Screenful gesture project

Enables natural user interaction for the Screenful dashboard software. The user can change the active slide and get more detail by making simple hand gestures.

Implemented using OpenNI 2.2 and NiTE 2.2 for 64-bit Linux.

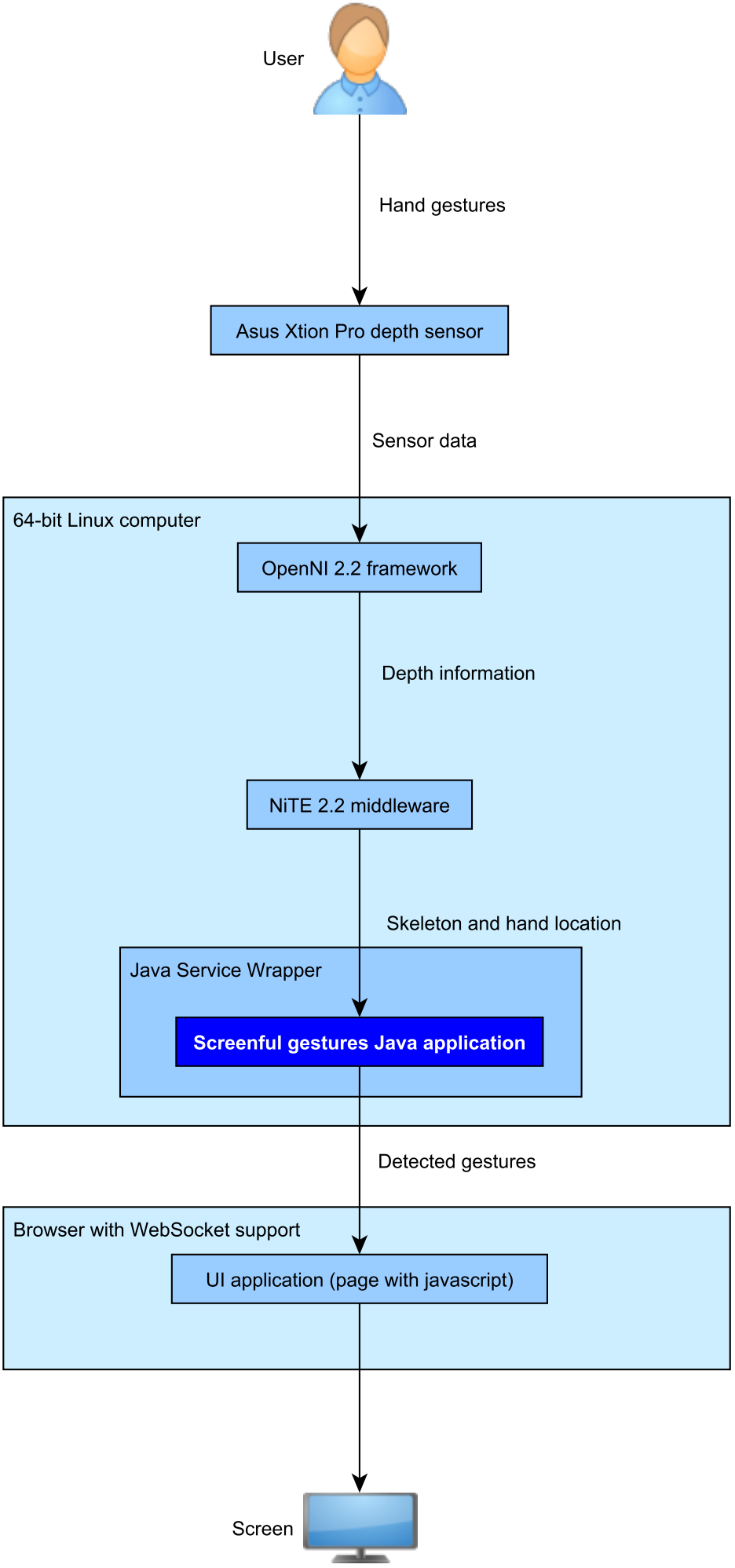
High-level overview

Hardware

- Xtion Pro depth sensor
- 64-bit Linux computer
    - There is no NiTE 2.x for the ARM architecture

Main software component breakdown

- **Client side:**
    - Browser with WebSocket support
    - Page with javascript to handle UI effects of gesture messages
- **Server side:**
    - Java Service Wrapper
        - Used to daemonize the server and handle JVM crashes and other problems that seem to arise with JNI and NiTE
    - server.conf contains server settings
    - libpaths.conf allows to specify an optional path to OpenNI/NiTE libraries
- **Internals:**
    - GestureServer
        - Main server program, listens for WebSocket connections
        - Spawned by the service wrapper
        - Initializes the tracker
        - Detected gestures are sent to browsers connected via WebSocket
    - NiTETracker
        - Main tracker object, keeps track of hands and skeletons in the scene and listeners can attach to it
        - As a listener to NiTE user and hand tracker classes, receives hand and user tracker frames as they're processed by the library
        - Manages the sensor and attempts to detect and handle USB disconnects
    - Gesture
        - Defines a gesture using a user-defined Detector. A gesture is defined to be great enough movement of the hand that satisfies the condition checked by the Detector for a period of frames
        - In the GestureServer, a DirectionDetector is used for frame comparison, allowing for sensitivity adjustable left/right/up/down/in/out gestures
    - DirectionDetector
        - Uses NiTETracker to supply hand tracker frames and compares the relative movement of two consecutive frames to determine which direction the hand is currently moving


User
Hand gestures
Asus Xtion Pro depth sensor
Sensor data
64-bit Linux computer
OpenNI 2.2 framework
Depth information
NiTE 2.2 middleware
Skeleton and hand location
Java Service Wrapper
Screenful gestures Java application
Detected gestures
Browser with WebSocket support
UI application (page with javascript)
Screen

Screenful gesture library

Overview

Screenful gesture library utilizes **`OpenNI 2.2`** and **`NiTE 2.2`** middleware to implement prototype software for sending 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] and operates stand-alone without additional web server software. The client code in the browser is decorated with **`ReconnectingWebSocket`**[2] to enable reconnecting after disconnects.

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.

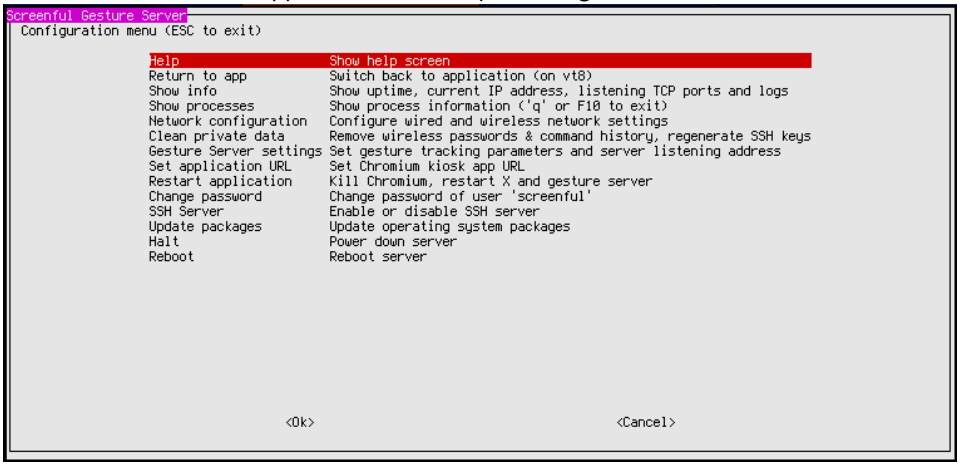
The messages sent over the WebSocket are simple strings, like “left”, “right”, “out”, “user-exit”. The server program is accompanied by a configuration file that allows setting which directional gestures are recognized and sent and which are used to stop interaction. The server also sends events when 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. Screenshot of the configuration menu:



[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

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.

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 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 server listening address should be set to `192.168.1.20` and 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 outbound connections from the clients should be allowed through any firewalls.

Reading events in Javascript

To read gesture events after connecting the WebSocket (`socket`) one should define an `onmessage` function:

[5] https://github.com/TooTallNate/Java-WebSocket/blob/master/src/main/example/ChatServer.java

```
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 which cardinal direction (`left` / `right` / `up` / `down` / `in` / `out`) the hand is moving. The DirectionDetector returns a boolean value signifying whether the magnitude of the movement between the current and the previous frame is greater than `traveldistance` and provides the direction. A Gesture calls the detector to compare every frame to the previous one 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 what the direction was, and when this has happened enough times in a row, the hand is considered to have performed a directional gesture. The actual gesture direction is determined when the last required frame is handled - that is, there is no interpolation or smoothing used but in practice it works, since with typical parameters the magnitude of the movement is great enough to trigger the gesture in the middle of the motion.

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 gesture 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 for a successful gesture

- `cooldown (500 ms)`
  - the cooldown time between sending gesture events
  - 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 relative to the camera for at least 150 millimeters within half a second. Should the motion stop at some point, the Gesture starts counting from zero until the movement meets the criteria again.

The parameters "travel frames" and "travel distance" should be kept fairly low for fast swipes and flicking motions. Using too long motions 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 gestures. 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`**.

Bugs and shortcomings

- The code base needs refactoring, settings are scattered and functionality dispersed
- Server does not listen to messages sent by the clients
  - If implemented, the client could for example request changing of parameters

[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.

- Sent events are minimalistic and do not convey extra information such as the location where hand tracking starts
- Needs a lot more testing, littered with bugs and oversights
  - Configuration file parsing is flaky and may be prone to errors
- Gesture detection does not utilize skeleton tracking
  - The DirectionDetector could be generalized to work with skeleton joints
- The detection method is crude and simple
  - Could smooth out movement and calculate a running average for the vector
  - Should compare the motion to a vector instead of having predefined cardinal directions
  - Could use a completely different detection scheme
- If the sensor is at an angle, the user needs to perform the motions at an angle so they're aligned with the axes from the sensor's perspective
  - Could have some automatic calibration to alleviate
- NiTE library is unstable and prone to various crashes in native code
  - No source available, unfixable, alleviated with Java Service Wrapper, using a watchdog for the depth stream and tweaking heap size
  - The entire library should be reimplemented on top of an open source equivalent when such becomes available

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 sends strings of the detected gestures to the UI running in the browser.

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)** 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** Add a listener for new user frames.
addHandsListener	**HandsListener listener : void** Add a listener for new hand frames.
addTrackerListener	**TrackerListener listener : void** Add a listener for hand tracker start/stop events.
forgetHand	**short id : void** Stop tracking a specific hand ID.
forgetHands	**no arguments : void** Stop all hand tracking.
getBones	**no arguments : List<UserData>** Return list of detected skeletons.
getBufferedImage	**no arguments : BufferedImage** Access current depth image frame.
getHandFrame	**no arguments : HandTrackerFrameRef frame** Access current hand tracker frame.
getHandTracker	**no arguments : HandTracker** Return hand tracker for reading hand positions and gestures.
getHands	**no arguments : List<HandData>** Return list of detected hands.
getLastHandTrackingStartTime	**no arguments : long** Return the last time hand tracking was started.
getTrackedHands	**no arguments : List<HandData>** Return a list of all hands that are currently being tracked.
getUserFrame	**no arguments : UserTrackerFrameRef** Access current user tracker frame.
getUserTracker	**no arguments : UserTracker** Return user tracker for reading user poses and skeletal data.
onDeviceConnected	**DeviceInfo di : void** Actions to be done when a sensor connects.
onDeviceDisconnected	**DeviceInfo di : void** Actions to be done when a sensor disconnects.
onNewFrame	**HandTracker ht : void** Handle hand tracker frame.
onNewFrame	**UserTracker ut : void** Handle user tracker frame.
removeAllListeners	**no arguments : void** Remove all hands, bones and tracking listeners.
removeBonesListener	**BonesListener listener : void** Remove a bones listener.
removeHandsListener	**HandsListener listener : void** Remove a hands listener.
removeTrackingListener	**TrackingListener listener : void** 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.

Screenful gesture UI WebSocket server for NiTE 2.2

(Note: the data files for NiTE are not included, you must get them elsewhere. Why? NiTE's license.)

Requirements for usage

- 64-bit OpenNI 2.2 and NiTE 2.2 must be installed
  - see tools/kinect-ubuntu-install
- Copy NiTE2 data files (`h.dat`, `s.dat` etc.) into `server/Screenful-GestureServer/NiTE2/`
- Edit or verify `server.conf`
- Edit `libpaths.conf` to optionally fill in (or comment out) the path to OpenNI/NiTE libraries
- Run `./run.sh`
  - If OpenNI and NiTE libraries have been installed system-wide, you may not need to define the extra library path and can comment it out
  - This will start the server in console, to start it as a daemon, use `wrapper/bin/gestureserver start`
- Open a web page in a browser that establishes a WebSocket connection to the server (port 8887 by default), for a sample page see screenful-ui-test.html.
- Connect the Xtion if it's not already connected and wave your hand to the sensor to start tracking the hand and pass commands to the browser.
  - *Note that while installing the libraries above should add support for the Kinect, for some reason NiTE hand tracker doesn't seem to work with it. The skeleton tracking works fine. For this reason the software in its current state* ***is not usable with the Kinect****.*

Using in a NetBeans project

1. Create a new Java project
2. Add org.openni.jar, com.primesense.nite.jar, java_websocket.jar to project libraries
3. (If needed) Adjust project properties, Run section: add VM parameter: `-Djava.library.path=<absolute path of the directory with all the .so files>`
4. Copy `<NiTE2 extraction directory>/Redist/NiTE2` directory inside the project root directory (`Screenful-GestureServer/`) to get required data files
5. See and modify GestureServer.java.

Websocket server protocol

The gesture server sends notifications of the detected motion events to the browser(s) connected to it as configured in the configuration file.

Event configuration

The file `server.conf` has three related lines:

- `exitdirections=<direction,direction,... | none>`
  - When these directions are detected, a 'user-exit' event is sent
  - Example: `exitdirections=out`
- `enableddirections=<direction,direction,... | none>`
  - Specify which directions send events such as 'left' and 'down'
  - Example: `enableddirections=left,right,down`
- `startgestures=<wave | click | wave,click | none>`
  - Specify which gestures to use for starting hand tracking and send a 'hands-start' event
  - Choices are 'wave', 'click', a combination of these or 'none'
  - When hand tracking stops, either after 'user-exit' or when the hand is lost, a 'hands-stop' event is sent

Installing OpenNI2, libfreenect and the Kinect sensor on Ubuntu 14.04 (64-bit)

Note: openni.org referenced in some documentation is not online anymore.

- Note: you need to be running a 64-bit OS
- Commands are for Ubuntu 14.04 LTS or similar Debian based system.
- The script has been tested with Ubuntu 14.04 LTS booted in live mode.

Quick install with `kinect-ubuntu-install.sh`

This script automates all the manual steps in the next section. Note that it needs to be run with sudo. This ensures that also the non-root account gets all the required permissions.

Note: runs commands as root. Tries to be careful though.
Note: the script installs Oracle Java 8. If you already have javac, you can comment out the `install-java8` line

1. Make sure you're connected to internet and start a terminal
2. Run:

```
git clone https://github.com/Screenful/screenful-gestures
sudo screenful-gestures/tools/kinect-ubuntu-install/kinect-ubuntu-install.sh
```

3. Wait a while for everything to install and answer **Yes** when the Java installer asks you to accept the license, no other interaction is needed.

Manual install

Get prerequisites:

1. Install packages

```
sudo apt-get install git-core g++ cmake libudev-dev libxi-dev libxmu-dev
python libusb-1.0-0-dev libudev-dev freeglut3-dev doxygen graphviz
```

2. Get OpenNI2 library from https://github.com/OpenNI/OpenNI2

```
git clone https://github.com/OpenNI/OpenNI2.git
cd OpenNI2
# Save path for further reference
OPENNI_DIR="${PWD}"
```

3. Fix some issues with compiling (in OpenNI2 commit 7bef8f639e4d64a85a794e85fe3049dbb2acd32e)

```
## Comment out "treat warnings as errors" option in PSCommon Makefile
sed -i '/-Werror/ s/^/#/'
${OPENNI_DIR}/ThirdParty/PSCommon/BuildSystem/CommonCppMakefile
## Fix NiViewer Makefile (missing -lpthread in linker arguments)
echo "LDFLAGS += -lpthread" >>
${OPENNI_DIR}/Source/Tools/NiViewer/Makefile
make
```

4. Add udev rules to access the sensor as normal user

```
## Add udev rules for Primesense sensor device IDs
```

```
sudo ${OPENNI_DIR}/Packaging/Linux/install.sh
```

- libfreenect ships with 51-kinect.rules file, but it doesn't seem to work with Ubuntu 14.04.
- The following rules allow video group to access the Kinect components (Motor, Audio, Camera). Save in /etc/udev/rules.d/
  - **eg. /etc/udev/rules.d/51-kinect.rules**

```
# Rules for Kinect & Kinect for Windows' Motor, Audio and Camera
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02b0",
MODE:="0666", OWNER:="root", GROUP:="video"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ad",
MODE:="0666", OWNER:="root", GROUP:="video"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ae",
MODE:="0666", OWNER:="root", GROUP:="video"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02c2",
MODE:="0666", OWNER:="root", GROUP:="video"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02be",
MODE:="0666", OWNER:="root", GROUP:="video"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02bf",
MODE:="0666", OWNER:="root", GROUP:="video"
```

5. Join user to group

```
## Add current user to video group (specified in udev rules)
sudo gpasswd -a ${USER} video
```

6. Install Oracle java8 (optional, but compiling OpenNI2 Java wrappers requires javac):

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

7. For Kinect support (Freenect bridge driver for Kinect to work under Linux/OSX):

   - https://github.com/OpenKinect/libfreenect/tree/master/OpenNI2-FreenectDriver
   - compile and copy driver to OpenNI2 directory:

```
git clone https://github.com/OpenKinect/libfreenect.git
cd libfreenect
mkdir build; cd build
cmake .. -DBUILD_OPENNI2_DRIVER=ON
make
cp -L lib/OpenNI2-FreenectDriver/libFreenectDriver.so
${OPENNI_DIR}/Bin/x64-Release/OpenNI2/Drivers/
```

8. If all went okay, you should now be able to plug in a Kinect / Xtion / Primesense sensor and run the example programs:

```
cd ${OPENNI_DIR}/Bin/x64-Release/
./NiViewer
```

9. To add the various .so library files to system wide configuration:

```
sudo echo "${OPENNI_DIR}/Bin/x64-Release/" >
/etc/ld.so.conf.d/openni2.conf
sudo ldconfig
```

- **Press '?' in NiViewer to get help and try out the various modes.**
- **To start recording sensor data into a file, press 'c'. To stop recording, press 'x'. A .oni file is generated in the working directory.**

Troubleshooting

- **Installation failed and I can't remove the leftovers**

Use sudo to remove the files - they were copied there as root

- **The sensor is not found**
  - Try unplugging the sensor and plugging it back again, then wait for some seconds (10 to be sure).
  - Check last lines of `dmesg` output to see if the device shows up
  - Try running the program with sudo
    - If it works then, make sure you've either logged out and back in again after you added your user to the `video` group, or type `newgrp video` and see if you can then run it as normal user.
- **Something fails with the compile**
  - The script makes some fixes to build files and these may be very specific to a certain commit. If you uncomment the lines with `NICOMMIT=...` and `FREENECTCOMMIT=...` you can specify which version to checkout for the builds.
    - In `kinect-ubuntu-install.sh`:

```
# uncomment to check out the versions this script was written for
#NICOMMIT="7bef8f639e4d64a85a794e85fe3049dbb2acd32e"
#FREENECTCOMMIT="cb0254a10dbeae8bdb8095d390b4ff69a2becc6e"
```

Installing the gesture server on Intel NUC DN2820FYKH

A preconfigured appliance image was created for this project, it is likely to become available at some point but is too big for pushing to this repository.

Hardware notes

- **Loosen the four screws holding the bottom plate and lift the drive bay**
    - Note the orientation: there is a bar code sticker on the bottom and this should be positioned towards the Ethernet port, otherwise the plate may bend when it's put back together.
- **Insert RAM in memory slot**
- **Insert 2.5" SSD in disk slot (secure with included two tiny screws)**
- **Put the drive bay back and fasten screws**
- **Connect NUC to a display, turn it on and enter BIOS setup by pressing F2**
    - **Disable "Wake on USB" from Power options and save the settings**
        - Without this setting it seems the Xtion wakes up the NUC from a powered down state.
- **You may want to update the BIOS**
    - You can use the option within BIOS (didn't work for me, it complained about not being able to read some product ID) or by copying the BIOS update from Intel's site to a USB stick, inserting the stick to the NUC and pressing F7 at bootup to select the file.
    - **However, HDMI output seems broken in versions below 0037!**
    - If you don't get an image after flashing the BIOS and rebooting, don't despair, just put another version (at the time of writing **0038** was the latest) to the USB stick and follow the update procedure blind. That is, boot with the F7 option, select the device and file, wait, and hopefully when it reboots you will get a picture.
        - **It is a good idea to rehearse the procedure or record a video of it to know how to do it blind.**

OS installation

- **Install Ubuntu Server 14.04 (64-bit).**
    - The server flavour is optional but the kiosk instructions below are for an Ubuntu **without the default desktop environment**.
    - Server installation process asks more questions than the desktop version, but most can be answered with a default or blank line.
        - Do not choose to encrypt the disk.
    - Create a user called **screenful**.
- **Install kernel 3.14.1**
    - Needed for Xtion to work with USB 3.0 controller.

        - You may also need to update the Xtion firmware!

```
wget http://kernel.ubuntu.com/~kernel-ppa/mainline/v3.14.1-
trusty/linux-headers-3.14.1-031401_3.14.1-031401.201404141220_all.deb
wget http://kernel.ubuntu.com/~kernel-ppa/mainline/v3.14.1-
trusty/linux-headers-3.14.1-031401-generic_3.14.1-
031401.201404141220_amd64.deb
wget http://kernel.ubuntu.com/~kernel-ppa/mainline/v3.14.1-
trusty/linux-image-3.14.1-031401-generic_3.14.1-
031401.201404141220_amd64.deb
sudo dpkg -i linux-headers-3.14.1-*.deb linux-image-3.14.1-*.deb
```

    - Reboot.

- **Install some needed packages**

    - This assumes server installation.

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install xserver-xorg xinit x11-apps chromium-browser
screen elinks unclutter oracle-java8-installer git-core nodm aosd-cat
```

    - screen, elinks, unclutter, nodm and aosd-cat are not required for the server but in the actual appliance image are utilities for the admin and allow automatic login, hiding the mouse cursor and showing messages in the GUI.

- **Add user to group 'video'**

```
sudo gpasswd -a screenful video
```

- **Add udev rules for the sensor**

    - Copy the file `55-primesense.rules` to `/etc/udev/rules.d/` to allow the 'video' group to access the sensor.

Gesture server installation

- **Create a library directory**

```
mkdir /home/screenful/libs
sudo echo '/home/screenful/libs' > /etc/ld.so.conf.d/openni2.conf
```

- **Extract the required files**

    - **You need to acquire the NiTE library files from somewhere, they are not included.**
        - Place `libNiTE2.jni.so` and `libNiTE2.so` in `/home/screenful/libs` (the directory created above).
        - Place the `NiTE2` directory as a subdirectory in `screenful-gestures/server/Screenful-GestureServer`.
    - Copy all OpenNI2 and NiTE2 .so files in the library directory.
    - Run `sudo ldconfig` to update shared library cache.

- **Edit libpaths.conf**

    - Edit `wrapper.java.library.path.2=/home/screenful/libs` to point to the library directory above.

- **The resulting library directory should look like this:**

```
libs
├── libMWClosestPoint.so
├── libNiTE2.jni.so
├── libNiTE2.so
├── libOniFile.so
├── libOpenNI2.jni.so
├── libOpenNI2.so
├── libPS1080.so
├── libPSLink.so
└── OpenNI2
    └── Drivers
        ├── libDummyDevice.so
        ├── libFreenectDriver.so
        ├── libOniFile.so
        ├── libPS1080.so
        └── libPSLink.so
```

Configuring the server

- **Edit `server.conf` to change tracking parameters and to specify gesture events.**

Running the server

- **Use `run.sh` to start the server in 'console' mode (foreground).**
- **To install the server as a system service, run `sudo wrapper/bin/gestureserver install`**
  - More information: http://wrapper.tanukisoftware.com/doc/english/launch-nix.html#boot
  - After installing the service, you can use it as a normal system service in Ubuntu, using `sudo service gestureserver start/stop/restart` to control its state.