Universal Hand Control

Using your hand to control your connected devices and applications.

Introduction

Today technologies are available that allow us to do hand pose estimation (neural networks) or to measure the position of your hand in space (depth cameras). A device like the OAK-D even offers the possibility to combine the two techniques into one device.

Using your hand to control connected devices (includes smart speaker, smart plugs, TV, … an ever-growing market) or to interact with applications running on your computers without touching a keyboard or a mouse, becomes possible. More than that, with Universal Hand Control, it gets straightforward.

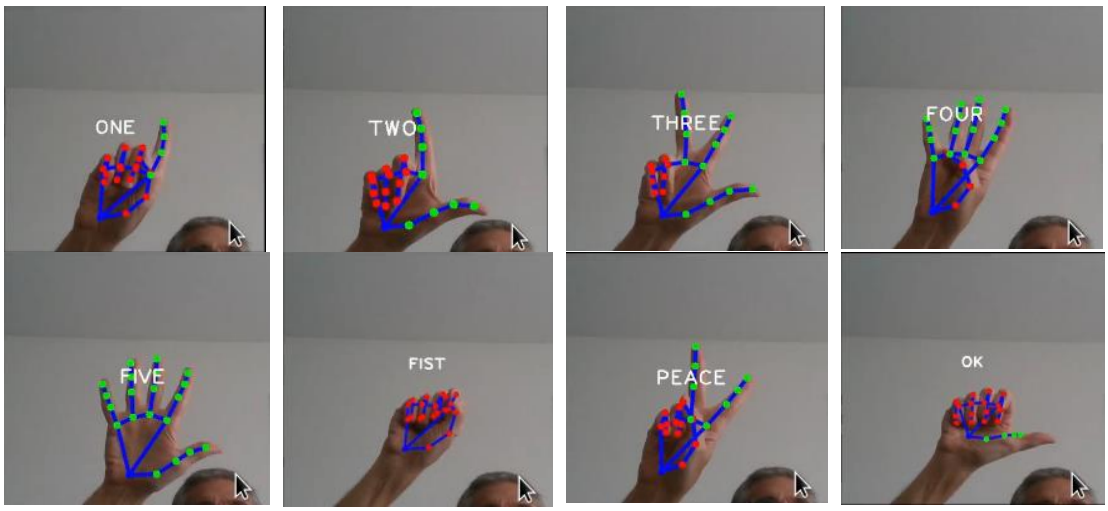
Universal Hand Control is a framework (dedicated hardware + python library) that helps you to easily integrate the « hand controlling » capability into your programs.

So, if you know how to write a program that, let's say, switch on your connected music speaker, then with a few more lines of code, you can do the same with a simple gesture.

Gestures, Airzones

Currently, Universal Hand Control proposes 2 types of events that it can recognize and transmit to your application:

1. **Gestures** = a set a predefined hand poses.

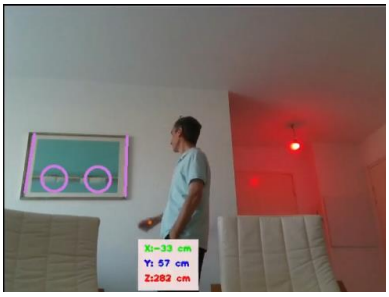


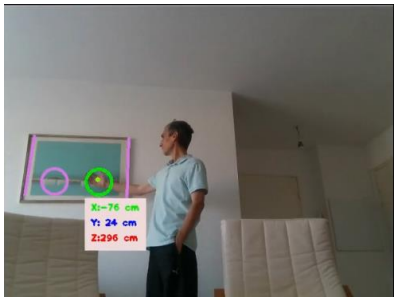
2. **Airzones :** an airzone is 3D volume defined by the user that generates events when a hand enters/leaves the zone or moves inside.
   Currently, there are 3 types of airzones:
   a. The **button** is a sphere defined by a center point given by its 3D coordinates in the camera coordinate system and a radius called 'tolerance'.

Typically, a button has a diameter of 15-20 cm and is used as a switch or a button. Events are sent to the client program when the hand enters or leaves the sphere.

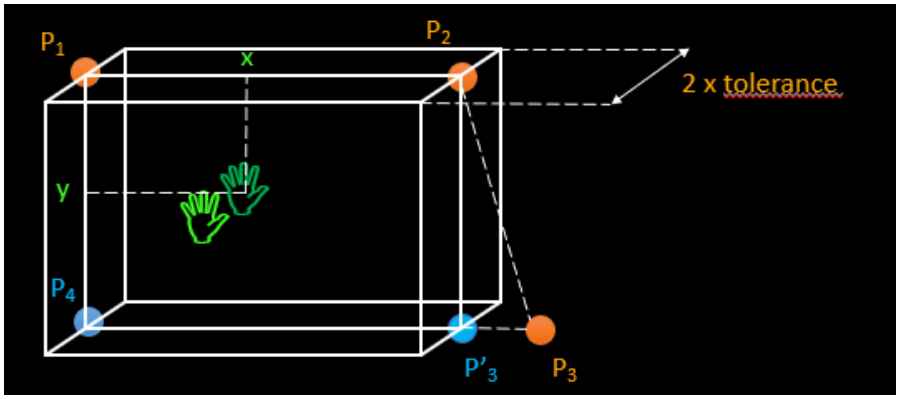
b. The **slider** is a cylinder defined by a line segment given by its 2 extremities and a radius 'tolerance'.
In the typical use of a slider, periodic events are sent to the client program as long as the hand is moving along the cylinder. The events will contain the normalized position of the hand along the axis (float value between 0 and 1). So, the slider is an convenient tool to interactively choose a single continuous value. The periodicity parameter which specifies the time between 2 consecutive events can be specified in the client program (usually between 1 and 5 tenths of a second).

In the picture below, there are 4 airzones: 2 buttons (represented by the circles drawn in the painting) and 2 sliders (represented by the 2 segments on the painting frame corresponding to the cylinders axis). The button on the right side acts as a toggle switch for the light, the slider as a dimmer. In a similar manner, the button and the slider on the left side control another device.

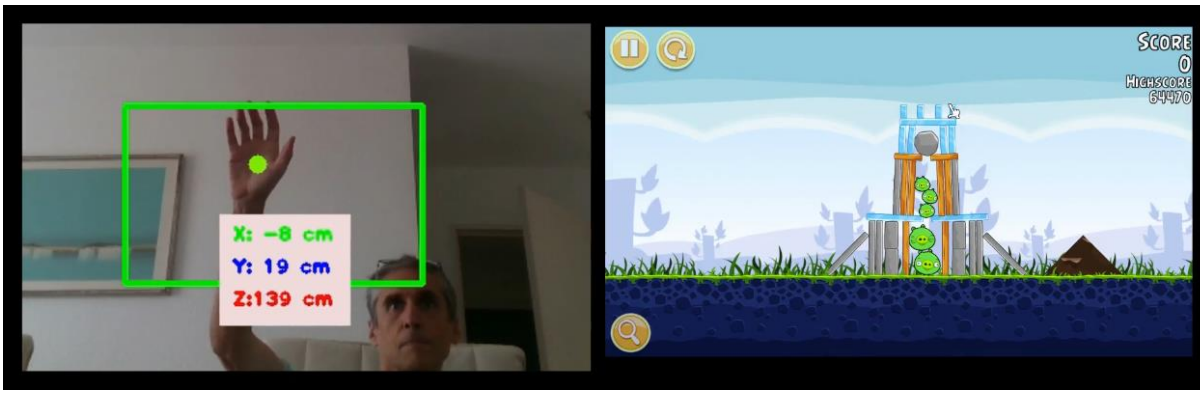




c. The **pad** is a « flat » parallelepiped defined by a foundation plane given by 3 of its points (P_1, P_2 and P_3 on the schema below) and a perpendicular depth 'tolerance'. A hand inside the parallelepiped can be projected onto the foundation plane. The projection has a position (x, y) in the coordinate system (origin P_1, P_1P_2 , P_1P_4) :



As for the slider, the normalized position of the hand is transmitted in the event to the client app. A typical use of a pad is to emulate a mouse pad as showed in the image below. Moving the hand in the pad can be used to move the mouse of a computer.



A gesture can be associated with an airzone: in that case, an event is generated if and only if the gesture happens inside the airzone.

Video illustration of airzones: https://youtu.be/blTn5f_NXoQ?t=143

Hardware

The hardware part of Universal Hand Control several consists of several required components. For each component, several options may exist. The following table list the components in 2 possible configurations:

	DepthAI configuration "The Box"	Prototyping configuration
RGB camera + aligned depth sensor (1)	OAK-D (or BW1092)	Intel Realsense D415
Processor to run models' inferences	MyriadX of OAK-D	PC Intel Core
Processor to run other code and communication with client app	Raspberry Pi	PC Intel Core
Network for communication with client app	Wifi	Wifi

(1): "aligned" means that knowing the position of a pixel in the color image, it is possible to get the depth of this pixel in the depth image and therefore, to get the 3D position of the corresponding point in the camera coordinate system.

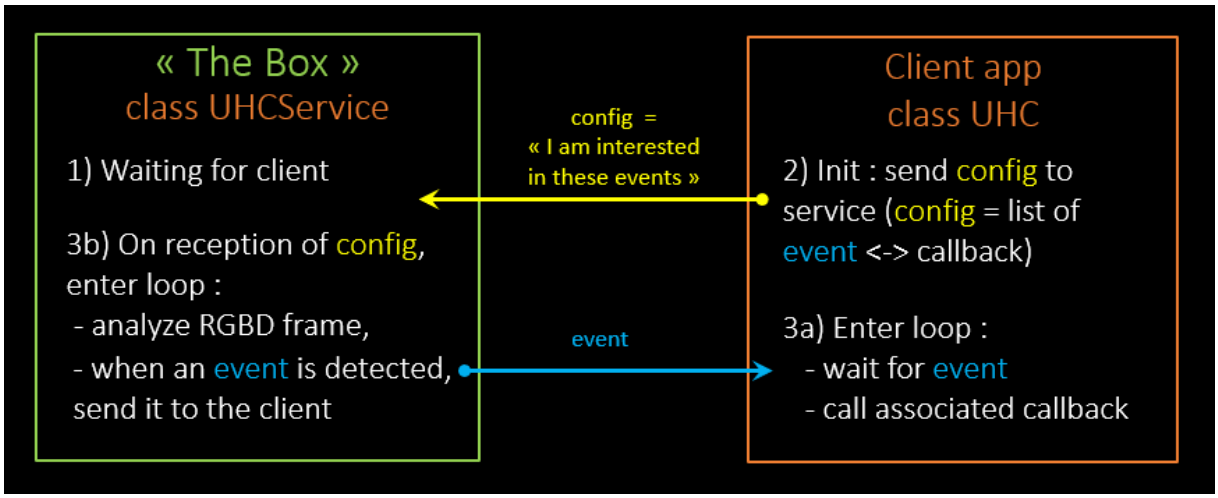
Important note: the *DepthAI configuration* is the configuration I first thought about when I had the idea of this project. Unfortunately, at that time, the DepthAI software didn't include some needed features (like the alignment depth/RGB or the pipeline Gen 2). So, to advance the agenda, I used the *Prototyping configuration* to develop the code. The examples shown in the companion video were made with this configuration. A few days before the end of the competition, Brandon kindly sent me a BW1092. On this board, the color camera is adjacent to the right mono camera. So, at first approximation, it is possible to map a pixel from the color frame to right mono frame and vice versa. Finally, I was able to successfully test Universal Hand Control in the following intermediate configuration: BW1092 as sensor and PC Intel Core for all processing. To run the models on the MyriadX, I still have to wait for the pipeline Gen 2.

Ideally, the *DepthAI configuration* could be very small, all components contained in a box (hence the name "The Box"), with only one outgoing wire (for alimentation), easy to carry and to place wherever it suits your application needs.

Software architecture

Universal Hand Control relies on a client-server architecture with:

- On the server side, an instance of the python class UniversalHandControlService, running on the Box, is waiting for a client to send its configuration (which contains the Gestures and Airzones the client is interested in), analyzing the frames produced by the RGB/Depth sensor, and notifying the client when events occur;
- On the client side, the app has instantiated the class UniversalHandControl with its configuration as parameter, is waiting for events coming from the Box and calling associated callbacks.



The communication between the service and the client use the MQTT protocol, with the MQTT broker running on the Box. Note that messages are very light (config, events, service messages) and no heavy object like image is transferred.

Analyzing RGB/depth frames

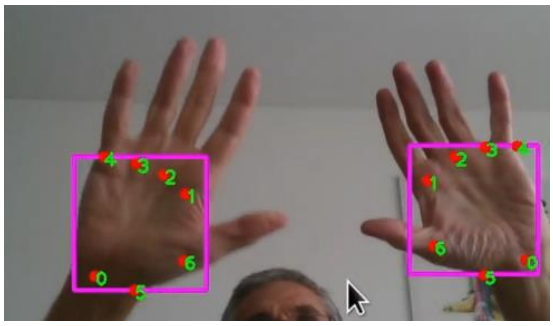
The main task of the service is to analyze the frames from the RGB/depth sensor in order to:

- detect hand occurrences in the RGB frame;
- recognizing gestures if requested by the client;
- find the location of the hand in space with the help of the depth frame.

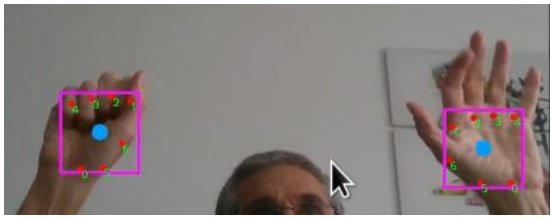
Note: there is a video illustrating this chapter:

For the first two tasks, we rely on 2 models trained by Google for its Mediapipe Hands solution and converted in OpenVino IR format by the amazing PINTO0309.

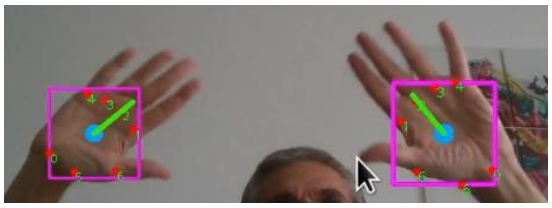
The first model is a **palm detector**. It outputs a bounding box and 7 keypoints for each hand detected.



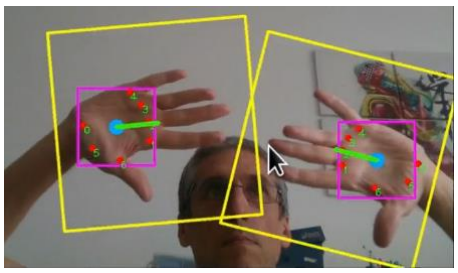
We use the center bounding of the bounding box (blue circle in the image below) as the reference for the hand. For instance, later when we want to find the location in space of the hand, we actually measure the position of its center:



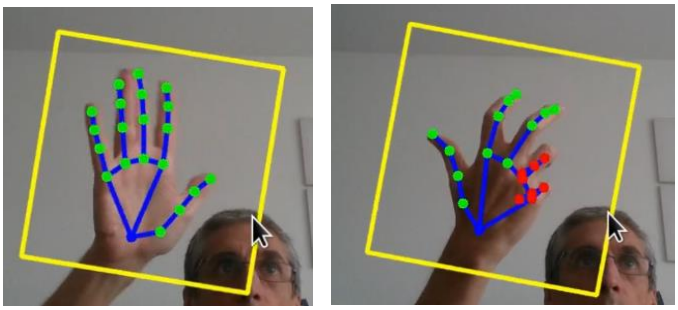
The line connecting the wrist keypoint and the MCP of the middle finger gives a rotation of the hand. This information can be transferred and used by the client app.



From the center and with the rotation of the palm, a bigger rectangle (in yellow) is calculated that encloses the whole hand:



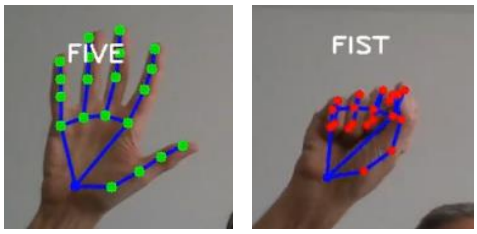
The cropped image region delimited by this rotated rectangle is then given as input to the **hand landmark model** which is able to infer 21 keypoints for the whole hand (4 keypoints per finger + wrist keypoint).



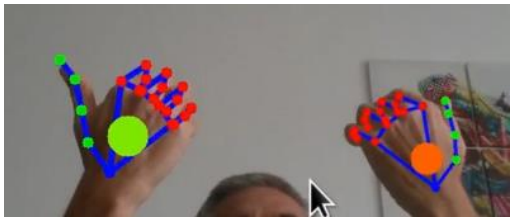
With the landmarks coordinates, we can evaluate the status of each finger (open or closed).

Then knowing the status of the 5 fingers, we can recognize the gesture. For instance :

- if all 5 fingers are open, gesture = FIVE;
- if all 5 fingers are closed, gesture = FIST.



The hand landmark model also evaluates the handedness (is it a right hand • or a left hand • ?).



Finally, from the depth frame aligned with the color image, it is possible to know the position of the hand center in the camera coordinate system:

Video illustrating this chapter: https://youtu.be/blTn5f_NXoQ?t=6

Examples

Example 1 – Light switch

Video: https://youtu.be/blTn5f_NXoQ?t=193

Here, we want to switch on/off a light when we do the FIST gesture. This basic example is a pretext to present the general structure of any Universal Hand Control client program.

Below is the corresponding python code:

```python
from UniversalHandControl import *

# Yeelight controller
from yeelight import Bulb, discover_bulbs
bulbs = discover_bulbs()
bulb = Bulb(bulbs[0]['ip'])

config = {
    # app_name: name of the current application
    # (used as MQTT topic name for communication with service)
    'app_name': 'Toggle light',
    # service_connect: arguments to connect to MQTT broker
    'service_connect': {'host':"192.168.1.21"},
    # List of gestures and associated callbacks
    'gestures' : [
        {'name': 'ON_OFF',
        'gesture':'FIST',
        'hand':'right',
        'callback': 'toggle_light'},
    ]
}

def toggle_light(event):
    uhc.sound("sounds/finger_click.wav")
    event.print_line()
    bulb.toggle()

uhc = UniversalHandControl(config)
uhc.loop_forever()

```

```python
from UniversalHandControl import *
```

Import of class UniversalHandControl.

```python
config = {
```
...
```python
}
```

The definition of the configuration is the most important part of the program ! Let's look in more detail.

```python
    'app_name': 'Toggle light',
```

'app_name' is used as a MQTT topic for subscribing to/publishing events.

```python
    'service_connect': {'host':"192.168.1.21"},
```

'service_connect' define the parameters for connection to the MQTT broker and contacting the UHC service via a reserved topic name.

```
15      'gestures' : [
16          {'name': 'ON_OFF',
17          'gesture':'FIST',
18          'hand':'right',
19          'callback': 'toggle_light'},
20      ]
```

In ‘gestures’, we specify the events we want to track, and the callbacks called on notification of these events. Here we have only one entry that says: when a FIST pose with the right hand is detected, I, the client app, will call the function ‘toggle_light()’ which is defined at line 23.

```
28  uhc = UniversalHandControl(config)
29  uhc.loop_forever()
```

Instantiation of an UHC object with the config previously defined, then enter the loop “1) wait for event, 2) call associated callback”.

In an app using airzones, we would declare an “airzones” list similarly to the “gestures” list.

Example 2 – Controlling devices with gestures

Video: https://youtu.be/bITn5f_NXoQ?t=215

This example is an extension of the previous one. Now we can control a light, a TV and a music speaker. A gesture ONE, TWO or THREE with the left hand selects one of the 3 devices. The right hand is used to control the selected device:

- FIST or OK to switch on/off,
- ONE, TWO or THREE to change presets,
- FIVE to change the volume or brightness.

Below is the “gestures” list from the config:

```
'gestures' : [
    {'name': 'LIGHT', 'gesture':'ONE', 'hand':'left', 'callback': 'set_context'},
    {'name': 'TV', 'gesture':'TWO', 'hand':'left', 'callback': 'set_context'},
    {'name': 'MUSIC', 'gesture':'THREE', 'hand':'left', 'callback': 'set_context'},
    {'name': 'ON_OFF', 'gesture':['FIST','OK'], 'hand':'right', 'callback': 'command_on_off'},
    {'name': 'LEVEL', 'gesture':'FIVE', 'hand':'right', 'callback': 'command_level', \
       "trigger":"periodic", "first_trigger_delay":0.3, "next_trigger_delay":0.3, "params": ["rotation"]},
    {'name': 'PRESET 1', 'gesture':'ONE', 'hand':'right', 'callback': 'command_preset'},
    {'name': 'PRESET 2', 'gesture':'TWO', 'hand':'right', 'callback': 'command_preset'},
    {'name': 'PRESET 3', 'gesture':'THREE', 'hand':'right', 'callback': 'command_preset'},
]
```

Note the parameter ‘trigger’ for the gesture named ‘LEVEL’. The value ‘periodic’ means that an event will be sent every 0.3s (value of ‘next_trigger_delay’) as long as the user is doing the FIVE gesture. When ‘trigger’ is not specified, as it is for other gestures, the default behavior is to send only one event to the client.
Note also the “params”: [“rotation”]. It means that the event will be loaded with the value of the palm rotation. In this app, we use the rotation to determine if we want to lower or to raise the volume or brightness.

Example 3 – Controlling devices with airzones

Video: https://youtu.be/bITn5f_NXoQ?t=319

This example has already been addressed in the introduction of the Airzone concept.

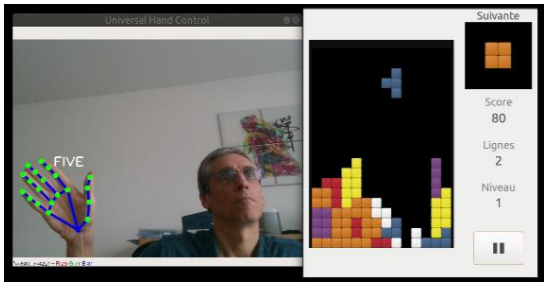
Below is the definition of the “airzones” list from the config:

```
'airzones' : [
    {'name': 'MUSIC_ONOFF', 'type': 'button', 'points': [(-1.1, 0.26, 2.97)], 'callback': 'music_on_off'},
    {'name': 'MUSIC_VOLUME', 'type': 'slider', 'points': [(-1.33, 0.37, 2.97), (-1.31, -0.1, 3.1)], \
        'trigger': 'periodic', 'callback': 'music_volume'},
    {'name': 'LIGHT_ONOFF', 'type': 'button', 'points': [(-0.76, 0.27, 3)], 'callback': 'light_on_off'},
    {'name': 'LIGHT_BRIGHTNESS', 'type': 'slider', 'points': [(-0.54, 0.4, 3), (-0.56, -0.08, 3.19)], \
        'trigger': 'periodic', "next_trigger_delay": 0.5, 'callback': 'light_brightness'},
],
```

Example 4 – Controlling the keyboard to play Tetris

Video: https://youtu.be/blTn5f_NXoQ?t=367

This example demonstrates the use of gestures to generate key presses.



Example 5 – Controlling the mouse to play Angry Birds

Video: https://youtu.be/blTn5f_NXoQ?t=409

This example uses both airzone and gesture. Moving a hand in a pad airzone makes the computer mouse moves accordingly. And the FIST or OK gesture triggers and hold a pressing on the left button.

```
'airzones' : [
    {'name': 'MOUSE', 'type': 'pad', 'coordinates_type': "relative_to_anchor" ,
    'points': [(w/2,-h/2,0), (-w/2,-h/2, 0), (-w/2,h/2,0)],
    'tolerance':0.4, 'trigger': 'continuous', 'callback': 'move'},
],
'gestures' : [
    {'name': 'PRESS_RELEASE', 'gesture':['FIST', 'OK'], 'callback': 'press_release', "trigger":"enter_leave",
    "first_trigger_delay":0.1, "max_missing_frames":3},
],
```

'coordinates_type': "relative_to_anchor" in the airzone definition, means that the following 'points’: [(w/2,-h/2,0), (-w/2,-h/2, 0), (-w/2,h/2,0)] are coordinates relative to an anchor point, instead of the camera origin.
During the app initialization, the UniversalHandControl library asks the user to choose the anchor point location by keeping his hand still at the desired location during a few seconds.

Example 6 – Virtual synth

Video: https://youtu.be/blTn5f_NXoQ?t=458

Another example that uses a mix of airzones and gestures: a pad airzone for the keyboard, some gestures to select the MIDI channel and a slider to change the volume of the current MIDI channel.

Limitations, robustness, accuracy

Unfortunately, the palm detection and hand landmark models are not perfect:

- A common problem for detection models is the false positives: a “ghost” palm is detected in a frame whereas there is no hand. In that case, we don’t want to trigger an event. Thanks to the confidence score computed by the model, we can filter out some of the false positives. For the remaining ones, let’s use the fact that false positives are usually not consistent: they appear in one frame but not in the following. So, to send an event to the client, the event must have appeared in consecutive frames during a minimum duration (generally between 0.1s and 0.8s). This duration can be changed in the parameter ‘first_trigger_delay’ of a gesture or an airzone;
- Currently, we suppose there is only one body in the image, so only 2 hands max, one right hand and one left hand. In case, we detect for instance more than one right hand, we keep the instance which have a better global score. The global score is calculated from the palm detection score, the landmark score and handedness score (the three scores are outputs of the models).
- The hand landmark model accuracy highly depends on the lighting and the background scene behind the hand. A uniform background is better. A distance to the camera up to 1.5m is advised. Some gestures are more robustly detected than others (FIVE is the winner). Of course, presenting the side of the hand to the camera does not work well. Handedness also is not an easy task: for instance, it can be very hard to distinguish between the palm side of a right hand and the back side of a left hand, when doing the FIVE gesture far from the camera.
- Palm detection model supports longer distance (up to 3m without problem). So, an app which uses only airzones and no gesture, and therefore uses only the palm detection model could support longer distance than an app that uses gestures.

What’s next?

Here is a list of possible future improvements:

- Symbols: a new type of events, in addition to gestures and airzones. The hand can draw a symbol in mid air (ex: triangle, square, 8) that can be recognized by a classifier.
- Tracking: if we have a tracking algorithm faster than the palm detection algorithm, we could use a loop-sequence (palm detection inference x 1 frame | tracking x N frames) to speed up the global inference time.
- Escape-gesture (or escape-symbol): a special reserved gesture that gives the possibility for the user to dialog with the UniversalHandControlService (instead of the client app).
- Jukebox of apps: with the escape-gesture, the user could ask to stop the current app and start another app chosen from a set of apps.
- Body pose: probably much more accurate than the hand landmark model for inferring handedness. A body pose model is slow but it could be called every n frames.
- Dynamic airzone position: the body pose could also allow the dynamic position of an airzone. For instance, in the example where the user controls the mouse to play Angry Birds, instead of asking the user to set the position of the pad when starting the app, the pad airzone could be automatically placed relatively to the right shoulder (a bit in front and lower).

Thank you!