SenseGlove Unity SDK

Documentation and troubleshooting

This guide is meant to help developers understand the SenseGlove SDK and how to work with it. Please note that the SenseGlove SDK is still in an early stage, and that you will be working with prototypes. We may not be able to guarantee that everything works 100% of the time, but we can 100% guarantee you that we will do your best to help you in any way we can!

Contents

Getting Started

The SenseGlove SDK works without any 3rd party plugins. Note that most of its features are still in an experimental stage until we reach version 1.0.

Please run through the following steps to ensure your SenseGlove is working:

1. Clone or download the latest version of the SenseGlove SDK, available from https://github.com/Adjuvo/SenseGlove-Unity
2. Import the unitypackage into your Unity project.
3. Ensure your SenseGlove is connected to and recognized by your computer.
    a. It should show up in the Device Manager under "Ports (COM & LPT)", as "Teensy USB Serial".
4. Open any of the example scenes and press play. Verify that your SenseGlove is working and that the glove's orientations match that of your own. You might want to calibrate the wrist or fingers to get the most out of your glove. More on that in the Calibration section.

If you would like to add a SenseGlove Prefab to your existing Scene, follow steps 1-4 to ensure your glove is working, then continue with the following steps:

5. Drag one of the wireframes from the Prefab folder into your scene.
6. That's it! Your SenseGlove should now be moving in your scene!

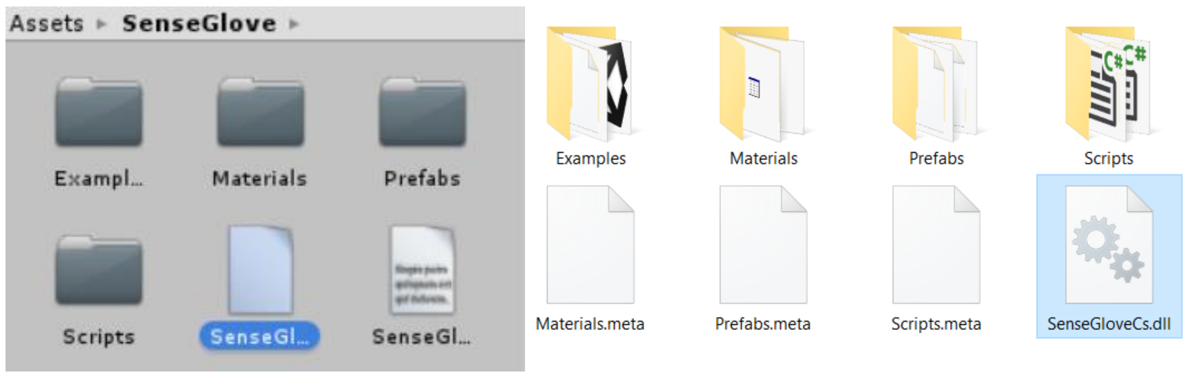
If you wish to link your SenseGlove to a GameObject under your control (An HTC Vive controller, for example), you need to assign this GameObject to the trackedObject property of the SenseGlove_Wireframe using the inspector.

The Wireframe model copies its relative position to the trackedObject on StartUp(), so perhaps some adjustments are required to get the wrist and/or lower arm to align correctly.

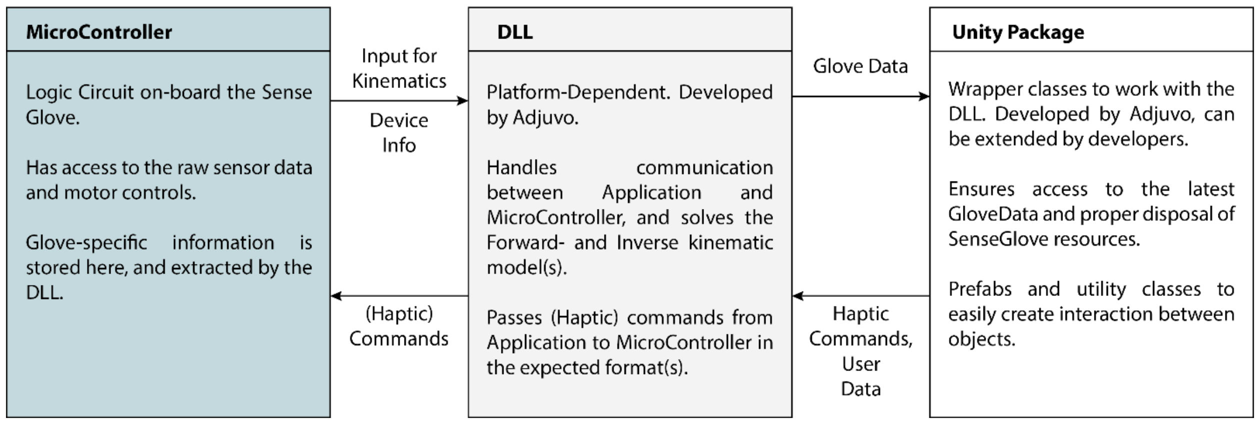
If you wish to know how to integrate the SenseGlove into your own hand-model, consult the "Creating your own hand model" section.

The SDK

Before we get into greater detail, let's talk about the core of the SenseGlove SDK: The SenseGloveCs library, which can be found in the root folder of the SenseGlove unitypackage.



This Dynamic Linked Library (DLL) contains a variety of classes and methods that facilitate the communication between the SenseGlove and the PC, as well as the mathematics used to calculate joint angles. The figure below should give you an idea of its role in the SDK.



MicroController

Logic Circuit on-board the Sense Glove.

Has access to the raw sensor data and motor controls.

Glove-specific information is stored here, and extracted by the DLL.

DLL

Platform-Dependent. Developed by Adjuvo.

Handles communication between Application and MicroController, and solves the Forward- and Inverse kinematic model(s).

Passes (Haptic) commands from Application to MicroController in the expected format(s).

Unity Package

Wrapper classes to work with the DLL. Developed by Adjuvo, can be extended by developers.

Ensures access to the latest GloveData and proper disposal of SenseGlove resources.

Prefabs and utility classes to easily create interaction between objects.

On its own, this DLL can be used by any .NET application (version 3.5+ and up) to work with the SenseGlove. In the Unity SDK, it is used almost exclusively by the *SenseGlove_Object* script. Most Unity developers will only deal with one class from this library; *GloveData*.

The DLL is accompanied by an XML file of the same name, which contains documentation generated by *Visual Studio* to provide tooltips for each of the classes and methods within.

Sense Glove

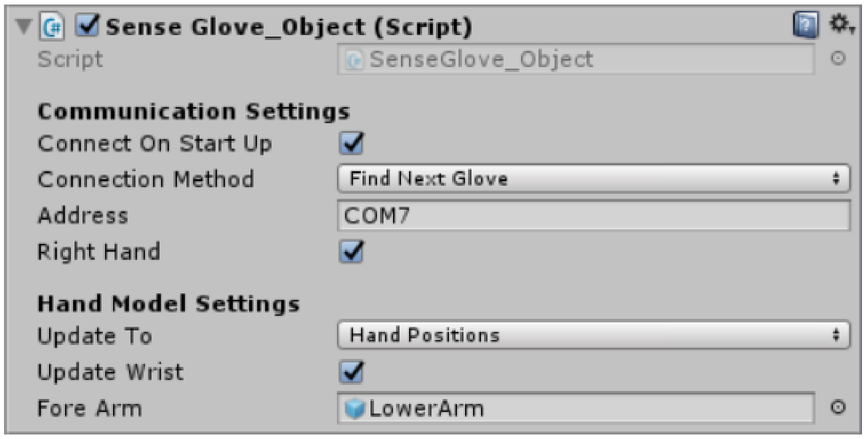
Unity Integration

Without touching any code, we will show you how to make the most out of the SenseGlove via the Inspector.

SenseGlove_Object

The *SenseGlove_Object* script uses several classes within the DLL access the data from a SenseGlove. No matter your application, you will most likely use this script.

Drag this script into your scene, and you can access a single SenseGlove's data: Every *Update()*, this class retrieves the latest data, such as joint angles and battery level. It also ensures that all resources and connections are properly disposed of when the program shuts down again.



Sense Glove_Object (Script)

Script	SenseGlove_Object
Communication Settings	
Connect On Start Up	✓
Connection Method	Find Next Glove
Address	COM7
Right Hand	✓
Hand Model Settings	
Update To	Hand Positions
Update Wrist	✓
Fore Arm	LowerArm

Communication Settings

The communication settings determine *how* Unity will establish a connection with the glove via the *connectionMethod* variable.

connectOnStartUp

Set this Boolean to true if you wish to look for SenseGloves once the application starts. If you wish to manually decide when the *SenseGlove_Object* connects, set this Boolean to false and use the *RetryConnection()* method instead.

connectionMethod

This enumerator determines how the connection is established. Its default value is *FindNextGlove*, which will have the *SenseGlove_Object* connect to the first unconnected glove, if any are available. *FindNextRightGlove* and *FindNextLeftGlove* work the same way, but will specifically connect to a left- or right handed glove. Finally, the *HardCoded* connection method has the script connect to the port listed in the address variable. The type of connection depends on the address format.

address
The address that the SenseGlove is connected to, or will connect to in case the *connectionMethod* is set to *HardCoded*. It is always set to the correct value during each *Update().*

rightHand
Shows if the glove connected to this *SenseGlove_Object* is a left or right hand. It is always set to the correct value during each *Update().*

HandModel Settings
These settings apply to the kinematic model of the hand that is used by the SenseGlove. By changing them, you can decrease the amount of calculations required by the DLL, and use only the data relevant for your project.

updateTo
You can change the level of kinematic calculations using the *UpdateTo* variable. Most calculations, such as the Hand Angles, require that we have access to the latest Glove Positions first.

Updating to the *GlovePositions* will give you enough to create a model of the SenseGlove, while updating to the *HandAngles* will give you access to the joint (Euler) angles relative to the previous joint. Finally, updating all the way to *HandPositions* will give you the location of each joint in the hand [x,y,z, in mm], relative to the common origin.

updateWrist
Using the *updateWrist*, you can determine if the SenseGlove's wrist model is also updated. This can be done 'on the fly'. Note that this option is disabled automatically when the *SenseGlove_Object* is tracked by a *SenseGlove_WireFrame* that is attached to the wrist.

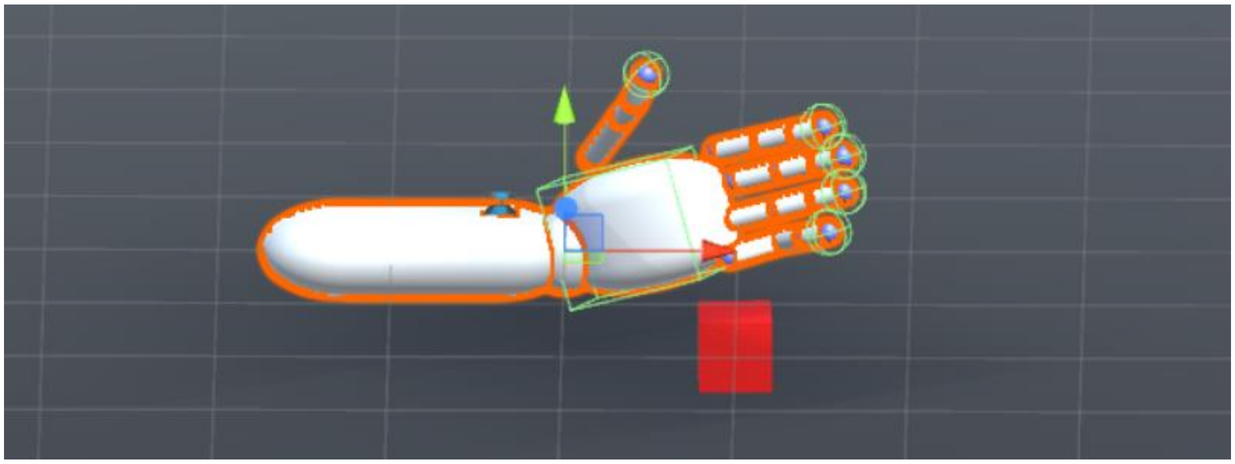
foreArm
By entering a GameObject into the *foreArm* property of the *SenseGlove_Object*, that object is treated as the 'lower arm' for its wrist model. This GameObject's orientation will become the 'base' position for the hand. If you are not using the lower arm functionality of the SenseGlove, you may ignore this setting.

Grabbing and Interacting

SenseGlove_PhysGrab

The SenseGlove GrabScript uses a series of colliders to determine when an object can be interacted with, and when this interaction should end.

Currently, the SenseGlove can interact with an object if it is being touched by the thumb collider(s) or the hand palm, and at least one other finger collider. These settings can be changed through the *SenseGlove_PhysGrab* script.



The SenseGlove grab script will only interact with a *GameObject* that has a *SenseGlove_Interactable* script attached to it. The object itself determines which type of interaction it provides. That way, the glove will not unintentionally pick up any GameObject that is part of the background.

SenseGlove_Interactable objects

This script can be extended to create dynamic controls, such as levers, buttons or drawers. Through the *isInteractable* property, one can control when these interactions become available to the user.

Currently, its only available 'Interactable' is the *SenseGlove_Grabable* script, which one can attach to any object with a rigidbody. With this script attached, the object can be picked up by the SenseGlove's grab script.

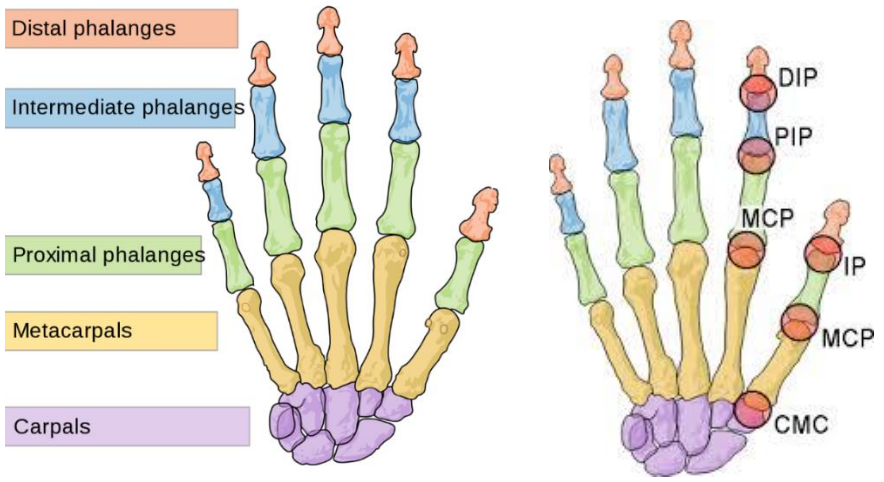
Gesture Recognition

Is currently not implemented, but is on the list of features for version 1.0.

It should offer methods similar to the *"GetKey()"*, *"GetKeyDown()"* and *"GetKeyUp()"* of Unity's *Input* class, allowing developers control over events like they would a keyboard. A rudimentary version can be implemented by third parties through the SenseGlove's *handAngles* variable.

Anatomy of the Hand

The SenseGlove was initially developed for orthopedic rehabilitation, and thus uses a number of anatomical terms in its documentation. We encourage you to read this quick introduction so you know what some of the more obscure terms and abbreviations stand for.



SenseGlove data related to the fingers is always given from thumb to pinky, from proximal (closest to the body) to distal (furthest from the body). The model of the thumb starts at the Carpo-MetaCarpal (CMC) joint, and that of the other fingers starts at the MetaCarpal-Phalangeal (MCP) Joint. The same structure is used for data relating to the glove itself.

Terms

Phalanges – The bones that make up the fingers.

Proximal– Positioned closer to the body / trunk.

Distal – Positioned further away from the body / trunk.

Pronation / Supination – Rotation to move the palm of the hand downward or upward.

Abduction / Adduction – Abduction and adduction refer to motions that move a structure away from or towards the center of the body. It refers to the spreading of the fingers and the opposition of the thumb.

Flexion / Extension – Flexion describes a bending movement that decreases the angle between a segment and its proximal segment. When you make a fist, you are flexing your fingers.

Abbreviations

CMC Joint – Carpo-Metacarpal Joint

MCP Joint – MetaCarpal-Phalangeal Joint

PIP Joint – Proximal Interphalangeal Joint

DIP Joint – Distal Interphalangeal Joint

Working with Glove Data

The *SenseGlove_Object* allows access to its *GloveData*, which can be seen as a ‘snapshot’ of all of the SenseGlove’s values for this particular frame. This *GloveData* is updated with every *Update()* event in Unity and converted into a *SenseGlove_Data* object. The *SenseGlove_Data*, which contains the *GloveData* converted into Unity-friendly variables, can be accessed by other scripts using the *GloveData()* method.

Note that both *GloveData* and *SenseGlove_Data* are copies from the *GloveData* contained within the DLL, and that any changes made to these objects will *not* be passed back to the *SenseGlove* object. If you wish to change any of the values within the SenseGlove (which you shouldn’t unless you know what you are doing), use the wrapper functions provided through the *SenseGlove_Object* script.

Initialization

Upon connecting, the DLL will begin to load glove-specific data from the SenseGlove, referred to as ‘constants’, which takes a few milliseconds. These constants include the lengths and number of segments in the glove, as well as other data that is required before it can begin calculations. It is therefore advised to subscribe scripts that are dependent on glove-specific data for their initialization to the *OnGloveLoaded* event of the *SenseGlove_Object*. This event fires just after the constants have been received and parsed.

Format & Notation

The *SenseGlove_Data* follows anatomical terms; always working from thumb to pinky, from proximal (closest to the body) to distal (furthest from the body).

All Euler angles in the DLL are given in degrees, while all positions are given in millimeters, relative to a common origin. Both the hand and the glove positions are relative to this common origin. Quaternion rotations are also given relative to the common origin.

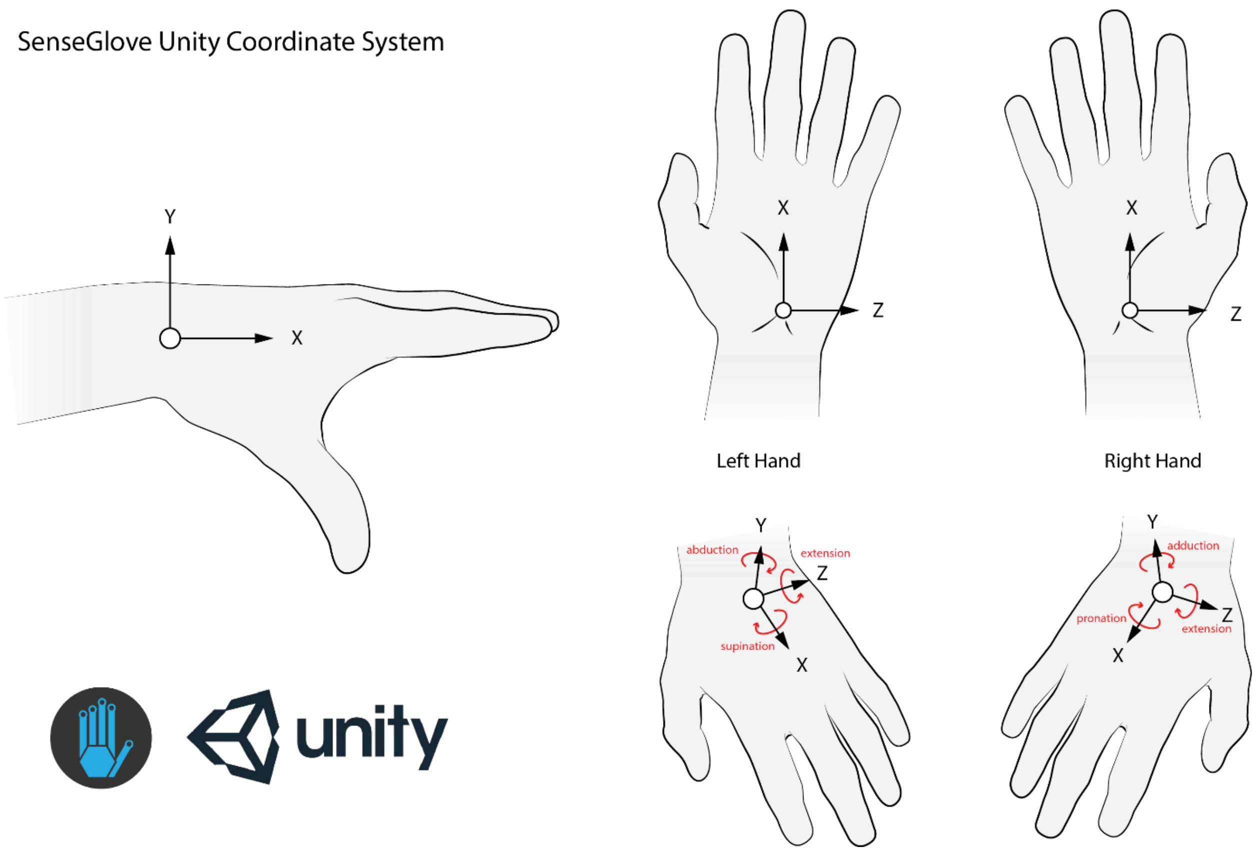
Multidimensional Arrays

Within the *SenseGlove_Data*, you will find a number of multidimensional arrays. These are used to separate the different types of data for each joint in the hand and the glove. Because one can iterate over these arrays, it keeps the models flexible.

When dealing with 2-dimensional (N x M) arrays, the first index (N) is used to indicate the fingers, from the thumb (0) to the pinky (4). The second index (M) is used to access a specific point / joint in that array, from proximal to distal.

Coordinate System

The *SenseGlove_Data* uses Unity's left-handed coordinate system. This coordinate system is the same for both the left- and right hand. Currently, the common origin is placed in the index finger link of the glove, where it meets the main body of the glove.



- The x-axis is aligned along the metacarpal bone of the middle finger.
- The y-axis runs perpendicular to the hand palm, going up from the dorsal side of the hand.
- The z-axis points towards the thumb of the right hand, and towards the pinky of the left hand.

When working with joint angles, note that pronation/supination and abduction/adduction have opposite signs (+/-) for a left- and right hand due to them being opposite sides of the body.

- Pronation / Supination, a.k.a. the twist of the finger, is the angle around the x-axis (roll).
- Abduction / Adduction, a.k.a. the spreading of the fingers, is the angle around the y-axis (yaw).
- Flexion / Extension of the finger is the angle around the z-axis (pitch)

General Data

Any data that tells you something about the SenseGlove before making any of the calculations. This data lends itself well for initialization, debugging or UI purposes.

Variable name	Variable Type	Description
dataLoaded	Boolean	Indicates if glove-related variables have been loaded from the SenseGlove.
deviceID	String	Identifier unique to this SenseGlove.
gloveVersion	String	The hardware version of the glove.
firmwareVersion	String	The firmware version running on the glove Microcontroller.
isRight	Boolean	Tells you if this is a left- or righthanded SenseGlove.

Sensor Values

(Raw) sensor values that are streamed form the SenseGlove to the PC.

Variable name	Variable Type	Description
gloveValues	float[5][]	The angles, in radians, received from the SenseGlove. Sorted per finger, from proximal to distal. Basically the gloveAngles but not yet assigned to the correct axes.
imuValues	float[4]	The raw xyzw values of the SenseGlove's Inertial Measurement Unit.
packetsPerSecond	int	The number of sensor packets received from the SenseGlove per second.

Wrist

Wrist angles are stored as quaternions, though one can still access their Euler notation through the *eulerAngles* property.

Variable name	Variable Type	Description
absoluteWrist	Quaternion	The absolute Quaternion rotation of the wrist, compensated with the IMU's orientation on the hand and glove.
absoluteCalibratedWrist	Quaternion	The absolute Quaternion rotation of the wrist, calibrated to move relative to the forearm.
relativeWrist	Quaternion	The wrist orientation relative to the GameObject designated as the foreArm.

Hand Model

All there is to know about the kinematic model of the fingers and thumb.

Variable name	Variable Type	Description
gloveRelPos	Vector3	Position of the common origin relative to the wrist.
gloveRelOrient	Quaternion	Quaternion rotation of the common origin relative to the wrist.
gloveLengths	Vector3[5][]	The lengths [xyz] of each segment of the glove, in mm. Some of these lengths can be negative.
gloveStartRotations	Quaternion[5]	The starting quaternion rotations of each glove segment.
gloveRotations	Quaternion[5][]	The quaternion rotations of the glove joints, relative to the common origin.
gloveAngles	Vector3[5]	The Euler angles relative to the previous link / phalange. [pronation, flexion, abduction]
glovePositions	Vector3[5]	The x,y,z position in mm of each glove joint, relative to the common origin.
handLengths	Vector3[5][3]	The lengths of the finger phalanges. Of these, the x-coordinate is most relevant.
handStartRotations	Quaternion[5]	The starting quaternion rotations of the MCP joints of the fingers and the CMC joint of the hand.
handAngles	Vector3[5][3]	The Euler angles relative to the previous link / phalange. [pronation, flexion, abduction]
handRotations	Quaternion[5][4]	The quaternion rotations of the finger, relative to the common origin. Includes that of the fingertip.
handPositions	Vector3[5][4]	The x,y,z position, in mm, of the finger joints relative to the common origin. Includes the fingertip.

Other data

The available data should be sufficient to create a fully working hand model. However, certain data, such as the battery level, is still missing from the *SenseGlove_Data*, because it has not been implemented on the firmware yet.

More (unconverted) data is also available in its raw form through the *SenseGlove_Object.GetGloveData()* function, which returns a *GloveData* object.

Should you come across an interesting data point that is missing from the *SenseGlove_Data*, be sure to let us know so we can incorporate it into the SDK.

Creating your own hand model

The SenseGlove_Wireframe model will suffice for testing purposes, but eventually you might want to control you own hand model with the SenseGlove. To do this, you only require an active *SenseGlove_Object* script in your scene. Your model must have reference to this script in order to access its *SenseGlove_Data*. You can use the *SenseGlove_WireFrame* script as an example on how to access the data form a *SenseGlove_Object* and how to use it in you model.

As mentioned before, the SenseGlove needs a few milliseconds after connecting to gather glove-specific data from the device. If your model depends on the *GloveData* for initialization, do not use Unity's *Start()* method. Instead, you should subscribe to the *SenseGlove_Object.OnGloveLoaded* event, which fires just after all *GloveData* has been loaded from the SenseGlove, and the kinematic model is ready. If you are only interested in the hand model, this information does not apply.

With the variables in the *SenseGlove_Data* object, it is possible to create a fully articulated hand model. The *handAngles* and *relativeWrist* , combined with the *isRight* property, are the most important ones to achieve this.

As you can read in the "Working with Glove Data" section, the SenseGlove's *handRotations* are given relative to a common origin, and not to the previous joint. However, most available hand models for Unity use a tree structure, where a joint's rotation must be relative to the previous one. For these type of hand models, one should use the *handAngles* variable, which can easily be converted into quaternions using Unity's *Quaternion.Euler(Vector3)* method.

If your model is not relying on *handRotations* or *handPositions*, you can save yourself calculation time by setting your *SenseGlove_Object.updateTo* variable to *UpdateLevel.HandAngles*. Doing so will still give you access to the latest *handAngles*. You can find out how to do this in the SenseGlove_Object section.

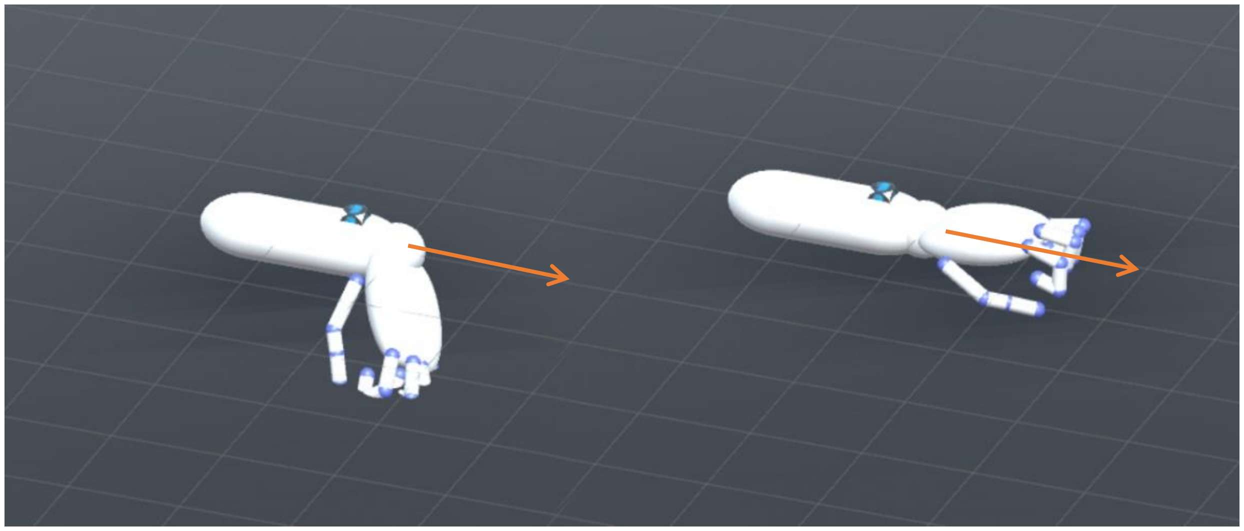
You could choose to let your model re-size its finger lengths to match that of the user, after they complete the calibration of their fingers. Doing so increases the accuracy of the on-screen model. You can subscribe to the *SenseGlove_Object.OnCalibrationFinished* event, which fires just after the new finger lengths are determined. It comes with finger calibration arguments that include the new finger lengths and new joint positions.

Calibration

To get the most out of your SenseGlove, you calibrate its mathematical model to use your finger lengths and joint positions.

Wrist

The Inertial Measurement Unit inside the SenseGlove knows nothing about the virtual world. The wrist model calibration tells the SenseGlove where its forearm is. After calibration, all wrist movement is performed relative to this forearm.



The wrist calibration can be called via the *SenseGlove_Object.CalibrateWrist()* function. This function automatically uses the orientation of the *SenseGlove_Object's* foreArm, if available. You can also pass a Quaternion rotation to use as a forearm instead.

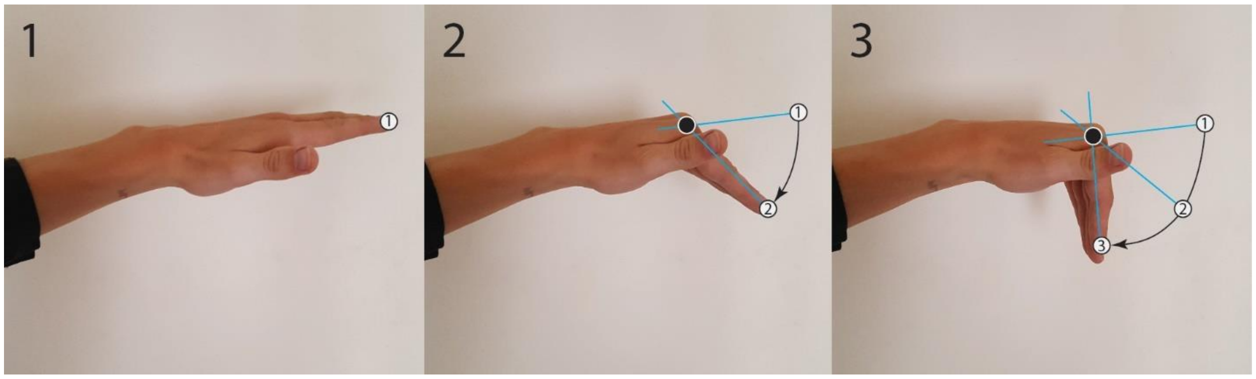
This calibration is called automatically when connecting to a SenseGlove. Of course, if you are not using the wrist model, or you have a tracker mounted directly onto your SenseGlove, this calibration step is not required.

Fingers

The SenseGlove needs to know its wearer's finger lengths and -starting positions in order to calculate the proper joint angles. When a new SenseGlove object is created, a default hand model is used.

The SenseGlove team has developed an algorithm to determine the wearer's finger lengths and MCP joint positions, based on three positions shown below, which is wrapped in the *SenseGlove_Object.NextCalibrationStep()* method.

The algorithm is based on 'determining the radius and origin of a circle, based on 3 points'. In this situation, the MCP joint is the origin, and the total finger length is the radius. By multiplying the total finger length with a set of ratio's, it determines the individual lengths. For this algorithm, any three points will do in any order, but the further apart the three calibration points are, the more accurate the result.



The first time the *NextCalibrationStep()* is called, no position is stored, but a (debug) message is printed.

By calling the *NextCalibrationStep()* method three more times for each of the steps, you can complete the calibration, receiving new instructions with each step. Once the last calibration step completes, the *SenseGlove_Object* fires the *OnCalibrationFinished* event, which can be used to rescale your hand model.

Creating your own calibration

You might decide to try your hand at making your own calibration algorithm, or maybe you want to load the last calculated finger lengths. The *SenseGlove_Object* offers two wrapper functions to interact with the finger lengths used in the DLL: The *GetFingerLengths()* and *SetFingerLengths()* methods.

The *GetFingerLengths()* method gives you a 5x3 array containing the lengths of each finger segment of each finger, in mm. The *SetFingerLengths()* method takes the same 5x3 array as an input, and updates the finger lengths that are used in the DLL. The latter method should be used when your custom calibration algorithm finishes, so that the mathematical model has access to the latest finger lengths.

If you're also working with the MCP or CMC joint positions, you can update these using the *GetStartJointPositions()* and *SetStartJointPositions()* method. These work with another 5x3 method containing the x,y,z positions of the first joints or each finger, relative to the common origin.

Force Feedback

The SenseGlove achieves its force-feedback by braking its glove sections in the grasping direction.

The force feedback is not yet fully integrated into the DLL at this revision of the Documentation. However, we can reveal several points which will prepare developers for its implementation:

1. There is a limit to how many motor commands the SenseGlove can process. After all, it needs time to send back its sensor data. The DLL will guard against an overflow of commands.

2. You can expect to send your force-feedback commands to the SenseGlove via a wrapper function in the *SenseGlove_Object* script; "SendForceFeedbackCommands()".

3. This method will allow you to specify the type of response (step/ramp/pulse), its intensity, and its duration.

4. The DLL will verify if the SenseGlove you are trying to reach has force-feedback capabilities. If not, the command is not sent. This means you *can* call the function without having to check if your SenseGlove actually has this capability.

5. You will be able to check your SenseGlove's capabilities through a *HasFunction()* method of the *SenseGlove_Object* script, which will take a *GloveFunctions* enumerator as input.

    a. For example " if (mySenseGlove.HasFunction(GloveFunction.ForceFeedback)) {   } "

F.A.Q.

If you don't find your question here, contact the SenseGlove team.

Hardware Troubleshooting

Q: My SenseGlove keeps disconnecting, then connecting again.

A: The SenseGlove is trying to initialize its Inertial Measurement Unit; the sensor that measures wrist movement. This chip has trouble 'waking up' sometimes, so we have to keep resetting it until it does. The glove should sort itself out in a few seconds, but if you cannot stand the connection sound, try unplugging the glove, then plug it back in.

Q: My virtual fingers are moving erratically or are spasming.

A: Unfortunately, the SenseGlove prototypes see heavy use during their lifetime, and it is possible that some of the sensors have been worn out. Please contact the SenseGlove team to arrange a suitable replacement.

Q: One or more of my virtual finger joints are not moving.

A: Open the diagnostics scene and check the model of the SenseGlove. Try to move a few of the individual segments of your glove to test if they are moving as expected. If none of them are moving, please ensure the glove is still connected and is not resetting itself. If some of the glove segments are not moving, or are moving in an odd way, one of the sensors might be broken. Please contact the SenseGlove team to arrange a suitable replacement.

Q: The SenseGlove wrist orientation does not match that of my hand.

A: Make sure you have calibrated the wrist using the *CalibrateWrist()* function of the *SenseGlove_Object*.

Not all Inertial Measurement Units are positioned the same way inside each SenseGlove. Prototypes 1 to 11 make use of a software based correction contained within the *SenseGlove_Object* script to compensate. If you are using prototype 1 to 11, this issue is fixed by updating your Unity Package to at least version 0.7, which contains all of the hardware corrections.

Prototypes 12 and onward will have this hardware correction built into their firmware. If you are using one of these gloves and the wrist orientation does not match, contact the SenseGlove Team.

Q: How fast does the SenseGlove update its variables?

A: The SenseGlove sends its latest sensor data every 10ms, (100 times per second or 100Hz). However, it only performs calculations when the *Update()* function is called, so the actual update rate is dependent on your framerate. In doing so, slower computers will automatically run fewer calculations, while faster computers will run them more often. It is possible to give the SenseGlove a fixed update rate by updating your *SenseGlove_Object* in Unity's *FixedUpdate()* instead.

Both the HTC Vive and Occulus Rift use refresh rates of up to 90Hz, which means that the SenseGlove, with its 100Hz, will always have access to 'fresh' sensor data when the *Update()* function is called.

Q: I don't feel any force feedback. Is my SenseGlove broken?

A: Your SenseGlove is fine. The first prototypes do not have force feedback yet.

Software Development

Q: Why do I have to wait before the DeviceScanner has identified my SenseGlove(s)? Can you not just give me a list of all connected SenseGloves?

A: Because in Unity, we can only access a list of all connected Serial Port addresses, nothing more.

Certain other devices, such as Arduino boards, WIFI-dongles or USB-monitors, also work via Serial Ports and will show up in this list. To ensure we are not trying to connect to your monitor, we send an identification request to every connected device, which takes the SenseGlove a few milliseconds to respond to, hence the real-time delay.

Q: Why does the glove take a few milliseconds to get 'ready'?

A: Because each glove has different size(s), certain hardware-related variables are actually stored on the device itself. Until we know these variables, it is not possible to perform kinematic calculations.

After identification, we load these variables, such as the lengths of each glove segment, from the device which, again, takes a few milliseconds to complete.

If you require glove-related variables for initialization, you can subscribe to the *OnGloveLoaded* event of the *SenseGlove_Object* Script for your setup methods.

Q: Why is the keyword 'this' used so frequently in the SenseGlove code?

A: (Unity) scripts contain a lot of global variables (class attributes), which are used together with local or temporary variables. We use the *'this'* keyword to make a clear distinction between a global variable and local variables.

Q: Why are the SenseGlove's Quaternion rotations all relative to a common origin?

A: It makes it easier to calculate the inverse kinematics of the hand. If you require a quaternion rotation relative to the previous joint, use the *handAngles* variable instead. This array contains the output from the inverse kinematics: Euler angles relative to the previous joint. These can then be converted into a quaternion using the Quaternion.Euler(Vector3) method.

Contact

If you have an issue that is not covered in this documentation, you can send a message to the Sense Glove Team via your dedicated slack channel, or send an e-mail to max@senseglove.com.

You can find the latest update of the SenseGlove Unity SDK on our Github: https://github.com/Adjuvo/SenseGlove-Unity, where you can also find more in-depth code documentation.