Cross-Compiling the Dependencies

Pieter P

Base Image

Before building the dependencies, I created a base image with Ubuntu and the necessary tools installed.

```
FROM ubuntu:latest as base-ubuntu

# Install some tools and compilers + clean up
RUN apt-get update && \
    apt-get install -y sudo git wget \
                       gcc g++ cmake make autoconf automake \
                       gperf diffutils bzip2 xz-utils \
                       flex gawk help2man libncurses-dev patch bison \
                       python-dev gnupg2 texinfo unzip libtool-bin \
                       autogen libtool m4 gettext pkg-config && \
    apt-get clean autoclean && \
    apt-get autoremove -y && \
    rm -rf /var/lib/apt/lists/*

# Add a user called `develop`, and add him to the sudo group
RUN useradd -m develop && \
    echo "develop:develop" | chpasswd && \
    adduser develop sudo

USER develop
WORKDIR /home/develop
```

Compiling the Dependencies for a 64-bit Raspberry Pi 3B+

Preparing the Sysroot and Staging Area

Because we don't have access to the actual root directory of the Raspberry Pi, the toolchain uses a so-called sysroot to install the files to. It contains the necessary libraries, such as glibc and the C++ standard library. It's also the folder where the configure scripts of other libraries will look for the necessary libraries and headers.

The sysroot of the toolchain is read-only, to keep it clean for future projects.
We'll make a copy of the sysroot for this build, and make it writable. We'll use it as the sysroot for all compilations, and we'll also install all files to this folder.

Apart from the sysroot, we also need a folder containing the files we want to install to the Pi. It doesn't contain the system libraries, because the Pi already has these installed.

Having both a sysroot and a staging area means we have to install every library twice, once in each of the two folders.

```
FROM base-ubuntu:v1 as python

# Copy the toolchain from the previous Docker image
COPY --from=aarch64-toolchain:v1 \
    /home/develop/x-tools/aarch64-rpi3-linux-gnu \
    /home/develop/x-tools/aarch64-rpi3-linux-gnu
# Set the path to the toolchain executables
ENV TOOLCHAIN_PATH=/home/develop/x-tools/aarch64-rpi3-linux-gnu
ENV PATH=${PATH}:${TOOLCHAIN_PATH}/bin

# Create a sysroot and staging area for the RPi
WORKDIR /home/develop
ENV RPI3_SYSROOT=/home/develop/RPi3-sysroot
ENV RPI3_STAGING=/home/develop/RPi3-staging
RUN mkdir RPi3-sysroot && \
    cp -rp $TOOLCHAIN_PATH/aarch64-rpi3-linux-gnu/sysroot/* ~/RPi3-sysroot/ && \
    chmod -R u+w /home/develop/RPi3-sysroot
```

Building the Python Dependencies

For most packages, the build procedure is very simple:

1. Download
2. Extract
3. Create a build directory
4. Run the `configure` script with the right options
5. `make`
6. `make install`

To cross-compile the libraries, we have to specify the right compiler (the one we built in the previous step), and we have to tell the compiler to use our copy of the sysroot instead of the toolchain's sysroot, using GCC's `--sysroot` option.

Software that isn't managed by the system's package manager should be installed to `/usr/local`, so we specify that path as the installation prefix.
Finally, we'll install everything in the sysroot and in the staging area.

In the first section, we'll build most packages for both the build machine (the Docker container) and for the host machine (the Raspberry Pi), because we need to build Python for both machines in order to cross-compile the OpenCV and NumPy modules later on.

```
# Zlib Download
WORKDIR /home/develop
RUN wget https://downloads.sourceforge.net/project/libpng/zlib/1.2.11/zlib-1.2.11.tar.gz && \
    tar xzf zlib-1.2.11.tar.gz && rm zlib-1.2.11.tar.gz

# Zlib build
RUN mkdir zlib-1.2.11/build
WORKDIR /home/develop/zlib-1.2.11/build
RUN ../configure && \
    make -j$(($(nproc) * 2))
USER root
RUN make install
USER develop
RUN cd && rm -rf zlib-1.2.11/build

# Use the pkg-config folder inside of the RPi's root filesystem
ENV PKG_CONFIG_LIBDIR=/home/develop/RPi3-sysroot/usr/local/lib \
    PKG_CONFIG_PATH=/home/develop/RPi3-sysroot/usr/local/lib/pkgconfig \
    PKG_CONFIG_SYSROOT_DIR=/home/develop/RPi3-sysroot

# Zlib ARM
WORKDIR /home/develop
RUN mkdir zlib-1.2.11/build-arm
WORKDIR /home/develop/zlib-1.2.11/build-arm
RUN CFLAGS="--sysroot=${RPI3_SYSROOT}" \
    LDFLAGS="--sysroot=${RPI3_SYSROOT}" \
    CC="aarch64-rpi3-linux-gnu-gcc" \
    LD="aarch64-rpi3-linux-gnu-ld" \
    ../configure \
        --prefix="/usr/local" && \
    make -j$(($(nproc) * 2)) && \
    make install DESTDIR="${RPI3_SYSROOT}" && \
    make install DESTDIR="${RPI3_STAGING}" && \
    cd && rm -rf zlib-1.2.11

# OpenSSL Download
WORKDIR /home/develop
RUN wget https://github.com/openssl/openssl/archive/OpenSSL_1_1_1c.tar.gz && \
    tar xzf OpenSSL_1_1_1c.tar.gz

# Use the build system's pkg-config folders
ENV PKG_CONFIG_LIBDIR="" \
    PKG_CONFIG_PATH="" \
    PKG_CONFIG_SYSROOT_DIR=""

# OpenSSL build
WORKDIR /home/develop/openssl-OpenSSL_1_1_1c
RUN ./config --prefix="/usr/local" && \
    make -j$(($(nproc) * 2))
USER root
RUN make install_sw && \
    make distclean && \
    cd && rm -rf openssl-OpenSSL_1_1_1c
USER develop

# Use the pkg-config folder inside of the RPi's root filesystem
ENV PKG_CONFIG_LIBDIR=/home/develop/RPi3-sysroot/usr/local/lib \
    PKG_CONFIG_PATH=/home/develop/RPi3-sysroot/usr/local/lib/pkgconfig \
    PKG_CONFIG_SYSROOT_DIR=/home/develop/RPi3-sysroot

# OpenSSL ARM
WORKDIR /home/develop
RUN tar xzf OpenSSL_1_1_1c.tar.gz && rm OpenSSL_1_1_1c.tar.gz
WORKDIR /home/develop/openssl-OpenSSL_1_1_1c
RUN ./Configure \
        --prefix="/usr/local" \
        --cross-compile-prefix="aarch64-rpi3-linux-gnu-" \
        CFLAGS="--sysroot=${RPI3_SYSROOT}" \
        CPPFLAGS="--sysroot=${RPI3_SYSROOT}" \
        LDFLAGS="--sysroot=${RPI3_SYSROOT}" \
        linux-aarch64 && \
    make -j$(($(nproc) * 2)) && \
    make install_sw DESTDIR="${RPI3_SYSROOT}" && \
    make install_sw DESTDIR="${RPI3_STAGING}" && \
    cd && rm -rf openssl-OpenSSL_1_1_1c

# FFI Download
WORKDIR /home/develop
RUN wget -O libffi-3.2.1.tar.gz https://codeload.github.com/libffi/libffi/tar.gz/v3.2.1 && \
    tar xzf libffi-3.2.1.tar.gz && rm libffi-3.2.1.tar.gz

# Use the build system's pkg-config folders
ENV PKG_CONFIG_LIBDIR="" \
    PKG_CONFIG_PATH="" \
    PKG_CONFIG_SYSROOT_DIR=""

# FFI build
WORKDIR /home/develop/libffi-3.2.1
RUN ./autogen.sh && \
    mkdir build
WORKDIR /home/develop/libffi-3.2.1/build
RUN ../configure CFLAGS="-O2" CXXFLAGS="-O2" && \
    make -j$(($(nproc) * 2))
USER root
RUN make install
USER develop
RUN cd && rm -rf libffi-3.2.1/build

# Use the pkg-config folder inside of the RPi's root filesystem
ENV PKG_CONFIG_LIBDIR=/home/develop/RPi3-sysroot/usr/local/lib \
    PKG_CONFIG_PATH=/home/develop/RPi3-sysroot/usr/local/lib/pkgconfig \
    PKG_CONFIG_SYSROOT_DIR=/home/develop/RPi3-sysroot
```

```
121
122 # FFI ARM
123 WORKDIR /home/develop/libffi-3.2.1
124 RUN mkdir build-arm
125 WORKDIR /home/develop/libffi-3.2.1/build-arm
126 RUN ../configure \
127         --host="aarch64-linux-gnu" \
128         --prefix="/usr/local" \
129         CFLAGS="-O2" CXXFLAGS="-O2" \
130         --with-sysroot="${RPI3_SYSROOT}" \
131         CC="aarch64-rpi3-linux-gnu-gcc" \
132         CXX="aarch64-rpi3-linux-gnu-g++" \
133         LD="aarch64-rpi3-linux-gnu-ld" && \
134     make -j$(($(nproc) * 2)) && \
135     make install DESTDIR="${RPI3_SYSROOT}" && \
136     make install DESTDIR="${RPI3_STAGING}" && \
137     cd && rm -rf libffi-3.2.1
```

Compiling the Dependencies for a 32-bit Raspberry Pi 3B+