VMG30 SDK
User Manual
Version 1.2.0

25 August, 2015

Index

SDK Description

The VMG30 Software Development Kit allows programmer to quickly interface the VMG30 Dataglove inside they projects. The SDK is written in C++ and compiled with Visual Studio 2010.

SDK Contents:

VMG30_SDK.dll -> dynamic linked library (to be copied in your distribution);

VMG30_SDK.lib -> staticalliy linked library (to be linked in your project);

VMG30.h -> header file declaring the VMG30 class;

VMG30_SDK.pdf -> this manual;

TEST_SDK.zip -> zip file containing a demo project, showing basic SDK calls;

Requirements:

VMG30 3.0 dataglove (USB or Wifi version) with firmware 1.2.3 or later;

Microsoft Windowx XP or later;

Microsoft Visual C++ 2010 (also Express Edition), or later editions;

Middle C++ programming skills;

Compiling with the SDK

In order to use the SDK inside a Visual C++ project the following step must be done:

- copy the SDK in a folder (in example C:\VMG30_SDK\ ), in the following called “Installation Path”

- open the existing Visual Studio C++ project or create a new one (Win32 Project)

- add the VMG30 “Installation Path” to Project->Properties->C/C++>General->Additional Include Directory

- add the VMG30 “Installation Path” to Project->Properties->Linker->General->Addition Library Directories;

- add **VMG30_SDK.lib** to Project->Properties->Linker->Input->Additional Dependencies:

- add **#include “VMG30.h”** into the fikles where you will use the VMG30 class;

If the previous steps have been made correctly you will be now ready to use the VMG30 SDK inside your project.

VMG30 SDK Methods

By Including the VMG30.h header file some methods will be ready to use in order to manage the communication with the VMG30 dataglove.

VMG30 methods description:

Initialization and settings

Get a new VMG30 Handle:

VMG30HANDLE GetVMGlove();
Return an handle to the newly allocated dataglove; It must be called before any other method call.

Set Connection Parameters

int VMGGloveSetConnPar(VMG30HANDLE vmgh, int comport, char *ip)
This method initialize the connection parameters used by the dataglove.

Parameters*:*
vmgh (in): handle to a previously created dataglove;
comport (in): COMPort created by the dataglove once the USB cable is connected;
ip (in): IP Address of the Wifi module (use VMG303Manager to discover dataglove IP address), this value applies only for Wifi models, otherwise use “127.0.0.1”

Return Value
GLOVE_OK

Get Connection Parameters

int VMGloveGetConnPar(VMG30HANDLE vmgh, int *comport, char *ip)
This method returns the connection parameters set by the SetConnectionParameters method.

Parameters*:*
vmgh (in): handle to a previously created dataglove;
comport (out): COMPort
ip (out): IP Address of the Wifi module (use VMG303Manager to discover dataglove IP address), this value applies only for Wifi models, otherwise use “127.0.0.1”

Return Value
GLOVE_OK

Start connection

int VMGloveConnect(VMG30HANDLE vmgh, int ConnMethod, int StMode)
This method starts the connection with the dataglove. The connection can be viwa USB or via Wifi (if supported by the device)

Parameters*:*

vmgh (in): handle to a previously created dataglove;
ConnMethod (in): connection type:

```
CONN_USB will start a USB connection;
CONN_WIFI will start a Wifi connection;
```

StMode (in): determines the package type sent by the dataglove, it can be one of the following values:

```
STREAM__QUATERNION -> dataglove will stream quaternion values (reporting   the
attitude information of the hand) and the finger bending information (from
0.0 to 1000.0)
STREAM_RAW -> dataglove will stream fingers bending information and raw    data
reporting instantaneour gyroscopes, accelerometer and magnetometer  measures;
```

Return Value

GLOVE_OK if the connection started correctly
GLOVE_ERROR_STREAMING if the dataglove streaming is already running;
GLOVE_ERROR_NO_CONN if it is not possible to start streaming thread;

Stop Connection

int VMGloveDisconnect(VMG30HANDLE vmgh)
This method stops the dataglove streaming;

Get Connection Method

int VMG30GetConnectionMethod(VMG30HANDLE vmgh, int *connmethod)
This method reports the setted connection method

Parameters*:*

vmgh (in): handle to a previously created dataglove;
ConnMethod (out): setted connection type:

```
CONN_USB:_ USB connection setted;
CONN_WIFI: Wifi connection setted;
```

Return Value

GLOVE_OK

Get Streaming mode

int VMGloveGetStreamMode(VMG30HANDLE vmgh, int *strmode)
Reports the setted streaming mode.

Parameters*:*

vmgh (in): handle to a previously created dataglove;

strmode (out): setted streaming mode:

```
STREAM_QUATERNION -> dataglove will stream quaternion values (reporting    the
attitude information of the hand) and the finger bending information (from
0.0 to 100.0)
STREAM_RAW -> dataglove will stream fingers bending information and raw    data
reporting instantaneour gyroscopes, accelerometer and magnetometer  measures;
```

Return Value

GLOVE_OK

Get Connection Status

int VMGloveGetConnectionStatus(VMG30HANDLE vmgh)
This method reports the dataglove connection status;

Parameters

vmgh (in): handle to a previously created dataglove;

Return Value

NOT_CONNECTED: the dataglove streaming is disabled;
USB_CONNECTED: the dataglove is streaming over the USB and values can be read;
WIFI_CONNECTED: the dataglove is streaming over the Wifi and values can be read;

Dataglove firmware version

int VMGloveGetFWVersion(VMG30HANDLE vmgh, int *fw1, int *fw2, int *fw3)
This methos reports the running firmware version. At least a 1.2.3 version is required by the SDK to correctly work.

Parameters*:*

vmgh (in): handle to a previously created dataglove;
fw1 (out): major version
fw2 (out): middle version
fw3 (out): minor version

Return Value
GLOVE_OK

Dataglove Settings

Dataglove information

int VMGloveGetID(VMG30HANDLE vmgh, char *label, int *id)
This methos reports the stored label and ID of the dataglove. This methos should be called only once a connection has been made.

Parameters*:*
vmgh (in): handle to a previously created dataglove;
label (out): stored dataglove label;
id (out): stored dataglove ID;

Return Value
GLOVE_OK

---

int VMGloveSetID(VMG30HANDLE vmgh, char *label, int id)
This methos se the stored label and ID of the dataglove. This methos should be called only once a connection has been made.

Parameters*:*
vmgh (in): handle to a previously created dataglove;
label (in): stored dataglove label;
id (in): stored dataglove ID;

Return Value
GLOVE_OK

USB Settings

int VMGloveGetCOMSettings(VMG30HANDLE vmgh, int *comport)
This methos reports the used comport for USB communication.

Parameters*:*
vmgh (in): handle to a previously created dataglove;
comport (out): comport used by the dataglove for the USB communication;

Return Value
GLOVE_OK

Wifi Settings

int VMGloveGetWiFiSettings(VMG30HANDLE vmgh, char *ip, char *netmask, char *gateway, int *dhcp)
This methos reports the Wifi configuration of the dataglove, it must be called only once at least a connection has been made.

Parameters*:*
vmgh (in): handle to a previously created dataglove;
ip (out): stored IP address;
netmask (out): stored Netmask;
gateway (out): stored Gateway
dhcp (out): stored DHCP mode (0 = OFF, 1 = ON);

Return Value
GLOVE_OK

---

int VMGloveGetAPNSettings(VMG30HANDLE vmgh, char *ssid, char *password)
This methos reports the Wifi access point configuration of the dataglove, it must be called only once at least a connection has been made.

Parameters*:*
vmgh (in): handle to a previously created dataglove;
ssid (out): stored SSID name;
password (out): stored Access Point password;

Return Value
GLOVE_OK

---

int VMGloveStoreSettings(VMG30HANDLE vmgh)
This method store the new WiFi settings in the internal memory storage and reboot the WiFi.

Parameters*:*
vmgh (in): handle to a previously created dataglove;

Return Value
GLOVE_OK

int VMGloveSetWiFiSettings(VMG30HANDLE vmgh, char *ip, char *netmask, char *gateway, int dhcp)
This methos sets the Wifi configuration of the dataglove, **it must be called only when the dataglove is not connected. This operation requires an USB connection to be done**.

Parameters*:*

vmgh (in): handle to a previously created dataglove;
ip (out): new IP address (if dhcp = 0);
netmask (in): new Netmask (if dhcp = 0);
gateway (in): new Gateway (if dhcp = 0)
dhcp (in): new DHCP mode (0 = OFF, 1 = ON);

Return Value

```
GLOVE_OK if parameters succesfully updated
GLOVE_ERROR: if parameters not correctly updated
```

---

int VMGloveSetAPNSettings(VMG30HANDLE vmgh, char *ssid, char *password)
This methos sets the Wifi access point configuration of the dataglove. **It must be called when the dataglove is not connected. This operation requires an USB connection to be done**.

Parameters*:*

vmgh (in): handle to a previously created dataglove;
ssid (in): new SSID name;
password (in): new Access Point password;

Return Value

```
GLOVE_OK if parameters succesfully updated
GLOVE_ERROR: if parameters not correctly updated
```

Dataglove sensors values:

Fingers bending values

int VMGloveGetFingers(VMG30HANDLE vmgh, double *fingers)
Returns the fingers bending values. An array of 10 values must be previously allocated.
A value of 0 report minimum bending, a value of 1000.0 reports maximum bending.

Parameters*:*

vmgh (in): handle to a previously created dataglove;
fingers (out): array reporting the bendign values of the fingers. An array of 10 double must be previously allocated:

```
double fingers[10];
glove->GetFingers(fingers);
```

The fingers are ordered from Thumb to Little, i.e. fingers[0] and fingers[1] contain the upper finger bending and bottom finger bending of the thumb.

Return Value

```
GLOVE_OK if streaming is running
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (fingers data niot available)
```

Abductions values

int VMG30::GetAbductions(VMG30HANDLE vmgh, double *abd)
Returns the in fingers abduction values. An array of 4 values must be previously allocated.
A value of 0 reports maximum abduction, a value of 1000.0 reports minimum abduction.

Parameters*:*

vmgh (in): handle to a previously created dataglove;
abd (out): array reporting the abduction values between adjacent fingers, starting from thumb. An array of 4 double must be previously allocated:

```
double abd[10];
glove->GetAbductions(abd);
```

The abductions are ordered from Thumb to Little.

Return Value

```
GLOVE_OK if streaming is running
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (fingers data niot available)
```

Palm arch and thumb cross over

int VMGloveGetPalmArch(VMG30HANDLE vmgh, double *palmarch)
Returns the palm arch measurement in raw values. A 0 reports palm not bended, 1000.0 reports maximum palm bending.

Parameters*:*
vmgh (in): handle to a previously created dataglove;
palmarch (out): palm bending value;

Return Value
```
GLOVE_OK if streaming is running
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (fingers data niot available)
```

---

int VMGloveGetThumbCrossOver(VMG30HANDLE vmgh, double *thumb)
Returns the thumb position.

Parameters*:*
vmgh (in): handle to a previously created dataglove;
thumb (out): thumb cross over position

Return Value
```
GLOVE_OK if streaming is running
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (fingers data niot available)
```

Dataglove hand and wrist attitude and IMUs raw values

int VMGloveGetAttitudeHand(VMG30HANDLE vmgh, double *roll, double *pitch, double *yaw)
Returns the attitude values of the hand.

Parameters*:*
vmgh (in): handle to a previously created dataglove;
roll (out): Roll value in degree;
pitch (out): Pitch value in degree;
yaw (out): Yaw value in degree;

Return Value
```
GLOVE_OK if streaming is running
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (quaternion vector not available)
```

---

int VMGloveGetQuaternionHand(VMG30HANDLE vmgh, double *quat1, double *quat2,
double *quat3, double *quat4)
Returns the raw quaternion vector of the Hand attitude;

Parameters*:*
vmgh (in): handle to a previously created dataglove;
quat1,quat2,quat3,quat4 (out): quaternion vector;

Return Value
```
GLOVE_OK if streaming is running and value are valid;
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (quaternion vector not available)
```

---

int VMGloveGetAttitudeWrist(VMG30HANDLE vmgh, double *roll, double *pitch, double *yaw)
Returns the attitude values of the wrist.

Parameters*:*
vmgh (in): handle to a previously created dataglove;
roll (out): Roll value in degree;
pitch (out): Pitch value in degree;
yaw (out): Yaw value in degree;

Return Value
```
GLOVE_OK if streaming is running
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (quaternion vector not available)
```

---

int VMGloveGetQuaternionWrist(VMG30HANDLE vmgh, double *quat1, double *quat2,
double *quat3, double *quat4)
Returns the raw quaternion vector of the Wrist;

Parameters*:*
vmgh (in): handle to a previously created dataglove;
quat1,quat2,quat3,quat4 (out): quaternion vector;

Return Value
```
GLOVE_OK if streaming is running and value are valid;
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (quaternion vector not available)
```

---

int VMGloveGetGyrosHand(VMG30HANDLE vmgh, double *gyro1, double *gyro2,
double *gyro3)
Returns the raw gyroscope information coming from the imu placed on the hand;

Parameters*:*
vmgh (in): handle to a previously created dataglove;
gyro1 (out): X axes gyro information;
gyro2 (out): Y axes gyro information;
gyro3 (out): Z axes gyro information;

Return Value
```
GLOVE_OK if streaming is running and value are valid;
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (raw data not available)
```

---

int VMGloveGetAccelsHand(VMG30HANDLE vmgh, double *accel1, double *accel2,
double *accel3)
Returns the raw accelerometer information of the hand;

Parameters*:*
vmgh (in): handle to a previously created dataglove;
accel1 (out): X axes accelerometer information;
accel2 (out): Y axes accelerometer information;
accel3 (out): Z axes accelerometer information;

Return Value
```
GLOVE_OK if streaming is running and value are valid;
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (raw data not available)
```

---

int VMGloveGetMagnsHand(VMG30HANDLE vmgh, double *accel1, double *accel2,
double *accel3)
Returns the raw magnetometer information coming from the imu hand;

Parameters*:*
vmgh (in): handle to a previously created dataglove;
magn1 (out): X axes magnetometer information;
magn2 (out): Y axes magnetometer information;
magn3 (out): Z axes magnetometer information;

Return Value
```
GLOVE_OK if streaming is running and value are valid;
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (raw data not available)
```

---

int VMGloveGetGyrosWrist(VMG30HANDLE vmgh, double *gyro1, double *gyro2,
double *gyro3)
Returns the raw gyroscope information coming from the imu placed on the wrist;

Parameters*:*
vmgh (in): handle to a previously created dataglove;
gyro1 (out): X axes gyro information;
gyro2 (out): Y axes gyro information;
gyro3 (out): Z axes gyro information;

Return Value
```
GLOVE_OK if streaming is running and value are valid;
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (raw data not available)
```

---

int VMGloveGetAccelsWrist(VMG30HANDLE vmgh, double *accel1, double *accel2,
double*accel3)
Returns the raw accelerometer information of the wrist;

Parameters*:*
vmgh (in): handle to a previously created dataglove;
accel1 (out): X axes accelerometer information;
accel2 (out): Y axes accelerometer information;
accel3 (out): Z axes accelerometer information;

Return Value
```
GLOVE_OK if streaming is running and value are valid;
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (raw data not available)
```

---

int VMGloveGetMagnsWrist(VMG30HANDLE vmgh, double *accel1, double *accel2,
double *accel3)
Returns the raw magnetometer information coming from the imu on the wrist;

Parameters*:*

vmgh (in): handle to a previously created dataglove;
magn1 (out): X axes magnetometer information;
magn2 (out): Y axes magnetometer information;
magn3 (out): Z axes magnetometer information;

Return Value

```
GLOVE_OK if streaming is running and value are valid;
GLOVE_ERROR_NO_CONN if the streaming is not running
GLOVE_ERROR_NO_DATA if the StreamMode is not correct (raw data not available)
```

Additional Information and Operations

unsigned int VMGloveGetLastPackageTime(**VMG30HANDLE** vmgh)
Returns the last snaptime of the package creation in milliseconds, computed from the device startup.

Parameters*:*
vmgh (in): handle to a previously created dataglove;

Return Value
`Instant time of the last package sent in milliseconds;`

Module Turn OFF

int VMGloveTurnOFF(VMG30HANDLE vmgh, int ConnMethod)
Turn Off the module

Parameters*:*
vmgh (in): handle to a previously created dataglove;
ConnMethod (communication methofd:
USB_CONN: turn off the device via USB
WIFI_CONN: turn off the device via Wifi;

Return Value
`GLOVE_OK, operation succesfully executed;`
`GLOVE_ERROR_STREAMING: sdataglove is strraming, stop the streaming and then turn off the module with this method;`

Dataglove node positioning

Set glove hand type

int VMGloveSetLeftHanded(VMG30HANDLE vmgh, int lefthanded)

Sets right/left handed dataglove. If lefthanded = 1 then left handed dataglove, if lefthanded = 0 then righthanded dataglove; By defualt the dataglove is represented as a right handed one.

Parameters*:*

vmgh (in): handle to a previously created dataglove;
lefthanded (int): dataglove hand type

Return Value

`GLOVE_OK if streaming is running and value are valid;`

Nodes positions

int VMGloveGetPosition(VMG30HANDLE vmgh, char *nodename, double *x, double *y, double *z)

Returns the absolute position of the node “nodename” with respèect to the root.

Parameters*:*

vmgh (in): handle to a previously created dataglove;
nodename (in): name of the node (please refer to the list of name at the end of the document)
x(out): X position of the node;
y (out): Y position of the node;
z (out): Z position of the node;

Return Value

`GLOVE_OK if streaming is running and value are valid;`
`GLOVE_NODE_ERROR if node does not exists`

Bones lenghts

int VMGloveSetBoneLenght(VMG30HANDLE vmgh, char *nodename, double l)

Sets the lenght of the bone connecting the node “nodename” with its parent.

Parameters*:*

vmgh (in): handle to a previously created dataglove;
nodename (in): name of the node (please refer to the list of name at the end of the document)
l (in): lenght of the bone in mm

Return Value

```
GLOVE_OK
GLOVE_NODE_ERROR if node does not exists
```

---

int VMGloveGetBoneLenght(VMG30HANDLE vmgh, char *nodename, double *l)

Return the lenght of the bone connecting the node “nodename” with its parent.

Parameters*:*

vmgh (in): handle to a previously created dataglove;
nodename (in): name of the node (please refer to the list of name at the end of the document)
l (out): lenght of the bone in mm

Return Value

```
GLOVE_OK if streaming is running and value are valid;
GLOVE_NODE_ERROR if node does not exists
```

Code Example - USB Communication

```
#include "stdafx.h"
#include <stdio.h>
#include <string>
#include <windows.h>
#include "wxVTK.h"		//rendering class

//include dataglove class
#include "VMG30.h"

//arg[1] = comport, arg[2] = ipaddress
int main(int argc, char* argv[])
{
	//init rendering and skeleton structure structure
	wxVTK *vtk = new wxVTK();

	//create a new dataglove instance
	VMG30HANDLE gloveH = GetVMGlove();

	//uncomment this line to set left handed dataglove
	//VMGloveSetLeftHanded(gloveH,1);

	//get comport and ipaddress
	char ipaddr[256];sprintf(ipaddr,"127.0.0.1");
	int comport = 1;
	if (argc>1) comport = atoi((char *)argv[1]);
	if (argc>2) sprintf(ipaddr,"%s",(char *)argv[2]);
	VMGloveSetConnPar(gloveH,comport,ipaddr);

	//check conn parameters
	int comp;
	char ip[VHAND_STRLEN];
	VMGloveGetConnPar(gloveH,&comp,ip);
	fprintf(stderr,"COMP:%d IP:%s\n",comp,ip);

startconn:
	//connecto to the dataglove
	VMGloveConnect(gloveH,CONN_USB,PKG_QUAT_FINGER);

	int cnt = 0;
	long start = ::GetTickCount();

	//1 minute sampling
	while ((::GetTickCount()-start)<60000){
		//get the connection status
		int cs = VMGloveGetConnectionStatus(gloveH);
		Sleep(50);
		fprintf(stderr,"CONNSTATUS: %d\n",cs);
		if (cs == USB_CONNECTED){
			//get wrist attitudwe
			double roll, pitch, yaw;
			VMGloveGetAttitudeWrist(gloveH,&roll,&pitch,&yaw);
			fprintf(stderr,"WRIST ROLL: %.1f PITCH:%.1f YAW:%.1f\n",roll,pitch,yaw);
			//get hand attitude
			VMGloveGetAttitudeHand(gloveH,&roll,&pitch,&yaw);
			fprintf(stderr,"HAND ROLL: %.1f PITCH:%.1f YAW:%.1f\n",roll,pitch,yaw);
			//get finger bending
```

```
double fingers[10];
VMGloveGetFingers(gloveH,fingers);
int i = 0;
for (i=0;i<5;i++){
        fprintf(stderr,"FINGER%d: %.1f %.1f\n",i+1,fingers[2*i],fingers[2*i+1]);
}
//get abdution sensors values
double abds[4];
VMGloveGetAbductions(gloveH,abds);
for (i=0;i<4;i++){
        fprintf(stderr,"ABD%d: %.1f\n",i+1,abds[i]);
}
//get pressure sensors status
double press[10];
VMGloveGetPressures(gloveH,press);
for (i=0;i<5;i++){
        fprintf(stderr,"PRESS%d: %.1f\n",i+1,press[i]);
}
//get palm arch and thumbco values
double palmarch = 0.0f, thumbco = 0.0f;
VMGloveGetPalmArch(gloveH,&palmarch);
VMGloveGetThumbCrossOver(gloveH,&thumbco);
fprintf(stderr,"PALMARCH: %.2f THUMBCO:%.2f\n",palmarch,thumbco);

//update rendering, getting nodes positions  from the sdk
double x,y,z;
VMGloveGetPosition(gloveH,"WRIST",&x,&y,&z);
vtk->SetNodePosition("WRIST",x,y,z);
VMGloveGetPosition(gloveH,"HAND",&x,&y,&z);
vtk->SetNodePosition("HAND",x,y,z);

VMGloveGetPosition(gloveH,"THUMB0",&x,&y,&z);
vtk->SetNodePosition("THUMB0",x,y,z);
VMGloveGetPosition(gloveH,"THUMB1",&x,&y,&z);
vtk->SetNodePosition("THUMB1",x,y,z);
VMGloveGetPosition(gloveH,"THUMB2",&x,&y,&z);
vtk->SetNodePosition("THUMB2",x,y,z);
VMGloveGetPosition(gloveH,"THUMB3",&x,&y,&z);
vtk->SetNodePosition("THUMB3",x,y,z);

VMGloveGetPosition(gloveH,"INDEX0",&x,&y,&z);
vtk->SetNodePosition("INDEX0",x,y,z);
VMGloveGetPosition(gloveH,"INDEX1",&x,&y,&z);
vtk->SetNodePosition("INDEX1",x,y,z);
VMGloveGetPosition(gloveH,"INDEX2",&x,&y,&z);
vtk->SetNodePosition("INDEX2",x,y,z);
VMGloveGetPosition(gloveH,"INDEX3",&x,&y,&z);
vtk->SetNodePosition("INDEX3",x,y,z);

VMGloveGetPosition(gloveH,"MIDDLE0",&x,&y,&z);
vtk->SetNodePosition("MIDDLE0",x,y,z);
VMGloveGetPosition(gloveH,"MIDDLE1",&x,&y,&z);
vtk->SetNodePosition("MIDDLE1",x,y,z);
VMGloveGetPosition(gloveH,"MIDDLE2",&x,&y,&z);
vtk->SetNodePosition("MIDDLE2",x,y,z);
VMGloveGetPosition(gloveH,"MIDDLE3",&x,&y,&z);
vtk->SetNodePosition("MIDDLE3",x,y,z);

VMGloveGetPosition(gloveH,"RING0",&x,&y,&z);
```

```
                    vtk->SetNodePosition("RING0",x,y,z);
                    VMGloveGetPosition(gloveH,"RING1",&x,&y,&z);
                    vtk->SetNodePosition("RING1",x,y,z);
                    VMGloveGetPosition(gloveH,"RING2",&x,&y,&z);
                    vtk->SetNodePosition("RING2",x,y,z);
                    VMGloveGetPosition(gloveH,"RING3",&x,&y,&z);
                    vtk->SetNodePosition("RING3",x,y,z);

                    VMGloveGetPosition(gloveH,"LITTLE0",&x,&y,&z);
                    vtk->SetNodePosition("LITTLE0",x,y,z);
                    VMGloveGetPosition(gloveH,"LITTLE1",&x,&y,&z);
                    vtk->SetNodePosition("LITTLE1",x,y,z);
                    VMGloveGetPosition(gloveH,"LITTLE2",&x,&y,&z);
                    vtk->SetNodePosition("LITTLE2",x,y,z);
                    VMGloveGetPosition(gloveH,"LITTLE3",&x,&y,&z);
                    vtk->SetNodePosition("LITTLE3",x,y,z);

                    //update fingertips colour from pressure sensors (GREEN = no pressure, RED = MAX
                    PRESSURE)
                    vtk->SetNodeColour("THUMB3",t_Colour(255-255*press[0]/100.0,255*press[0]/100.0,0));
                    vtk->SetNodeColour("INDEX3",t_Colour(255-255*press[1]/100.0,255*press[1]/100.0,0));
                    vtk->SetNodeColour("MIDDLE3",t_Colour(255-255*press[2]/100.0,255*press[2]/100.0,0));
                    vtk->SetNodeColour("RING3",t_Colour(255-255*press[3]/100.0,255*press[3]/100.0,0));
                    vtk->SetNodeColour("LITTLE3",t_Colour(255-255*press[4]/100.0,255*press[4]/100.0,0));


                    vtk->Update();
            }
            cnt++;
}
VMGloveDisconnect(gloveH);

Sleep(1000);

goto startconn;
```

Node names

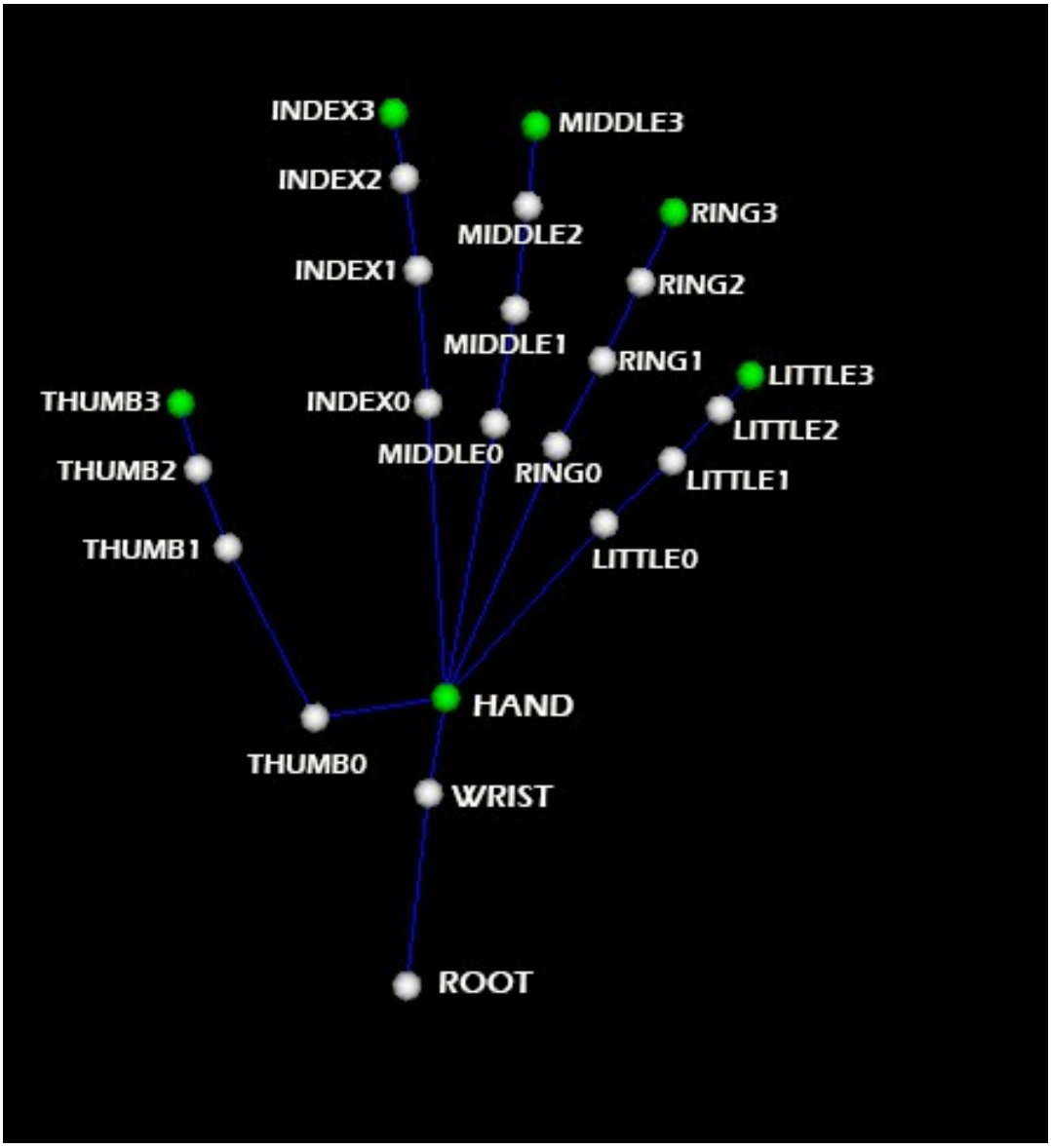

Illustrazione 1: Skeleton of the hand and nodes name

Node Name	Parent	Default Bone Length (mm)	Description
ROOT	None	0.0	Reference point
WRIST	ROOT	60.0	Position of the wrist
HAND	WRIST	30.0	Position of the hand
THUMB0	HAND	45.0	Position of thumb's first node
THUMB1	THUMB0	60.0	Position of thumb's second node
THUMB2	THUMB1	28.0	Position of thumb's third node
THUMB3	THUMB2	25.0	Position of thumb's fourth node
INDEX0	HAND	90.0	Position of index's first node
INDEX1	INDEX0	40.0	Position of index's second node
INDEX2	INDEX1	28.0	Position of index's third node
INDEX3	INDEX2	20.0	Position of index's fourth node

MIDDLE0	HAND	85.0	Position of middle's first node
MIDDLE1	MIDDLE0	35.0	Position of middle's second node
MIDDLE2	MIDDLE1	32.0	Position of middle's third node
MIDDLE3	MIDDLE2	25.0	Position of middle's fourth node
RING0	HAND	85.0	Position of ring's first node
RING1	RING0	30.0	Position of ring's second node
RING2	RING1	28.0	Position of ring's third node
RING3	RING2	25.0	Position of ring's fourth node
LITTLE0	HAND	75.0	Position of little's first node
LITTLE1	LITTLE0	30.0	Position of little's second node
LITTLE2	LITTLE1	25.0	Position of little's third node
LITTLE3	LITTLE2	17.0	Position of little's fourth node