# Screenful gesture library

**September 5th, 2014**

## Overview

Screenful gesture library utilizes `OpenNI 2.2` and `NiTE 2.2` middleware to send user interface commands to a browser running the Screenful dashboard software. This allows users to interact with the screen without external input devices. The gestures are detected by server software running on a computer with a depth sensor connected.

A browser can initiate a WebSocket connection to the gesture server, handle the available events as they are received from the server and carry out the UI commands in Javascript. The server uses `Java-WebSocket`[1] decorated with `ReconnectingWebSocket`[2] and operates stand-alone without additional web server software.

The gesture server has been developed for `Asus Xtion Pro`[3]. The server is run via `Java Service Wrapper`[4], which takes care of restarts on crashes, logging and integration into system services. Configuration of parameters and library paths is done via `server.conf` and `libpaths.conf`.

Since the server operates over a WebSocket, it can control multiple browsers running anywhere on the network. There is also no practical limitation on the number of hands tracked simultaneously as long as they fit in the field of view of the sensor.

The messages sent over the WebSocket are simple strings, like "left", "right", "out", "user-exit". The server program is accompanied with a configuration file that allows setting which directional gestures are recognized and sent and which are used to stop interaction. The server also sends events when hand tracking starts and stops. These messages should be handled by Javascript code in the browser.

For hand tracking to work optimally, the sensor should face the user from about two meters away without too much tilt. The user should have the hand upright with the palm facing the sensor and fingers together during interaction.

## Server appliance hardware

The complete prototype system consists of:

- Intel NUC DN2820FYKH
  - 4 GB of RAM
  - 60 GB SSD
- Asus Xtion Pro sensor
- A customized, clonable "kiosk" type `Ubuntu Server 14.04 LTS` configuration that starts the Chromium browser with a certain application URL to pass the gestures to.

## WebSocket events

Events are configured in `server.conf`. The user can set the directions that should be used for sending directional events, the directions that are used for exiting interaction and the gestures ("wave" or "click") to be used for starting hand tracking.

The server only sends data and does not respond to messages sent to it. It would be useful if the UI code could change the tracker parameters or request a depth image or other data from the server, but this is unimplemented as of now. To add such support, one should look in `GestureServer` (which extends `WebSocketServer`) and implement the `onMessage` method in some appropriate way. The Java-WebSocket chat server examples[5] will help.

[1] http://java-websocket.org/
[2] https://github.com/joewalnes/reconnecting-websocket
[3] The NiTE hand tracker did not seem to work with the Kinect.
[4] http://wrapper.tanukisoftware.com/doc/english/download.jsp
[5] https://github.com/TooTallNate/Java-WebSocket/blob/master/src/main/example/ChatServer.java

| Event | String | Description |
|---|---|---|
| Start hand tracking | hands-start | Sent when a start gesture is recognized and hand tracking starts |
| Stop hand tracking | hands-stop | Sent when hand tracking stops, will also be sent after user-exit |
| User exit | user-exit | Sent if an "exit" motion was recognized |
| Left | left | Sent when a directional gesture to the left is recognized |
| Right | right | Sent when a directional gesture to the right is recognized |
| Up | up | Sent when a directional gesture upwards is recognized |
| Down | down | Sent when a directional gesture downwards is recognized |
| In | in | Sent when a directional gesture towards the sensor is recognized |
| Out | out | Sent when a directional gesture away from the sensor is recognized |

## Using with Javascript

The file `cube.html` contains an example implementation for pushing a 3D cube around with gestures. The required steps are connecting to the server WebSocket and defining event handlers that parse the messages into meaningful action.

To create a ReconnectingWebSocket in Javascript:
```
socket = new ReconnectingWebSocket("ws://localhost:8887/");
```

Typically the gesture server would run on the same host the browser runs on and the UI code creating the socket should point to the localhost address. By default the gesture server binds to the localhost address which means it will not be accessible externally. The listening address can be changed in the server configuration file. A `ReconnectingWebSocket` is used so the client keeps trying to reconnect to the server if a disconnect occurs because of a network problem or the server becoming unreachable otherwise.

In cases where multiple applications running on different computers wish to access the same sensor data, the server listening address should be set to an IP the clients can contact. In a real-world scenario - using IP addresses instead of hostnames - this might mean that there is a gesture server computer (192.168.1.20) within a (virtual) local area network (subnet 192.168.1.0/24) and several client computers (192.168.1.[1-15]) where the browser application connects to this common gesture server.

The applications running on the computers could be different but they will all receive the same events from the server. To create such a scenario, the application webpages should contain the following line:
```
socket = new ReconnectingWebSocket("ws://192.168.1.20:8887/");
```

The port number can also be configured in server settings and should be allowed in any firewalls for outbound connections from the clients.

## Reading events in Javascript

To read gesture events after connecting the WebSocket (`socket`) one should define an onmessage function:
```
socket.onmessage = function(a) { ...  }
```

Simple example:

```
function startClient() {
    socket = new ReconnectingWebSocket("ws://localhost:8887/");
    socket.onmessage = function(msg) {
        switch(msg.data) {
            case "hands-start": ... ; break;
            case "hands-stop": ... ; break;
            case "left": ... ; break;
            case "right": ... ; break;
            ...
        }
    }
}

$(document).ready(function(){
    startClient();
});
```

## Gesture recognition

Gesture recognition is implemented in a simple way by tracking movement between frames - not depth frames but hand and user tracker frames from NiTE. A NiTE hand tracker frame is a data container of all currently detected hands (and other information) that the library has recognized from the depth data. Likewise user tracker frames contain all currently tracked users and their skeleton joint positions. The NiTE library calculates frames in real-time as the depth frames arrive from the sensor.

The hand tracker is used to supply real world 3D coordinates of tracked hands relative to the sensor in millimeters. The user should wave or push with their palm towards the sensor to start tracking. These are "start gestures" and the NiTE library supports two: "**click**" or "**wave**". The coordinates of all tracked hands are inspected between two consecutive frames. The `Gesture` class implements keeping track of how many frames a motion has been performed for and it uses a class implementing the `Detector` interface to determine if the movement was appropriate.

### Detector interface

In the simple case of directional gestures, a `DirectionDetector` compares the motion vector between the hand points of current and the previous hand tracker frame, and determines whether a hand was moving mainly in some of the cardinal directions (`left` / `right` / `up` / `down` / `in` / `out`). The DirectionDetector returns a boolean value signifying whether the movement between the current and the last frame meets the criteria of the parameters and is along one of the axes. A Gesture calls the detector to inspect the frames and should "true" be returned the desired number of times - parameter `travelframes` - the Gesture triggers and notifies its listeners that a directional gesture has been recognized.

In practice this means that for each frame the DirectionDetector tells if the movement was big enough and along one of the defined directions, and when this has happened enough times in a row, the hand is considered to have performed a directional gesture.

### Parameters and performance

The parameters that can be adjusted are listed below. The recommended values in parenthesis are suitable for a presentation application that detects fairly short motions with some cooldown time between the gestures.

- `startdelay (800 ms)`
  - initial delay after hand tracking starts and gestures events are sent to avoid accidental gestures
  - in milliseconds
- `traveldistance (10 mm)`
  - minimum distance a hand needs to move per frame to be considered moving
  - in millimeters
- `travelframes (5)`

  - number of frames the movement needs to continue to the same cardinal direction

- `cooldown (500 ms)`
  - the cooldown time after a gesture event is sent until new ones
  - in milliseconds

NiTE should be able to process depth frames in real-time as they're received, so assuming 30 depth frames per second, setting the "travel distance" to 10 millimeters and "travel frames" to 15 would mean that a hand should move in the same cardinal direction relative to the camera for at least 150 millimeters within half a second. Should the motion stop at some point or start going in the wrong direction, the Gesture starts counting from zero until the movement meets the criteria again.

The number of "travel frames" directly affects the length of the detection period and should be kept fairly low for fast swipes and flicking motions. Using too long a detection period can be straining for the user and look awkward. With very short travel distance and a small number of frames even the minute motions of the hand will trigger the gesture. Using a short cooldown time such as below 50 milliseconds will make the control feel very responsive, which may be useful if the application requires sliders or similar continuous motion.

The intermediate tracking object, **`NiTETracker`** keeps track of both hand and user trackers and can be queried for any tracking information. It also handles USB disconnects and waits for the sensor to be plugged back in again. The skeleton joints could be used with the gesture detection similarly to hand coordinates, but as of now a detector for it is unimplemented due to the hand tracker being more suitable for this application.

After creating a NiTETracker object, you can add a **`HandsListener`**, **`BonesListener`** or **`TrackingListener`** to it to receive hand tracker frames, user tracker frames or notifications of start/stop of hand tracking, respectively. To look for directional gestures from the hand tracker, one would create a Gesture, giving it a DirectionDetector in the constructor along with the parameters, then attach the Gesture to NiTETracker's hand listeners. Then a class implementing **`GestureListener`** can be added to the gesture's listeners and receive direction events as they're recognized.

## Visualizing

The library allows rendering a NiTETracker's hand and user tracker information in a GUI window. The rendering methods are adapted from NiTE examples. Graphical windows can be spawned to show the depth image overlaid with tracker data. The hand tracker window will show the depth image and draw a blue box centered on any tracked hand. The skeleton tracker window also shows the depth image and draws stick figures of the skeletons it recognizes.

Assuming hand and user tracker is enabled[6] a GUI window for both can be spawned:

```
// Create a NiTETracker
NiTETracker tracker = new NiTETracker(new TrackerSettings());
// Add visualizations
BonesVisualization bones = new BonesVisualization(tracker, "Skeletons");
HandsVisualization hands = new HandsVisualization(tracker, "Hands");
// Show visualizations
bones.show();
hands.show();
// The program will run until both windows
// have been closed (with Escape)
```

This piece of code also works as a "hello world" program for the library and is the content of the **`main`** method in **`screenful.testapps.BonesAndHandsViewer`**.

[6] By default the NiTETracker is created with both hand and skeleton tracker enabled, but the `GestureServer` disables skeleton tracking because it is not used in the application.

# API for public class methods

For readability, classes and methods that contribute to the library generally use "Hands" and "Bones" in their names when referring to "hand tracking" and "skeletal tracking" functionality to distinguish them from "Hand"/"User"/"Skeleton" used in NiTE naming scheme. Accessors will work as expected, ie. `NiTETracker.getUserTracker()` will return a NiTE `UserTracker.`

The source code contains more in-depth information about each class. Only methods are included in this listing - container classes such as TrackerSettings have public fields that can be set to change the settings, but for brevity these are omitted. The settings classes are somewhat convoluted.

## Package screenful.server

**`GestureServer`** `extends` **`WebSocketServer`**

The main application for browser communication. Implements a WebSocket server for a browser to connect to and passes detected gestures to the UI.

| Method | Description |
|---|---|
| `GestureServer` | `String address, int port : (constructor)` |
| | Creates a new gesture server listening on the specified address and port. |
| `main` | `String[] args : void` |
| | Initializes NiTETracker and gesture detection and starts listening for web socket connections. |
| `onClose` | `WebSocket conn, int code, String reason, boolean remote : void` |
| | Actions to be done when a socket closes. |
| `onError` | `WebSocket conn, Exception ex : void` |
| | Actions to be done when an error occurs with the WebSocket. |
| `onMessage` | `WebSocket conn, String message : void` |
| | Actions to be done when a message is received. |
| `onOpen` | `WebSocket conn, ClientHandShake handshake : void` |
| | Actions to be done when a connection is opened. |

**`GestureServer.Messenger`** `implements` **`WebSocketServer`**

Inner class of GestureServer sends the command to the browser via WebSockets.

| Method | Description |
|---|---|
| `Messenger` | `NiTETracker tracker, int startdelay : (constructor)` |
| | Create a new Messenger specifying the NiTETracker to use and the time to wait after tracking has started before sending events. |
| `onGesture` | `Displacement Gesture : void` |
| | When a detected gesture notifies the Messenger, it checks whether the gesture is assigned to exiting interaction or providing input and either stops hand tracking or sends the appropriate messages over WebSockets. |

### GestureSettings

Directional gesture detection parameter container.

| Method | Description |
|---|---|
| `GestureSettings` | `no arguments : (constructor)` |
| | Creates empty settings. |
| `GestureSettings` | `int travelDistance, int travelFrames, int cooldown, int startDelay, HashSet<CardinalDirection> exitDirections, HashSet<CardinalDirection> enabledDirections : (constructor)` |
| | Create new settings with specified values. |

### ServerSettings

ServerSettings class for GestureServer. Reads and writes a configuration file and allows accessing the fields via a Properties object.

| Method | Description |
|---|---|
| `ServerSettings` | `no arguments : (constructor)` |
| | Create default settings. |
| `ServerSettings` | `String filename : (constructor)` |
| | Load settings from a file. |
| `save` | `no arguments : void` |
| | Save settings into a file. |

## Package screenful.basic

### interface BonesListener

Any class that wants skeleton data should implement this interface and add itself to NiTETracker's listeners.

| Method | Description |
|---|---|
| `onNewBonesFrame` | `UserTrackerFrameRef frame : void` |
| | Override to implement behavior using the user tracker frame data. |

### interface HandsListener

Any class that wants hand data should implement this interface and add itself to NiTETracker's listeners.

| Method | Description |
|---|---|
| `onNewHandsFrame` | `HandTrackerFrameRef frame : void` |
| | Override to implement behavior using the hand tracker frame data. |

**NiTETracker** implements **HandTracker.NewFrameListener, UserTracker.NewFrameListener OpenNI.DeviceDisconnectedListener, OpenNI.DeviceConnectedListener**

Tracker for NUI features. When a NiTETracker is created, it creates instances of **UserTracker** and **HandTracker**, starts them and adds itself to their listeners to get notified of new data.

| Method | Description |
|---|---|
| NiTETracker | **TrackerSettings settings : (constructor)**<br>Initialize OpenNI and NiTE, create user and hand trackers. A watchdog is started to make sure the depth stream actually starts or the program exits (to be respawned again). |
| addBonesListener | **BonesListener listener : void**<br>Add a listener for new user frames. |
| addHandsListener | **HandsListener listener : void**<br>Add a listener for new hand frames. |
| addTrackerListener | **TrackerListener listener : void**<br>Add a listener for hand tracker start/stop events. |
| forgetHand | **short id : void**<br>Stop tracking a specific hand ID. |
| forgetHands | **no arguments : void**<br>Stop all hand tracking. |
| getBones | **no arguments : List<UserData>**<br>Return list of detected skeletons. |
| getBufferedImage | **no arguments : BufferedImage**<br>Access current depth image frame. |
| getHandFrame | **no arguments : HandTrackerFrameRef frame**<br>Access current hand tracker frame. |
| getHandTracker | **no arguments : HandTracker**<br>Return hand tracker for reading hand positions and gestures. |
| getHands | **no arguments : List<HandData>**<br>Return list of detected hands. |
| getLastHandTrackingStartTime | **no arguments : long**<br>Return the last time hand tracking was started. |
| getTrackedHands | **no arguments : List<HandData>**<br>Return a list of all hands that are currently being tracked. |
| getUserFrame | **no arguments : UserTrackerFrameRef**<br>Access current user tracker frame. |
| getUserTracker | **no arguments : UserTracker**<br>Return user tracker for reading user poses and skeletal data. |
| onDeviceConnected | **DeviceInfo di : void**<br>Actions to be done when a sensor connects. |
| onDeviceDisconnected | **DeviceInfo di : void**<br>Actions to be done when a sensor disconnects. |
| onNewFrame | **HandTracker ht : void**<br>Handle hand tracker frame. |
| onNewFrame | **UserTracker ut : void**<br>Handle user tracker frame. |
| removeAllListeners | **no arguments : void**<br>Remove all hands, bones and tracking listeners. |
| removeBonesListener | **BonesListener listener : void**<br>Remove a bones listener. |
| removeHandsListener | **HandsListener listener : void**<br>Remove a hands listener. |
| removeTrackingListener | **TrackingListener listener : void**<br>Remove a tracking listener. |

### `TrackerSettings`

Container class for NiTETracker settings.

| Method | Description |
|---|---|
| `TrackerSettings` | `no arguments : (constructor)` |
| | Create default settings - enable hand and user tracking with empty start gesture list. |
| `TrackerSettings` | `boolean handEnable, boolean skeletonEnable, HashSet<GestureType> startGestures : (constructor)` |
| | Create settings with hand and/or user tracker enabled and a set of start gestures defined (click / wave). |

### `interface TrackingListener`

As NiTE hand tracker lacks its own mechanism of notifying when tracking starts or stops, a class can implement TrackingListener to get the events.

| Method | Description |
|---|---|
| `onHandTrackingStarted` | `no arguments : void` |
| | Called when hand tracking has been started. |
| `onHandTrackingStopped` | `no arguments : void` |
| | Called when hand tracking stops (there are no more tracked hands). |

## Package screenful.gestures

### `Gesture` implements `HandsListener, BonesListener`

Gesture implements a generic gesture that notifies its listeners when a gesture has been detected for long enough.

| Method | Description |
|---|---|
| `Gesture` | `Detector detector, int framecount, int cooldown : (constructor)` |
| | Create a new gesture. |
| `addListener` | `GestureListener listener : void` |
| | Add a listener for the gesture. |
| `onNewBonesFrame` | `UserTrackerFrameRef frame : void` |
| | Handle user frames. |
| `onNewHandsFrame` | `HandTrackerFrameRef frame : void` |
| | Handle hand frames. |

### `interface GestureListener`

A class that wants notifications when a gesture is detected should implement GestureListener and add itself to a gesture's listeners.

| Method | Description |
|---|---|
| `onGesture` | `Displacement gesture : void` |
| | Override to implement behavior with the displacement data. |

### JointMetrics

Static methods for getting skeleton measurements etc.

| Method | Description |
|---|---|
| `jointToJointDistance` | `UserData user, JointType from, JointType to : double` |
| | Calculate euclidean distance between two joints (`JointType.RIGHT_HAND` and `JointType.NECK` for example) of a user in space. |

### Poses

Some examples of basic poses that can be calculated in a single frame based on user data.

| Method | Description |
|---|---|
| `dorkyClick` | `UserData user : boolean` |
| | "Click" by essentially clapping the hands in front of the face or otherwise bringing them close enough. Returns true if hands were above the neck and distance between the hands was less than 150 mm. |
| `handsAboveNeck` | `UserData user : boolean` |
| | Returns true if a user's both hands are above the neck. |

### Utilities

Some examples of basic poses that can be calculated in a single frame based on user data.

| Method | Description |
|---|---|
| `convertPoint` | `Point3D nitepoint : javafx.geometry.Point3D` |
| | Convert a NiTE Point3D into a JavaFX Point3D. For readability, round coordinates. |
| `determineCardinalDirection` | `javafx.geometry.Point3D vector, int minSensitivity : CardinalDirection` |
| | Detect general direction of a vector. |
| `displacementVector` | `Point3D from, Point3D to : javafx.geometry.Point3D` |
| | Return displacement vector from starting point to endpoint (simple subtraction). Note that the returned Point3D is JavaFX, the handled Point3Ds are from NiTE. |
| `distance3d` | `Point3D from, Point3D to : double` |
| | Calculate euclidean distance between two 3D points. |

## Package screenful.gestures.detectors

### ConsecutiveFrames

A container object for passing two consecutive hand and user tracker frames.

| Method | Description |
|---|---|
| `ConsecutiveFrames` | `HandTrackerFrameRef handsFrame, HandTrackerFrameRef previousHandsFrame, UserTrackerFrameRef bonesFrame, UserTrackerFrameRef previousBonesFrame : (constructor)` |
| | Create a new object with current and previous hand and user frames. |

## interface Detector

Interface for detectors, ie. an object that determines whether the movement during two consecutive frames was appropriate (true / false).

| Method | Description |
|---|---|
| `detected` | `Displacement gesture : boolean` |
| | Override to implement behavior. |
| `getData` | `no arguments : Displacement` |
| | Return the displacement. |

## DirectionDetector implements Detector

Detects hand point movement direction by calculating the displacement vector of each hand point between two consecutive frames. When any tracked hand performs a big enough movement towards the cardinal directions, the `detected(..)` method returns true. This class is used by the Gesture class for defining the logic of detecting a gesture.

| Method | Description |
|---|---|
| `DirectionDetector` | `int sensitivity : (constructor)` |
| | Create a new direction detector. Sensitivity is in millimeters, the server parameter `traveldistance` is used here. |
| `detected` | `Displacement gesture : boolean` |
| | Return true when movement was detected along the cardinal directions. |
| `getData` | `no arguments : Displacement` |
| | Return the displacement. |
| `getSensitivity` | `no arguments : int` |
| | Return the sensitivity setting. |
| `setSensitivity` | `int sensitivity : void` |
| | Set the sensitivity. |

## Displacement

Extra data related to the movement of a point (joint, hand) between two frames. Stores the general direction and the actual vector.

| Method | Description |
|---|---|
| `Displacement` | `CardinalDirection direction : (constructor)` |
| | Create a new displacement with only a general direction. |
| `Displacement` | `Point3D directionVector, CardinalDirection direction : (constructor)` |
| | Create a new displacement with a vector and the direction. |
| `Displacement` | `Point3D directionVector, CardinalDirection direction, short id : (constructor)` |
| | Create a new displacement with a vector, direction and an ID number. |
| `getDirection` | `no arguments : CardinalDirection` |
| | Return the general direction of the displacement. |
| `getDirectionVector` | `no arguments : Point3D` |
| | Return the direction vector. |
| `getId` | `no arguments : short` |
| | Return the ID number. |
| `setDirection` | `CardinalDirection dir : void` |
| | Set the general direction. |
| `setDirectionVector` | `Point3D directionVector : void` |
| | Set the direction vector. |

## Package screenful.gui

**`GenericWindow`** implements **`Runnable`**

Generic frame for displaying graphics. Can be closed by pressing Escape.

| Method | Description |
|---|---|
| `GenericWindow` | `String name : (constructor)` |
| | Create new window with a chosen title. |
| `run` | `no arguments : void` |
| | Keeps the window running until it is closed. |

## Package screenful.gui.rendering

**`BonesRenderer`** extends **`Component`** implements **`BonesListener`**

Draw stick characters from skeleton data on top of depth image. Adapted from NiTE examples.

| Method | Description |
|---|---|
| `BonesRenderer` | `UserTracker skel : (constructor)` |
| | Create a new renderer for a UserTracker. |
| `onNewBonesFrame` | `UserTrackerFrameRef frame : void` |
| | Handle the user frame. |
| `paint` | `Graphics g : void` |
| | Draw image and skeletons. |

**`DirectionRenderer`** extends **`Component`** implements **`GestureListener`**

Draw some feedback to directional gestures, ie. text signifying the detected direction.

| Method | Description |
|---|---|
| `DirectionRenderer` | `no arguments : (constructor)` |
| | Create a new renderer for directional gestures. |
| `onGesture` | `Displacement gesture : void` |
| | Reads the direction from the gesture. |
| `paint` | `Graphics g : void` |
| | Draw image. |

**`HandsRenderer`** extends **`Component`** implements **`HandsListener`**

Draw boxes over tracked hands in the depth image. Adapted from NiTE examples.

| Method | Description |
|---|---|
| `HandsRenderer` | `HandTracker hands : (constructor)` |
| | Create a new renderer for hand tracker data. |
| `onNewHandsFrame` | `HandTrackerFrameRef frame : void` |
| | Handles the hand frame and calls `repaint`. |
| `paint` | `Graphics g : void` |
| | Draw image. |

## Package screenful.gui.visualization

**`BonesVisualization`** extends **`GenericWindow`** implements **`Visualization`**

Skeleton tracker visualization window.

| Method | Description |
|---|---|
| `BonesVisualization` | `NiTETracker tracker, String name : (constructor)` |
| | Create a new visualization window for the user tracker with a custom title. |
| `show` | `no arguments : void` |
| | Makes the window visible. |

**`DirectionVisualization`** extends **`GenericWindow`** implements **`Visualization`**

Directional gesture visualization window.

| Method | Description |
|---|---|
| `DirectionVisualization` | `Gesture direction, String name : (constructor)` |
| | Create a new visualization window for directional gestures with a custom title. |
| `show` | `no arguments : void` |
| | Makes the window visible. |

**`HandsVisualization`** extends **`GenericWindow`** implements **`Visualization`**

Hand tracker visualization window.

| Method | Description |
|---|---|
| `HandsVisualization` | `NiTETracker tracker, String name : (constructor)` |
| | Create a new visualization window for the hand tracker with a custom title. |
| `show` | `no arguments : void` |
| | Makes the window visible. |

**`interface Visualization`**

Interface for classes that provide some sort of graphical presentation of the sensor data.

| Method | Description |
|---|---|
| `show` | `no arguments : void` |
| | Makes the window visible. |

## Package screenful.testapps

### `BonesAndHandsViewer`

Shows two graphical windows, one for hand tracker and one for user tracker. Program exits when both windows have been closed with Escape.

| Method | Description |
|---|---|
| `main` | `String[] args : void` |
| | Runs the program, spawning two GUI windows for the tracker data. |

### `HandsDirectionViewer`

Show the hand tracker visualization and general direction of tracked hand movement.

| Method | Description |
|---|---|
| `main` | `String[] args : void` |
| | Runs the program, spawning two GUI windows for the tracker data. |