MiVRy – 3D Gesture Recognition AI

V2.11

©2024 MARUI-PlugIn (inc.)

MiVRy - 3D GESTURE RECOGNITION PLUG-IN FOR UNREAL

Copyright (c) 2024 MARUI-PlugIn (inc.)

[IMPORTANT!] This plug-in is free to use. However, the “free” license use is limited to 100 gesture recognitions (or 100 seconds of continuous gesture recognition) per session. To unlock unlimited gesture recognition, please purchase a license at https://www.marui-plugin.com/mivry/

[IMPORTANT!] This license is for the use of the MiVRy Gesture Recognition plug-in (.dll and .so files as well as the source code) and does not include the use of the asset files used in the sample levels.
Please see the license statement at the end of this document.

If you want to use this plug-in in a commercial application, please contact us at support@marui-plugin.com for a commercial license.

Check out our YouTube channel for tutorials, demos, and news updates:
https://www.youtube.com/playlist?list=PLYt4XosVlCmWtmDmx1lS8OVGU70tmQOfv

Content:

1: What is 3D Gesture Recognition?

Making good user interaction for VR is hard. The number of buttons often isn't enough and memorizing button combinations is challenging for users.
Gestures are a great solution! Allow your users to wave their 3D controllers like a magic wand and have wonderful things happen. Draw an arrow to shoot a magic missile, make a spiral to summon a hurricane, shake your controller to reload your gun, or just swipe left and right to "undo" or "redo" previous operations.
MARUI has many years of experience of creating VR/AR/XR user interfaces for 3D design software. Now YOU can use its powerful gesture recognition module in Unreal.
This is a highly advanced artificial intelligence that can learn to understand your 3D controller motions.
The gestures can be both direction specific ("swipe left" vs. "swipe right") or direction independent ("draw an arrow facing in any direction") - either way, you will receive the direction, position, and scale at which the user performed the gesture.
Draw a large 3d cube and there it will appear, with the appropriate scale and orientation.
Both one-handed and two-handed gestures are supported and you can even build combos of sequential gestures.

Key features:
- Real 3D gestures - like waving a magic wand in all three dimensions
- Support for multi-part gesture combinations such as two-handed gestures or sequential combinations of gestures
- Record your own gestures - simple and straightforward
- Easy to use C++ classes and convenient wrapper Actors
- Can have multiple sets of gestures simultaneously (for example: different sets of gestures for different buttons)
- High recognition fidelity
- Outputs the position, scale, and orientation at which the gesture was performed
- High performance (back-end written in optimized C/C++)
- Includes a Unreal sample levels that illustrate how to use the plug-in
- Save gestures to file for later loading
- Support for Windows, Android-based devices (Oculus Quest, Smartphones, …), UWP devices (Hololens), MacOS, and Linux

2: Quick Start Guide:

This guide explains the simplest way to use MiVRy in your Unreal project. Necessarily, a lot of features are not fully explained here. Please read the rest of this document for more details and additional explanations of features.
A video tutorial is available on YouTube: https://youtu.be/qouf9MgXREw
A sample project is available for download at:
https://www.marui-plugin.com/download/mivry/MiVRy_UE5.3_Sample.zip

2.1: Use the Gesture Manager to record your gestures:

You can download the GestureManager from https://www.marui-plugin.com/documentation-mivry-unreal/#gesturemanager

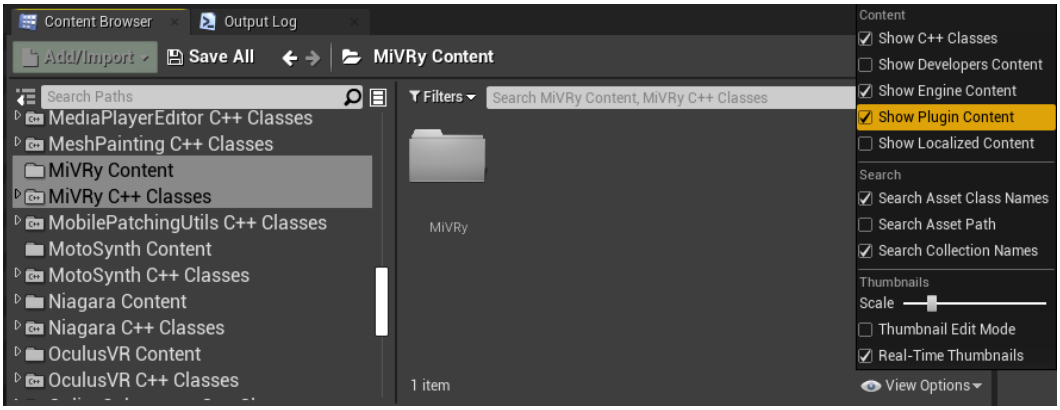
A video tutorial on how to use the GestureManager is available on YouTube:
https://www.youtube.com/watch?v=kcoobj7V2-o

When you are happy with your recorded gestures, save the recorded gestures to a Gesture Database File (.dat file).

2.2: Import the plug-in into your project:

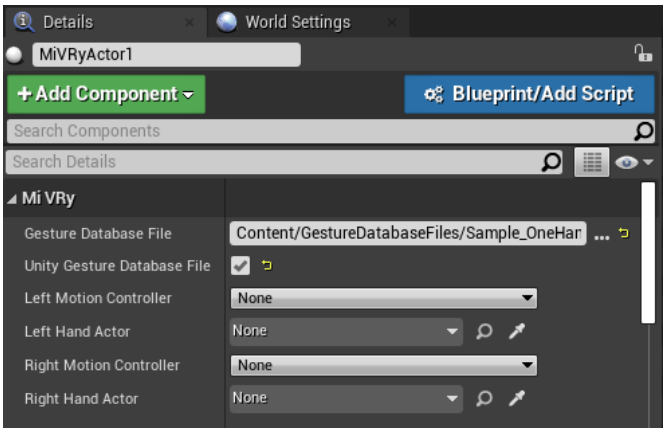
Copy the *Plugins/MiVRy/* folder into your own project's *Plugins/* folder. (If your project does not yet have a *Plugins/* folder just create a now folder named "Plugins").

Open your project in Unreal Editor and check that the MiVRy plugin files are now available in the Content Browser. (You may have to check the "Show Plugin Content" checkbox in the View Options).



2.3: Add MiVRyActor to your level:

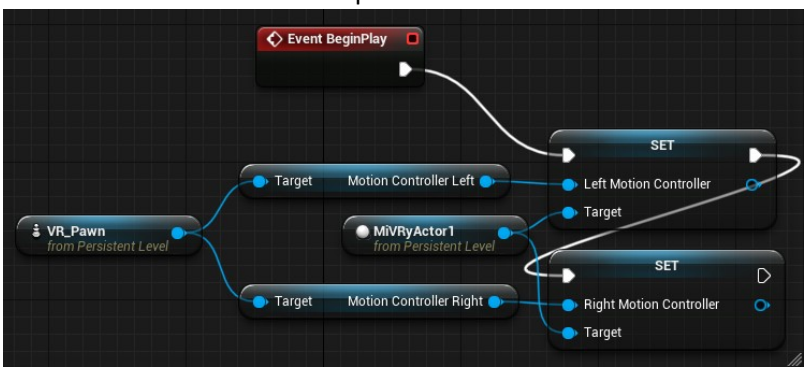
From the MiVRy C++ Classes, select the MiVRyActor and add it to your level.
In the Details panel, set the properties of the MiVRyActor to comply with your project.



As “Gesture Database File”, set the path to the .DAT file created with the GestureManager. This can be an absolute path (such as C:\MyProject\...) or a relative path in your project (such as Content/MyFiles/…).

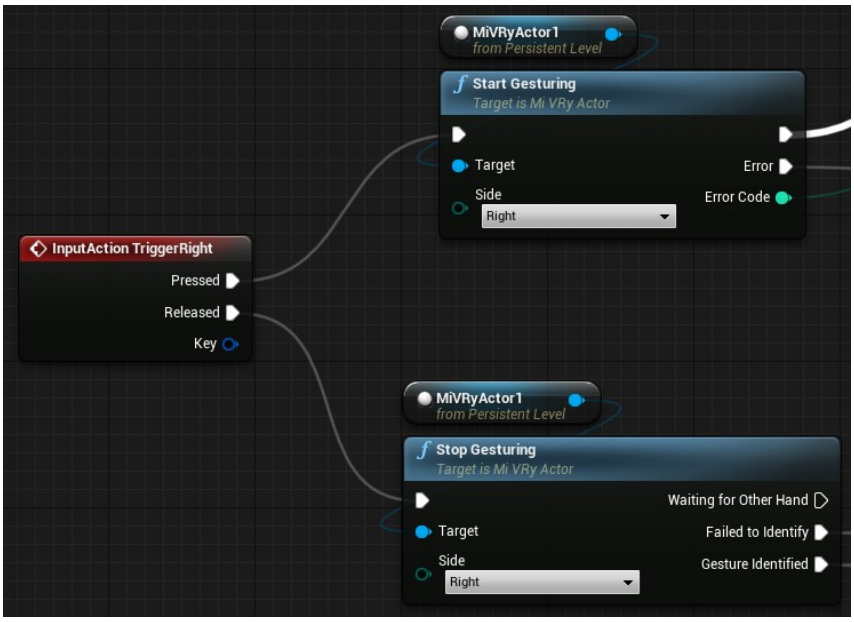
Since the GestureManager uses the Unity coordinate system, check the “Unity Gesture Database File” option.

The “Left Motion Controller”, “Left Hand Actor”, “Right Motion Controller” and “Right Hand Actor” are optional ways to decide how the MiVRy actor will track the motion of your hands while gesturing.
You can either set the “Hand Actor” to an actor in your level (can be an empty actor which is attached to your motion controller, or you can set the “Motion Controller” to a Motion Controller Component whose position represents the hand.
You can also set these in Blueprint or via C++ at run-time:

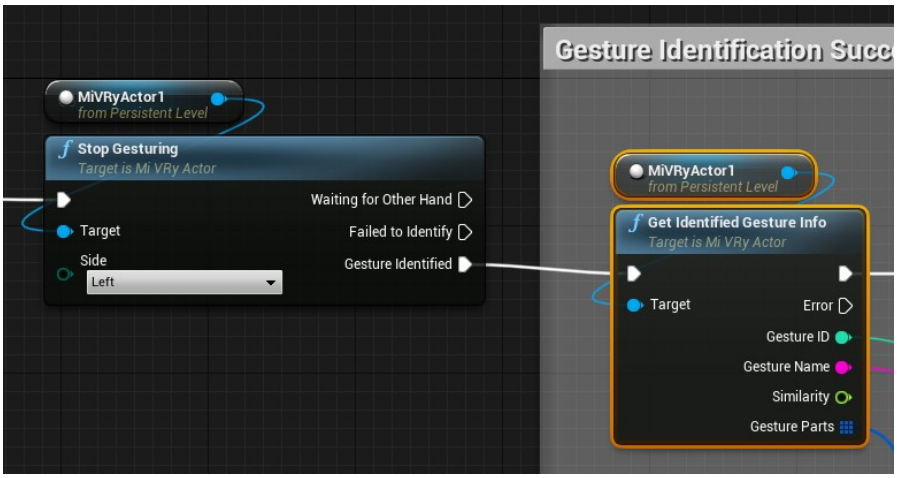


You can set *either* “Hand Actor” or “Motion Controller” –or you can set neither of them. If you set neither, then Unreal’s `XRSystem->GetMotionControllerData` is used to acquire motion controller positional data. However, this does not work with all Unreal VR Plugins. Currently, only the OpenXR plug-in works with this option. So if you want to have your controllers tracked without setting either “Hand Actor” or “Motion Controller” parameters, please deactivate other VR plug-ins like Oculus-VR or Steam-VR and enable the OpenXR plug-in in the Unreal Editor “Edit” -> “Plugins” menu.

Then, in your level Blueprint, add an input action to trigger the stand and end of the gesture motion for each hand:



At the “Stop Gesturing” function, use the “Gesture Identified” result out-pin to further decide how your game will react to the gesture. You can query details on the identified gesture with the “Get Identified Gesture Info” function.



The “Gesture Parts” array will give you information on the individual motions that comprised the gesture. If the gesture was a one-handed single-motion gesture, this array will only have one element.

Now when you run your project, press the input button that you connected to start / stop the gesture and perform a gesture, MiVRy will identify your gesture and provide you with helpful information such as the position, direction, and scale of the gesture motion.

3: Package Overview:

(1) Plug-in library files (binaries):

In the Plugins/MiVRy/Source/ Thirdparty/ folder you can find the plugin library files.
The Plugins/MiVRy/Source/MiVRy/MiVRy.Build.cs script informs Unreal which file to use for which platform.

(2) Plug-in wrapper classes (C++)

MiVRy provides three different C++ Actor classes to use the plug-in library. You ever only need *one* of the three, depending on your requirements and development goals.
- GestureRecognitionActor : Low-level interface for using one-handed (one-part) gestures.
- GestureCombinationsActor: Low-level interface for using two-handed or multi-part gestures.
- MivryActor : High-level actor for simple use of pre-recorded gestures without coding.

To use MiVRy in your own project, decide wether you only need gesture recognition based on a pre-recorded file (then MiVRyActor is the easiest), or if you need more control for example to create new gestures at run-time. If so, use either the GestureRecognitionActor for one-handed single-motion gestures, or use GestureCombinationsActor for two-handed or multi-part gestures. The *GestureRecognition.h* and *GestureCombinations.h* files are pure C++ headers to access the binary DLL libraries.

(3) Samples:

The Content/ folder offers several Unreal levels that illustrate the various use cases of MiVRy.
- Sample_Simple : Unity sample scene and script on how to use the MivryActor.
- Sample_OneHanded : Unity sample scene and script for one-handed gestures.
- Sample_TwoHanded : Unity sample scene and script for two-handed gestures.

[IMPORTANT] The samples include several assets (prefabs, textures, …). The MiVRy license does NOT include these assets! They are only included as part of the samples. You may NOT use any of the items in the "*Content/*" folder in your project.

4: License and Activation

MiVRy is free to use for commercial, personal, and academic use.
However, the free version of MiVRy has certain limitations.
The free version of MiVRy can only be used to identify 100 gestures per session (meaning every time you run the app). When using continuous gesture identification, it can only be used for a total of 100 seconds of identifying gestures.
To unlock unlimited gesture recognition, you must purchase a license at:
https://www.marui-plugin.com/mivry/

The license key will be sent to you automatically and immediately after purchase.
If the license email does not arrive, please check your spam filter, and contact support@marui-plugin.com

The license credentials must then be used to activate MiVRy.
This activation is local – no internet connection is required and no data is transmitted.
If you're using the MivryActor component, you can just insert the license name and license key in the Details of the component.
If you're using the GestureRecognitionActor or GestureCombinationsActor, you must activate the object by using the activateLicense() function (during runtime).

Using a License File:

Alternatively, you can save the license name (ID) and key into a file and load it with the `activateLicenseFile()` function or input the path to the license file into the MivryActor component if you use it.
The license file is a simple text file that you can create with any text editor, that contains the keywords "NAME" and "KEY", each followed by a colon (":") or equal sign ("=") and then your respective license credentials.
Here is an example of how the contents of a valid license file may look:

```
NAME: your@email.com_3z0UvQ3GBkAc74VW9nQKPlbm
KEY : b701b7235a483698e61a2b8d69479ed013a03069fcb9b892302277a0f394c257
```

5: How to use the GestureManager

You can get the GestureManager app at
https://www.marui-plugin.com/download/mivry/MiVRy_GestureManager.zip

This app allows you to record and edit Gesture Database Files easily in VR.

Important: The GestureManager is written in Unity, so when you use Gesture Database Files created with the GestureManager in Unreal, be sure to check the "Unity Compatibility" option in the MiVRyActor or GestureRecognitionActor.

When you run the GestureManager, a floating panel will appear. You can move the panel by touching the red ball on it's top. The ball is 'sticky', allowing you to move the panel. To stop dragging the panel, just pull your controller away with a sudden "yanking" motion.
A video tutorial on how to use the GestureManager in VR is available on YouTube:
https://www.youtube.com/watch?v=kcoobj7V2-o

Important input fields in the GestureManager:

Number of Parts: How many motions – at most – comprise a gesture. A gesture consisting of one single hand motion has one part. A two-handed gesture has two parts, one for the left hand and one for the right hand. It is also possible to use gesture combinations where one hand has to perform multiple sequential motions (such as writing three letters – the individual letters are parts to a combination triplet). The number you put in this field decides the maximum. You can still also have combinations with less parts (for example: one-handed gestures among two-handed gestures).

Rotational Frame of Reference: How direction like "up", "down", "left", "right", "forward" and "back" are defined. For example, if a player is looking at the ceiling and performs a gesture in front of his face, in the "world" frame-of-reference, the gesture was performed "upward" because it was performed above the player's head. But in the "head" frame-of-reference, the gesture was performed "forward". This can decide which gesture is identified. For example, if you have a "punch the ceiling" gesture and a "punch the ground" gesture, you must choose a "world" frame-of-reference, but if you have a "touch my forehead" gesture and a "touch my chin" gesture, a "head" frame-of-reference may be more appropriate. The frame of reference can be selected separately for yaw (left-right / north-south), pitch (up/down) and roll (tiling the head). The "Rotation Order" should be Yaw -> Pitch -> Roll (ZYX in Unreal, YXZ in Unity).

Record Gesture Samples: This selects I for which gesture you want to record new samples or if you want to test the identification instead (please note that new samples do not have any effect until the "training" was performed). When you record samples, please make sure that you record the gesture many different ways. For example, if the player should be allowed to perform the gesture with a small motion and a large motion, be sure to record both small and large samples. It can also

help to record gesture samples from several people to ensure that particular habits of one person don’t affect the recognition for other players.

Start Training / Stop Training: This starts or interrupts the training process where the AI tries to learn your gestures. The “Performance” value which is updated during the training indicates how many of your gestures the AI can already correctly identify. Even when the training the training is stopped prematurely the result is still preserved, so you can stop it as soon as you are satisfied. Sometimes the AI ‘misunderstands’ your intentions and the future recognition of gestures is not satisfactory. In this case, just re-run the training process. If the result still is not good, please record more gesture samples with greater variation to make it clearer to the AI what you intend.

6: How to use the MiVRyActor gesture recognition object

(1) Add the MivryActor to your level. (If you can't find it in the Content Browser, check the View Options that "Plugin Content" is enabled).

(2) In the Details panel, choose the file path of the Gesture Database file to load (can be an absolute path or a relative path inside the project). If the file was created with the Unity version of MiVRy (including the GestureManager), make sure to check the "Unity Gesture Database File" checkbox.

(3) In the Details Panel, set the fields of the MiVRyActor:

- "Gesture Database File":
The path to the gesture recognition database file to load.
This can be an absolute path (such as "C:\MyGestures\MyGestureFile.dat") or relative to the projects root directory (such as "Content/MyGestureFile.dat")/

- "Left Motion Controller" / "Right Motion Controller": (Optional)
A motion controller component that will be used as the position of the left hand.

- "Left Hand Actor" / "Right Hand Actor" (Optional)
An actor to use as the left and right hand position in the level.

If you set neither of the "Motion Controller" nor "Hand Actor" fields, MiVRy will try to use Unreals AR functions to get the position of your motion controllers, which may not work with all VR-PlugIns.

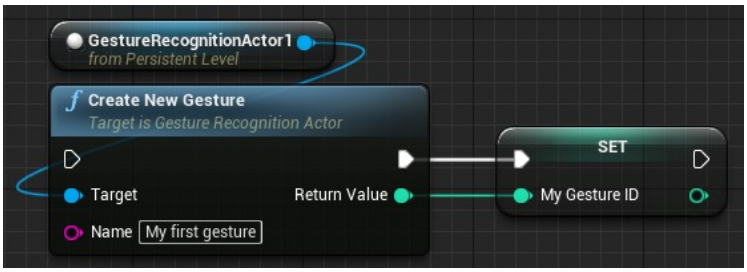
- "LeftTriggerInput" / "RightTriggerInput": (Optional)
The name of the input in the Input Manager (in Project settings) which will be used to start/end the gesture.

If you don't set these, you will have to use the Blueprint or C++ functions of the MiVRy actor to trigger the start and end of a gesture.

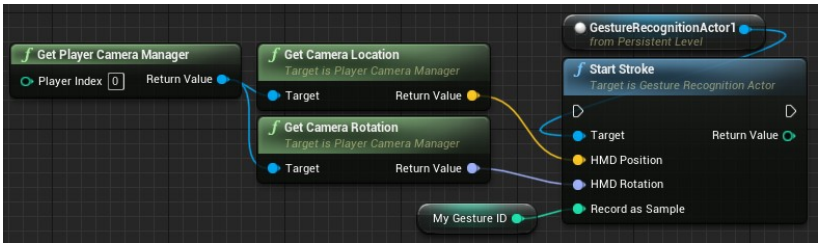
7: How to use the GestureRecognitionActor (for one-handed gestures):

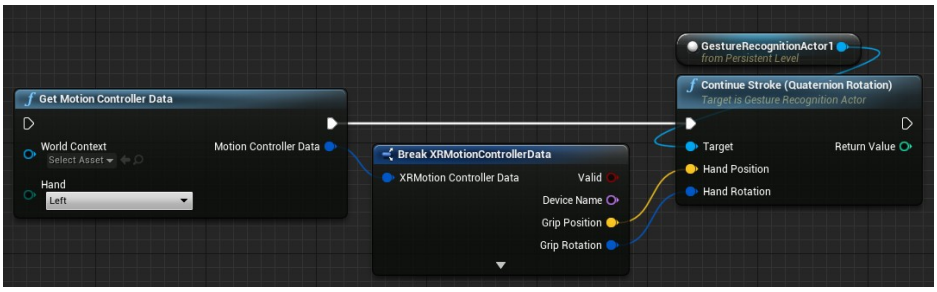
(1) Add a GestureRecognitionActor to your level.

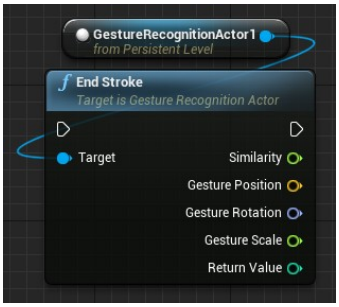
(2) Use the “Create New Gesture” function to create new gestures.



(3) Record a number of samples for each gesture by using the “startStroke” function with the, “contdStroke” and “endStroke” functions for your registered gestures, each time inputting the location and rotation of the headset or controller respectively.

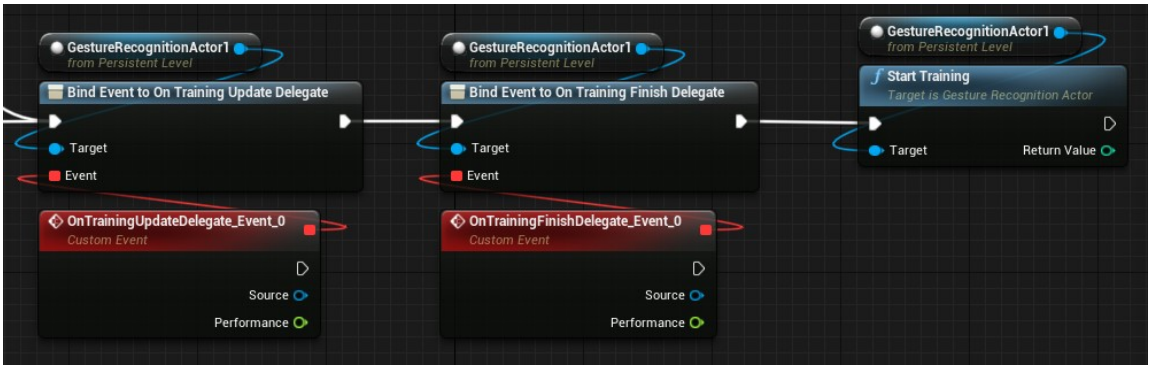






Repeat this multiple times for each gesture you want to identify.
We recommend recording at least 20 samples for each gesture.

(4) Start the training process by using the “startTraining” function.
You can optionally register delegates / callback events to receive updates on the learning progress.



You can stop the training process with the “stopTraining” function.
After training, you can check the gesture identification performance with the “recognitionScore” function.

(5) Now you can identify new gestures performed by the user in the same way as you were recording samples. Just set the “Record as Sample” parameter of the “startStroke” function to “-1”. The “endStroke” function will provide the ID and name of the identified gesture, together with a similarity measure (0 to 1) of how closely the gesture performance resembled the recorded gesture samples.

(6) You can save and load your gestures to a gesture database file with the “Save Gestures To Gesture Database File” function.

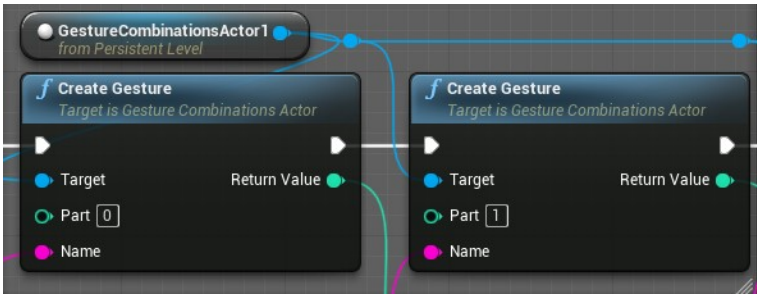
IMPORTANT: If you wish to use your gestures in a Unity app (for example with the Unity-based “GestureManager”, then make sure you enable the “Unity Compatibility Mode” in the Details Panel of the GestureRecognitionActor ***before*** you record any gesture samples!

8: How to use the GestureCombinationsActor (for two-handed gestures or gesture combos):

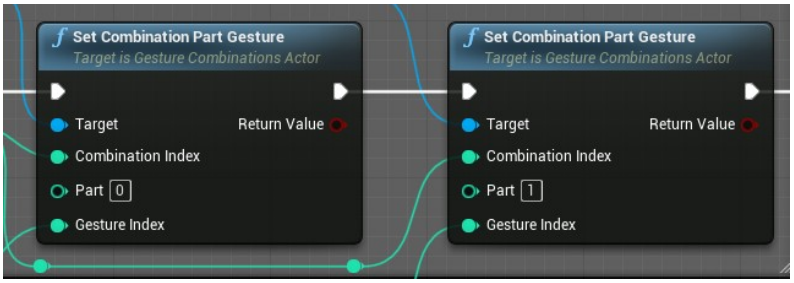
(1) Place GestureCombinationsActor in your level and set the desired “Number of Parts” in the Details Panel. For two-handed gestures, this is usually “2”, but if you intend to use combos of multiple sequential gesture motions for one or two hands, you can choose a different number.

(2) Create a new Gesture Combination.

(3) Create new Gestures for each part, starting with part number “0”. (You can use "0" to mean "left hand" and gesture part "1" to mean right hand, or any other way to identify the different parts of the Gesture Combination.)



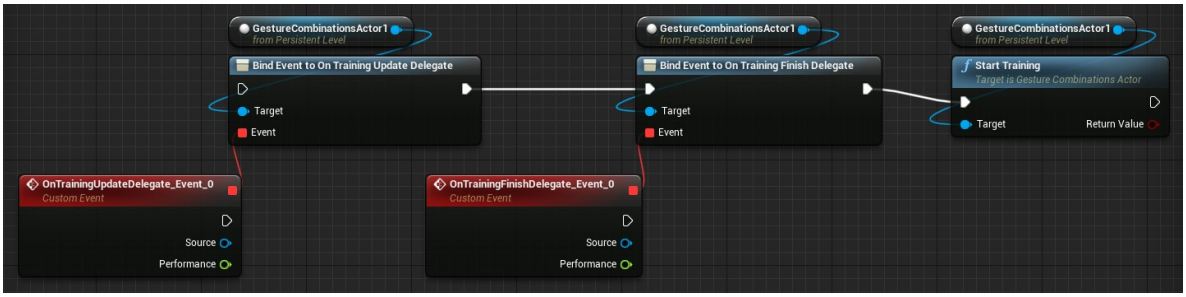
(4) Then set the Gesture Combination to be the combination of those gestures.



(5) Record a number of samples for each gesture using the startStroke, contdStroke and endStroke for your registered gestures. See Section 6 Point 3 of this document for details. The gestures for the various parts can be recorded in any order (first left hand then right or first right hand then left) or simultaneously. We recommend recording at least 20 samples for each gesture, and have different people perform each gesture in different ways.

(6) Start the training process with the startTraining function.

You can optionally register delegates / callback events to receive updates on the learning progress and the end of the training.

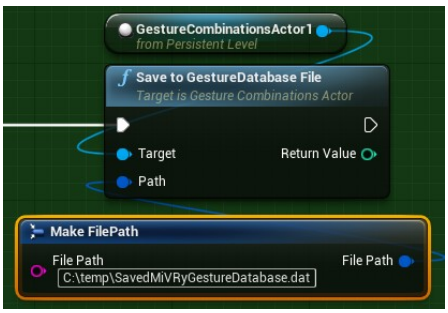


You can stop the training process by using the stopTraining function. After training, you can check the gesture identification performance with the recognitionScore function (a value of 1 means 100% correct recognition).

(5) Now you can identify new gestures performed by the user in the same way as you were recording samples by using the “startStroke”, “contdStoke”, and “endStroke” functions, just by setting the “Record as Sample” parameter to “-1”. Again, the order of performances (first left then right, first right then left, or simultaneously) does not matter.

After all parts (for example left and right hand, or just one hand when only one hand was gesturing) have been completed, use the “Identify Gesture Combination” to find out which Gesture Combination was performed by the user.

(6) Now you can save and load the gestures (and the trained AI) by using the “Save to Gesture Database File” and “Load Gesture Database File” functions. The path can be either absolute or relative within your project.

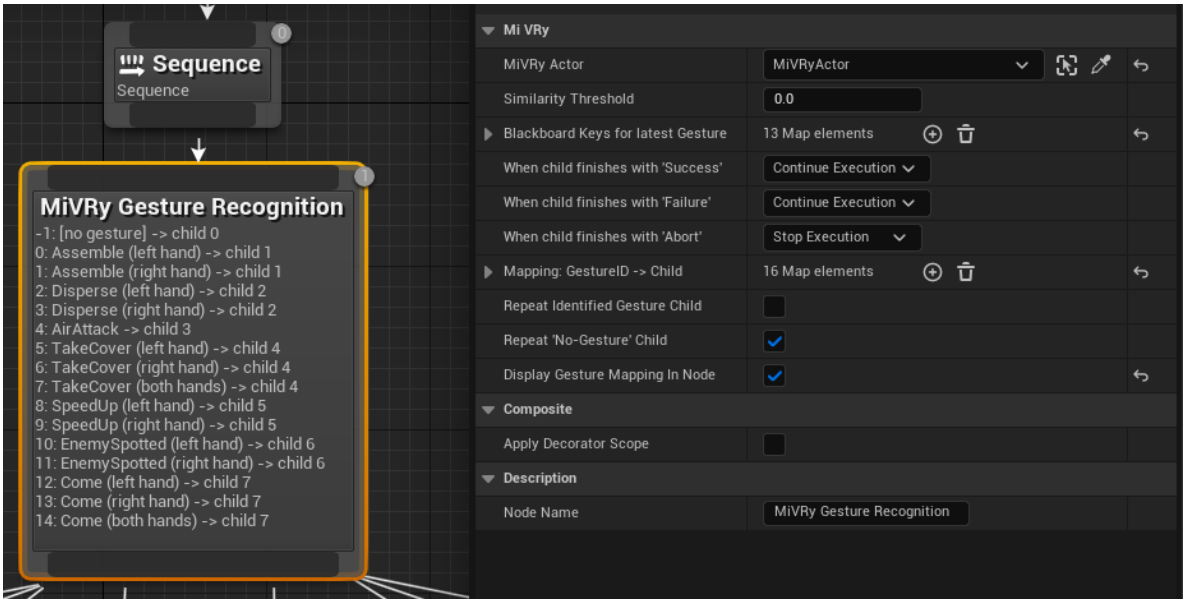


9 How to use the MiVRy Composite and Decorator in UE Behavior Trees

When using Unreal Behavior Trees to control the behavior of NPCs, you can use the MiRy Composite and MiVRy Decorator to control your NPC's behavior via Gestures.

To use the MiVRy Composite or Decorator, you first have to add a MiVRy actor to your level and configure it as described in section 6.

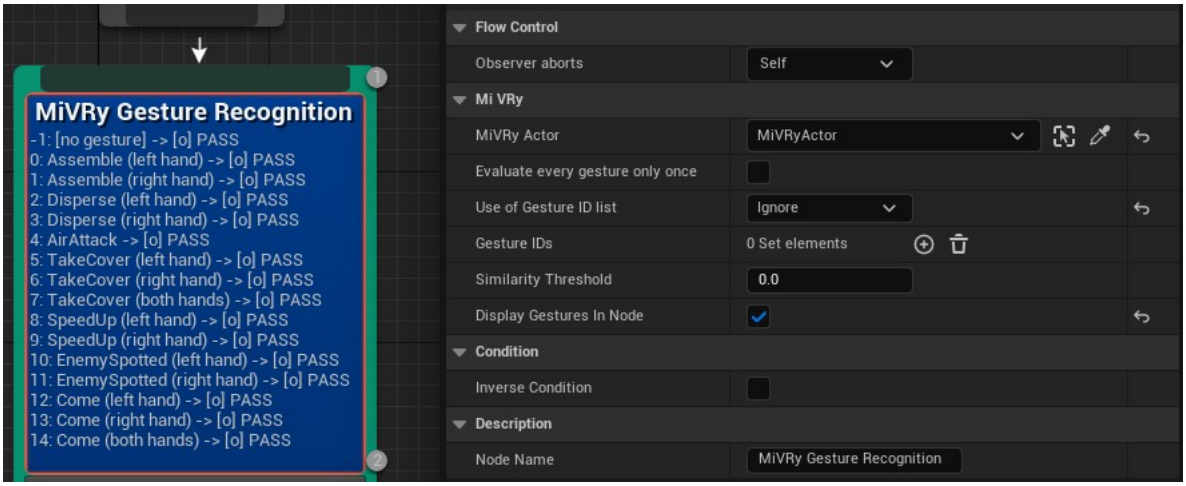
Then open your Behavior Tree and an add a new "MiVRyComposite".



Set the "MiVRy Actor" setting to point to the MiVRyActor which you added to your level.
When you hover your mouse cursor over the various settings, tooltips are displayed which explain the various settings. Most importantly, the "Mapping GestureID -> Child" setting controls which gesture triggers which behavior (sub-tree). Here, it is helpful to activate "Display Gesture Mapping in Node" so that you can see the different gesture names and which child they are mapped to in your Behavior Tree. Please note that the mapping allows every Gesture ID only once. This can cause trouble when the Gesture ID "zero" is already mapped, because then Unreal won't be able to add a new mapping (which would default to "zero" as well). To solve this, temporarily set your "zero" mapping to something else and then add a new mapping.
You can also allow the MiVRy Composite to set Blackboard variables to store details about the performed gesture such as its direction or scale. You can then use these Blackboard variables in your Tasks – for example to make an NPC walk into the direction of a gesture motion or walk faster if the gesture was performed at a larger scale.

However, the Composite node by itself cannot interrupt the current behavior (sub-tree) when a new gesture is performed. It would have to wait for the current behavior to complete before it is

evaluated again. To interrupt the current behavior with a new gesture, you need to add a MiVRy Decorator to the Composite.



The MiVRy Composite can also be used with other Components and can block that Component when certain gestures were (or weren't) performed. See the "Gesture IDs" setting for details.
If you se the Decorator's "Observer aborts" setting to "Self" it will interrupt the current behavior (sub-tree) when a new gesture is identified.

10 Troubleshooting and Frequently Asked Questions (FAQ):

(1) Where and when in my own program do I have to create the MiVRyActor, GestureRecognitionActor or GestureCombinationActor?

You can add any of the actors before run-time or spawn them during run-time. You can also spawn several actors.

(2) How can I use combinations of one-handed and two-handed gestures?

Use a GestureCombinationsActor with two parts (for left hand and right hand) and then create Gesture Combinations that only define a gesture for one part (ie hand). The following table shows an example:

Gesture Combination:	**Gesture on part 0 ("left hand"):**	**Gesture on part 1 ("right hand"):**
Wave (both hands)	Wave	Wave
Wave (left hand)	Wave	-
Wave (right hand)	-	Wave
Salute	-	Salute
Clap hands	Clap	Clap

You can also use the GestureManager to create such GestureCombinations and then load the resulting file with a MiVRy actor.

(3) How do I identify gestures without having to trigger the “start” and “end” of the gesture.

To identify gestures continuously without a clear “beginning” and “end”, use the “contdIdentify” function. You still have to call “startStroke” once (for example at the start of the level), and have to continuously provide the controller position with the “contdStroke” function. Then, after the “contdStroke”, use the “contdIdentify” function to identify the currently performed gesture. Use the contdIdentificationPeriod value to control how long of a time frame to consider in the identification. You can also use contdIdentificationSmoothing to avoid the identification result from jumping from one gesture ID to another too easily.

(4) How can I open and edit gesture database (.DAT) files?

Please use the "GestureManager" (https://www.marui-plugin.com/documentation-mivry-unreal/#gesturemanager) to open and edit “.DAT” gesture database files. Please note that the GestureManager is based on Unity, so if you record gestures in the GestureManager and want to use them in Unreal, enable the “Unity Compatibility Mode” in your MiVRyActor, GestureRecognitionActor, or GestureCombinationsActor.

(5) The Gesture Recognition library does not detect if a gesture is different from all recorded gestures. How do I find out if the user makes a completely new (or “wrong”) gesture?

The gesture recognition plug-in will always return the number of the gesture which is most similar to the one you just performed.
If you want to check if the gesture you made is different from all the recorded gestures, check the “Similarity” value that you receive when identifying a gesture. This is a value between “0” and “1”, where “zero” means that the gestures are completely different, and “one” means that the performed gesture is a perfect average of all the recorded samples.

Thus, a value of one will indicate perfect similarity, a low value close to zero indicate great differences between the performed gesture and the recorded gesture samples. You can use this value to judge if the performed gesture is sufficiently similar to the recorded one.

(6) I want to use Gesture Recognition in my commercial project. What commercial licensing options do you provide?

Please contact us at support@marui-plugin.com for details.

(7) Do I have to use the "startTraining" function every time I start my game? Does it have to keep running in the background while my app is running?

No, you only need to call startTraining after you have recorded new gesture data (samples) and want these new recordings to be used by the AI. However, you need to save the AI after training to a database file (.DAT) and load this file in your game before using the other gesture recognition functions.
While the training is running, you cannot use any of the other functions, so you cannot let training run in the background. You must start (and stop) training in between using the AI.

(8) How long should I let the training run to achieve optimal recognition performance?

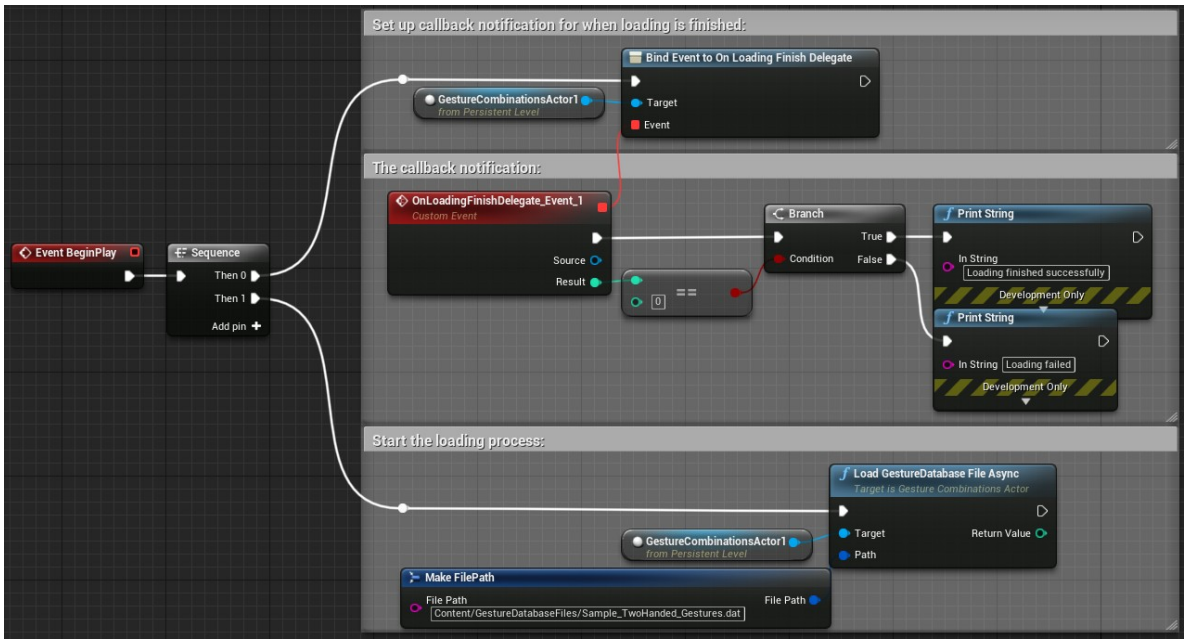
Usually, the AI will reach its peak performance within one minute of training, but if you're using a large number of gestures and samples, it may take longer. You can check the current recognition performance from the training callback functions and see if the performance still keeps increasing. If not, feel free to stop the training.

(9) Gestures aren't recognized correctly when I look up/down/left/right or tilt my head.

You can choose if the frame of reference for your gestures are the players point of view ("head") or the real world or game world ("world"). For example, if the player is looking up to the sky when performing a gesture towards the sky, then from a "world" frame-of-reference the direction is "up", but from players "head" point-of-view, the direction is "forward". Therefore, if you consider your gestures to be relative to the world "up" (sky) and "down" (ground) rather than the visual "upper end of the screen" and "lower end of the screen", then change the ***frameOfReferenceUpDownPitch*** *to* ***FrameOfReference.World****. The same setting is available for the yaw (compass direction) and head tilt.*

(10) Loading GestureDatabase files takes a long time, freezing the app for a few seconds. How can this be avoided?

You can "Load Gesture Database File Async" (or "Load Gesture Database Buffer Async") functions instead. These start the loading process in the background and will return almost immediately. Note that you will be unable to use the GestureRecognitionActor / GestureCombinationsActor while the loading takes place in the background. You can use the "Is Loading" function to test if the Actor is still loading, and you can also set up a callback event ("On Loading Finish Delegate") to receive a notification when the loading process finished. The callback will include the result (zero on success or a negative error code on failure) of the loading process.



(11) After some time, all attempts to identify a gesture fail with error code -16.?

You have used up all "free" gesture recognitions of the free version of MiVRy for this session. To identify more gestures, restart the app, or purchase an "unlimited" license at https://www.marui-plugin.com/mivry/

(12) MiVRy identifies any motion as some gesture, even when it doesn't resemble any of the recorded gestures. Why? How can I tell if no valid gesture motion was performed?

MiVRy will always tell you the "most likely" best guess as to which gesture was just performed, no matter how different the currently performed motion is from all recorded gestures. This is because we cannot decide for you how much difference is tolerable.
In order to disqualify "wrong" motions, you have two options:
(A) you can check the "similarity" value returned by MiVRy. This value describes how similar the gesture motion was compared to previous recordings on a scale from 0 (very different) to 1 (very similar).
(B) you can check the "probability" value. Especially when you compare the probability values for all recorded gestures (for example via the "endStrokeAndGetAllProbabilitiesAndSimilarities" function) and see that they are all very low and not very different from one another, you may want to decide that the current gesture performance was not valid.

(13) What exactly does die "similarity" value of a gesture performance mean? How is it different from the probability value?

The "similarity" value expresses how much the identified gesture differs from the average of the recorded samples for that gesture. When you record several samples, MiVRy internally calculates a "mean" ("average", "typical") gesture motion based on those samples. It also calculates how much the recorded samples differ from this "mean" (ie. the "variance" of the samples). The "similarity" value is then calculated based on this "mean". If your newly performed gesture motion hits exactly this "average", then the similarity value will be one. The more it differs, the lower the "similarity" value will be, going towards zero. How fast it will fall depends on how similar the recorded samples were. If all recorded samples looked exactly the same, then MiVRy will be very strict, and the "similarity" value will fall fast when the currently performed motion isn't also exactly alike. If, however the samples differed a lot, MiVRy will be more tolerant when calculating the "similarity" value and it will be higher. The value is always between 0 and 1. This "similarity" is different from the "probability" values, which are estimates by the artificial intelligence (neural network). "Probability" may contain many more considerations, for example if there are other gestures who resemble the identified gesture (probability may drop, similarity is unaffected), or if there are a multitude of distinct motions lumped together as one "gesture" (for example: having a gesture "alphabet" which contains drawings of "A", "B", "C" etc all lumped together as one gesture - then "similarity" will be calculated based on an "average" character that doesn't resemble any sample, but the AI may successfully understand what you mean and give high "probability" values).

(14) Instead of triggering the start and end of a gesture motion, I want MiVRy to constantly run in the background and detect gestures as they occur.

You can use the "Continuous Gesture Identification" feature of MiVRy. When using the "GestureRecognition" or "GestureCombinations" objects directly, use the "contdIdentify" function - you can call this function repeatedly (for example on every frame or when something in your app happens) and every time it will tell you which gesture is currently being performed. When using the Unity "Mivry" component or the UnrealEngine "MivryActor", use the "Continuous Gesture Identification" switch. Either way, two settings are important for Continuous Gesture Identification: "Continuous Gesture Period" and "Continuous Gesture Smoothing". "Continuous Gesture Period" is the time frame (in milliseconds) that continuous gestures are expected to be. So if your gestures take 1 second to perform, set this to "1000" so that MiVRy will consider the last 1000 milliseconds to identify the gesture. "Continuous Gesture Smoothing" is the number of samples (previous calls to "contdIdentify" to use for smoothing continuous gesture identification results). When setting this to zero, each attempt to identify the gesture will stand alone, which may lead to sudden changes when switching from one gesture to

another. If ContinuousGestureSmoothing is higher than zero, MiVRy will remember previous attempts to identify the gesture and will produce more stable output.

11 Build / Packaging Instructions

(1) Add the folder of your GestureDatabase files to be included in the build. In the "Project Settings", go to the "Packaging" section find the "**Additional Non-Asset Directories to Copy**" settings and add the folder where your GestureDatabase files are located.
If you don't want the folder to be *copied* (resulting in a directory on windows or .OBB file on Android/Quest), but instead want the gesture database files to be included in the PAK or APK file, then add the gesture database file folder to "**Additional Non-Asset Directories to *Package***". However, if you do, you will not be able to use *GestureRecognitionActor::loadFromFile()* and *GestureCombinations::loadFromFile()* functions. Use *FFileHelper::LoadFileToArray()* and *GestureRecognitionActor:: loadFromBuffer()* functions instead.

(2) If you're building for the Quest, please follow the official guide:
https://developer.oculus.com/documentation/unreal/unreal-quick-start-guide-quest/
Also, in the Project Settings, "Android" section, you have to enable **either** "Support for arm64" **or** "Support for armv7" and disable the other.
Furthermore, if in (1) you chose to add the GestureDatabase files folder be *copied* (into an OBB file) then you might have to disable the "**Package game data inside .apk**" option.

12 Software license statement (EULA):

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

[IMPORTANT!] This license is for the gesture recognition plug-in (.dll and .so filed and source code) and does NOT include any permission to use the asset and resource files used in the samples (3D character model and textures etc.)