README

SLAMBench Release Candidate 1.1

Developed in the PAMELA project, EPSRC Programme Grant EP/K008730/1

What is it for?

- A SLAM performance benchmark that combines a framework for quantifying quality-of-result with instrumentation of execution time and energy consumption.
- It contains a KinectFusion (http://research.microsoft.com/pubs/155378/ismar2011.pdf) implementation in C++, OpenMP, OpenCL and CUDA (inspired by https://github.com/GerhardR).
- It offers a platform for a broad spectrum of future research in jointly exploring the design space of algorithmic and implementation-level optimisations.
- Target desktop, laptop, mobile and embedded platforms. Tested on Ubuntu, OS X and Android (on Android only the benchmark application has been ported, see later).

If you use SLAMBench in scientific publications, we would appreciate citations to the following paper (http://arxiv.org/abs/1410.2167):

L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. J. Kelly, A. J. Davison, M. Luján, M. F. P. O’Boyle, G. Riley, N. Topham, and S. Furber. Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM. In IEEE Intl. Conf. on Robotics and Automation (ICRA), May 2015. arXiv:1410.2167.

Bibtex entry:

```
#!latex

@inproceedings{Nardi2015,
    author={Nardi, Luigi and Bodin, Bruno and Zia, M. Zeeshan and Mawer, John and
Nisbet, Andy and Kelly, Paul H. J. and Davison, Andrew J. and Luj\'an, Mikel and
O'Boyle, Michael F. P. and Riley, Graham and Topham, Nigel and Furber, Steve},
    title = "{Introducing SLAMBench, a performance and accuracy benchmarking
methodology for SLAM}",
    booktitle = "{IEEE Intl. Conf. on Robotics and Automation (ICRA)}",
    year = {2015},
    month = {May},
    NOTE = {arXiv:1410.2167}
    }
```

How do I get set up on Ubuntu?

If you want to set up for OS X go to the relevant section.

Dependencies

Required

- TooN: maths library.
- CMake 2.8+ : building tool.

Install TooN and CMake

```
#!shell
git clone git://github.com/edrosten/TooN.git
cd TooN
./configure
sudo make install
sudo apt-get install cmake
```

+(with Ubuntu, you might need to install the build-essential package using `sudo apt-get update && sudo apt-get install build-essential`)

Optional

- OpenMP : for the OpenMP version
- CUDA : for the CUDA version
- OpenCL : for the OpenCL version (OpenCL 1.1 or greater)
- OpenGL / GLUT : used by the graphical interface
- OpenNI : for the live mode, and for `oni2raw` tool (which convert an OpenNI file to the SLAMBench internal format)
- Freenect Drivers : In order to use the live mode.
- PkgConfig / Qt5 (using OpenGL) : used by the Qt graphical interface (not fully required to get a graphical interface)
- Python (numpy) : use by benchmarking scripts (`mean`, `max`, `min` functions)

Installation of Qt5 with an ARM board (ie. Arndale, ODROID,...)

On ARM board, the default release of Qt5 was compile using OpenEGL, to use the Qt interface, you will have to compile Qt :

```
#!
cd ~
wget
http://download.qt-project.org/official_releases/qt/5.2/5.2.1/single/qt-everywhere-
opensource-src-5.2.1.tar.gz
tar xzf qt-everywhere-opensource-src-5.2.1.tar.gz
cd ~/qt-everywhere-opensource-src-5.2.1
./configure -prefix ~/.local/qt/ -no-compile-examples  -confirm-license  -release
-nomake tests   -nomake examples
make
make install
```

Compilation of SLAMBench

Then simply build doing:

```
#!
make
```

To use qt, if you compile the source as explained above, you should need to specify the Qt install dir :

```
#!
CMAKE_PREFIX_PATH=~/.local/qt/ make
```

Running SLAMBench

The compilation builds 3 application modes which act like wrappers (the kernels are the same for all applications):

1. benchmark: terminal user interface mode for benchmarking purposes
2. main: GLUT GUI visualisation mode
3. qmain: Qt GUI visualisation mode

Each application mode is also declined in 4 different builds/implementations:

- C++ (*./build/kfusion/kfusion-main-cpp*)
- OpenMP (*./build/kfusion/kfusion-main-openmp*)
- OpenCL (*./build/kfusion/kfusion-main-cpp*)
- CUDA (*./build/kfusion/kfusion-main-cuda*)

The Makefile will automatically build the executable with satisfied dependencies, e.g. the OpenCL application version will be built only if the OpenCL tool kit is available on the system and so on for CUDA, OpenMP and Qt.

All application modes and implementations share the same set of arguments:

```
#!plain
-c  (--compute-size-ratio)       : default is 1   (same size)
-d  (--dump-volume) <filename>   : Output volume file
-f  (--fps)                      : default is 0
-i  (--input-file) <filename>    : Input camera file
-k  (--camera)                   : default is defined by input
-l  (--icp-threshold)        : default is 1e-05
-o  (--log-file) <filename>      : default is stdout
-m  (--mu)                       : default is 0.1
-p  (--init-pose)                : default is 0.5,0.5,0
-q (--no-gui)                    : disable any gui used by the executable
```

```
-r  (--integration-rate)          : default is 1
-s  (--volume-size)               : default is 2,2,2
-t  (--tracking-rate)             : default is 1
-v  (--volume-resolution)         : default is 256,256,256
-y  (--pyramid-levels)            : default is 10,5,4
-z  (--rendering-rate)    : default is 4
```

SLAMBench supports several input streams (how to use these inputs is described later):

- ICL-NUIM dataset (http://www.doc.ic.ac.uk/~ahanda/VaFRIC/iclnuim.html)
- RGB-D camera like Microsoft Kinect or other PrimeSense cameras using the OpenNI interface
- OpenNI pre-recorded file
- Raw format

1. benchmark mode

Use this mode for benchmarking proposes. The output is:

- frame : ID number of the current frame
- acquisition : input data acquisition elapsed time (file reading)
- preprocessing : pre-processing elapsed time (includes kernels mm2meters bilateralFilter)
- tracking : tracking elapsed time (includes kernels halfSample, depth2vertex vertex2normal, track, reduce and solve)
- integration : integration elapsed time (includes kernel integrate)
- raycast : raycast elapsed time (include kernel raycast)
- rendering : rendering elapsed time (includes kernels renderDepth renderTrack and renderVolume)
- computation : pre-processing + tracking + integration + raycast. This is the total elapsed time for processing a frame but not including the acquiring and the visualisation kernels
- total : computation + acquisition + rendering. This is the total elapsed time for processing one frame (including the acquiring and the visualisation kernels)
- X,Y,Z : estimation of the camera position (tracking result)
- tracked : this boolean indicates if for the current frame we have not lost the tracking of the camera (1 = tracking, 0 = tracking lost)
- integrated : this boolean indicates if the integration step occurs for the current frame (depending of the tracking result and of the integration rate)

How to use the benchmark mode with the ICL-NUIM dataset

SLAMBench provides an interface to the ICL-NUIM dataset. This enables the accuracy evaluation on a SLAM implementation via the ICL-NUIM ground truth. ICL-NUIM provides 4 trajectories, we pick trajectory 2 and show how to use the dataset (for the download of each trajectory we recommend 2 GB of space available on the system):

```
#!plain
mkdir living_room_traj2_loop
cd living_room_traj2_loop
wget http://www.doc.ic.ac.uk/~ahanda/living_room_traj2_loop.tgz
```

```
tar xzf living_room_traj2_loop.tgz
cd ..
```

You can use the ICL-NUIM dataset in its native format or in a RAW format (with the latter acquiring speed increases). RAW is the format to be used for benchmarking purposes, to generate the RAW file:

```
#!plain
./build/kfusion/thirdparty/scene2raw living_room_traj2_loop
living_room_traj2_loop.raw
```

Run SLAMBench:

```
#!plain
./build/kfusion/kfusion-benchmark-cuda -i living_room_traj2_loop.raw  -s 4.8 -p
0.34,0.5,0.24 -z 4 -c 2 -r 1 -k 481.2,480,320,240  > benchmark.log
```

You can replace *cuda* by *openmp*, *opencl* or *cpp*.

In order to check the accuracy of your tracking compared to the ground truth trajectory, first download the ground truth trajectory file:

```
#!plain
wget http://www.doc.ic.ac.uk/~ahanda/VaFRIC/livingRoom2.gt.freiburg
```

And then use the following tool:

```
#!plain
./kfusion/thirdparty/checkPos.py benchmark.log livingRoom2.gt.freiburg
Get slambench data.
slambench result        : 882 positions.
NUIM  result        : 880 positions.
Working position is : 880
Runtimes are in seconds and the absolute trajectory error (ATE) is in meters.
The ATE measure accuracy, check this number to see how precise your computation is.
Acceptable values are in the range of few centimeters.

            tracking    Min : 0.005833  Max : 0.046185  Mean : 0.020473     Total :
18.05721755
         integration    Min : 0.003629  Max : 0.041839  Mean : 0.021960     Total :
19.36882799
           rendering    Min : 0.018242  Max : 0.022144  Mean : 0.018486     Total :
16.30500085
       preprocessing    Min : 0.000633  Max : 0.002064  Mean : 0.000719     Total :
0.63432674
         computation    Min : 0.010156  Max : 0.075946  Mean : 0.043152     Total :
38.06037227
           total        Min : 0.028520  Max : 0.094302  Mean : 0.061724     Total :
```

```
54.44080794
                 ATE    Min : 0.000000  Max : 0.044235  Mean : 0.018392     Total :
16.18503741
         acquisition    Min : 0.000069  Max : 0.000404  Mean : 0.000086     Total :
0.07543481
```

2. main mode

This is a GUI mode which internally uses GLUT for the visualisation step. We do not suggest to use this mode for benchmarking purposes because the visualisation step can interfere with the computation elapsed time (see http://arxiv.org/abs/1410.2167 for more information).

An example of use of the main application is:

```
#!plain
./build/kfusion/kfusion-main-cpp -i ~/Downloads/living_room_traj2_loop/
```

3. mainQt mode

This is a GUI mode which internally uses Qt for the visualisation step. We do not suggest to use this mode for benchmarking purposes because the visualisation step can interfere with the computation elapsed time (see http://arxiv.org/abs/1410.2167 for more information).

An example of use of the mainQt application is:

```
#!plain
./build/kfusion/kfusion-qt-cpp -i ~/Downloads/living_room_traj2_loop/
```

Kernel timings

CPP

```
#!plain
KERNEL_TIMINGS=1 ./build/kfusion/kfusion-benchmark-cpp -s 4.8 -p 0.34,0.5,0.24 -z 4
-c 2 -r 1 -k 481.2,480,320,240 -i  living_room_traj2_loop.raw -o
benchmark.2.cpp.log 2> kernels.2.cpp.log
```

OpernMP

```
#!plain
KERNEL_TIMINGS=1 ./build/kfusion/kfusion-benchmark-openmp -s 4.8 -p 0.34,0.5,0.24
-z 4 -c 2 -r 1 -k 481.2,480,320,240 -i  living_room_traj2_loop.raw -o
benchmark.2.openmp.log 2> kernels.2.openmp.log
```

OpenCL

It is possible to profile OpenCL kernels using an OpenCL wrapper from the thirdparty folder :

```
#!plain
LD_PRELOAD=./build/kfusion/thirdparty/liboclwrapper.so
./build/kfusion/kfusion-benchmark-opencl -s 4.8 -p 0.34,0.5,0.24 -z 4 -c 2 -r 1 -k
481.2,480,320,240 -i  living_room_traj2_loop.raw -o benchmark.2.opencl.log 2>
kernels.2.opencl.log
```

CUDA

The CUDA profiling takes advantage of the NVIDIA nvprof profiling tool.

```
#!plain
nvprof --print-gpu-trace ./build/kfusion/kfusion-benchmark-cuda -s 4.8 -p
0.34,0.5,0.24 -z 4 -c 2 -r 1 -k 481.2,480,320,240 -i  living_room_traj2_loop.raw -o
benchmark.2.cuda.log 2> tmpkernels.2.cuda.log || true
cat  tmpkernels.2.cuda.log | kfusion/thirdparty/nvprof2log.py >
kernels.2.cuda.log
```

Automatic timings

When using the ICL-NUIM dataset, it is possible to generate the timings using only one command. If the living room trajectory files (raw + trajectory ground truth) are not in the directory they will be automatically downloaded.

In the following command, we use trajectory 2 and we generate the timings for the OpenCL version only:

```
make 2.opencl.log
```

In order to use all the available languages and for all the available trajectories do the following:

```
make 0.cpp.log 0.opencl.log 0.openmp.log 0.cuda.log
make 1.cpp.log 1.opencl.log 1.openmp.log 1.cuda.log
make 2.cpp.log 2.opencl.log 2.openmp.log 2.cuda.log
make 3.cpp.log 3.opencl.log 3.openmp.log 3.cuda.log
```

Parameters for different dataset trajectories (Living Room)

These parameters are also present in the Makefile and can be used with the command above, see Automatic timings section.

Trajectory 0 : offset = 0.34,0.5,0.24, voxel grid size = 5.0m (max ATE = 0.1m, mean ATE = 0.01m)

Trajectory 1 : offset = 0.485,0.5,0.55, voxel grid size = 5.0m (max ATE = 0.06m, mean ATE = 0.028m)

Trajectory 2 : offset = 0.34,0.5,0.24, voxel grid size = 4.8m (max ATE = 0.044m, mean ATE = 0.02m)

Trajectory 3 : offset = 0.2685,0.5,0.4, voxel grid size = 5.0m (max ATE = 0.292m, mean ATE = 0.117m)

File organization

```
#!Plain

SlamBench
   ├── (build)      :  will content the compilation and test result.
   ├── cmake        :  cmake module file
   └── kfusion      :  kfusion test case.
       ├── include  : common files (including the tested kernel prototypes)
       ├── src
       │   ├── cpp      : C++/OpenMP implementation
       │   ├── opencl   : OpenCL implementation
       │   ├── cuda    : CUDA implementation
       └── thridparty    : Includes several tools use by Makefile and 3rd party
headers.
```

Power measurement

Currently power measurement is only implemented on Hardkernel boards implementing power monitoring using on board sensors, ODROID-XUE and ODROID-XU3. When executing on these platforms a file power.rpt is produced at the end of each run, this will contain a frame by frame analysis of mean power and time for each frame. The power is separated between, the Cortex-A7, Cortex-A15, GPU and DRAM.

Running kfusion/thirdparty/processPowerRpt <power.rpt> will produce total energy used by each resource during the processing of the sequence.

In the Qt interface power is monitored and graphed on a frame by frame basis, the graphing will only be visible on a suitably equipped board. The power monitor, and indeed all statistics logging, is on a per sequence basis i.e. if you open/restart a scene/file the logging will restart. The statistics for a run can be saved by selecting File->Save power from the main menu when the run has been completed, statistics are only recorded when frames are processing, pausing the sequence should not impact on the final statistics.

In the future we propose to add support for power estimation using on chip counters where available, this should be included in a future release.

Set up on OS X

Warning: SLAMBench is widely tested and fully supported on Ubuntu. The OS X version may result instable. We reckon to use the Ubuntu version.

Install Homebrew:

```
#!bash

ruby -e "$(curl
-fsSLhttps://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Install and change the default to gcc and g++ (only clang/LLVM available otherwise):

```
#!bash
brew tap homebrew/versions
brew install gcc48
sudo mv /usr/bin/gcc /usr/bin/gccORIG
sudo ln -s /usr/local/bin/gcc-4.8 /usr/bin/gcc
sudo mv /usr/bin/g++ /usr/bin/g++ORIG
sudo ln -s /usr/local/bin/g++-4.8 /usr/bin/g++
sudo mv /usr/bin/c++ /usr/bin/c++ORIG
sudo ln -s /usr/local/bin/g++-4.8 /usr/bin/c++
```

OpenCL is already installed out-of-the-box no need to install. Install CUDA (guide here: http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-mac-os-x/#axzz3L7QjZMEC)

Install Qt:

```
#!bash
brew install qt
```

Add to your ~/.bashrc:

```
#!bash

export PATH=/usr/local/opt/qt5/bin/:$PATH
```

Install OpenNI and dependencies (needs MacPorts already installed https://www.macports.org/install.php): Download OpenNI here and try to run the Samples/Bin/SimpleViewer program to check if the camera is working.

```
#!bash

sudo port install libtool
sudo port install libusb + universal
```

```
cd OpenNI
./install.sh
```

In SLAMBench, modify cmake/FindOpenNI.cmake adding to the lib path:

/Users/lnardi/sw/OpenNI-MacOSX-x64-2.2/Samples/Bin

And to the include path:

/Users/lnardi/sw/OpenNI-MacOSX-x64-2.2/Include

Add to your ~/.bashrc:

```
#!bash

export
DYLD_LIBRARY_PATH=/Users/lnardi/sw/OpenNI-MacOSX-x64-2.2/Samples/Bin:$DYLD_LIBRARY_
PATH
```

Known Issues

- ** Failure to track using OpenCL on AMD ** -Some issues have been reported on AMD platforms, we will look into this
- **Visualisation using QT** - may be offset on some platforms - notably ARM
- **Build issues using QT on ARM ** - Visualisation requires opengl but Qt on ARM is often built using GLES, including packages obtained from distribution repo. Building from source with opengl set to desktop resolves this.
- ** Frame rates on QT GUI appear optimistic** - The rate shown in the status bar is by default the computation time to process the frame and render any output, it excludes the time take by the QT interface to display the rendered images and acquire frame
- ** performance difference between CUDA/OpenCL** - This is a known issue that we are investigating. It's mainly cause by a difference of global work-group size between the both version and a major slowdown of CUDA in the rendering kernels is cause by the use of float3 instead of float4 which result by an alignment issue. this alignment issue doesn't appear in OpenCL as cl_float3 are the same as cl_float4.
- ** CUDA nvprof slows down the performance on some platforms** - the nvprof instrumentation has a 2x slowdown on MAC OS for the high-level KFusion building blocks. So if we run using make 2.cuda.log we will not measure the maximum speed of the machine for the high-level building blocks. It is questionable then if we should keep measuring the CUDA high-level and low-level performance at the same time or in order to be more accurate it is better to run the two measurements in two separate runs.
- ** OS X version has not been widely tested **

Release history

Release candidat 1.1 (17 Mar 2015)
* Bugfix : Move bilateralFilterKernel from preprocessing to tracking * Bugfix : Wrong interpretation of ICP Threshold parameter. * Esthetic : Uniformisation of HalfSampleRobustImage kernel * Performance : Change float3 to float4 for the rendering kernels (No effect on OpenCL, but high performance improvement with CUDA) * Performance : Add a dedicated buffer for the OpenCL rendering * Feature : Add OSX support

Release candidat 1.0 (12 Nov 2014) * First public release