

Kinect Linux/Windows Programming Documentation

Introduction:

Browsing the web, you can find many cool hacks which allow Kinect be used by a PC rather than an XBOX. Via these hacks, we gain access to Kinect's cameras (color and depth), LED, accelerometer and motor. Obtaining both the raw and the depth image, it is easier to build Machine Vision applications.

Requirements:

Computer:

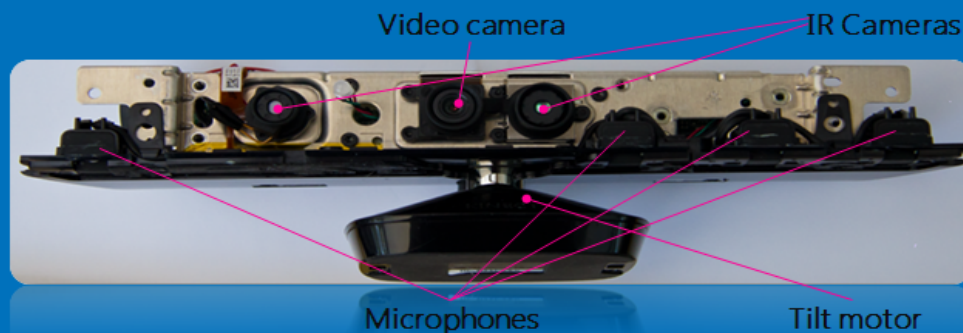
- Windows XP or later
- Ubuntu 12.04 or later (May work on other Linux Systems)
- Must not have CLNUI or Microsoft Kinect SDK

Programming:

- C++
- Microsoft Visual C++ Express or Professional 2010 (Windows)
- Any Linux C++ Compiler (Linux)

What is inside a Kinect:

Inside Kinect



- Video camera (640x480 and 1280x960 12, 15 and 30 frames per second(fps))
- 2 IR Cameras – depth data 80x60, 640x480 and 320x240 30 fps
- 4 Microphones
- Tilt motor

The Kinect depth sensor range is: minimum 800mm(2.62467 ft) and maximum 4000mm(13.12336 ft). The Kinect for Windows Hardware can however be switched to Near Mode which provides a range of 500mm(1.64042 ft) to 3000mm(9.84252 ft) instead of the default range. ***If you are using an Xbox Kinect then Near Mode is not supported.***

Setting Up Your Kinect and Computer (Linux):

(Note: Everything should be run under “sudo”)

Step 1

Check if we have all required libraries and are up to date:

```
apt-get install libusb-1.0-0-dev freeglut3-dev g++
```

Step 2

Create a folder for holding the download and installation:

```
cd ~
```

```
mkdir kinect
```

Step 3

Download the latest stable builds

- OpenNI Stable build for Ubuntu 12.04- - v1.5.4
- PrimeSense NITE Stable build for Ubuntu 12.04 - v1.5.2.21
- Avin2 Hardware Driver for Kinect

<http://www.openni.org/Downloads/OpenNIModules.aspx>

<https://www.github.com/avin2/SensorKinect/zipball/unstable>

Step 4

Your 'kinect' folder should have 3 zipped files. Extract the OpenNI zip file and rename the folder to 'openni'. Repeat the procedure for NITE

renaming the new folder to 'nite' and for avin2 renaming it to 'sensorkin'.

Step 5 Install OpenNI

```
cd ~/kinect/openni/
```

```
chmod a+x install.sh
```

```
./install.sh
```

Step 6 Install AVIN2

```
cd ~/kinect/sensorkin/Platform/Linux/CreateRedist/
```

```
chmod a+x RedistMaker
```

```
./RedistMaker
```

```
cd ../Redist/Sensor-Bin-Linux-x64-v5.1.0.25 (May not be exact)
```

Step 7 Install NITE

```
cd ~/kinect/nite/
```

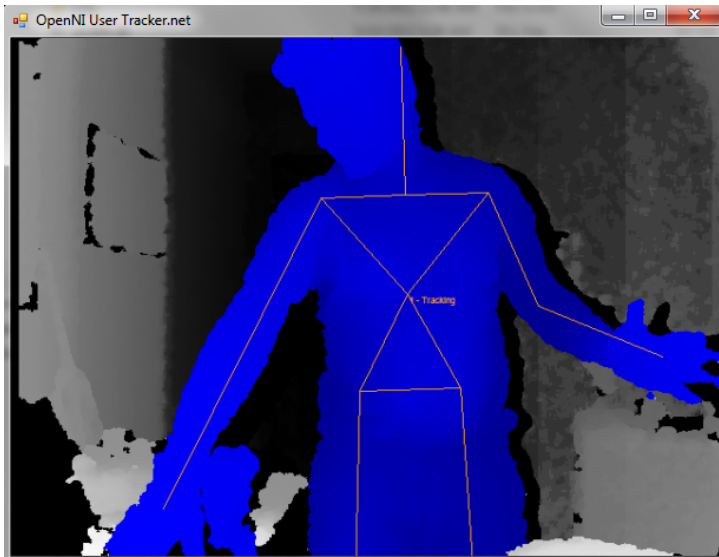
```
chmod a+x install.sh
```

```
./install.sh
```

Step 8 Testing

```
cd ~/kinect/openni/Samples/Bin/x64-Release/
```

```
./Sample-NiSimpleSkeleton
```



You should see the skeletal tracking of your body on your screen. If so, congratulations you have successfully setup your Kinect and Linux System.

Creating a Template C++ Project (Linux):

Step 1

Add the "Include" folder to your list of included directories in your compiler

Add the "Lib" folder to your list of included libraries in your compiler

Step 2

Insert the following code:

```
#include <XnOS.h>
```

```
#include <math.h>
```

```
#include <XnOpenNI.h>
```

```
#define SAMPLE_XML_PATH "../../Data/SamplesConfig.xml"
```

```

#define CHECK_RC(rc, what)
    \
    if (rc != XN_STATUS_OK)
        \
    {
        \
        printf("%s failed: %s\n", what, xnGetStatusString(rc)); \
        return rc;
    }

int main()
{
    XnStatus nRetVal = XN_STATUS_OK;
    XnContext* pContext;
    XnNodeHandle hScriptNode;
    XnEnumerationErrors* pErrors;
    XnNodeHandle hDepth;
    XnDepthMetaData* pDepthMD;
    const XnDepthPixel* pDepthMap;
    XnDepthPixel middlePoint;

    nRetVal = xnEnumerationErrorsAllocate(&pErrors);
    CHECK_RC(nRetVal, "Allocate errors object");

    nRetVal = xnInitFromXmlFileEx(SAMPLE_XML_PATH, &pContext,
    pErrors, &hScriptNode);
    if (nRetVal == XN_STATUS_NO_NODE_PRESENT)
    {

```

```

        XnChar strError[1024];
        xnEnumerationErrorsToString(pErrors, strError, 1024);
        printf("%s\n", strError);
        xnEnumerationErrorsFree(pErrors);
        return (nRetVal);
    }
    else if (nRetVal != XN_STATUS_OK)
    {
        printf("Open failed: %s\n", xnGetStatusString(nRetVal));
        xnEnumerationErrorsFree(pErrors);
        return (nRetVal);
    }

    xnEnumerationErrorsFree(pErrors);

    nRetVal = xnFindExistingRefNodeByType(pContext,
    XN_NODE_TYPE_DEPTH, &hDepth);
    CHECK_RC(nRetVal, "Find depth generator");

    pDepthMD = xnAllocateDepthMetaData();

    while (!xnOSWasKeyboardHit())
    {
        nRetVal = xnWaitOneUpdateAll(pContext, hDepth);
        if (nRetVal != XN_STATUS_OK)
        {
            printf("UpdateData failed: %s\n",

```

```

xnGetString(nRetVal));

        continue;
    }

    xnGetDepthMetaData(hDepth, pDepthMD);

    pDepthMap = pDepthMD->pData;

    middlePoint = pDepthMap[pDepthMD->pMap->Res.X * pDepthMD->pMap->Res.Y/2 + pDepthMD->pMap->Res.X/2];

    printf("Frame %d Middle point is: %u\n", pDepthMD->pMap->pOutput->nFrameID, middlePoint);
}

xnFreeDepthMetaData(pDepthMD);

xnProductionNodeRelease(hDepth);
xnProductionNodeRelease(hScriptNode);
xnContextRelease(pContext);

return 0;
}

```

Step 3

You should be able to just build and run the code, and see a bunch of lines printing the depth of the middle pixel. Try moving your hand in front of the Kinect, the numbers should change. Note that if you are too far away/too close, the depth reading will be zero.

Setting Up Your Kinect and Computer (Windows):

Step 1

Uninstall any previous Kinect drivers, such as CLNUI or Kinect SDK.

Step 2

- Download the Kinect Drivers for your 32 or 64 system here:
<https://github.com/avin2/SensorKinect/tree/unstable/Bin>
- Open the unzipped folder and navigate to Bin folder.
- Run the msi Windows file.

The drivers should now installed in your PC.

Step 3

Download and install the latest stable OpenNI Modules:
<http://www.openni.org/Downloads/OpenNIModules.aspx>

Step 4

Download and install the latest stable OpenNI Compliant Middleware Binaries:

<http://www.openni.org/Downloads/OpenNIModules.aspx>

During installation, provide the following (free) PrimeSense key:
0KOIk2JeIBYCIpWVnMoRKn5cdY4=

Step 5

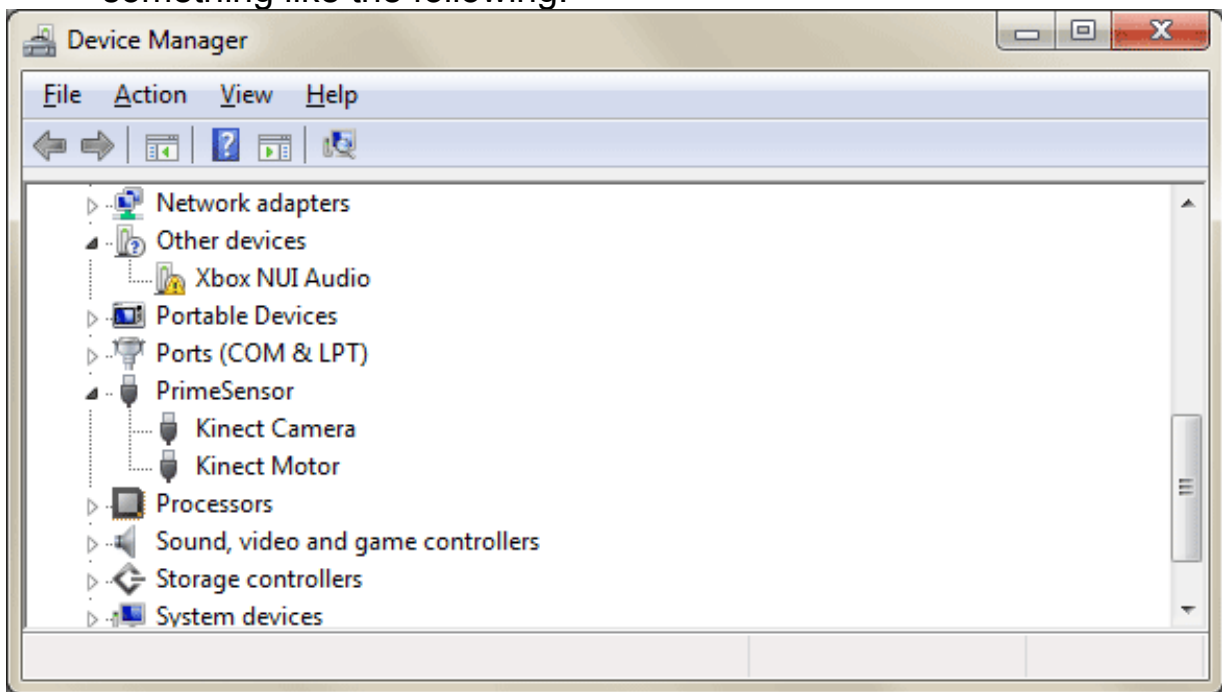
Download and install the latest stable or unstable OpenNI Compliant Hardware Binaries:

<http://www.openni.org/Downloads/OpenNIModules.aspx>

Step 6

- Plug in your Kinect device and connect its USB port with your PC.

- Wait until the driver software is found and applied.
- Navigate to the Device Manager (Control Panel). You should see something like the following:



Step 7

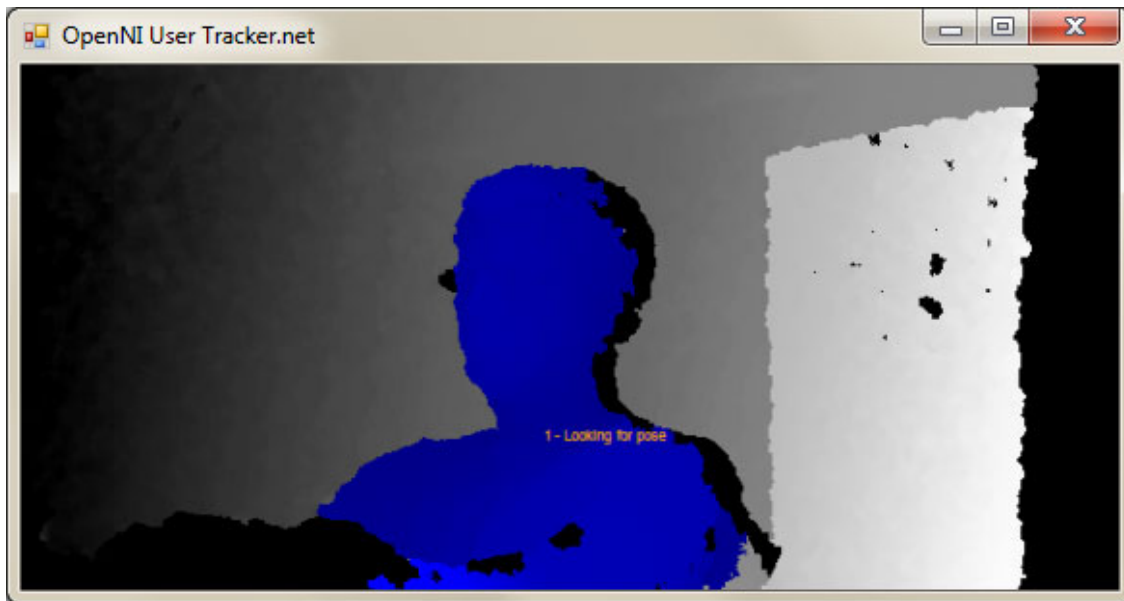
Navigate to C:\Program Files\OpenNI\Samples\Bin\Release (or C:\Program Files (x86)\OpenNI\Samples\Bin\Release) and try out the existing demo applications. Try the demos found in C:\Program Files\Prime Sense\NITE\Samples\Bin\Release (or C:\Program Files (x86)\Prime Sense\NITE\Samples\Bin\Release), too. If they work properly, then you are done.

Step 8

You have successfully installed Kinect in your Windows PC. Testing assemblies can be found in the following locations:

- C:\Program Files\OpenNI\Bin
and
 - C:\Program Files\Prime Sense\NITE\Bin
- OpenNI is the primary assembly you'll need when developing Natural

User Interfaces applications.



Setting Up Your C++ Compiler (Windows):

Step 1

Creating a console project in Microsoft Visual C++ Express 2010 and a main.cpp file

Step 2

- Open your project properties (right click your project in Solution Explorer, click properties).
- Select “All Configurations” in the Configurations dropdown.
- In the sidebar, select Configuration Properties->C/C++->General. In Additional Include Directories, paste the following:
`$(XN_NITE_INSTALL_PATH)\Include;$(OPEN_NI_INCLUDE);`
- Select Configuration Properties->Linker->General. In Additional Library Directories, paste the following:
`$(XN_NITE_INSTALL_PATH)\Lib;$(OPEN_NI_LIB);`

Select Configuration Properties->Linker->Input. Add the following to the Additional Dependencies property:

OpenNI.lib;XnVNite.lib;

Step 3

Go to your main .cpp file (the one that contains your main function), and paste the following code:

```
// Include OpenNI
#include <XnCppWrapper.h>
// Include NITE
#include "XnVNite.h"

// This macro checks the return code that all OpenNI calls make
// and throws an error if the return code is an error. Use this
// after all calls to OpenNI objects. Great for debugging.
#define CHECK_RC(rc, what)
    if (rc != XN_STATUS_OK)
    {
        printf("%s failed: %s\n", what, xnGetStatusString(rc));
        return rc;
    }

int _tmain(int argc, _TCHAR* argv[])
{
    // Variables
    // Keep track of the return code of all OpenNI calls
    XnStatus nRetVal = XN_STATUS_OK;
    // context is the object that holds most of the things related to OpenNI
    xn::Context context;
    // The DepthGenerator generates a depth map that we can then use to do
    // cool stuff with. Other interesting generators are gesture generators
    // and hand generators.
    xn::DepthGenerator depth;

    //
    // Initialization
    // Initialize context object
```

```

nRetVal = context.Init();
CHECK_RC(nRetVal, "Initialize context");
// Create the depth object
nRetVal = depth.Create(context);
CHECK_RC(nRetVal, "Create Depth");

// Tell the context object to start generating data
nRetVal = context.StartGeneratingAll();
CHECK_RC(nRetVal, "Start Generating All Data");

// We wish to print the middle pixel's depth, get the index
// of the middle pixel.
XnUInt32 nMiddleIndex = XN_QQVGA_X_RES * XN_QVGA_Y_RES / 2 +
    XN_QVGA_X_RES / 2;

// Main loop
while (true)
{
    // Wait for new data to be available
    nRetVal = context.WaitOneUpdateAll(depth);
    CHECK_RC(nRetVal, "Updating depth");
    // Get the new depth map
    const XnDepthPixel* pDepthMap = depth.GetDepthMap();
    // Print out the value of the middle pixel
    printf("Middle pixel is %u millimeters away\n",
pDepthMap[nMiddleIndex]);
}

// Clean-up
context.Shutdown();
return 0;
}

```

Step 3

You should be able to just build and run the code, and see a bunch of lines printing the depth of the middle pixel. Try moving your hand in front of the Kinect, the numbers should change. Note that if you are too far away/too close, the depth reading will be zero.

More Documentation:

Programmer's Guide – Learn about the OpenNI framework and see

how its rich programming features achieve natural interaction goals. This is a comprehensive introduction to the OpenNI system architecture and programming object model. Included are code snippets to illustrate the purpose and usage of the main OpenNI software objects.

<http://openni.org/Documentation/ProgrammerGuide.html>

Tutorials – Work through a series of useful OpenNI sample applications showing how to get started programming with the OpenNI framework. The tutorials go from the most basic to the more complex and all OpenNI programming features are thoroughly explained as they are introduced.

<http://openni.org/Documentation/Tutorial/index.html>

Reference Guide – Search the entire OpenNI reference documentation for detailed explanations of each class and its members. Each major class comes with a full introduction to its purpose, usage, and its place within the OpenNI framework.

<http://openni.org/Documentation/Reference/index.html>

Technical Support:

Please contact HackerSpace for technical support or more information.

Since 90% of people won't reach the very end to read this. I thought I would just write a few words about not creating an android with the Kinect and also we are not responsible for what you may do with this system.