Microsoft Kinect – Sample RGB Project

Overview

As previously mentioned, the OpenNI API is written in C++ but once you follow the installation procedures covered, here, you will have some pre-compiled wrappers that will give you access to use OpenNI in a few other languages if you need.

Since there is a plethora of existing tutorials regarding setting up various development environments in C++ and corresponding example projects, this article will show you how to setup the 32-bit OpenNI JAR (OpenNI Java wrapper) in Eclipse IDE. We will then initialize an OpenNI production node to begin accessing Kinect data and to get the RGB stream into OpenCv, which is a popular computer vision library.

Before going on to the following project, make sure that you have all of the dependent libraries installed on your machine. For the instructions on getting the 3rd party libraries and for setting up the development environment check out this post.

Also, I want to clarify that this code is merely one solution that I managed to successfully execute. This said, it may have bugs and/or mayb be done more successfully or more easily using a different solution. If you have any suggestions or find errors, please don't hesitate to contact us and I will change the post immediately. These posts follow and continue to follow exploration and collaboration.

Using the Kinect's RGB feed

In this project we will:

1. *Make a simple program to capture the RGB feed from the Kinect in Java*
2. *Get the data into an OpenCV image data structure*
3. *Display the data on the screen*

A high-level overview of the steps we need to take are as follows:

1. *Create a new 'context' for the Kinect to be started*
2. *Create and start a 'generator' which acts as the mechanism for delivering both data and metadata about its corresponding feed*
3. *Translate the raw Kinect data into a Java data structure to use in native Java libraries*
4. *Capture a "frame" and display it on screen*

The next tab is the commented code for you to use as you wish.

NOTE: For extremely in-depth and excellent instruction on using JavaCV, the Kinect, along with various other related projects I extremely the book(s) by Andrew Davison from the Imperial College London. A list of his works can be found here http://www.doc.ic.ac.uk/~ajd/publications.html and here http://fivedots.coe.psu.ac.th/~ad/.

Sample RGB Project – Part 1

First, let's import the required libraries:

```
import org.OpenNI.*;

import com.googlecode.javacv.*;
import static com.googlecode.javacv.cpp.opencv_imgproc.*;
import com.googlecode.javacv.cpp.opencv_core.IplImage;
```

Then eclipse nicely fills out our class information.

```
public class sampleRGB {
```

We define some global variables

```
static int imWidth, imHeight;

static ImageGenerator imageGen;
static Context contex
```

Eclipse will also fill out our "main" statement for use by checking the box on the project set up. One addition we will need to make is to surround the following code block with a statement for any exceptions that may be thrown when starting the data feed from the Kinect. Here we are starting a new "context" for the Kinect

```
public static void main(String[] args) throws GeneralException {
Create a "context"


context = new Context();
```

Sample RGB Project – Part 2

We are manually adding the license information from Primesense. You can also directly reference the xml documents located in the install directory of both the OpenNI and Primesense.

```
License license = new License("PrimeSense", "0KOIk2JeIBYClPWVnMoRKn5cdY4=");
context.addLicense(license);
```

Create a "generator" which is the machine that will pump out RGB data

```
imageGen = ImageGenerator.create(context);
```

We need to define the resolution of the data coming from the image generator. OpenNI calls this mapmode (imageMaps, depthMaps, etc.). We will use the standard resolution.

First initialize it to null.

```
MapOutputMode mapMode = null;
mapMode = new MapOutputMode(640, 480, 30);
imageGen.setMapOutputMode(mapMode);
```

We also need to pick the pixel format to display from the Image Generator. We will use the Red-Green-Blue 8-bit 3 channel

or "RGB24"

```
imageGen.setPixelFormat(PixelFormat.RGB24);
```

Sample RGB Project – Part 3

OpenNI also allows us to easily mirror the image so movement in 3d space is reflected in the image plane

```
context.setGlobalMirror(true);
```

Create an Iplimage(opencv image) with the same size and format as the feed from the kinect.

```
IplImage rgbImage = IplImage.create(imWidth,imHeight, 8, 3);
```

Next we will use the easy route and utilize JFrame/Javacv optimized canvas to show the image

```
CanvasFrame canvas = new CanvasFrame("RGB Demo");
```

Now we will create a never-ending loop to update the data and frames being displayed on the screen. Going line by line we will, update the context every time the image generator gets new data.
Set the opencv image data to the byte buffer created from the imageGen.

> ***NOTE: For some reason the channels coming from the Kinect to the opencv image are ordered differently so we will simply use the opencv convert color to set the "BGR" to "RGB". We tell the canvas frame that we created to show image.***

Sample RGB Project – Part 4

Finally, we need to also release the Kinect context or we will get an error the next time we try to start a node because the needed files will be locked

```
while (true){
context.waitOneUpdateAll(imageGen);
rgbImage.imageData(imageGen.createDataByteBuffer());
cvCvtColor(rgbImage, rgbImage, CV_BGR2RGB);
canvas.showImage(rgbImage);

canvas.setDefaultCloseOperation(CanvasFrame.EXIT_ON_CLOSE);
}
```

****Note that you can add an argument to the canvas frame to reverse the channels of the image**

You are reading the series: Working with the Microsoft Kinect
Microsoft Kinect – An Overview of Working With Data
Microsoft Kinect – Sample RGB Project
Microsoft Kinect – Setting Up the Development Environment
Microsoft Kinect – Additional Resources