UnrealROX Documentation

Release 0.0.1

Pablo Martinez-Gonzalez, Sergiu Oprea, Alberto Garcia-Garcia, Se

Nov 16, 2018

Getting Started

1 Welcome **1**
- 1.1 What is UnrealROX? . . . 1
- 1.2 Our motivation . . . 1
- 1.3 UnrealROX features . . . 1
- 1.4 Our contribution . . . 2

2 How to use UnrealROX **3**

3 Project structure **5**

4 Hardware and Software prerequisites **9**

5 Installation **11**

6 Scene configuration **13**
- 6.1 Import scene to UnrealROX . . . 13
- 6.2 Do you need a robot for you scene? . . . 13
- 6.3 Check and configure scene objects! . . . 14
- 6.4 Build scene lighting . . . 14

7 VR Controllers **17**
- 7.1 OculusVR . . . 17
- 7.2 HTC Vive PRO . . . 19

8 Recording **21**
- 8.1 General configuration . . . 21
- 8.2 Configure HMD position . . . 22
- 8.3 Begin/Stop recording . . . 22

9 Playback **23**
- 9.1 Convert recorded sequences to JSON . . . 23
- 9.2 Configure playback process . . . 23
- 9.3 Run playback process . . . 24

10 Troubleshooting **25**

11 Changelog **27**

12 Contact **29**
12.1 Team Members 29

13 Indices and tables **31**

CHAPTER 1

Welcome

1.1 What is UnrealROX?

UnrealROX is an extremely photorealistic virtual reality environment built over Unreal Engine 4 and designed for generating synthetic data for various robotic vision tasks. This virtual reality environment enables robotic vision researchers to generate realistic and visually plausible data with full ground truth for a wide variety of problems such as class and instance semantic segmentation, object detection, depth estimation, visual grasping, navigation, and more.

1.2 Our motivation

State-of-the-art deep learning architectures need large amounts of accurately annotated data for achieving a good performance. The lack of large-scale datasets which provide accurate ground truth (e.g. semantic segmentation, depth and normal maps, etc.) is mainly because data annotation task is tedious and time-consuming. Because of that, photorealistic virtual reality environments are becoming increasingly popular and widely used to generate huge amounts of data with an accurate ground truth.

Taking advantage of this trend, we aim to provide a large-scale photorealistic dataset of indoor scenes where different robots can explore, manipulate, and interact with different objects.

1.3 UnrealROX features

1. Grasping system for robot manipulation is independent of the object geometry and orientation, allowing the robot to adopt different finger configurations.
2. Available for both Oculus Rift and HTC Vive Pro.
3. After recording, all scene, robot and camera information can be exported to a JSON file from Unreal Engine 4 (UE4)
4. With the JSON file and the unchanged scene where the sequence was recorded, you are able to generate the data you need.

5. Scene cameras are fully configurable. You can place different cameras in the scene and also attach them to specific robot joints.
6. This project is open-source. Anyone is welcome to contribute!

1.4 Our contribution

To the best of our knowledge, UnrealROX is the first photorealistic virtual reality environment based on Unreal Engine 4 and used for generating synthetic data for various robotic vision tasks.

CHAPTER 2

How to use UnrealROX

The main purpose of this section is teaching you yo successfully use UnrealROX with your photorrealistic scene![1].

To successfully use UnrealROX you need to perform the following steps:

1. **Check for hardware and software prerrequisites**: please check and satisfy all the *Hardware and Software prerequisites*.
2. **UnrealROX installation**: consists basically on clone or download the UnrealROX github. For more details go to *Installation* section.
3. **Configure your scene**: get or create your own awesome and photorrealistic scene. Migrate it to the UnrealROX project. Put a robot in your scene and choose the objects to interact with. Configure the objects properly and finally build scene lighting. The step by step guide for the scene configuration can be found in *Scene configuration* section.
4. **Learn how to control the robot**: go to *VR Controllers* section and see the input map for your VR hardware.
5. **Record your first sequence**: configure the recording process and start recording your first sequence. For knowing how to do this, go to *Recording* section.
6. **Get your perfect data**: generate the ground truth for the recorded sequences. Go to *Playback* section and finish this tutorial!

- For any question/request or contribute to UnrealROX project and make it grow, please contact any of the authors of this project.
- To report any bug or issue, please use Github issue tracker.

Thanks for using UnrealROX project!

[1] this tutorial was done with Unreal Engine 4.18. We cannot guarantee UnrealROX work properly with other UE version.

CHAPTER 3

Project structure

In the following figure we can see the directory structure of UnrealROX:

- **docs** folder contains all the UnrealROX documentation generated with Sphinx.
- **Config** folder contains main configuration files for setting values that control engine behavior. Values set in the game project Config files override the values set in the Engine/Config directory. Config files are generated by default when creating a blank UE4 project.
- **Content** folder contains content for the engine or game including asset packages and maps. Here you can find subfolders such as:
  - **Mannequin** which contains UE4 mannequin's annimation, material, meshes and texture assets.
  - **Maps** which contains a basic scene where you can graps objects with basic geometries and test robot's behaviour. Here you should migrate your photorrealistic scene!
- **Plugins** folder contains plugins used in the engine.
- **Source** folder contains source files with the all project's implementation. Code is structured in:
  - **TODO**
- **RecordedSequences** folder which contain the sequences you have recorded which are stored in *.txt* files.
- **GeneratedSequences** folder which contain the ground truth generated for the recorded sequences. In Figure 2 we can see the directory structure of the generated data. At the root (*vienese_001*) we have the sequence name which is composed by the *scene name* (e.g. vienese) and the number of sequence (e.g. 001 because it is the first sequence). Then we create a folder for each data modality, in which we store the frames organized in directories for each one of the cameras.

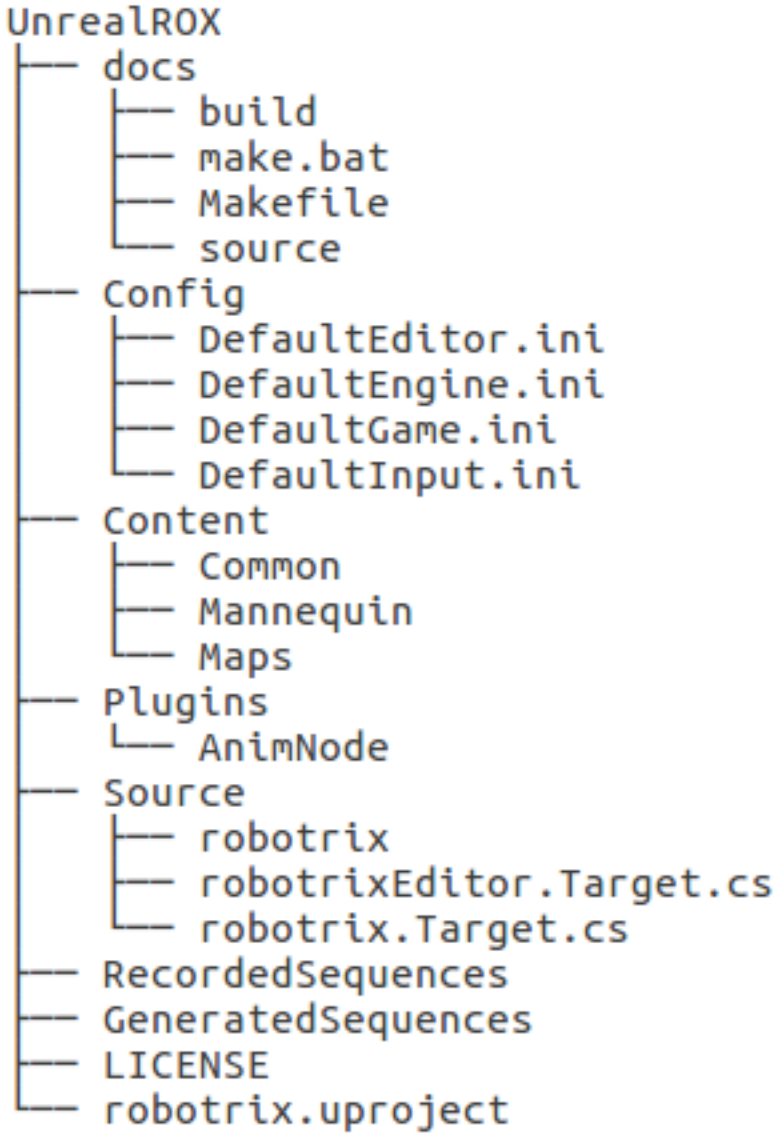


Fig. 1: Figure 1. UnrealROX project directory tree.

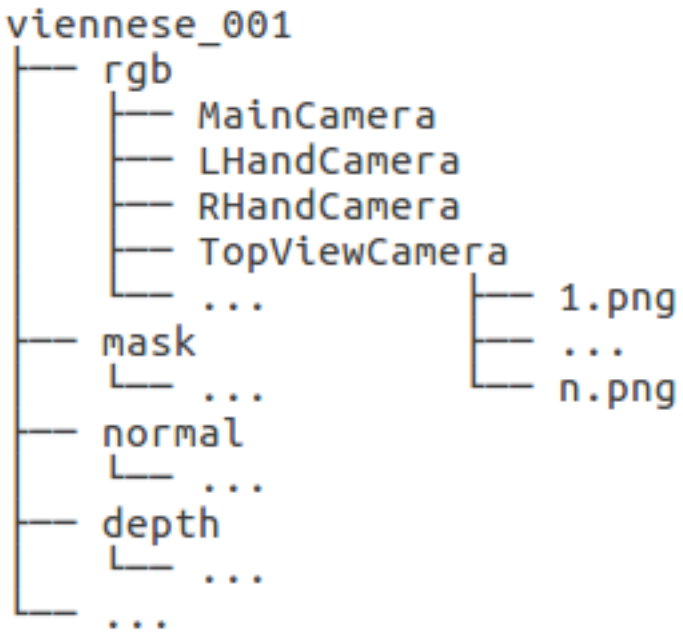


Fig. 2: Figure 2. Ground truth directory tree.

CHAPTER 4

Hardware and Software prerequisites

First of all and before installation, you will need to statisfy some hardware and software prerequisites:

1. **Software:** For the development of UnrealROX we used Windows 10 OS. UnrealROX was not tested on Linux or MacOS. Software requirements are the following:

   1.1 **VR software**. You have to install either the Oculus or HTC Vive Pro software, or both.

   1.1.1 **Oculus SDK**. Download and install Oculus SDK for Windows. Oculus SDK version used for the development of UnrealROX was 1.30.0 and published on 08/10/2018.

   1.1.2 **HTC Vive Pro SDK**. Download and install Vive Software.

   1.2 **Visual Studio**. In order to compile UnrealROX and Unreal Engine you will need Visual Studio. For this purpose you need to install Visual Studio 2017 with the configuration file we provide (put link). In the Visual Studio Installer you can import a VS configuration file and all required individual packages and workloads will be automatically installed. This is important in order to avoid some errors when compiling UnrealROX software or different versions of Unreal Engine.

   1.3 **Unreal Engine 4.18 or higher**. UnrealROX was originally implemented using UE 4.18 version (implementation on 4.19 and 4.20 comming soon). UE4 installation can be done in two ways depending if you need the mask segmentations of your recorded data.

   1.3.1 **I don't need mask segmentations**. In this case you will only need to follow Unreal Engine official instalation guide and install the compatible UE4 version with UnrealROX.

   1.3.2 **I need mask segmentations**. In order to get mask segmentations from UE4 you need to recompile the complete engine from scratch. The first step is to get the source code for your UE4 version following the Unreal Engine get source code guide (you should use the release branch in the Unreal Engine github repo). In the aforementioned guide there are also compilation instructions depending on your operating system. Before compiling the engine do the following:

   - Go to */Engine/Source/Runtime/Engine/Private/StaticMeshRender.cpp* and search for *if (bProxyIsSelected && EngineShowFlags.VertexColors && AllowDebugViewmodes())* condition (Line 982 for UE 4.18, line 1020 for UE 4.19 and line 1063 for UE 4.20).
   - Remove **bProxyIsSelected** flag. The presence of this flag disables the visualization of per vertex coloring.

- Compile engine code with Visual Studio 2017 or higher.

2. **Hardware**:

   2.1. **GPU**. Make sure your GPU driver is well installed and updated. You need a good GPU to run smoothly a photorealistic scene alongside UnrealROX system. We used a Titan X GPU.

   2.2 **Overall hardware requirements**. For a smooth experience in photorealistic virtual environments rendered by Unreal Engine we recommend a good performance hardware configuration.

   2.3 **VR headset**. Check if Oculus VR and/or HTC Vive PRO perform properly and if their installation and calibration was done correctly to achieve a good tracking.

3. **You are now ready to install UnrealROX, import your photorrealistic scene and create your own awesome dataset!**

CHAPTER 5

Installation

After checking *Hardware and Software prerequisites* you can proceed with the installation of UnrealROX. You will only need to clone or download UnrealROX github and that's all. UnrealROX is a UE4 project you can easily compile with Visual Studio.

Before the compilation make sure UnrealROX is selected in Visual Studio as *entry point* in order to avoid compiling the engine! Also you should run the project in *Development Editor* mode to achieve a good performance.

When first openning the UnrealROX project you might get the following message:

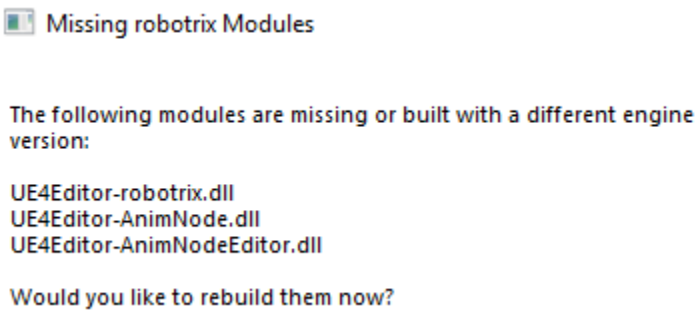


Fig. 1: Figure 1. You need to rebuild the required modules in order to run the UnrealROX project.

Rebuild the required modules in order to properly run the UnrealROX project.

CHAPTER 6

Scene configuration

Now you have the UnrealROX project you will learn how to use it properly. First of all you need to get a photorrealistic scene or create your own. The scenes we used are the following:

- Interactive House
- Viennese Apartment
- Xoio Berlin Flat
- Hamburg House
- Studio Apartment
- etc.

6.1 Import scene to UnrealROX

To import your scene to the UnrealROX project you need to:

1. **Migrate your scene to UnrealROX**. On the one hand we have the UnrealROX project to which we want to import the scene from other UE4 project. Open your scene project and go to the Content Browser and localize *.umap* file of your scene, *right-click* and go to *Asset Actions->Migrate*. Now you need to navigate to the *Content* folder of UnrealROX project and select it as target for migrating your scene.

6.2 Do you need a robot for you scene?

1. **(Experimental) Import your own robot**. We are currently working on this feature (import a robot using its *URDF* robot model). In order to import your own robot you need the corresponding meshes, textures and skeleton. You also need to define constraints to control robot's movement and physics simulation. Then you need to configure the grasping system placing some *triggers* (e.g. *sphere triggers*) on robot's fingers/gripper and also code a new logic for the new hand or gripper.

1. **Put the robot in your scene**. From this point we will only work on the UnrealROX project. In the UE4 editor go to the Content Browser and navigate to *Mannequin->Meshes* selecting your pawn (e.g. *ROXMannequinPawn*) and drag it to the scene. You can scale your robot to match with your scene scale.

6.3 Check and configure scene objects!

You need to check object pivots are placed correctly in order to properly simulate physics during interaction. You also need to configure the objects you will interact with. To do this, following the next steps:

1. **Check object pivots**. Run your scene and check that all object pivots are placed at the lowest geometric center of the mesh. The mesh center is determined by the X and Y axes values, meanwhile the Z axis value should be the lowest of the mesh. This is important to track all the objects correctly during the recording and playback steps and also for the physics simulation during interaction. However, this is a tedious task thus realistic scenes have lots of objects. Due to this, we use the plugin Pivot Tool which works like a charm (we don't include it in the project because of license conflicts). We highly recommend to use Pivot Tool, thus you can configure all the objects in seconds.
2. **Configure interactable objects**. Choose the object you want to interact with from the World Outlier (by default placed on the right side in the UE4 editor) and do the following:

    2.1. **You need object to be movable**. Go to *Transform->Mobility* and set the object to *Movable*. Almost all the objects are static by default.

    2.2. **You need physics simulation**. Go to *Physics* and check *Simulate Physics* option.

    2.3. **You need overlap events**. Go to *Collision* and check *Generate Overlap Events* option. **This is a must** for the grasping system in order to grab an object correctly. If this option is disabled, grasping wouldn't work.

    2.4. **You need an accurate collision mesh**. You also need to check object geometry to achieve a visually plausible grasping. *Right-click* on the object and *Edit*. In the *Object Editor* you need to visualize the simple and complex collision meshes and visually check its accuracy. If object geometry is complex and the collision mesh is rough, you should improve this by auto generating convex collision (go to editor menubar and *Collision->Auto Convex Collision*) with maximum hull verts and accuracy. For the objects with a complex geometry you should set the *Collision Complexity* to *Use simple collision as complex*. In this way you will achieve a more realistic grasping, however, physics simulation will be much more complicated so when interacting with two or more objects you may notice an unstable behavior.

6.4 Build scene lighting

Lighting configuration is a time-consuming step. In order to optimize this process pay attention to the general scene lightning configuration.

1. **Build scene lighting**. Now you need to build project lighting! The steps above are very important, especially the step of setting the objects as movable. You need to set lighting configuration according to your PC specs. By default, lighting configuration is too demanding for a mid-high range computer. For a fast and feasible lighting generation we recommend the following configuration you can do in the *World Settings->Lightmass*. See the Figure 1.

Scene ready to record your own sequences!



Fig. 1: Figure 1. Scene light configuration we used to build lighting and produce photorealistic results.

CHAPTER 7

VR Controllers

In this section we will describe the input map for both OculusVR and HTC Vive PRO controllers.

7.1 OculusVR

In the following figure (Figure 2) we can see a representation of Oculus controllers with the functions assigned to each of the buttons.

Using:

- **Left and Right Joysticks:**
  - to move and orient the robot in the scene
  - (by first pressing the left joystick) user will be able to position the first person camera according to its height. Use right joystick to move on Z axis and left joystick for movement on X and Y axes[1].
- **Left and Right Grasp**: grab an object with the left or right hand correspondly.
- **Y button**: restart the level placing all the objects to its initial position. First person camera configuration is mantained.
- **X button**: reset VR changing first person camera to its default position and configuration
- **B button**: turn ON/OFF HUD used for debugging purposes. It enables a mirror to see better robot head position while configuring first person camera
- **A button**: begin/stop recording process which will dump all the scene information to a .txt file. This file will be used for the playback process.
- **Unused buttons**: oculus button and left and right triggers.

[1] Robot head is attached to the VR headset tracking user's head position. This entails some problems such as, user's height. You will need to configure camera position according with your height before recording.

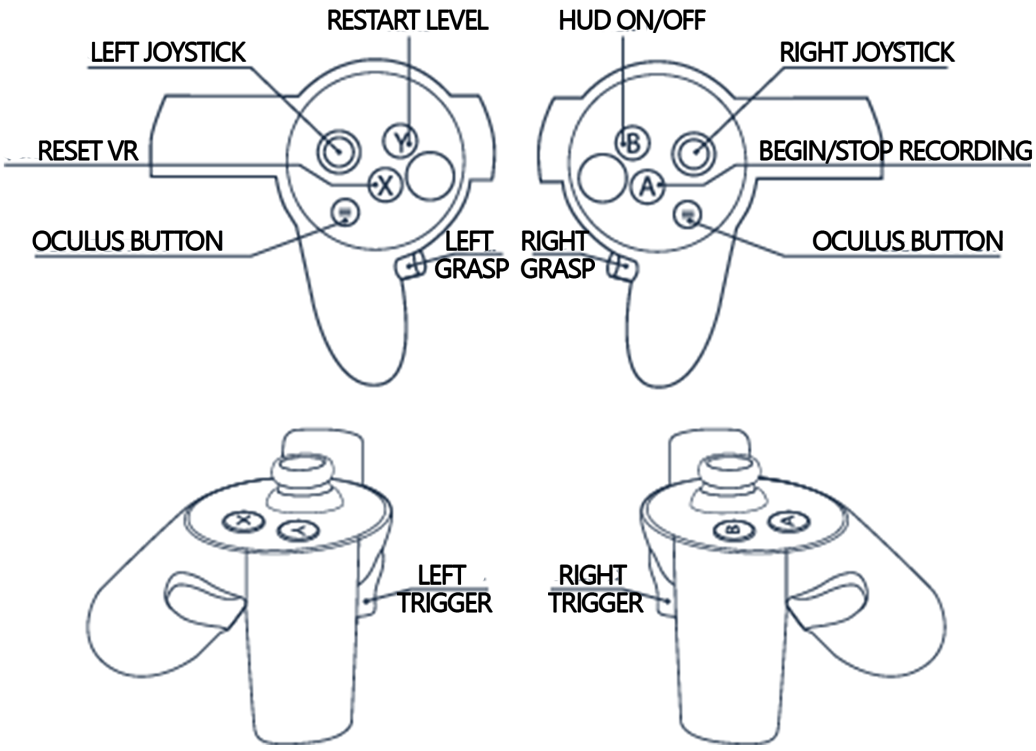


Fig. 1: Figure 2. Oculus controllers representation with the correspondence between the robot actions and cotroller buttons.

7.2 HTC Vive PRO

TODO: HTC Vive PRO controllers

CHAPTER 8

Recording

During the recording process all scene information such as, camera location, its configuration and objects pose and orientation, among others, are dumped in a *.txt* file which is stored by default on the *RecordedSequences* folder located in the root of the UnrealROX project. Recording process is the same regardless of used VR hardware. VR controllers have their own input map, so there are different instructions for each of the used hardware.

8.1 General configuration

To configure recording process you need to search for *ROXTracker* in the *World Outliner* panel from UE4 editor. *ROXTracker* contains general, recording and playback configuration options. *Figure 1 and 2* represent general and specific recording parameters.

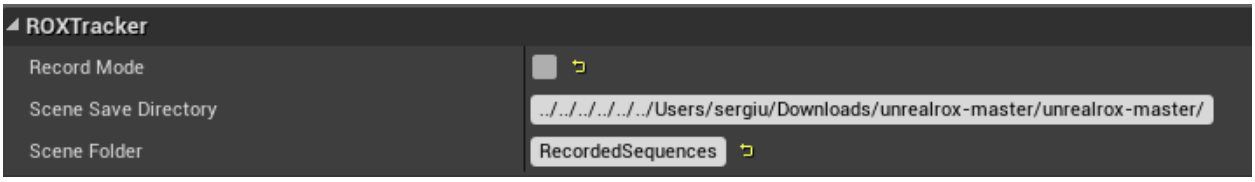


Fig. 1: Figure 1. General parameters for recording and playback steps. Check *Record Mode* if you want to record your sequence.

The default path for the *Scene Save Directory* is the project's root directory. *Scene Folder* indicates the folder where recorded sequences are stored. You can freely change these parameters.

Before start recording you should configure the following parameters:

- **Record mode**: you should check this option if you want to record your own sequences (*see Figure 1*).
- **Select scene cameras**: you should select scene cameras which you want to capture data. Go to *Recording->Camera Actors* and add the cameras you want to use during recording (*see Figure 2*).
- **Scene file name prefix**: you can also configure the prefix filename for the *.txt* files with the recorded sequences. Default: scene.

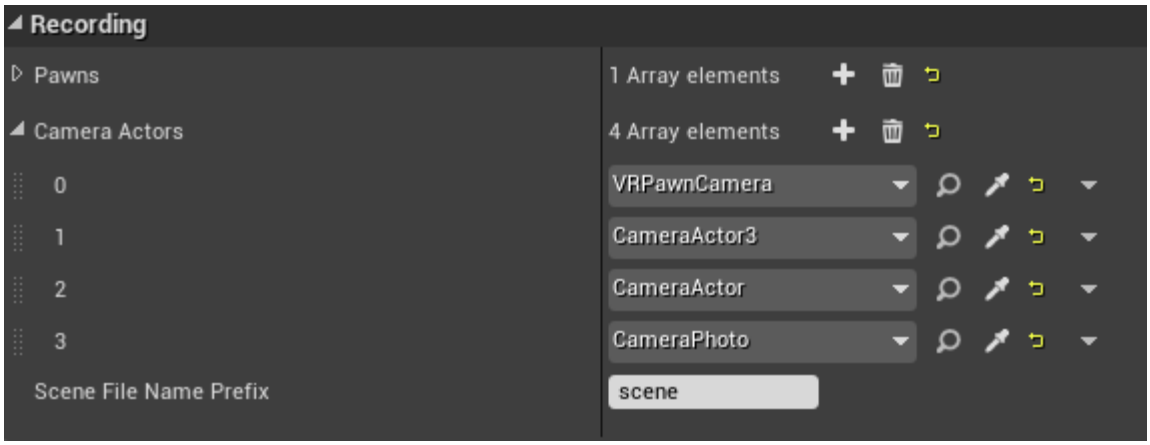


Fig. 2: Figure 2. Specific configurable parameters for recording process.

8.2 Configure HMD position

Before recording you need to check for the **HMD position**. You need to check if your HMD and its attached camera are in the correct position. This is variable according to user's height, so, you need to manually configure your main camera position (the main camera is what you see on your Oculus headset). For the configuration[1]:

- **with OculusVR**: you need to press *left joystick* to enable HMD position changes. You are now able to change camera position along Z axis using right joystick and also camera translation on X and Y axes with the *left joystick*. A good tip for adjusting camera position is checking if shoulders position is corresponding with user's real shoulder position. User's should also be able to see its chest by looking down.
- **with HTC Vive PRO**: TODO

8.3 Begin/Stop recording

To begin/stop recording, user need to:

- **with OculusVR**: press *A button* from the right controller. Press again to stop recording.
- **with HTC Vive PRO**: TODO

A red collored message will appear both on the screen and HMD when recording a sequence. Once recording process is stopped, a *scene.txt* file will be created with all scene information.

With the above instructions you will be able to record your own dataset!

[1] Once HMD position is configured, restarting the scene using VR controllers wouldn't change this configuration.

CHAPTER 9

Playback

With this final step you will generate the ground truth (e.g. semantic segmentation and depth maps, normal maps, stereo pairs, and also instance segmentation, etc.) of your recorded sequence. First of all you need to convert your recordings stored as *.txt* files in the *RecordedSequences* folder to *.json* files compatible with the playback process.

9.1 Convert recorded sequences to JSON

Go to *ROXTracker* located in the *World Outliner* panel in the UE4 editor and search for *JSON Management* (*see Figure 1*).



Fig. 1: Figure 1. Convert recorded sequences to JSON.

Set the *Input Scene TXT File Name* and also the *Output Scene Json File Name* and click on *Generate Sequence Json*. Json files will be stored in the *RecordedSequences* folder.

9.2 Configure playback process

Go to *ROXTracker* located in the *World Outliner* panel in the UE4 editor and search for *Playback* (*see Figure 2*).

Proceed with the following steps:

- **Specify JSON files**: add JSON file names to the *Json File Names* array.

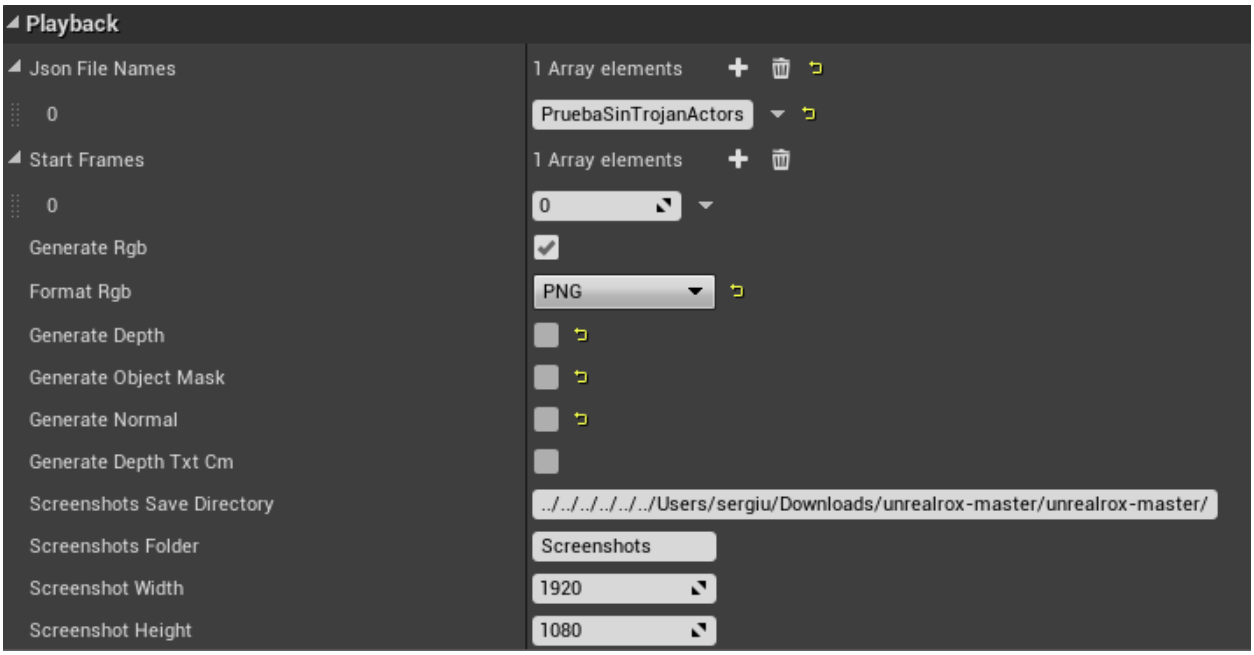


Fig. 2: Figure 2. Configurable parameters for the playback process.

- **Start from a given frame**: if playback process was accidentally interrupted you can resume the process indicating the latest generated frame (Default: 0).
- **Select the desired data to generate**: check the desired options you want to generate. You can also choose RGB data format.
- **Path**: choose where to save the data. *Screenshots Save Directory* and *Screenshots Folder* parameters.
- **Data resolution**: choose generated data resolution (Default: 1920x1080).

9.3 Run playback process

In order to proceed with the playback process, you will need to uncheck *Record mode* from the general *ROXTracker* configuration (*see Figure 1 from Recording section*). Run project in the *Selected Viewport* mode[1]. All the data will be saved by default on GeneratedSequences folder located in the root of UnrealROX project.

[1] If your main purpose is to generate data and you run the project in *VR Preview* mode, UnrealROX wouldn't work properly.

CHAPTER 10

Troubleshooting

Issues to be fixed. Any other problems and solutions?

In the following table we will list all the issues:

Issues	Description
1	Pawn and scene seems to be huge when playing the project in VR mode.

For each issue ID we will provide a solution:

Solution	Description
1	Check “VR” section in “Settings” -> “World Settings”. “World to Meters” must be 100.0.

CHAPTER 11

Changelog

Indicate the most important changes and new features!

CHAPTER 12

Contact

UnrealROX is an open source project hosted at UnrealROX github .

- Please use the issue tracker if you found any bug or want a new feature.
- Everyone is welcomed to contribute and make this project grow! Please contact any of the following team members.

12.1 Team Members

- Alberto Garcia-Garcia
- Pablo Martinez-Gonzalez
- Sergiu Oprea
- John Castro-Vargas
- Sergio Orts-Escolano
- Jose Garcia-Rodriguez
- Alvaro Jover-Alvarez

CHAPTER 13

Indices and tables

- genindex
- modindex
- search