LAB 2

1 Target:

Using video output in an embedded system.

2 Demonstration:

2.1 Show `sample.bmp` on the screen on the development board (40%).

2.2 Explain your code to Tas (10%).

2.3 Answer the questions when demonstrating to Tas (20%).

2.4 Upload your source code to New E3.

2.5 Tools like `fbi` or `FIM` is not allowed.

2.6 Last date to demo: **Oct. 27th, 2022**.

3 Notes:

3.1 The following instructions will only lead you to prepare a development environment with OpenCV. If you don't need OpenCV to finish the work, you could just skip it.

3.2 We provide a non-completed source code for you to finish the important part only. However, it is not necessary to use the provided code. You can do this all by yourself.

3.3 In the provided source code, we will open a device file, `/dev/fb0`, you should check the target device is `/dev/fb0` or `/dev/fb1` on your development board. You should correct it with the corresponding device file name in the provided source code.

3.4 The framebuffer is an important idea you should know before coding. https://bit.ly/3klU829

4 Instructions:

4.1 Prepare `cmake-gui`

4.1.1 Install.

```
sudo apt-get install cmake-gui
```

4.1.2 Launch `cmake`.

```
sudo cmake-gui
```

4.2 Configure building arguments of `opencv` with cmake.

4.2.1 Unzip provided OpenCV source code `opencv-3.4.7.zip`.

```
unzip opencv-3.4.7.zip
```

4.2.2 Create directory for `opencv` libs.

```
cd /usr/local
sudo mkdir arm-opencv
cd arm-opencv
sudo mkdir build
sudo mkdir install
```

4.2.3 Back to `camke-gui` window.

4.2.4 Compilation configurations:

4.2.4.1 Click "**Browse Source…**" and **select opencv-3.4.7 source folder**.

4.2.4.2 Click "**Browse Build…**": and select path **`/usr/local/arm-opencv/build`**.

4.2.4.3 Click "**Configure**".

4.2.4.4 In the pop-up window:

4.2.4.4.1 Select "**Specify options for cross-compiling**".

4.2.4.4.2 Click "**Next**".

4.2.4.4.3 Operating System: **Linux**.

4.2.4.4.4 version: **4.1**, processor: **arm**.

4.2.4.4.5 Compilers C: select path to "**arm-linux-gnueabihf-gcc**".

(`/opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc`)

4.2.4.4.6 Compilers C++: select path to "**arm-linux-gnueabihf-g++**".

(`/opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-g++`)

4.2.4.4.7 Target Root: **`/opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf`**.

4.2.4.4.8 Click "**Finish**".

4.2.4.5 It should show "**Configure done**" at the end of the blank below.

4.2.5 `opencv` compilation configuration

You could search the below arguments in the cmake-gui window.

Checked the "Advanced" option.

4.2.5.1 Unchecked "**`Build_PERF_TESTS`**".

4.2.5.2 Unchecked "**`BUILD_TESTS`**".

4.2.5.3 Unchecked "**`BUILD_opencv_python_bindings_generator`**".

4.2.5.4 Unchecked "**`BUILD_opencv_python_tests`**".

4.2.5.5 Unchecked "**`BUILD_opencv_ts`**".

4.2.5.6 Checked "**`BUILD_opencv_world`**".

4.2.5.7 Checked "**`OPENCV_FORCE_3RDPARTY_BUILD`**".

4.2.5.8 CMAKE_CXX_FLAGS: **`-DWITH_PARALLEL_PF=OFF`**.

4.2.5.9 CMAKE_C_FLAGS: **`-DWITH_PARALLEL_PF=OFF`**.

4.2.5.10 CMAKE_EXE_LINKER_FLAGS:

(**There are no newline characters in FLAGS below**. If you copy the flags straightly, you have to delete newline characters by yourself. Otherwise, the flag will fail to generate makefile.)

```
-lpthread -ldl -lrt
-Wl,-rpath-link=/opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/qt5.5/root
fs_imx6q_V3_qt5.5_env/qt5.5_env/lib
-Wl,-rpath-link=/opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/qt5.5/root
fs_imx6q_V3_qt5.5_env/usr/lib
-Wl,-rpath-link=/opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/qt5.5/root
fs_imx6q_V3_qt5.5_env/lib
```

4.2.5.11 CMAKE_INSTALL_PREFIX: **/usr/local/arm-opencv/install.**

4.2.5.12 Checked "**ENABLE_CXX11**".

4.2.5.13 Unchecked all **WITH_***

* Only checked "**WITH_QT**" and "**WITH_V4L**".

4.2.5.14 Click "**Configure**".

4.2.5.15 It will show "**ERROR**".

4.2.5.16 Qt5_DIR:
/opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/qt5.5/rootfs_imx6q_V3_qt5.5_env/qt5.5_env/lib/cmake/Qt5.

4.2.5.17 Click "**Configure**".

4.2.5.17.1 It should show "**Configuring done**" at the end of the blank below.

4.2.5.18 Click "**Generate**".

4.2.5.18.1 It should show "**Generating done**" at the end of the blank below.

4.3 Build opencv.

```
cd /usr/local/arm-opencv/build
sudo make
sudo make install
```

4.4 Compile code using opencv.

4.4.1 Prepare your source code `source.cpp`.

4.4.2 Compilation.

```
arm-linux-gnueabihf-g++ source.cpp -o demo \
-I /opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/include/ \
-I /usr/local/arm-opencv/install/include/ -L /usr/local/arm-opencv/install/lib/ \
-Wl,-rpath-link=/opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/lib/ \
-Wl,-rpath-link=/opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/qt5.5/rootfs_imx6q_V3_qt5.5_env/lib/ \
-Wl,-rpath-link=/opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/qt5.5/rootfs_imx6q_V3_qt5.5_env/qt5.5_env/lib/ \
-Wl,-rpath-link=/opt/EmbedSky/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/qt5.5/rootfs_imx6q_V3_qt5.5_env/usr/lib/ \
-lpthread -lopencv_world
```

4.5 Run on board:

4.5.1 Copy both **shared library** and executable to the SD card.

```
/usr/local/arm-opencv/install/lib/libopencv_world.so.3.4.7 <Your SD Card>
```

4.5.2 Rename copied `libopencv_world.so.3.4.7` to `libopencv_world.so.3.4`.

4.5.3 Run executable on board.

```
LIBRARY_PATH=. ./demo
```

5 Questions: (2 points for 5.1, 3 points for 5.2~5.7)

5.1 What are the `cmake` and `make` for? What is the relationship between them?(2%)

5.2 Why there are so many arguments in the compilation command(step 4.4.2)? What are they for?

5.3 What is `libopencv_world.so.3.4.7` for? Why do we need to use `LD_LIBRARY_PATH=. ./demo` to run the executable? What would happen if we just run with `./demo`? Why?

5.4 It is so complex and difficult to show a picture by using the framebuffer. Why don't we just use `cv::imshow()` to do this work?

5.5 What is a framebuffer?

5.6 What is the result of the command below? Why?

```
sudo cat /dev/fb0 > fb0
sudo cat fb0 > /dev/fb0
```

5.7 You can find there is a file named `fb1` under `/dev` directory. What is the difference between `/dev/fb0` and `/dev/fb1`? Why we use `/dev/fb0` rather than `/dev/fb1`?

6 Advanced(30%):

Show `advanced.png` on the screen. It is ok to modify either your code or configurations above. However, **the provided picture is not allowed to be modified**. Board is **not allowed to connect to the Internet** also.