Kinect Server Data Acquisition: Compilation and Running Requirements and Directions

Vahid Soleimani, Majid Mirmehdi, Dima Damen, Sion Hannuna, Massimo Camplani

This document introduces hardware and software resources required for compiling and running the source code of “Kinect data Acquisition” system [1]. It also explains how to establish the necessary libraries dependencies to be able to successfully compile and run the source code. Further, the software specifications and details, and the format of its output data are described in this document.

1- Requirements and Dependencies

The following hardware, software and libraries are required to successfully compile and run the source code.

- A second generation of Kinect for Windows
- A Kinect–compatible Microsoft Windows (8.1 or 10) PC or laptop
- Microsoft Visual Studio C++
- Kinect for Windows SDK 2.0
- Intel® Threading Building Blocks (TBB)
- OpenCV

According to Microsoft, not all laptops and PCs are compatible with Kinect V2. A detailed information about hardware and software requirements can be found in this page and also here. The Kinect data acquisition source code has been developed and successfully compiled under Microsoft Visual Studio C++ 2012, but you can also use the later versions if the following dependencies have been provided for them. Kinect for Windows SDK 2.0, Intel® Threading Building Blocks and OpenCV are the libraries which are required for compiling and executing the source code. Establishment of these libraries dependencies in Visual Studio 2012 is explained in Section 2.

2- Visual Studio Dependencies Establishment

After opening the solution file, provided in the repository, active solution platform and configuration must be changed to x64 and Released, respectively, as shown in Fig. 1.

Kinect for Windows SDK 2.0 – This standard library is provided by Microsoft and is essential for making the Kinect work under Microsoft Windows. It is released under free commercial license and can be downloaded from here. After installation, SDK provides a default environment variable named KINECTS SDK20_DIR, which is used to add the SDK include and library dependencies to Visual Studio. First, the project property page must be opened (Fig. 2a), and then project “Additional Include Directories” page can be accessed through steps 1 to 3 in Fig.2b.
Fig. 1: (a) Change the platform to x64. (b) Changing the active solution configuration to Release.

Fig. 2: (a) Accessing the “project property page”. (b) Accessing the “Additional Include Directories”.
Any required include directory can be added in “Additional Include Directories” page (Fig. 3), by either using the absolute path or using the environment variable. For the Kinect SDK, the default environment variable is used as shown in Fig. 3.

Fig. 3: Adding required includes to the project “Additional Include Directories” page.

To add Kinect SDK library, first, the library path should be declared at “Additional Library Directories” in the project property page. To do this, after accessing to project property page (Fig. 2a), “Additional Library Directories” is accessible through steps 1 to 3 in Fig. 4a.
Then, Kinect SDK library path is added as $\text{KINECTS\_SDK20\_DIR}\text{Lib}\text{x64}$ (Fig. 4b).

Finally, libraries’ name that are used must be added to the “Additional Dependencies”. “Additional Dependencies” page can be accessed through steps 1 to 3 in Fig. 5a. For the Kinect SDK, the only library which must be added is $\text{kinect20.lib}$. 

(a)

(b)
Fig. 5: (a) Accessing “Additional Dependencies”, (b) Adding required library files.

**Intel® Threading Building Blocks (TBB) —** This library is used to create parallel threads for acquiring, visualising and recording RGB, depth and skeleton data streams separately at full frame rate. This library is licensed as GPLv2. Although any version can be used for our purpose, the version we used and tested is “tbb43_20140724oss” and can be downloaded from [here](#). After extracting, an environment variable should be defined for the main path (e.g. `C:\TBB2014\tbb43_20140724oss`). We named this variable as `TBB_DIR` and used to establish include and library dependencies similar to Kinect SDK. After accessing to project property page (Fig. 2a), `$(TBB_DIR)\include` is added to include directories (Fig. 3). `$(TBB_DIR)\lib\intel64\vc11` is then added to the “Additional Library Directory” page using (Fig. 4b). If Visual Studio 2013 is used then `vc11` sub-folder of `$(TBB_DIR)\lib\intel64\vc11` needs to be changed to `vc12`. The only TBB library that is needed here is `tbb.lib` (Fig. 5b).

**OpenCV —** This is an open source computer vision library, which consists of considerable number of computer vision programming functions and is released under a BSD license. Similar to TBB, any version can be used for our purpose but we tested our code with pre-built libraries version 2.4.11. An official and comprehensive explanation about installing OpenCV on Windows can be found [here](#). The determined environment variable for OpenCV (here named `OPENCV_DIR`) is used to establish the dependencies similar to TBB (Figs. 2–4). However, there are more library files that must be added to library “Additional Dependencies” as shown in Fig. 5b.
Dynamic Link Libraries (DLL) – In order to successfully run the Kinect data acquisition software, path of binary files of third party libraries i.e. Kinect SDK, TBB and OpenCV, must be added to the system path. For the Kinect SDK, it is added automatically by installing the SDK. However, for the TBB and OpenCV, the following paths must be added to the system path as shown in Fig. 6.

- `%OPENCV_DIR%\bin`
- `%TBB_DIR%\bin\intel64\vc11`

This can be done by accessing Environment Variables from the System Properties.

![Environment Variables](image)

Fig. 6: Adding TBB and OpenCV libraries binary files to the system path.

3- Software and Output Data Specifications

After compiling and building the source code successfully, the application is ready to be used. Fig. 7 shows a sample running of the software.

As can be seen in Fig. 7a, the software is featured the following specifications:

1- Recording path.
2- RGB, depth and skeleton recording options.
3- RGB, depth and skeleton online visualisation options.
4- JPEG or BMP encoding option for saving RGB data frames.
5- RGB scaling factor (0-1) to reduce the RGB image size.
6- Start/Stop button.
Fig. 7: A sample running of the Kinect data acquisition software.

Fig. 7b presents a sample of online visualisation of RGB, depth and skeleton data. In addition to saving RGB data frames in JPEG or BMP formats, and depth data frames in PNG format, the software also provides separate RGB, depth and skeleton Meta data for the whole captured sequence.

Before data capture starts, server and client machines must be locally networked and their system time must be synchronised using NTP. First, a TCP/IP connection is established at the beginning of capture by pushing the `connect!` button on both server and client applications. Note that the server button must be pressed first. Then, both Kinects must be turned on by pushing `Kinect ON!` buttons. Data acquisition in both server and client machines is starts by pressing the `start/stop Recording!` button of the Server Data Acquisition application.
**RGB and depth Meta data** – For each sequence, RGB and depth Meta data are stored in two separate text files, named “rgbMeta.txt” and “depthMeta.txt”. These files present the following information for each RGB and depth captured frames:

- system timestamp
- Kinect provided timestamp
- TBB thread timestamp

![Fig. 8: RGB and depth metadata files content.](image)

Fig. 8 shows part of an RGB and depth Meta data files. Each line of these text files correspond to a single captured frame. As can be seen in Fig. 8, the first column is the system timestamp and the second column is the frame number, which is equal to the corresponding the RGB and depth file names. Note that RGB and depth frames must be matched based on the Kinect provided timestamps, not the frame numbers. The third column is Kinect provided timestamp and the last column is the TBB timestamp obtained from the RGB and depth frame acquisition thread.

**Skeleton Meta data** – If a person appears in the Kinect field of view, their skeleton data is obtained and stored in a text file, named “bodyMeta.txt”. Similar to RGB and depth Meta data, each line is related to the corresponding depth captured frames. In addition to system, Kinect and TBB timestamps that are similar to the RGB and depth Meta data with the same storing order, skeleton body data which includes 25 joints position and their detection confidence, are also stored in the skeleton Meta data file. A quadruple is associated with each joint, in which the first number states the joint tracking confidence. 0 means no joint is detected, 1 means joint is weekly detected and 2 means joint is fully detected. The other three numbers present each joint location, i.e.
(x, y, z), in which x and y are presented in pixel (to match depth image), and z is real world coordinate system (in meter). x and y can be easily converted to real world coordinate using the Kinect depth sensor camera parameters. Fig. 9 shows a part of skeleton Meta data file.

![Skeleton Meta data file content](image)

Order of the joints for each frame in the skeleton Meta data file (Fig. 9) is presented is the Table 1. This is a standard order introduced by Microsoft.

Table1: Skeleton metadata joints order and name.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Spine−Base</td>
<td>Spine−Mid</td>
<td>Neck</td>
<td>Head</td>
<td>Shoulder−Left</td>
<td>Elbow−Left</td>
<td>Wrist−Left</td>
<td>Hand−Left</td>
<td>Shoulder−Right</td>
<td>Elbow−Right</td>
<td>Wrist−Right</td>
<td>Hand−Right</td>
<td>Hip−Left</td>
<td>Ankle−Left</td>
<td>Foot−Left</td>
<td>Hip−Right</td>
<td>Knee−Right</td>
<td>Ankle−Right</td>
<td>Foot−Right</td>
<td>Spine−Shoulder</td>
<td>HandTip−Left</td>
<td>Thumb−Left</td>
<td>HandTip−Right</td>
<td>Thumb−Right</td>
<td></td>
</tr>
</tbody>
</table>

Note that the correct way to synchronise RGB, depth and skeleton frames, is using Kinect or system timestamps rather than the frame numbers.

References