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.

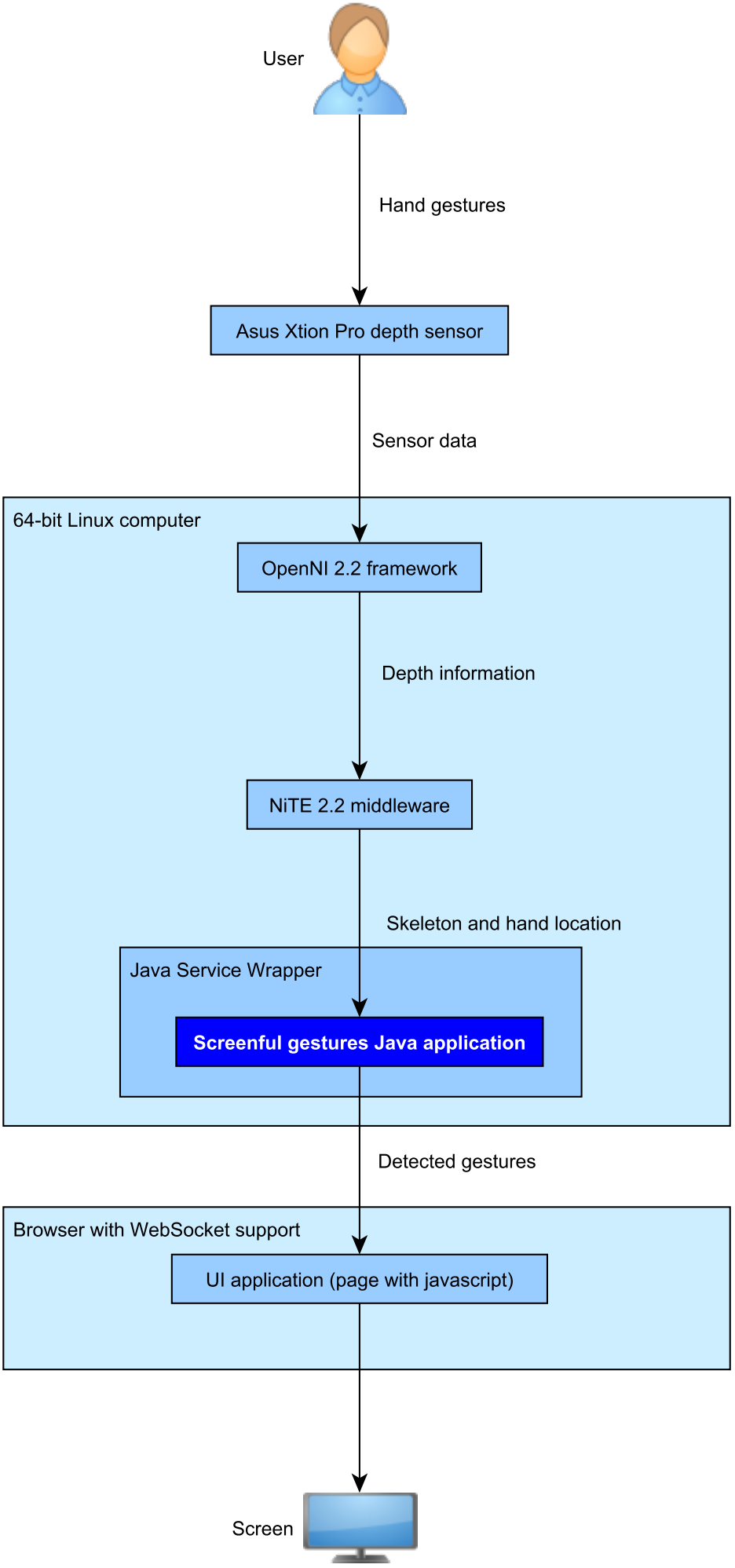
High-level overview

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 API

August 6th, 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 (`http://java-websocket.org/`) and operates stand-alone.

The gesture server has been developed for Asus Xtion Pro.[1] The server is run via Java Service Wrapper, 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 the Javascript in the browser.

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 hand tracker is used to supply coordinates of tracked hands - the user should wave or push with their palm at the sensor to start tracking - and these coordinates 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.

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). When the DirectionDetector has returned "true" enough times when a Gesture uses it, 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 which direction it was, and when this has happened enough times in a row, the hand has moved towards the direction. The parameters that can be adjusted is initial delay after hand tracking starts and gestures are sent (in milliseconds, to avoid accidental gestures after waving), the minimum distance that a hand needs to move per frame (in millimeters) to be considered moving, and the amount of frames the movement needs to continue to the same general direction.

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 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.

Public class members:

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"

[1] The NiTE hand tracker did not seem to work with the Kinect.

used in NiTE naming scheme. Accessors will work as expected, ie. NiTETracker.getUserTracker() will return a NiTE UserTracker.

screenful.server - main server program

- **GestureServer**: The main application for browser communication. Implements a WebSocket server for a browser to connect to and passes detected gestures to the UI.
  - **main(String[] args)**
    - Initializes NiTETracker and gesture detection and starts listening for web socket connections.
  - **onClose(WebSocket conn, int code, String reason, boolean remote)**
    - Actions to be done when a socket closes
  - **onError(WebSocket conn, Exception ex)**
    - Actions to be done when an error occurs with the WebSocket
  - **onMessage(WebSocket conn, String message)**
    - Actions to be done when a message is received
  - **onOpen(WebSocket conn, ClientHandShake handshake)**
    - Actions to be done when a connection is opened
- **GestureServer.Messenger**: Inner class of GestureServer sends the command to the browser via WebSockets.
  - **onGesture(Displacement gesture)**
    - 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.
- **Settings**: A settings object for the GestureServer.
  - **Settings()**
    - Create default settings
    - Default filename "default.conf"
  - **Settings(String filename)**
    - Load settings from a configuration file
  - **save()**
    - Saves the current settings

screenful.basic - reading NiTE data

- **BonesListener (Interface)**: Any class that wants skeleton data should implement this interface and add itself to NiTETracker's bones listeners.
  - **onNewBonesFrame(UserTrackerFrameRef frame)**
- **HandsListener (Interface)**: Any class that wants hand data should implement this interface and add itself to NiTETracker's hands listeners.
  - **onNewHandFrame(HandTrackerFrameRef frame)**
- **TrackingListener (Interface)**: Listener interface to receive hand tracking start/stop events.
  - **onHandTrackingStarted()**
    - Called when hand tracking starts.
  - **onHandTrackingStopped()**
    - Called when hand tracking stops (there are no more tracked hands).
- **NiTETracker**: Tracker for NUI features (implements HandListener and SkeletonListener from the NiTE library).
  - **NiTETracker(boolean enableHands, boolean enableBones)**

    * Constructor, true will enable hand and/or user tracker and false will disable them.
- **addBonesListener(BonesListener listener) / removeBonesListener(BonesListener listener)**
    * Add/remove a listener for user tracking
- **addHandsListener(HandsListener listener) / removeHandsListener(HandsListener listener)**
    * Add/remove a listener for hand tracking
- **addTrackingListener(TrackingListener listener) / removeTrackingListener(TrackingListener listener)**
    * Add/remove a listener for hand tracker start/stop events.
- **removeAllListeners()**
    * Clears all listener lists.
- **forgetHand(short id)**
    * Stop tracking a specific tracked hand ID
- **forgetHands()**
    * Stop tracking all tracked hands
- **getBones()**
    * Return currently detected skeletons
- **getBufferedImage(): BufferedImage**
    * Return the buffered depth image
- **getHandFrame(): HandTrackerFrameRef**
    * Return current hand tracker frame
- **getHandTracker(): HandTracker**
    * Return the hand tracker
- **getHands(): List<HandData>**
    * Get hand tracking data
- **getTrackedHands(): List<HandData>**
    * Get a list of hands that are being tracked (a hand ID can be present but not tracked)
- **getUserFrame(): UserTrackerFrameRef**
    * Return current user tracker frame
- **getUserTracker(): UserTracker**
    * Return the user tracker
- **getBones(): List<UserData>**
    * Get skeleton data
- **onDeviceConnected(DeviceInfo di)**
    * Handles OpenNI's device state changes
- **onDeviceDisconnected(DeviceInfo di)**
    * Handles OpenNI's device state changes
- **onNewFrame(HandTracker ht)**
    * Handles hand tracker frames
- **onNewFrame(UserTracker ut)**
    * Handles user tracker frames

screenful.detectors - movement detection in frames

- **ConsecutiveFrames**: A container object for passing two consecutive hand and user tracker frames.
  - **ConsecutiveFrames(HandTrackerFrameRef handsFrame, HandTrackerFrameRef previousHandsFrame, UserTrackerFrameRef bonesFrame, UserTrackerFrameRef previousBonesFrame)**
- **Detector**: Interface for detectors, ie. an object that determines whether the movement of something during two consecutive (hand and/or skeleton) frames was appropriate (true / false).
  - **detected(ConsecutiveFrames frames): boolean**
  - **getData(): GestureData**
- **DirectionDetector**: Detects hand point movement direction by calculating the displacement vector of each hand point (could also use skeletons) between two consecutive frames. When any tracked hand performs a big enough movement (sensitivity set in millimeters), the detected(..) method returns true.
  - **getSensitivity(): int**
  - **setSensitivity(int sensitivity)**
  - **DirectionDetector(int sensitivity)**
    - constructor
  - **detected(ConsecutiveFrames frames): boolean**
  - **getData(): GestureData**

screenful.gestures - gesture detection based on movement

- **Gesture**: Gesture implements a generic gesture that notifies its listeners when a gesture has been detected for long enough.
  - **Gesture(Detector detector, int framecount, int cooldown)**
    - Create a new gesture using a specified Detector. Framecount is the amount of consecutive frames where the detector's condition should be true for the gesture to be recognized. Cooldown is in milliseconds and defines a period of inaction after a gesture has been successfully detected.
  - **addListener(GestureListener listener)**
    - Add a listener for the gesture
  - **onNewBonesFrame(UserTrackerFrameRef frame)**
    - Handle user tracker frames
  - **onNewHandsFrame(HandTrackerFrameRef frame)**
    - Handle hand tracker frames
- **GestureListener**: A class that wants notifications when a gesture is detected should implement GestureListener and add itself to the gesture's listeners.
  - **onGesture(GestureData gesture)**
    - Override to implement behavior
- **JointMetrics**: Static methods for getting skeleton measurements etc.
  - **jointToJointDistance(UserData user, JointType from, JointType to): double**
    - Returns euclidian distance in space between two skeletal joints, eg. JointType.RIGHT_HAND, JointType.NECK etc. in millimeters
- **Poses**: Methods to detect skeletal poses based on joint positions, eg. "hands above neck" etc.
  - **dorkyClick(UserData user): boolean**
    - Returns true if hands are 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 general utility methods

  - **convertPoint(Point3D nitepoint): javafx.Geometry.Point3D**
    * Convert a NiTE Point3D into javafx Point3D and round the coordinates to integers
  - **determineCardinalDirection(javafx.geometry.Point3D vector, int minSensitivity): CardinalDirection**
    * Return a cardinal direction that most closely matches the vector
  - **displacementVector(Point3D from, Point3D to): javafx.geometry.Point3D**
    * Calculate displacement vector between two points
  - **distance3d(Point3D from, Point3D to): double**
    * Return distance between two 3D points

screenful.gestures.detectors

- **ConsecutiveFrames**: A container for passing two consecutive hand and user tracker frames.
  - **ConsecutiveFrames(HandTrackerFrameRef handsFrame, HandTrackerFrameRef previousHandsFrame, UserTrackerFrameRef bonesFrame, UserTrackerFrameRef previousBonesFrame)**
- **Detector**: An interface for defining frame-to-frame detection logic.
  - **detected(ConsecutiveFrames frames): boolean**
    * Override. Should return true when the difference between frames is appropriate. A Gesture will use this boolean value to determine if a frame contributed to the gesture.
  - **getData(): Displacement**
    * Return last direction data.
- **DirectionDetector**: Detect movement in the cardinal directions.
  - **detected(ConsecutiveFrames frames): boolean**
    * Returns true when big enough movement is detected towards some direction.
  - **getData(): Displacement**
    * Return last direction data.
  - **getSensitivity(): int**
    * Return the chosen sensitivity.
  - **setSensitivity(int sensitivity)**
    * Set the sensitivity in millimeters.
- **Displacement**: Extra data related to a detected gesture.
  - **Displacement(CardinalDirection direction)**
  - **Displacement(Point3D directionVector, CardinalDirection direction)**
  - **Displacement(Point3D directionVector, CardinalDirection direction, short id)**
    * Constructors
  - **getDirection(): CardinalDirection**
    * Return cardinal direction of displacement (left/right/up/down/in/out)
  - **getDirectionVector(): Point3D**
    * Return 3D vector of displacement
  - **getId(): short**
    * Get the ID
  - **setDirection(CardinalDirection dir)**
    * Set the direction
  - **setDirectionVector(Point3D directionVector)**
    * Set the vector

screenful.gui - GUI windows

- **GenericWindow**: Generic frame for displaying graphics
  - **run()**

screenful.gui.rendering - graphics renderers for GUI windows

- **HandsRenderer**: Draw tracked hands (red rectangles when tracking) on top of depth image.
  - **onNewHandsFrame(HandTrackerFrameRef frame)**
  - **paint(Graphics g)**
    - Draw depth image and tracked hands.
- **BonesRenderer**: Draw stick characters from skeleton data on top of depth image.
  - **onNewBonesFrame(UserTrackerFrameRef frame)**
  - **paint(Graphics g)**
    - Draw depth image and skeletons.
- **DirectionRenderer**: Draw some feedback to directional gestures.
  - **onGesture(GestureData gesture)**
  - **paint(Graphics g)**
    - Draw text to indicate directions.

screenful.gui.visualization - containers for GUI windows

- **Visualization (Interface)**: Interface for classes that provide some sort of graphical presentation of the sensor data. A visualization will take a NiTETracker in its constructor to start listening to events.
  - **show()**
- **DirectionVisualization**: Directional gesture visualization window
- **HandsVisualization**: Hand tracker visualization window
  - **show()**
- **BonesVisualization**: Skeleton tracker visualization window
  - **show()**

screenful.testapps - applications for testing

- **BonesAndHandsViewer**: Views both input methods (skeleton / hand) simultaneously.
  - **main(String[] args)**
- **HandsDirectionViewer**: Show general direction of tracked hand movement.
  - **main(String[] args)**

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

(TODO)

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.

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"
```