

## Gesture control

Smart city laboratory 6.

### Intoduction

The measurement talks about a gesture controlled Webcam. There are an webcam which have IP based control API and a Leap Motion device is available. The Leap Motion device can recognize and track the user's fingers, hands, arms and it can identify some built in. During the measurement, you can learn programming the webcam control and Leap Motion device. To solve the task C++ sample programs are reachable.

### Leap motion [1][2][3]

The Leap Motion system recognizes and tracks hands and fingers. The device operates in an intimate proximity with high precision and tracking frame rate and reports discrete positions and motion.



The Leap Motion controller's view of your hands.

The Leap Motion controller uses optical sensors and infrared light. The sensors are directed along the y-axis – upward when the controller is in its standard operating position – and have a field of view of about 150 degrees. The effective range of the Leap Motion Controller extends from approximately 25 to 600 millimeters above the device (1 inch to 2 feet).

Detection and tracking work best when the controller has a clear, high-contrast view of an object's silhouette. The Leap Motion software combines its sensor data with an internal model of the human

hand to help cope with challenging tracking conditions.



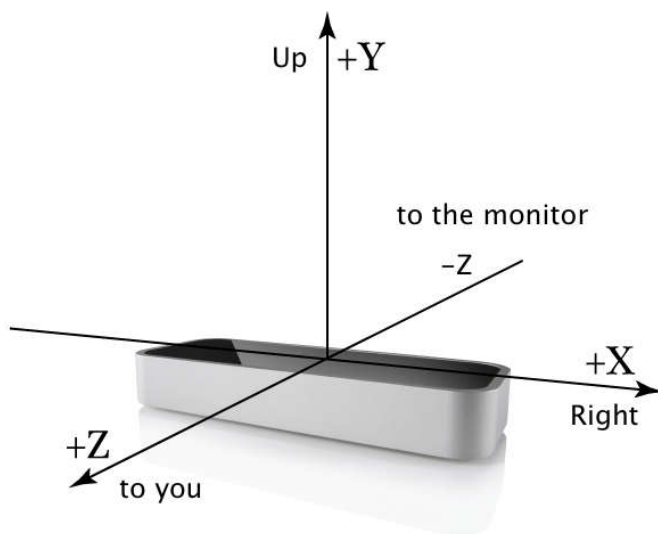
## Hardware

The data takes the form of a grayscale stereo image of the near-infrared light spectrum, separated into the left and right cameras. Typically, the only objects you'll see are those directly illuminated by the Leap Motion Controller's LEDs. However, incandescent light bulbs, halogens, and daylight will also light up the scene in infrared. You might also notice that certain things, like cotton shirts, can appear white even though they are dark in the visible spectrum.



Stereo image of cameras

## Coordinate system



The Leap Motion right-handed coordinate system

The Leap Motion system employs a right-handed Cartesian coordinate system. The origin is centered at the top of the Leap Motion Controller. The x- and z-axes lie in the horizontal plane, with the x-axis running parallel to the long edge of the device. The y-axis is vertical, with positive values increasing upwards (in contrast to the downward orientation of most computer graphics coordinate systems). The z-axis has positive values increasing toward the user.

The Leap Motion API measures physical quantities with the following units:

Distance: millimeters

Time: microseconds (unless otherwise noted)

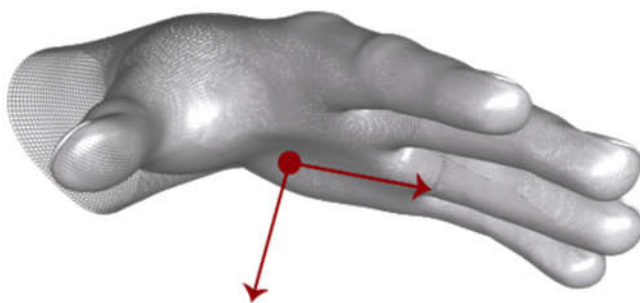
Speed: millimeters/second

Angle: radians

As the Leap Motion controller tracks hands and fingers in its field of view, it provides updates as a set – or frame – of data. Each Frame object representing a frame contains any tracked hands, detailing their properties at a single moment in time. The Frame object is essentially the root of the Leap Motion data model.

## Hands

The hand model provides information about the identity, position, and other characteristics of a detected hand, the arm to which the hand is attached, and lists of the fingers associated with the hand.



The `Hand` [palmNormal\(\)](#) and [direction\(\)](#) vectors define the orientation of the hand.

The Leap Motion software uses an internal model of a human hand to provide predictive tracking even when parts of a hand are not visible. The hand model always provides positions for five fingers, although tracking is optimal when the silhouette of a hand and all its fingers are clearly visible. The software uses the visible parts of the hand, its internal model, and past observations to calculate the most likely positions of the parts that are not currently visible.

## Arms

An Arm is a bone-like object that provides the orientation, length, width, and end points of an arm. When the elbow is not in view, the Leap Motion controller estimates its position based on past observations as well as typical human proportion.

## Fingers

The Leap Motion controller provides information about each finger on a hand. If all or part of a finger is not visible, the finger characteristics are estimated based on recent observations and the anatomical model of the hand. Fingers are identified by type name, i.e. *thumb*, *index*, *middle*, *ring*, and *pinky*.



Finger |Finger\_tipPosition|\_ and |Finger\_direction|\_ vectors provide the position of a finger tip and the general direction in which a finger is pointing.

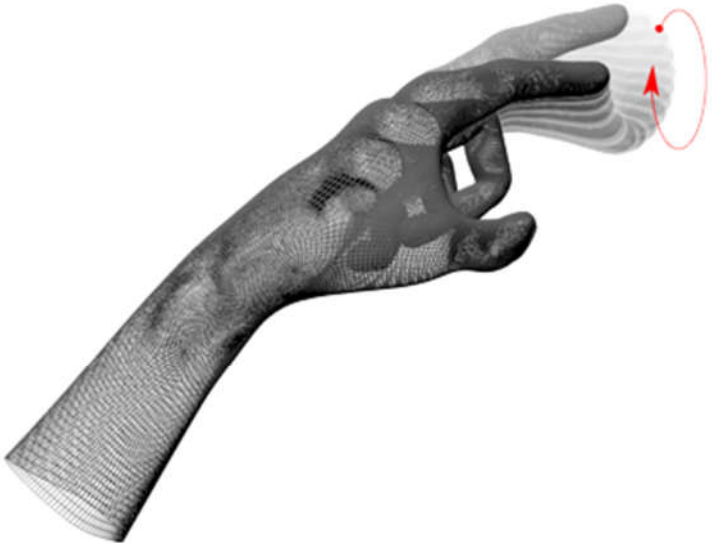


## Tool:

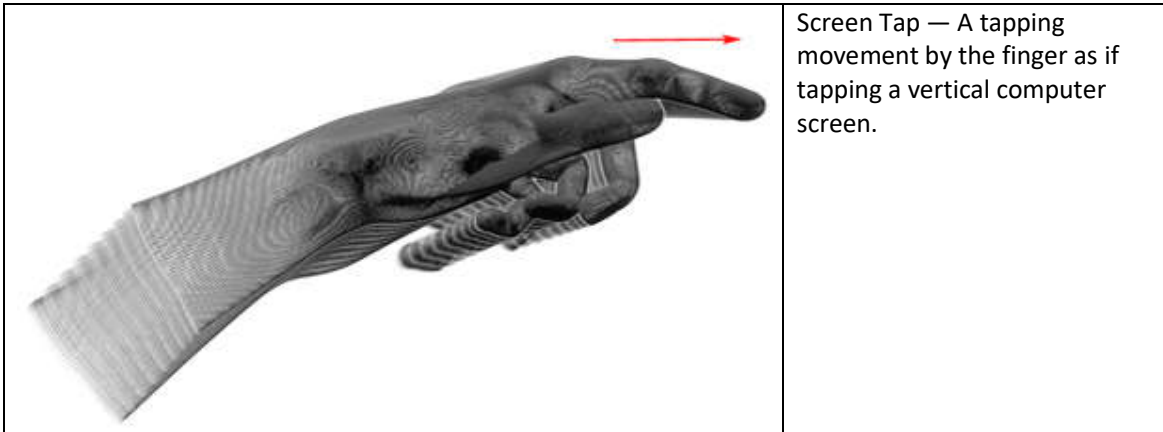
The [Tool](#) class represents a tracked tool.



## Gestures

The Leap Motion software recognizes certain movement patterns as gestures which could indicate a user intent or command. Gestures are observed for each finger or tool individually. The Leap Motion software reports gestures observed in a frame the in the same way that it reports other motion tracking data like fingers and hands.

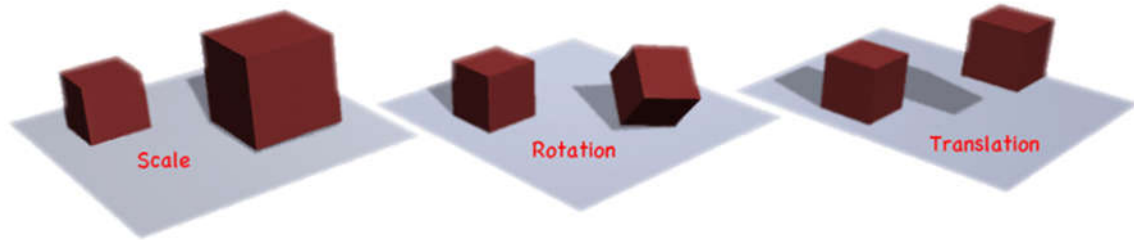
|   |   |
|---|---|
|    | <p>Circle — A finger tracing a circle.</p>                                    |
|   | <p>Swipe — A long, linear movement of a hand and its fingers.</p>             |
|  | <p>Key Tap — A tapping movement by a finger as if tapping a keyboard key.</p> |



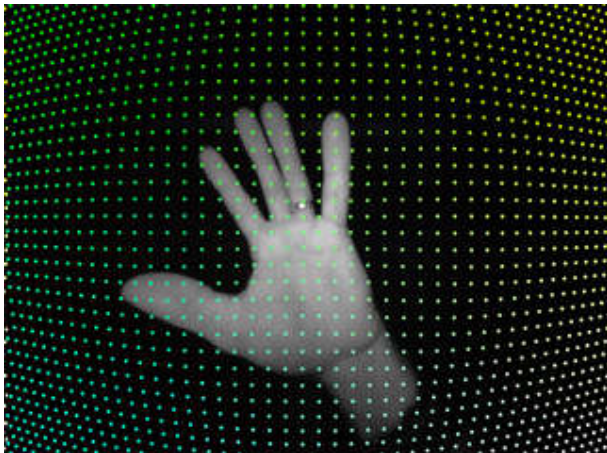
**Important:** before using gestures in your application, you must enable recognition for each gesture you intend to use. The Controller class has an *enableGesture()* method that you can use to enable recognition for the types of gestures you use.

### Motions:

Motions are estimates of the basic types of movements inherent in the change of a user's hands over a period of time. Motions include scale, rotation, and translation (change in position).



### Sensor Images:



Along with the computed tracking data, you can get the raw sensor images from the Leap Motion cameras.

## Programming of Leap motion

Supported programming languages and platforms (V2):

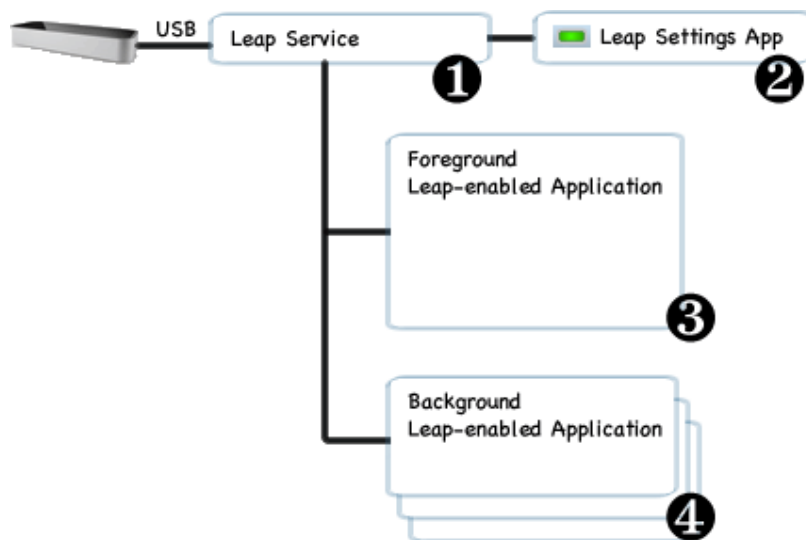


During the measurement we use C++.

### Leap Motion API:

#### Native Application Interface

The native application interface is provided through a dynamically loaded library. This library connects to the Leap Motion service and provides tracking data to your application:



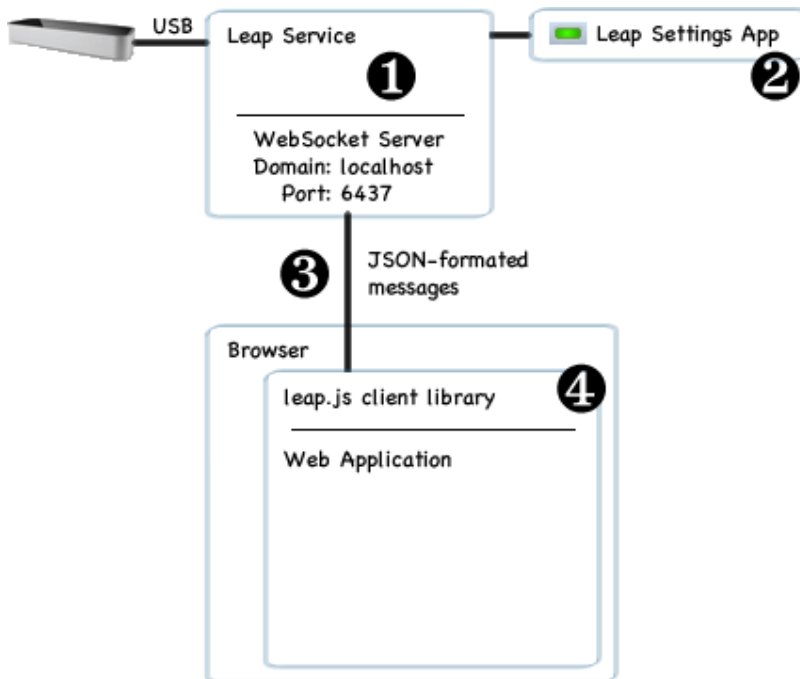
#### Leap-enabled applications

1. The Leap Motion service receives data from the Leap Motion Controller over the USB bus. It processes that information and sends it to running Leap-enabled applications. By default, the service only sends tracking data to the foreground application. However, applications can request that they receive data in the background (a request that can be denied by the user).
2. The Leap Motion application runs separately from the service and allows the computer user to configure their Leap Motion installation. The Leap Motion application is a Control Panel applet on Windows and a Menu Bar application on Mac OS X.
3. The foreground Leap-enabled application receives motion tracking data from the service. A Leap-enabled application can connect to the Leap Motion service using the Leap Motion native library. Your application can link against the Leap Motion native library either directly (C++ and Objective-C) or through one of the available language wrapper libraries (Java, C#, and Python).

4. When a Leap-enabled application loses the operating system focus, the Leap Motion service stops sending data to it. Applications intended to work in the background can request permission to receive data even when in the background. When in the background, the configuration settings are determined by the foreground application.

### WebSocket Interface

The Leap Motion service runs a WebSocket server on the localhost domain at port 6437. The WebSocket interface provides tracking data in the form of JSON messages



1. The Leap Motion service provides a WebSocket server listening on <http://127.0.0.1:6437>.
2. The Leap Motion control panel allows end users to enable or disable the WebSocket server.
3. The server sends tracking data in the form of JSON messages. An application can send configuration messages back to the server.
4. The `leap.js` client JavaScript library should be used in web applications. The library establishes the connection to the server and consumes the JSON messages. The API presented by the JavaScript library is similar in philosophy and structure to the native API.

### IP camera

type: FI9826W





Specification:

resolution: 1280 x 960 pixel max. 30 fps

Lens: f: 4-9mm, F1.8

Compression: H. 264 (video), PCM/G.726 (audio)

Moving: horizontal: 300°

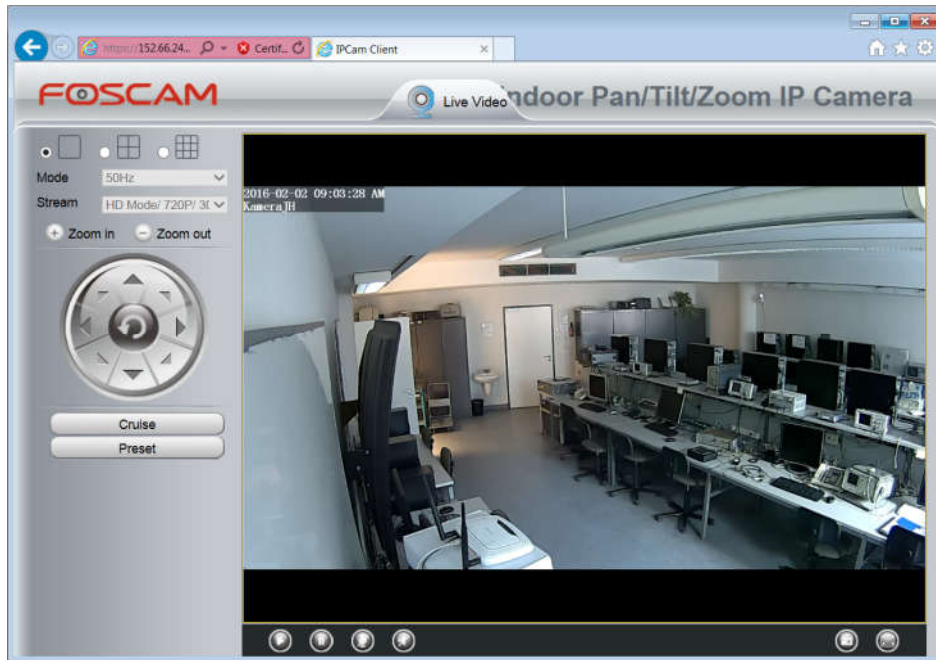
vertical: 120°

Control: Web based

Login screen:

A web-based login screen for the camera. It features a Username field with 'operator' entered, a Password field, a Stream dropdown menu set to 'Main stream', and a Language dropdown menu set to 'English'. A Login button is located at the bottom right of the form.

Control screen:



The camera is controlled by the arrows and “zoom in” and “zoom out”.

### RTSP connection [4]

The video stream of the camera is controlled via RTSP protocol too.

RTSP: Real Time Streaming Protocol

RTSP (RFC 2326) is a client-server multimedia presentation control protocol, designed to address the needs for efficient delivery of streamed multimedia over IP networks. The transmitting of video frequently uses RTP protocol. It is a packet format for multimedia data streams.

### Camera API:

The API is http based. The general format of a request is:

<http://w.x.y.z/cgi-bin/CGIProxy.fcgi&usr=username&pwd=password&cmd=ABC>

The XML result:

```
e.g.: <CGI_Result>
  <result>0</result>
  <resolution0>6</resolution0>
  <resolution1>1</resolution1>
  <resolution2>1</resolution2>
  <resolution3>1</resolution3>
  <bitRate0>4194304</bitRate0>
  <bitRate1>2097152</bitRate1>
  <bitRate2>1048576</bitRate2>
  <bitRate3>204800</bitRate3>
  <frameRate0>10</frameRate0>
  <frameRate1>25</frameRate1>
  <frameRate2>15</frameRate2>
  <frameRate3>10</frameRate3>
  <GOP0>10</GOP0>
  <GOP1>30</GOP1>
```

```

<GOP2>45</GOP2>
<GOP3>60</GOP3>
<isVBR0>1</isVBR0>
<isVBR1>1</isVBR1>
<isVBR2>0</isVBR2>
<isVBR3>0</isVBR3>
</CGI_Result>

```

The values of the result code:

<result></result> means the common execute result

| value | mean                            |
|-------|---------------------------------|
| 0     | Success                         |
| -1    | CGI request string format error |
| -2    | Username or password error      |
| -3    | Access deny                     |
| -4    | CGI execute fail                |
| -5    | Timeout                         |
| -6    | Reserve                         |
| -7    | Unknown error                   |
| -8    | Reserve                         |

## Pan, Tilt and Zoom control

There are two steps of moving. The first command you need to start the camera movement. The camera continues moving until it reaches the end state or you stop moving with ptzStopRun command.

### Possible commands:

/cgi-bin/CGIProxy.fcgi?cmd=**ptzStopRun**

/cgi-bin/CGIProxy.fcgi?cmd=**ptzMoveUp**

/cgi-bin/CGIProxy.fcgi?cmd=**ptzMoveDown**

/cgi-bin/CGIProxy.fcgi?cmd=**ptzMoveLeft**

/cgi-bin/CGIProxy.fcgi?cmd=**ptzMoveRight**

/cgi-bin/CGIProxy.fcgi?cmd=**ptzMoveTopLeft**

/cgi-bin/CGIProxy.fcgi?cmd=**ptzMoveTopRight**

/cgi-bin/CGIProxy.fcgi?cmd=**ptzMoveBottomLeft**

/cgi-bin/CGIProxy.fcgi?cmd=**ptzMoveBottomRight**

The zoom commands are similar to pan and tilt functions. It starts zooming after the commands until it reaches the maximum or minimum zoom, or you stop it with the zoomStop command.

```
/cgi-bin/CGIProxy.fcgi?cmd=zoomStop
```

```
/cgi-bin/CGIProxy.fcgi?cmd=zoomOut
```

```
/cgi-bin/CGIProxy.fcgi?cmd=zoomIn
```

Reset pan and tilt to default position:

```
/cgi-bin/CGIProxy.fcgi?cmd=ptzReset
```

Get the speed of pan and tilt:

```
/cgi-bin/CGIProxy.fcgi?cmd=getPTZSpeed
```

Get all preset points:

```
/cgi-bin/CGIProxy.fcgi?cmd=getPTZPresetPointList
```

Delete preset point:

```
/cgi-bin/CGIProxy.fcgi?cmd=ptzDeletePresetPoint&name=test
```

Add new preset point:


```
/cgi-bin/CGIProxy.fcgi?cmd=ptzAddPresetPoint&name=test
```

Go to preset point:

```
/cgi-bin/CGIProxy.fcgi?cmd=ptzGotoPresetPoint&name=Alap
```

## TASKS:

(T.1) Start the Leap Motion Diagnostic Visualizer.

hint: When the Leap Motion control panel application is running, it displays an icon in the notification area of the Windows Taskbar:  The Leap Motion control panel application icon menu provides the Visualizer application.

Check the operation of Leap Motion, hold your hand over the Leap Motion device.

Set the visualizer to fullscreen (s)

Hide the camera images (f)

Check the help, display the Visualizer framerate, Leap Motion framerate, and key commands. (h)

Observe the speed of the device! (upper left corner: device fps) What is the range of values?

Note the speed into the measurement report! Save the screenshot!

hint: Switch off the full screen mode, you can take the screenshot only window mode (s). Press Alt-PrtSc to copy the screenshot to clipboard.

Move the 3D point of view (→ ←)

Try to use the other functions too! Save the screenshot!

Hold both of your hands above of Leap Motion device. Find a position when the Leap motion can see both of your hands, but there is a recognition error. Save the screenshot!

## Webcamera

(T.2a) Login to web administration page of webcamera! Use the Internet Explorer!

<http://w.x.y.z:88> (The conductor give the proper IP address) Port number:88

Username: operator

Password: oper

Move the camera, use the controls. Zoom in, and show the green plant in center of image! Save the screenshot!

(T.2b) Following IP stream.

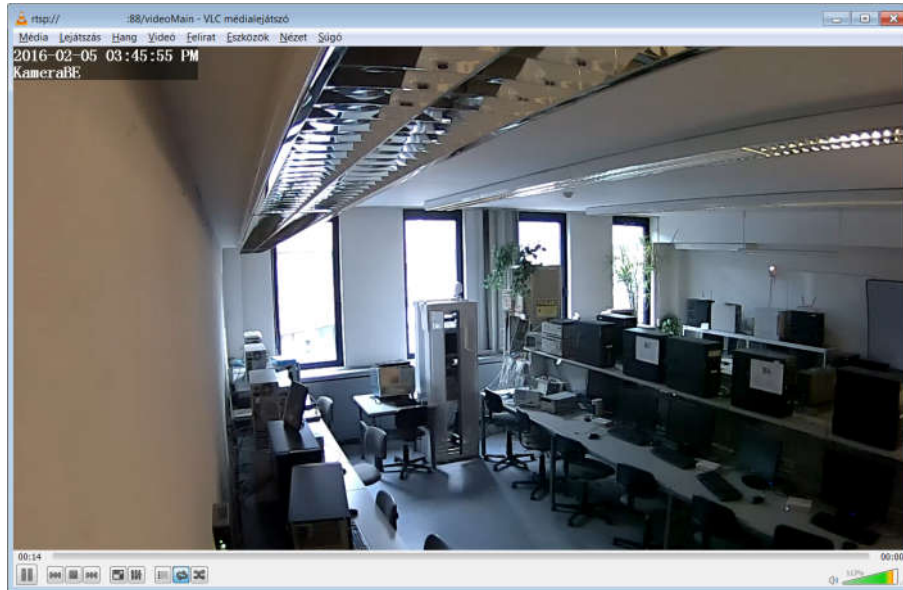
Close the Internet Explorer! Start the VLC media player!

Menu: Média/Hálózati műsor megnyitása:

rtsp://w.x.y.z:88/videoMain

Username/Password: visitor/visi

Save the screenshot:



(T.3) Follow the Camera picture with VLC player and control the camera with HTTP request!

Do not use web administration page to control camera

First, check the zoom speed. Use Internet Explorer, Google Chrome or Mozilla Firefox:

<http://w.x.y.z:88/cgi-bin/CGIProxy.fcgi?cmd=getZoomSpeed&usr=operator&pwd=oper>

- 0- Fast
- 1- Normal
- 2- Slow

Note the speed into the measurement report!

Move the camera to the door!

Save the screenshot and note the commands into the measurement report!

hint: If the HTTP request does not work, press the Reload button! The internet browser may answer you request from cache and the camera does not get the command.

Calculate the rotation speed (degrees per second) !

## Hello world program

(T.4)

Use Visual Studio!

The help is accessible in D:\tmp\docs.

Copy this code to sample.cpp. The sample.cpp is in directory d:\tmp\helloworld.

```
#include <iostream>
#include <string.h>
#include "Leap.h"

using namespace Leap;

int main(int argc, char** argv) {

    // Keep this process running until Enter is pressed
    std::cout << "Press Enter to quit..." << std::endl;
    std::cin.get();

    return 0;
}
```

Compile and run it! Check the functionality!

### Connection checking:

The connection between your program and the Leap Motion service is working via object Controller.

Modify your program with these lines:

```
int main(int argc, char** argv) {
    Controller controller;

    // Keep this process running until Enter is pressed
    std::cout << "Press Enter to quit..." << std::endl;
    std::cin.get();

    return 0;
}
```

Compile and run it! Check the functionality!

Extend your program with an object Listener. It is necessary to get data. Use these lines:

```
class SampleListener : public Listener {
public:
    virtual void onConnect(const Controller&);
    virtual void onFrame(const Controller&);
};

void SampleListener::onConnect(const Controller& controller) {
    std::cout << "Connected" << std::endl;
}

void SampleListener::onFrame(const Controller& controller) {
    std::cout << "Frame available" << std::endl;
}
```

Modify the function main! You can connect the listener to the controller:

```

int main(int argc, char** argv) {
    SampleListener listener;
    Controller controller;

    controller.addListener(listener);

    // Keep this process running until Enter is pressed
    std::cout << "Press Enter to quit..." << std::endl;
    std::cin.get();

    // Remove the sample listener when done
    controller.removeListener(listener);

    return 0;
}

```

Compile and run it! Check the functionality!

To recognize the default gestures you have to enable the recognition of them. Modify with these lines:

```

void SampleListener::onConnect(const Controller& controller) {
    std::cout << "Connected" << std::endl;
    controller.enableGesture(Gesture::TYPE_SWIPE);
}

```

Compile and run it! Check the functionality!

The program gets the data frame by frame. Modify the code to show the most important parts of data:

```

void SampleListener::onFrame(const Controller& controller) {
    const Frame frame = controller.frame();
    std::cout << "Frame id: " << frame.id()
        << ", timestamp: " << frame.timestamp()
        << ", hands: " << frame.hands().count()
        << ", fingers: " << frame.fingers().count()
        << ", tools: " << frame.tools().count()
        << ", gestures: " << frame.gestures().count() << std::endl;
}

```

Compile and run it! Check the functionality!

(T.5) Modify the program! If there was a gesture swipe, the program should download the content of webpage [alpha.tmit.bme.hu](http://alpha.tmit.bme.hu). Use the following cURL based class:

First you need an instance of the class CurlEasyGet:

```

#include "CurlEasyGet.h"
CurlEasyGet mycurl;

```

Use the get member function to take an HTML request:

```

mycurl.get("http://alpha.tmit.bme.hu/");

```



Show the result:

```
printf("%i bytes: %s\n",mycurl.mem.size,mycurl.mem.memory);
```

Extend the program code to solve this task: Move the camera to the door from any possible position. Zoom in! Use preset position as a starting point, but you cannot use preset position as an end point. The moving should be based on timing e.g. Sleep function. Specify the timing data in experimental way. The program writes short summary of the actual activity to the consol.

You can use function Sleep() to timing! The parameter is given in milliseconds:

```
#include "Windows.h"
```

```
Sleep(100);
```

## Gesture control

[Close the previous project, and load another form directory d:\tmp\Gestusdemo!](#)

(T.6) Make an c++ which has the following features:

- The program uses the Leap Motion to control the webcam!
- The webcam can move to all direction and zoom in/out. **We offer to use the position of the palm or a finger.**
- Use a gesture to move a preset position.
- Show the basic information about the moving on the consol. Do not show the irrelevant information.
- Use the HTTP based webcam cgiProxy.
- Write a short user manual about the control.
- Eliminate the unused code parts from the sample code

Reference:

- [1] [https://developer.leapmotion.com/documentation/cpp/devguide/Leap\\_Overview.html](https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Overview.html)
- [2] <http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/>
- [3] <https://developer.leapmotion.com/getting-started/javascript>
- [4] <https://web.archive.org/web/20070306232735/http://www.rtsp.org/2001/faq.html>

## Measurement report

VITMMB04 Smart City Laboratory, Gesture Control

|                                       |
|---------------------------------------|
| Place:                                |
| Date, time:                           |
| Name and neptun code of the students: |

The equipment's identifiers:

|                                |
|--------------------------------|
| Computer name:                 |
| Computer IP:                   |
| OS type and version:           |
| Leap Motion Id (1,2 or 3):     |
| Leap Motion Software Version:  |
| Leap Motion Controller ID:     |
| Leap Motion Firmware Revision: |
| Webcamera type:                |
| Webcamera IP:                  |
| VLC player version:            |
| Web browser type and version:  |
| Visual Studio version:         |

Task 1:

Task 2:

Task 3:

Task 4:

Task 5:

Task 6:

Attachment:

Source of T4

Source of T5

Source of T6

Control questions:

1. What is the RTSP protocol?
2. What is the working principle of Leap Motion device?
3. List at least three gestures, which the Leap Motion API can recognize by default!
4. List at least three objects that manage the Leap Motion API by default!!
5. What is the format of the result of webcam's CGI call?
6. What is the commands of the camera moving. Give at least 3 examples!
7. What is the two basic programming interfaces of Leap Motion?
8. What is the function of onFrame function in Leap Motion API?
9. What is the effect of controller.enableGesture(Gesture::TYPE\_SWIPE) call? Why necessary to call this function?
10. What can the Leap Motion identify as a tool? Give an example!