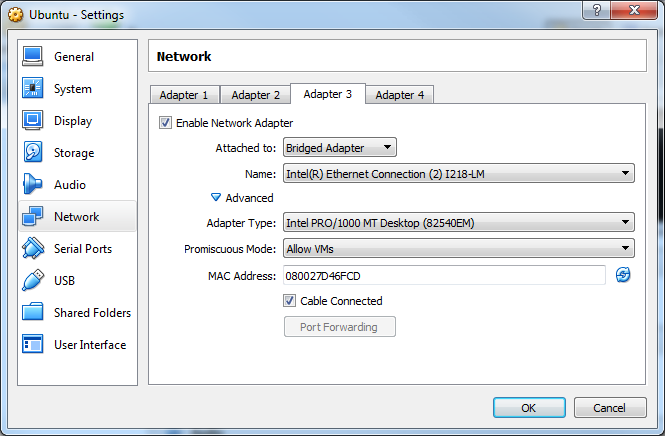
And finally make sure that you have added the “Enable Open GL” script to the camera.

For the next step we will take a look at our Ubuntu machine. In our case we use virtual machines. If you are not using a virtual machine but an actual Ubuntu machine you can skip this step. In Oracle VirtualBox, make sure the Ubuntu VM is powered off. Then open the settings for the Ubuntu VM. In the Network tab, add two new network adapters:

* Host-only adapter:



* Network bridge:



These settings are needed so that the RosBridgeClient running in Windows, and the ROSBridge Server running on Ubuntu can communicate. In Ubuntu type: $ ifconfig to check the network configuration and to verify your Ethernet connection to the Windows OS. The IP address (enp0s8) of the Ubuntu system will be used by rosbridge\_suite and RosBridgeClient.

Now we need to set up the connection from the ROS side. Luckily, we don’t need to write our own publisher and subscribers, instead we will make use of the ROSBridge. This is a part of ROS and is an easy way of establishing a connection between ROS and Unity. To install ROSBridge the user simply has to enter the following command into the terminal:

$ sudo apt-get install ros-<rosdistro>-rosbridge-suite

<rosdistro> is the distribution of ROS the user is using, for our project it is “melodic”.

To now run the ROSBridge the user needs to first source the setup of their ROS directory:

$ source /opt/ros/<rosdistro>/setup.bash

Now all the user has to do is run the following command:

$ roslaunch rosbridge\_server rosbridge\_websocket.launch

And that is it, ROSBridge will now launch and create a websocket on port 9090 by default. Now it’s time to play the Unity project, if everything went well you should see the following messages:

* In Unity:



* In ROS:



It is important that it notifies you of the subscription as well as the connection. If one of these two messages is missing something has gone wrong.

Now that the connection is established, it is time to start streaming data to Unity. In a separate terminal, navigate to your custom made package and run the command to generate a point cloud from the ROSbag:

$ cd ~/catkin\_ws/src/pointcloud\_visualizer

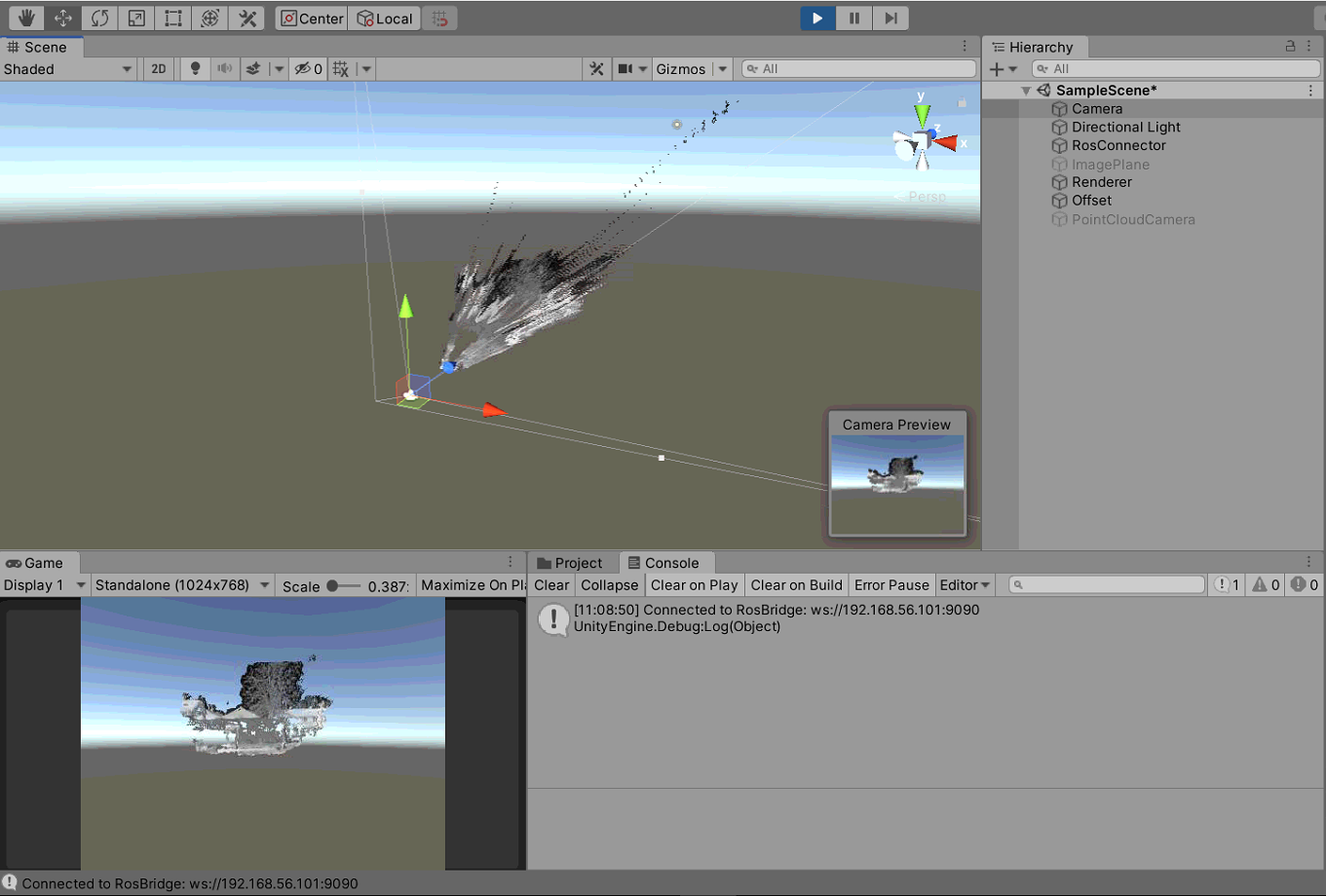
$ roslaunch pointcloud\_visualizer stereo\_image\_proc.launch

While the stereo\_image\_proc.launch is getting launched, we can also start playing the ROSbag by playing it from the “launch” directory:

$ cd ..

$ rosbag play [name of rosbag]

Now, when you open up the Unity project you should be able to see the point cloud, which looks like this:



Sources:

<https://github.com/inmo-jang/unity_assets/tree/master/PointCloudStreaming>

<https://github.com/siemens/ros-sharp>

<https://github.com/siemens/ros-sharp/wiki/User_Inst_Unity3DOnWindows>

<https://github.com/siemens/ros-sharp/wiki/User_Inst_UbuntuOnOracleVM>

<http://wiki.ros.org/rosbridge_suite/Tutorials/RunningRosbridge>