How to Use Gestures or Create Your Own Gestures

First off, a simple way to test the pre-defined gestures in Unity is to utilize the SimpleGestureListener-script component. It can be found in KinectScripts/Samples-folder. Attach it to a game object in the scene and then add the gestures you want to test to its 'Detect gestures'-list setting. Set the 'Gesture Info'-setting to point to some UI-Text in the scene, then run the scene and try each one of these gestures. The status of the currently detected gesture should be displayed on the referenced UI-Text.

SimpleGestureListener demonstrates how to setup the gesture detection as Unity script. Please look at its code, to see how it implements the GestureListenerInterface. You can create similar scripts to provide your own gesture detection and process the detected gestures as needed. Some examples can be found in the demo scenes in KinectDemos/GestureDemo-folder. Here is a short description of the methods of GestureListenerInterface:

- UserDetected() – invoked when a new user is detected. It may be used to create the list of gestures of interest.
- UserLost() – invoked when a user is lost. It may be used to free the allocated resources, if needed. Detection of gestures, added in UserDetected(), is stopped automatically.
- GestureInProgress() - invoked when a gesture in progress is detected, but the gesture is not yet completed or cancelled. Can be used to report the gesture progress, or to process the continuous gestures, because they never get completed.
- GestureCompleted() - invoked when a discrete gesture is completed. You can process the gesture here, and decide whether to reset the gesture (i.e. restart its detection) or not.
- GestureCancelled() - invoked, if the gesture gets cancelled. The gesture is automatically cancelled, when it is not completed within the allowed time interval. Here you can decide, whether to reset the gesture (i.e. restart its detection) or not.

Currently Detected Gestures

Here is the list of currently detected gestures, along with short descriptions for each one of them:

- *RaiseRightHand / RaiseLeftHand* – left or right hand is raised above the shoulder and the user stays in this pose for at least 1.0 seconds (pose).
- *Psi* – both hands are raised above the shoulder and the user stays in this pose for 1.0 seconds (pose).
- *Tpose* – the hands are to the sides, perpendicular to the body (T-pose), for 1.0 seconds (pose).
- *Apose* – the hands are to the sides, at least 20 cm away from the waist, for 1.0 seconds (pose).
- *Stop* – right hand is down and left hand is slightly to the side, but below the waist, or left hand is down and right hand is slightly to the side, but below the waist (pose).
- *Wave* – right hand is waved left and then back right, or left hand is waved right and then back left.
- *SwipeLeft* – right hand swipes left.
- *SwipeRight* – left hand swipes right.
- *SwipeUp / SwipeDown* – swipe up or down with the left or right hand

- *ZoomIn* - left and right hands are to the front and put together at the beginning, then the hands move apart in different directions (continuous gesture, reports zoom-factor).
- *ZoomOut* - left and right hands are to the front and at least 0.7 meter apart at the beginning, then the hands move closer to each other (continuous gesture, reports zoom-factor).
- *Wheel* - left and right hands are to the front and shoulder size apart at the beginning, then the hands keep the distance and move as if they turn an imaginary wheel counter clockwise, which returns positive angle, or clockwise - negative angle (continuous gesture, reports wheel angle).
- *Jump* – the hip center gets at least 10 cm above its last position within 1.5 seconds.
- *Squat* - the hip center gets at least 10 cm below its last position within 1.5 seconds
- *Push* – push forward with the left or right hand within 1.5 seconds.
- *Pull* - pull backward with the left or right hand within 1.5 seconds.
- *ShoulderLeftFront* – move the left shoulder to the front.
- *ShoulderRightFront* – move the right shoulder to the front.
- *LeanLeft* - lean left with the whole body (continuous gesture, reports lean angle).
- *LeanRight* – lean right with the whole body (continuous gesture, reports lean angle).
- *LeanForward* – lean forward with the whole body (continuous gesture, reports lean angle).
- *LeanBack* – lean back with the whole body (continuous gesture, reports lean angle).
- *KickLeft* – move the left foot to the front.
- *KickRight* – move the right foot to the front.
- *Run* – checks if the left knee is at least 10cm above the right knee, then – if the right knee is at least 10 cm above the left knee, then – back to the left knee check (continuous gesture, reports progress).
- *Raised right horizontal left hand* – the right hand is raised above the shoulder, while the left hand stays horizontal at shoulder level (courtesy of Andrzej W).
- *Raised left horizontal right hand* – the left hand is raised above the shoulder, while the right hand stays horizontal at shoulder level (courtesy of Andrzej W).
- *TouchRightElbow* – the fingers of the left hand touch the right elbow for at least 1.5 seconds.
- *TouchLeftElbow* – the fingers of the right hand touch the left elbow for at least 1.5 seconds.
- *MoveLeft* – the hip center moves at least 25 cm to the left within 1.5 seconds.
- *MoveRight* - the hip center moves at least 25 cm to the right within 1.5 seconds.

How to Add Your Own Gestures

The detection of gestures is implemented in KinectScripts/KinectGestureManager.cs or in class that extends it. The detection of a gesture consists of checking for gesture-specific poses in the respective gesture state. To add your own gesture detection procedure, please do as follows:

1. Open KinectScripts/KinectGestureManager.cs and add the name of your gesture to the GestureType-enum. Unfortunately, the enums in C# cannot be extended. That's why you should modify it, to add your gesture name here. Alternatively, you can useone of the predefined UserGestureX-names for your gesture, if you prefer.
2. Find CheckForGesture()-method in KinectGestureManager, and add a 'case' for your gesture, at the end of its internal 'switch'. It will contain the code that detects the gesture.

3. In the gesture's 'case', add an internal 'switch' that will check for gesture's pose in the respective state. Each gesture is a state machine with numerical states (0, 1, 2, 3…). Its current state along with other needed data, is stored in an internal structure of called GestureData. This data-structure is created automatically for each gesture that has to be detected in the scene.

The initial state of each gesture is 0. When in this state, your code needs to detect whether the gesture is started or not. It usually checks for some pose condition and stores the position of the active joint (e.g. left or right hand). If the pose condition is met, it increments the gesture state. See SetGestureJoint()-method.

In the next state, the gesture detection routine usually checks, if some other pose condition is met, like if the active joint has reached some target position or distance within some time interval. If the pose condition is met (e.g. the joint has reached its target position or distance), it increments the state and repeats this step, or marks the gesture as completed. See SetGestureJoint() and CheckPoseComplete()-methods.

If the condition is not met (e.g. timed out), the gesture is marked as cancelled. Then the gesture detection starts again from state 0. See SetGestureCancelled()-method.

Please note, in your code you have access to the local 'jointsPos'-array, containing the joint positions and 'jointsTracked'-array, that contain info whether the the respective joints are currently tracked or not. The joint positions are in world coordinates, in meters.

Feel free to see the code of some simple gestures (like RaiseLeftHand, RaiseRightHand, SwipeLeft or SwipeRight), if you need examples.

More Information, Support and Feedback

Online Documentation: https://github.com/rfilkov/k4adocs/wiki
Tip and Tricks: https://rfilkov.com/2019/08/26/azure-kinect-tips-tricks/
Web: https://rfilkov.com/2019/07/24/azure-kinect-examples-for-unity/

Contact: http://rfilkov.com/about/#contact (please mention your invoice number)
Twitter: https://twitter.com/roumenf
LinkedIn: https://www.linkedin.com/in/rfilkov/