RUIS for Unity 1.21

- 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/

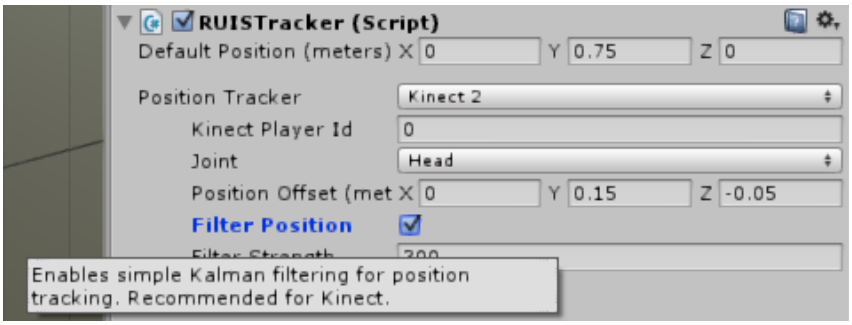
Introduction

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 **HTC Vive, Oculus Rift, Kinect** and other **motion capture systems** 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 their full body 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 *"Shared Coordinate Frame for Multiple Tracking Devices"* section of this document. **Vive developers should note that the term 'OpenVR' is used to substitute 'Vive' throughout this document, RUIS variable names, component properties, and tooltip help.**

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



Requirements

Device	Win	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.
Mocap (OptiTrack, Vicon, Perception Neuron, etc.)	X	X	**Any mocap system that can stream joint poses to Unity is supported.** Requires installing the Unity plugin of the mocap system.
Razer Hydra / PS Move	(X)	(?)	These legacy devices are now only supported via OpenVR.

Known Issues

- **When calibrating Oculus Rift coordinate frame with other devices, the Oculus Rift headset must be visible to the Oculus cameras, and the headset must be "worn" (i.e. its proximity sensor must be kept triggered).**
- **Kinect v1/v2: If the avatar is too shaky, then enable "Filter Rotations" from "RUIS Skeleton Controller" component. If you use other mocap systems, disable it.**
- **Extreme avatar limb thickness values create non-uniform scaling side effects which are compensated. A future release will mitigate the remaining issues: scaling of hands/feet and decelerated/magnified rotations around specific axes.**
- **When using RUISKinectAndMecanimCombiner script to blend animations into the real-time mocap avatar pose, then any (non-uniform) avatar limb thickness will not work correctly. This will be fixed in a near future release.**
- **Shadows become buggy with RUISCamera if used with CAVE displays or keystone correction. This is a Unity bug related to custom projection matrices. To enable correct shadows in the CAVE display case, see this guide.**

Installation

RUIS for Unity requires Unity 5.6.5 or later (both Windows and OSX are supported). It has been tested with **version 5.6.5f1**, but not with Unity 2017 or 2018.

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
- Unity plugin of your mocap solution (OptiTrack, Perception Neuron, Vicon, Xsens…)

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.

Full-body Avatars

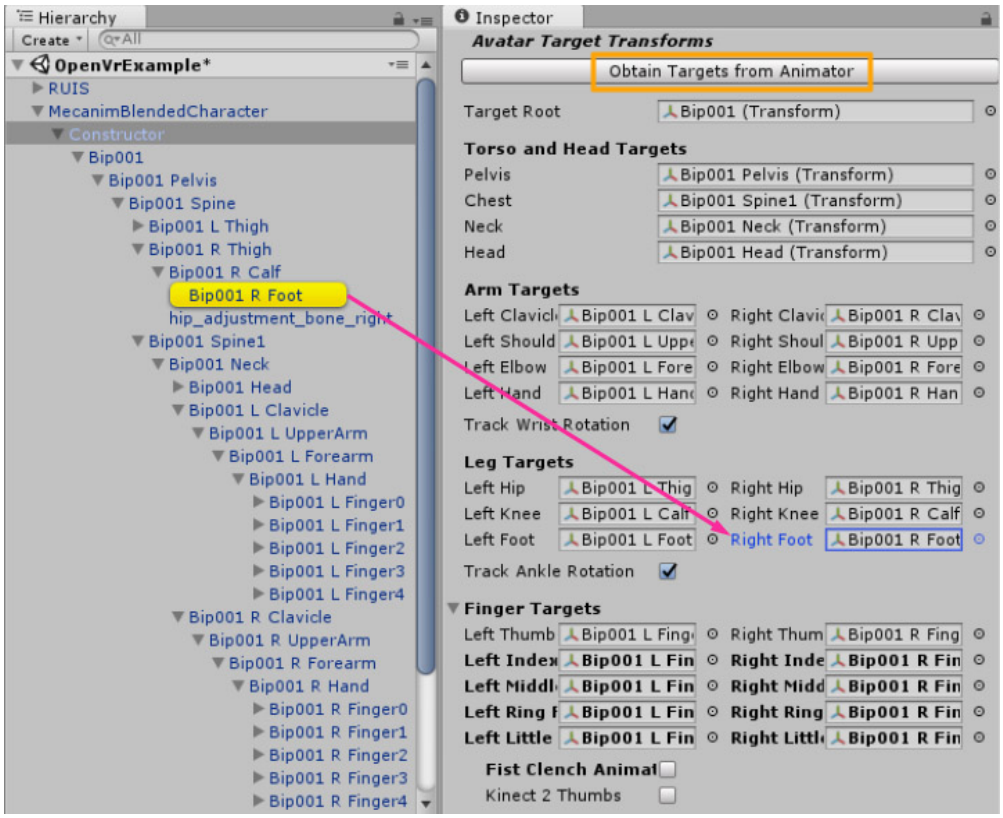
RUIS can be used with **any Unity compatible full-body motion capture (mocap) system** to animate **arbitrary humanoid 3D avatars** in real-time. RUIS comes with Kinect v1 and v2 Unity addons. For using input from other motion capture systems (e.g. Perception Neuron, OptiTrack) you need to install their respective Unity plugins.

The folder \RUISunity\Assets\RUIS\Resources\RUIS\Prefabs\Main RUIS\Common Objects\ contains five avatar prefabs that you can use in your scenes: *Mannequin, ConstructorSkeleton, ConstructorSkeletonWithColliders, ControllableCharacter, and MecanimBlendedCharacter*. These avatar prefabs and their differences are listed in the below table (prefab names are on the top row). **RUISSkeletonController script is the main component for using avatars in RUIS**, and each prefab comes with it. The use of *Mannequin* prefab is not recommended, which is explained in BowlingAlley scene description at the *"Example Scenes"* section.

	Mannequin	**Constructor Skeleton**	**Constructor SkeletonWith Colliders**	**Controllable Character**	**MecanimBlended Character**
Uses "Skinned Mesh Renderer"		x	x	x	x
Has colliders	x		x	x	x
Is affected by physics				x	x
Supports gamepad locomotion				x	x
Real-time mocap input is blended with animations					x

You can **translate, rotate, and scale your avatar** and its motion by parenting it under an empty gameobject upon which you apply the transformations.

All the avatar prefabs (except *Mannequin*) use the Constructor character 3D model provided by Unity. **You can replace the prefabs' 3D model with your own avatar model.** In that case you need to add RUISSkeletonController component to your avatar gameobject. Then link your avatar's joint gameobjects to the RUISSkeletonController by dragging them into their corresponding "Avatar Target Transforms" fields (see below image), as exemplified by the magenta arrow in the below image. Some of the fields can be left to "None", but we recommend that you link all available gameobjects; at least Target Root, Pelvis, Shoulders, Elbows, Hands, Hips, Knees, and Feet.



When using your own 3D model for the avatar, you should make sure that its “Animation Type” is set to “Humanoid” under the “Rig” tab in the 3D model’s “Import Settings”, and that the “Optimize Game Objects” option is disabled. Then you can automatically find the “Avatar Target Transforms” by clicking the “Obtain Targets from Animator” button of RUISSkeletonController component, which is outlined by the yellow rectangle.

If your avatar’s rig has vertices attached to the root Transform, then link that Transform both to “Target Root” and “Pelvis” in RUISSkeletonController. If you intend to use some form of finger tracking (for example when enabling RUISSkeletonController’s “Fist Clench Animation” option with Kinect v2), then make sure that your avatar rig’s finger joint gameobjects all include the substring “finger” or “Finger” in their name. In case you are using a mocap system with finger tracking, you should note that Finger Targets are not assigned in the RUIS avatar prefabs. With *MecanimBlendedCharacter* prefab you can click the “Obtain Targets from Animator” button, but with other prefabs you need to drag and drop the transforms manually. This will be fixed for the next RUIS version. You also need to disable the “Fist Clench Animation” option.

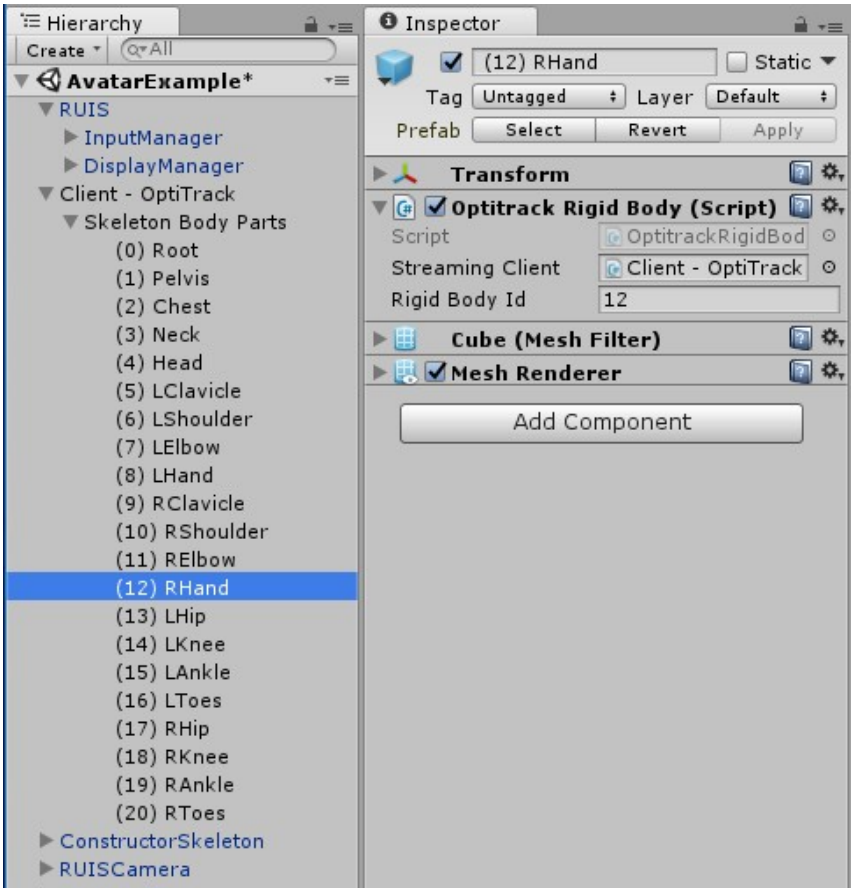
If you are using *MecanimBlendedCharacter* prefab and want to replace its 3D model with your own avatar model, there are some considerations to make: firstly, see this guide to make sure that you maintain all the correct scripts and links between them in the gameobjects of *MecanimBlendedCharacter*. Secondly, you need to add Collider components to the avatar. Thirdly, if you are not using Kinect to animate your character, then you might also need to adjust the Y-coordinate of the *StabilizingCollider* gameobject’s Transform.

Controlling an avatar with a mocap system

This section describes how to use an arbitrary full-body motion capture system to control your avatars. You may skip the whole section if you are using Kinect v1 or v2. Read this section carefully if you are using Perception Neuron, Xsens, OptiTrack, Vicon etc.

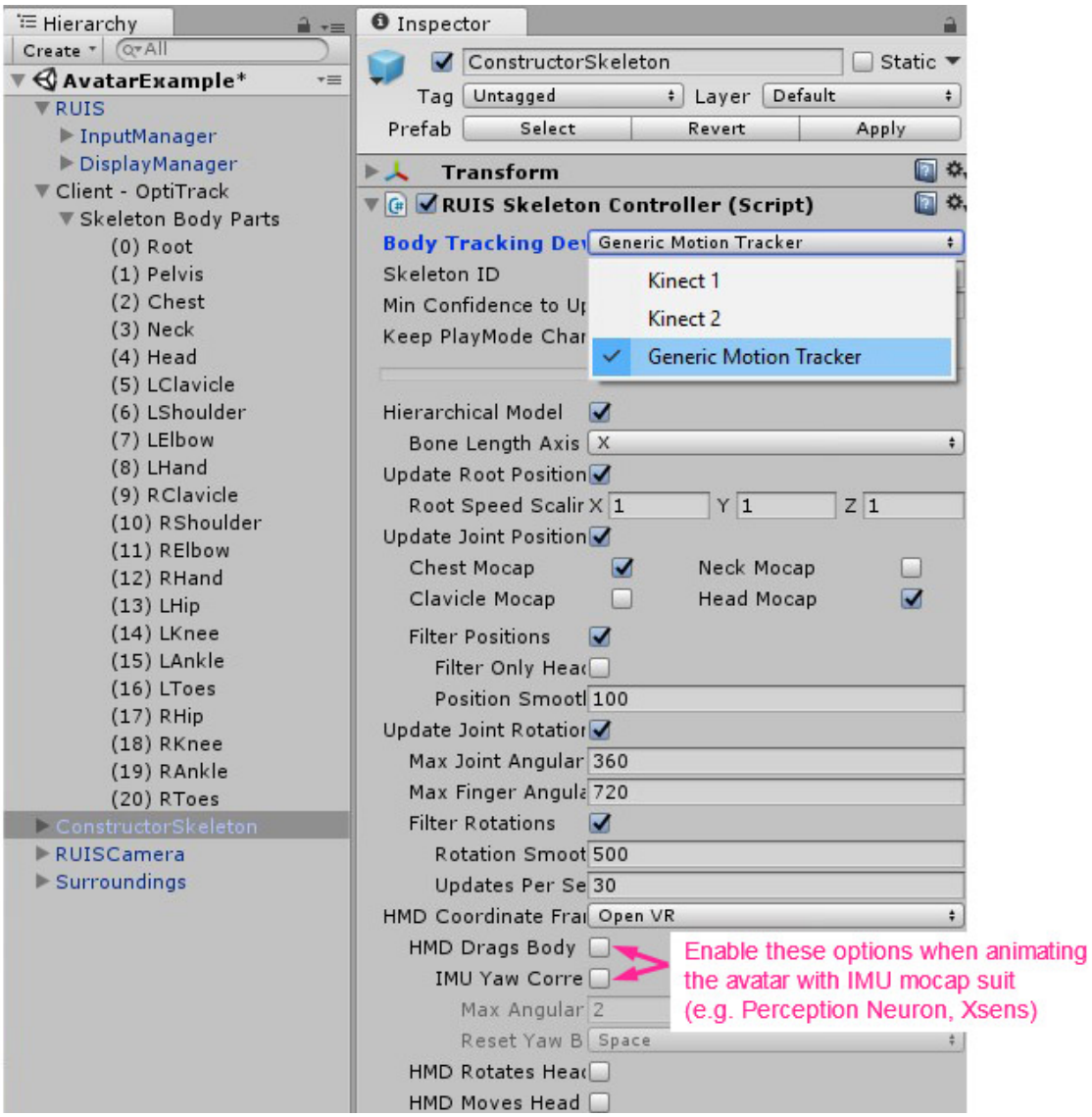
Below cropped screenshot of Unity Editor presents an example of using OptiTrack motion capture system to animate avatars in real-time with RUIS. First, you need to import the Unity plugin of your mocap system into your project. Then create gameobjects for each tracked body joint, whose Transforms will be updated with the world position and rotation of the tracked joints.

In this example the joint pose updating is achieved via the OptitrackRigidBody script, that comes with the MotiveToUnity plugin. In the case of OptiTrack, stream the skeleton as individual rigid body joints instead of streaming the skeleton data format, because the joint position data is not used in the OptiTrack plugin's own example scene, where the whole skeleton object is streamed. When using OptiTrack you should also write a script that finds out the streamed rigid body joint ID numbers and assigns them to all the OptitrackRigidBody components upon playing the scene. Whenever you stream mocap data from another PC to your application, **remember to allow Unity Editor / application executable through firewall** (both UDP and TCP).



Your avatar gameobject has to have the RUISSkeletonController script. At first use the ConstructorSkeleton prefab as a ready made example, and make sure that your scene also includes the *RUIS* prefab that contains the *InputManager* and *DisplayManager*. Please note that there is no “AvatarExample” scene (as seen on the image) within the RUIS project. You can use for example the MinimalScene example as a starting point for your avatar experiments in RUIS.

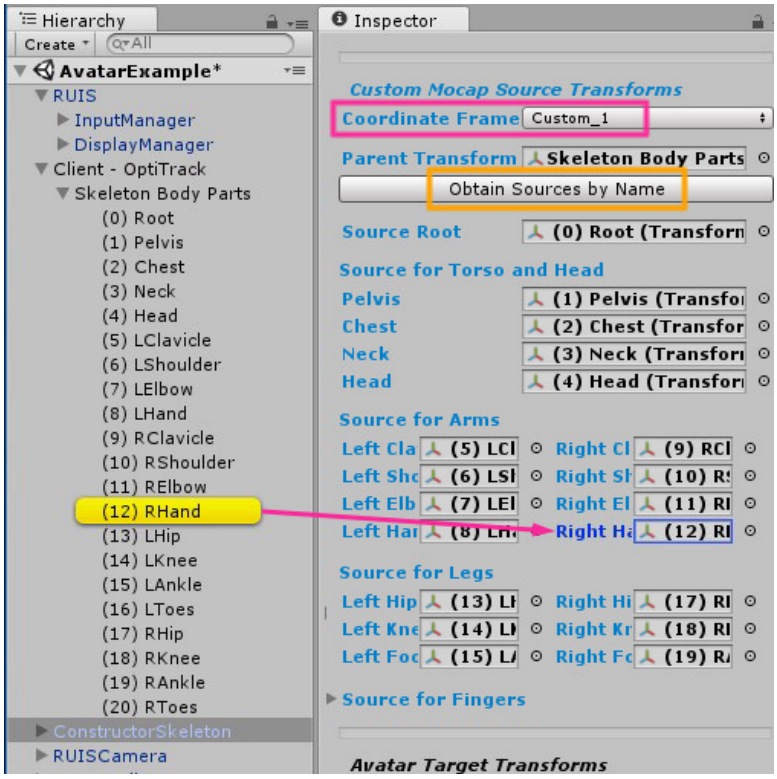
When using other motion capture systems besides Kinect, be sure to set the “Body Tracking Device” property to “Generic Motion Tracker” in RUISSkeletonController. Also disable the “Filter Rotations” option, or adjust the “Updates Per Second” property to match the mocap system update rate if you absolutely need rotation filtering. Note the two settings pointed by the magenta arrows, which should be enabled when using a IMU mocap suit (e.g. Perception Neuron, Xsens) together with a head-mounted display.



Scroll down in the Inspector to see the “Custom Mocap Source Transforms” -section of RUISSkeletonController, which is only visible if “Body Tracking Device” is set to “Generic Motion Tracker”. Give RUISSkeletonController access to the aforementioned gameobjects (that will be updated with the world position and rotation of the mocap tracked joints) by linking their parent to the “Parent Transform” field, and clicking the “Obtain Sources by Name” button (indicated by yellow rectangle). Be sure to double-check that the results of this automatic linking process are correct. Alternatively, you can drag the individual gameobjects into the corresponding “Custom Mocap Source Transforms” fields, as exemplified by the magenta arrow in the below image. Some of the fields can be left to “None”, but we recommend that you link all available mocap tracked joints; at least Source Root, Pelvis, Shoulders, Elbows, Hands, Hips, Knees, and Feet.

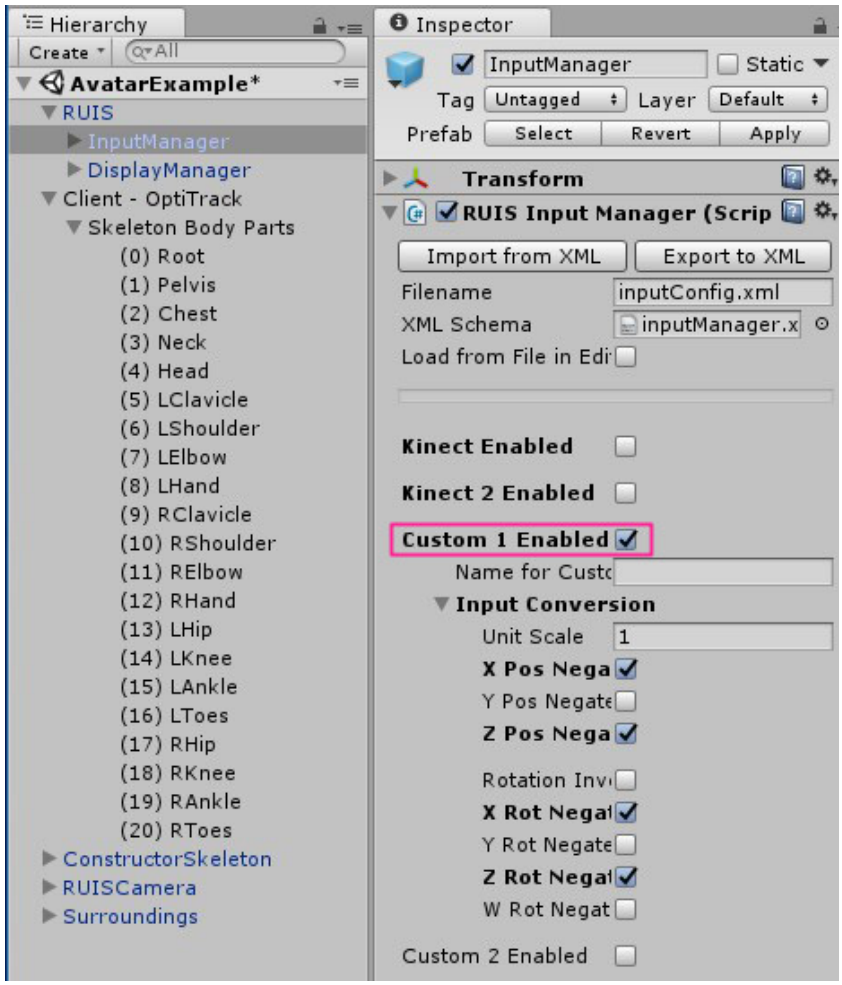
The avatar should make a T-pose in Play Mode, when the mocap tracked joint gameobjects all have an identity world rotation (0, 0, 0) and their world positions correspond to that of a T-pose. Your mocap system plugin might not input joint poses in that format. In that case the joint gameobjects should have child gameobjects with a rotation offset that fulfills the T-pose requirement, when the child gameobjects are linked to the “Custom Mocap Source Transforms” fields instead of their parents. This method can also be used to create joint position offsets.

Note the “Coordinate Frame [and Conversion]” property outlined by the magenta rectangle. That setting associates a specific coordinate frame (“Custom_1”) with the avatar and its mocap system, which allows applying any coordinate alignment and conversions that are required to make the avatar function properly in Unity and together with other devices supported by RUIS. If you are using Perception Neuron, leave this property to “None”.



To access the coordinate conversion settings, you should enable the associated “device” (Custom 1) from the RUISInputManager component, which is located at the *InputManager* gameobject (parented under *RUIS* gameobject). You only need to adjust these settings if the avatar ends up being animated wrong, for example if the joints point at different directions in Unity than in the motion capture software (e.g. Axis Neuron, if you are using Perception Neuron).

The below example shows what “Input Conversion” settings are needed to make avatars work properly with joint data that is streamed from OptiTrack’s old Motive 1.0 software from early 2013. Basically the input conversion is used to make the streamed motion capture joint position and rotation format to conform with Unity’s left-handed coordinate system. You can adjust the “Input Conversion” settings in Play Mode to see their effects in real-time.

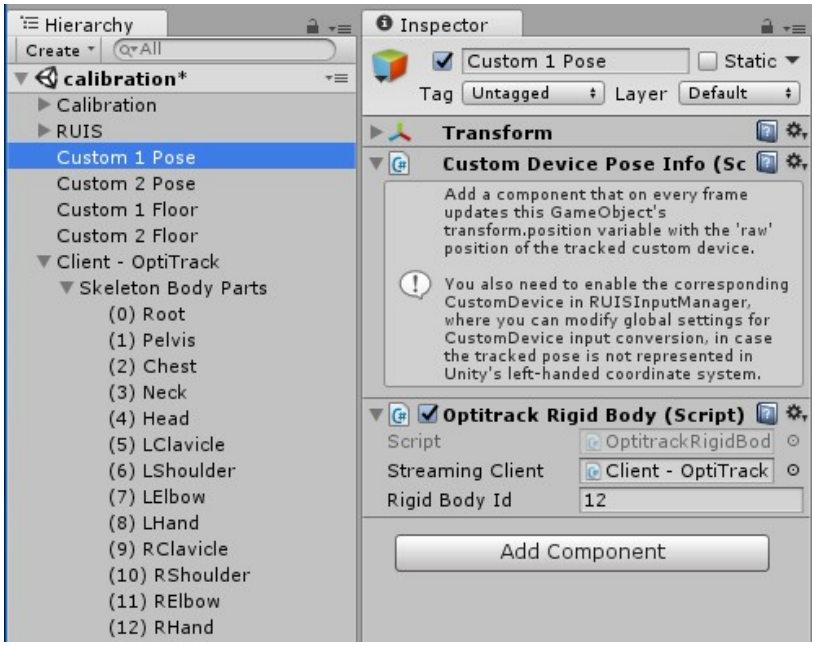


Using a VR headset with a mocap system

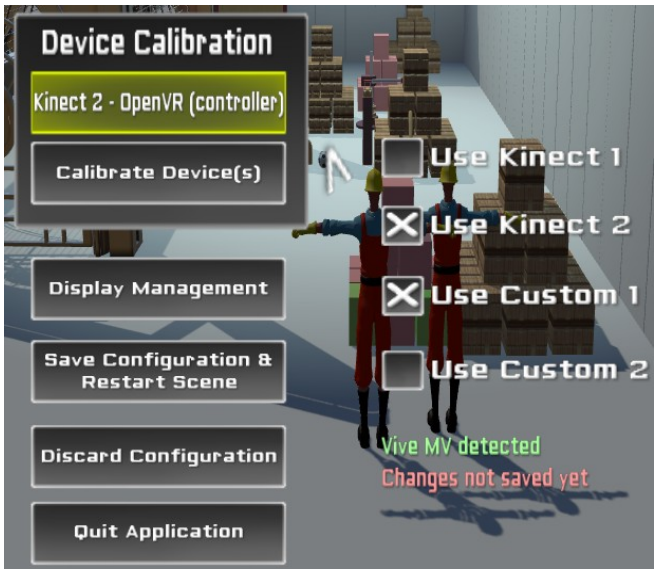
If you want to implement first-person avatars by using a VR headset together with a separate, full-body mocap system, then it is best to utilize the VR headset's tracking system for moving the virtual cameras. That will minimize motion-to-photon latency and allow time-warp optimizations. Consequently, you will then be operating two motion tracking systems simultaneously. If the mocap system is optical (e.g. Kinect, OptiTrack, Vicon), then in most cases you want to align the coordinate frame of the mocap system with the coordinate frame of the VR headset's tracking system. An alternative to this alignment is to enable the "HMD Drags Body" and "IMU Yaw Correct" options in RUISSkeletonController, which only works if the mocap system accurately tracks head yaw rotation, ruling out Kinect v1 and v2. This alternative approach has a side effect of making the avatar "slide" if there is noticeable latency between the mocap and the VR headset tracking.

When using a VR headset, **enable the "Update When Offscreen" option of the avatar's SkinnedMeshRenderer component**, for avoiding mesh blinking in first person view. In RUIS 1.21 this option is disabled by default in all RUIS avatar prefabs (will be fixed for next version).

Aligning coordinate frames happens via a calibration process, which is **not** required when using a IMU mocap suit (e.g. Perception Neuron, Xsens) together with the VR headset. The calibration occurs in calibration.scene that comes with RUIS. When using some other mocap system than Kinect v1 or v2, then you need to edit the scene so that the "Custom 1 Pose" gameobject's world position and rotation will have their values from a joint that will be streamed from your mocap system. If necessary, also edit the "Input Conversion" settings of the RUISInputManager component that is located at *InputManager* gameobject (parented under *RUIS* gameobject of the scene).



You can align the coordinate frames of two input devices by running the calibration.scene in Unity Editor; in this case just make sure that you have the intended two devices selected in the RUISCoordinateCalibration component, which is located at the *Calibration* gameobject of the scene. Alternatively, you can initiate the calibration process via RUIS menu, which can be accessed in Play Mode by pressing the ESC key in any of the RUIS example scenes. Use a mouse to click the green button under the “Device Calibration” label, which opens up a drop-down menu of devices that can be aligned; the available menu items depend on the enabled devices and the detected VR headset.



Once you have selected the device pair from the drop-down menu, click “Calibrate Device(s)” button to start the process for aligning their coordinate frames. See section *“Shared Coordinate Frame for Multiple Tracking Devices”* for details on the process.

Avatar customization and automatic scaling

RUISSkeletonController allows the customization (affecting looks) of arbitrary avatars via relative offsets in translation, rotation, and scaling of individual body segments. These properties can be animated via scripting, which facilitates the creation of interactive effects, for-example power-ups that make the avatar's arms bigger etc.

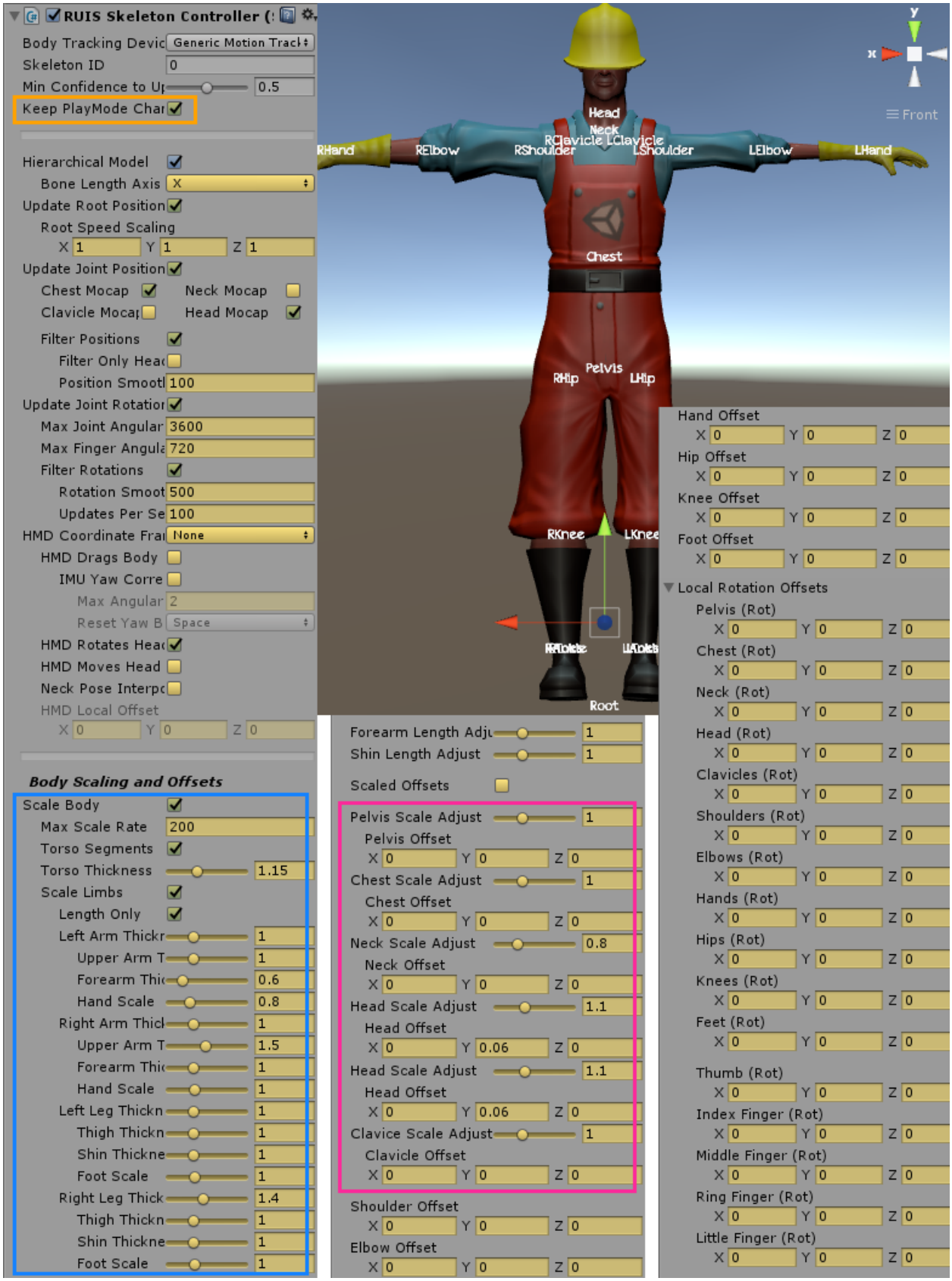
Below image shows the most important settings of the RUISSkeletonController component. If "Keep PlayMode Changes" (yellow rectangle) option is enabled, majority the properties are highlighted with a light yellow background during Play Mode: the values of these highlighted properties will retain their values when exiting Play Mode. This is useful because you need to be in Play Mode to see the effects of the scaling and offset properties, and the default Unity Editor behaviour is to reset all changes made to properties when exiting Play Mode.

The properties within the blue rectangle are the most significant avatar scaling options. By modifying them you can adjust body segment thickness and scaling granularity: "Scale Body" (scales whole avatar, required by all other scaling options), "Torso Segments" (scales individual torso segments), "Scale Limbs" (scales limb segments uniformly to affect their length), and "Length Only" (scales limbs non-uniformly to preserve their thickness). Limbs refer to forearms, upper arms, thighs, and shins. **Enabling "Scale Body" and "Scale Limbs" options matches the avatar's proportions with those of the user** (lengthwise).

The magenta rectangle surrounds the most important "Scale Adjust" and (translation) "Offset" properties that affect the looks of the avatar's torso and head.

Besides affecting looks, **correcting retargeting issues is the secondary function of avatar body segment settings** that affect translation, rotation, and scale. Such issues arise if an avatar and the utilized mocap system use different location and orientation conventions for corresponding joints. For example, mocap systems often have a specific ratio between spine bone (pelvis, chest, neck, head) lengths, which vary little between users of different height. The corresponding ratios can be vastly different for various 3D model rigs that are used as avatars.

If "Scale Body" and "Torso Segments" options are enabled, then the avatar's spine bones will be scaled so that their lengths correspond to the input from the mocap system. This can lead to peculiar body scaling (e.g. neck gets too thin or thick), if the spine bone ratios are different between the avatar and the input from the mocap system. This can be corrected by adjusting the "Scale Adjust" or (translation) "Offset" properties of the affected bone. In similar instances with mismatched bone ratios, the avatar's torso can look peculiar even if "Torso Segments" is disabled, in case any of the individual body segment mocap options (e.g. "Chest Mocap") is enabled under "Update Joint Positions". This can also be addressed by adjusting the "Scale Adjust" or "Offset" properties. Each time you switch to a new avatar or to a different mocap system, you might need to modify the avatar's "Scale Adjust" or "Offset" properties.


RUIS Skeleton Controller (
Body Tracking Devic
Generic Motion Tracl
Skeleton ID
0
Min Confidence to U
0.5
Keep PlayMode Char
Hierarchical Model
Bone Length Axis
X
Update Root Position
Root Speed Scaling
X 1 Y 1 Z 1
Update Joint Position
Chest Mocap
Neck Mocap
Clavicle Mocap
Head Mocap
Filter Positions
Filter Only Hea
Position Smoothl
100
Update Joint Rotatior
Max Joint Angular
3600
Max Finger Angula
720
Filter Rotations
Rotation Smoot
500
Updates Per Se
100
HMD Coordinate Fra
None
HMD Drags Body
IMU Yaw Corre
Max Angular
2
Reset Yaw B
Space
HMD Rotates Hea
HMD Moves Head
Neck Pose Interpc
HMD Local Offset
X 0 Y 0 Z 0
Body Scaling and Offsets
Scale Body
Max Scale Rate
200
Torso Segments
Torso Thickness
1.15
Scale Limbs
Length Only
Left Arm Thickr
1
Upper Arm T
1
Forearm Thi
0.6
Hand Scale
0.8
Right Arm Thicl
1
Upper Arm T
1.5
Forearm Thi
1
Hand Scale
1
Left Leg Thickn
1
Thigh Thickn
1
Shin Thickne
1
Foot Scale
1
Right Leg Thick
1.4
Thigh Thickn
1
Shin Thickne
1
Foot Scale
1
Head
Neck
RClavicle
LClavicle
RHand
RElbow
RShoulder
LShoulder
LElbow
LHand
Chest
Pelvis
RHip
LHip
RKnee
LKnee
Root
Front
Forearm Length Adju
1
Shin Length Adjust
1
Scaled Offsets
Pelvis Scale Adjust
1
Pelvis Offset
X 0 Y 0 Z 0
Chest Scale Adjust
1
Chest Offset
X 0 Y 0 Z 0
Neck Scale Adjust
0.8
Neck Offset
X 0 Y 0 Z 0
Head Scale Adjust
1.1
Head Offset
X 0 Y 0.06 Z 0
Head Scale Adjust
1.1
Head Offset
X 0 Y 0.06 Z 0
Clavice Scale Adjust
1
Clavicle Offset
X 0 Y 0 Z 0
Shoulder Offset
X 0 Y 0 Z 0
Elbow Offset
X 0 Y 0 Z 0
Hand Offset
X 0 Y 0 Z 0
Hip Offset
X 0 Y 0 Z 0
Knee Offset
X 0 Y 0 Z 0
Foot Offset
X 0 Y 0 Z 0
Local Rotation Offsets
Pelvis (Rot)
X 0 Y 0 Z 0
Chest (Rot)
X 0 Y 0 Z 0
Neck (Rot)
X 0 Y 0 Z 0
Head (Rot)
X 0 Y 0 Z 0
Clavicles (Rot)
X 0 Y 0 Z 0
Shoulders (Rot)
X 0 Y 0 Z 0
Elbows (Rot)
X 0 Y 0 Z 0
Hands (Rot)
X 0 Y 0 Z 0
Hips (Rot)
X 0 Y 0 Z 0
Knees (Rot)
X 0 Y 0 Z 0
Feet (Rot)
X 0 Y 0 Z 0
Thumb (Rot)
X 0 Y 0 Z 0
Index Finger (Rot)
X 0 Y 0 Z 0
Middle Finger (Rot)
X 0 Y 0 Z 0
Ring Finger (Rot)
X 0 Y 0 Z 0
Little Finger (Rot)
X 0 Y 0 Z 0

The “Forearm Length Adjust” and “Shin Length Adjust” can be used to shorten or lengthen forearms and shins. This is particularly useful when the relative location of hand or foot pivot differs between the mocap system and the avatar’s rig. For example, the default Constructor avatar (the guy in the image with red overalls) has its foot pivot slightly above the ankle. You can verify this via selecting the corresponding gameobject by double-clicking the “Left Foot” field in “Avatar Target Transforms” section, and by having the “Pivot” option of “Transform Gizmo” chosen in Unity Toolbar. If you animate the Constructor avatar with a mocap system, whose input feet positions are closer to the user’s heel bone than ankle, then the avatar’s virtual feet will end up below the virtual floor plane, unless you set the “Shin Length Adjust” value to somewhere below 1.

The properties in “Local Rotation Offsets” rarely require any other values than the default identity rotation (0, 0, 0). The most common use case is adjusting “Feet (Rot)” on occasions where the avatar rig and the utilized mocap system use different rotation angles for the same feet pose. For example, consider a case where the chosen mocap system gives a 90-degree angle between shin and foot if the user is making a T-pose, while utilizing an avatar whose foot joint has a clearly oblique angle in the default T-pose (when not animated by RUIS).

Please note that the effects of RUISSkeletonController’s scaling and offset properties can be seen during Play Mode, when the “Body Tracking Device” property is set to Kinect (1 or 2), even if no Kinect is connected or enabled in RUISInputManager component (translation offsets have no effect in this case). However, the exact appearance of the avatar can only be verified when the selected “Body Tracking Device” is connected and enabled.

In the case of “Generic Motion Tracker”, no mocap system is needed if the “Custom Mocap Source Transforms” properties are linked to gameobjects, whose world positions form a T-pose and each has an identity world rotation. An example of this is seen in the above image, where the linked gameobjects each have a TextMesh component (e.g. Head, Neck, Chest, …). This specific pose is not really required, it just happens to be the easiest one to setup that has congruent positions and rotations.

Enabling the “Length Only” option for scaling works only with certain avatar rigs, where there exists a single local axis (X, Y, or Z), which points the bone length direction consistently among all the limb joint Transforms (shoulders, elbows, hips, and knees). Set “Bone Length Axis” to that axis. You can discover the correct axis by selecting individual limb joint Transforms of the rig while having the “Pivot” option of “Transform Gizmo” and “Move Tool” chosen in Unity Toolbar. This makes the “Move Tool” to indicate localScale axes of the selected joint Transform. The correct axis is the one that is aligned with the bone length direction. In some avatar rigs that alignment is not consistent among all the limb joints (e.g. many Mixamo rigs), in which case you should disable “Length Only”. In the next RUIS version you will be able to set “Bone Length Axis” separately for arms and legs, covering Mixamo rigs and others with inconsistent bone length axis directions.

The default “Max Scale Rate” of 0.5 (units per second) is too high for Kinect and other mocap systems that continuously estimate user’s bone lengths. This default was chosen because “Max Scale Rate” also limits how quickly any changes to “Thickness” and “Scale Adjust” properties are manifested in the avatar, and smaller values would have made the changes less apparent. In a future version these properties will not be limited by “Max Scale Rate”, and the default will be set to 0.01.

Finally, remember that **tooltips provide additional information about the properties** of RUISSkeletonController. Tooltips appear when hovering the mouse cursor over the property name in Inspector. Tooltips do not show up during Play Mode.

Kinect controlled full-body avatars

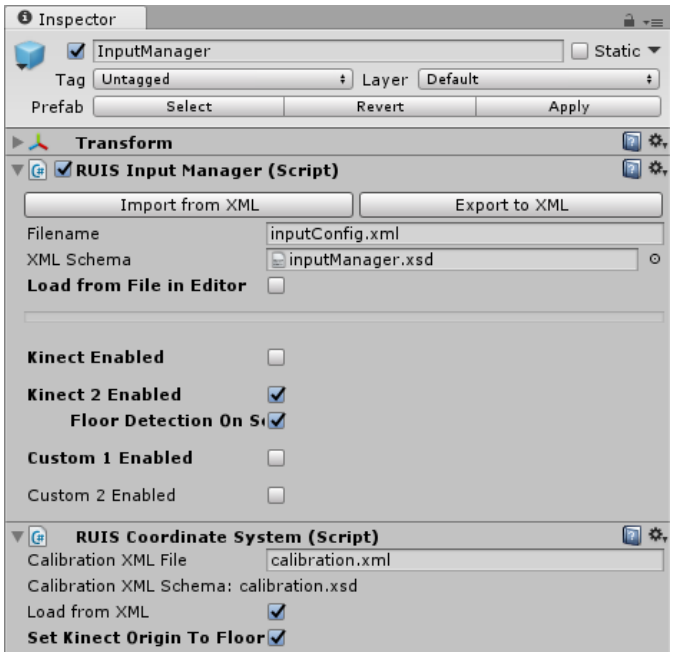
If you have successfully installed the drivers for Kinect v1 or v2 and want to use it in your RUIS for Unity scene, make sure to tick the “Kinect Enabled” / “Kinect 2 Enabled” option in *InputManager* gameobject (parented under *RUIS* gameobject).

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 Kinect v2 can also track you while you are sitting on a chair.

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.



Kinect avatars and a head-mounted display

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.

Avatar with head-mounted display but without mocap

Without a mocap system like Kinect, using RUIS avatars have only a limited use. Best case scenario would be to use an external inverse kinematics library to infer the avatar pose from the VR headset and controller poses, and use that as a custom mocap system input. This would make sense if you could calibrate or manually input the users' body proportions, and the inverse kinematics library would not support scaling of avatar's individual body segments.

The avatar position follows the head-mounted display's location if "HMD Drags Body" is enabled in RUISSkeletonController. You should also enable the "HMD Rotates Head" option. Make sure that the "Master Coordinate System Sensor" is set to OpenVR or UnityXR in the *InputManager* gameobject that is parented under *RUIS* gameobject. If you use mouse and keyboard for controlling the *MecanimBlendedCharacter* locomotion, see *"Avatar Controls"* section below.

In current RUIS version you also need to set "Body Tracking Device" property to "Generic Motion Tracker" in RUISSkeletonController, and "Pelvis Source" to *RUIS* or some other gameobject whose position and rotation is constant (other "Custom Mocap Source Transforms" can be left to "None"). Finally, adjust RUISSkeletonController's "Pelvis Offset" during Play Mode until the avatar head position is co-located with the virtual camera.

Avatars with Locomotion, Physics, and Animation

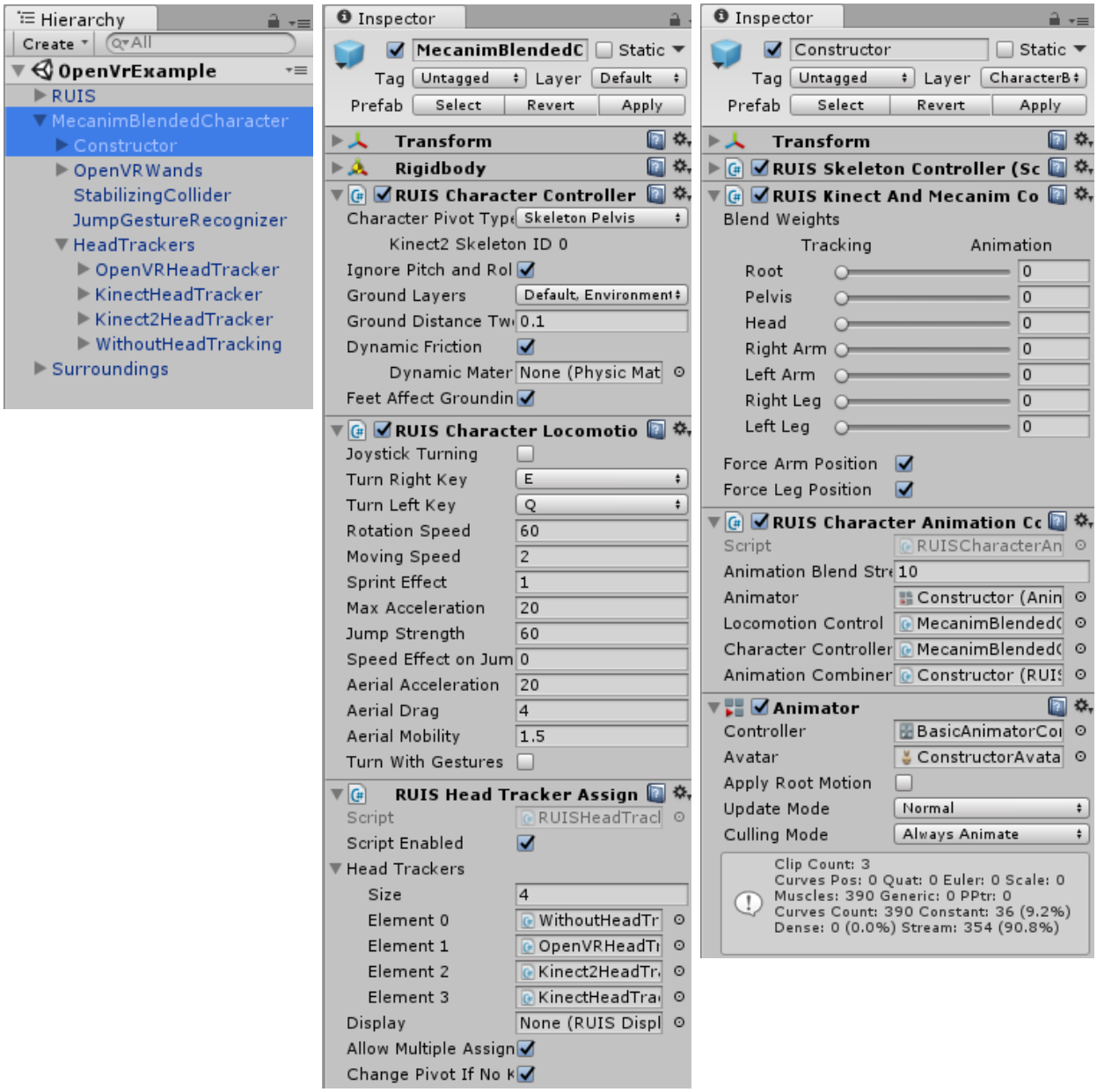
RUIS features *MecanimBlendedCharacter* prefab, which is a feature-rich character controller for applications with first- or third-person view. The prefab contains a human 3D model that is animated with Kinect v1, Kinect v2, or a "Generic Motion Tracker" (any mocap system). You can substitute the default 3D model with your own.

MecanimBlendedCharacter prefab comes with so called "full locomotion" or analog stick walking. Whenever the player is moving the avatar either with a keyboard, gamepad, or OpenVR (e.g. Vive) controller, the avatar's leg poses will blend from mocap tracking input into Mecanim walking animation. You can **use your own Mecanim animation graph** and utilize RUIS features to write scripts that **blend Mecanim animation with mocap tracking in real-time** (currently this blending does not work with "Length Only" limb scaling). See KinectTwoPlayers or OpenVrExample at \RUISunity\Assets\RUIS\Examples\ and modify the *MecanimBlendedCharacter* gameobjects to get started.

By default the *MecanimBlendedCharacter* is affected by gravity, so you should place it on a static Collider, otherwise it will keep falling downwards in Play Mode. When the user is in control of the *MecanimBlendedCharacter* via a mocap system, they can climb on rigidbodies and push or hit dynamic rigid bodies. *MecanimBlendedCharacter* contains *StabilizingCollider* gameobject, which is attached to the avatar's pivot (either pelvis or head). It serves two functions when the avatar is affected by gravity: 1. If the mocap tracked user jumps, then the *StabilizingCollider*'s height increases accordingly (albeit in a highly filtered manner), so that the jump is reflected in avatar's vertical position. 2. In first person VR applications, *StabilizingCollider* should be the only contact point to static floor Colliders, so that the VR camera is as stable as possible. If the mocap tracked feet interact with a static floor Collider by pushing the avatar upwards, this propagates the feet tracking jitter into *MecanimBlendedCharacter*'s VR camera.

MecanimBlendedCharacter prefab's scripts automatically select the most suitable head-mounted display tracking between OpenVR tracking (e.g. Lighthouse base stations), Kinect 1, or Kinect 2. 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 1.21. 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.

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*.



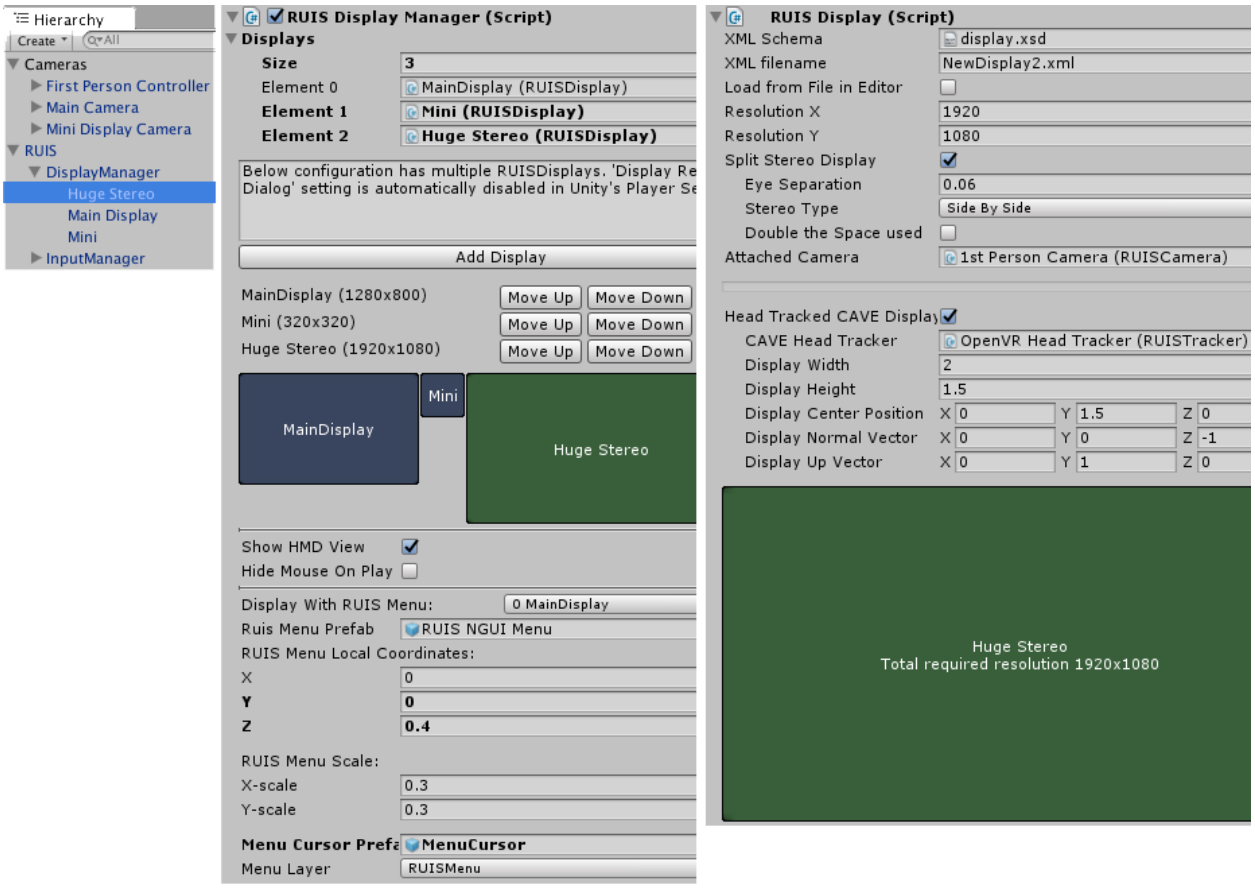
The *MecanimBlendedCharacter* has *OpenVRWands* child gameobject, which contains 3D Wands that can grab and manipulate objects. You can delete these Wands or modify them. The *JumpGestureRecognizer* child gameobject contains scripts that make the character jump if Kinect is enabled and detects the player jumping. The recognition is far from perfect however, and the functionality of *JumpGestureRecognizer* is disabled by default.

A more simplified version of *MecanimBlendedCharacter* is *ControllableCharacter* prefab, which can be animated by mocap systems and has the same features, but does not blend Mecanim animation.

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 property 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.

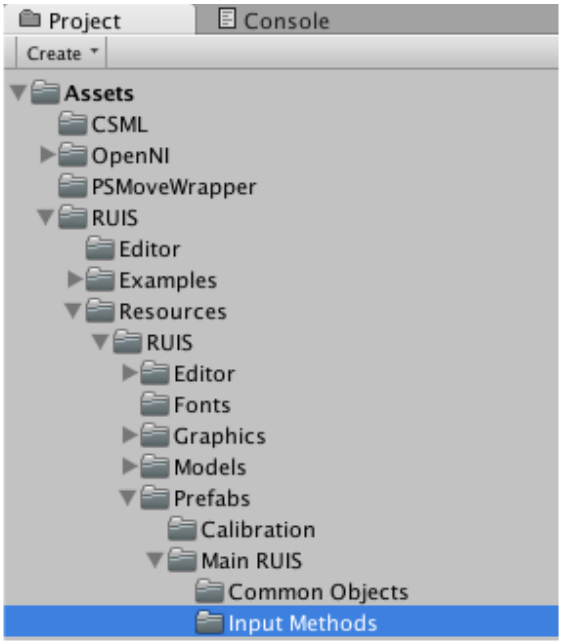
RUISCamera component has only two visible properties in the Unity Inspector: Horizontal and Vertical FOV. They only have effect if the RUISCamera is acting as a non-head-mounted display camera, rendering on a RUISDisplay that has the "Head Tracked CAVE Display" option disabled.

3D UI Prefabs for Selection and Manipulation

Note: Introduced in 2016, VRTK now offers far better 3D UI interaction building blocks than come with RUIS. I have not yet tested how compatible they are with RUIS.

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**, 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.

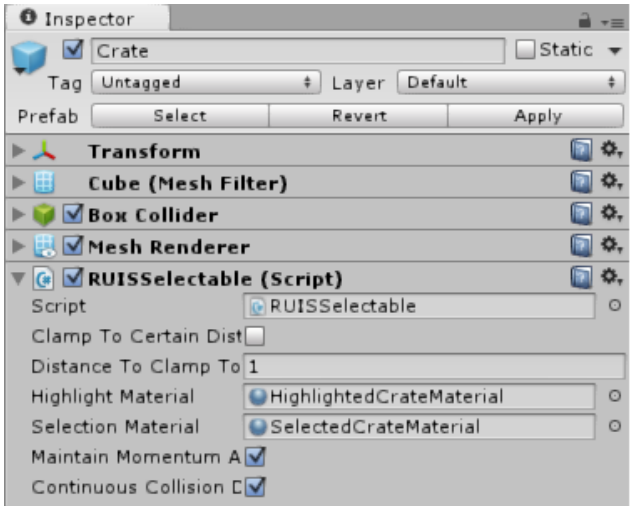


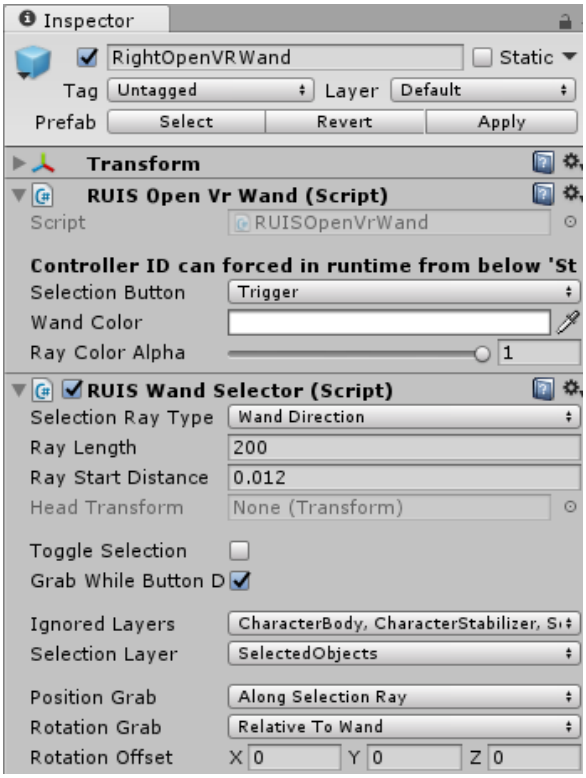
If you do not have Kinect or OpenVR controllers, then use MouseWand prefab that relies on 2D mouse for object manipulation purposes. When your scene is playing, the above mentioned Wands are used to manipulate gameobjects that have a **RUISSelectable** script, Mesh Renderer, Rigidbody, and Collider components. 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 RUISOpenVrWand component, "Selection Button" property specifies the button that is used for selecting gameobjects with RUISSelectable script. 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.

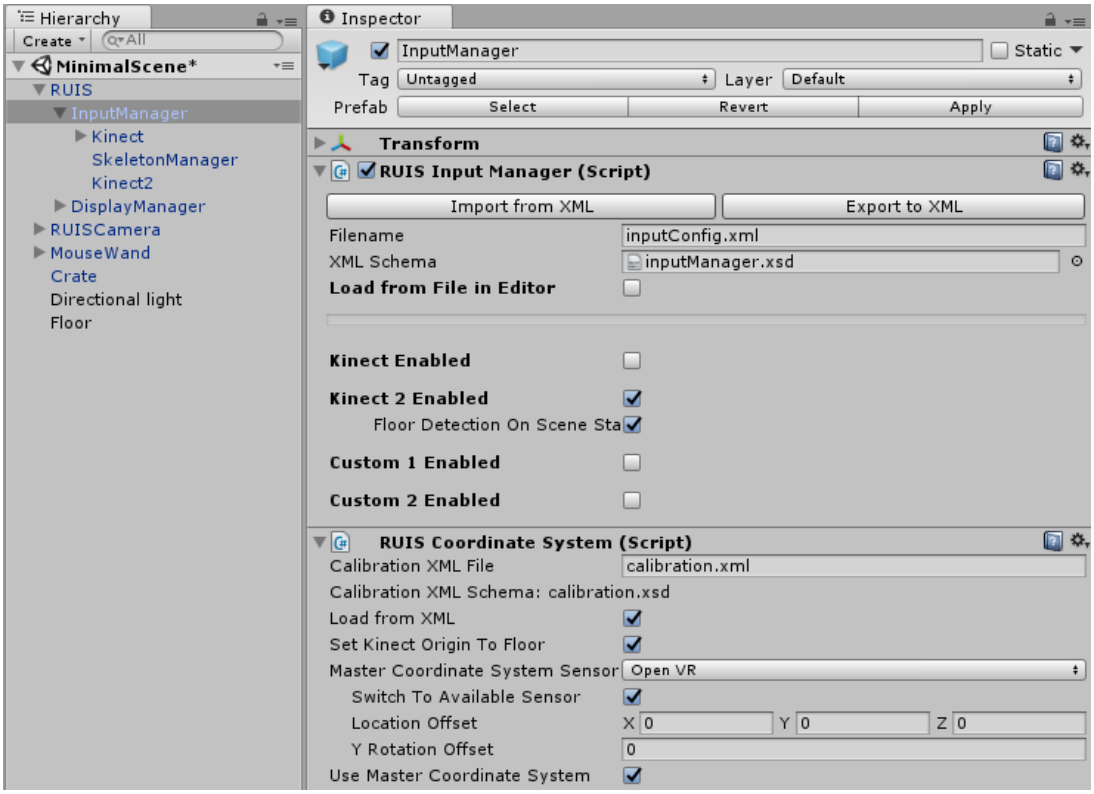




Shared Coordinate Frame for Multiple Tracking Devices

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 below 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 Custom_1 (e.g. OptiTrack). Then you would need to calibrate two device pairs: Kinect_2–OpenVR and Custom_1–OpenVR. The following section will tell you how to do that.

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 when attempting to make the OpenVR coordinate system to conform the other device's coordinate system. RUISTracker component acts differently from its normal behaviour, if its "Position Tracker" property is set to "Open VR": instead of applying OpenVR position input to the gameobject with the component, it only applies a constant translation, rotation, and scale that approximates the transformation from OpenVR coordinate frame to "Master Coordinate System Sensor" frame.



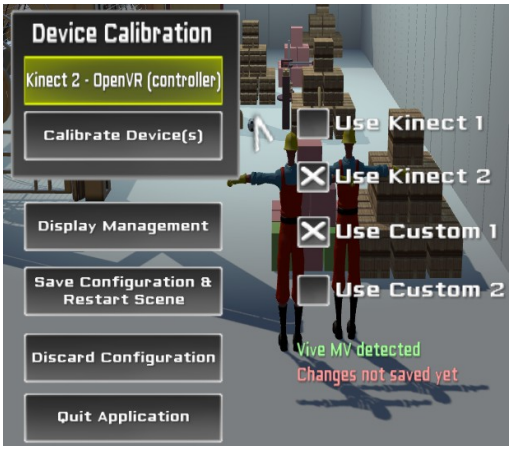
Calibrating two different motion trackers to use the same coordinate system

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. **RUIS menu can only be interacted with by using a mouse.**

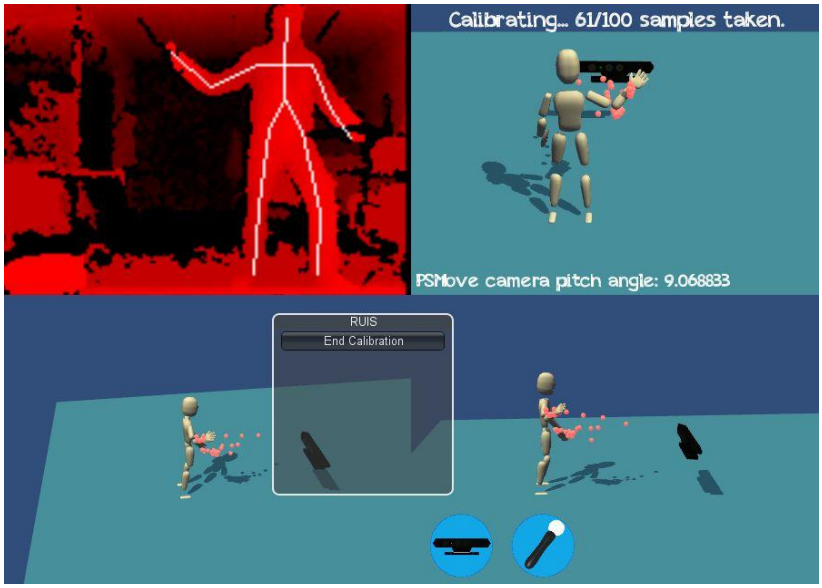
All head-mounted displays can be calibrated by choosing the "[OtherDeviceName] - OpenVR (HMD)" option in the Device Calibration drop-down menu. This option has erroneously the same name even if you are calibrating Oculus Rift without using SteamVR, and the results will be saved to "OpenVR-[OtherDeviceName]" 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.

When using Kinect or other tracking system with Vive, it is important to keep in mind that **Vive's Lighthouse-tracking coordinate origin can drift**, based on the stability of your power source. Voltage fluctuations in your mains power can affect the angular velocity of the basestation motor, potentially leading into several inches of displacement in the coordinate origin. All that can be avoided by having stable mains power or using an "online" or "double-conversion" uninterruptible power supply (UPS).



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, or OpenVR controller.

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.

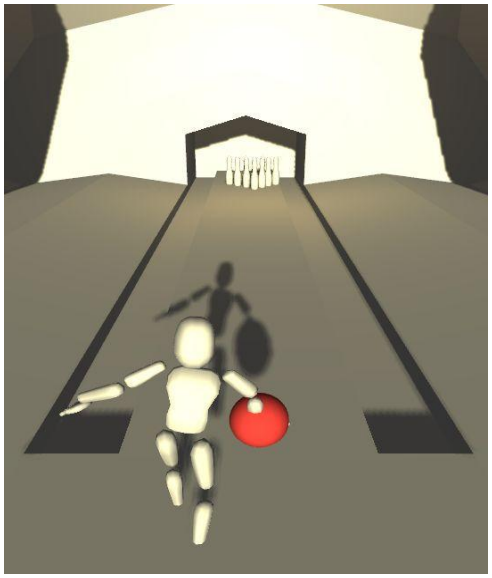
KinectTwoPlayers

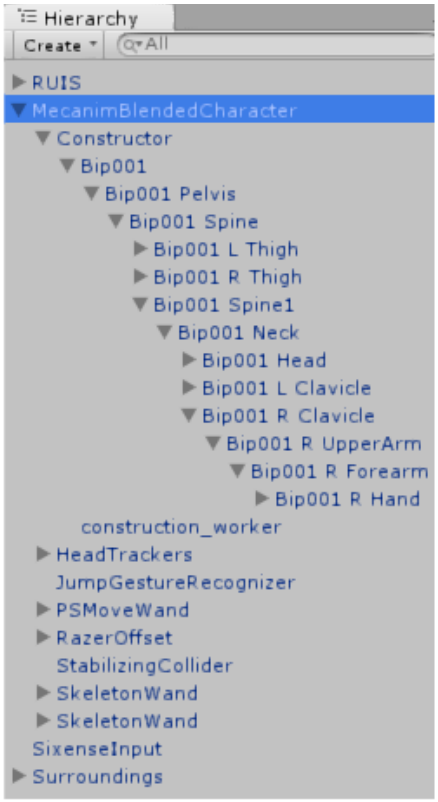
This example demonstrates how you can create a multiuser Kinect application in RUIS. The Kinect avatars are equipped with Collider components so that they can push objects around. Also note how the Kinect-controlled SkeletonWands can be used for object manipulation.

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). Note how *Mannequin*'s body parts are all parented in a flat fashion and that the "Hierarchical Model" option is unchecked in its RUISSkeletonController script, as opposed to the body parts of *MecanimBlendedCharacter* gameobject in OpenVrExample scenes. The Transform hierarchy of both avatar prefabs are depicted in the below illustration.

The *Mannequin* prefab and its flat, one-level deep bone setup is so rare, that not all the avatar customization options of RUISSkeletonController are functional when "Hierarchical Model" is disabled. Also note that the *Mannequin* prefab does not work well as an avatar viewed from first-person. We recommend that you utilize other prefabs (*ConstructorSkeleton, ConstructorSkeletonWithColliders, ControllableCharacter, MecanimBlendedCharacter*), when working with avatars.





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 (with “Target Eye” set to “None (Main Display)” if VR is enabled) 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

	OpenVR controller
Move forward / backward	Left controller trackpad
Strafe left / right	Left controller trackpad
Turn left / right	Right controller trackpad
Jump	Right Controller Menu Button
Run	Left Controller Menu Button
Grab object	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 2 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)

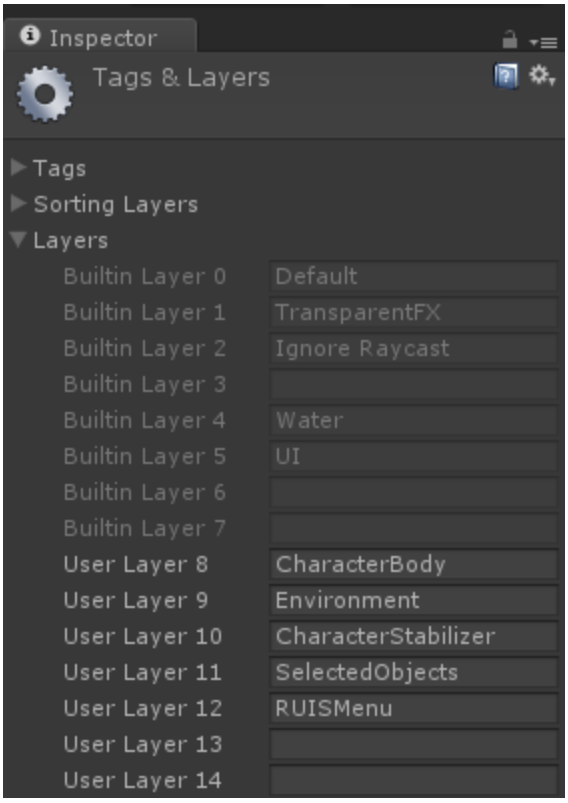
Microsoft released Kinect for Windows v1 and Kinect v1 SDK, but they are not compatible with OpenNI. The kinect-mssdk-openni-bridge is an experimental module that connects Kinect SDK to OpenNI and allows people with Kinect for Windows to use OpenNI applications. This bridge _might_ get RUIS to work with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

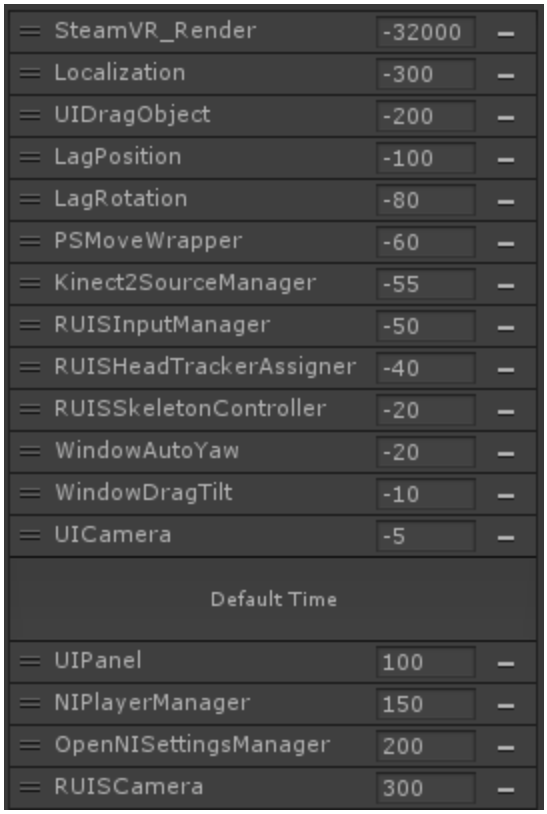
Firewall settings for mocap systems

It is often necessary to adjust your computer's firewall settings, when you use RUIS together with a mocap system Unity plugin that streams mocap tracking data from the mocap computer. Check that the mocap and application (Unity) computers are connected to the same network, and that they both can successfully ping each other. You might need to allow both outgoing and incoming TCP and UDP data.

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 for RUIS 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).