Qualcomm Technologies, Inc.

Machine Vision

Quick Start Guide

80-H9220-3 Rev. A

January 15, 2018

Qualcomm Snapdragon and Qualcomm Snapdragon Flight are products of Qualcomm Technologies, Inc. Other Qualcomm products referenced herein are products of Qualcomm Technologies, Inc. or its other subsidiaries.

Qualcomm and Snapdragon are trademarks of Qualcomm Incorporated, registered in the United States and other countries. Snapdragon Flight is a trademark of Qualcomm Incorporated. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer (“export”) laws. Diversion contrary to U.S. and international law is strictly prohibited.

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

© 2018 Qualcomm Technologies, Inc. All rights reserved.

1 Introduction

The Machine Vision (MV) SDK is a C/C++ programming API comprised of a library, header files, and applications.

1.1 Board Setup

For Snapdragon Flight boards with an OS version of 3.1.3.1 or earlier, the ownership of the home directory needs to be fixed after first flashing the board.

```
cd /home
sudo chown linaro linaro
sudo chgrp linaro linaro
```

After the first successful X window opens after flashing, the authority file should be copied.

```
cp /home/linaro/.Xauthority /root
```

1.2 Workstation Setup

Some of the MV applications have the ability to open an *X11* window for viewing. Linux workstations have the native ability to display these windows but *Microsoft Windows* needs a program like *Xming* installed first.

On *Windows* machines, the application *PuTTY* is commonly used for serial line connections. *PuTTY* can also be used for connecting to board via SSH for X windows forwarding by setting the option.

```
Session > Connection type
Connection > SSH > X11 > Enable X11 forwarding
```

1.3 Installation

To install MV one needs to obtain both a valid license file (`snapdragon-flight-license.bin`) and the MV Debian package (`mv_1.1.7_8x74.deb`). After that, two more steps are needed.

- Copy package from a local computer to the board.
  ```
  scp tmp/mv_1.1.7_8x74.deb linaro@<BOARD_IP_ADDRESS>:/home/linaro/mv.deb
  scp snapdragon-flight-license.bin \
  linaro@<BOARD_IP_ADDRESS>:/home/linaro/snapdragon-flight-license.bin
  ```
- Install the license to where the library will be installed or add to `LD_LIBRARY_PATH`.
  ```
  mv snapdragon-flight-license.bin /usr/lib
  ```
- Install the package on the board.
  ```
  sudo dpkg -i mv.deb
  ```

The installation puts:

- The applications into `/usr/bin`;
- The library files into `/usr/lib`;

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

- The header files into `/usr/include/mv;` and
- The application source code into `/usr/share/mv`.

1.4 Initialization After Boot

Some of the applications require access to the IMU regardless of whether or not the data is used. Therefore, it is important to start the `imu_app` after each boot if you have not already added it to your startup script. The application can be started from the command line with the following two commands:

```
sudo su
imu_app &
```

2 Applications

The applications are a good way to get going with MV. Although it will not be shown throughout the examples, because of the permission scheme currently deployed on the board, one typically needs to prepend each command line with `sudo`. For example, running `mvCameraView` with a number one afterwards would really look as follows:

```
sudo mvCameraView 1
```

These examples will also assume that one connects to the board via SSH with X forwarding enabled as previously described.

The applications are divided into two categories for easier understanding: online and offline (*i.e.*, playback). The applications are listed herein in sort of the natural order a person would use them. Therefore, the online applications are a good place to start.

2.1 Online Applications

The online applications run real-time on target to demonstrate the various technologies using the onboard sensors. There are also some extra applications beyond demonstrating technologies to help a developer get going.

mvCameraView

Since most of the SDK relies on camera data, testing the cameras out is a natural place to start. The camera view application also does not rely on the MV library so it may also be useful in debugging platform problems. The camera view application is very simple to run for the downward-facing tracking camera.

```
mvCameraView 1
```

As listed by the help option, one can try each different camera out and multiple cameras in parallel.

A real-time window should open showing the camera view. Although the camera frames are requested at a 30 FPS rate. they actual achieved rate is shown by the FPS value in the window. Because piping the X window over SSH is very processing and bandwidth intensive, the displayed frame rate is much lower. Hence, the viewed update rate may look choppy.

Because the application does not use the MV library, it implements a very rudimentary exposure control for the cameras (tracking and stereo) that do not have automatic gain and exposure control built in.

mvCPA

The CPA feature is the way MV applications typically implement automatic gain and exposure control. This application demonstrates the CPA performance in a way very similar to `mvCameraView`. Running it looks very much the same too except for the added specification of the algorithm choice.

```
mvCPA 1 1
```

The window also displays an estimate of the textured quality of the image scene. This is to give one an idea of which scenes are good for tracking and which are not. This knowledge is useful later on when doing sequence captures for calibration.

mvCapture

The capture application is one way to capture sensor data sequences that can later be used in playback applications or offline development and debugging. The application currently only supports capturing one camera at a time. To capture a sequence of data from the tracking camera, run it for a certain period of time and then control-C out.

```
mvCapture -o -t -d record_tracking
```

This sequence can then be used directly with a playback application (*e.g.*, `mvVISLAMPlayback`) or copied off the board as support data to help identify a problem. Timed captures are easy by prepending a timeout command.

```
timeout -s SIGINT 30s mvCapture -o -t -d record_tracking
```

What this application is used for most is to calibrate the cameras offline via the CAC and SAC playback applications. That calibration procedure is covered in a separate section to make the steps easier to read.

mvVISLAM

NOTE: Good performance requires that camera calibration be done first for the tracking camera and the source code be modified to include that calibration and recompiled.

The VISLAM application is very easy to run with the default calibration since it needs no parameters and will usually work with the default calibration.

```
mvVISLAM
```

The only display is the 6-DOF poses writing out to the terminal. The location will be all zero values until VISLAM initializes. To help it initialize, one needs to move and rotate the board around some while the tracking camera sees enough texture to track with (*i.e.*, not pointing at an all-white desk).

mvDFS

NOTE: Good performance requires that stereo camera and stereo rig calibration be done first. After that the source code can be modified to include those calibrations and recompiled or there is an option to read the calibrations from a file.

The application is typically run with many options specified.

```
mvDFS -m 1 -j 0 -i 0 -a -F 4 -p 0
```

Since this application was intended only for development and is very rudimentary, it is better to study the source code before attempting to use it. Therefore, running it will not be described further herein.

2.2 Playback Applications

The playback applications are examples of how to exercise the various technologies using offline data. That offline data is an SRW sequence and is typically captured by `mvCapture`, SNAV, or a developer's own application using the SRW API. Most of the playback applications take in a path to sequence data as either the first argument or via the `-s` option.

mvCACPlayback

The application is designed to run from an offline SRW sequence of data.

```
mvCACPlayback -b eagle -s 100 <PATH>/seq1
```

Ignore any AR ERROR messages. The output should end something like:

```
Reprojection error=1.024093
Inlier ratio=0.977944
DG.input-size=640x480
DG.K=276.807434,0,321.829407,0,276.807434,248.027313,0,0,1
DG.distortion=-0.002200,0.006687,-
0.002951,0.000253,0.000000,0.000000,0.000000,0
.000000
DG.distortion-model=10
DG.RBC=-0.001901,-0.189321,0.981914,-0.999962,0.008671,-0.000264,-
0.008464,-0.98
1877,-0.189330
DG.timing-offset-in-ms=-7.678854
Rolling shutter skew=0.000000ms
```

The resulting calibration file is added directly to the sequence directory.

mvDFSPlayback

NOTE: Good performance requires that stereo camera and stereo rig calibration be done first. After that the source code can be modified to include those calibrations and recompiled or there is an option to read the calibrations from a file.

The application is designed to run from an offline SRW sequence of data.

```
mvDFSPlayback -m 11 -d 0 -D 32 -s <PATH>/seq1 -c <PATH>/seq1/sac.xml
```

The output images should be in a local sub directory (`tmp/`) unless specified using `-o` option.

mvDFTPlayback

The application is designed to run from built-in data and the calibration values built into the source code match that data.

```
mvDFTPlayback
```

The end of the output should look something close to the following:

```
DFT failed. Iteration: 0
DFT Worked. Movement: x: 3.32349 y: 0.0497742
Test complete
```

mvSACPlayback

The application is designed to run from an offline SRW sequence of data:

```
mvSACPlayback -s 100 -e 200 <PATH>/seq1
```

The output should end something like:

```
Performing calibration ...
Reprojection error=0.924225
Inlier ratio=0.909164
Stereo calibration:
```

```
rRL=0.035636 0.016273 -0.005733
tRL=-0.080000 0.000000 0.000000
```

mvVISLAMPlayback

NOTE: Good performance requires that camera calibration be done first for the tracking camera and the source code be modified to include that calibration and recompiled.

The application is designed to run from an offline SRW sequence of data.

```
mvVISLAMPlayback <PATH>/seq1
```

The output should end with something like:

```
No more frames
AR INFO:
FIT:FeatureName=>SensorFusion, TSBDrift=>0.01366 (0.001)
FIT:FeatureName=>SensorFusion, TSBDrift0=>0.002 (0.000)
FIT:FeatureName=>SensorFusion, TSBDrift1=>0.003 (0.000)
FIT:FeatureName=>SensorFusion, TSBDrift2=>-0.013 (0.000)
FIT:FeatureName=>SensorFusion, RSBDrift=>0.11181
FIT:FeatureName=>SensorFusion, AccelBias0=>0.25693
FIT:FeatureName=>SensorFusion, AccelBias1=>-0.33832
FIT:FeatureName=>SensorFusion, AccelBias2=>-0.04446
FIT:FeatureName=>SensorFusion, GyroBias0=>-0.00643
FIT:FeatureName=>SensorFusion, GyroBias1=>-0.00062
FIT:FeatureName=>SensorFusion, GyroBias2=>-0.00355
FIT:FeatureName=>SensorFusion, GravityBias0=>-2.34048
FIT:FeatureName=>SensorFusion, GravityBias1=>1.08543
FIT:FeatureName=>SensorFusion, GravityBias2=>9.44810
FIT:FeatureName=>SensorFusion, FrameCount=>256
FIT:FeatureName=>SensorFusion, ImuCount=>8270
FIT:FeatureName=>SensorFusion, AvgTimeFrameUpdate=>4.05ms
FIT:FeatureName=>SensorFusion, NumResets=>3
AR INFO:
TrckLnDst:392 117 71 77 49 40 43 31 31 32 29 21 25 24 24 13 13 14 11 9 11 6
5 6 3 3 6 2 2 3 4 5 8 3 5 0 6 5 3 2 2 1 0 0 0 2 2 2 1 2 2 1 0 1 2 1 1 2 0 1
1 3 1 2 3 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 2 0 2 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
AR INFO:
Distribution of num of consec keyframes w/ GN failing all features:83 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AR INFO:
Distribution of num of consec frames w/ increasing depth uncertainty:245 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
mempool num blocks 4
Frames read: 256
Playback finished
```

mvSRWPlayback

The application is designed to create a small offline SRW sequence of data in the directory specified by the sole argument.

```
mvSRWPlayback srw
```

The output images should be in a local subdirectory (`srw/`). The images inside the `srw/camera0/` directory should look like:

There may be several error messages along the way. The end of the output should look something close to the following:

```
nCameras=1
cameraName=camera0  cameraType=mono
CameraParameters
pxlWidth=320  pxlHeight=240  memStride=320  uvOffset=0
principalPoint = (10.000000, 11.000000)
focalLength = (1.100000, 1.200000)
distortionModel=10  distortion=[2.100000, 2.200000, 2.300000, 2.400000,
2.500000, 2.600000, 2.700000, 2.800000]
Pau hanna...
```

mvVMPlayback

The application is designed to run from an offline sequence of depth data.

```
mvVMPlayback -s <PATH>/seq1
```

The output should end with something like:

```
Collision for box -0.0945612 -0.099557 -0.0970303 - 0.105439 0.100443
0.102970 at 1.96182e-44 1.96182e-44 1.96182e-44
Collision for box -0.0945612 -0.099557 -0.0970303 - 0.105439 0.100443
0.102970 at 1.96182e-44 1.96182e-44 1.96182e-44
Collision for box -0.0940224 -0.0991988 -0.0962258 - 0.105978 0.100801
0.1037740 at 1.96182e-44 1.96182e-44 1.96182e-44
```

3 Calibration

Although designed to demonstrate MV that would be integrated into a system, the applications can be used to calibrate the cameras and stereo rig as well.

3.1 Good Location

While the calibration does not need a known target such as a checkerboard, it does need a stationary scene with good texture and lighting condition. Far-away landscape in daylight works best, such as the example below.

Indoor scene such as below works too, but requires more care to get good results. Make sure the scene has enough depth (> 3 *m*), texture, and good lighting condition.

NOTE: Please inspect the input frames and make sure there's no visible motion blur. Otherwise the calibration result is unreliable. The calibration algorithm tolerates noisy or dark images, much better than blurry images.

3.2 Tracking Camera

Capture Data

Rotate the camera around all 3 axes, while minimizing translation (sideways motion).

Run the following commands:

```
rm -rf record_track
timeout -s SIGINT 30s mvCapture -o -t -d record_track
```

Calibrate Camera Intrinsic Parameters

Run the following command:

```
mvCACPlayback -b eagle -s 100 record_track -o track_undistorted.avi
```

If the calibration is successful, the output will look like below:

```
Tracking state = 0
Reprojection error=0.855193
Inlier ratio=0.989893
Distortion model error=0.588824
DG.input-size=640x480
DG.K=432.916797,0,331.528716,0,432.916797,226.316235,0,0,1
DG.distortion=0.018457,-0.032563,0.000050,-
0.000066,0.000000,0.000000,0.000000,0.000000
DG.distortion-model=10
DG.RBC=0.017096,-0.218179,0.975759,-0.999854,-0.004407,0.016533,0.000693,-
0.975899,-0.218222
DG.timing-offset-in-ms=-8.629679
```

It will also output a video file (`track_undistorted.avi`). It is always a good idea to make sure the undistorted video looks reasonable.

The output distortion model can be specified by -m flag. The default model is the fisheye model (model 10). Change to model 4, 5, or 8 for 4, 5, or 8 parameter polynomial models. The calibration uses fisheye model internally and converts to polynomial models at the end. Distortion model error measures the conversion error. A warning message is generated if the error is large. In that case please consider using higher order model or fisheye model to reduce the conversion error.

NOTE: The polynomial models always have zeros in the tangential distortion parameters (third and fourth parameter), because fisheye model is radial distortion only.

It will also generate a file at `record_track/cacLeft.cal`, which contains pretty much the same information:

```
<?xml version='1.0' encoding='UTF-8'?>
<Camera>
<Parameters>
<Intrinsics size="640 480" pp="331.528716 226.316235" fl =
"432.916797 432.916797" distModel = "10" dist = "0.018457 -0.032563
0.000050 -0.000066 0.000000 0.000000 0.000000 0.000000"/>
</Parameters>
</Camera>
```

NOTE: The distortion model used in `*.cal` file is always fisheye model, regardless the `-m` flag value.

NOTE: The example above is to show the format of the output. The calibration parameter values in the example may not reflect the true values.

Store Calibration

Because the calibration is particular to the board and not any specific data sequence, it is common to move the final calibration files to a well-known location (*e.g.*, `/usr/share/mv/cal/`) to be used by all applications.

Quick Tip for Troubleshooting

`mvCACPlayback` takes a considerable amount of time to finish, and it is very time consuming to debug by looking at the final result. You can use the `-v` flag to display frame-by-frame information as shown below:

```
Processed frame 100
TS(-1); RE(inf); IR(0.00); PP(0.50,0.50); FR(0.60,0.60); D(0.00,0.00);
tGC(0.0); tRSS(0.0);
Processed frame 101
TS(-1); RE(1.02); IR(0.97); PP(0.33,0.50); FR(0.26,0.26); D(-0.14,0.28);
tGC(9.6); tRSS(0.3);
Processed frame 102
TS(-1); RE(0.82); IR(0.98); PP(0.50,0.45); FR(0.31,0.31); D(-0.14,0.34);
tGC(9.0); tRSS(0.7);
Processed frame 103
TS(-1); RE(0.71); IR(0.99); PP(0.50,0.50); FR(0.31,0.31); D(-0.05,0.08);
tGC(8.1); tRSS(1.2);
Processed frame 104
TS(-1); RE(0.66); IR(0.99); PP(0.49,0.50); FR(0.31,0.31); D(0.01,-0.02);
tGC(8.4); tRSS(1.2);
Processed frame 105
TS(-1); RE(0.63); IR(0.99); PP(0.49,0.50); FR(0.31,0.31); D(-0.03,0.02);
tGC(8.4); tRSS(1.2);
Processed frame 106
TS(1); RE(0.69); IR(0.99); PP(0.49,0.48); FR(0.31,0.31); D(-0.02,0.02);
tGC(9.2); tRSS(1.0);
Processed frame 107
TS(1); RE(0.76); IR(0.99); PP(0.49,0.49); FR(0.30,0.30); D(-0.00,0.00);
tGC(9.2); tRSS(0.9);
Processed frame 108
TS(1); RE(0.77); IR(0.99); PP(0.49,0.52); FR(0.31,0.31); D(-0.06,0.04);
tGC(8.5); tRSS(1.1);
Processed frame 109
TS(1); RE(1.04); IR(0.98); PP(0.49,0.51); FR(0.31,0.31); D(-0.03,0.02);
tGC(8.3); tRSS(1.3);
Processed frame 110
TS(1); RE(1.02); IR(0.98); PP(0.49,0.50); FR(0.30,0.30); D(-0.01,-0.01);
tGC(8.4); tRSS(1.2);
Processed frame 111
TS(1); RE(1.01); IR(0.98); PP(0.49,0.50); FR(0.30,0.30); D(-0.03,0.02);
tGC(8.3); tRSS(1.7);
Processed frame 112
TS(1); RE(0.97); IR(0.98); PP(0.49,0.49); FR(0.30,0.30); D(-0.01,0.01);
tGC(8.5); tRSS(1.4);
```

```
Processed frame 113
TS(1); RE(0.95); IR(0.98); PP(0.49,0.49); FR(0.30,0.30); D(-0.01,0.00);
tGC(8.5); tRSS(0.9);
```

TS: tracking state (0: high quality, 1: low quality, -1: initializing, -2: failed), RE: reprojection error in pixels, IR: inlier ratio, PP: normalized principal point, FR: focal ratios in X and Y, D: first two distortion parameters, tGC: time offset from camera to gyro in milliseconds, tRSS: adjustment to initial rolling shutter skew in milliseconds.

One can usually tell whether CAC will converge by looking at the first 50 frames or so.

3.3 Stereo Camera Calibration

Capture Data

Run the following command while rotating the camera:

```
rm -rf record_stereo
timeout -s SIGINT 30s mvCapture -s -t -d record_stereo
```

If too many MIPI messages, one can turn them off with:

```
dmesg -n 1
```

See the tracking camera section for more details.

Calibrate Intrinsic Parameters

Calibrate the left camera by running:

```
mvCACPlayback -b eagle -s 100 record_stereo -o left_undist.avi
```

The output file is located at `record_stereo/cacLeft.cal`. Calibrate the right camera by running:

```
mvCACPlayback -b eagle -s 100 -u record_stereo -o right_undist.avi
```

The output file is located at `record_stereo/cacRight.cal`. See the tracking camera section for more details.

Calibrate Extrinsic Parameters

Run the following command:

```
mvSACPlayback -s 100 -e 200 record_stereo -o stereo_undist.avi
```

`mvSACPlayback` reads intrinsic parameters from `record_stereo/cacLeft.cal` and `record_stereo/cacRight.cal`. The output looks like below:

```
Reprojection error=0.659759
Inlier ratio=0.960402
Stereo calibration:
rRL=0.007858 -0.017013 -0.003425
tRL=-0.080000 0.000000 0.000000
```

It will also generate a file named `record_stereo/sac.xml`, which can be used directly by `mvDFS`.

NOTE: The default distortion model in `sac.xml` is fisheye model. Use the `-m` flag to change to other models.

Verify calibration quality

Play the output video file (`output.avi`). Far away objects should appear gray, without any red or green edges.

Store Calibrations

Because the calibration is particular to the board and not any specific data sequence, it is common to move the final calibration files to a well-known location (*e.g.*, `/usr/share/mv/cal/`) to be used by all applications.

3.4 High-Resolution Camera Calibration

Capture Data

Run the following command while rotating the camera:

```
timeout <ARGS> mvCapture -r -t -p 2 -d record_hires
```

The `-p 2` option sets the resolution to 720p. One may want to use a different resolution to match the aspect ratio of the intended use case. Please use caution when going higher than 720p, as it may strain the internal storage space and file I/O. Usually one can calibrate at lower resolution and scale the result.

See downward facing camera section for more details.

Calibrate Camera Intrinsic Parameters

Run the following command:

```
mvCACPlayback -b eagle -s 100 record_hires
```

See downward facing camera section for more details.

Store Calibration

Because the calibration is particular to the board and not any specific data sequence, it is common to move the final calibration files to a well-known location (*e.g.*, `/usr/share/mv/cal/`) to be used by all applications.