CROSS-COMPILATION FOR AR DRONE

9:11 AM Hassan 0 Comments

Installing Prerequisites

```
sudo apt-get update

sudo apt-get install build-essential linux-libc-dev wget bzip2 ncurses-dev git cma
ke cmake-curses-gui cmake-qt-gui config-manager wput
```

Downloading Compiler

```
mkdir ~/ARM_Files

wget https://sourcery.mentor.com/public/gnu_toolchain/arm-none-linux-gnueabi/arm-2
009q3-67-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2

tar jxf arm-2009q3-67-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2
```

Configuring System Paths

Create a new bash script to set up the environment variables that will help the system find the compiler.

```
gedit ~/ARM_Files/setupCrossCompiler
```

```
echo "Setting up the Cross Compiler Environment"


# Path to bin directory of the compiler

export PATH="/home/$USER/ARM_Files/arm-2009q3/bin":$PATH


# prefix of all the tools in a toolchain

export CCPREFIX="/home/$USER/ARM_Files/arm-2009q3/bin/arm-none-linux-gnueabi-"
```

Now make the script executable

```
chmod +x ~/setupCrossCompiler
```

Add the script to .bashrc so that the cross compilation environment is set up every time you open a terminal.

```
echo "source /home/$USER/ARM_Files/setupCrossCompiler" >> ~/.bashrc
```

Close and Reopen the terminal, if you see a message in the terminal saying "Setting up the Cross Compiler Environment" then your cross compilation environment is ready and you can proceed to the next step.
Writing a simple C Program We test our cross compilation setup by compiling a simple C/C++ program. Add the following hello word code to a file hello.c

```
#include <stdio.h>

int main(){

 printf("Hello Drone\n");

 return 0;

}
```

Compile and run the program on native system to test the code.

```
gcc hello.c -o PC_hello

./PC_hello
```

If this prints out "Hello Drone" on your terminal you can go ahead the compile the same code for AR Drone.

```
arm-none-linux-gnueabi-gcc hello.c -o AR_hello
```

Transfer the file to AR Drone
Now that you have the compiled AR Drone object file you can transfer the file to AR Drone via ftp and run it using telnet. Make sure you are in the directory of hello.c file and run the following commands. Just press enter when prompted to passwords.

```
ftp 192.168.1.1

put AR_hello

exit

telnet 192.168.1.1

cd /data/video

chmod +x AR_hello

./AR_hello
```

Congratulations! You just ran your first cross-compiled project on AR Drone.
For more advanced projects like cross compiling OpenCV and OpenCV Projects, see my other posts.

CROSS-COMPILE OPENCV WITH FFMPEG (X264 AND XVID) FOR AR DRONE 2.0 (ARM PROCESSOR)

9:14 AM Hassan 0 Comments

I compiled the openCV on Ubuntu 14.04 (32 bit) running on a virtual machine with root account. But the steps below should work on other platforms as well. The purpose of this project was to port opencv to ARM processor on AR Drone 2.0, with the ability of capturing the video streams published by the Drone on tcp port 5555 from within the Drone without killing its program.elf.

Pre-Requisites
Visit my previous blog entry to setup up your cross compilation environment.

Setting Up Environment Variables
Open the 'setupCrossCompiler' file created in the previous post.

```
gedit ~/ARM_Files/setupCrossCompiler
```

And add the following lines at the end of the file.

```
export ARMPREFIX=/home/$USER/ARM_Install #path where the cross compiled programs w
ill be installed
```

Compiling xVideo

```
wget http://downloads.xvid.org/downloads/xvidcore-1.3.3.tar.gz

tar -zxvf xvidcore-1.3.3.tar.gz

cd xvidcore/build/generic/

./configure --prefix=${ARMPREFIX} --host=arm-none-linux-gnueabi --disable-assembly
```

Compiling x264

```
git clone git://git.videolan.org/x264

./configure --enable-shared --host=arm-none-linux --disable-asm --prefix=${ARMPREF
IX} --cross-prefix=${CCPREFIX}
```

Compiling FFmpeg

```
git clone git://source.ffmpeg.org/ffmpeg.git

git checkout release/2.6
```

```
./configure --enable-cross-compile --cross-prefix=${CCPREFIX} --target-os=linux --arch=arm --enable-shared --disable-static --enable-gpl --enable-nonfree --enable-ffmpeg --disable-ffplay --enable-ffserver --enable-swscale --enable-pthreads --disable-yasm --disable-stripping --enable-libx264 --enable-libxvid --prefix=${ARMPREFIX} --extra-cflags="-I"${ARMPREFIX}"/include" --extra-ldflags="-L"${ARMPREFIX}"/lib"
```

Compiling OpenCV

Download OpenCV from github and checkout version 2.4.10

```
git clone https://github.com/Itseez/opencv.git

cd opencv

git checkout 2.4.10
```

Create a new toolchain file in opencv directory, this file will tell cmake how to build opencv.

```
gedit my.toolchain.cmake
```

Paste the following code in the file

```
set(ENV{PKG_CONFIG_PATH} $ENV{ARMPREFIX}/lib/pkgconfig)

set(ENV{LD_LIBRARY_PATH} $ENV{ARMPREFIX}/lib)

set(ENV{C_INCLUDE_PATH} $ENV{ARMPREFIX}/include)

set(ENV{CPLUS_INCLUDE_PATH} $ENV{ARMPREFIX}/include)


set (CMAKE_SYSTEM_NAME Linux)

set (CMAKE_SYSTEM_PROCESSOR arm)

set (CMAKE_C_COMPILER arm-none-linux-gnueabi-gcc)

set (CMAKE_CXX_COMPILER arm-none-linux-gnueabi-g++)

set (ARM_LINUX_SYSROOT /root/work/codesourcery/arm-2009q3 CACHE PATH "ARM cross compilation system root")

set (CMAKE_FIND_ROOT_PATH ${CMAKE_FIND_ROOT_PATH} ${ARM_LINUX_SYSROOT})

set(CMAKE_INSTALL_PREFIX /root/ARM_Install)

set(CMAKE_CXX_FLAGS           ""                    CACHE STRING "c++ flags")

set(CMAKE_C_FLAGS             ""                    CACHE STRING "c flags")

set(CMAKE_SHARED_LINKER_FLAGS ""                    CACHE STRING "shared linker flags")

set(CMAKE_MODULE_LINKER_FLAGS ""                    CACHE STRING "module linker flags")
```

```
set(CMAKE_EXE_LINKER_FLAGS    "-Wl,-z,nocopyreloc"  CACHE STRING "executable linke
r flags")

set(MY_FLAGS "-I$ENV{ARMPREFIX}/include -L$ENV{ARMPREFIX}/lib -lxvidcore -lx264 -l
swscale -lavformat -lavutil -lswresample -lavcodec")

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}  ${MY_FLAGS}")

set(CMAKE_C_FLAGS   "${CMAKE_C_FLAGS}  ${MY_FLAGS}")
```

Now generate the build files using toolchain file you just created.

```
mkdir build

cd build

cmake -DENABLE_PRECOMPILED_HEADERS=OFF -DWITH_FFMPEG=ON -DCMAKE_TOOLCHAIN_FILE=../
my.toolchain.cmake ../../..
```

After this is done you can scroll up to confirm if OpenCV has found FFMPEG. You should see something like the following:

```
--   Video I/O:

--     DC1394 1.x:                  NO

--     DC1394 2.x:                  NO

--     FFMPEG:                      YES

--       codec:                     YES (ver 56.35.101)

--       format:                    YES (ver 56.30.100)

--       util:                      YES (ver 54.23.101)

--       swscale:                   YES (ver 3.1.101)
```

You can add/remove openCV components using cmake-gui

```
cmake-gui .
```

Don't forget to "Configure" and "Generate" after making changes.

```
make -j2
```

Make will take a while, so grab a coffee or something. If you get a compilation error regarding some of OpenCV component(s), you can try and disable that component if you don't need it.

At this point OpenCV has been compiled now all that is left to do is install it.

```
make install
```

This will copy the cross-compiled OpenCV to your /home/$USER/ARM_Install directory.

You have successfully cross compiler OpenCV, you are now ready to build your first OpenCV program.

CROSS COMPILING OPENCV PROJECT FOR ARM, AR DRONE 2.0

2:42 AM Hassan 0 Comments

If you have followed my previous tutorial on Setting up Cross-Compilation environment and Cross-Compiling OpenCV.

You have already cross compiled OpenCV in your home/username/ARM_INSTALL directory. All you need to do to compile your project is to specify the include directory of header files and path oto shared libraries to the linker.

You can do this using

-I option to "add the directory dir to the list of directories to be searched for header files. Directories named by -I are searched before the standard system include directories."

-L option to "specify directory of linker files"

-l option to "search the named library when linking"

Creating a Test Project

Lets create a simple OpenCV project that takes an image and saves a grey-scale version of that image to disk.

```
mkdir ~/testProject

gedit imgToGrey.cpp
```

paste the following code into the file.

```
#include <cv.h>

#include <highgui.h>

using namespace cv;

int main( int argc, char** argv ){

 char* imageName = argv[1];

 Mat image;

 if( argc != 2){

   printf( "ERROR: Please specify Image name in argument\n " );

   return -1;

 }

 image = imread( imageName, 1 );

  if(!image.data ){

   printf( "ERROR: No image\n " );
```

```
   return -1;
 }
 Mat gray_image;
 cvtColor( image, gray_image, CV_BGR2GRAY );
 imwrite( "Gray_Image.jpg", gray_image );
 return 0;
}
```

Compiling for Host Machine

If you have OpenCV installed on your host computer, it would be good idea to compile and test the code on host machine. Else you can skip this.

```
g++ -I/usr/local/include/opencv -I/usr/local/include/opencv2 -L/usr/local/lib/ -g
imgToGrey.cpp -o PC_imgToGrey -lopencv_core -lopencv_imgproc -lopencv_highgui

./PC_imgToGrey test.jpg
```

Compiling for ARM

```
arm-none-linux-gnueabi-g++ -I$ARMPREFIX/include -I$ARMPREFIX/include/opencv -I$ARM
PREFIX/include/opencv2/imgproc -I$ARMPREFIX/include/opencv2 -L$ARMPREFIX/lib -g -o
AR_imgToGrey imgToGrey.cpp -lopencv_core -lopencv_imgproc -lopencv_highgui
```

Writing these lengthy commands for compilation can get cumbersome, so I have written a bash script that makes the cross-compilation process easier and automatically transfers the cross-compiled executable to AR Drone. See Cross Compilation made easy.

CROSS COMPILATION FOR AR DRONE MADE EASY

3:33 AM Hassan 0 Comments

I have written a series of bash scripts that makes cross compilation and the process of transferring the cross compiled binary to AR Drone a breeze.

telnet_delete

```
#!/bin/bash

host=192.168.1.1

port=23

echo open ${host} ${port}

sleep 0.1

echo rm /data/video/$1

sleep 0.1

echo exit
```

This script accepts file name as an argument and deletes that file on AR Drone using telnet. I use this script to delete the cross-compiled binary before cross-compilation to avoid running older version of a program in-case the compilation fails.

telnet_chmod

```
#!/bin/bash

host=192.168.1.1

port=23

echo open ${host} ${port}

sleep 0.1

echo chmod 777 /data/video/$1

sleep 0.1


echo exit
```

This script telnets into AR Drone and provides execution rights to the binary. This is necessary as file transferred via ftp looses its properties.

make

```
#!/bin/bash

#AR_CV_Path="/home/hassan/OpenCV_ARM/install"

AR_CV_Path="/home/hassan/ARM_Install"
```

```
fName=$1


#~ Identify File Name
if [ -f $fName".cpp" ]
   then srcFile=$1".cpp"
elif [ -f $fName".c" ]
   then srcFile=$1".c"
   else echo "File Not Found"
   exit
fi


#~ Object File Names
AR_OB="AR_"$1
PC_OB="PC_"$1


#~ echo $srcFile
#~ echo $AR_OB
#~ echo $PC_OB


#~ Remove Old Complilations
#~ {
rm $AR_OB
rm $PC_OB
#~ remove file from AR_Drone
if [ $2 ]
   then ./telnet_delete $AR_OB | telnet
fi
#~ } &>/dev/null # Silencer



#~ Compile for PC
```

```
echo "*******__PC__**********"

g++ -I/home/hassan/ARM_Files/boost_1_58_0 -L/home/hassan/ARM_Files/boost_1_58_0/li
bs -I/usr/local/include/opencv -I/usr/local/include/opencv2 -L/usr/local/lib/ -g -
o $PC_OB  $srcFile -lopencv_core -lopencv_imgproc -lopencv_highgui -lopencv_ml -lo
pencv_video -lopencv_features2d -lopencv_calib3d -lopencv_objdetect -lopencv_contr
ib -lopencv_legacy -lopencv_stitching -lboost_thread -lboost_filesystem -lboost_sy
stem

echo "******__ARM__**********"

#~ Compile for ARDrone

arm-none-linux-gnueabi-g++ -L/home/hassan/arm-2009q3/arm-none-linux-gnueabi/libc/l
ib -I$AR_CV_Path/include -I$AR_CV_Path/include/opencv -I$AR_CV_Path/include/opencv
2/imgproc -I$AR_CV_Path/include/opencv2 -L$AR_CV_Path/lib -g -o $AR_OB $srcFile -l
xvidcore -lx264 -lswscale -lavformat -lavutil -lswresample -lavcodec -lopencv_core
-lopencv_imgproc -lopencv_highgui -lboost_thread -lboost_filesystem -lboost_system


if [ -f $PC_OB ]

   then echo "PC Compilation Successful"

   else echo "PC FAILED!!!!"

fi


if [ -f $AR_OB ]

   then echo "AR_Drone Compilation Successful"

   if [ $2 ]

      then echo "Uploading File"

      wput -u $AR_OB ftp://192.168.1.1

      echo "chaning permissions"

      ./telnet_chmod $AR_OB | telnet

   fi

fi
```

This script takes 2 arguments, name of file to be compiled and another argument if you want to transfer the cross-compiled binary to AR Drone. You can run the script like this

```
./make testCode
```

It will look for source code file name with .c and .cpp extension, delete the old compiled binaries, compile and cross compile the source code and finally transfer and chmod the cross-compiled binary into AR Drone.

RUNNING CROSS COMPILED OPENCV PROJECT ON AR DRONE

8:17 AM Hassan 0 Comments

USB Flash Drive with AR Drone

If your compiled libraries are too large to be stored in AR Drone's internal memory, you will need to use a USB drive. Make sure that the USB drive is formatted with fat32 filesystem. I have heard that a lot of people have problems with getting AR Drone to recognize the flash drive. I have gotten lucky in this regard. Both my Kingston 16 GB and a Chinese No-Name 8 GB have worked with AR Drone flawlessly.
After you have plugged in the USB to the Drones usb port, you wait for about 15-30 seconds for the Drone's firmware to auto-mount the drive. You can do a simple

```
df -h
```

to see a list of file systems in a human readable format.
I get the following list:

```
# df -h
Filesystem                Size      Used Available Use% Mounted on
ubi1:system              26.3M     14.1M     10.9M  56% /
tmp                      57.7M    688.0K     57.0M   1% /tmp
dev                      57.7M         0     57.7M   0% /dev
ubi0:factory              4.8M     76.0K      4.5M   2% /factory
ubi2:update              13.2M     28.0K     12.5M   0% /update
ubi2:data                53.5M      8.1M     42.7M  16% /data
/tmp/udev/dev/sda1        7.6G      1.8G      5.8G  23% /tmp/udev/dev/.mnt/sda1
/tmp/udev/dev/sda1        7.6G      1.8G      5.8G  23% /data/video/usb0
```

Notice the last line. It means that my 8 GB usb flash drive in mounted on /data/video/usb0. I can navigate to this directory and see the contents of my flash drive.

Configuring Linker

You have your cross-compiled binary on you AR Drone. You have the shared libraries on the USB. Now all that is left is to configure the dynamic linker so that it can "see" the libraries on your flash drive. The dynamic linker loads and links the shared libraries needed by an executable when it is executed at run time from the environment variables $LD_LIBRARY_PATH. So we add the path to lib folder in our flash drive to $LD_LIBRARY_PATH.

```
export LD_LIBRARY_PATH=/data/video/usb0/lib
```

You will need to do this on every telnet session. Or if you are smart you will make it so that path is exported automatically.

Running the Program

Now you can simply run the program like so:

```
./programName
```

Exiting Program

If your program is in an infinite while loop, and you want to stop the program, Ctrl + C will not work here. If you want to exit a program you will need to kill the process.
Get the list of running processes with

```
ps
```

you will get a list like the following

```
 1176 root      2740 S    /bin/sh

 1229 root     43264 R    ./AR_embedded

 1259 root      2604 S    sleep 10

 1260 root      2740 S    /bin/sh
```

my program AR_embedded has process ID of 1229, now that I have noted the ID of the process I want to terminate, I can go ahead and kill it

```
kill 1229
```