Gesture Recognition Interface & Algorithm – User Manual

Gesture EVK version 1.0.0
User Manual - Table of contents

1. Overview

2. Getting started
   - Step 1: Hardware setup
   - Step 2: Firmware loading
   - Step 3: Software launching

3. Interface & widget guide

4. Log & Replay

5. Gesture Library: Programming Guide
Overview
• **Gesture solution** aims to predict basic gestures and track the hand of the user in front of the ToF sensor.

• **Gesture EVK** has been developed as a tool to give an overview of the ST ToF sensor performances and for Gesture-related algorithm development.
Gesture + Hand tracking Demo

Gesture Evaluation Kit

Examples of applicative development

Sensor output visualization

Webcam monitored

284.9 | 79.1 %
Getting Started
Step 1: Hardware setup

Compatible operating system:
- Windows

Compatible boards:

Mandatory step

Connect the board to the PC via a USB to Micro-USB cable
Step 2: Firmware loading

Open the .FW folder in the Gesture EVK repository to find the firmware for VL53L5CX sensor

Drag and drop the .bin file to flash the board

Mandatory step
Step 3: Software launching

Double-click on the .exe file in the GestureEVK folder to launch the Gesture GUI software.

Make sure:
- Board is automatically detected and selected. If not, refresh Com port list and select a board flashed with the firmware.
- Corresponding sensor driver is selected.
- Choose the preset of your choice, this configuration will be applied at start.

Press Start ranging to start the playing with the different functionalities.
Interface & widget guide
Open this user manual
Refresh \textit{Com port} list
The board is automatically detected
Launch the Gesture EVK
Before, make sure of the settings listed in \textit{Step 3}
Load a previously saved dataset using the GestureEVK
The tool supports full replay of a dataset for analysis
Click to open release notes
Select the \textit{Com port} on which the Nucleo board + sensor flex are connected
Nucleo firmware must be already loaded
Select the corresponding sensor driver type
Select a sensing preset
This configuration will be applied at start
Configure how information are displayed in the above window. Details on next page.
Display settings menu

- Select data to display
- Font size
- Change the viewing mode (only color, full data...)
- Number of decimal
- Turn on/off the camera
- Change camera frequency
- Change camera source to display
- Blur camera output display
- Change camera resolution and format
- Return to preset shift
- Shift sensor field of view relative to the camera picture (from -50% to +50% of the picture dimension)

Tip: Scroll through the menu using the mouse wheel!
**Widgets**

- **Xtalk Calibration**
  Performs an Xtalk Calibration and loads it in the sensor.

- **Curves plot**
  Complete tool to plot curves among all the data outputted by the device.

- **Circular Level Control**
  Control a level using circle gestures.

- **Hand Tracking**
  Open a display window (like the main one), connected to the same sensor. The view is adding a « colored dot » to track the hand position.

- **Gesture Detection**
  Detects the Gestures performed by the user and displays it.

- **Mouse Control**
  Open a joystick-controlled by the hand position (displayed by the green dot). It controls the mouse.

- **Photo Viewer Control**
  Enables a list of gestures emulating the main viewer actions (Next, Previous, Play, Zoom…)

- **Slideshow Control**
  Control a slideshow by emulating the main PowerPoint actions (Present, Next, Previous…)

- **Cooking Plate**
  Example of a touchless cooking plate controlled by Gestures.

- **Ground Truth Labeler**
  (available in replay mode)
  Allows the user to add labels in the data log.
DESCRIPTION:
Gesture Detection widget is a graphical tool for users who want to evaluate the Gesture algorithm.

FEATURES:
• Gesture Recognition
• Gestures Counting
• Level Control
  • Activated after 1s hold by default in the algo
  • The bar graph on the side display the Level Control value return by the algo, default value: 0% = 10cm, 100% = 40cm
**DESCRIPTION:**

Circular Level Control widget is a graphical tool for users who want to evaluate the Circle hand detection

**FEATURES:**

- Display the polar angle of the hand position
- Translate the angle to a percentage
- Activated after 1s hold by default in the algo

*Note: it is interesting to open the Hang Tracking Widget on parallel of the Circular Level Control*
DESCRIPTION:

Hand Tracking widget is a graphical tool for users who want to evaluate the Hand Tracker algorithm.

FEATURES:

• Hand Position tracking
• Colored Dot aligned with the Z distance
• Distance (in mm) written in the dot
• Display hand trace of the 20 previous positions

Note: All the settings are defined in “Display settings menu” as this widget is a child of the main window and inherits all its features.
Description:

Photo Viewer Control widget is a real application of the gesture’s recognition applied on a common use-case that is the photo slide control.

Features:

- Right swipe = Previous photo/video
- Left swipe = Next photo
- Tap = Play / Pause on video media
- Level Control = Zoom In / Out
  - No Zoom = 30cm
  - Max Zoom In = 10 cm
  - Granularity 10 steps defined by software
DESCRIPTION:
Slideshow Control widget is a real application of the gesture's recognition applied to a common use-case is the PowerPoint slide control.

FEATURES:
• Right swipe = Previous slide
• Left swipe = Next slide
• Raised hand = Switch from Edition to Slideshow mode
• Tap = Play / Pause the media
• Double Tap = Exit the Slideshow mode
DESCRIPTION:
Cooking Plate widget is a real application of the gesture’s recognition applied on a common use-case that is a cooking plate HMI

FEATURES:
• Hold hand below 10cm distance = Turn ON / Turn OFF / Unlock the HMI
• Left swipe = Next plate (1 → 2 → 3 → 4 → Timer → 1…)
• Right swipe = Previous plate
• Double Tap = Lock the Plate
• Level Control = Increase / Decrease the Power or Timer
DESCRIPTION:
Mouse Control widget is a joystick-controlled by the hand position (displayed by the green dot). It emulates the computer mouse

FEATURES:
• Hand position
  • Angle = cursor motion direction
  • Radius = motion velocity
• Tap = Press / Release left click
• Double Tap = Double left click
Log & Replay
Using Gesture EVK for gesture capture allows to monitor, log and replay all Gesture algorithm internal variables. To start replay you need to load the file: `\data\log_EwokMZ__gesture...`

Thanks to numerous features (2D display over camera frame, curve plotting, dedicated post-processing...), Gesture EVK allows for deep analysis of any data from the VL53L5CX sensor and all parameters of the Gesture algorithm.
Example: Laptop
Recording a Gesture sequence 1 / 4

Note: In order to have a full understanding of the scene, it is important that this procedure is strictly followed.

Step 1:
Place the sensor close to the laptop webcam

Step 2:
Follow the *Getting Started* procedure

- Connect the board to the PC
- Flash the board with the firmware
- Launch the Gesture EVK software

Laptop screen

ToF sensor

Webcam
Step 3: Display the live webcam feed

The camera image should appear after clicking on « Display camera »
- If not, ensure your webcam is functional.
- If several webcams are connected to the laptop, change the camera id

Step 4: Click on Log data button to start data logging (recording) Button should turn gray
Step 5:
Run through a gesture sequence of your choice

Example: 10 left swipe

Step 6:
When finished, click again on the Log data button to stop data logging

Gesture EVK can be close afterward
Step 7: Retrieve your log in your PC

- Go to the folder `\GestureEVK_vx.x.x\data`
- A new folder whose name contains the date and time of capture has been created
  
  Example: `log__EwokMZ__gesture_api__8x8__20211206_172015`
- Compress the folder

Step 8: Replay your sequence step by step

Share with your colleagues!
Gesture Library: Technical Guide
• The Middleware uses a custom structure to store sensor rangings: SEN_data_t
• This stores all the necessary information about the sensor and most recent ranging
• You will need to update this structure upon receiving new data from your sensor
• You will also need to specify some information through some setters
• Init is done when initializing the hand tracker or gesture predictor

```c
typedef struct {
    SEN_info_t info;
    long timestamp_ms;
    float ranging[SEN_MAX_ZONES];
    float peak[SEN_MAX_ZONES];
    bool valid[SEN_MAX_ZONES];
} SEN_data_t;
```
• The Sensor can have 8 different orientations:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

• Default is 1

• Change with this call

```c
SEN_set_orientation(&sensor_data, Params.senOrientation);
```
You will need to copy 3 values for each zone and set the timestamp
  - Ranging (zone distance in mm)
  - Peak (Photon count)
  - Valid (Should we use the zone)

You can declare valid as you want, recommended is

```c
valid = (nb_targets > 0) &&
  (target_status == DCI_TARGET_STATUS__RANGECOMPLETE ||
   target_status == DCI_TARGET_STATUS__RANGECOMPLETE_MERGED_PULSE);
```

Then update structure with this call

```c
SEN_set_data(&sensor_data, idx, ranging, peak, valid);
```
• This is the structure containing everything related to the hand tracking
  • Params has all the parameters, tuning the tracking
  • Hand is the current tracked hand
  • Path is an history of the tracked hand
• If you use the gesture prediction, it will be initialized automatically, otherwise use

```c
HT_init(&hand_tracker, &sensor_data);
```

```c
typedef struct {
    HT_params_t params;
    HT_hand_t hand;
    HT_path_t path;
} HT_proc_t;
```
• Upon starting the sensor you should specify the resolution and frequency it will be using.
  • Set the resolution with the number of zones using
    
    ```c
    HT_set_square_resolution(&hand_tracker, &sensor_data, Params.Resolution);
    ```

  • If you use the gesture prediction, set the frequency through it, otherwise use

    ```c
    HT_set_frequency(&hand_tracker, &sensor_data, 1000.0/Params.RangingPeriod);
    ```
• After loading the sensor data in a SEN_data_t structure, just call

```c
HT_run(&hand_tracker, &sensor_data);
```

• The results are stored inside the given HT_proc_t struct

• Hand position: `hand_tracker.hand.hand_x, hand_tracker.hand.hand_y, hand_tracker.hand.hand_z`

• Same hand as before: `hand_tracker.hand.subject_changed`

• Get last hand found:

```c
RFBVM_get_element_value(&hand_tracker.path.hand_x, hand_tracker.path.hand_x.count - hand_tracker.path.last_hand);
RFBVM_get_element_value(&hand_tracker.path.hand_y, hand_tracker.path.hand_y.count - hand_tracker.path.last_hand);
RFBVM_get_element_value(&hand_tracker.path.hand_z, hand_tracker.path.hand_z.count - hand_tracker.path.last_hand);
```
• This is the structure containing everything related to the gesture prediction
  • Params has all the parameters, tuning the prediction
  • State is the current state of the tracking
  • Speed_{(x|y|z)}_{mm_s} is the hand speed
  • _computer is a container for intermediate variables used for computations, don’t write in it
  • Gesture is the current prediction
• Initialize the structure and all dependencies with

```c
GW_init(&gest_predictor, &hand_tracker, &sensor_data);
```

```c
typedef struct {
    GW_params_t params;
    GW_state_t state;

    float speed_x_mm_s;
    float speed_y_mm_s;
    float speed_z_mm_s;

    GW_comp_t _computer;
    GW_gesture_t gesture;
} GW_proc_t;
```
• This is what you should access to get gesture information
• The current flag means the gesture is ongoing
• The probation flag goes to 0 after the are sure about the prediction
• The ready flag is equal to 1 when probation goes from 1 to 0
• Then you can read the label
• Level Control are continuous values – we advise to monitor the \textit{lc\_state} like in the example code to know if the hand is stable

```
typedef struct {
    GW_label_t label;   // Label of gesture
    long start_time_ms; // Start time
    long end_time_ms;   // End time
    bool current;       // Is the gesture ongoing
    bool probation;     // Is the gesture screened
    bool ready;         // Is the gesture to be used
    float distance_mm;  // Distance of the gesture
}
GW_gesture_t;
```

//Level Control
uint32_t lc_currentLevel; // Current level value in percent
float lc_polar_r;        // r in polar coordinates
float lc_polar_theta;    // theta in polar coordinates
```
typedef enum {
    GW_NONE  = 0,///< No gesture
    GW_LEFT  = 1,///< Swipe to the left
    GW_RIGHT = 2,///< Swipe to the right
    GW_DOWN  = 3,///< Swipe down
    GW_UP    = 4,///< Swipe up
    GW_TOWARD= 5,///< Tap (Toward the sensor)
    GW_AWAY  = 6,///< Away from the sensor
    GW_DOUBLETAP = 7,///< Double Tap (Toward - Away - Toward)
} GW_label_t;
• Upon starting the sensor you should specify the frequency it will be using
  • This will also affect every dependencies

```c
GW_set_frequency(&gest_predictor, &hand_tracker, &sensor_data, 1000.0/Params.RangingPeriod);
```
• After loading the sensor data in a SEN_data_t structure, just call

\[
\text{GW\_run}(&\text{gest\_predictor}, &\text{hand\_tracker}, &\text{sensor\_data});
\]

• The results are stored inside the given GW_proc_t and HT_proc_t struct

• Hand speed:

\[
\text{gest\_predictor.speed\_x\_mm\_s}, \quad \text{gest\_predictor.speed\_y\_mm\_s}, \quad \text{gest\_predictor.speed\_z\_mm\_s}
\]

• Gesture label:

\[
\text{gest\_predictor.gesture.label}
\]

• Can you trust gesture: (Flag going from 1 to 0)

\[
\text{gest\_predictor.gesture.probation}
\]

(Flag at 1)

\[
\text{gest\_predictor.gesture.ready}
\]

• When did gesture start:

\[
\text{gest\_predictor.gesture.start\_time\_ms}
\]
Commands managed by the example code

- List of UART commands:
  - help
  - enable
  - disable
  - params
  - set
  - calibrate
  - get_caldata

- help commands:
  - help

- params commands:
  - params
### Memory consumption (Bytes)

- Middleware

<table>
<thead>
<tr>
<th>.text</th>
<th>.data</th>
<th>.bss</th>
<th>dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>20,789</td>
<td>0</td>
<td>0</td>
<td>20,789</td>
</tr>
</tbody>
</table>

- Main variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>dec</th>
<th>hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>GW_proc_t</td>
<td>2,216</td>
<td>0x8A8</td>
</tr>
<tr>
<td>HT_proc_t</td>
<td>1,976</td>
<td>0x7B8</td>
</tr>
<tr>
<td>SEN_data_t</td>
<td>608</td>
<td>0x260</td>
</tr>
</tbody>
</table>

### Execution time (Cycles)

- GW

<table>
<thead>
<tr>
<th>Function</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>220,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>run</td>
<td>13,500</td>
<td>140,000</td>
<td>700,000</td>
</tr>
</tbody>
</table>

- HT only

<table>
<thead>
<tr>
<th>Function</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>160,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>run</td>
<td>10,500</td>
<td>135,000</td>
<td>700,000</td>
</tr>
</tbody>
</table>

- Data load

<table>
<thead>
<tr>
<th>Function</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>load</td>
<td>21,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Gesture Recommendations

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand distance</td>
<td>Distance between the hand and the sensor</td>
<td>Min: 15cm, Optimal: 30cm, Max: 50cm</td>
</tr>
<tr>
<td>Speed</td>
<td>The execution speed of a gesture</td>
<td>Limitations: fast swipes should be avoided to get better results</td>
</tr>
<tr>
<td>Swipe Amplitude</td>
<td>The amplitude of a horizontal swipe is defined by the distance between the start position and the stop position of the hand</td>
<td>Min: 15cm, Max: No limit</td>
</tr>
<tr>
<td>Tap Amplitude</td>
<td>The amplitude of a TAP motion is defined by the distance between the start position and the stop position of the hand</td>
<td>Min: 10cm, Max: No limit</td>
</tr>
<tr>
<td>Double Tap Interval</td>
<td>The double tap interval is the time between both taps.</td>
<td>Min: No Limit, Max: 250ms</td>
</tr>
<tr>
<td>Setup motion</td>
<td>A Setup motion is the motion done by the user to bring the hand in the sensor field of view</td>
<td>For swipes, we advise to bring the hand from the side, in order to avoid generating a &quot;tap&quot; (not in the direction of the sensor)</td>
</tr>
<tr>
<td>Reset motion</td>
<td>Any Gesture finishes by a « reset » motion</td>
<td>We advise:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- To return the hand outside of the sensor field of view</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- To do a reset motion away from the sensor, moving the hand back in the user direction</td>
</tr>
<tr>
<td>Gesture Interval</td>
<td>It is the minimum delay between 2 consecutive gestures</td>
<td>Ideally, the minimum delay should be 1s to avoid overlap gesture in case of a different gesture (Ex: Swipe -&gt; TAP)</td>
</tr>
<tr>
<td>Hand palm posture</td>
<td>How is orientated the hand palm of the user?</td>
<td>Opening the hand and keeping it flat (parallel to the screen) during the gesture is bringing better results</td>
</tr>
<tr>
<td>User body position</td>
<td>Position of the user in the Field of View</td>
<td>We recommend:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- to keep the arm stretched to keep a minimum distance between the body and the hand</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Ideally the user body/head distance from the sensor is above 50cm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Do not keep the hand in the « gesture area » when no gesture is wanted</td>
</tr>
</tbody>
</table>
Thank you