RUIS for Unity 1.10

- Tuukka Takala	*technical design, implementation*
- Heikki Heiskanen	*implementation*
- Mikael Matveinen	*implementation*

For updates and other information, see http://ruisystem.net/
For help, visit our forum: https://forum.ruisystem.net/

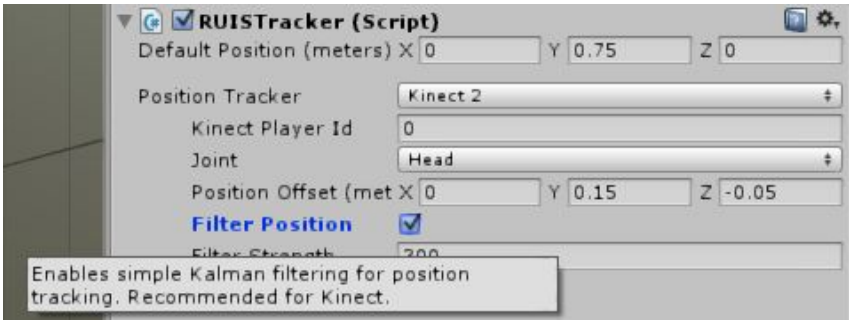
Introduction

RUIS (Reality-based User Interface System) gives hobbyists and seasoned developers an easy access to the state-of-the-art interaction devices, so they can bring their innovations into the field of virtual reality and motion controlled applications. Currently RUIS for Unity includes a versatile **display manager** for handling several display devices, and supports the use of **Kinect v1, Kinect v2, HTC Vive, Oculus Rift** and **PlayStation Move** together **in the same coordinate system**. This means that avatars controlled by Kinect can interact with virtual tools represented by OpenVR controllers; a player can see and control the full body of their avatar, and grab a Vive controller that is rendered as a Swiss army knife within the application for example.

Quickstart

Try example scenes at \RUISunity\Assets\RUIS\Examples\ -directory. You can develop and test your own motion controlled applications even if you have only a mouse and keyboard, because in RUIS they can emulate 3D input devices. If you want to use HTC Vive or Oculus Rift together with Kinect v1 or v2, open the OpenVRExample scene and read the "*Using multiple motion trackers with a common coordinate systems*" section of this document. **Vive developers should note that the term 'OpenVR' is used to substitute 'Vive' throughout this document, RUIS variable names, and tooltip help.**

Most RUIS scripts have extensive tooltip information, so hover the mouse cursor over any variables of RUIS components in Unity Editor's Inspector tab to learn about RUIS.



Requirements

Device	Windows	OSX	Additional details
HTC Vive	X		Steam and SteamVR required. **All OpenVR compatible VR headsets and controllers are supported.**
Oculus Rift	X		Latest Oculus Home required. Oculus Touch controllers work only through OpenVR, which requires SteamVR.
Kinect v1	X		32-bit Unity Editor and standalone builds only. Win32-bit version of OpenNI 1.5.4.0 required.
Kinect v2	X		Windows 8.1 and 10 only. Windows Kinect SDK 2.0 October 2014 release (2.0.1410) required.
PS Move	X	X	PlayStation 3, PS Eye, and Move.me software required.
Razer Hydra	X	(X)	Enabling Razer Hydra in OSX's Unity Editor or 64-bit builds causes instability. 32-bit builds should be fine.

Known issues

- **Shadows work only when rendering for head-mounted displays. This is a Unity bug related to custom projection matrices. To enable correct shadows for other displays, see this guide.**
- **Kinect v2: If the avatar is too shaky, then enable "Filter Rotations" from "RUIS Skeleton Controller" component. This filtering currently causes a lot of garbage collection, potentially resulting in poor framerates.**
- **RUISSelectableHingeJoint component used in *BigLever* and *ThrustLever* prefabs has a bug: the closer its orientation in world coordinates upon startup is to Euler angles (90, 0, 0), the less the hinge interaction works. Orthogonal orientations such as (0, 90, 0) work perfectly, even if they are rotated to (90, 0, 0) after startup.**

Installation

RUIS for Unity requires Unity 5.4 or later (both Windows and OSX are supported). It has been tested with **version 5.4.0F3**.

Optional drivers and software

- Kinect v1 and PrimeSense sensors are supported only via OpenNI software
- Kinect v2 is supported via Kinect for Windows SDK 2.0 (*Windows 8 and 10 only*)
- Oculus Rift requires the latest Oculus Home
- All other head-mounted displays, including HTC Vive, require Steam and SteamVR
- PS Move controllers are supported via Move.me software for PlayStation 3

Installing a head-mounted display

If you only intend to use Oculus Rift, download and install Oculus Home. If you are also using Oculus Touch controllers or any other head-mounted displays such as HTC Vive, then do the following: go to Steam website, download and install Steam. Upon installation, you need to create a Steam account. After signing in, use Steam's search tool to find SteamVR software and install it. **Make sure that your RUIS project has the "Virtual Reality Supported" option enabled in Unity's "Player settings".**

Installing Kinect v2

Go to Kinect for Windows website, download Kinect for Windows SDK 2.0 and install it. We have tested that RUIS works at least with October 2014 release (2.0.1410). Use the Kinect Configuration Verifier tool to ensure that your system is compatible with Kinect v2.

Installing OpenNI for Kinect v1 / ASUS Xtion / PrimeSense Sensor

You only need to follow through this section if you plan to use Kinect v1 with RUIS on your computer, otherwise you can skip this section. RUIS for Unity takes advantage of "OpenNI Unity Toolkit" that requires **Win32-bit version of OpenNI** 1.5.4.0. **For Kinect v1 only 32-bit Unity Editor and Windows standalone builds are supported**. If you have Kinect for Windows v1 (as opposed to Kinect for Xbox 360) you should also read the Troubleshooting section in the end of this readme.

You need to install OpenNI and NITE middleware before using Kinect v1 in RUIS. Check your installation validity by running the NiSimpleViewer example application at \OpenNI\Samples\Bin\Release\ directory. If it shows depth image from Kinect v1, you have successfully installed OpenNI.

Kinect v1 and Windows 7 / Vista / XP

Before installing Kinect v1, make sure that you have uninstalled all existing OpenNI, NITE, Primesense, and SensorKinect instances from Control Panel's "Uninstall a program" section, and reboot your computer. Download the following OpenNI installation file package:
https://drive.google.com/file/d/0B0dcx4DSNNn0WVFwVExDRnBBUkk/edit?usp=sharing

Unzip the downloaded package, and install its files in the following order (If the download link was dead, you need to google for the below files):

1. OpenNI-Win32-1.5.4.0-Dev1.zip
2. NITE-Win32-1.5.2.21-Dev.zip
3. SensorKinect093-Bin-Win32-v5.1.2.1.msi
4. Sensor-Win32-5.1.2.1-Redist.zip

Kinect v1 and Windows 8/10

Using the same files as for Windows 7, follow this procedure to install drivers for Kinect v1:

1. Uninstall any existing OpenNi, Nite, and the Kinect v1 drivers.
2. Windows key + R to open the run prompt
3. shutdown.exe /r /o /f /t 00
4. Select Troubleshoot
5. Select Advanced
6. Select Windows startup and then restart
7. Enter the option for Disable Driver Signature
8. Reinstall OpenNi (32-bit version), Nite, and the Kinect v1 driver.

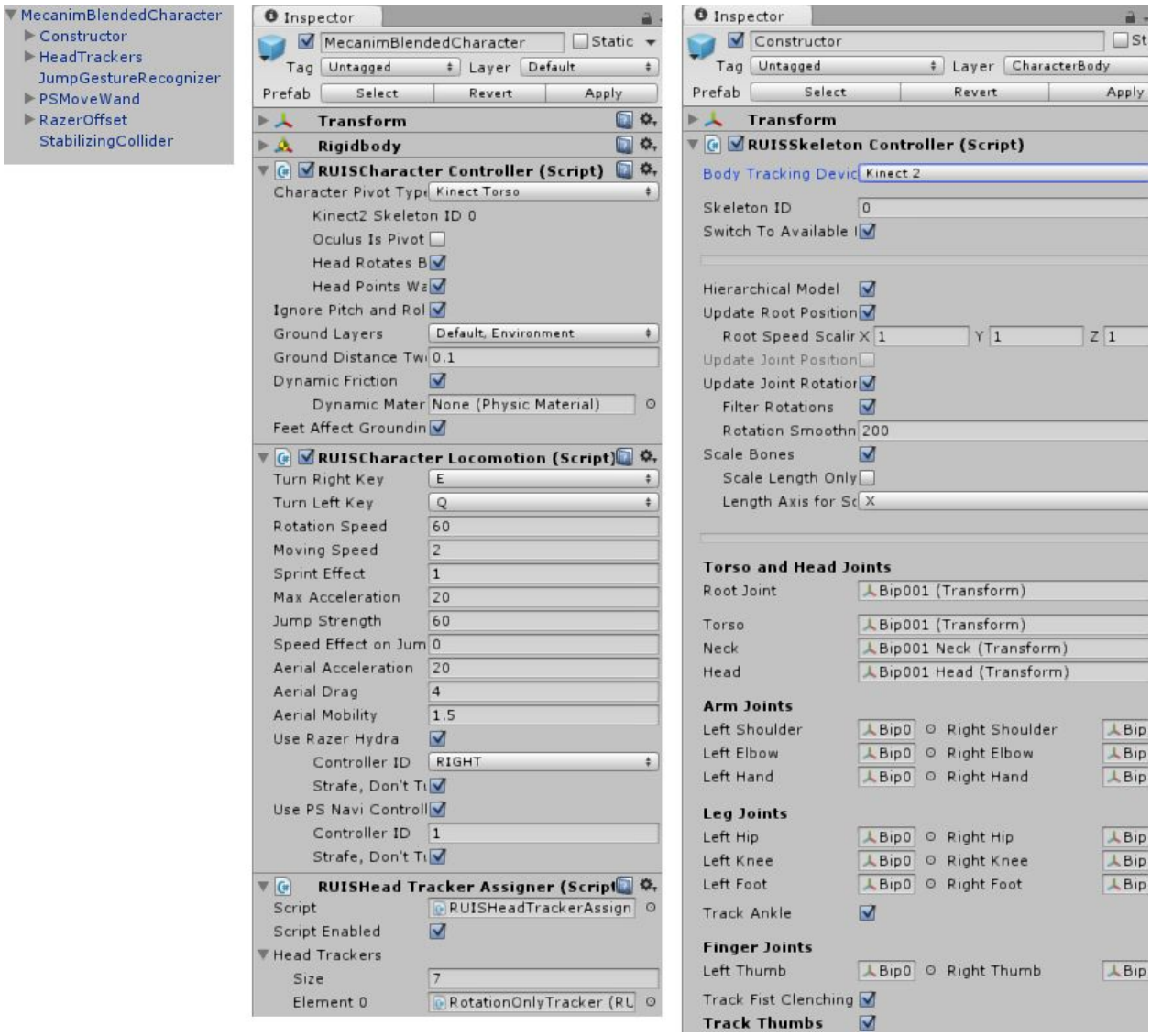
Kinect v1 and OSX

Kinect v1 for OSX is not supported. RUIS uses "OpenNI Unity Toolkit" that supports Win32-bit version of OpenNI only.

VR Headset and Motion Controllers with Kinect

RUIS features *MecanimBlendedCharacter* prefab, which is a feature-rich character controller for applications with first- or third-person view. This prefab's scripts automatically select the most suitable head-mounted display tracking between Oculus Camera, OpenVR tracking (e.g. Lighthouse base stations), Kinect, PS Move, or Razer Hydra. The head tracking device is decided at runtime depending on which devices are enabled in RUIS' InputManager or otherwise detected. Note: As of RUIS 1.10, the automatic head position tracker selection is mostly relevant for Oculus Rift DK1 and similar devices, which have not been tested with RUIS 1.10 since we don't posses such devices anymore. If "Virtual Reality Supported" option in enabled in Unity's "Player settings" and a head-mounted display is detected, then the OpenVRHeadTracker is selected upon startup. If that is not the case and all the input devices are disabled in RUIS' InputManager, then WithoutHeadTracking is selected.

The *MecanimBlendedCharacter* prefab contains a human 3D model that is animated with Kinect v1, Kinect v2, or a “Generic Motion Tracker”. You can substitute the default model with your own. Mecanim walking animation overtakes pose input from Kinect whenever the player is moving the character either with keyboard, gamepad, OpenVR (e.g. Vive) controller [support coming soon], PS Move Navigation controller, or Razer Hydra controller. You can **use your own Mecanim animation graph** and use RUIS features to write a script that **blends Mecanim animation with Kinect pose data in real-time**. See KinectTwoPlayers or OpenVrExample at \RUISunity\Assets\RUIS\Examples\ and modify the *MecanimBlendedCharacter* gameobjects to get started.



You can make an application with a third-person view using *MecanimBlendedCharacter* if you remove its *RUISHeadTrackerAssigner* component and the *HeadTrackers* gameobject parented under it, and create a new RUISCamera that follows the *MecanimBlendedCharacter*. A more simplified version of *MecanimBlendedCharacter* is *ControllableCharacter* prefab, that is

animated by Kinect and has the same features but does not include components for blending Mecanim animation.

By default the *MecanimBlendedCharacter* is affected by gravity, so you should place it on a static Collider. It has *OpenVRWands*, *PSMoveWand,* and *RazerOffset* child gameobjects, which contain 3D Wands that can grab and manipulate objects. You can delete or modify those objects. The *JumpGestureRecognizer* child gameobject contains scripts that make the character jump if Kinect is enabled and detects the player jumping. The recognition is far from perfect however, and you can disable the *JumpGestureRecognizer* if you like.

The *MecanimBlendedCharacter* prefab uses the Constructor character 3D model provided by Unity. **You can replace the prefab's 3D model with your own avatar model.** In that case you need to relink the Joint Transforms in the RUISSkeletonController component (see the right side of the above image), found in the *Constructor* gameobject. If your avatar's rig has vertices attached to the root Transform, then link that Transform both to "Root Joint" and "Torso" in RUISSkeletonController. If you use Kinect v2 and have "Track Fist Clenching" enabled, you should make sure that your avatar rig's finger joints all include the substring "finger" or "Finger" in their name. See this guide to make sure that you maintain all the correct scripts and links between them when replacing *MecanimBlendedCharacter*'s 3D model with your own. If you don't use Kinect to animate your character, then you might also need to adjust the Y-coordinate of the *StabilizingCollider* gameobject's Transform.

Using only a head-mounted display

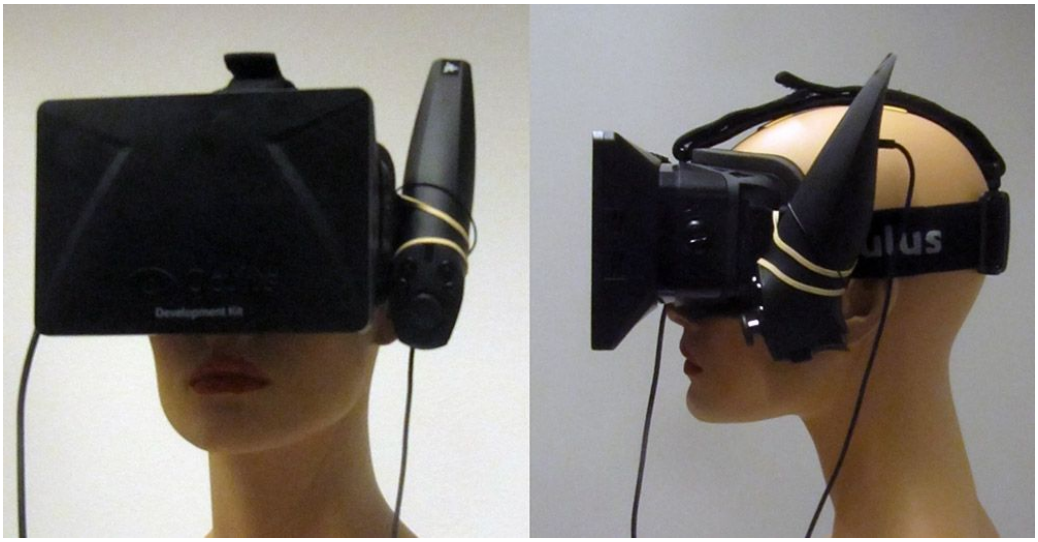
Use mouse and keyboard for controlling the *MecanimBlendedCharacter*. See *Avatar Controls* section below. If you are not using body-tracking devices like Kinect, and instead use only OpenVR compatible tracking devices (e.g. Lighthouse Basestations, Oculus Cameras), then the avatar position follows the head-mounted display's location, in which case you should do the following: find the *InputManager* gameobject that is parented under *RUIS* gameobject, change the "Master Coordinate System Sensor" to OpenVR. **Experiment with different combinations of "Head Rotates Body" and "Head Rotates Walking Direction" parameters** toggled on and off in RUISCharacterController component, to see which head-mounted display control schemes are available to you.

Head-mounted display + Kinect v1 or v2

Place the Kinect so that it can see you and the floor. The avatar's pose and limbs' length is tracked by Kinect. With Kinect v1 you need to stand in front of the Kinect during gameplay while wearing the head-mounted display, while Kinect v2 can also track you while you are sitting on a chair. You might need extension cords with Oculus Rift. **You need to calibrate the OpenVR and Kinect coordinate systems** by displaying the RUIS menu (ESC) in run-time, selecting "Kinect - OpenVR" from the Device Calibration drop-down menu, and clicking the "Calibrate Device(s)" -button. We recommend that you use a OpenVR compatible wireless controller (e.g. Vive controller, Oculus Touch) to control the *MecanimBlendedCharacter* locomotion. **NOTE: RUIS menu can only be interacted with by using a mouse.**

Oculus Rift + Razer Hydra (LEGACY)

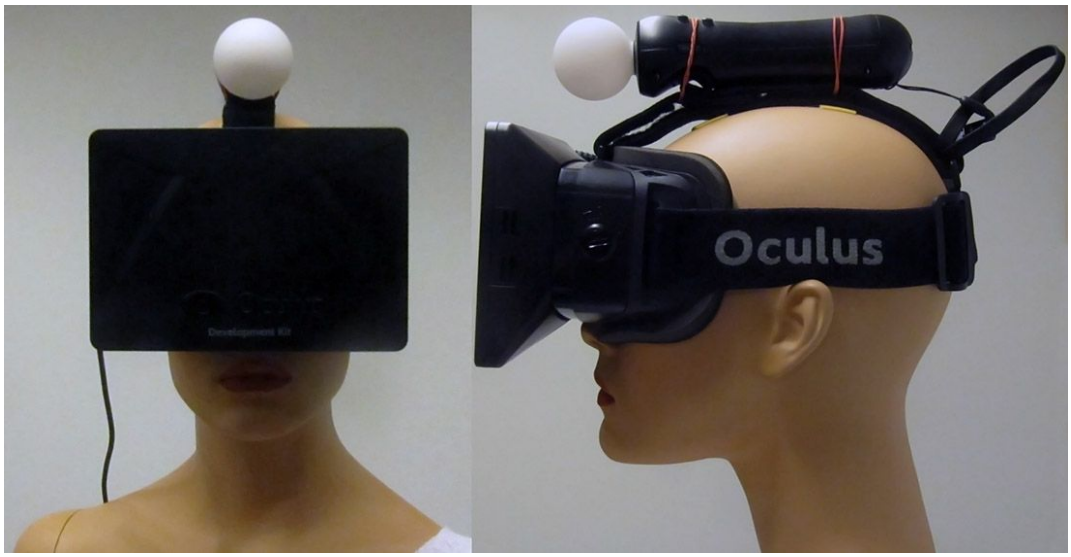
Place the Razer Hydra **base station on your desk** so that its cable ports are facing away from you, like you would do with any Razer Hydra game. One controller (RIGHT) will be wielded normally in your hand and is used for avatar locomotion, grabbing objects, etc., while the other (LEFT) needs to be attached on the **left side of your head** for head tracking:



You can use e.g. a rubber band to tie the Razer onto the Rift's strap. When the scene starts, it asks you to point the LEFT controller (on the head) towards the Razer Hydra base station and to press the trigger button. Same is repeated for the hand-held RIGHT controller.

Oculus Rift + PlayStation Move (LEGACY)

Attach the PS Move controller designated as GEM[0] in Move.me software on the topmost strap of Oculus Rift (two rubber bands work well):



The strap and the rubber bands should be kept tight to minimize any controller wobble when you move your head. The Move button should be pointing up towards the ceiling when you're

standing straight. PS Move GEM[1] acts as a 3D Wand that can grab and manipulate objects. If you want to use PS Navigation controller to make the *MecanimBlendedCharacter* walk, run, and jump, then be sure that the "Controller ID" in RUISCharacterLocomotion component of *MecanimBlendedCharacter* corresponds to the ID that can be seen in the Controller Settings of PlayStation. You can access those settings if you press and hold the PS Navigation controller's PlayStation button.

If both PS Move and Kinect are enabled, then you need to calibrate their coordinate systems by displaying the RUIS menu (ESC) in run-time, selecting "Kinect 1 - PS Move" from the Device Calibration drop-down menu, and clicking the "Calibrate Device(s)" -button (see *Example: Kinect and PS Move calibration* section for details).

If PS Move is enabled and Kinect is disabled, you may need to edit y-value of *translate* element in the file 'calibration.xml' for the head position to appear at correct altitude.

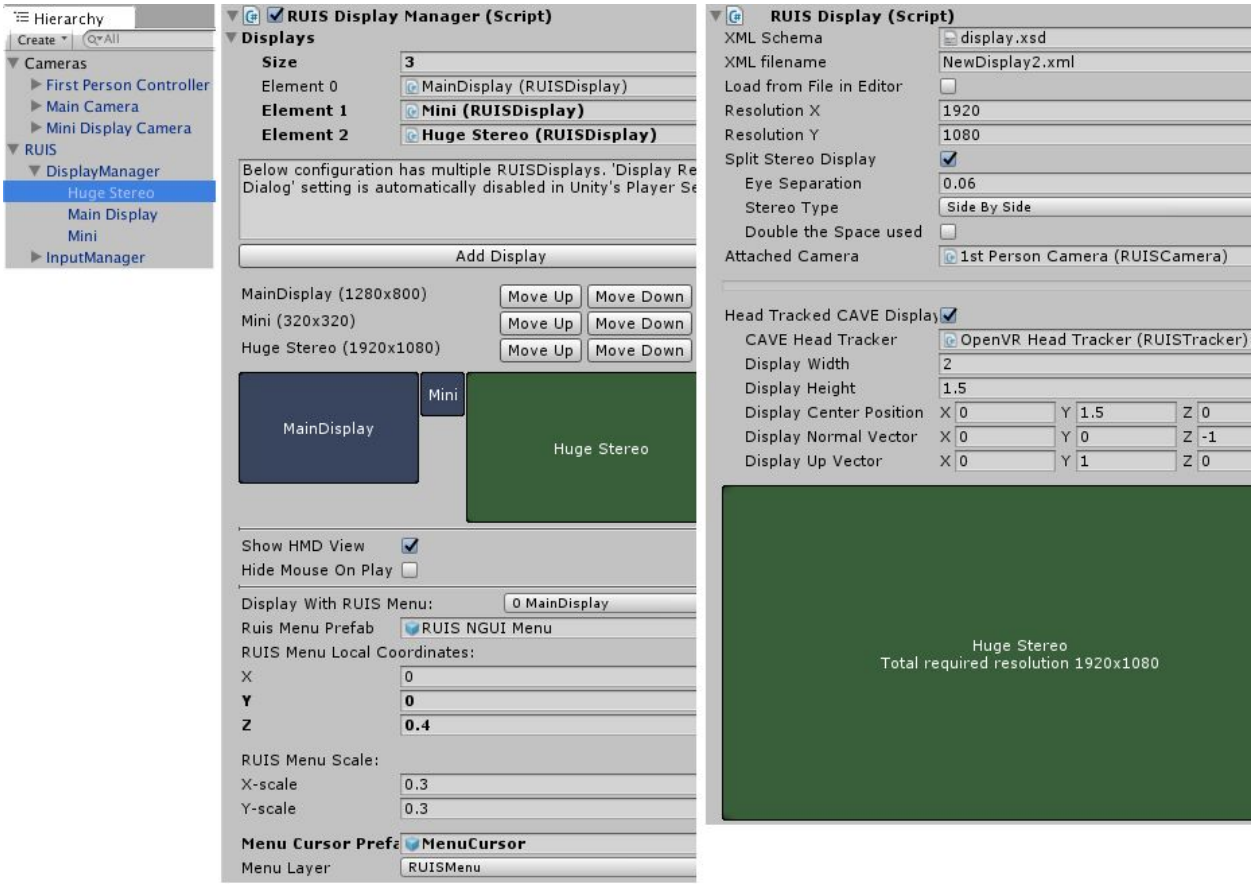
Display Manager

You can have your Unity application render 3D graphics on **any number of mono and stereo displays**, together with one head-mounted display, when you use RUIS and run your application in windowed mode. You need to have your displays arranged sideways in your operating system's display settings, because RUIS automatically creates a game window where all the viewports are side-by-side. **When you intend to use multiple displays, change the "D3D9/11 Fullscreen Mode" settings to "Fullscreen Window" from Unity's Standalone Player Options**.

RUIS display configuration can be edited through the *DisplayManager* gameobject that is parented under *RUIS* gameobject. The RUISDisplayManager script shows an overview of your current display setup, whose individual displays are parented under the *DisplayManager* gameobject. When adding new displays in RUIS, keep in mind that **each *RUISDisplay* needs to have a *RUISCamera* gameobject linked to it** when you want something to be rendered on those displays. If you have a RUISHeadTrackerAssigner script (comes with the *MecanimBlendedCharacter* prefab) in your scene, it will attempt to link one of its RUISCameras to any RUISDisplay without an existing link.

If you want to simultaneously render to a head-mounted display and other displays, then you should disable the "Show HMD View" option in RUISDisplayManager.

A basic example of using RUISDisplayManager to create a multi-display setup without head-mounted displays is presented in the DisplayManagerExample, which you can find at \RUISunity\Assets\RUIS\Examples\ folder. Please note that currently RUISDisplayManager does NOT utilize Unity's recently added multi-display capabilities. We will explore that prospect in the future.



RUISCamera

Each scene in a RUIS project should be rendered using a *RUISCamera*, a prefab that can be found in \RUISunity\Assets\RUIS\Resources\RUIS\Prefabs\Main RUIS\ folder. The "*MecanimBlendedCharacter*", "*RUIS OpenVR Rig*", and "*ControllableCharacter*" prefabs also come equipped with RUISCameras. **If the "Virtual Reality Supported" option is enabled, then RUISCamera will by default render to a head-mounted display. If the "Show HMD View" is also enabled in RUISDisplayManager, the default RUISCameras will also render on the game window, possibly covering other rendered viewports. This will occur even if the RUISCameras are not linked to RUISDisplays when VR is enabled in Unity,** because we wanted to keep layered rendering simple for head-mounted displays. **You can enforce RUISCamera to always act as a non-head-mounted display camera by setting the "Target Eye" to "None (Main Display)"** in the Camera component of RUISCamera->CenterCamera. Please note that the aforementioned variable is hidden by Unity if "Virtual Reality Supported" option is disabled.

RUISCamera also contains LeftCamera and RightCamera child gameobjects, each with Camera components. They are only used for stereo 3D displays, when the RUISCamera is linked to a RUISDisplay that has the "Split Stereo Display" option enabled. For stereo 3D displays, side-by-side and top-and-bottom modes are supported.

RUISCamera component has only two visible variables in the Unity Inspector: Horizontal and Vertical FOV. They only have effect if the RUISCamera is acting as a non-head-mounted display camera, rendering on a RUISDisplay that has the “Head Tracked CAVE Display” option disabled.

Other Features

3D user interface prefabs for selection and manipulation

RUIS for Unity can be used to easily create a custom 3D user interfaces with custom selection and manipulation schemes by the use of so called Wand prefabs. Currently supported wands (input devices) are: **MouseWand**, **OpenVRWands**, **PSMoveWand**, **RazerHydraWand**, and **SkeletonWand** (Kinect). These prefabs are found at \RUISunity\Assets\RUIS\Resources\RUIS\Prefabs\Main RUIS\Input Methods\. To see how to use these prefabs, check out BowlingAlley (*OpenVRWands*), MinimalScene (*MouseWand*), OpenVrExample (*OpenVRWands*), and KinectTwoPlayers (*SkeletonWand*) at \RUISunity\Assets\RUIS\Examples\ .

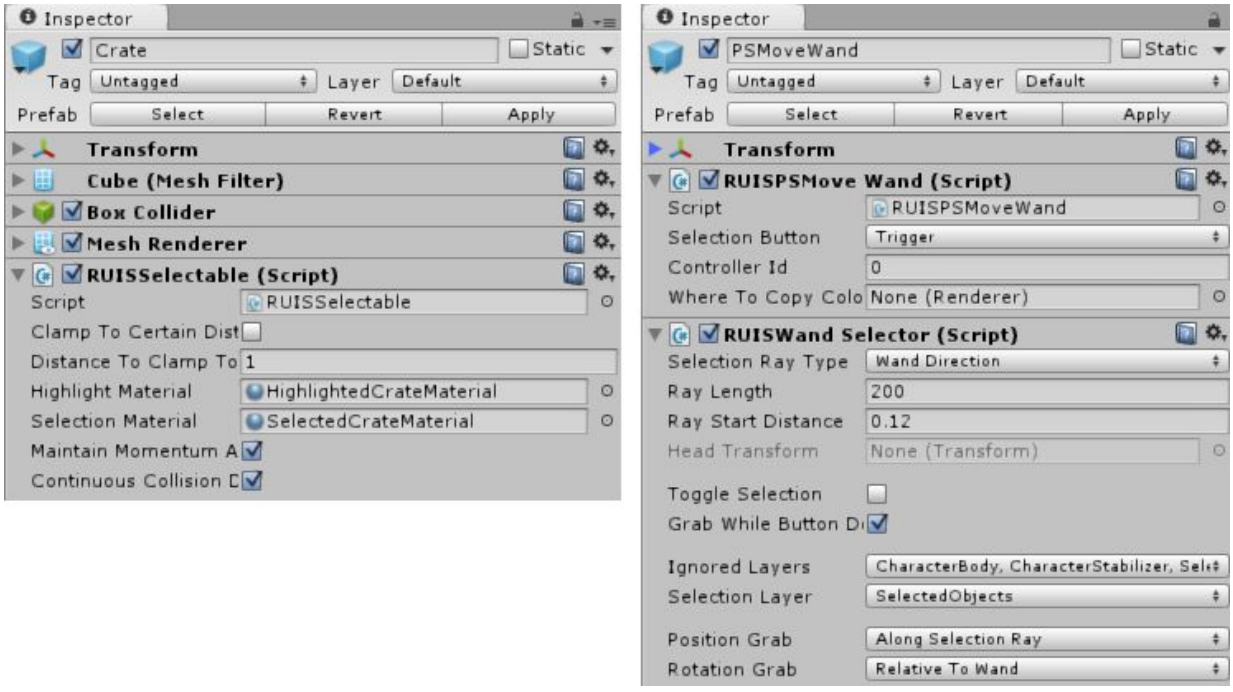
SkeletonWand is different from the other Wands because selection events are not activated via buttons, but using gestures. Currently the only **available selection gestures are *hold* and *fist* gestures** (latter is just for Kinect v2). Selection with the hold gesture works by holding the SkeletonWand (your hand) still for 2 seconds while pointing at the object to be selected. Release (deselect) works the same way. The fist gesture works if infrared features of Kinect v2 work properly, and this depends on your GPU drivers. You can test that by running the Kinect v2 Infrared Basics demo from SDK Browser v2.0, which should display an infrared image. For Nvidia GPUs, we have tested that the infrared features of Kinect v2 work at least with GeForce 340.52 drivers.

If you do not have Kinect, OpenVR controllers, Razer Hydra, or PS Move, then use MouseWand prefab that relies on 2D mouse for object manipulation purposes. When your scene is playing, the above mentioned Wands are used to manipulate gameobjects that have a **RUISSelectable** script, Mesh Renderer, Rigidbody, and Collider components. See the *Crate* gameobjects in any of our example scenes. The selection ray of a Wand is checked against the Collider components (you can have several of them in a hierarchy under one object) of a gameobject to see whether it can be selected (triggered with a button or a gesture in case of Kinect) and manipulated by the Wand.

In RUIS for Unity the **3D coordinate system unit is meters**, which is reflected in the position values of the Wands, Kinect-controlled avatars, and gameobjets with RUISTracker component. You can **translate, rotate, and scale the coordinate systems** of Wands by parenting them under an empty gameobject and applying the transformations on it.

Please note that in many 3D user interfaces it makes sense to disable gravity and other physical effects of the manipulated objects; For example, in a CAD interface you don't want geometric shapes to fall down after moving them.

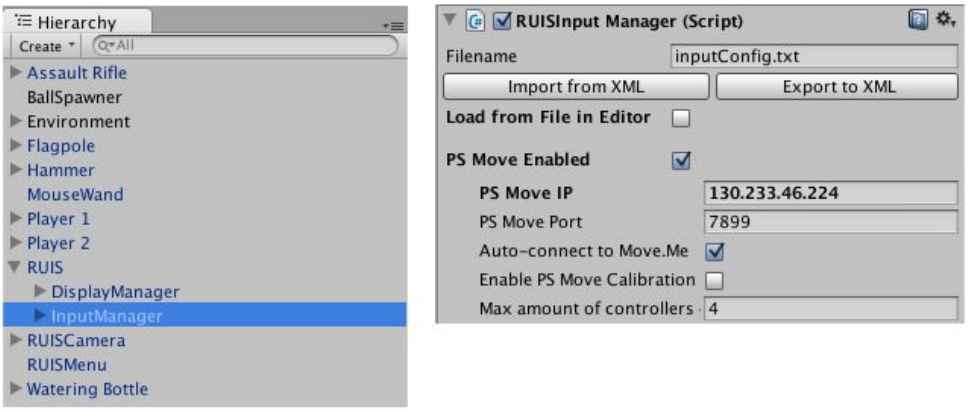
In the below figure's RUISPSMoveWand component, "Controller Id 0" corresponds to controller referred as GEM[0] in the Move.me screen. The "Selection Layer" is the Layer onto which the selected object will be transferred for the duration of the selection. **You can alter the object manipulation scheme** by changing the "Position Grab" and "Rotation Grab" options.



PlayStation Move controllers

If you have Move.me software for PlayStation 3 and want to use PS Move controllers in your RUIS for Unity scenes, tick the “PS Move Enabled” option in *InputManager* gameobject (parented under *RUIS* gameobject). Otherwise keep that option unchecked, because **the scene will crash or freeze for a long time when entering playmode if RUIS is trying to connect to a IP address that is not available**.

Be sure to set the IP address and port parameters of *InputManager* to correspond to the ones that Move.me software is displaying. When building your application with PS Move controller support, **remember to allow the executable file through your firewall** (both UDP and TCP) so that it can connect with Move.me server. Please note that pressing SELECT button will turn off the PS Move controller’s light and that controller is not tracked anymore. This is a feature of Move.me software.



Kinect controlled full-body avatars

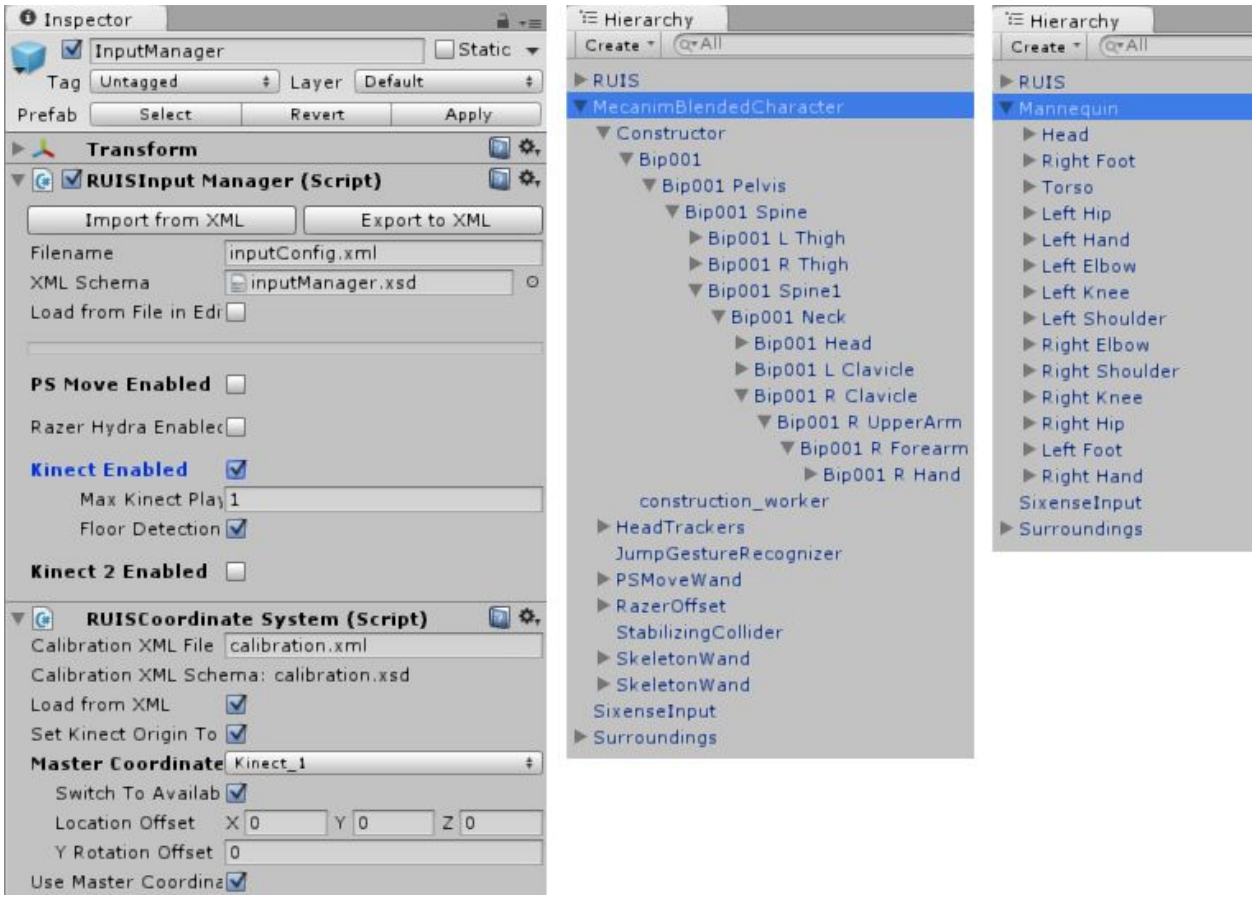
If you have successfully installed the drivers for Kinect v1 or v2 and want to use it in your RUIS for Unity scene, make sure to tick the “Kinect Enabled” / “Kinect 2 Enabled” option in *InputManager* gameobject (parented under *RUIS* gameobject). To get started, use the *ConstructorSkeleton, Mannequin, or MecanimBlendedCharacter* prefabs that are located in \RUISunity\Assets\RUIS\Resources\RUIS\Prefabs\Main RUIS\Common Objects\ . **You can replace the prefabs’ 3D models with your own.** Please note that you can use rigged models with either a hierarchical bone setup (e.g. *ConstructorSkeleton*) or a flat, one-level deep bone setup (e.g. *Mannequin*). For latter ones, you need to uncheck the “Hierarchical Model” option in the RUISSkeletonController script.

You can **translate, rotate, and scale your Kinect-controlled avatars** by parenting them under an empty gameobject and applying the transformations on it.

If you have “Floor Detection On Scene Start” enabled in RUISInputManager, or if you have calibrated Kinect using RUIS, you can take advantage of the following useful features:

1. **Set Kinect Origin To Floor** -feature can be turned on from the *InputManager* gameobject. With this option enabled the Kinect-controlled avatars will always have their feet on the XZ-plane (provided that the avatar gameobjects have model-dependent, correct Y-offsets), no matter what height your Kinect is placed on.

2. **You can tilt your Kinect downwards**, and RUIS will use a corrected coordinate system where the XZ-plane is aligned along the floor, preventing Kinect avatars from being skewed in Unity.



Using multiple motion trackers with a common coordinate system

If you have multiple motion tracking devices that you want use simultaneously with a shared coordinate system, then **choose one of the devices as the "Master Coordinate System Sensor" and enable the "Use Master Coordinate System" toggle** from the RUISCoordinateSystem component (see above image). The RUISCoordinateSystem component is located in the *InputManager* gameobject, which is parented under *RUIS* gameobject. Lets say that you chose *OpenVR* as the "Master Coordinate System Sensor", and you want to use it together with Kinect_2 and PS Move. Then you would need to calibrate two device pairs: Kinect_2–OpenVR and PS_Move–OpenVR. The following section will tell you how

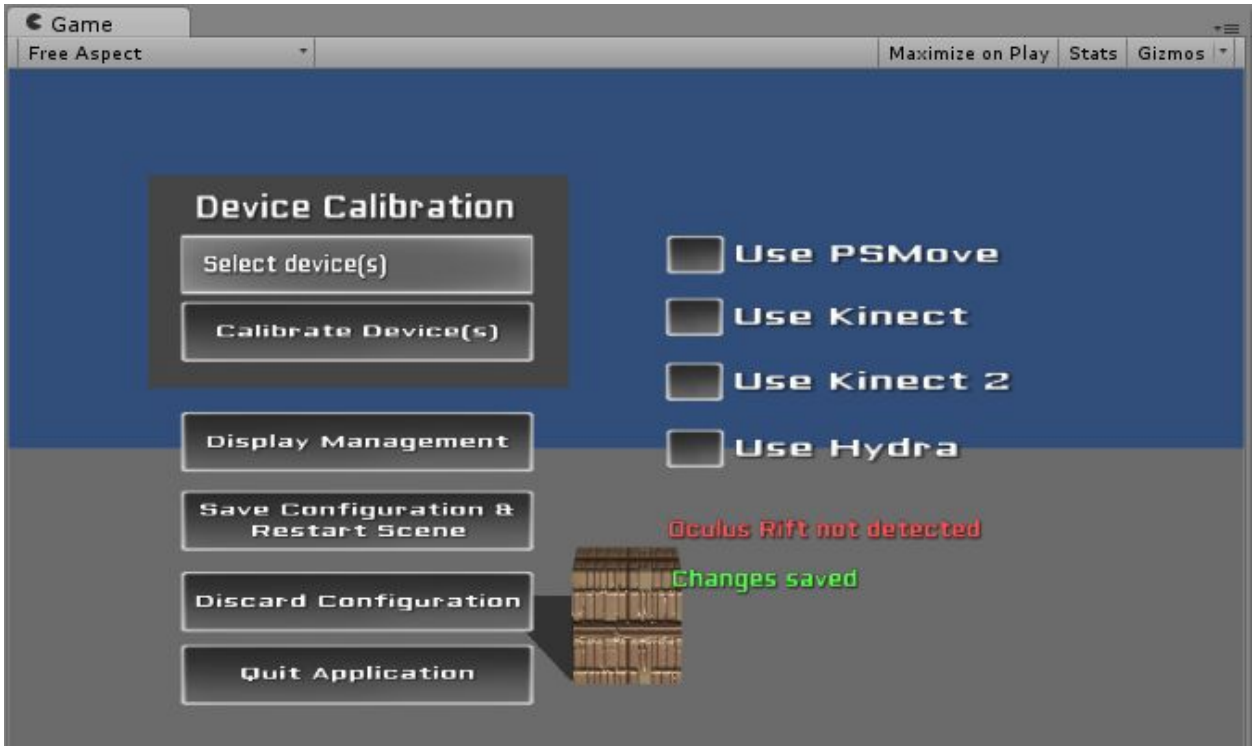
to do that. **NOTE: If you are using SteamVR with head-mounted displays, then it is best to set the "Master Coordinate System Sensor" to "OpenVR".** This is to avoid a bug in SteamVR that causes peculiar extra translations to head-tracked positions when non-uniform scale is applied by RUISTracker to RUISCamera by RUIS when attempting to make the OpenVR coordinate system to conform the other device's coordinate system.

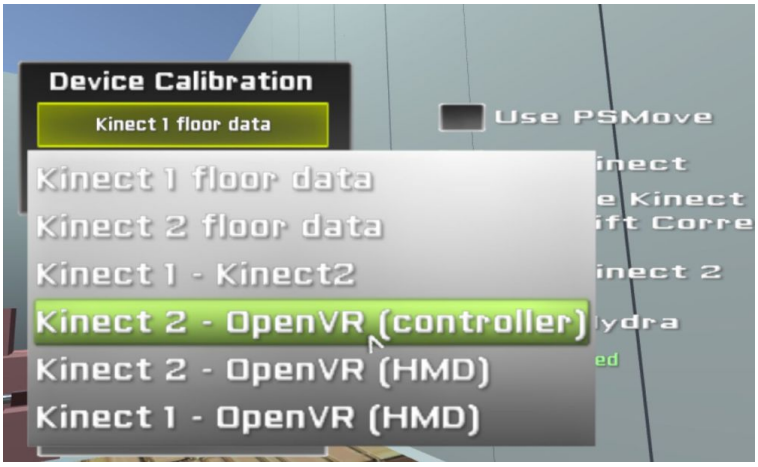
Calibrating two different motion trackers to use the same coordinate system

Calibration is needed for using two or more different motion trackers (e.g. **Kinect and OpenVR controllers**) together **in the same coordinate system**, and also for aligning Kinect v1 or Kinect v2 coordinate system with the room floor. Calibration needs to be performed only once, but you have to do it again if you move either one of the calibrated sensors. Results of the calibration (a 3x3 transformation matrix and a translation vector) are printed in Unity's output log and are also saved in an XML-file at \RUISunity\calibration.xml.

You can calibrate devices by running any of our example scenes in Unity Editor, pressing ESC key to show RUIS menu, enabling those devices whose drivers you have successfully installed, selecting the device(s) that you want to calibrate, e.g. "Kinect 2 - OpenVR (controller)", from the **Device Calibration drop-down menu**, and clicking the "Calibrate Device(s)" -button. This will start the interactive calibration process by loading \RUISunity\Assets\RUIS\Scenes\calibration.unity, which we will describe below in more detail for Kinect 2 and OpenVR controller.

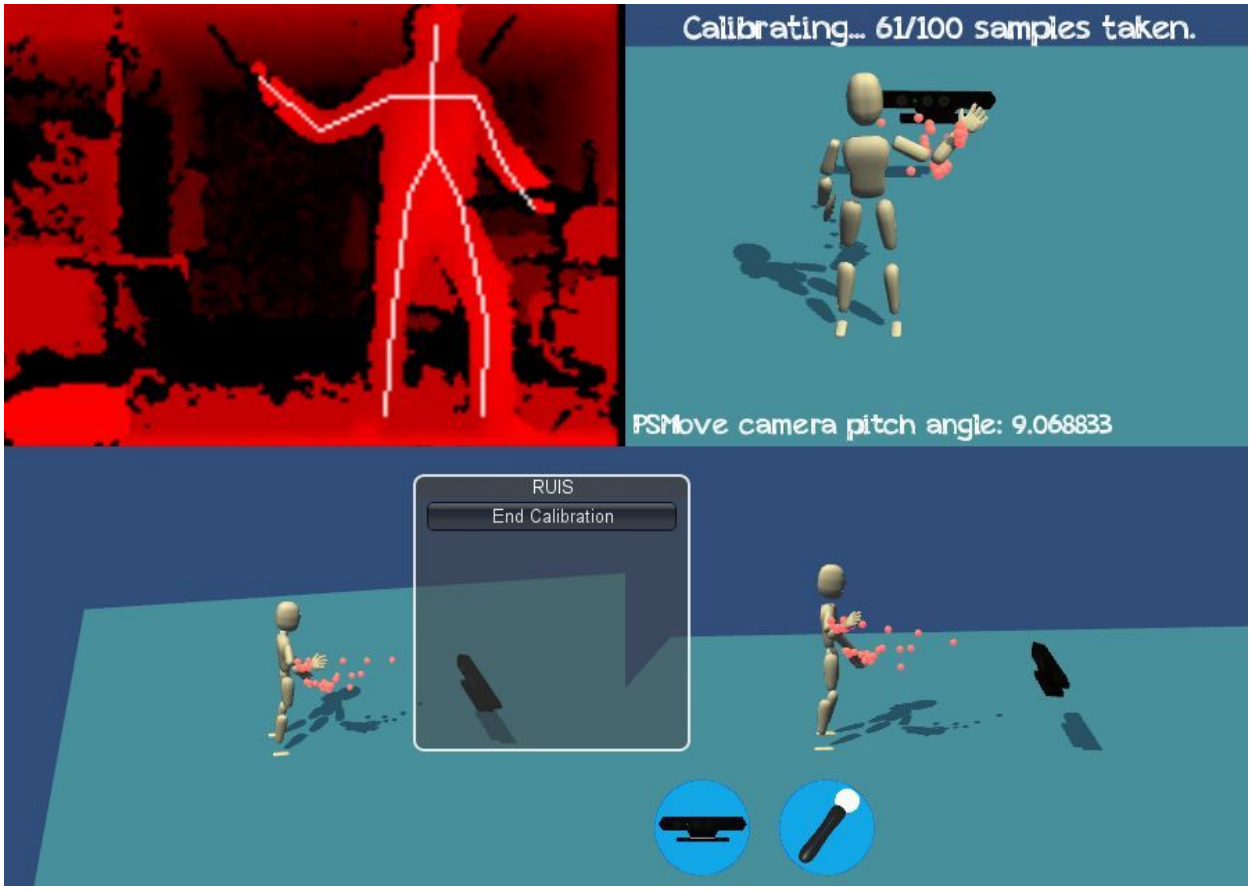
NOTE: All head-mounted displays can be calibrated by choosing the "[OtherDeviceName] - OpenVR (HMD)" option in the Device Calibration drop-down menu. This option has erroneously the same name even if you are calibrating Oculus Rift without using SteamVR, and the results will be saved to "OpenVR-[OtherDeviceName]" element in the calibration.xml file. **If you have Kinect 2 we recommend using "Kinect 2 - OpenVR (controller)" calibration method**, because it is currently the only process that allows interactive finetuning of the calibration result.





Example: Kinect 2 and OpenVR controller calibration

Once the calibration scene has loaded, hold OpenVR controller (e.g. Vive controller) in your right hand, and step in the front of the Kinect. The interactive calibration process instructs you to press the trigger button of the OpenVR controller to start the calibration. During the calibration process, keep the OpenVR controller in your right hand and move it slowly so that there is a clear line of sight between it and both the Kinect and OpenVR sensors (e.g. Lighthouse base stations).



When the calibration is complete, you will see the results. After calibrating Kinect 2 with OpenVR controllers, you can put on the OpenVR head-mounted display to see the avatar's first-person view. At that point **you can finetune the translation part of the calibration by pressing down the trackpad button of the OpenVR controller** (analog stick button in Oculus Touch) and moving the controller into the direction where you want to modify the translation.
For an example of how to use Kinect and OpenVR controllers (e.g. Vive) together, please see the BowlingAlley or OpenVrExample at \RUISunity\Assets\RUIS\Examples\

If you are holding a OpenVR controller in your hand, the Kinect-controlled avatar's hand and the OpenVR controller's virtual representation do not alway appear exactly in the same position because no calibration gives perfect results and Kinect is less accurate than OpenVR controllers. Snapping of hand locations to handheld OpenVR controller locations might be added in a future release of RUIS for Unity.

When you are calibrating Kinect v1 or v2 (just themselves or with other devices), **it is important that Kinect sees the floor properly** (see the red Kinect depth view in the above screenshot for an example), so that the Kinect can detect its distance from the floor and its orientation with regards to the floor. That data is used to align the Kinect coordinate system's XZ-plane with floor plane.

Example scenes

Examples of using RUIS for Unity can be found at \RUISunity\Assets\RUIS\Examples\
Following two points are important in BowlingAlley, KinectTwoPlayers, and OpenVrExample scenes:

1. Choose the *InputManager* gameobject (parented under *RUIS* gameobject) and **enable those input devices that you have connected to your computer**. Head-mounted displays and OpenVR controllers are automatically detected and you don't need to enable those.

2. If you have two or more input devices, select one of them as the "Master Coordinate System Sensor" from RUISCoordinateSystem component (select OpenVR if you are using SteamVR), start the scene, press ESC to open the RUIS menu, choose one of the device pairs from "Device Calibration", and click "Calibrate Device(s)". After the calibration is completed, repeat the process for the remaining device pairs that contain the "Master Coordinate System Sensor" (you can skip "Kinect floor data" calibration for Kinect v1 and v2).

OpenVrExample

This example presents *MecanimBlendedCharacter* gameobject, which is a versatile beast. In this demo Kinect v1 or Kinect v2 animate the player avatar, while head-mounted displays offer a first-person view from the eyes of the avatar. Additionally OpenVR controllers can be used to interact with objects. When the scene is running, you can control the constructor character with keyboard, gamepad, PS Move Navigation controller, or Razer Hydra controller (OpenVR locomotion coming later).

MinimalScene

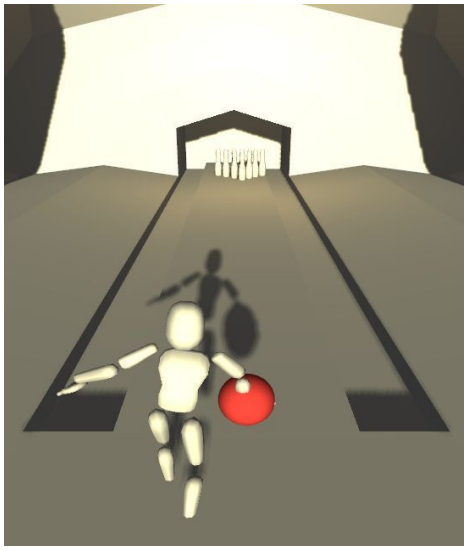
This scene is a good starting point for a blank RUIS scene. You can delete the Floor, Crate, Directional light, and MouseWand gameobjects. **If you want to change this scene to a minimal scene with OpenVR (e.g. Vive) controllers, then replace the RUISCamera gameobject with the *"RUIS OpenVR Rig"* prefab** (at "\RUISunity\Assets\RUIS\Resources\RUIS\Prefabs\Main RUIS\"). The "*RUIS OpenVR Rig*" prefab is RUIS' equivalent for SteamVR's *[CameraRig]* prefab.

KinectTwoPlayers

This example demonstrates how you can create a multiuser Kinect application in RUIS. The Kinect avatars are equipped with Collider components so that they can push objects around. Also notice how the Kinect-controlled SkeletonWands can be used for object manipulation.

BowlingAlley

Bowling with OpenVR controller (e.g. Vive controller): Use trigger button to grab the bowling ball and release it on your throw. Menu-button resets the bowling ball position, and trackpad-button places the bowling pins. Kinect is used to control a simple mannequin avatar (*Mannequin* gameobject). Notice how *Mannequin*'s body parts are all parented in a flat fashion and that the "Hierarchical Model" option is unchecked in its script, as opposed to *Constructor* gameobject parented under the *MecanimBlendedCharacter* gameobjects in KinectTwoPlayers and OpenVrExample scenes. Also note that the Mannequin model does not work well as an avatar viewed from first-person.



DisplayManagerExample

Run the scene to see how settings at *DisplayManager* gameobject affect the rendered multi-display output. Additionally you can use mouse, space-key, and WASD-keys to control a simple first-person movement. A MouseWand prefab is present, so you can use left mouse button to grab cube objects.

CaveExample

This example with three RUISDisplays illustrates how RUIS can be used for CAVE systems (with head tracking and all). **You can apply keystone correction for projector-based display walls** by accessing the RUIS menu with ESC-key when your scene is running, clicking "Display Management", and dragging the viewport corners. In this demo you can press the IJKLUO-keys on the keyboard to simulate head-tracking, which shows you how the asymmetric view frustums get distorted as the simulated head moves. Alternatively, **you can change the "Head Tracker" gameobject to get its position from some other source, like a Vive controller, Kinect head joint, etc**. For locomotion in the CAVE environment, just apply translation and rotation to the "Cameras" gameobject under which all the RUISCameras of the scene are. You can also add more RUISDisplays into the scene by using the "Add Display" button in RUISDisplayManager to

match your own physical CAVE setup. Remember to add a new RUISCamera and enable the "Head Tracked CAVE Display" for each new RUISDisplay, and configure the physical display parameters (Display Width, Height, Center Position, Normal Vector, Up Vector) to correspond the physical dimensions and tracking coordinates of your CAVE setup. Please note that **with RUIS it is possible create an application that simultaneously renders to a head-mounted display and a CAVE setup**, by configuring the RUISDisplayManager and RUISCameras. Alternatively, the application could switch between CAVE and head-mounted rendering, depending on detected devices (you need your own scripting to switch between RUISCameras in this case).

Avatar controls

ControllableCharacter and MecanimBlendedCharacter

	Keyboard	**Gamepad**
Move forward / backward	W / S	Left analog stick
Strafe left / right	A / D	Left analog stick
Turn left / right	Q / E	Right analog stick
Jump	Space	Joystick button 1, 5
Run	Shift	Joystick button 0, 4, 7

When Kinect and Jump Gesture are enable, you can jump in real life to make your avatar jump; You need to stand at least 2 meters away from the Kinect, and your both feet need to clearly lift from the ground.

	Razer Hydra (RIGHT, hand-held)	**PS Navigation controller (ID 1, hand-held)**
Move forward / backward	Analog joystick	Analog joystick
Strafe left / right	Analog joystick	Analog joystick
Turn left / right	Buttons 3 / 4	X / O
Jump	Bumper button	L1
Run	Joystick button	L2
		PS Move controller (GEM[1], hand-held)
Grab object	Trigger button	Trigger button

Troubleshooting

Kinect v1 issues

- OpenNI (and Windows SDK) often have problems tracking the user properly. This is manifested as limbs jumping around randomly, legs facing backwards, and other poor tracking results. **It is also very common that the lengths of different body parts are poorly detected**: arms are either too short or long, or legs are too big and go underground. Keep enough distance to Kinect (between 2 - 4 meters) so that it can see your whole body.
- For an example of how much floor area the Kinect should be able to perceive, see the screenshot from *Example: Kinect and OpenVR controller calibration* section.
- On some computers it is sometimes necessary to unplug Kinect's USB connector and plug it into a different USB port to get OpenNI examples working.
- **Kinect v1 floor detection works erratically on some systems.** Be sure that nothing blocks Kinect's view and that it can see enough floor area when starting a new scene or calibration process. If Kinect v1 floor detection doesn't work at all, then you should disable "Floor Detection On Scene Start" for Kinect v1. In this case the RUISCoordinateSystem script's "Set Kinect Origin To Floor" toggle works only if you manually edit the 'kinectDistanceFromFloor' value from calibration.xml. Such editing needs to be applied AFTER running Kinect calibration, because it resets the distance value. Your Kinect controlled characters might also appear leaning forward or backward, if your sensor is tilted downwards or upwards.

Kinect for Windows (Kinect v1)

Microsoft released Kinect for Windows v1 and Kinect v1 SDK, but they are not compatible with OpenNI. The kinect-mssdk-openni-bridge is an experimental module that connects Kinect SDK to OpenNI and allows people with Kinect for Windows to use OpenNI applications. This bridge _might_ get RUIS to work with Kinect for Windows but there are no guarantees:
https://code.google.com/p/kinect-mssdk-openni-bridge/

Razer Hydra

If you use Razer Hydra, **add the *SixenseInput* prefab into your scenes**, and toggle on 'Enable Razer Hydra' from *InputManager* gameobject (parented under *RUIS* gameobject).

If you use a head-mounted display together with Razer Hydra, you won't see the calibration instructions upon loading a scene: 1. Point the left controller towards the base station and then press the shoulder button. 2. Do the same with right controller.

When starting a scene with Razer Hydra, its buttons sometimes get "stuck" and for example the MecanimBlendedCharacter moves automatically even without touching the buttons. If this

happens, restarting the scene or unplugging and reconnecting the Razer Hydra USB cord can help. Razer Hydra can also sometimes get confused about directions or lose one controller altogether, in which case you need to restart the demo.

PS Move

Check that your computer and PlayStation 3 are connected to the same network, and that the PlayStation is able to obtain an IP address. Make sure that the address for Move.me server and port in InputManager gameobject is the same as displayed in the Move.me software on PlayStation. Also make sure that “Load from File in Editor” is disabled in the InputManager. If you successfully connect RUIS to Move.me server, the PlayStation3 screen should display something like this:

If RUIS for Unity is not able to connect to PlayStation via TCP (Move.me software displays “Connections: 0”), please check your firewall settings. If your application is connected to Move.me server but does not update PS Move state this may also be a firewall issue (Move.me sents PS Move state over UDP to RUIS).

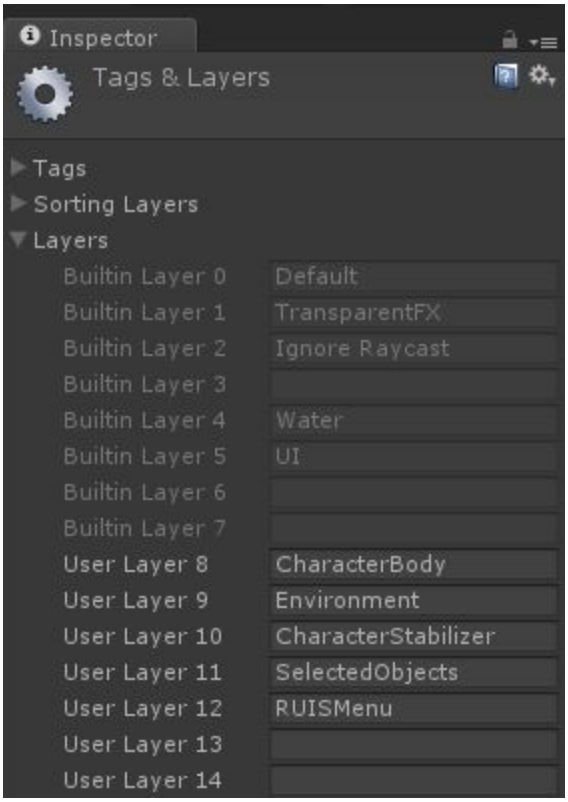
Unity editor and individual standalone executables have to be allowed through the firewall. In a standalone build you will have to set the IP address and port inside a file named inputConfig.txt that needs to be located in the same folder where the standalone executable file is. For an example of the file format please check the file provided in \RUISunity\ .

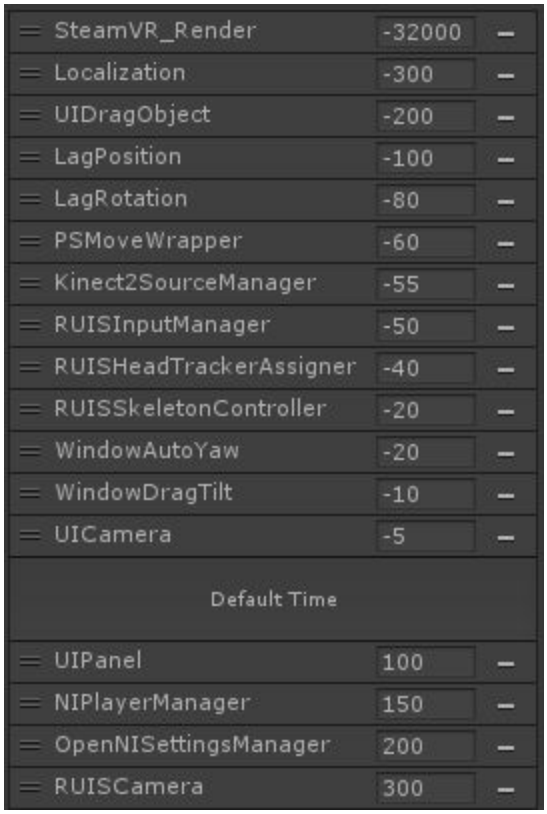
Before calibrating PS Move with any other devices, you should keep thrusting your PS Move controller towards and away from your PS Eye camera, until the “PS Eye pitch angle” in the

RUIS calibration scene's viewport converges within 0.1 degree. This is because PS Eye needs to see PS Move controller moving for awhile before Move.me software can reliably estimate the pitch orientation of PS Eye. When you calibrate after the pitch angle has converged, this ensures that the saved coordinate system calibration between Kinect and PS Move will be as accurate as it can be even after restarting your computer and PlayStation. Please note that Move.me running on PlayStation does not save the pitch angle, and after restarting it, the pitch angle converges again slowly while you are using the PS Move controller in front of the PS Eye camera. To get the best possible initial pitch angle, align the PS Move controller along the PS Eye camera's viewing axis when pressing the Move button to "light up" the controller in Move.me.

Layers, Script Execution Order, and creating a UnityPackage of RUIS

If you create a UnityPackage of RUIS with the intention of importing RUIS to your existing Unity project, you need to create the layers displayed in the below left image (with the same indices and names):





You also need to set up the Script Execution Order presented in above right image. Nearly half of the scripts come from NGUI, which we use in our in-game menu.

Safety Warning

Wearing head-mounted-displays while standing up, moving, walking, or jumping is dangerous to your health and potentially deadly. Author of this software recommends you to avoid the aforementioned actions, and if you choose to perform them anyway, you do it at your own risk. The author of this software cannot be held responsible in any way for any consequences.

Software License Limitation of Liabilities

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Licensing

RUIS is distributed under the LGPL Version 3 license for non-commercial use. If you intend to use RUIS for commercial work, please contact us first (tmtakala@gmail.com).