Kinect v2 tutorial for its usage in ros as well as the calibration tutorial

7 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

taochenshh committed on GitHub Update README.md Latest commit cc3efae on Oct 8, 2016

035623243247	Initial Commit	9 months ago
kinect2_calibration	Initial Commit	9 months ago
README.md	Update README.md	9 months ago
get_extrinsics.py	Initial Commit	9 months ago
image_to_world.py	Initial Commit	9 months ago
pose_estimation.py	Initial Commit	9 months ago

README.md

Kinect-v2-Tutorial

Kinect v2 with ROS

Install the Driver for Kinect v2

- Download libfreenect2 source

```
git clone https://github.com/OpenKinect/libfreenect2.git
cd libfreenect2
```

- (Ubuntu 14.04 only) Download upgrade deb files

```
cd depends; ./download_debs_trusty.sh
```

- Install build tools

```
sudo apt-get install build-essential cmake pkg-config
```

- Install libusb. The version must be >= 1.0.20 .

1. (Ubuntu 14.04 only) `sudo dpkg -i debs/libusb*deb`
2. (Other) `sudo apt-get install libusb-1.0-0-dev`

- Install TurboJPEG

1. (Ubuntu 14.04 and newer) `sudo apt-get install libturbojpeg libjpeg-turbo8-dev`
2. (Debian) `sudo apt-get install libturbojpeg0-dev`

- Install OpenGL

1. (Ubuntu 14.04 only) `sudo dpkg -i debs/libglfw3*deb; sudo apt-get install -f; sudo apt-get install libgl1-mesa-dri-lts-vivid` (If the last command conflicts with other packages, don't do it.)

2. (Odroid XU4) OpenGL 3.1 is not supported on this platform. Use `cmake -DENABLE_OPENGL=OFF` later.
3. (Other) `sudo apt-get install libglfw3-dev`

- Install OpenCL (optional) If you are using **Intel GPU**: (Ubuntu 14.04 only)

```
sudo apt-add-repository ppa:floe/beignet; sudo apt-get update; sudo apt-get install beignet-dev; sudo dpkg
```

If you are using **AMD GPU**: Install the latest version of the AMD Catalyst drivers from https://support.amd.com and `apt-get install opencl-headers`.

- Install CUDA (optional, Nvidia only): (Ubuntu 14.04 only) Download `cuda-repo-ubuntu1404...*.deb` ("deb (network)") from Nvidia website, follow their installation instructions, including `apt-get install cuda` which installs Nvidia graphics driver.

- Install VAAPI (optional, Intel only) (Ubuntu 14.04 only) `sudo dpkg -i debs/{libva,i965}*deb; sudo apt-get install -f`

- Install OpenNI2 (optional) (Ubuntu 14.04 only) `sudo apt-add-repository ppa:deb-rob/ros-trusty && sudo apt-get update` (You don't need this if you have ROS repos), then `sudo apt-get install libopenni2-dev`

- Build

```
cd ..
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/freenect2
make
make install
```

- Set up udev rules for device access: `sudo cp ../platform/linux/udev/90-kinect2.rules /etc/udev/rules.d/`, then **replug the Kinect**.
- Run the test program: ./bin/Protonect

If you encounter the following error, it's very likely that you haven't installed the Nvidia driver.

```
Version: 0.2.0
Environment variables: LOGFILE=<protonect.log>
Usage: ./bin/Protonect [-gpu=<id>] [gl | cl | cuda | cpu] [<device serial>]
        [-noviewer] [-norgb | -nodepth] [-help] [-version]
        [-frames <number of frames to process>]
To pause and unpause: pkill -USR1 Protonect
[Info] [Freenect2Impl] enumerating devices...
[Info] [Freenect2Impl] 8 usb devices connected
[Info] [Freenect2Impl] found valid Kinect v2 @2:5 with serial 035623243247
[Info] [Freenect2Impl] found 1 devices
libGL error: failed to load driver: swrast
[Error] [OpenGLDepthPacketProcessorImpl] GLFW error 65543 GLX: Failed to create context: BadMatch (invalid
[Error] [OpenGLDepthPacketProcessor] Failed to create opengl window.
```

To solve this problem, just install the nvidia driver for your ubuntu system. For example, if you are using GeForce GTX 960, then run:

```
sudo apt-get install nvidia-352
```

- Run OpenNI2 test (optional): `sudo apt-get install openni2-utils && sudo make install-openni2 && NiViewer2`. Environment variable LIBFREENECT2_PIPELINE can be set to cl, cuda, etc to specify the pipeline.

- Clone this repository into your catkin workspace, install the dependencies and build it

```
cd ~/catkin_ws/src/
git clone https://github.com/code-iai/iai_kinect2.git
cd iai_kinect2
rosdep install -r --from-paths .
cd ~/catkin_ws
catkin_make -DCMAKE_BUILD_TYPE="Release"
```

Note: rosdep will output errors on not being able to locate **[kinect2_bridge]** and **[depth_registration]**. That is fine because they are all part of the iai_kinect2 package and rosdep does not know these packages.

Note: If you installed libfreenect2 somewhere else than in `$HOME/freenect2` or a standard location like `/usr/local` you have to specify the path to it by adding `-Dfreenect2_DIR=path_to_freenect2/lib/cmake/freenect2` to catkin_make.

Running Example

- Connect your sensor and run kinect2_bridge:

```
roslaunch kinect2_bridge kinect2_bridge.launch
```

- View Cloud

```
rosrun kinect2_viewer kinect2_viewer sd cloud
```

Calibration Tutorial

The calibration process for each camera is separated into two steps, record and calibrate. That's to say, we first record images on which chess corners are detected into a folder, then we run calibrate program.The main package we are gonna use here is kinect2_calibration.

Key bindings

Windows:

- ESC, q: Quit
- SPACE, s: Save the current image for calibration
- l: decrease min and max value for IR value rage
- h: increase min and max value for IR value rage
- 1: decrease min value for IR value rage
- 2: increase min value for IR value rage
- 3: decrease max value for IR value rage
- 4: increase max value for IR value rage

Terminal:

- `CRTL` + `c` : Quit

Calibration

In the record stage, (an) image window(s) will be opened showing the subscribed images. What you need to do is to move the chessboard around in front of the camera, and then press `SPACE` key to save the corner points when chess corners are detected and colored lines show up in the image window. You need to save enough number of groups of points so as to get good calibration results. The images and corner points will be saved in the folder ~/kinect_cal_data.

In the calibrate stage, just run the program in a `calibrate` mode. Again, the result will be saved in the folder ~/kinect_cal_data.

Detailed steps:

- Start kinect2_bridge

```
roslaunch kinect2_bridge kinect2_bridge.launch
```

You can limit the frames per second (to make it easy on your CPU) by adding `_fps_limit:=2` at the end of this command.

- Create a directory for the calibration data files

```
mkdir ~/kinect_cal_data; cd ~/kinect_cal_data
```

- Calibrate color camera

Record images for the color camera:

```
rosrun kinect2_calibration kinect2_calibration chess8x6x0.0243 record color
```

Note: The first number of the chessboard parameters (8 here) represents the number of columns in the chessboard, while the second number(6 here) is the number of rows in the chessboard, the third number (0.0243) is the width of the square.

Calibrate the intrinsics:

```
rosrun kinect2_calibration kinect2_calibration chess8x6x0.0243 calibrate color
```

- Calibrate ir camera

Record images for the color camera:

```
rosrun kinect2_calibration kinect2_calibration chess8x6x0.0243 record ir
```

Calibrate the intrinsics:

```
rosrun kinect2_calibration kinect2_calibration chess8x6x0.0243 calibrate ir
```

- Calibrate Extrinsics:

Record images on both cameras synchronized:

```
rosrun kinect2_calibration kinect2_calibration chess8x6x0.0243 record sync
```

Calibrate extrinsics:

```
rosrun kinect2_calibration kinect2_calibration chess8x6x0.0243 calibrate sync
```

- Calibrate the depth measurements

```
rosrun kinect2_calibration kinect2_calibration chess8x6x0.0243 calibrate depth
```

- Store the calibration results into kinect2_bridge directory Find out the serial number of your kinect2 by looking at the first lines printed out by the kinect2_bridge when you launched it. The line looks like this: `device serial: 035623243247` Create the calibration results directory in kinect2_bridge/data/$serial: `roscd kinect2_bridge/data; mkdir 035623243247` Copy the following files from your calibration directory (~/kinect_cal_data) into the directory you just created: `calib_color.yaml calib_depth.yaml calib_ir.yaml calib_pose.yaml` You need to remove all `!!opencv-matrix` in these files in order to work with these yaml files in python. As for the first line `%YAML:1.0`, you can keep it and skip the first line in your code when you open the file or you can remove it, too.

- Restart the kinect2_bridge

Next, refer to kinect v1 calibration tutorialto get the kinect's position in world coordinate system.