roboyVR

Release 0.0.1

Jun 19, 2017

Usage and Installation

What is it?

RoboyVR is a virtual reality experience in which the user can watch, but also interact with roboy (a humanoid robot) while he performs specific tasks, e.g. walking, waving, etc. A virtual environment opens up a whole new set of perspectives for the user to enjoy and spectate roboy from all kinds of POVs. In addition to the rendering of the virtual roboy a mobile HUD shows detailed information about various roboy components, for example displaying the powerconsuption of particular motors. As the user chooses to take a more active part, roboy's pose can be influenced and altered by physical contact, e.g. shooting a projectile at the virtual model.

How does it work?

Roboy and its behavior is simulated on a virtual machine via Gazebo/ROS. Important information regarding roboy's movement are then sent through a ROSbridge(e.g. messages) towards Unity. In Unity roboy is rendered and constantly updated concerning positions, rotations, etc. On top of that detailed data (time lapsed) about components is displayed via graph rendering on different UI panels. With the help of a VR-Headset you can watch roboy move around in a virtual space.

Current status of the project and goals

Currently the project can render roboy with his pose and generate random data about his motors to visualize them. Our next tasks are as follow:

- Use real motor data and visualize that.
- Implement an interface to track the newest models and automate the process of creating the model in Unity.
- Implement an interface to record a simulation with all the data and save/ load it on runtime.
- Make the project completely Plug&Play meaning that you can send all kinds of data with a given format.

CHAPTER 1

Relevant Background Information and Pre-Requisits

For the user:

One of roboyVR design goals is to be as user friendly and intuitively as possible. Therefore the explorer in the virtual reality does not need to be familiar with explicit requirements. Yet it does no harm to have a basic understanding of how the HTC Vive and its tracking mechanism work.

Putting on the head mounted display in a way that fits the user is important for a frust-free experience, you can adjust the distance from the lenses to your eyes as well as the distance between the lenses itself, these tweaks help immensely when it comes to maintaining a sharp field of view.

Apart from that the tracking system needs to be setup correctly, too. The two base station should be able to see each other clearly with no viewing obstructions in their sights. They should be put up diagonal spanning a virtual room of two by five meters. For additional information take a look at this guide HTC Vive setup.

For the developer:

RoboyVR uses Unity3D to create an immersive and exciting virtual environment. Extensive expierence with Unity is recommended. Unity natively relies on C#, so advanced knowledge in this field is highly advised. Otherwise see Unity3D.

The roboy simulation which runs on Gazebo/ROS is written in C++, for this section a basic overview is sufficient to be able to understand/construct messages which are then sent via a ROSbridge. For starting the simulation you should be familiar with Linux/Ubuntu. Further it is useful to have some understanding of python in order to transform the roboy models via the Blender-api(an early python script already exists for this purpose).

The following links can be seen as a guideline, of course you can do the research by yourself.

- Unity provides a lot of tutorials for the editor and the API with code samples and videos: https://unity3d.com/de/learn/tutorials
- The UnifyWiki has a lot of example scripts for all kind of extensions: http://wiki.unity3d.com/index.php/Main_Page
- StackOverflow is a forum where you can search for answers regarding your coding problems: http://stackoverflow.com/
- UnityAnswers, similar to StackOverflow but only for Unity specific questions. The community is not as active and most questions are really basic, so bear that in mind: http://answers.unity3d.com/

- As we use ROS and our own custom messages, it is important to understand how ROS works and how ROS messages are built: http://wiki.ros.org/

If you have any further questions about the project, feel free to contact us via email: roboyvr@gmail.com

CHAPTER 2

Contents:

Installation

Roboy and its behavior is simulated on the virtual machine via ROS. Important information regarding roboy's movement are then sent through a ROSbridge(e.g. messages) towards Unity. In Unity roboy is rendered and constantly updated concerning positions, rotations, etc. With the help of a VR-Headset you can watch roboy move around in a virtual space.

This tutorial will help you setup roboyVR with all necessities it comes with.

Part 1: Setup Virtualbox with Ubuntu

1. Download and install Virtualbox for your OS https://www.virtualbox.org/
2. Download Ubuntu 16.04 (64bit) https://www.ubuntu.com/download/desktop
3. Mount the .iso and setup Virtualbox with the following settings (if available):

1. 4 cores (Settings->System->Processor)
2. 6 GB of RAM (Settings->System->Motherboard)
3. 128 MB of VRAM (Settings->Display->Screen)
4. 30 GB HDD space (Settings->Storage)

4. Set network settings to Bridged-Adapter or Host-Only Adapter

Part 2: Simulation Setup

1. Open Terminal and install the following packages

```
sudo add-apt-repository -y ppa:letrend/libcmaes
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /
↪etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 0xB01FA116
sudo apt-get update
sudo apt install libcmaes
sudo apt-get install ros-kinetic-desktop-full
sudo apt install ros-kinetic-controller-interface ros-kinetic-controller-manager ros-
↪kinetic-gazebo-ros-control ros-kinetic-ros-controllers
sudo apt install ros-kinetic-ecl-geometry
sudo apt install libncurses-dev
sudo apt-get install catkin
sudo apt-get install git
```

2. Clone the git repository into a ros working space

```
mkdir -p ~/ros_ws/src
cd ~/ros_ws/src
git clone https://github.com/Roboy/roboy-ros-control --recursive
```

3. Get additional dependencies

```
cd roboy-ros-control
git submodule update --init --recursive
cd src/flexrayusbinterface
sudo dpkg -i lib/libftd2xx_1.1.12_amd64.deb
```

4. Source the setup.bash

```
source /opt/ros/kinetic/setup.bash
cd ~/ros_ws
catkin_make
```

5. OPTIONAL: add this to your bash script (otherwise you have to type this commands in every new terminal window)

```
echo 'source /opt/ros/kinetic/setup.bash' >> ~/.bashrc
echo 'source ~/ros_ws/devel/setup.bash' >> ~/.bashrc
```

6. Create symlinks for gazebo to your roboy models

```
cd ~
mkdir -p ~/.gazebo/models
ln -s ~/ros_ws/src/roboy-ros-control/src/roboy_models/legs_with_upper_body ~/.gazebo/
↪models/
```

7. Install rosbridge

```
sudo apt install ros-kinetic-rosbridge-suite
```

Part 3: Unity Setup

1. Download Unity

- (latest working version with roboyVR is 5.6.0: https://unity3d.com/de/get-unity/download/archive)

2. Install Unity

- During the install process make sure to check also the standalone build option.
- Visual studio is recommended to use with Unity3D, as it is free and more user friendly than MonoDevelop (standard option).

3. Download this project

- Clone this github repository (master branch) to your system: https://github.com/sheveg/roboyVR.git
- Command: git clone -b master https://github.com/sheveg/roboyVR.git

Part 4: Blender & Python

- Install the latest version of Blender
- Install the latest version of Python

Getting started

Part 1: Run rosbridge and roboySimulation

```
roslaunch rosbridge_server rosbridge_websocket.launch
rosrun roboy_simulation VRRoboy
```

Part 2: Open the project in Unity

Unity is organized in Scenes. In order to watch the simulation in Unity which is running on the VM (in gazebo), open the ViveScene.

Part 3: Setup the scene

In the Scene you can observe the simulation from the VM within Unity. To do that you need to communicate the IP adress of your VM towards RoboyManager. The IP information is quickly found in Ubuntu by clicking on the two arrows pointing in opposite directions, right next to the system time. Afterwards a drop down menu will open, click on connection information. Remember the IP and paste it in the respective field in Unity.

You also need to drag the roboy prefab onto the RoboyManager if it is not already done. Each roboy model is tagged as a *RoboyPart*. If you import new models for roboy you need to change the tag accordingly and change the roboy prefab.

You can reset the simulation with the **R** key, you can also change the key in *RoboyManager*. To get a better view of the simulation we recommend to set the simulation to slow motion in rviz in the VM:

- If you want to start rviz, open a terminal (in the VM) and simply type **rviz**
- Set Fixed Frame to World (Displays->Fixed Frame)
- Add a marker (Add(Button)->marker)
- Add walking plugin (Panels->Add New Panel->WalkingPlugin)
- Turn slow motion on (within the walking plugin, it is a toggle button)

Extra: Update roboy models

IMPORTANT: The next part will be soon outdated as we plan update the models and automate the process, so be aware!

In /roboyVR/Assets/RoboyModel/OriginModels there is a script **meshDownloadScript.py**. When executed, it downloads roboy models from this location:

- https://github.com/Roboy/roboy_models/tree/master/legs_with_upper_body/cad.

After the download process is complete the models will be converted by blender so that they work fine with unity (.fbx format). Obviously you need blender and python installed on your system so that the script can do it's work. You can use the template **runScript** bat file for Windows.

The format:

```
start "" "pathToBlender/blender.exe" -P "pathToScript\meshDownloadScript.py" **meshes name seperated by ',' without whitespace and file format**
```

Example:

```
start "" "C:\Programs\Blender\blender.exe" -P "C:\Documents\roboyVR\Assets\RoboyModel\OriginModels\meshDownloadScript.py" hip,torso,thigh_left,head
```

Context

The core of RoboyVR renders and updates roboy's pose as its receiving data from the simulation via ROS-messages. Additional information inside messages like current powerconsumption or motorforce is displayed on an interactive GUI. Apart from that the user can actively manipulate the simulation through various tools. On top of that the system can check for the latest roboymodel with the help of github and update it if necessary.

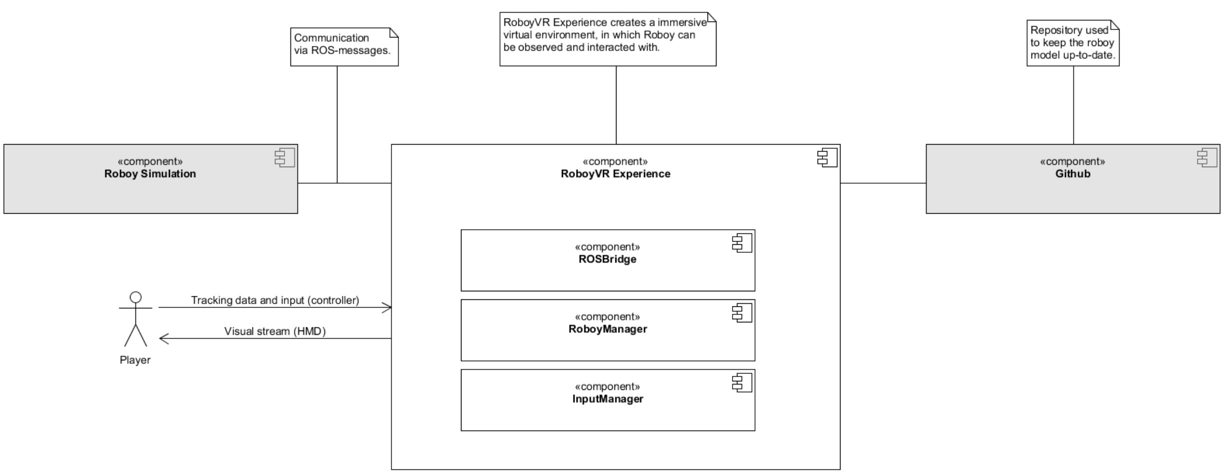

Fig. 2.1: RoboyVR Experience has two neighboring systems. Roboy simulation to receive pose data and Github for model updates.

Conventions

We follow the coding guidelines:

Table 2.1: Coding Guidelines

Language	Guideline	Tools
Python	https://www.python.org/dev/peps/pep-0008/	
C++	http://wiki.ros.org/CppStyleGuide	clang-format: https://github.com/davetcoleman/roscpp_code_format

The project follows custom guidelines:

1. All scripts are structured like this:
    1. The script is ordered in regions:
    - PUBLIC_MEMBER_VARIABLES
    - PRIVATE_MEMBER_VARIABLES
    - UNTIY_MONOBEHAVIOUR_METHODS
    - PUBLIC_METHODS
    - PRIVATE_METHODS
    2. In PUBLIC_MEMBER_VARIABLES you have define at first your properties and then public variables.
    3. In PRIVATE_MEMBER_VARIABLES you have define at first your serialized private variables and then the normal ones.
    4. In UNTIY_MONOBEHAVIOUR_METHODS the order is as follows: Awake, Start, OnEnable, OnDisable, Update
2. All variables and functions where it is not instantly clear what it does, have to be commented with a summary.
3. Make variables only public if they need to be. Mark variables as Serializable when you need to edit them in the editor.
4. The capitalization follows a specific set of rules:
- public variables and properties start with an uppercase
- private variables and properties start with a lowercase
- public functions start with an uppercase
- private functions start with an lowercase
5. Coroutines which are accessed in other classes must have a public interface.
6. When you store components in a variable, which are directly on the object itself, put a [RequireComponent(typeof(ComponentType))] on top of the class.

We include a template class with all rules implemented.

class `TemplateClass`

Describe your class shortly here.

Inherits from Monobehaviour

Public Functions

void TemplateClass.SomePublicMethod()
Describe the function shortly here.

void TemplateClass.ActivateBear()
Describe the function shortly here.

Public Members

string TemplateClass.SomePublicVariable
Describe your public variable shortly here.

Property

property TemplateClass::SomeProperty
Describe your property shortly here.

Private Functions

void TemplateClass.Awake()
Describe the function shortly here.

void TemplateClass.Start()
Describe the function shortly here.

void TemplateClass.OnEnable()
Describe the function shortly here.

void TemplateClass.OnDisable()
Describe the function shortly here.

void TemplateClass.Update()
Describe the function shortly here.

void TemplateClass.somePrivateMethod()
Describe the function shortly here.

IEnumerator TemplateClass.someBearCoroutine()
Describe the coroutine shortly here.

Private Members

float TemplateClass.m_SomeSerializedVariable
Describe your serialized variable shortly here.

Rigidbody TemplateClass.m_Rigidbody
Rigidbody component on the object

int TemplateClass.m_SomePrivateVariable
Describe your private variable shortly here.

Architecture Constraints

Table 2.2: Hardware Constraints

Constraint Name	Description
HTC Vive	We need user position tracking and movement tracking.

Table 2.3: Software Constraints

Constraint Name	Description
Unity3D	Unity provides an interface for the HTC Vive with the steamVR plugin. On top of that it renders the simulation.
Gazebo&ROS	The simulation uses both systems.
OracleVM	We use the VM for running Ubuntu on the same machine. You can also just use Ubuntu on a separate machine.
Blender	We used blender to convert the roboy models so that Unity can import them.

Table 2.4: Additional Plugins

Constraint Name	Description
ROSBridge	It connects the simulation on Ubuntu with Unity on Windows.
Vuforia	This interface connects the HTC Vive with Unity.
steamVR	We use this interface to use the API of the HTC Vive.

Table 2.5: Operating System Constraints

Constraint Name	Description
Windows 10	We did not test it yet on other Windows versions. It may also work on older machines.
Ubuntu 16.04	The simulation runs on Ubuntu.

Table 2.6: Programming Constraints

Constraint Name	Description
C++	The simulation is written in C++.
C#	Unity uses C# as the standard programming language.
Python	We use Python with the Blender API to automate the process of converting the roboy models.

User Interfaces

In the following figures you can see multiple tools to interact with roboy. The user can select different parts of roboy and inspect these parts further with detailed information about the motors. On top of that the user can actively interact with roboy with the Shooting Tool. It triggers an external force in the simulation and displays the result in real time in the VR environment. In the future it will be possible to control time, so to rewind the simulation and save/ load them on runtime.

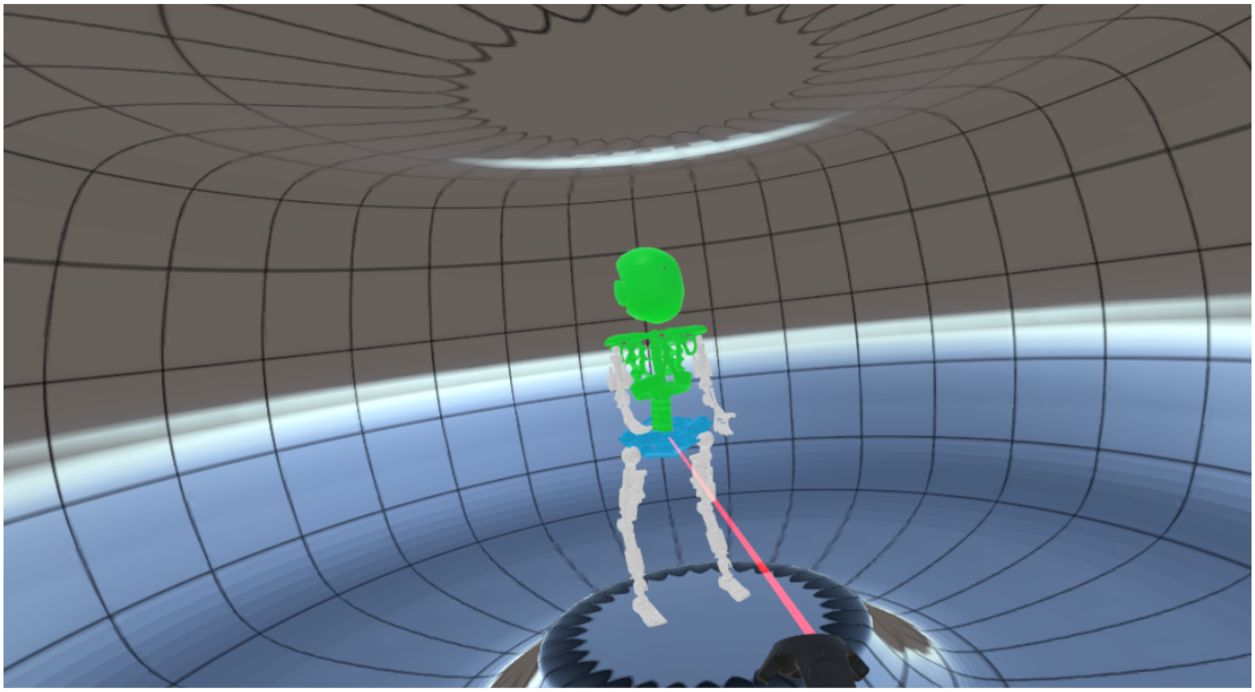

Fig. 2.2: Tool for selecting roboy parts.

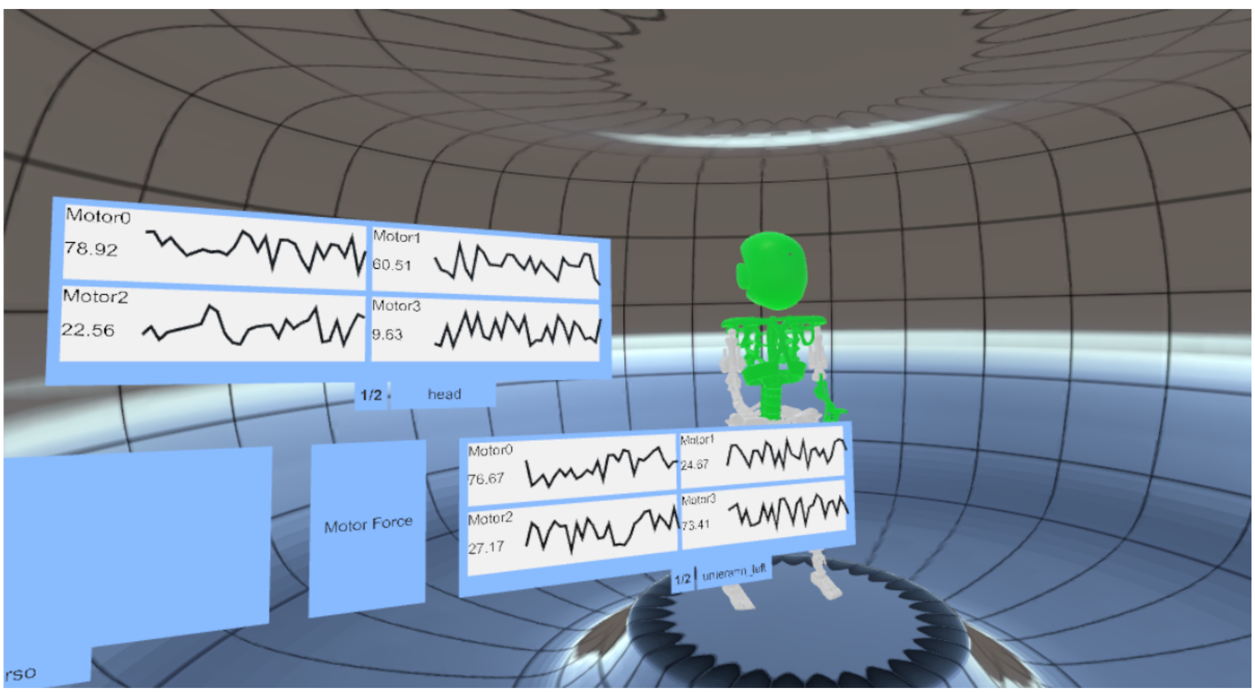

Fig. 2.3: UI Panels displaying motor force of several roboy parts.

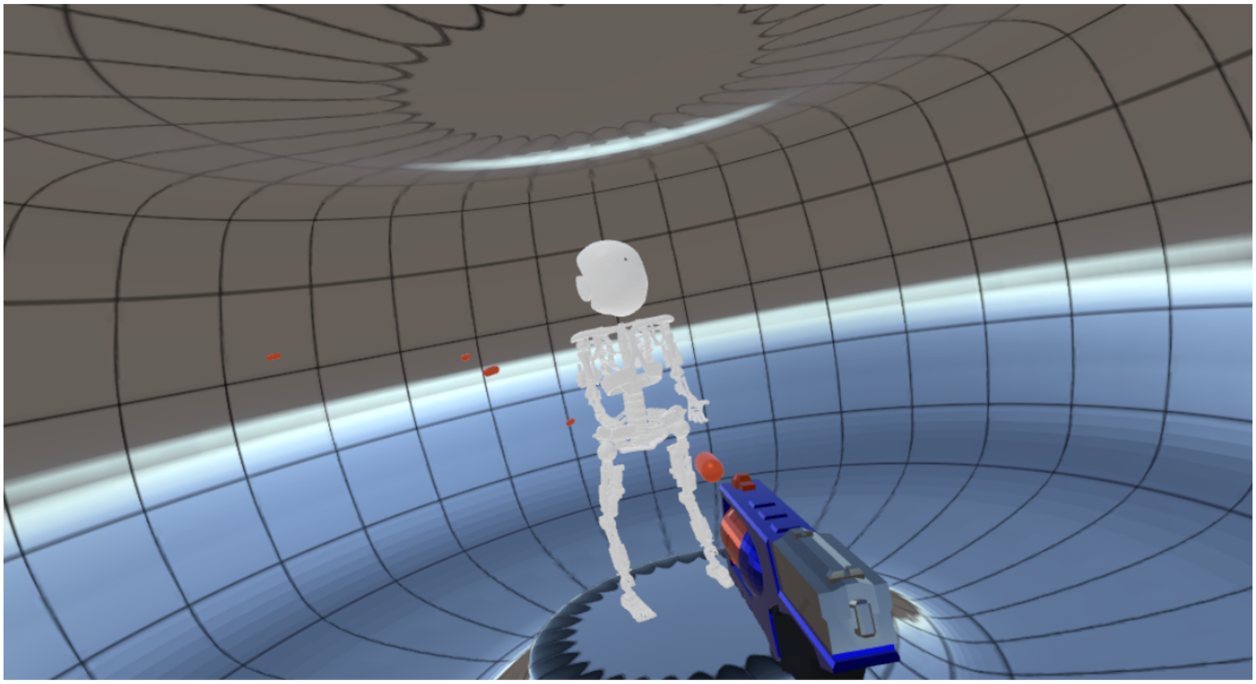

Fig. 2.4: Tool to shoot roboy and trigger an external force.

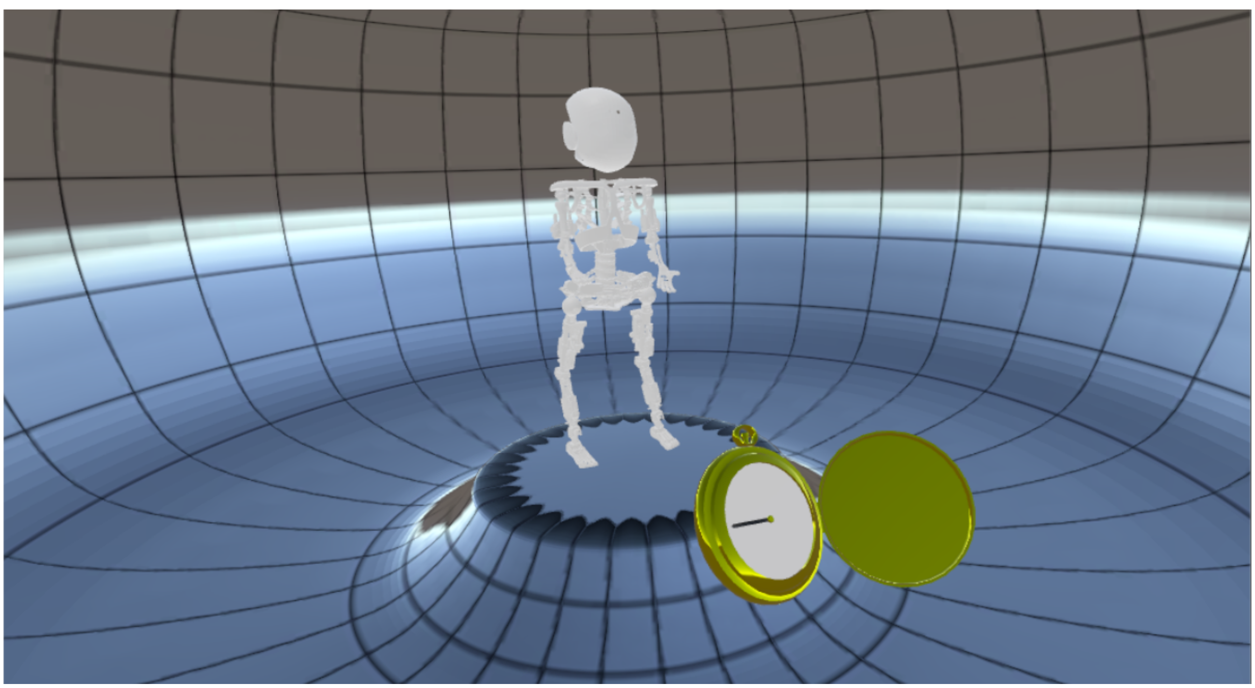

Fig. 2.5: Tool to alter flow of time.

Public Interfaces

Managers

RoboyManager

class RoboyManager

Roboymanager has different tasks:

- Run ROS:

1.Connect to the simulation.

2.Add subscriber to the pose.

3.Add publisher for external force.

4.Add service response for world reset.

- Receive and send ROS messages:

1.receive pose msg to adjust roboy pose.

2.subscribe to external force event and send msg to simulation.

3.send service call for world reset.

4.FUTURE: receive motor msg and forward it to the according motors.

Inherits from Singleton< RoboyManager >

Public Functions

void RoboyManager.InitializeRoboyParts()

Initializes the roboy parts with a random count of motors => WILL BE CHANGED IN THE FUTURE, for now just a template

void RoboyManager.ReceiveMessage(RoboyPoseMsg msg)

Main function to receive messages from ROSBridge.

Adjusts the roboy pose and the motors values (future).

Parameters

- `msg`: JSON msg containing roboy pose.

void RoboyManager.ReceiveExternalForce(RoboyPart roboyPart, Vector3 position, Vector3

Sends a message to the simualation to apply an external force on a certain position.

Parameters

- `roboyPart`: The roboypart where the force should be applied.
- `position`: The relative position of the force to the roboypart.
- `force`: The direction and the amount of force relative to roboypart.
- `duration`: The duration for which the force should be applied.

Public Members

string RoboyManager.VM_IP = “”
The IP address of the VM or the machine where the simulation is running

Transform RoboyManager.Roboy
Transform of roboy with all roboy parts as child objects

Property

property RoboyManager::RoboyParts
Public variable for the dictionary with all roboyparts, used to adjust pose and motor values

Private Functions

void RoboyManager.Awake()
Initialize ROSBridge and roboy parts

void RoboyManager.Update()
Run ROSBridge

void RoboyManager.OnApplicationQuit()
Disconnect from the simulation when Unity is not running.

Quaternion RoboyManager.gazeboRotationToUnity(Quaternion gazeboRot)
Converts a quaternion in gazebo coordinate frame to unity coordinate frame.

Return Quaternion in unity coordinate frame.

Parameters

- `gazeboRot`: Quaternion in gazebo coordinate frame.

Vector3 RoboyManager.gazeboPositionToUnity(Vector3 gazeboPos)
Converts a vector in gazebo coordinate frame to unity coordinate frame.

Return Vector in unity coordinate frame.

Parameters

- `gazeboPos`: Vector in gazebo coordinate frame.

Vector3 RoboyManager.unityPositionToGazebo(Vector3 unityPos)
Converts a vector in unity coordinate frame to gazebo coordinate frame.

Return Vector in gazebo coordinate frame.

Parameters

- `unityPos`: Vector in unity coordinate frame.

Quaternion RoboyManager.unityRotationToGazebo(Quaternion unityRot)
Converts a quaternion in unity coordinate frame to gazebo coordinate frame.

Return Quaternion in gazebo coordinate frame.

Parameters

- `unityRot`: Quaternion in unity coordinate frame.

void RoboyManager.drawTendons()
Test function to draw tendons.

For now draws only random lines. TEMPLATE!

void RoboyManager.adjustPose(RoboyPoseMsg msg)
Adjusts roboy pose for all parts with the values from the simulation.

Parameters

- `msg`: JSON msg containing the roboy pose.

Private Members

ROSBridgeWebSocketConnection RoboyManager.m_Ros = null
ROSBridge websocket

RoboyPoseMsg RoboyManager.m_RoboyPoseMessage
Pose message of roboy in our build in class

Dictionary<string, RoboyPart> RoboyManager.m_RoboyParts = new Dictionary<string, RoboyPart>()
Dictionary with all roboyparts, used to adjust pose and motor values

InputManager

class InputManager
InputManager holds a reference of every tool.

On top of that it listens to button events from these tools and forwards touchpad input to the respective classes.

Inherits from Singleton< InputManager >

Public Types

enum TouchpadStatus
Possible touchpad positions.

Values:

Right

Left

Top

Bottom

None

Public Functions

void InputManager.GUIControllerSideButtons(object sender, ClickedEventArgs e)
Changes view mode when the user presses the side button on the controller.

Parameters

- `sender`

- e

void InputManager.ToolControllerSideButtons(object sender, ClickedEventArgs e)
Changes the tool when the user presses the side button on the controller.

Parameters

- sender
- e

void InputManager.GetTouchpadInput(object sender, ClickedEventArgs e)
Retrives the touchpad input of the tool controller and updates the values.

Parameters

- sender
- e

Property

property InputManager::GUI_Controller
Public *GUIController* reference.

property InputManager::Selector_Tool
Public *SelectorTool* reference.

property InputManager::ShootingTool
Public *ShootingTool* reference.

property InputManager::TimeTool
Public TimeTool reference.

property InputManager::SelectorTool_TouchpadStatus
Touchpad status of the controller where selector tool is attached to.

property InputManager::GUIController_TouchpadStatus
Touchpad status of the controller where gui controller tool is attached to.

Private Functions

void InputManager.Start()
Initialize all tools.

void InputManager.Update()
Calls the ray cast from the selector tool if it is active.

IEnumerator InputManager.InitControllers()
Initializes all controllers and tools.

Return

Private Members

SelectorTool InputManager.m_SelectorTool
Private *SelectorTool* reference.

Is serialized so it can be dragged in the editor.

ShootingTool InputManager.m_ShootingTool
Private *ShootingTool* reference.

Is serialized so it can be dragged in the editor.

TimeTool InputManager.m_TimeTool
Private TimeTool reference.

Is serialized so it can be dragged in the editor.

GUIController InputManager.m_GUIController
Private *GUIController* reference.

Is serialized so it can be dragged in the editor.

ModeManager

class ModeManager
ModeManager holds a reference of every active mode and provides function to switch between them.

This includes:

- Current tool: *ShootingTool*, SelectionTool etc.
- Current view mode: singe vs. comparison
- Current GUI mode: selection vs. GUI panels
- Current panel mode: motorforce, motorvoltage etc.

Inherits from Singleton< ModeManager >

Public Types

enum Viewmode
We change between Single view where we can choose only one objet at a time and comparison view with three maximum objects at a time.

Values:

Single

Comparison

enum Panelmode
Describes the different modes for panel visualization.

Values:

Motor_Force

Motor_Voltage

Motor_Current

Energy_Consumption

Tendon_Forces

enum `GUIMode`
Enum for current GUI mode.

Values:

`Selection`

`GUIPanels`

enum `ToolMode`
SelectorTool: Select roboy meshes.

ShooterTool: Shoot projectiles at roboy. TimeTool: Reverse/stop time.

Values:

`SelectorTool`

`ShooterTool`

`TimeTool`

Public Functions

`void ModeManager.ChangeViewMode()`
Changes between single and comparison view.

`void ModeManager.ChangeGUIMode()`
Switches between selection and panels GUI mode.

`void ModeManager.ChangeToolMode()`
Switches between all tools.

`void ModeManager.ChangePanelModeNext()`
Changes the panel mode to the next one based on the order in the enum defintion.

`void ModeManager.ChangePanelModePrevious()`
Changes the panel mode to the previous one based on the order in the enum defintion.

`void ModeManager.ResetPanelMode()`
Resets current panel mode to MotorForce.

Property

property `ModeManager::CurrentViewmode`
Current view mode, READ ONLY.

property `ModeManager::CurrentPanelmode`
Current panel mode, READ ONLY.

property `ModeManager::CurrentGUIMode`
Current GUI mode, READ ONLY.

property `ModeManager::CurrentToolMode`
Current Tool mode, READ ONLY.

Private Members

Viewmode ModeManager.m_CurrentViewmode = Viewmode.Comparison
Private variable for current view mode.

Panelmode ModeManager.m_CurrentPanelmode = Panelmode.Motor_Force
Private variable for current panel mode.

GUIMode ModeManager.m_CurrentGUIMode = GUIMode.Selection
Private variable for current GUI mode.

ToolMode ModeManager.m_CurrentToolMode = ToolMode.SelectorTool
Private variable for current Tool mode.

SelectorManager

class SelectorManager
SelectorManager is responsible to hold references of all selected roboy parts and the corresponding UI elements.

Inherits from Singleton< SelectorManager >

Public Functions

void SelectorManager.AddSelectedObject(SelectableObject obj)
Adds the roboy part to selected objects.

Parameters

- `obj`: *SelectableObject* component of the roboy part.

void SelectorManager.RemoveSelectedObject(SelectableObject obj)
Removes the roboy part from the selected objects.

Parameters

- `obj`: *SelectableObject* component of the roboy part.

void SelectorManager.ResetSelectedObjects()
Resets all roboy parts to default state and empties the selected objects list.

Property

property SelectorManager::UI_Elements
Property which returns a dictionary of all UI elements in the *SelectionPanel*.

property SelectorManager::SelectedParts
Reference of all currently selected roboy parts.

property SelectorManager::MaximumSelectableObjects
Integer to switch between single mode selection and normal mode collection.

Private Functions

IEnumerator SelectorManager.Start()
Initializes all variables.

Return

Private Members

Transform SelectorManager.m_Roboy
Transform of roboy model.

List<SelectableObject> SelectorManager.m_RoboyParts = new List<*SelectableObject*>()
List of *SelectableObject* components of all roboy parts.

List<SelectableObject> SelectorManager.m_SelectedParts = new List<*SelectableObject*>()
List of *SelectableObject* components of all selected parts.

int SelectorManager.m_MaximumSelectableObjects = 3
Maximum cound of selectable objects in multiple selection mode.

int SelectorManager.m_CurrentMaximumSelectedObjects = 3
Current count of maximum selectable objects.

Dictionary<string, GameObject> SelectorManager.m_UI_Elements = new Dictionary<string, GameObject
Private reference to all UI elements.

Material SelectorManager.m_UI_Line_Material
I am not sure what this is.

Will be deleted soon.

Tools

ControllerTool

class **ControllerTool**
ControllerTool is a base class for all tools which are attached to a controller.

It provides access to steamVR functions to track the input of the controllers. On top of that it provides a function to vibrate the controller for a defined time.

Inherits from MonoBehaviour

Subclassed by *SelectorTool*, *ShootingTool*

Public Functions

void ControllerTool.Vibrate()
Starts a coroutine to vibrate the controller for a fixed time.

void ControllerTool.Initialize()
Initiliazes the controller in a coroutine.

Intermediate function for outside classes.

Property

property `ControllerTool::Controller`
Returns the controller identity for verification purposes for outside classes.

property `ControllerTool::ControllerEventListener`
Returns a component which listens to controller events like OnTouchpad.

Private Functions

`void ControllerTool.Awake()`
Calls initialize for all controller members.

`IEnumerator ControllerTool.vibrateController()`
Coroutine to vibrate the controller for a fixed time.

Return

`IEnumerator ControllerTool.initializeCoroutine()`
Coroutine to initialize all controller members.

Return

SelectorTool

class `SelectorTool`
SelectorTool provides a functionality to select parts of roboy on the mesh itself or through the GUI.

Inherits from *ControllerTool*

Public Functions

`void SelectorTool.GetRayFromController()`
Starts a ray from the controller.

If the ray hits a roboy part, it changes its selection status. Otherwise it resets the last selected/targeted roboy part.

Private Functions

`void SelectorTool.Start()`
Initializes the lineRenderer component.

Private Members

`LineRenderer SelectorTool.m_LineRenderer`
LineRenderer to draw the laser for selection.

`SelectableObject SelectorTool.m_LastSelectedObject`
Variable to track the last selected object for comparison.

`float SelectorTool.m_RayDistance` = 3f
Maximum ray length for selection.

ShootingTool

class **ShootingTool**

ShootingTool is used to shoot a projectile on roboy.

The projectile then triggers a ROS message to send an external force to the simulation.

Inherits from *ControllerTool*

Public Members

Projectile ShootingTool.ProjectilePrefab
Projectile prefab which is responsible to send the ROS message.

Transform ShootingTool.SpawnPoint
Spawn transform to retrieve the spawn position and direction.

Transform ShootingTool.Trigger
Trigger transform for trigger animation.

Transform ShootingTool.TriggerBack
Transform of the position when trigger is fully pressed.

float ShootingTool.ShootDelay = 0.5f
Reload time between shots.

Private Functions

void ShootingTool.Start()
Initializes trigger position.

void ShootingTool.Update()
Shoots when the user presses the trigger to maximum value if shooting is not on cooldown.

void ShootingTool.Shoot()
Instantiates a projectile prefab on the SpawnPoint.

void ShootingTool.animateTrigger()
Animates trigger based on current trigger value.

Private Members

Vector3 ShootingTool.m_InitTriggerPosition
The standard trigger position.

float ShootingTool.m_CurrentShootCooldown = 0f
Variable for tracking current shooting cooldown.

GUIController

class **GUIController**

GUIController is attached on another controller as the Tools like *ShootingTool* or *SelectorTool*.

It is mainly responsible for animating so the following tasks refer always to animation:

- manage the switch between selection mode and panel mode

- manage switch between different panel modes
- manage page switch inside a panel mode **NOTICE: Right now *GUIController* is not inheriting from *ControllerTool* as we implemented this script at the beginning of the project. This will be changed soon, so be aware that this documentation could be out of date!**

Inherits from MonoBehaviour

Public Types

enum `UIPanelAlignment`
Enum for possible panel alignments.

Values:

`Left`

`Top`

`Right`

Public Functions

`void GUIController.CheckTouchPad(InputManager.TouchpadStatus touchpadStatus)`
Checks the touchpad input of the controller and acts accordingly:

1. Left: changes to previous panel if in panel mode
2. Right: changes to next panel if in panel mode
3. Top: changes between GUI modes
4. Bottom: changes the page of the current panel if in panel mode

Parameters

- `touchpadStatus`

`void GUIController.InitializePanels()`
Initialize the position of all panels and set their corresponding roboy part reference.

Public Members

`UIPanelRoboyPart GUIController.UIPanelRoboyPartPrefab`
Prefab variable for a roboy UI panel.

Property

property `GUIController::Controller`
Public variable for outside classes to track input.

property `GUIController::ControllerEventListener`
Public variable for outside classes to track controller events.

property `GUIController::UIFadePanels`
Property which holds a dictionary to store a reference to the standard position of panels in panel mode.

Private Functions

void GUIController.Start()
Initializes the controller variables.

Intializes the UI Panels and creates them for every roboy part for every panel mode.

void GUIController.changePageOfPanel()
Changes the page of the current panel if the current GUI mode is set to panel mode.

void GUIController.changepanelsToNextMode()
Changes to the next panel if the current GUI mode if set to panel mode.

void GUIController.changeToPreviousMode()
Changes to the previous panel if the current GUI mode if set to panel mode.

IEnumerator GUIController.changeGUIMode()
Changes GUI mode between selection and panel mode.

Return

void GUIController.positionPanels()
Positions the panels according to the template panel positions in the editor.

Private Members

SteamVR_Controller.Device GUIController.m_SteamVRDevice
Private variable to track controller input.

SteamVR_TrackedObject GUIController.m_SteamVRController
Private variable to track controller identity.

SteamVR_TrackedController GUIController.m_SteamVRTrackedController
Private variable to track controller events.

Dictionary<RoboyPart, UIPanelRoboyPart> GUIController.m_RoboyPartPanelsDic = new Dictionary
Dictionary to store a reference to all UI Panels which are created at the start of the scene.

Dictionary<UIPanelAlignment, FadePanelStruct> GUIController.m_UIFadePanels = new Dictionary
Dictionary to store a reference to the standard position of panels in panel mode.

SelectionPanel GUIController.m_SelectionPanel
Reference to the *SelectionPanel*.

struct FadePanelStruct
Struct to store the position where a panel should fade in and out.

Additional classes

SelectableObject

class SelectableObject
SelectableObject is attached on every roboy part.

Is used to switch between selection states, which then again changes the material and manages GUI highlighting.

Inherits from MonoBehaviour

Public Types

enum State
Enum for possible selection states.

Values:

DEFAULT

TARGETED

SELECTED

Public Functions

void SelectableObject.SetStateSelected()
Changes the state depending on the current state and updates the result in *SelectorManager*.

void SelectableObject.SetStateTargeted()
Sets the state to targeted if the last state was default.

void SelectableObject.SetStateDefault(bool *forceMode* = *false*)
Resets the state to default if the last state was targeted (without force mode).

Parameters

- `forceMode`: Boolean to force the state switch.

Public Members

Material SelectableObject.TargetedMaterial
Material of meshes which are targeted.

Material SelectableObject.SelectedMaterial
Material of meshes which are selected.

Property

property SelectableObject::CurrentState
Public property to track the selection state for outside classes.

Private Functions

void SelectableObject.Awake()
Initializes the renderer array and default material.

void SelectableObject.changeState(State *s*)
Switches the state based on the parameter and manages GUI highlighting.

Parameters

- `s`: State to which the object should switch to.

Private Members

State SelectableObject.m_CurrentState = State.DEFAULT
Variable to track the current selection state.

Renderer [] SelectableObject.m_Renderers
Array of all renderer to change the material.

Material SelectableObject.m_DefaultMaterial
Default material of all meshes.

SelectionPanel

class SelectionPanel
SelectionPanel is the panel where you can select roboy parts with the *SelectorTool* on a GUI interface.

Whereas the components inside the panel provide functions to switch between selection states, this class is responsible to animate the switch between Selection Mode and GUI Panel mode.

Inherits from MonoBehaviour

Public Functions

void SelectionPanel.Shrink()
Starts a coroutine to shrink the selection panel.

void SelectionPanel.Enlarge()
Starts a coroutine to enlarge the selection panel.

IEnumerator SelectionPanel.shrinkCoroutine()
Coroutine to shrink the selection panel.

Fades out the UI elements, turns off the colliders and shrinks the selection panel.

Return

Public Members

Text SelectionPanel.CurrentPanelMode
Reference to the text component to display the current panel mode like MotorForce etc.

Private Functions

void SelectionPanel.Awake()
Initializes all variables like the RectTransform and the lists.

IEnumerator SelectionPanel.enlargeCoroutine()
Coroutine to enlarge the selection panel.

Fades in the UI elements, turns on the colliders and enlarges the selection panel.

Return

Private Members

RectTransform SelectionPanel.m_RectTransform
Private RectTransform component for animation purposes.

List<CanvasGroup> SelectionPanel.m_ChildCanvasGroups = new List<CanvasGroup>()
List of all canvas groups to change the alpha value.

List<BoxCollider> SelectionPanel.m_ChildBoxColliders = new List<BoxCollider>()
List of all colliders on the UI elements to switch them off and on.

Projectile

class Projectile
Inherits from MonoBehaviour

Public Members

float Projectile.projectileSpeed
The speed of the projectile.

Private Functions

void Projectile.Update()
Move forward and destroy yourself if you are not in the roboy cave.

void Projectile.OnCollisionEnter(Collision collision)
Triggers a ROS external force message.

Transforms the hit point from world space to roboy local space.

Parameters

- collision

Solution Strategy

RoboyVR consists of different components which work together. One big part deals with the transition between the different coordinate frames of Gazebo and Unity. At first the rotations were represented via Euler Angles, this lead to gimbal locks. To avoid this we switched to quaternions. Roboy's pose needs to be converted to Unity's coordinate frame. In addition we convert the model of roboy to a unity friendly format. The other part deals with user interaction. RoboyVR uses user input to manipulate the simulation and renders the result on a GUI.

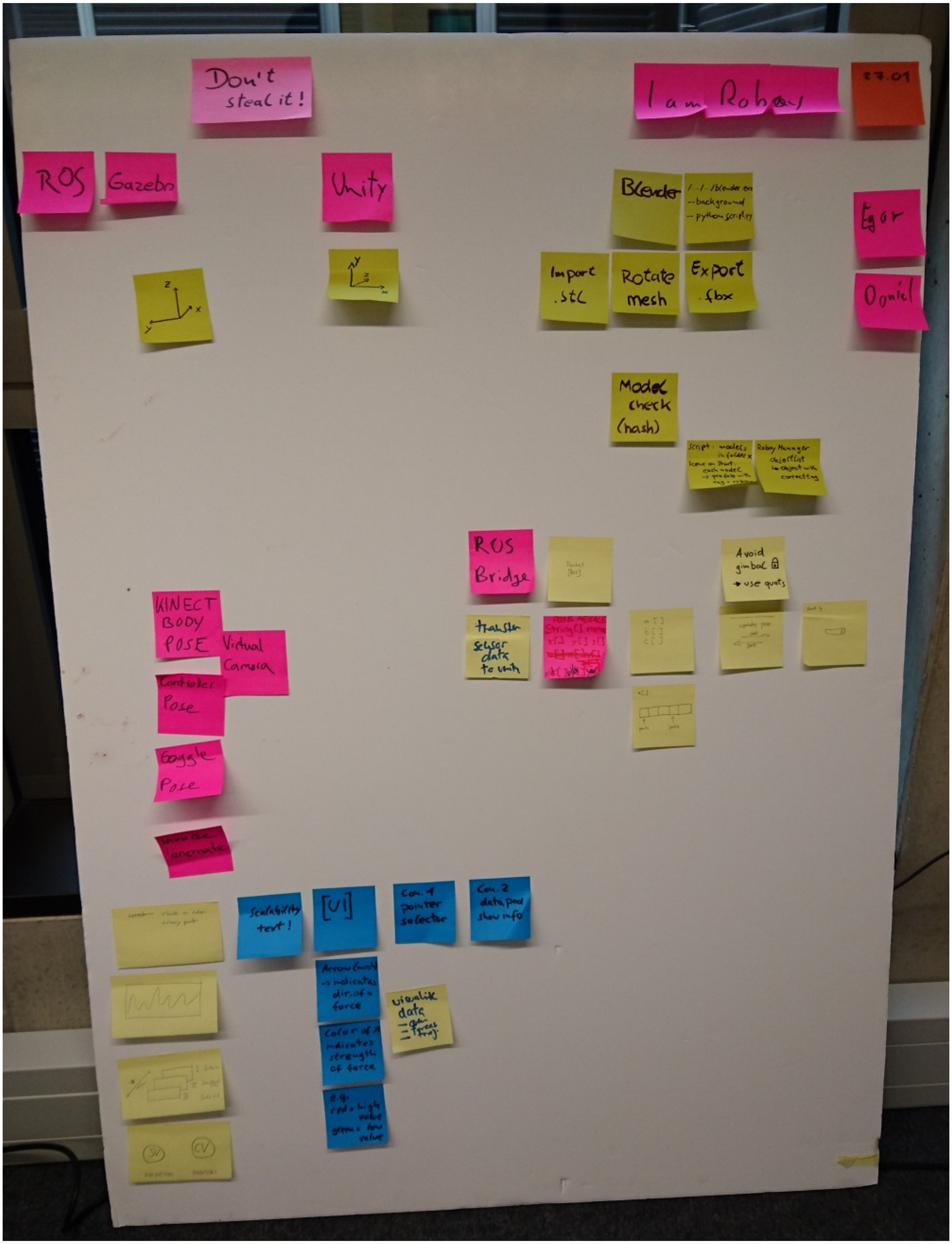

Fig. 2.6: Whiteboard showing problems and solutions that occured during development of roboyVR.

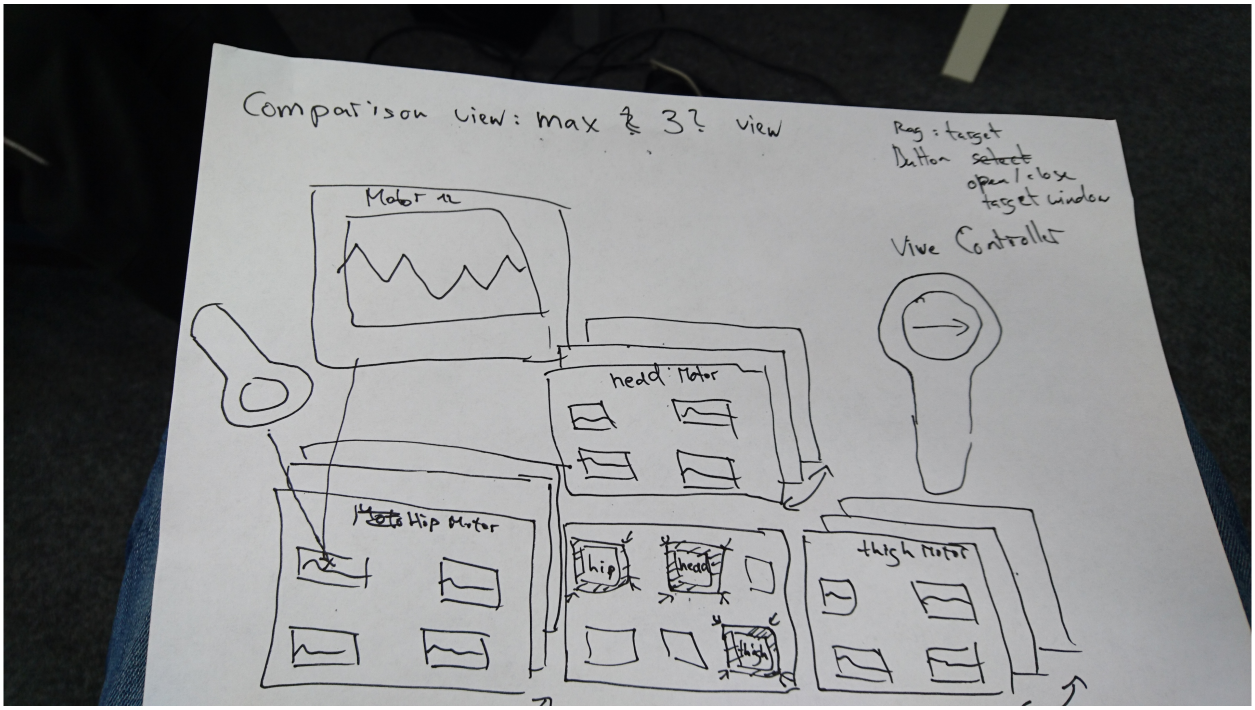

Fig. 2.7: Handdrawn sketch showcasing the design of a specific UI Panelmode (comparison).

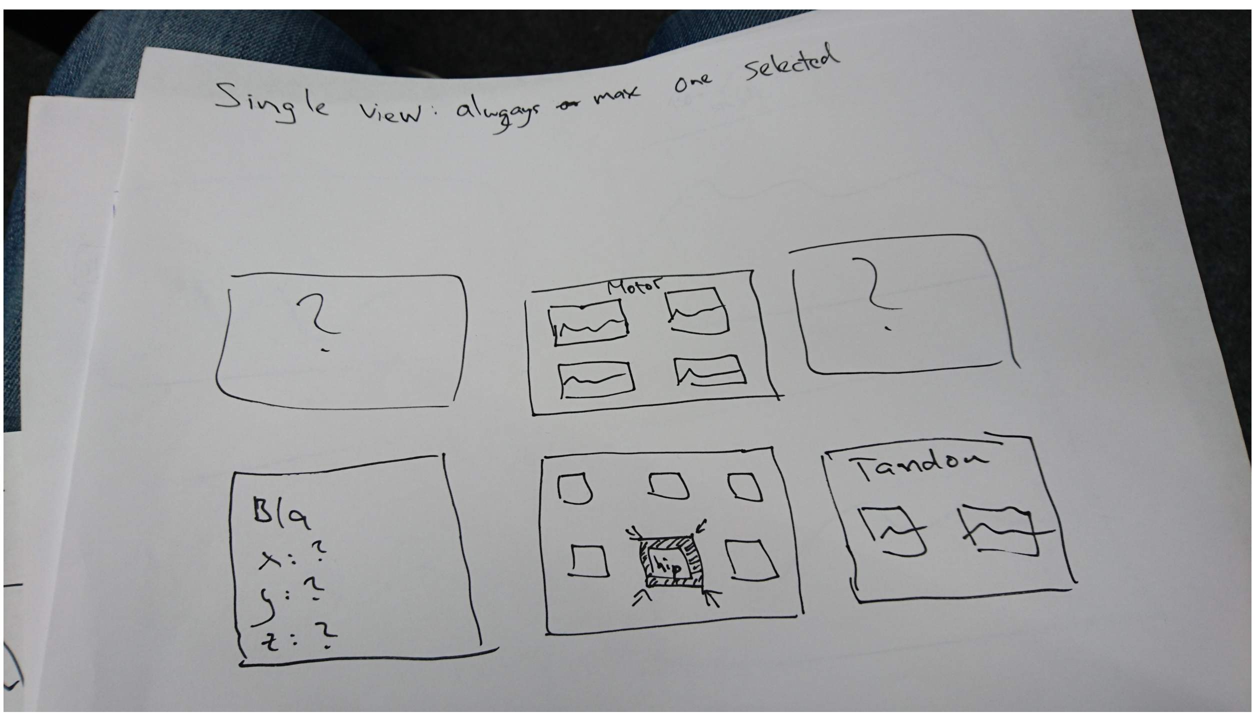

Fig. 2.8: Handdrawn sketch showcasing the design of a specific UI Panelmode (single).

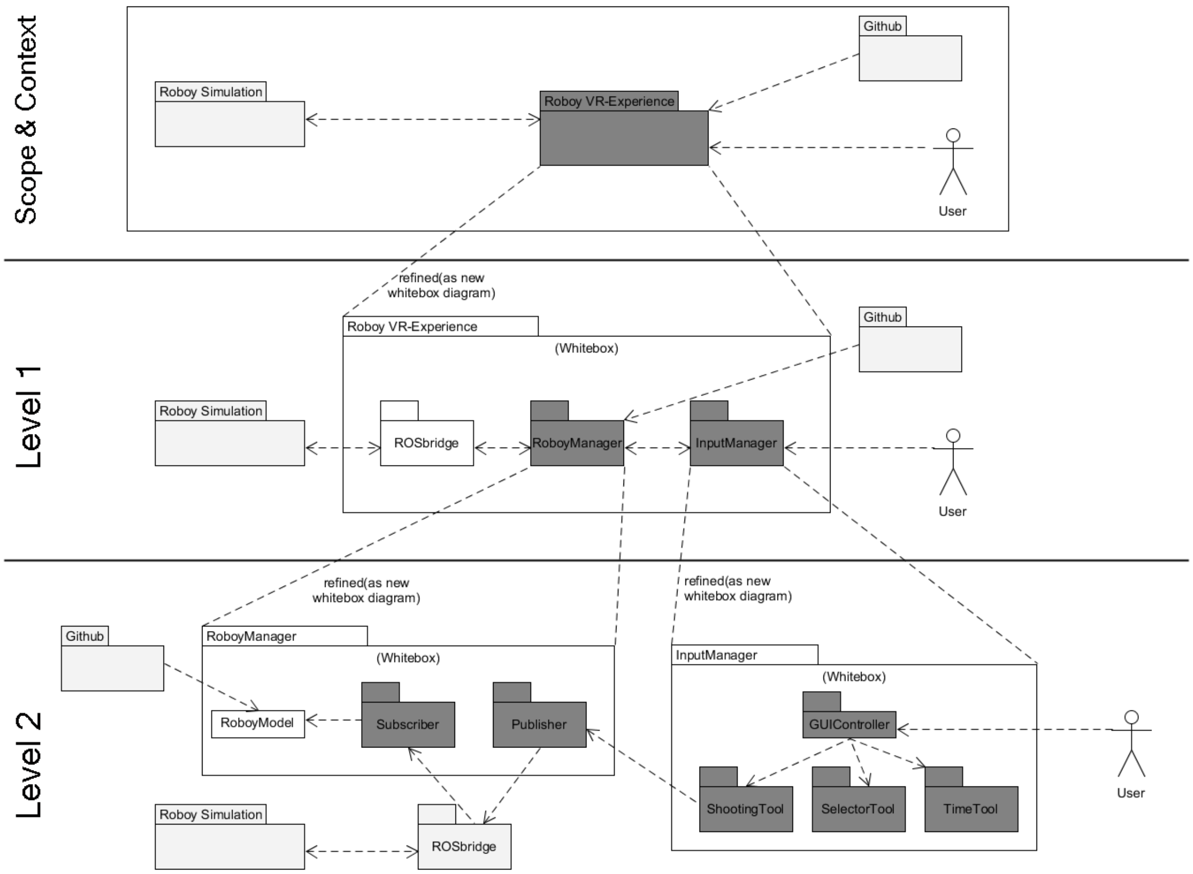

Fig. 2.9: RoboyVR Experience has several neighbouring systems like the simulation and github, it consists of various components like RoboyManager/Inputmanager and can be manipulated by the user through the HMD system.

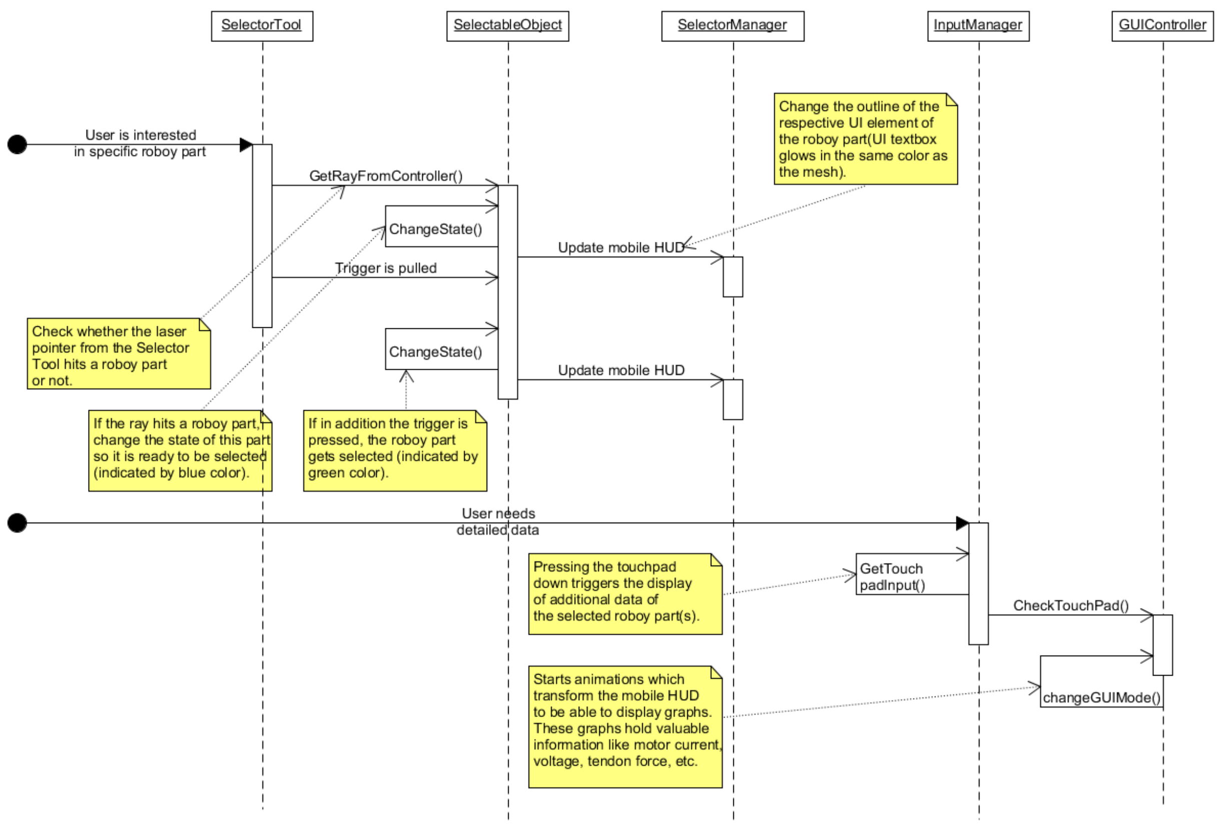

Fig. 2.10: User needs detailed information regarding specific roboy parts, e.g. power-consumption in motor24 upper_left_arm.

Building Block View

Runtime View

Runtime Display Information regarding Roboyparts

Runtime Physical impact on roboy (shooting)

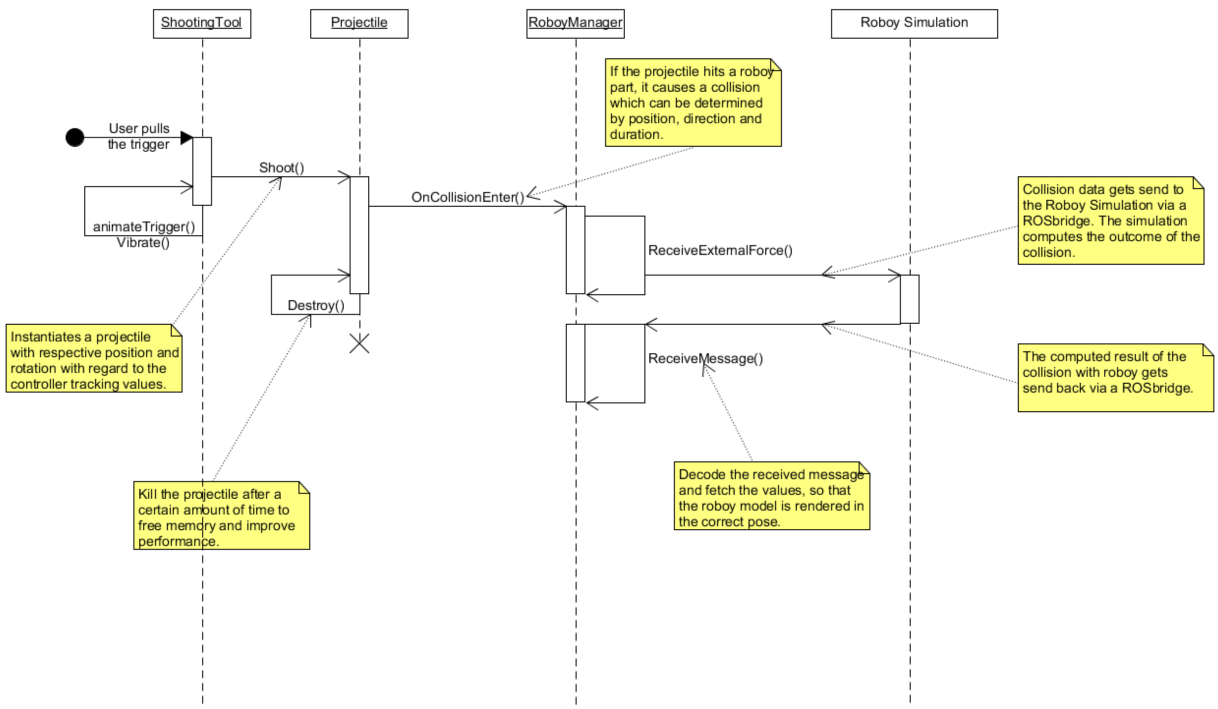

Fig. 2.11: User wants to physically harm the poor roboy and shoots a nerf dart towards him.

...

Deployment View

Presentations

Midterm WS16/17: https://drive.google.com/open?id=0BxLtAtPNIIYQOHFIRjdrajR0UVk

Endterm WS16/17: https://drive.google.com/open?id=0BxLtAtPNIIYQUVhzNHY5NlVHbVE

Libraries and external Software

Contains a list of the libraries and external software used by this system.

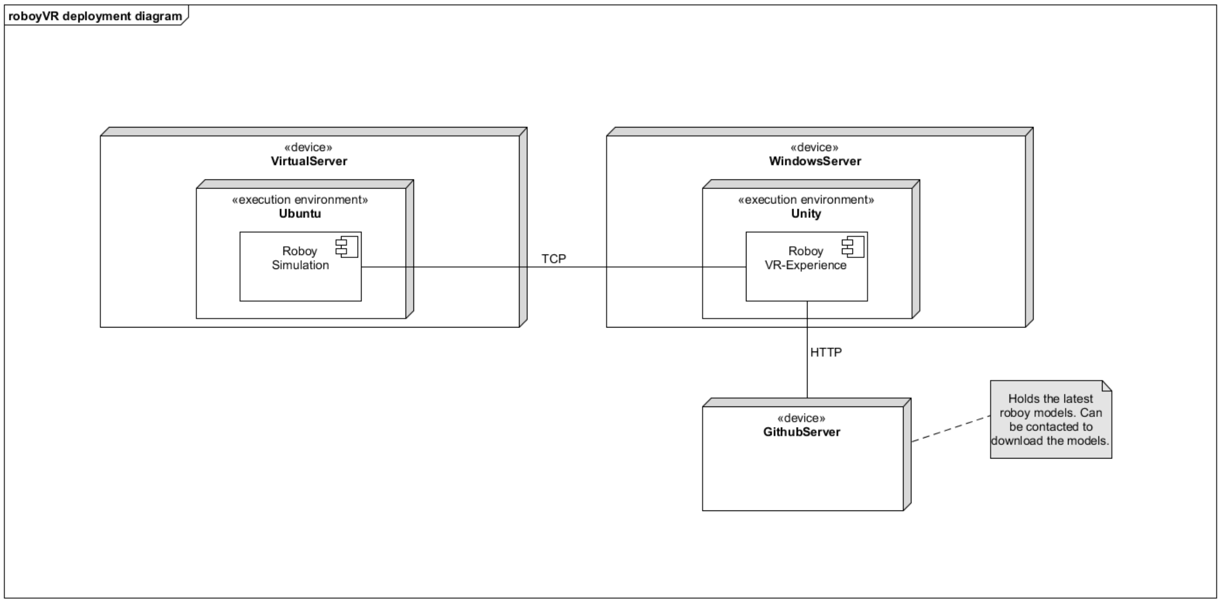

Fig. 2.12: Roboy simulation runs on a virtual machine, RoboyVR Experience runs on Unity.

Todo

List libraries you are using

Table 2.7: Libraries and external Software

Name	URL/Author	License	Description
Unity	https://unity3d.com/	Creative Commens Attribution license.	Game engine for developing interactive software.
SteamVR Plugin for Unity	https://www.assetstore.unity3d.com/en/#!/content/32647	Creative Commens Attribution license.	Unity-Plugin for HTC Vive Headset support.
Vuforia Plugin for Unity	https://developer.vuforia.com/downloads/sdk	Creative Commens Attribution license.	Unity-Plugin for a VR interface.
Blender	https://www.blender.org/	Creative Commens Attribution license.	Tool for modeling and animating.
Oracle Virtual Machine	https://www.oracle.com	Creative Commens Attribution license.	Tool to run a virtual machine.
arc42	http://www.arc42.de/template/	Creative Commens Attribution license.	Template for documenting and developing software

About arc42

This information should stay in every repository as per their license: http://www.arc42.de/template/licence.html

arc42, the Template for documentation of software and system architecture.

By Dr. Gernot Starke, Dr. Peter Hruschka and contributors.

Template Revision: 6.5 EN (based on asciidoc), Juni 2014

Note

This version of the template contains some help and explanations. It is used for familiarization with arc42 and the understanding of the concepts. For documentation of your own system you use better the *plain* version.

Literature and references

Starke-2014 Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden. Carl Hanser Verlag, 6, Auflage 2014.

Starke-Hruschka-2011 Gernot Starke und Peter Hruschka: Softwarearchitektur kompakt. Springer Akademischer Verlag, 2. Auflage 2011.

Zörner-2013 Softwarearchitekturen dokumentieren und kommunizieren, Carl Hanser Verlag, 2012

Examples

- HTML Sanity Checker
- DocChess (german)
- Gradle (german)
- MaMa CRM (german)
- Financial Data Migration (german)

Acknowledgements and collaborations

arc42 originally envisioned by Dr. Peter Hruschka and Dr. Gernot Starke.

Sources We maintain arc42 in *asciidoc* format at the moment, hosted in GitHub under the aim42-Organisation.

Issues We maintain a list of open topics and bugs.

We are looking forward to your corrections and clarifications! Please fork the repository mentioned over this lines and send us a *pull request*!

Collaborators

We are very thankful and acknowledge the support and help provided by all active and former collaborators, uncountable (anonymous) advisors, bug finders and users of this method.

Currently active

- Gernot Starke
- Stefan Zörner
- Markus Schärtel

- Ralf D. Müller
- Peter Hruschka
- Jürgen Krey

Former collaborators

(in alphabetical order)

- Anne Aloysius
- Matthias Bohlen
- Karl Eilebrecht
- Manfred Ferken
- Phillip Ghadir
- Carsten Klein
- Prof. Arne Koschel
- Axel Scheithauer

Index