

---

# **smarttof Documentation**

***Release 1.62.0***

**smarttof**

**Feb 01, 2019**



# 介绍 (Introduction)

<b>1 概述</b>	<b>1</b>
1.1 SmartToF 模组介绍 . . . . .	1
1.2 SmarToF SDK 介绍 . . . . .	1
1.3 SDK 文档说明 . . . . .	3
<b>2 基本开发流程</b>	<b>5</b>
2.1 开发流程框图 . . . . .	5
2.2 开发流程说明 . . . . .	7
2.2.1 模组基本性能评估 . . . . .	7
2.2.2 模组样例集成开发 . . . . .	7
2.2.3 模组算法评估开发 . . . . .	7
<b>3 快速入门</b>	<b>9</b>
3.1 SmartToF SDK 配置需求 . . . . .	9
3.1.1 最低配置 . . . . .	9
3.1.2 推荐配置 . . . . .	9
3.1.3 SmartToF SDK 运行要求 . . . . .	10
3.2 模组连接 . . . . .	10
3.2.1 windows 下模组连接 . . . . .	10
3.2.2 linux 下模组连接 . . . . .	12
3.3 工具初步使用 . . . . .	13
3.3.1 SmartToFViewer 使用 . . . . .	13
3.3.2 dmcam-cli 命令行工具使用 . . . . .	16
<b>4 C/C++</b>	<b>19</b>
4.1 采集 . . . . .	19
4.2 参数设置 . . . . .	20
4.3 滤波 . . . . .	21
4.4 录像 . . . . .	21
4.5 编译生成 . . . . .	22
4.5.1 在 windows 下生成 vs 工程 . . . . .	22
4.5.2 在 windows 下生成可执行文件 . . . . .	24

<b>5 Python</b>	<b>25</b>
5.1 最简采集 . . . . .	25
5.2 参数设置 . . . . .	26
5.3 采集 UI 显示 . . . . .	27
5.4 运行 . . . . .	28
5.4.1 运行 python 样例相关包的安装 . . . . .	28
<b>6 C#</b>	<b>31</b>
6.1 最简采集 . . . . .	31
6.2 采集 UI 显示 . . . . .	32
6.3 编译生成 . . . . .	32
6.3.1 windows 平台 . . . . .	32
6.3.2 Linux 平台 . . . . .	33
<b>7 Java</b>	<b>35</b>
7.1 最简采集 . . . . .	35
7.2 采集 UI 显示 . . . . .	36
7.3 编译生成 . . . . .	37
7.3.1 windows 平台 . . . . .	37
7.3.2 linux 平台 . . . . .	37
<b>8 ROS</b>	<b>39</b>
8.1 ROS 深度显示 . . . . .	39
8.2 ROS 灰度显示 . . . . .	40
8.3 ROS 点云显示 . . . . .	41
8.4 ROS 下使用模组环境搭建 . . . . .	42
8.4.1 Ubuntu 下安装 ros . . . . .	42
8.4.2 ROS 环境配置 . . . . .	43
8.4.3 SmartToF ros 包编译 . . . . .	43
8.5 ROS rviz 工具使用 . . . . .	44
8.5.1 rviz 使用前准备 . . . . .	44
8.5.2 rviz 显示深度图像 . . . . .	44
8.5.3 rviz 显示点云图像 . . . . .	45
8.6 ROS 滤波 . . . . .	46
8.6.1 滤波功能的使用 . . . . .	46
<b>9 Openni2</b>	<b>49</b>
9.1 Niviewer 采集显示 . . . . .	49
9.2 最简采集 . . . . .	50
9.3 参数设置 . . . . .	52
<b>10 Android</b>	<b>53</b>
10.1 APP 采集显示 . . . . .	53

<b>11 工具详细说明</b>	<b>57</b>
11.1 SmartToFViewer 使用说明 . . . . .	57
11.1.1 简介 . . . . .	57
11.1.2 界面介绍 . . . . .	57
11.1.3 详细使用说明 . . . . .	58
11.2 SmartToFCli 使用说明 . . . . .	65
11.2.1 工具概述 . . . . .	65
11.2.2 工作方式 . . . . .	66
11.2.3 详细命令 . . . . .	66
<b>12 C/C++ 核心库 (libdmcam) 参考</b>	<b>73</b>
12.1 核心 API . . . . .	73
12.1.1 dmcam.h . . . . .	73
12.2 模组参数和滤波类型说明 . . . . .	88
12.2.1 模组参数说明 . . . . .	88
12.2.2 模组滤波类型说明 . . . . .	89
12.3 参数和滤波代码示例 . . . . .	90
12.3.1 模组参数设置和读取 . . . . .	90
12.3.2 滤波功能开启和关闭 . . . . .	91
12.4 录像生成与读取回放代码说明 . . . . .	93
12.4.1 录像文件的设置 . . . . .	94
12.4.2 录像文件的读取 . . . . .	94
<b>13 Python 扩展</b>	<b>97</b>
13.1 dmcam python 扩展概述 . . . . .	97
13.1.1 python 扩展的安装 . . . . .	97
13.1.2 python API 说明 . . . . .	97
13.2 python 中参数设置和滤波相关 . . . . .	99
13.2.1 python 下参数设置和读取 . . . . .	100
13.2.2 python 下滤波功能开启和关闭 . . . . .	100
<b>14 C# 扩展说明</b>	<b>103</b>
14.1 dmcam C# 扩展概述 . . . . .	103
14.1.1 C# 扩展的安装 . . . . .	103
14.1.2 C# API 说明 . . . . .	104
<b>15 Java 扩展说明</b>	<b>107</b>
15.1 dmcam Java 扩展概述 . . . . .	107
15.1.1 Java 扩展的安装 . . . . .	107
15.1.2 Java API 说明 . . . . .	108
<b>16 Android 扩展说明</b>	<b>111</b>
16.1 dmcam Android 扩展概述 . . . . .	111

16.1.1	Android 扩展的安装	111
16.1.2	Android API 说明	111
<b>17</b>	<b>ROS 扩展</b>	<b>115</b>
17.1	ROS 设计介绍	115
17.1.1	ROS 概述	115
17.1.2	ROS 框架	115
17.2	ROS API 说明	116
17.2.1	dmcam_ros 发布的话题	116
17.2.2	dmcam_ros 发布的服务	117
<b>18</b>	<b>OpenNI2 扩展</b>	<b>121</b>
18.1	OpenNI2 驱动说明	121
18.1.1	SmartToF 模组的设置	121
<b>19</b>	<b>Indices and tables</b>	<b>123</b>
<b>Index</b>		<b>125</b>

# Chapter 1

## 概述

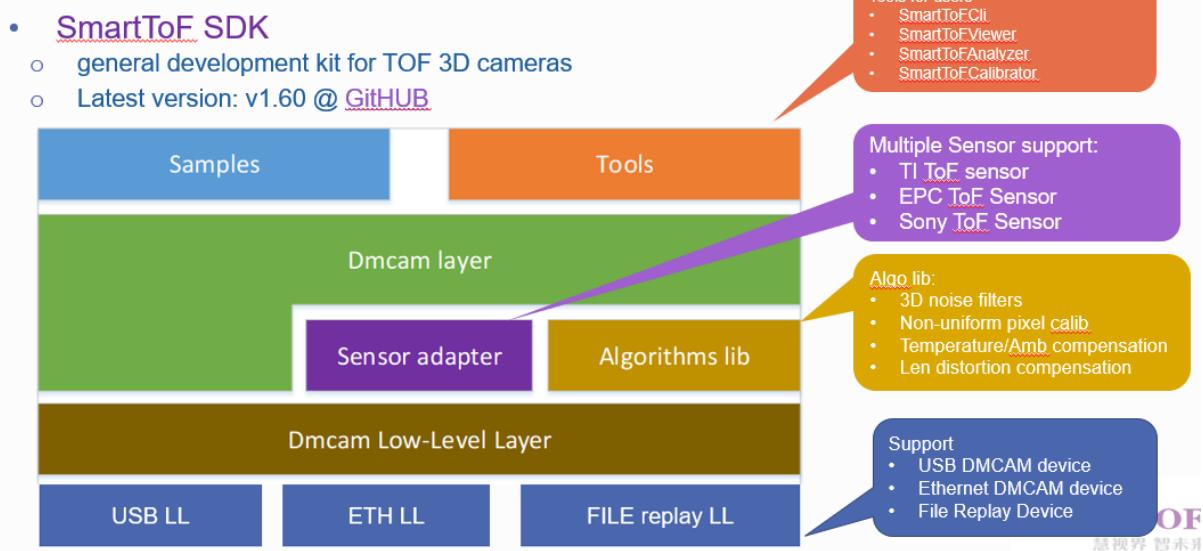
欢迎来到 SmartToF 模组的 SDK 文档，通过这个文档你可以了解到 SmartToF 系列模组配套 SDK 包含的内容、SDK 中工具的使用方法、SmartToF 模组二次开发时的一般过程以及相关 API 的使用说明。

### 1.1 SmartToF 模组介绍

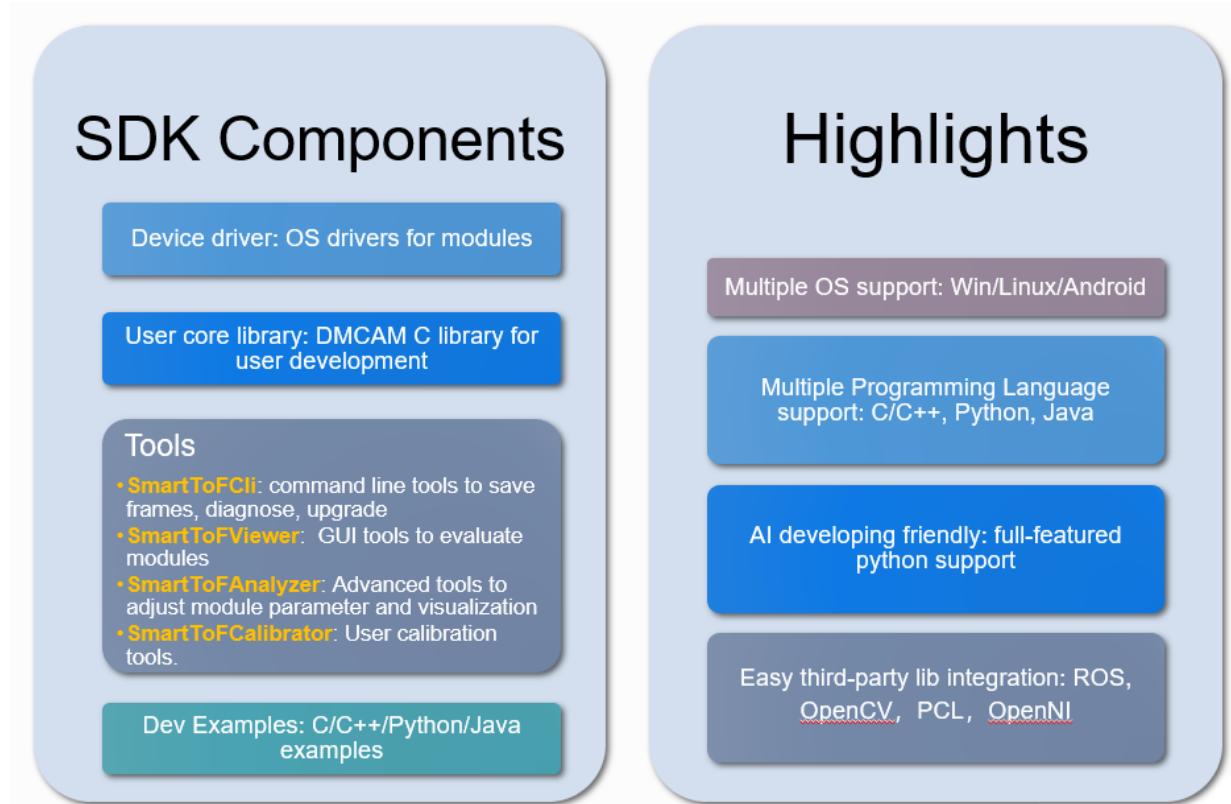
SmartToF TC 系列模组是数迹公司采用 TOF 技术开发的 3D 视觉模组，采用业界领先的传感器芯片，具有测量精度高、抗干扰能力强、外观小巧等优点。模组可用于精确的人流统计、物流仓储、手势识别、机器人避障和车载控制等新兴技术领域。

### 1.2 SmarToF SDK 介绍

SmartToF SDK 是配套 SmartToF 系列模组进行开发的软件工具包，支持 windows、linux、Android 等主流平台，SDK 的总体架构图如下：



SDK 中架构中的主要部分说明和特点如下图所示：



SDK 中支持的多平台列表如下所示：

Table 1: 表 1 SDK 支持说明

内容	Windows	Linux	Android
核心 API C 库	*	*	*
Python	*	*	
java	*	*	*
ROS		*	
C#	*	*	
MATLAB			
USB driver	*	*	*
SmartTofAnalyzer	*		
SmartTofCli	*	*	
SmartTofViewer	*	*	*

## 1.3 SDK 文档说明

本 SDK 文档说明是由数迹公司人员持续编写、校正、修改的，本文档主要包括四个部分，每个部分的内容根据实际情况分布，每个部分的内容是相对直观的。

- 介绍 (*Introduction*) 部分概述了对 SmartToF 模组和 SDK 的总体介绍，列出了 SDK 的相关资源，概述还包含了 SmartToF 模组的基本开发流程。
- 开始 (*GettingStart*) 部分包含使用平台快速搭建、工具的相关使用。
- 教程 (*Tutorial*) 教程部分展示了在主要平台下配套 SDK 样例的使用和相关图示，样例中的部分代码可以作为二次开发时参考。
- 详细参考 (*Reference*) 对 SDK 中核心 API 库、python、Java、C#、ros 和 Android 等作了扩展说明，在进行模组的二次开发时，如果需要了解详细的信息，可以从本章内容中找到。

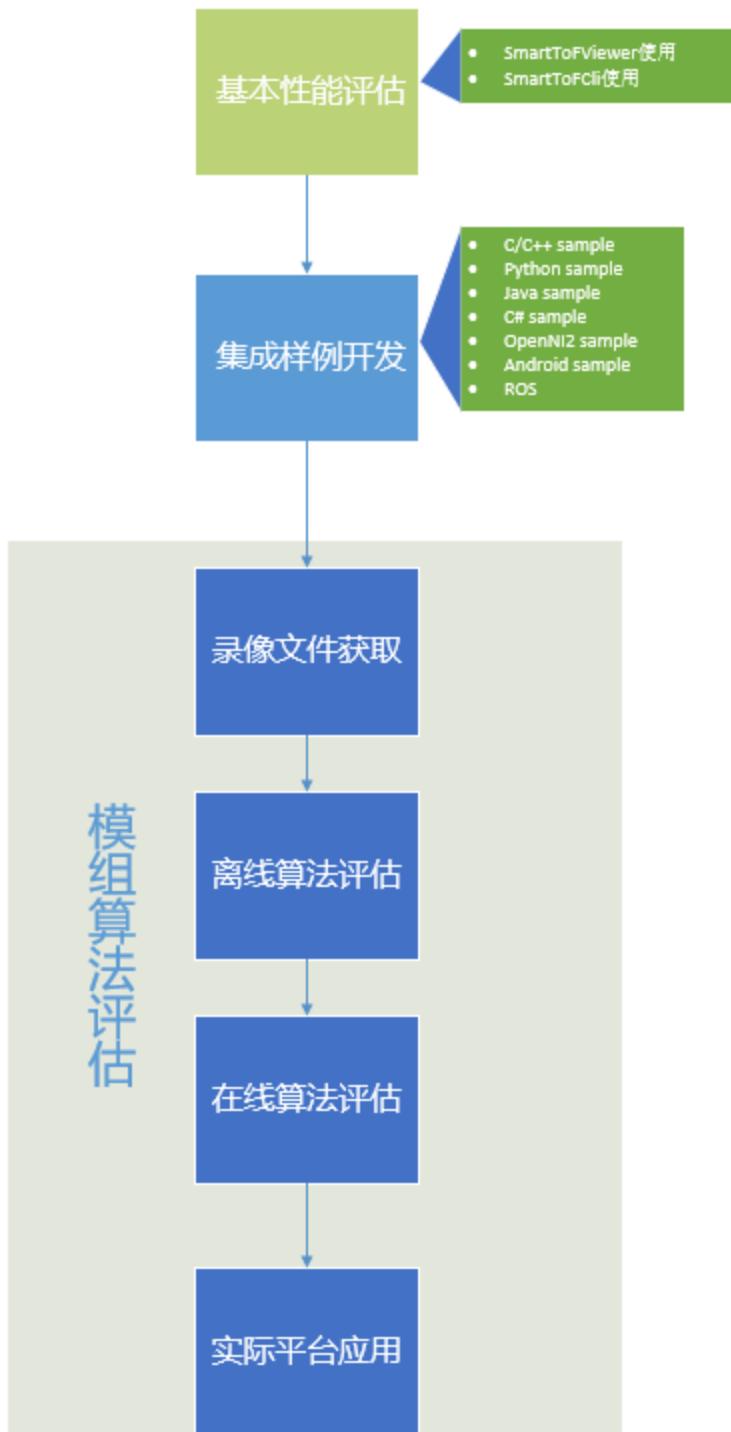


# Chapter 2

## 基本开发流程

### 2.1 开发流程框图

在对 Smartof 模组进行二次开发时，一般的开发流程图如下



## 2.2 开发流程说明

### 2.2.1 模组基本性能评估

在获得 SmartToF 模组后，使用 SDK 中 tools 目录下提供的 SmartToFViewer 工具进行被测物体的实时显示。显示时根据被测物体的远近、运动状态等调节 SmartToFViewer 上的相关参数，通过显示效果评估模组的成像质量、深度距离的精确度等信息。如果要评估模组的点云效果，需要在开启 SmartToFViewer 的同时开启 SmartToF\_PCLViewer。

### 2.2.2 模组样例集成开发

通过使用 SmartToFViewer 进行模组的评估，能够基本了解模组的一些参数和滤波功能，后续可以按照教程 (*Tutorial*) 的说明熟悉和运行 SDK 中提供的样例。掌握自己所要使用的开发平台和语言环境下使用 SDK 的主要步骤，参考样例代码，代码中相关的 API 详细说明参考 [核心 API](#) 中的对应章节，编写集成自己的测试样例进行模组数据的采集和显示。

- C/C++ 的用户参考 *C/C++* 的相关样例和编译运行 C/C++ 样例的方法。
- Python 用户参考 *Python* 的相关样例和运行样例方法。
- Java 用户参考 *Java* 的 basic 和 basicUI 样例以及运行样例的方法。
- C# 用户参考 *C#* 的 basic 和 basicUI 样例以及运行样例的方法。

### 2.2.3 模组算法评估开发

为了方便快捷的使用 smartttof 模组进行算法开发，SmartToF SDK 提供了一套完整的开发流程说明，将整个开发流程分为录像文件获取、离线算法评估、在线算法评估、实际平台应用等四个阶段，每一阶段的具体说明如下。

#### 录像文件获取

算法评估的基础是建立在完整准确的数据之上，针对前期没有模组进行数据采集或者不确定采集的图像数据是否正确的情况下，可以先期通过 smartttofviewer 的录像功能进行获取，也可以通过 dmcam-cli 工具的 rx 命令采集。如何使用 Smartttofviewer 的录像功能录制录像文件参考 [SmartToFViewer 录像功能](#) 中的工具说明，使用 dmcam-cli 工具的 rx 命令采集保存文件参考 `help rx` 帮助说明。

#### 离线算法评估

在获得录像文件后，通过调用 `dmcam_dev_open_by_uri` 接口打开录像文件，将录像文件模拟成标准的模组 DMCAM 设备。然后在获得的录像文件上加载运行评估算法，对比算法加入后运行的实际效果，如要对原始深度数据进行滤波，可以加入常用的中值或者双边滤波，也可以是改进后的深度滤波，通过滤波前后的图像

效果评估算法效果。这种离线算法评估解决了部分用户在没有模组或者未能正常采集模组数据的情况下，能够正常进行算法评估的相关工作。

### 在线算法评估

进行在线算法评估时，需要打开实际的 SmartToF 设备，打开设备的 API 不再跟打开录像文件时的 API 相同，为 `dmcam_dev_open` 接口。经过前期在离线录像文件上的算法评估，基本确定被评估算法的效果，判断评估算法是否达到设计要求，后续就是在模组上进行动态的实际效果的算法评估。在采集 SmartToF 模组实时数据的同时加入前面离线评估的算法处理，实时观察和评测算法在 pc 上的实际效果，最终确定在 SmartToF 上使用的算法是否达到使用要求。

---

**Tip:** 离线算法评估和在线算法评估的主要区别就是打开的设备不同，离线算法评估打开的是由录像文件模拟的设备，在线算法评估打开的是真实的模组设备。打开设备时的调用 API 也不相同，离线算法打开录像文件时调用的接口是 `dmcam_dev_open_by_uri`，在线算法打开正常模组时调用 `dmcam_dev_open` 接口。

---

### 实际平台应用

前期的离线算法评估和在线算法评估主要是基于 PC 平台，在实际应用中，SmartToF 模组可能需要被运行在各种不同的嵌入式平台。这时需要在对应的平台上运行相应的 SmartToF 的库，同时将前面的评估算法移植到对应的平台，并根据平台对算法进行相应的优化，最后在实际的嵌入式平台上进行应用的开发。

# Chapter 3

## 快速入门

### 3.1 SmartToF SDK 配置需求

#### 3.1.1 最低配置

Table 1: x86 或 x64 平台最低配置

CPU	Intel Atom X5-8350
内存	2GB
接口	USB2.0

Table 2: ARM 平台最低配置

CPU	Cortex-A7
内存	256M DDR3 RAM
接口	USB2.0

#### 3.1.2 推荐配置

Table 3: x86 或 x64 平台推荐配置

CPU	Intel Core i5
内存	4GB
接口	USB2.0

Table 4: ARM 平台推荐配置

CPU	Cortex-A53
内存	512M DDR3 RAM
接口	USB2.0

### 3.1.3 SmartToF SDK 运行要求

Table 5: Windows 下

数据接口	USB2.0
操作系统	Windows 7 / 10(32 位和 64 位)
编译器	Visual Studio 2013 或以上
编程语言	C/C++,C#,Java,Python

Table 6: Linux 下

数据接口	USB2.0
操作系统	Ubuntu 14.04 / 18.04(64 位)
编译器	GCC 4.8 或者更高
编程语言	C/C++,C#,Java,Python

## 3.2 模组连接

### 3.2.1 windows 下模组连接

在 windows 下连接使用模组时，需要安装模组的 usb 驱动，模组的一般安装流程如下：

#### 模组准备

模组套件包括 TC 系列模组、MicroUSB 线和电源（部分型号无需电源）如表 2 所示

Table 7: 表 2 模组套件

1	TC 系列模组		
2	Micro USB 电缆		
3	12V 电源		

## 模组安装

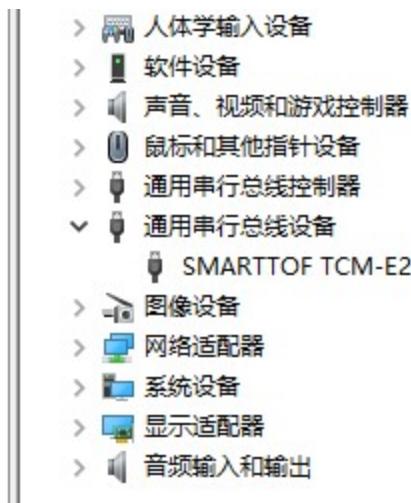
在 windows 系统下，运行 SDK 中 windows/drivers 目录下的 smarttof\_usb\_install.exe 程序进行驱动的安装，正常安装过程如下图所示

```
OPT: VID number 4379
OPT: PID number 4664
OPT: driver number 0
libwdi:info [extract_binaries] successfully extracted driver files to driver
libwdi:info [wdi_prepare_driver] successfully created 'driver\DMiracle_SmartTOF.inf'
libwdi:info [wdi_prepare_driver] Vista or later detected - creating and self-signing a .cat file...
libwdi:info [ScanDirAndHash] added hash for 'E:\WORK\SVN_ALL\packedSDK\smarttof_sdk_v1.31\windows\drivers\driver\amd64\wdfcoinstaller01011.dll'
libwdi:info [ScanDirAndHash] added hash for 'E:\WORK\SVN_ALL\packedSDK\smarttof_sdk_v1.31\windows\drivers\driver\amd64\winusbcoinstaller2.dll'
libwdi:info [ScanDirAndHash] added hash for 'E:\WORK\SVN_ALL\packedSDK\smarttof_sdk_v1.31\windows\drivers\driver\dmiracl_e_smarttof.inf'
libwdi:info [ScanDirAndHash] added hash for 'E:\WORK\SVN_ALL\packedSDK\smarttof_sdk_v1.31\windows\drivers\driver\x86\wdfcoinstaller01011.dll'
libwdi:info [ScanDirAndHash] added hash for 'E:\WORK\SVN_ALL\packedSDK\smarttof_sdk_v1.31\windows\drivers\driver\x86\winusbcoinstaller2.dll'
libwdi:info [CreateCat] successfully created file 'driver\DMiracle_SmartTOF.cat'
libwdi:info [RemoveCertFromStore] deleted existing certificate 'CN=USB\DMiracle\SmartTOF' from 'Root' store
libwdi:info [RemoveCertFromStore] deleted existing certificate 'CN=USB\DMiracle\SmartTOF' from 'TrustedPublisher' store
libwdi:info [CreateSelfSignedCert] created new self-signed certificate 'CN=USB\DMiracle\SmartTOF'
libwdi:info [SelfSignFile] added certificate 'CN=USB\DMiracle\SmartTOF' to 'Root' and 'TrustedPublisher' stores
libwdi:info [SelfSignFile] successfully signed file 'driver\DMiracle_SmartTOF.cat'
libwdi:info [SelfSignFile] successfully deleted private key
installing driver ... OK

Press any key to exit
```

搜狗拼音输入法 全 :

驱动正常安装后，将 USB 电缆连接模组和 PC，打开设备管理器中可以看到模组的设备名，如下图所示



### 3.2.2 linux 下模组连接

#### libusb 的安装

在使用 Linux 的发行版 ubuntu14 和 ubuntu16 时，默认自带 libusb，无需安装 usb 驱动，如果需要重装 libusb，删除原来的 libusb 后，在终端中重新输入命令进行安装：

```
sudo apt-get install libusb-1.0-0-dev
```

然后运行 SDK 目录边 tool/SmartToFViewer 目录下的 smartTOFViewer 工具查看设备是否连接正常，正常连接如下图所示显示设备号



## libusb 权限修改

为了便于每次使用模组时，不用重复输入密码，需要给 usb 增加相应的规则，SDK 中已经提供规则设置脚本，只要运行在 linux/lib 目录下的 setup.sh 脚本，运行脚本过程如下图

```
nux/lib$ ./setup.sh
/mnt/hgfs/packedSDK/Smarttof_sdk_v1.60/SDK-smarttof_sdk_v1.60/linux/lib
udev stop/waiting
udev start/running, process 3034
Bus 001 Device 002: ID 111b:1238
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
your system is 64bit
/mnt/hgfs/packedSDK/Smarttof_sdk_v1.60/SDK-smarttof_sdk_v1.60/linux/lib/linux64
```

## 3.3 工具初步使用

SDK 配套 SmartToFViewer 显示评估工具和 dmcam-cli 命令行工具，分别在 SDK 中的 tools 目录下，SmartToFViewer 同时提供 smartTOF\_PCLViewer 点云显示工具。

### 3.3.1 SmartToFViewer 使用

SmartToFViewer 可以直接用来评估被测物体图像，并通过 UI 设置模组相关参数，查看模组的相关信息和工作状态，SmartToFViewer 在模组 usb 正常安装时即可使用。

#### SmartToFViewer 界面介绍

SmartToFViewer 打开后的整体预览如下，SmartToFViewer 主要包括图像显示区、基础参数区、filter 设置区、信息区以及模组的开启关闭：



### SmartToFViewer 采集显示

双击运行 SmartToFViewer 工具，点击选择设备，如果同时连接多台模组，会列出设备列表，左击选择要使用的设备后点击图中的 OK 按钮，如下图：



选取设备后点击 开始按钮就可以采集图像显示了，默认视图模型为深度图-彩色编码，如下图所示，图中显示了物体的深度距离信息。拖动或者点击界面上的相关按钮，可以对模组的采集参数设置和滤波功能使能，具体参考详细说明中的 SmartToFViewer 的详细说明。

### SmartToFViewer 录像功能

SmartToFViewer 支持录像功能，录制的视频文件可以作为离线算法评估的图像数据。Viewer 打开后选择好设备后，点击图中的 录像设置选项卡，并勾选 录像使能，选择录像文件的保存地址，如下图：

**Caution:** 在 录像设置选项卡下还有个 OpenNI 格式兼容选项，该选项功能说明如下

- 使能模式下，如果用 NiViewer 播放录像文件时，输出深度图和灰度图
- 不能模式下，用 NiViewer 播放录像文件时，输出 4DCS 的原始图像。



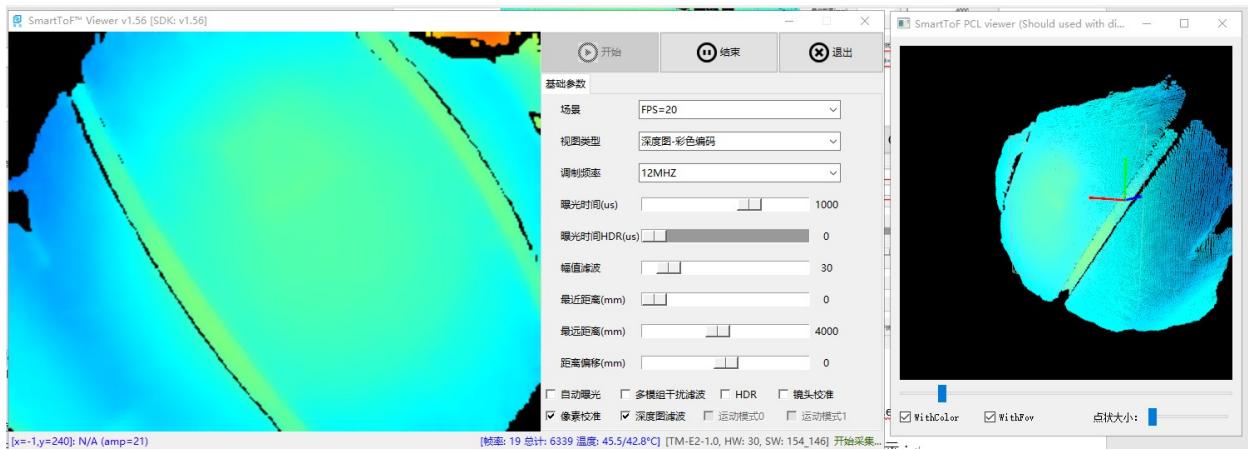
点击 SmartToFViewer 上的 结束则停止录像，并将 oni 格式的录像文件保存到刚才选择的地址。

### SmartToFViewer 播放录像文件

打开 SmartToFViewer，点击 打开回放按钮，选择保存的录像文件，选中后 SmartToFViewer 右下角的 USB 设备号变成录像的文件名，如下图所示



如果要开启点云图，则需要同时开启 SmartToFViewer 和 SmartToF\_PCLViewer：如下图



### 3.3.2 dmcam-cli 命令行工具使用

SDK 中的 dmcam-cli 工具是方便用户在二次开发时进行诊断和测试使用，主要包括以下几个功能：

- 硬件设备信息获取
- 硬件参数设置
- 数据采集和保存
- 固件更新

### dmcam-cli 基本信息获取

dmcam-cli 通常通过命令行参数方式、脚本文件方式、交互模式与硬件设备进行交互，如下图展示了命令行参数模式和交互模式获取设备信息：

```
niph@DESKTOP-I7KSTQ3 MINGW32 /e/Workdata/packedSDK/Smarttof_sdk_v1.60/SDK-smarttof_sdk_v1.60/windows/tools/SmartTofCli
$ ./dmcam-cli.exe --print info

Vendor : Data Miracle
Product : TM-E2-1.0
DevID   : 0xf558d0170701004d

Capability:
  max frame width      : 320
  max frame height     : 240
  max frame depth      : 2
  max fps               : 30
  max integrate time(us) : 1500

  Serial string : 07018039AA6D94A95834C487
  Serial info   : 0x07018039, 0xAA6D94A9, 0x5834C487

  Sensor info: 7, 257, 77

DFU Information:
  HW Version : 30
  FW Version : 158
  BTL Version : 146

niph@DESKTOP-I7KSTQ3 MINGW32 /e/Workdata/packedSDK/Smarttof_sdk_v1.60/SDK-smarttof_sdk_v1.60/windows/tools/SmartTofCli
$ ./dmcam-cli.exe -i
dmcam> info
  Vendor : Data Miracle
  Product : TM-E2-1.0
  DevID   : 0xf558d0170701004d

  Capability:
    max frame width      : 320
    max frame height     : 240
```

dmcam-cli 的其他具体功能介绍请参考详细参考中的 SDK 工具详细说明。



# Chapter 4

## C/C++

### 4.1 采集

SDK 中提供了基本采集、参数设置、滤波使能、保存录像文件等四个基本样例，基本覆盖了在模组二次开发中常用的一些 API 使用，可以作为用户进行开发时的基本参考。

运行 sample\_capture\_frames 采集样例程序如图.

```
f_sdk_v1.60/windows/samples/c/build
$ ./sample_capture_frames.exe
1 dmcam device found
get 10 frames: [0, 320x240, 2]
proc frames ....
proc pcl data ....
proc gray data ....
get 10 frames: [10, 320x240, 2]
proc frames ....
proc pcl data ....
proc gray data ....
get 10 frames: [20, 320x240, 2]
proc frames ....
proc pcl data ....
proc gray data ....
get 10 frames: [30, 320x240, 2]
proc frames ....
proc pcl data ....
proc gray data ....
get 10 frames: [40, 320x240, 2]
proc frames ....
proc pcl data ....
proc gray data ....
get 10 frames: [50, 320x240, 2]
proc frames ....
proc pcl data ....
proc gray data ....
get 10 frames: [60, 320x240, 2]
proc frames ....
proc pcl data ....
proc gray data ....
get 10 frames: [70, 320x240, 2]
proc frames ....
proc pcl data ....
```

样例中为先对模组进行采集前基本设置，然后开启采集，每次获取 10 帧，然后进行深度和灰度等计算，设置的采集总帧数为 100 帧，采集完成后停止采集。采集样例中包括了 dmcam\_cap\_config\_set 用于采集前的配置，dmcam\_cap\_start 开始采集，dmcam\_cap\_get\_frames 获取采集数据,dmcam\_frame\_get\_distance 等主要采集接口。

## 4.2 参数设置

sample\_set\_param 样例主要展示了对模组参数设置的一般方法，主要参数包模设置模组的积分时间、采集帧率、调制频率等，运行的样例程序结果如图。

```
$ ./sample_set_param.exe
Set mod_freq:24000000
Get MOD_FREQ:24000000 ok
Set mod_freq:24000000
Get MOD_FREQ:12000000 ok
Set mod_freq:24000000
Get MOD_FREQ:6000000 ok
Set mod_freq:24000000
Get MOD_FREQ:3000000 ok
Set mod_freq:24000000
Get MOD_FREQ:1500000 ok
Set mod_freq:24000000
Get MOD_FREQ:750000 ok
tl:33.20 C tr:33.40 C
bl:34.80 C br:32.50 C
```

模组参数相关接口包括模组参数的设置和获取，参数设置在调用 dmcam\_dev\_open 接口打开设备后可以进行设置，参数设置通过调用 dmcam\_param\_batch\_set，参数获取通过调用 dmcam\_param\_batch\_get。

## 4.3 濾波

SDK 中包括了对模组进行幅值滤波、深度滤波、自动曝光、运动模式等多种滤波功能，sample\_filer 样例主要展示了使能模组滤波功能的相关使能设置和关闭设置。

运行 sample\_filter 样例的程序如图

```
niph@DESKTOP-I7KSTQ3 MINGW32 /e/Workdata/packedSDK/Smarttof_sdk_v1.60/SDK-smarttof_sdk_v1.60/windows/samples/c
/build
$ ./sample_filter.exe
```

开启滤波功能调用的主要 API 为 dmcam\_filter\_enable，进行滤波功能设置时，有些需要设置参数值，详细参考 Reference 中 C/C++ 核心库中的模组参数和滤波类型说明以及代码示例。

## 4.4 录像

SDK 中提供了支持录像的功能，利用保存的录像文件可以进行前期的算法开发，保存的录像文件格式为 ONI 格式，在使用 dmcam\_cap\_config 进行采集前的设置时，指定保存的录像文件名。

运行 sample\_save\_replay 保存录像样例程序如图

```
f_sdk_v1.60/windows/samples/c/build
$ ./sample_save_replay.exe
-> replay file set to sample_replay.oni
1 dmcam device found
[00] usb://001:001:016
Get frame #0: [320x240, idx=0, fmt=2, sz=614400]
Get frame #1: [320x240, idx=1, fmt=2, sz=614400]
Get frame #2: [320x240, idx=1, fmt=2, sz=614400]
Get frame #3: [320x240, idx=1, fmt=2, sz=614400]
Get frame #4: [320x240, idx=1, fmt=2, sz=614400]
Get frame #5: [320x240, idx=2, fmt=2, sz=614400]
Get frame #6: [320x240, idx=2, fmt=2, sz=614400]
Get frame #7: [320x240, idx=2, fmt=2, sz=614400]
Get frame #8: [320x240, idx=2, fmt=2, sz=614400]
Get frame #9: [320x240, idx=3, fmt=2, sz=614400]
Get frame #10: [320x240, idx=3, fmt=2, sz=614400]
Get frame #11: [320x240, idx=3, fmt=2, sz=614400]
Get frame #12: [320x240, idx=3, fmt=2, sz=614400]
Get frame #13: [320x240, idx=4, fmt=2, sz=614400]
Get frame #14: [320x240, idx=4, fmt=2, sz=614400]
Get frame #15: [320x240, idx=4, fmt=2, sz=614400]
Get frame #16: [320x240, idx=4, fmt=2, sz=614400]
Get frame #17: [320x240, idx=5, fmt=2, sz=614400]
Get frame #18: [320x240, idx=5, fmt=2, sz=614400]
Get frame #19: [320x240, idx=5, fmt=2, sz=614400]
Get frame #20: [320x240, idx=5, fmt=2, sz=614400]
Get frame #21: [320x240, idx=6, fmt=2, sz=614400]
```

运行录像文件样例会生成一个名为 sample\_replay.oni 的文件，可以直接通过 SmartToFViewer 工具进行回放，SmartToFViewer 工具的详细使用参考 Reference 中工具的详细说明。

## 4.5 编译生成

### 4.5.1 在 windows 下生成 vs 工程

1. 在 windows/samples/c 下新建文件夹 vsbuild
2. 使用命令行工具或者 msys2 下 mingw 工具，进入 vsbuild 使用 cmake 生成 vs 工程，具体命令如下：

```
cd vsbuild

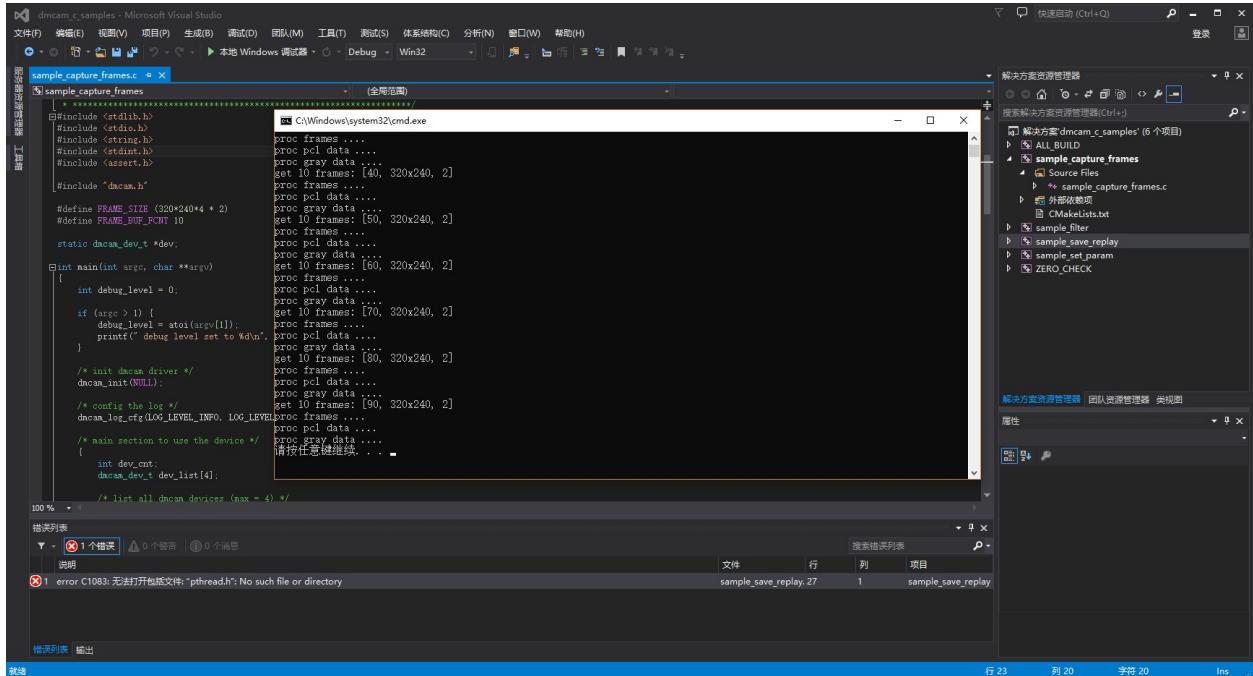
cmake .. -G "Visual Studio 12 2013" //根据自身所装的 vs 版本
```

3. 生成工程后打开 dmcam\_c\_sample.sln, 编译生成 C 样例的可执行文件。

生成的 VS 工程如下图：

名称	修改日期	类型	大小
CMakeFiles	2018/12/21 14:07	文件夹	
ALL_BUILD.vcxproj	2018/12/21 14:07	VC++ Project	45 Ki
ALL_BUILD.vcxproj.filters	2018/12/21 14:07	VC++ Project Fil...	1 Ki
cmake_install.cmake	2018/12/21 14:07	CMAKE 文件	2 Ki
CMakeCache.txt	2018/12/21 14:07	文本文档	15 Ki
<b>dmcam_c_samples.sln</b>	2018/12/21 14:07	Microsoft Visual...	7 Ki
libdmcam.dll	2018/11/23 17:03	应用程序扩展	602 Ki
sample_capture_frames.vcxproj	2018/12/21 14:07	VC++ Project	57 Ki
sample_capture_frames.vcxproj.filters	2018/12/21 14:07	VC++ Project Fil...	1 Ki
sample_filter.vcxproj	2018/12/21 14:07	VC++ Project	56 Ki
sample_filter.vcxproj.filters	2018/12/21 14:07	VC++ Project Fil...	1 Ki
sample_save_replay.vcxproj	2018/12/21 14:07	VC++ Project	57 Ki
sample_save_replay.vcxproj.filters	2018/12/21 14:07	VC++ Project Fil...	1 Ki
sample_set_param.vcxproj	2018/12/21 14:07	VC++ Project	56 Ki
sample_set_param.vcxproj.filters	2018/12/21 14:07	VC++ Project Fil...	1 Ki
ZERO_CHECK.vcxproj	2018/12/21 14:07	VC++ Project	43 Ki
ZERO_CHECK.vcxproj.filters	2018/12/21 14:07	VC++ Project Fil...	1 Ki

成功生成 VS 工程后，使用 VS 打开工程，生成解决方案，下图展示了生成解决方案后，运行 sample\_capture\_frames 程序



其他生成的几个 sample 程序同样可以运行，运行效果跟 windows 生成的可执行文件效果一样，具体可查看下面的附图。

#### 4.5.2 在 windows 下生成可执行文件

1. 在 windowssamplesc 下新建文件夹 build
2. 进入 build 使用 cmake 生成可执行文件，具体命令参考如下：

```
cd build  
  
cmake .. -G "MSYS Makefiles"  
  
make -j
```

3. 编译生成可执行文件后可双击运行

生成可执行文件后，在 build 文件夹下显示如下图所示的可执行文件：

名称	修改日期	类型	大小
CMakEFiles	2018/12/21 14:47	文件夹	
cmake_install.cmake	2018/12/21 14:47	CMAKE 文件	2 KB
CMakeCache.txt	2018/12/21 14:47	文本文档	17 KB
dmcam_20181221.log	2018/12/21 14:48	文本文档	642 KB
dmcam_param_268230777.bin	2018/10/9 15:06	BIN 文件	4,952 KB
libdmcam.dll	2018/11/23 17:03	应用程序扩展	602 KB
Makefile	2018/12/21 14:47	文件	10 KB
sample_capture_frames.exe	2018/12/21 14:47	应用程序	365 KB
sample_filter.exe	2018/12/21 14:47	应用程序	318 KB
sample_save_replay.exe	2018/12/21 14:47	应用程序	320 KB
sample_set_param.exe	2018/12/21 14:47	应用程序	370 KB

C++ 的样例程序的使用跟 C 一样，具体过程可以参考以上 C 的使用过程。

# Chapter 5

## Python

### 5.1 最简采集

SDK 中提供了基于 python 下开发的样例，包括了基本采集、基本参数设置和采集显示，覆盖了 python 下进行模组二次开发中的一些 API 的使用，可以作为用户进行开发时的基本参考。

运行 python 的 sample\_basic.py 采集程序：

```
python sample_basic.py
```

基本采集程序运行的结果如下图

```
f_sdk_v1.60/windows/samples/python
$ python sample_basic.py
Scanning dmcam device ..
found 1 device
DMCAM#0 [001:001:009]: VENDOR=, PROD=, SERIAL=
Open dmcam device ..
Set paramters ...
Start capture ...
sampling 100 frames ...
frame @ 0, 614400, 320x240
frame @ 1, 614400, 320x240
frame @ 2, 614400, 320x240
frame @ 3, 614400, 320x240
frame @ 4, 614400, 320x240
frame @ 5, 614400, 320x240
```

## 5.2 参数设置

sample\_param.py 样例主要展示了对模组参数进行设置的一般方法，包括模组参数的设置和读取。

运行 python 的 sample\_param.py 程序：

```
python sample_param.py
```

参数设置程序运行结果如下图

```
f_sdk_v1.60/windows/samples/python
$ python sample_param.py
  Open dmcam device ..
->test specified parameters writing...
->test specified parameters reading...
  val_mod_freq: 24000000
  val_frm_format: 2

->test batch parameter reading...
dev_mode = 1
mod_freq = 24000000
vendor: Data Miracle
product: TM-E3-1.0
max frame info: 320 x 240, depth=2, fps=30, intg_us=1500
['0xf01e027', '0xaaf5c82', '0x59f689d0']
version: sw:160, hw:30, sw2:146, hw2:0
frame format= 2
fps = 30
illum_power=0 %
intg = 1000 us
  Close dmcam device ..
(venv)
└─> DEEPTOP_T7KTOP_HTCUDA /c/Users/tao/Downloads/SmartAI/Downloads/CDK/cdk_swagger/edk_v1_60/CDK_swagger
```

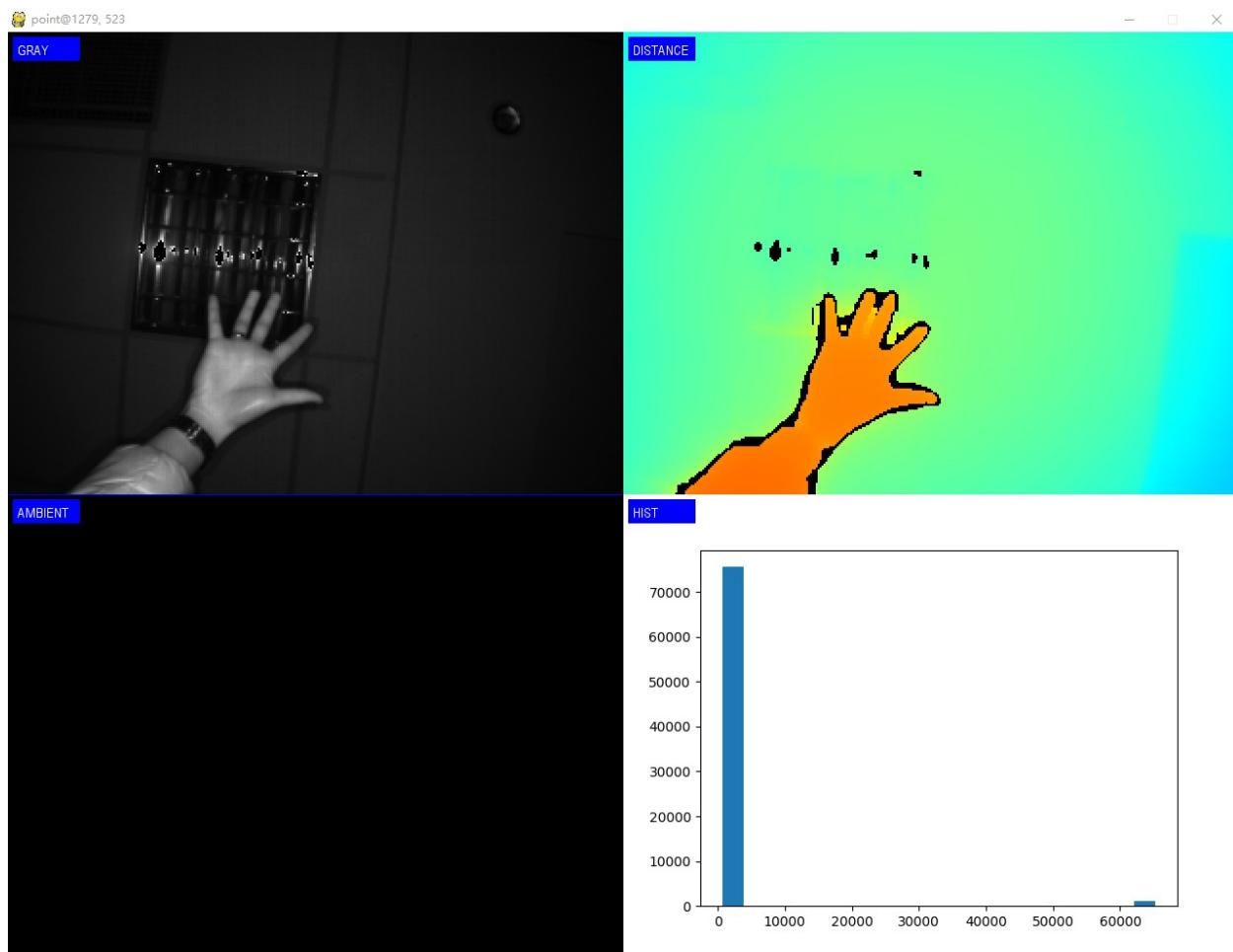
## 5.3 采集 UI 显示

SDK 中提供了在 python 下使用 pygame 等模块进行采集图像的显示，如 sample\_gui\_pygame.py.

运行 python 中的 sample\_gui\_pygame.py 程序:

```
python sample_gui_pygame.py
```

程序运行结果如下图



## 5.4 运行

运行 Python 样例需要安装对应 Python 版本的 dmcam 包、numpy、matplotlib、pygame、Pyqtgraph 等包。

### 5.4.1 运行 python 样例相关包的安装

#### 1. dmcam 库的安装

dmcam 包从 1.60 后开始，已经上传 python 各版本对应的 whl 包到 Pypi 网站，如要更新安装最新的 dmcam 包，安装命令如下：

```
pip install -U dmcam
```

如果要安装指定版本的 dmcam，则需加上版本号，如安装版本号为 1.57.7：

```
pip install dmcam==1.57.7
```

安装 dmcam 包的结果如下图：

```
3.4.0_mu_32bit\venv\Scripts
$ pip install dmcam==1.60.0
Collecting dmcam==1.60.0
  Using cached https://files.pythonhosted.org/packages/f3/74/506edbd69b8cee0babe59b3a0c4f6760
  1cf382f144f0c877e66646145074/dmcam-1.60.0-cp34-cp34m-win32.whl
Requirement already satisfied: numpy>=1.12.0 in g:\workdata\ sdk1214\ sdk_build\dmcam_win_x86\s
wig\python\build_py3.4.0_mu_32bit\venv\lib\site-packages (from dmcam==1.60.0) (1.12.0)
Installing collected packages: dmcam
  Found existing installation: dmcam 1.61.4
    Uninstalling dmcam-1.61.4:
      Successfully uninstalled dmcam-1.61.4
Successfully installed dmcam-1.60.0
(venv)
```

## 2. 样例其他库的安装

安装 numpy、matplotlib、pygame、PyQt5、pyqtgraph, 可以通过以下命令安装:

```
pip install -r requirement.txt
```

---

**Note:** 在 python2.7 或者 python3.4 环境下安装 PyQt5 可能会导致失败，可以换成安装 PyQt4

---

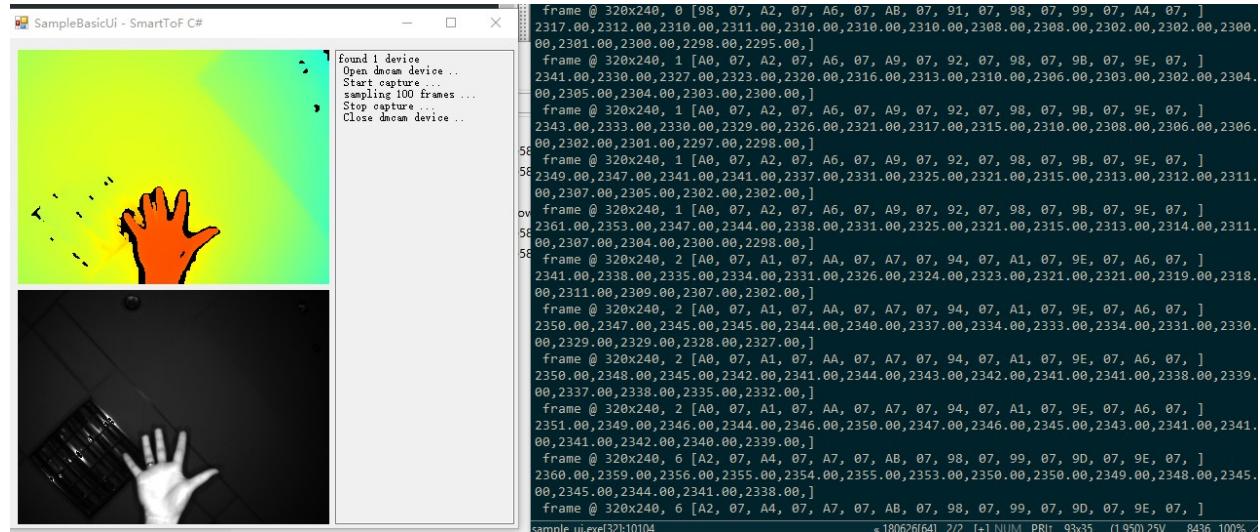
python 相关包安装好后，就可以运行指定的样例。





## 6.2 采集 UI 显示

sampleBasicUi 采集显示样例展示了在调用 SDK 中的采集接口进行采集的同时将采集图像显示出来，运行结果和 gui 界面如下图



## 6.3 编译生成

### 6.3.1 windows 平台

- 命令行进入到 windowssamplescsharp 目录运行下面的命令,<BUILD\_TYPE> 可以是 Release 或者 Debug:

```
mkdir build
cd build
cmake .. -G "Visual Studio 12 2013"           //根据自身所装的 vs 版本
cd ..
cmake --build ./build --config <BUILD_TYPE>
```

在 build 文件下生成了 sample\_basic.exe 和 sample\_ui.exe 两个可执行文件，如下图

名称	修改日期	类型	大小
cmake_install.cmake	2018/12/21 16:41	CMAKE 文件	
CMakeCache.txt	2018/12/21 16:41	文本文档	
dmcam_20181221.log	2018/12/21 16:41	文本文档	
dmcam_cs_samples.sln	2018/12/21 16:41	Microsoft Visual...	
dmcam_csharp.dll	2018/12/21 16:41	应用程序扩展	1
dmcam_csharp_adapter.dll	2018/12/21 16:41	应用程序扩展	3
dmcam_param_268230777.bin	2018/10/9 15:06	BIN 文件	4,9
libdmcam.dll	2018/12/21 16:41	应用程序扩展	6
sample_basic.exe	2018/12/21 16:41	应用程序	
sample_basic.vcxproj	2018/12/21 16:41	VC++ Project	
sample_basic.vcxproj.filters	2018/12/21 16:41	VC++ Project Fil...	
sample_ui.exe	2018/12/21 16:41	应用程序	
sample_ui.vcxproj	2018/12/21 16:41	VC++ Project	
sample_ui.vcxproj.filters	2018/12/21 16:41	VC++ Project Fil...	

### 6.3.2 Linux 平台

- 在 linux 下需要安装 mono 用于 C# 的编译扩展:

```
sudo apt-get install mono-complete
```

- 进入到 linux/sample/chsharp 目录下, 运行下面命令:

```
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Debug
cd ..
cmake --build build
```

在 build 文件夹下生成了 sample\_basic.exe 和 sample\_ui.exe 两个可执行文件, 如下图

```
osboxes@osboxes:/mnt/hgfs/packedSDK/Smarttof_sdk_v1.60/SDK-smarttof_sdk_v1.60/linux$ ls
Ubuntu Software Centre      lib$ ls
CMakeCache.txt                dmcam_csharp.dll          Makefile
CMakeFiles                   libdmcam_csharp_adapter.so  sample_basic.exe
cmake_install.cmake           libdmcam.so              sample ui.exe
```



# Chapter 7

## Java

### 7.1 最简采集

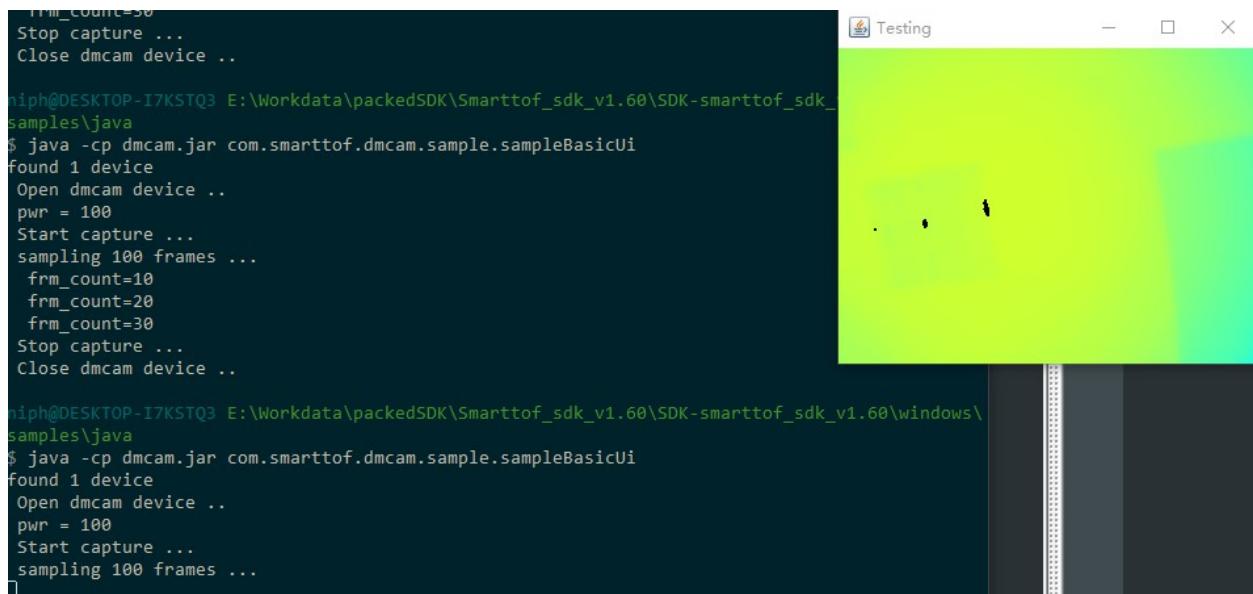
SDK 中提供了 Java 的 basic 和 basicUi 两个基本展示样例，展示了在 Java 下如果调用 SmartToF 的 api 进行模组的采集使用。

运行 basic 的程序运行结果如下图

```
niph@DESKTOP-I7KSTQ3 E:\Workdata\packedSDK\Smarttof_sdk_v1.60\SDK-smarttof_sdk_v1.60\windows\samples\java
$ java -cp dmcam.jar com.smarttof.dmcam.sample.sampleBasic
found 1 device
Open dmcam device ..
pwr = 100
Start capture ...
sampling 100 frames ...
frame @ 320x240, 0 [0xA2, 0x07, 0xA4, 0x07, 0xAB, 0x07, 0xAB, 0x07, 0x98, 0x07, 0xA2, 0x07,
0xA7, 0x07, 0xA6, 0x07, ]
2.381, 2.381, 2.380, 2.379, 2.376, 2.376, 2.376, 2.377, 2.375, 2.369, 2.362, 2.360, 2.355, 2.
348, 2.343, 2.337, ]
frame @ 320x240, 1 [0x94, 0x07, 0xA1, 0x07, 0xAC, 0x07, 0xA9, 0x07, 0x88, 0x07, 0x91, 0x07,
0x9B, 0x07, 0x9B, 0x07, ]
2.384, 2.387, 2.387, 2.387, 2.386, 2.382, 2.383, 2.382, 2.379, 2.372, 2.365, 2.361, 2.355, 2.
351, 2.347, 2.342, ]
frame @ 320x240, 1 [0x88, 0x07, 0x8A, 0x07, 0x96, 0x07, 0xB6, 0x07, 0xA2, 0x07, 0xAA, 0x07,
0xCC, 0x07, 0xD4, 0x07, ]
2.401, 2.398, 2.396, 2.395, 2.396, 2.391, 2.392, 2.390, 2.387, 2.378, 2.373, 2.367, 2.359, 2.
354, 2.349, 2.343, ]
frame @ 320x240, 1 [0x68, 0x07, 0xD2, 0x07, 0xDA, 0x07, 0xCB, 0x07, 0xCA, 0x07, 0xE4, 0x07,
0xEF, 0x07, 0xF5, 0x07, ]
2.391, 2.390, 2.390, 2.391, 2.391, 2.391, 2.390, 2.391, 2.391, 2.390, 2.376, 2.368, 2.363, 2.
359, 2.353, 2.347, ]
frame @ 320x240, 1 [0xD2, 0x07, 0x84, 0x07, 0x7B, 0x07, 0x94, 0x07, 0x64, 0x07, 0xA4, 0x07,
0x9A, 0x07, 0xC7, 0x07, ]
2.387, 2.386, 2.385, 2.388, 2.388, 2.385, 2.385, 2.388, 2.389, 2.386, 2.377, 2.373, 2.366, 2.
361, 2.356, 2.350, ]
frame @ 320x240, 2 [0x98, 0x07, 0xA9, 0x07, 0xBD, 0x07, 0xC6, 0x07, 0xD0, 0x07, 0xE4, 0x07,
0xE6, 0x07, 0x9C, 0x07, ]
2.385, 2.382, 2.379, 2.380, 2.380, 2.379, 2.379, 2.377, 2.375, 2.373, 2.370, 2.366, 2.362, 2.
359, 2.354, 2.348, ]
```

## 7.2 采集 UI 显示

Java basicUi 采集显示样例展示了在使用 Java 时调用 SDK 中的采集接口进行采集的同时将图像显示出来，样例运行的结果如下图



## 7.3 编译生成

### 7.3.1 windows 平台

1. 将 SDK 中 windows/java 目录下对应位数的 java 包和 windowslib 下对应得 libdmcam.dll 拷贝到 windows/samples/java 目录
2. 运行 javac 命令进行编译:

```

javac -cp dmcam.jar .\com\smarttof\dmcam\sample\sampleBasic.java
javac -cp dmcam.jar .\com\smarttof\dmcam\sample\sampleBasicUi.java

```

编译成功后，在 windows/samples/java/com/smarttof/dmcam/sample 下有相应的 class 文件生成

在 windows/samples/java 目录中 java 命令运行:

```
java -cp dmcam.jar com.smarttof.dmcam.sample.sampleBasic
```

在 windows/samples/java 目录中 java 命令运行:

```
java -cp dmcam.jar com.smarttof.dmcam.sample.sampleBasicUi
```

### 7.3.2 linux 平台

1. 在 linux 下安装 openjdk-7-jdk 用于 java 扩展:

```
sudo apt-get install openjdk-7-jdk
```

2. 在 SDK 中的 linux/java 目录下的对应位数的 java 包拷贝到 linux/samples/java 目录,
3. 运行 javac 命令进行编译:

```
javac -cp .:dmcam.jar ./com/smarttof/dmcam/sample/sampleBasic.java  
javac -cp .:dmcam.jar ./com/smarttof/dmcam/sample/sampleBasicUi.java
```

编译成功后，在 linux/samples/java/com/smarttof/dmcam/sample 下有相应的 class 文件生成  
在 linux/samples/java 目录运行，如果有些动态库没有找到，需要指定 LD\_LIBRARY\_PATH:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$(pwd) java -cp .:dmcam.jar com.smarttof.dmcam.sample.  
↳sampleBasic
```

在 linux/samples/java 目录中 java 目录运行，如果有些动态库没有找到，需要指定 LD\_LIBRARY\_PATH:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$(pwd) java -cp .:dmcam.jar com.smarttof.dmcam.sample.  
↳sampleBasicUi
```

# Chapter 8

## ROS

ROS 包中主要包括了 ros 系统对 smartttof API 的封装，分为 dmcam\_ros 和 cloud\_viewer 两个包，dmcam\_ros 包用来采集与显示深度、灰度数据和动态修改参数，cloud\_viewer 用来显示点云数据。

### 8.1 ROS 深度显示

1. 开启 ROS 环境:

```
roscore&
```

2. 进入 ros 所在文件夹初始化环境变量:

```
source ./devel/setup.bash
```

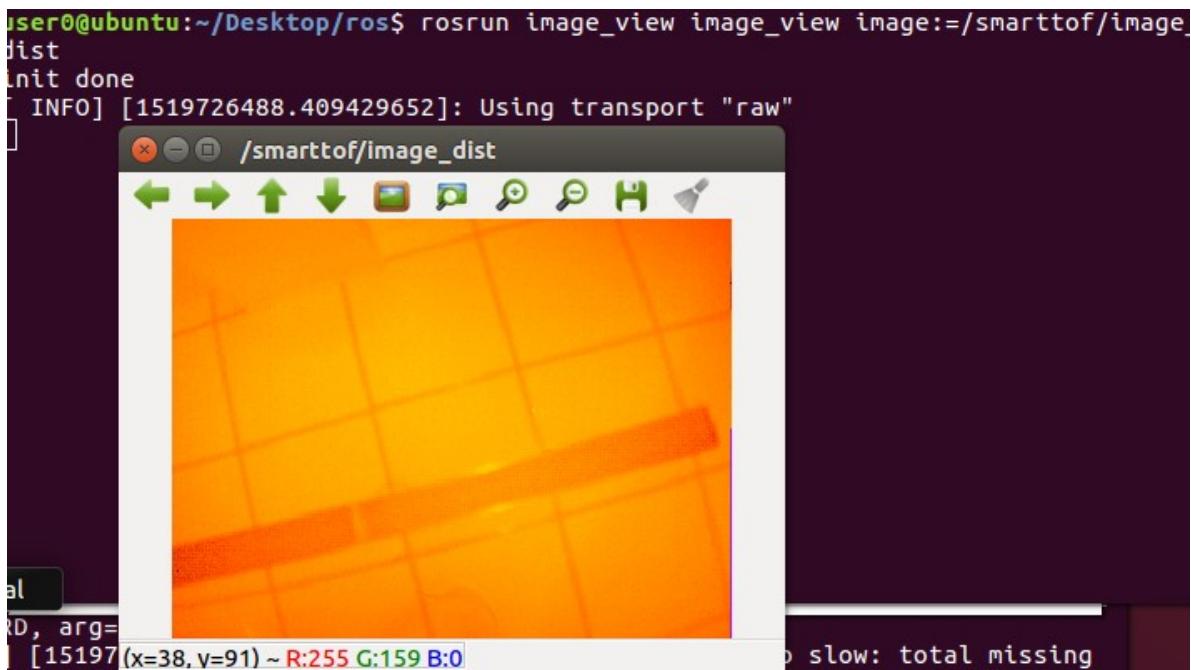
3. 运行 launch 文件:

```
roslaunch dmcam_ros start.launch
```

4. 显示深度图命令:

```
rosrun image_view image_view image:=/smartttof/image_dist
```

深度图像显示如下：



## 8.2 ROS 灰度显示

1. 开启 ROS 环境:

```
roscore&
```

2. 进入 ros 所在文件夹初始化环境变量:

```
source ./devel/setup.bash
```

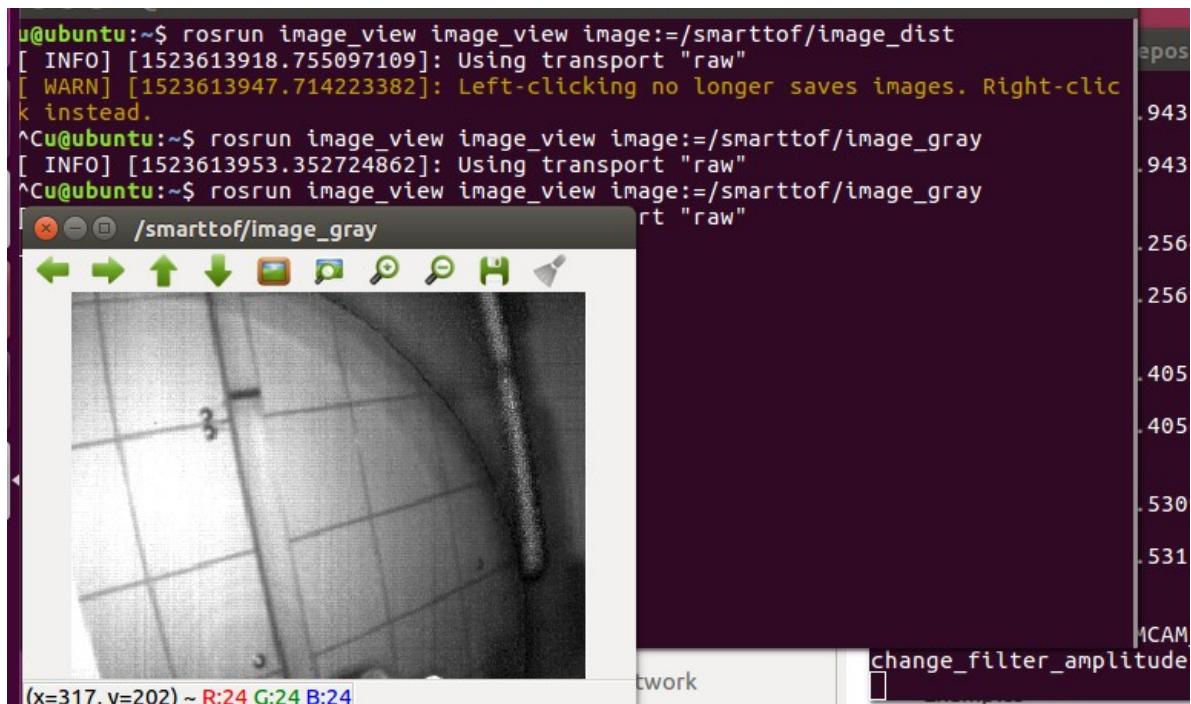
3. 运行 launch 文件:

```
roslaunch dmcam_ros start.launch
```

4. 显示灰度图命令:

```
rosrun image_view image_view image:=/smarttof/image_gray
```

灰度图像显示如下:



## 8.3 ROS 点云显示

cloud\_viewer 是一个简单的使用 smarttof ros 来显示点云数据的样例，这个样例简单的实现了怎么从 smarttof ros 发布的话题 pointcloud 中获取点云数据并显示出来。

1. 开启 ROS 环境:

```
roscore&
```

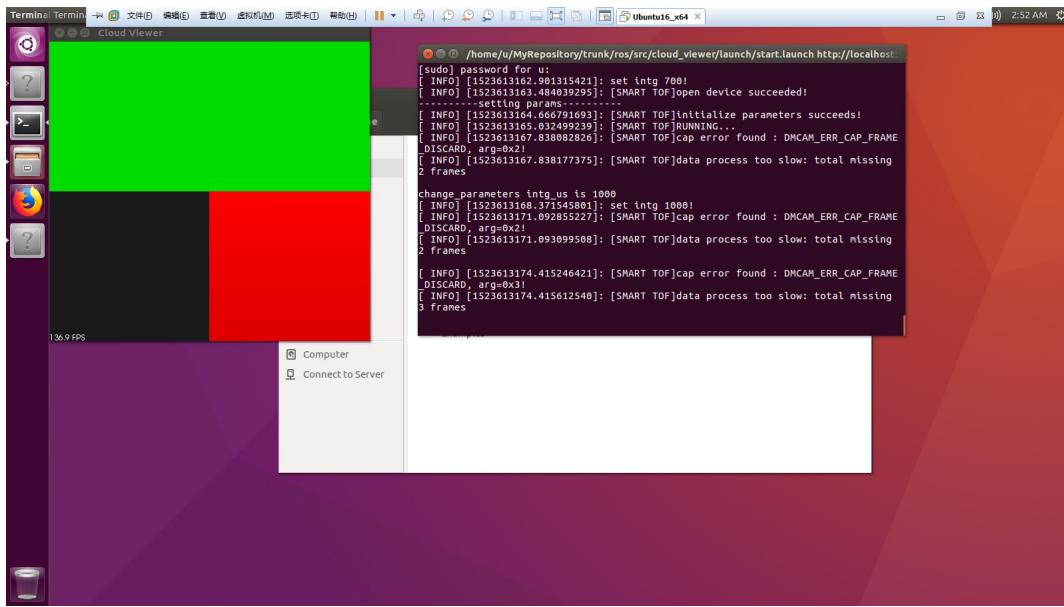
2. 进入 ros 所在文件夹初始化环境变量:

```
source ./devel/setup.sh
```

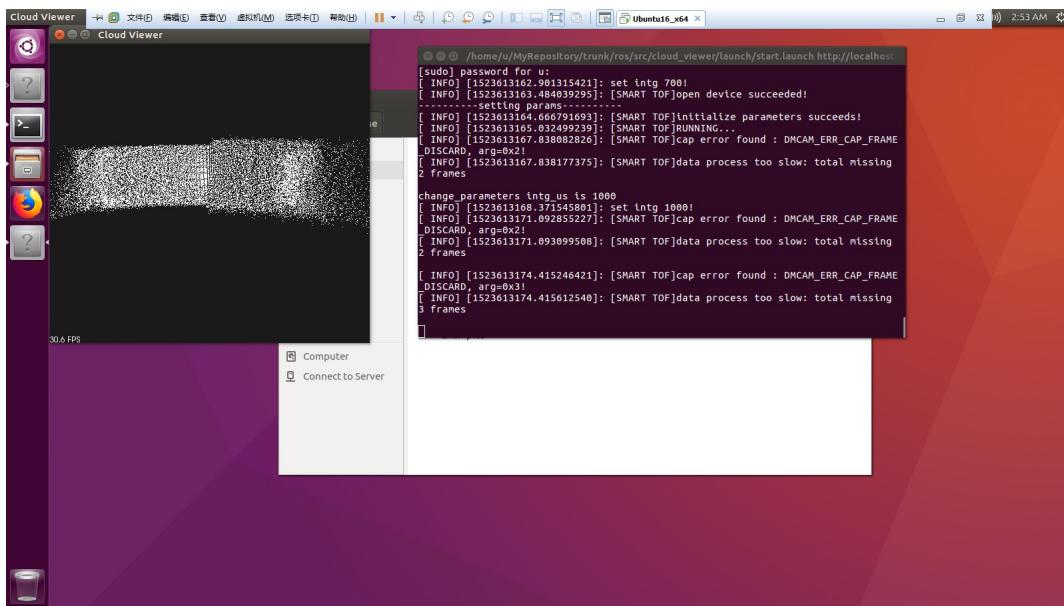
3. 运行 launch 文件:

```
roslaunch cloud_viewer start.launch
```

4. 显示点云图像



5. 通过鼠标中间的滑轮和鼠标左键调整点云显示图像，最终效果如图



## 8.4 ROS 下使用模组环境搭建

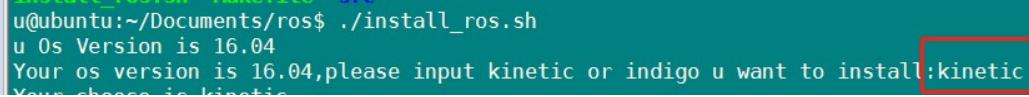
### 8.4.1 Ubuntu 下安装 ros

ubuntu 下安装 ros 过程如下：

1. 打开命令行终端，进入 SDK 中的 ros 文件夹中，运行如下命令：

```
sudo chmod 755 install_ros.sh
./install_ros.sh
```

2. 出现如下图所示的安装选择版本, 手动输入版本名称后按回车开始安装, Ubuntu14.04 推荐使用 indigo, Ubuntu16.04 推荐使用 kinetic。



```
u@ubuntu:~/Documents/ros$ ./install_ros.sh
u Os Version is 16.04
Your os version is 16.04,please input kinetic or indigo u want to install:kinetic
Your choice is kinetic
```

## 8.4.2 ROS 环境配置

每次使用 ROS 系统前需要初始化安装版本的环境变量, 以 Kinetic 为例, Kinetic 默认安装在/opt/ros/kinetic/目录下, 该环境变量配置文件位置/opt/ros/kinetic/setup.bash, 每次使用前需要初始化 ros 环境, 命令如下:

```
source /opt/ros/kinetic/setup.bash
```

为了简化配置环境变量的过程, 可以选择把环境变量的配置放在 ~/.bashrc 文件中, 这样每次打开一个新终端的时候, ROS 的环境变量会自动配置好:

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
```

## 8.4.3 SmartToF ros 包编译

1. 进入 ros 所在目录, 通过 ls 命令查看目录中文件如下:

```
install_ros.sh  Makefile  src
```

2. 使用 catkin\_make(Makefile 中实现了 catkin\_make 使用的步骤, 也可以使用 make 命令来编译):

```
source /opt/ros/kinetic/setup.bash(未在 bashrc 中设置初始化环境需要这一步)
catkin_make
```

3. 编译完成后会生成 devel 和 build 目录, 通过 ls 命令查看编译生成的文件:

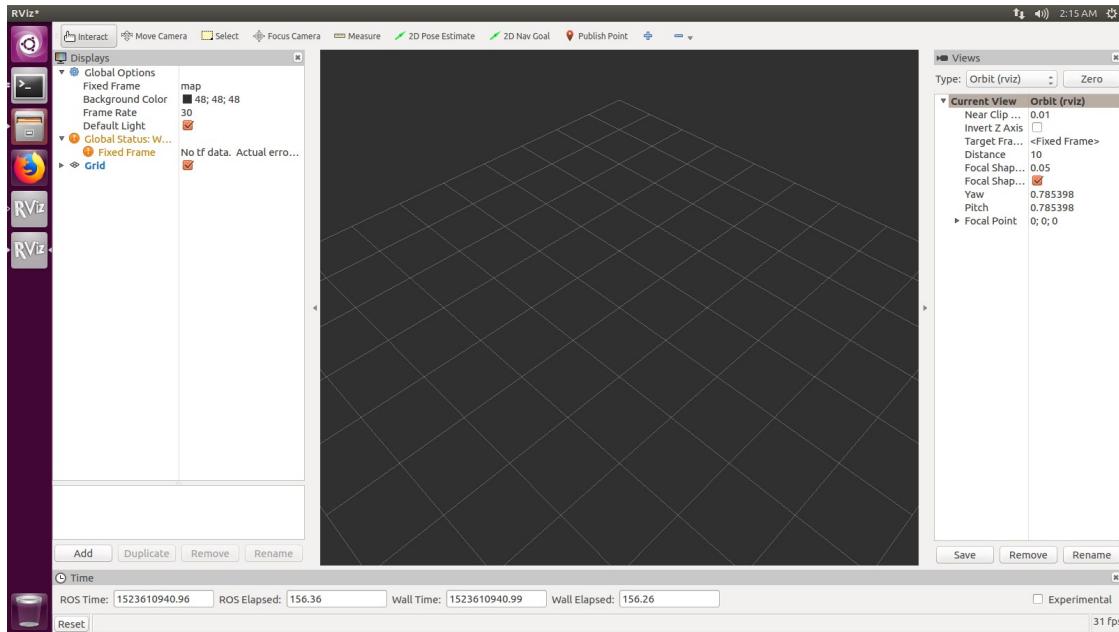
```
build  devel  install_ros.sh  Makefile  src
```

4. 初始化 devel 中的环境变量:

```
source devel/setup.bash
```

## 8.5 ROS rviz 工具使用

rviz 是 ros 自带的一个图形化工具，可以方便的对 ros 的程序进行图形化操作，整体界面如下图所示：



界面主要分为左侧的显示设置区域，中间的大的显示区域和右侧的视角设置区域。最上面是和导航相关的几个工具。最下面是 ros 状态相关的一些数据的显示。

### 8.5.1 rviz 使用前准备

1. 开启 ROS 环境:

```
Roscore&
```

2. 进入 ros 所在文件夹初始化环境变量:

```
source ./devel/setup.bash
```

3. 运行 launch 文件:

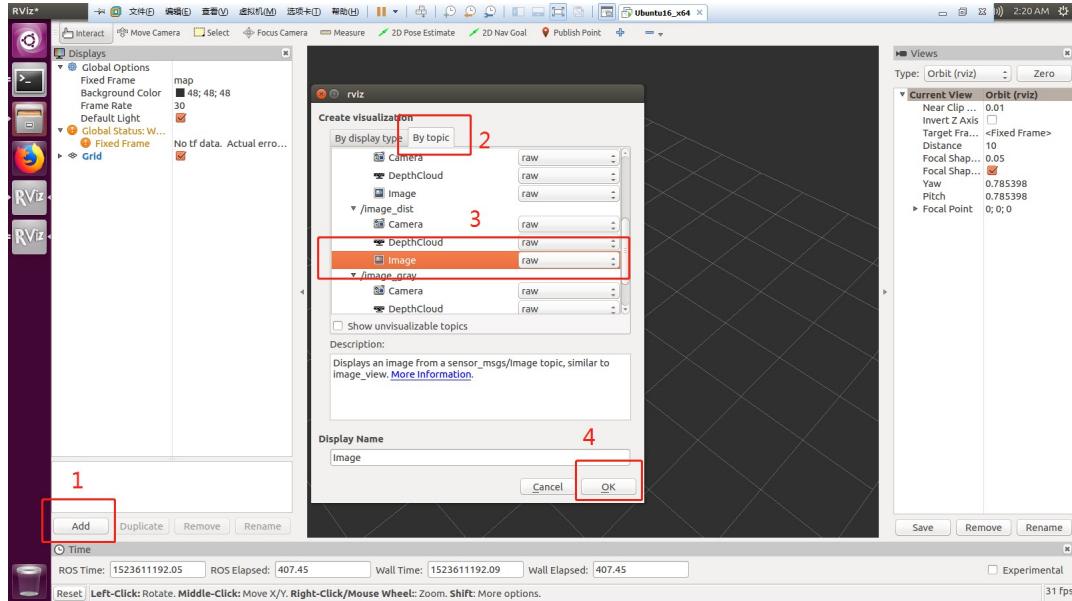
```
roslaunch dmcam_ros start.launch
```

### 8.5.2 rviz 显示深度图像

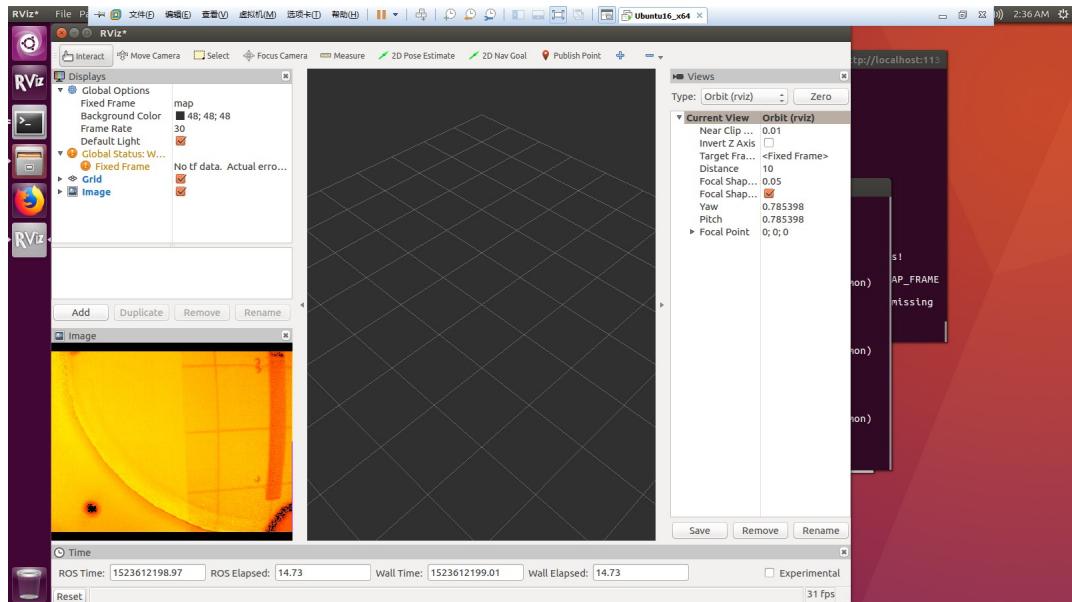
1. 打开一个终端，运行 rviz:

```
rviz
```

2. 选中 add, By topic 中选中 image\_dist 下的 Image, 最后确认添加, 如下图所示:



3. 显示效果如下图所示:

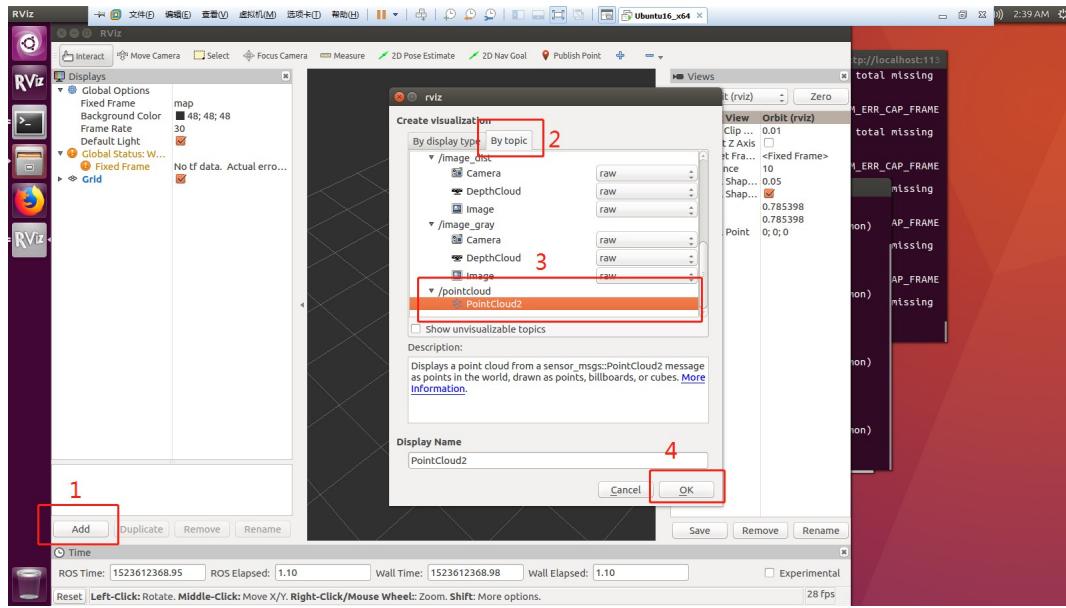


### 8.5.3 rviz 显示点云图像

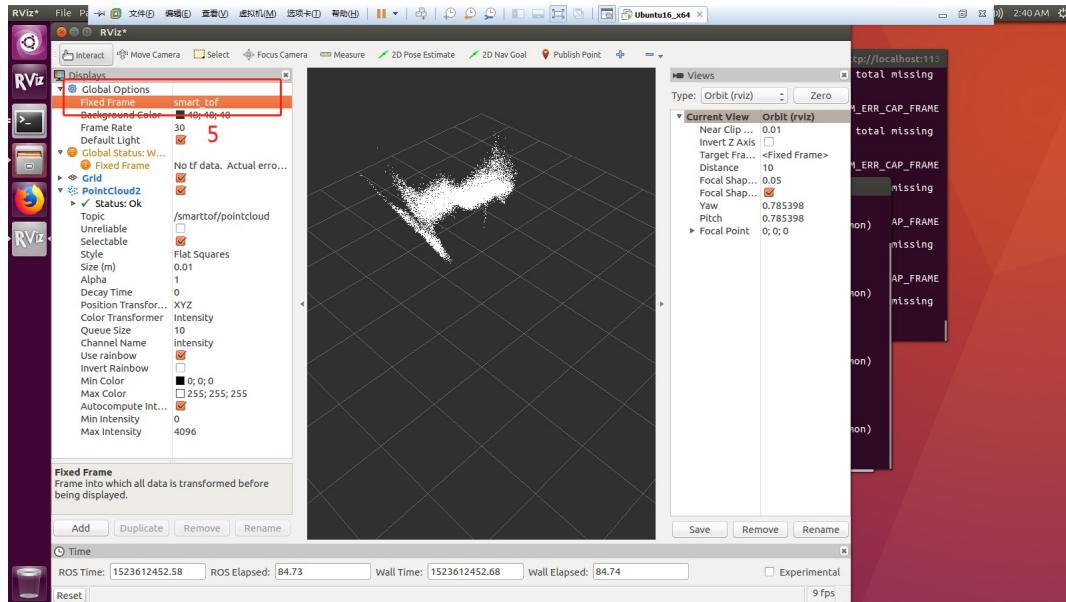
1. 打开一个终端, 运行 rviz:

```
rviz
```

2. 选中 add, byTopic 中选中 pointcloud 下的 PointCloud2, 最后确认添加.



3. 在 rviz 左上角的 displays 区域, 修改 GlobalOptions 下的变量 FixedFrame 值为 dmcam\_ros, 点云显示如下图



## 8.6 ROS 滤波

### 8.6.1 滤波功能的使用

1. 打开一个滤波功能, 如 DMCAM\_FILTER\_ID\_AUTO\_INTG:

```
rosservice call /smarttof/change_filter "filter_id:  
'DMCAM_FILTER_ID_AUTO_INTG'  
filter_value: 0"
```

2. 关闭一个滤波功能，如 DMCAM\_FILTER\_ID\_AUTO\_INTG:

```
rosservice call /smarttof/disable_filter "filter_id:  
'DMCAM_FILTER_ID_AUTO_INTG'"
```

详细的 ROS API 介绍请前往 ROS 扩展



# Chapter 9

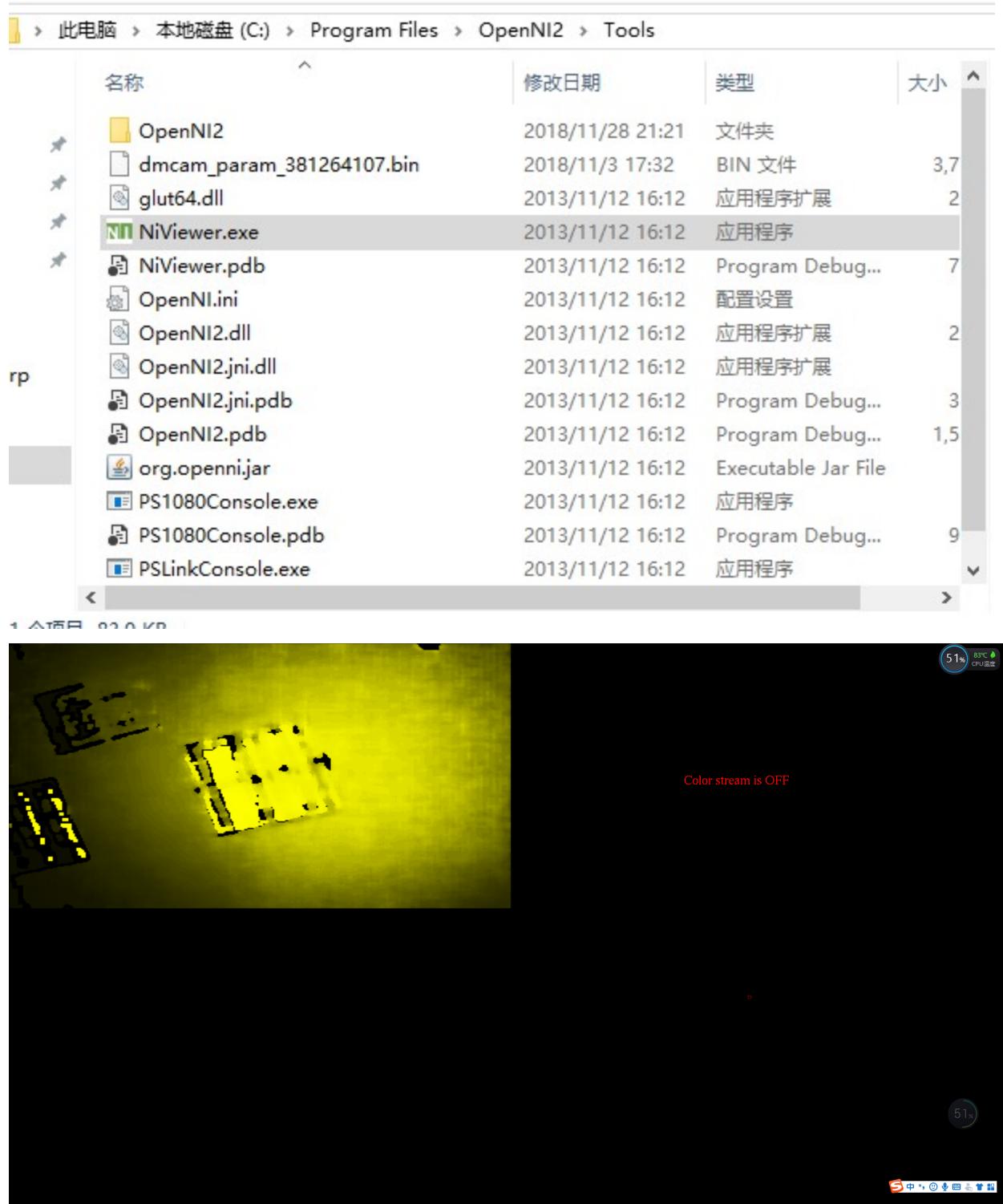
## Openni2

### 9.1 Niviewer 采集显示

SmartToF SDK 目前提供了支持 OpenNI2 的 Windows 平台相关驱动库，通过将 SDK 中 openni2 包中的 libdmcam.dll、smartttof.dll、smartttof.lib、smartttof.pdb 等库拷贝到 OpenNI2 安装路径下的 Tools/OpenNI2/Drivers 下，如下图

此电脑 > 本地磁盘 (C:) > Program Files > OpenNI2 > Tools > OpenNI2 > Drivers			
名称	修改日期	类型	大小
Kinect.pdb	2013/11/12 16:12	Program Debug...	8
libdmcam.dll	2018/11/23 17:03	应用程序扩展	5
OniFile.dll	2013/11/12 16:12	应用程序扩展	2
OniFile.pdb	2013/11/12 16:12	Program Debug...	1,1
PS1080.dll	2013/11/12 16:12	应用程序扩展	5
PS1080.ini	2013/11/12 16:12	配置设置	
PS1080.pdb	2013/11/12 16:12	Program Debug...	2,5
PSLink.dll	2013/11/12 16:12	应用程序扩展	2
PSLink.ini	2013/11/12 16:12	配置设置	
PSLink.pdb	2013/11/12 16:12	Program Debug...	1,5
smartttof.dll	2018/11/23 17:03	应用程序扩展	1
smartttof.exp	2018/11/23 17:03	Exports Library ...	
smartttof.lib	2018/11/23 17:03	Object File Library	
smartttof.pdb	2018/11/23 17:03	Program Debug...	7

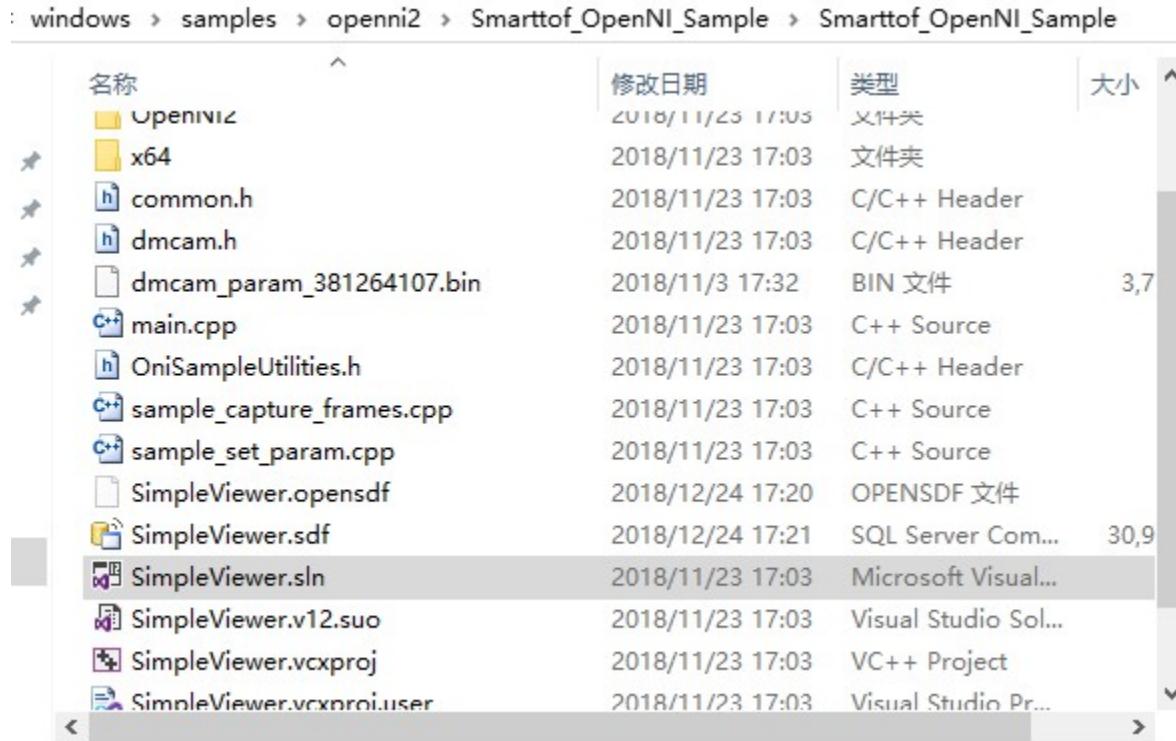
转到 Tools 目录，管理员运行 Tools 下 NiViewer.exe 程序，显示结果如下图



## 9.2 最简采集

SmartToF SDK 还提供了使用 OpenNI 的基础的采集和参数设置样例：

进入 SDK 中的 openni2 目录下的 Smarttof\_OpenNI\_Sample/Smarttof\_OpenNI\_Sample, 打开 VS 工程如下图



编译生成 main.cpp, 生成后运行结果如下图

```

main.cpp  E:\Workdata\packedSDK\Smarttof_Ldk_v1.60\SDK-smarttof_sdळ_v1.60\windows\samples\openni2\Smarttof_OpenNI_Sample\Smarttof_OpenNI_Sample\SimpleViewer\main.cpp
文件(F) 编辑(E) 视窗(V) 项目(P) 生成(G) 调试(D) 团队(M) 工具(T) 测试(S) 体系结构(C) 分析(N) 窗口(W) 帮助(H)
本地 Windows 调试器 -> x64 -> 快速启动 (Ctrl+Q) 登录
解决方案资源管理器 搜索解决方案资源管理器 (Ctrl+F) 
解压缩方案 SimpleViewer(1 个项目)
解压缩方案 SimpleViewer
  <> SimpleViewer
    <> 部分依赖项
    <> common.h
    <> main.cpp
      <> sample_set_param.cpp
      <> Viewer.cpp
    <> Viewer.h
解决方... 资源管理器 团队资源管理器 布局图 属性
输出 显示输出来源(S): 生成 行 1 列 1 字符 1
编辑列表 编辑
就选

```

```

#include "stdafx.h"
#include "dmcam.h"
#include "OniSampleUtilities.h"

using namespace smarttof;

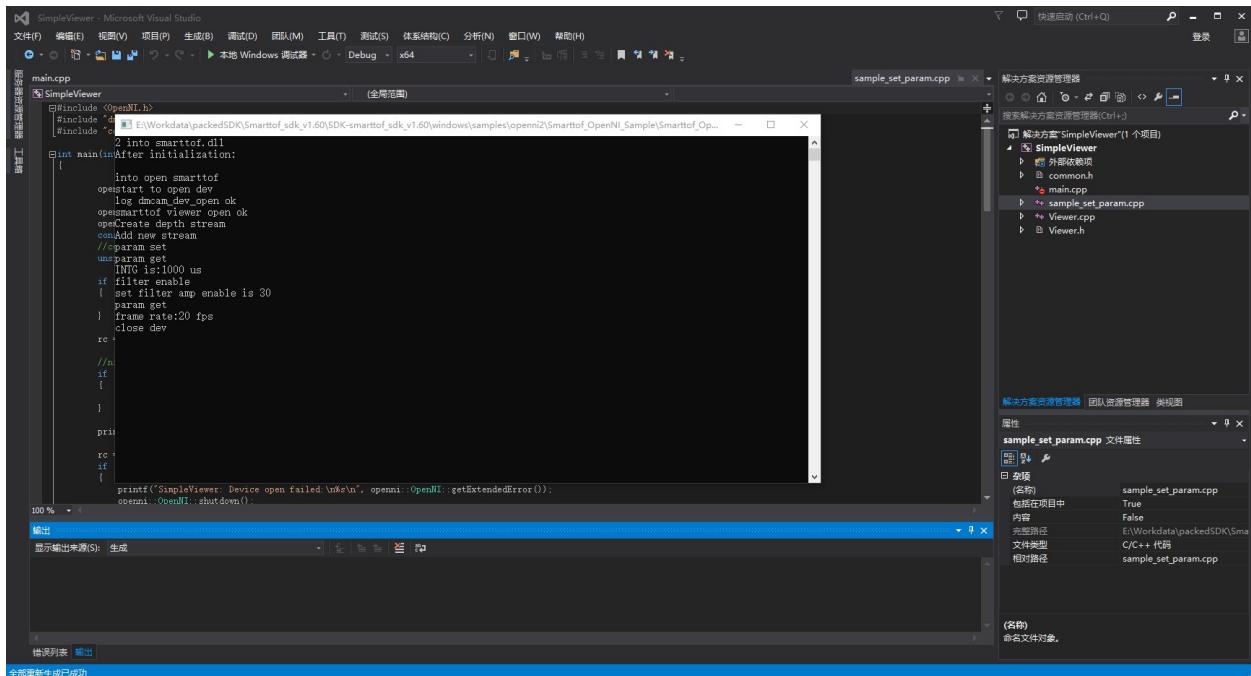
int main()
{
    // init smarttof
    if (!startToOpenDev())
    {
        return -1;
    }

    log_dmcam_dev_open_ok();
    CreateDepthStream();
    AddNewStream("depth");
    StartCapture();
    StartProcessDepthData();
    fsize:153600
    153600 , 320 240, *****
    Read 1 frame ok
    fsize:153600
    153600 , 320 240, *****
    Read 3 frame ok
    fsize:153600
    153600 , 320 240, *****
    Read 5 frame ok
    fsize:153600
    153600 , 320 240, *****
    Read 7 frame ok
    fsize:153600
    153600 , 320 240, *****
    Read 9 frame ok
    fsize:153600
    153600 , 320 240, *****
    Read 11 frame ok
    fsize:153600
    if (rc != openni::STATUS_OK)
    {
        printf("SimpleViewer: Device open failed\n%s\n", openni::OpenNI::getExtendedError());
        openni::OpenNI::shutdown();
        return 1;
    }
}

```

## 9.3 参数设置

将 Openni2 采集样例中改为 sample\_set\_param.cpp 加入编译，编译生成后运行结果如下图



# Chapter 10

## Android

### 10.1 APP 采集显示

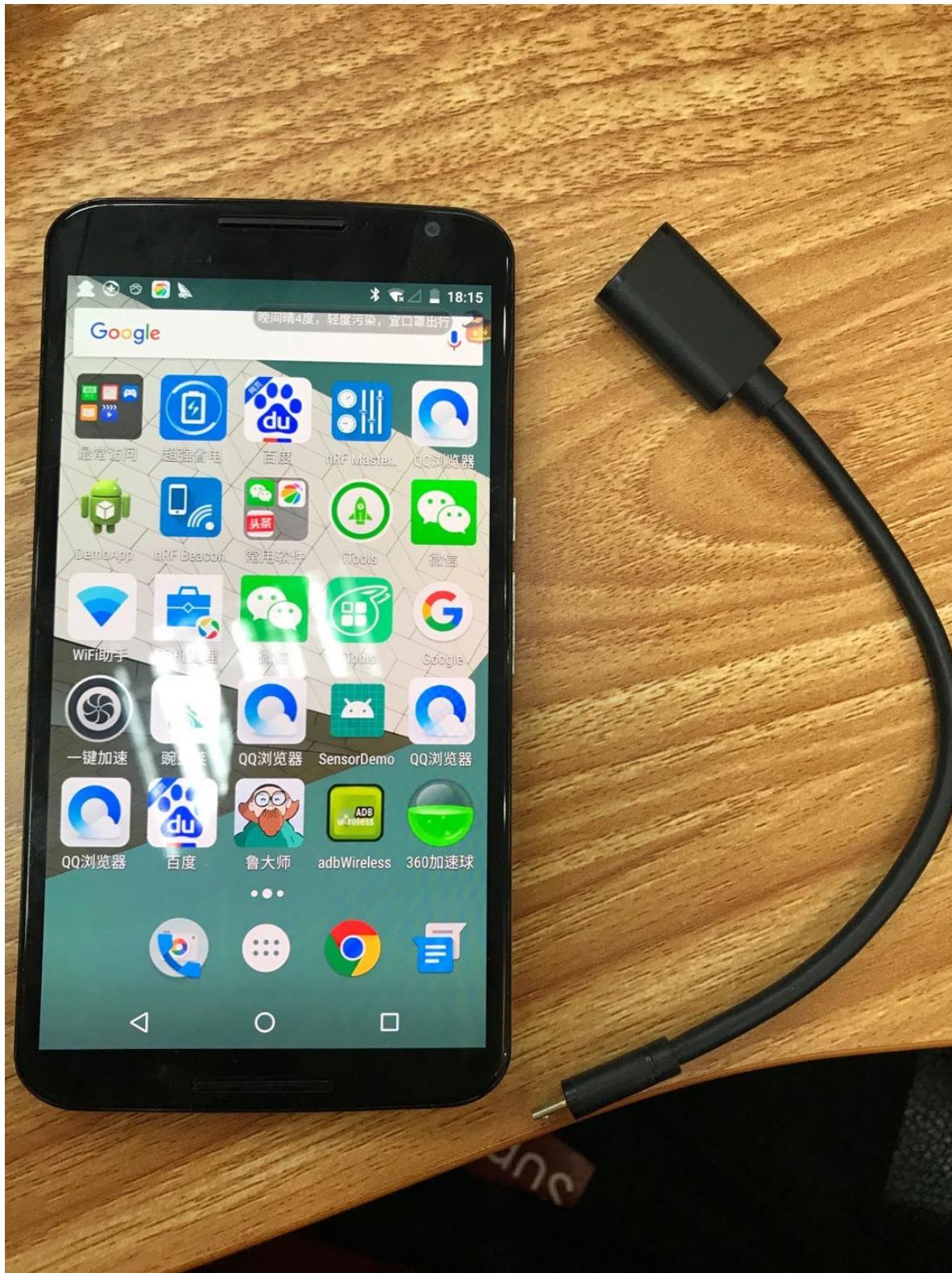
SDK 中的 Android/tools 目录下提供了 Android 的 apk 安装包和生成 apk 的 Eclipse 工程，使用 Android 手机安装好 APK 后，连接模组即可采集显示。

#### 1. 软件安装

安装 apk 时手机的系统要求 Android 4.2.3 以上，并且安装时选择允许 USB 权限。

#### 2. 硬件连接

手机连接模组时，除了需要手机和模组外，通常还需要准备一根手机的 OTG 连接线，OTG 线一端连接手机，并将模组原来连接 PC 的那端连接 OTG 线。手机和 OTG 如下图

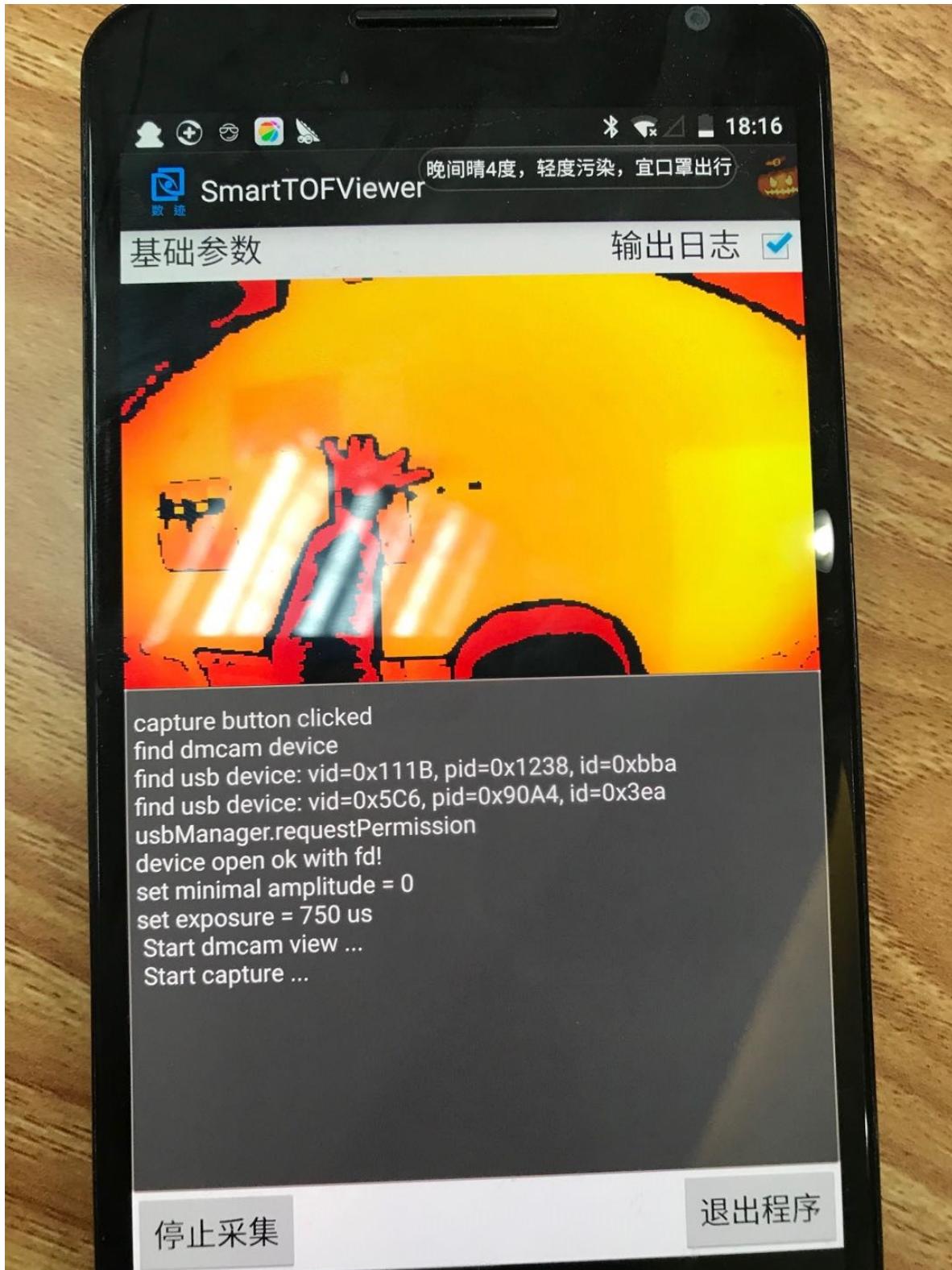


### 3. 运行样例

打开手机上的 SmartToFViewer 软件，界面如下图



点击开始采集按钮即可进行采集演示，采集显示如下图



# Chapter 11

## 工具详细说明

### 11.1 SmartToFViewer 使用说明

#### 11.1.1 简介

SmartToFViewer 是一款可视化工具，可以用来快速评估模组效果，熟悉不同参数对显示效果影响，并确定最佳参数辅助实际开发使用。

主要支持功能如下：

- 设备选择、打开、关闭等
- 显示图像深度图、灰度图等
- 配合 PCLViewer 显示点云图
- 查看物体和模组摄像头间距离
- 查看模组信息及工作状态
- 设置常用参数
- 设置滤波特性
- 设置运动模式
- 录像及回放录像

#### 11.1.2 界面介绍

SmartToFViewer 打开后的整体预览如下，SmartToFViewer 界面主要包括图像显示区、基础参数区、录像设置区、filter 设置区、信息区以及模组的开启关闭：



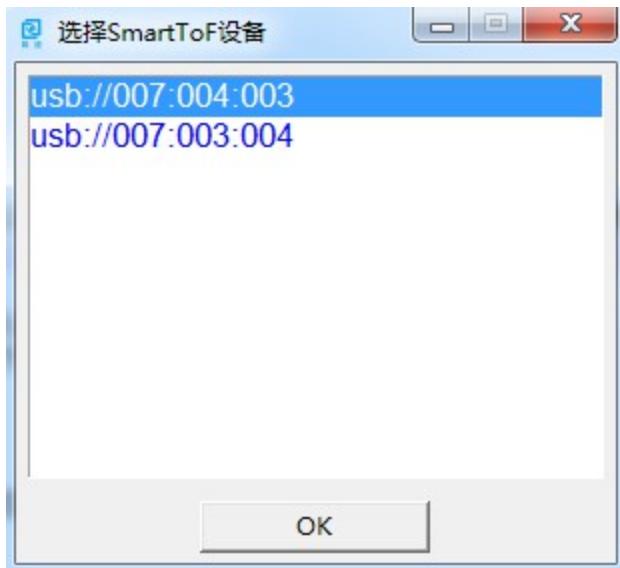
SmartToFViewer 功能区说明如下图：

名称	功能描述
“图像显示区”	显示图像，如深度图、灰度图等；
“开始”按钮	启动采集(首次启动时候，会从模组读取校准相关数据)
“结束”按钮	停止采集
“选择设备”	有多个设备时候，列出可用的设备，点击选择对应的设备
“打开回放”	选择回放文件
“设备状态提示”	打开SmartToFViewer.exe时候，状态由“初始化中...”变为“设备就绪”表示成功打开；点击“开始”按钮，状态由“设备就绪”变为“开始采集...”；失败状态为“无设备”和“打开设备”失败，遇到失败状态，则需要重启SmartToFViewer。 [TCM-E2-1.0:模组型号，HW:30:硬件版本，SW:161/146:固件版本号，设备URI]
“距离信息”	格式为：[像素点坐标]:像素点距离(m)（平均距离，标准差@偏差，幅值）；距离信息会随鼠标在“图像显示区”移动而更新。出现[N/A]表示距离无效。
“Filter设置区”	用来开启或者关闭滤波功能，如自动曝光、HDR模式、运动模式等
“基础参数区”	可用来调整图像显示效果，详见后续“基础参数区说明”章节

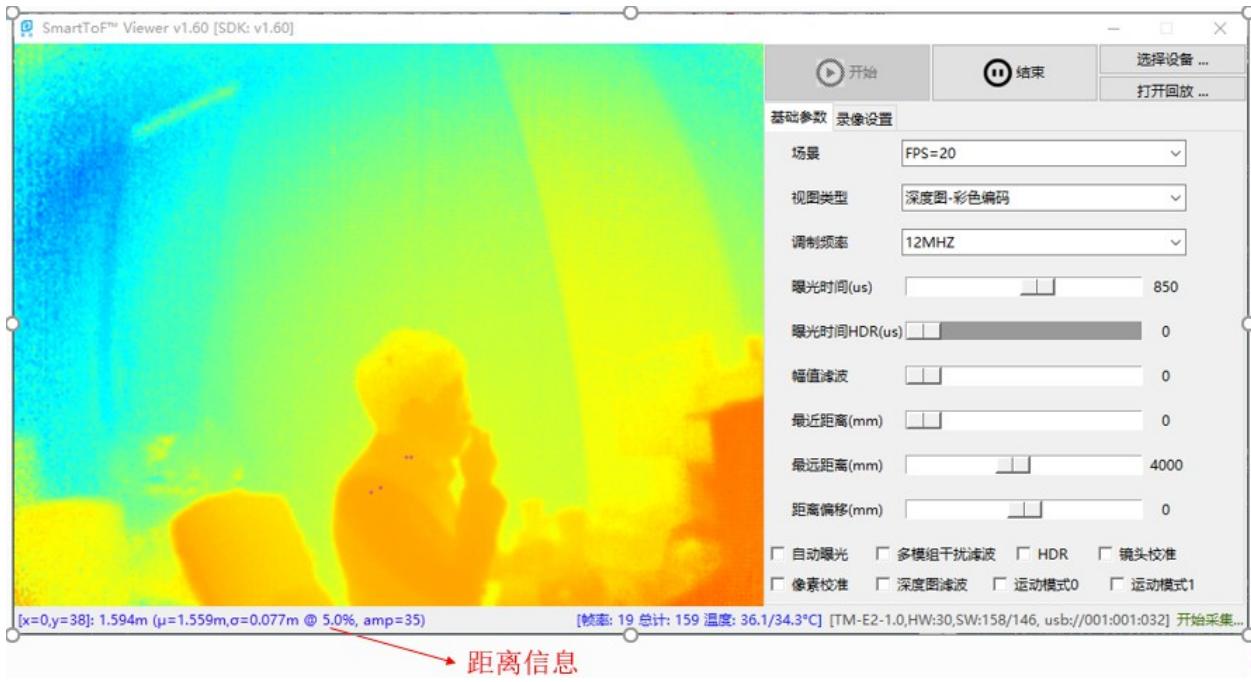
### 11.1.3 详细使用说明

#### 设备选择、开始、结束

选择设备（适用于多个设备接入情形，打开 GUI 时候，会提示选择，如果只接入一个设备，可以直接点击“开始”），点击“选择设备”按钮，如下图



点击“开始”后，图像区显示采集图像，默认开启为深度图模式，在图像显示区下面显示深度信息，结束采集点击“按钮。



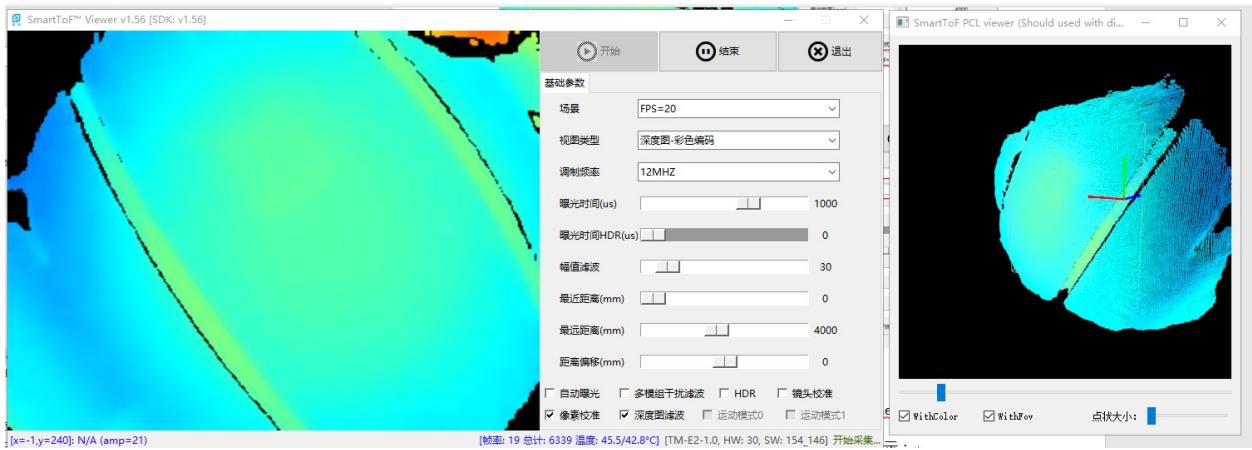
## 图像显示切换

通过基础参数区修改视图类型，可以选择查看灰度图、深度图等四种模式，如灰度图



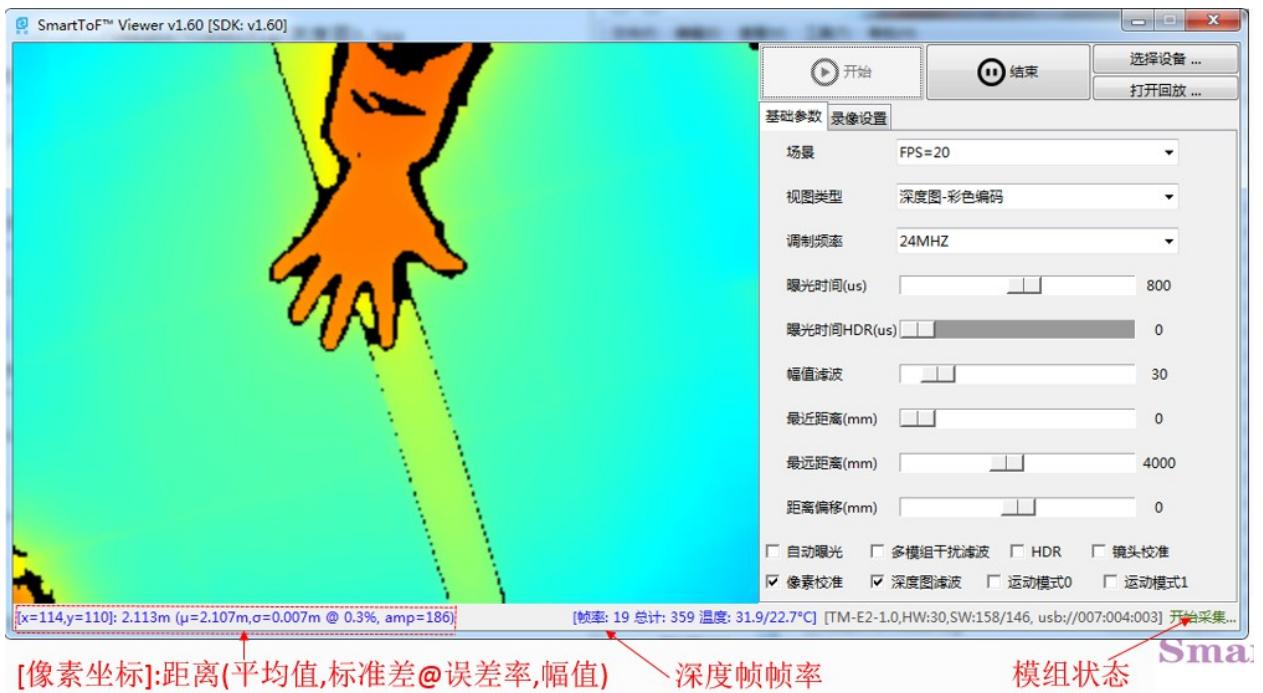
### PCLViewer 显示点云图

如果要开启点云图，则需要同时开启 SmartToFViewer 和 SmartToF\_PCLViewer：如下图



### 查看模组信息及工作状态

模组信息及工作状态可以参考下图

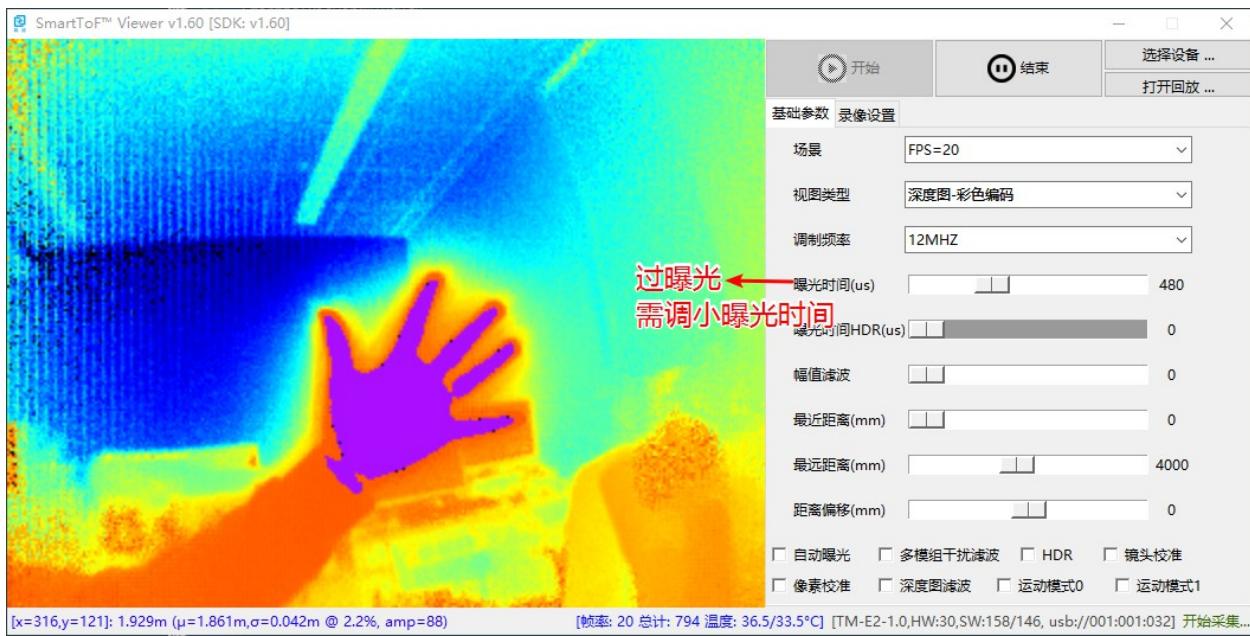


### 基础参数说明

通过 Smartviewer 可以直观的调节帧率、曝光时间、幅值滤波等，基础参数区的详细说明如下图

名称	功能说明
场景	帧率支持10, 15, 20, 30 FPS切换
视图类型	<p>“深度图（彩色编码）”：图像颜色表示不同距离，红-&gt;橙-&gt;黄-&gt;绿...&gt;蓝(由近到远)，如左图，蓝色最远距离，红色最近距离。</p> <p>曝光过度为紫色，如右图，手曝光过度：；黑色表示无效距离。</p> <p>“深度图（灰色编码）”：黑白深度图</p> <p>“灰度图”：黑白图像，可用来辅助镜头对焦。</p>
调制频率	切换红外灯调制频率使用
曝光时间	用来调整曝光时间，当物体图像显示曝光过度时候，可以减小这个值；当物体图像看起来噪点很多，可以增加这个值。
曝光时间HDR	HDR模式下调制第二积分时间使用
幅值滤波	可用来调整图像显示噪点(设定ADC最小值门槛，通过这个值过滤比这个值小的像素点距离信息，即过滤光照不佳的点)
最近距离	过滤用，距离比最小距离小的像素距离信息被设置为最近距离
最远距离	过滤用，距离比最远距离大的像素距离信息被设置为最远距离；通常设置为比被测物体实际距离大的值。
距离偏移	简单平面校准使用，当显示图像距离比实际距离大时，可以增加这个值，显示距离会动态减去这个偏移，达到简单校准效果。

如曝光时间一般在物体不过曝的情况下尽量调大，但不能过曝，过曝部分会呈现紫色如下图：



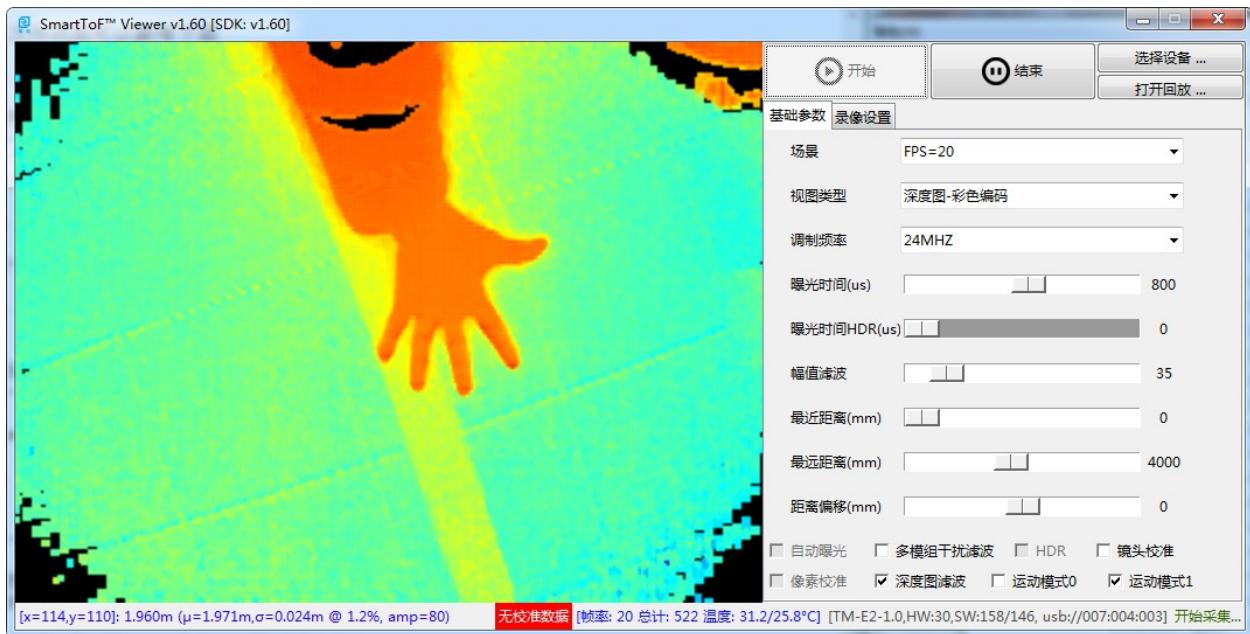
## filter 功能说明

SmartTofViewer 的 filter 功能区主要开启模组相关的滤波功能，滤波功能的具体描述如下图：

名称	功能说明
自动曝光	根据被测物的距离自动调整积分时间
多模组干扰滤波	多台模组同时工作时软件排除串扰
HDR	开启模组HDR功能，同时激活基础参数区的曝光时间HDR
镜头校准	使能模组的镜头校准功能
像素校准	使能模组的校准功能，加载模组的校准数据
深度图滤波	使能深度滤波，滤去噪点，使点云图像显示更美观
运动模式0	开启运动模式，减小物体移动时的拖影
运动模式1	开启运动模式，减小物体移动时拖影，和运动模式0的帧格式不同

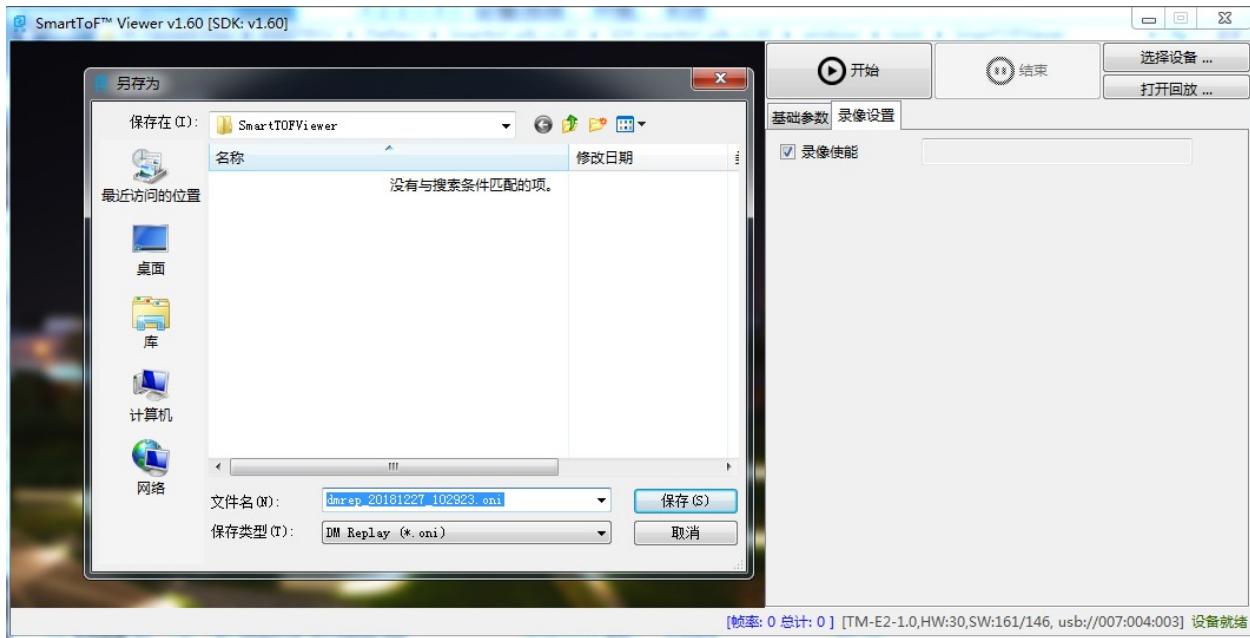
## 运动模式

对于运动速度比较快物体，正常模式图像会有类似拖影、重叠的运动模糊现象，可以勾选“运动模式 0”或者“运动模式 1”使能运动模式，减小运动模糊现象，如下图

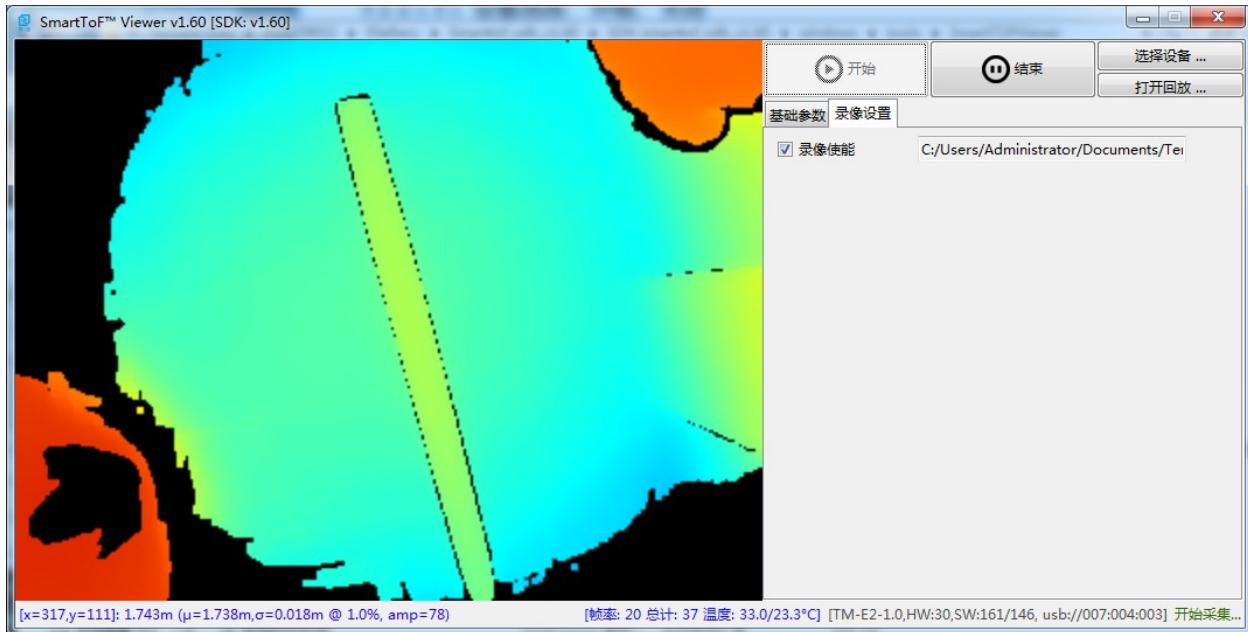


## 录像与回放

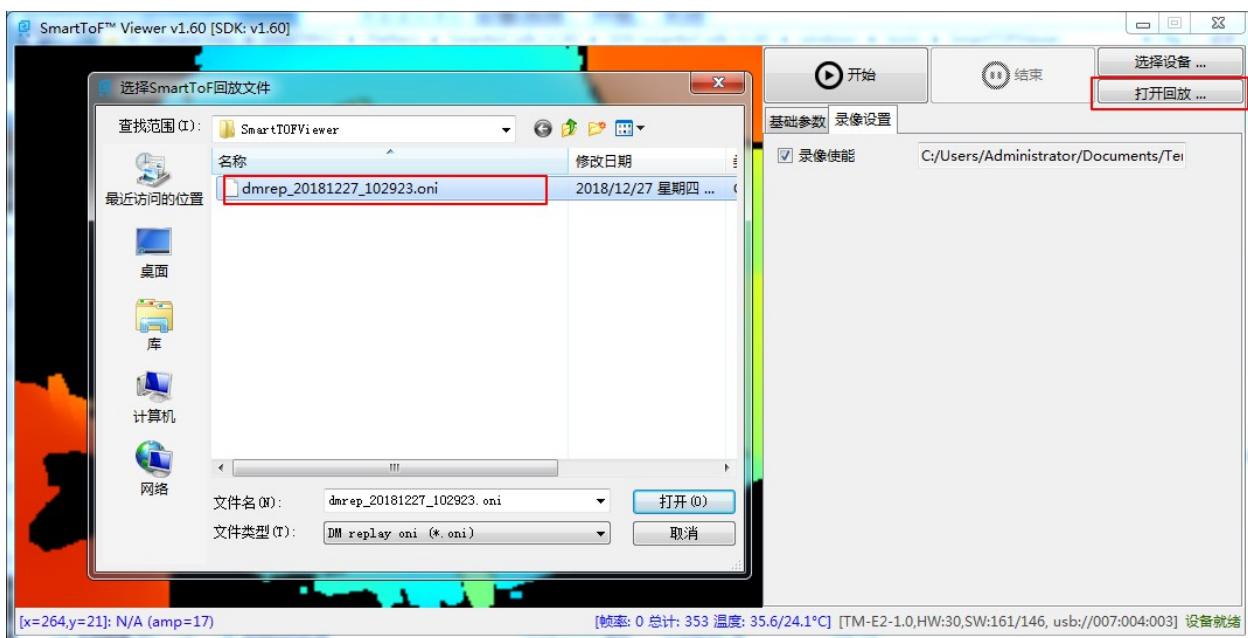
运行启动 SmartTofViewer，切换到“录像设置”菜单，勾选“录像使能”，弹出保存文件名字及目录对话框，保存录像文件，如下图



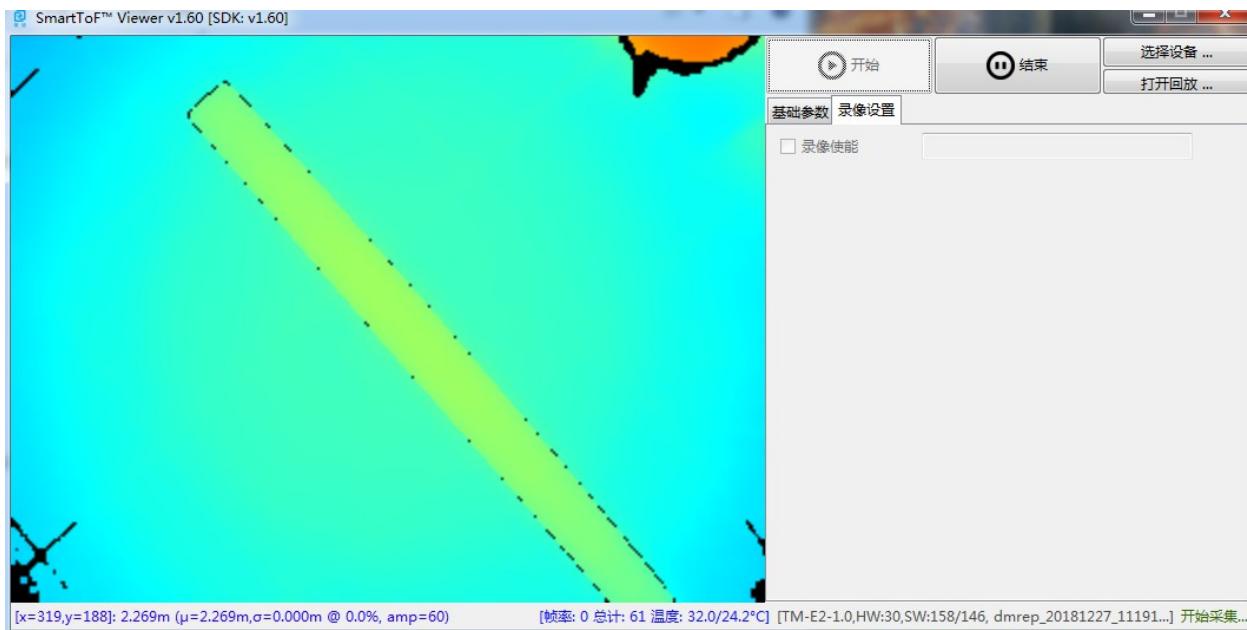
点击开始，此时采集的图像将会保存到本地文件，运行如下图，点击“结束”按钮，停止录像



播放录像文件，点击“打开回放”，选择刚录制的文件，如下图



点击“开始”，运行效果如下图



**Caution:** 录像文件下，“调制频率”、“运动模式 0”、“运动模式 1”、“HDR”、“曝光时间”设置调整无效

## 11.2 SmartToF Cli 使用说明

### 11.2.1 工具概述

SDK 中的 dmcam-cli 工具是方便用户在二次开发时进行诊断和测试使用，工具几乎覆盖了 dmcam 库的所有 API 支持，可以通过 dmcam-cli 工具制作批处理脚本，设计自己想要的场景，获取对应的数据。主要包括以下几个功能：

- 查看可用设备列表
- 硬件设备信息获取
- 目标寄存器配置
- 常用参数读取及设置
- 不同类型图像数据采集和保存
- 设备复位
- 设备固件更新



Script mode functions	options	appended op
复位设备	-r, --reset	
执行 interactive mode command, 可多次使用	-e, --exec <command>	
Run provided script	-s, --script <file>	
进入 interactive mode	-i, --interactive	
print cli version	--version	
print cli help info	-h, --help	
show interactive mode help info	--help-interactive	
固件升级	-f, --flash-MCU-firmware <file>	
写寄存器	--regwr	--target <
读寄存器	--regrd	--target <
写参数	--set <param>	[--param <
读参数	--print <param>	[--param <
<b>以下为交互模式</b>		
查看命令帮助信息	help <cmd>	
设备列表	list	
固件升级	flash <target> <version>	
写寄存器	regwr <target> <base> [<&file>   <P0> <P1>…<P4n>]	
读寄存器	regrd <parameter> <base> [cnt] [&file]	
写参数	set <parameter> <arguments>	
读参数	print/p [parameter]	
采集固定数量的 frame 到文件	rx <data src> [&file] <frame count>	
采集固定数量数据到 buffer	read <frame count>	
同 print info	info	
显示所有 version 信息	version	
采集指定时间距离数据	capture <option> <args>	
filter 参数配置	filter <id> <enabled> [args]	
复位命令	reset <target>	
others		

**Caution:** 针对 TC 系列模组, 谨慎涉及寄存器的读写操作, 误读写可能产生不可预知的问题。

## 查看可用设备信列表

当设备连接后, 可以通过 dmcam-cli -d 命令查看可用设备列表, 命令如下:

```
dmcam-cli -l
```

输出结果如下:

```
4 dmcam device found
[0]: Type=USB  BUS:PORT:ADDR=07:04:03
[1]: Type=USB  BUS:PORT:ADDR=07:03:04
[2]: Type=ETH  IP=192.168.1.38 CID=0xfbfb056c1
[3]: Type=ETH  IP=192.168.1.53 CID=0xf2a4fa3e
```

## 硬件设备信息获取

当设备连接后，可以通过 dmcam-cli 交互模式的 print 命令进行硬件设备信息获取，命令格式如下:

```
p [parameter]
```

```
dmcam> p info

Vendor : Digital Miracle
Product : TM-E2-1.0

Capability:
  max frame width      : 320
  max frame height     : 240
  max frame depth      : 2
  max fps               : 60
  max integrate time(us) : 1500

Serial info: 0x16003020, 0xAA6D94A9, 0x582C8E16

DFU Information:
  MCU Firmware Version : 114
  TFC Firmware Version : 0
```

## 常用参数设置

当设备连接后，可以通过 dmcam-cli 交互模式的 set 命令设置硬件参数，命令格式如下:

```
set <parameter> <arguments>
```

可以通过下面命令查看 set 命令有哪些参数可以设置及参数含义，命令如下，结果见下图:

```
help set
```

```
dmcam> h set
Usage: set <parameter> <arguments>

The set command takes a parameter and an arbitrary number of arguments
for that particular parameter. The parameter is one of:

----- Parameter Description -----
----- mode Device Mode settings
frequency Frequency settings
roi ROI of the device.
        In format: "srow scol erow ecol cur_fsize max_fsize".
frame_format Frame format of the device.
illum_power Illuminate power of the device
frame_rate Frame Rate n
intg_time integrate time.
sys_calib [offset] [slope] //set system calibration coefficients.
amb_light [kceoff]//set ambient calibration coefficient.
-----


dmcam> set intg_time 800
Set integrate time :800 us
dmcam>
```

## 不同类型图像数据采集和保存

当设备连接后，可以通过 dmcam-cli 交互模式的 rx 命令进行数据采集，并将数据存入指定文件，采集的数据格式包括原始数据、深度数据、灰度数据和点云数据，命令格式如下：

```
rx <data src> <&file> <frame count>
```

```
dmcam> help rx

Usage: rx <data format> <&file> <frame count>

Receive frames and write them to the specified file with specified format.
Below is a list of available parameters.

-----
Parameter Description
-----

<file>: Filename to write received frames to

<data format> can be:
replay   : dmcam replay format.
dist_f32: DEPTH in float32 meter unit
dist_u16: DEPTH in uint16 milimeter unit
raw      : RAW DCS
gray     : GRAY image. Each pixel in uint8 (0~255)
pcld    : PCLD image. Each pixel (x,y,z,d) is float32x4. (x,y,z) is 3d coordinate,
          d is distance in meter unit
<frame count>: Number of frames to be captured.

-----
dmcam> rx raw rawfile.bin 10
.....
Save 10 frames to rawfile.bin [frame: RAW DCS 320x240, unit:uint16_t]
```

### 保存或者打印指定区域像素点距离信息

当设备连接后，可以通过 dmcam-cli 交互模式的 capture 命令进行数据采集，并将数据存入指定文件，或者打印出来，采集的数据格式包括温度，距离，幅值，命令格式如下，详细使用可参考 help cap:

```
cap -s 10 -p 119,159,120,160 -c start
```

```

Usage: capture <cmd> <optargs>
The capture control command.

-----
Parameter Description
-----
<cmd> control of capture, include below:
    -s arg, /**>set capture duration in seconds,arg:timeout value*/
    -c arg, /**>capture control cmd,arg see below*/
        start, /**>start capture frames*/
        stop, /**>stop capture frames*/
    -p arg, /**>set print pixel start and end position,arg:x1,y1,x2,y2*/
    -f arg, /**>set print log name,arg:log file name*/
    -n arg, /**>set frame numbers for average calculation*/
    -o arg, /**>output control,arg:0 to console,1 to file*/
    -v arg, /**>verbose control,arg:0 temperature+distance+amplitude*/
-----
```

```

dmcam> cap -s 10 -p 119,159,120,160 -c start
Range(4,6),(323,245)
postion(119,159),(120,160)
start capture....
Total capture :10 seconds

dmcam> MOD_FREQ:24000000 INTG_TIME:800 FRAME_FMT:4 FRAME_RATE:20
Tcb   Tib   dist1   AMP1   dist2   AMP2   dist3   AMP3   dist4   AMP4
265 238 1.957950 78 1.960850 67 30.393050 103 1.608200 155
273 238 1.968000 78 1.970050 67 36.744050 104 1.614150 156
283 239 1.965700 77 1.968650 66 39.915800 105 1.616150 155
283 238 1.969650 78 1.974050 67 43.089150 105 1.616500 155
283 236 1.968200 78 1.971800 67 36.744400 105 1.616200 156
284 235 1.970150 78 1.972150 67 43.090650 105 1.618500 156
285 238 1.973700 78 1.976450 67 33.571850 105 1.620100 154
286 238 1.965100 79 1.969100 67 36.742850 104 1.616750 155
287 237 1.957350 77 1.964050 65 36.739700 102 1.615900 153
287 237 1.968050 77 1.971200 66 46.264150 105 1.618850 156
capture stop

```

**Caution:** cap 命令支持多条输入，比如 cap -o 0 回车，cap -f test.csv 回车，但必须所有参数选项都放在-c start 之前

## 设备复位

当设备连接后，可以通过 dmcam-cli 交互模式的 reset 命令复位设备，命令格式如下：

```
reset <target>
```

测试结果如下图

```
dmcam> help reset
Usage: reset <target>
The reset control command.

-----
Parameter Description
-----
<target> select which target to be reset, include below:
    SYS, /**>reset system*/
    TFC, /**>reset camera sensor*/
    USB, /**>reset device usb*/
    MCU, /**>reset system mcu*/
-----

dmcam> reset tfc
reset ok
dmcam> |
```

## 固件更新

固件更新的详细内容参考 SDK 固件升级

# Chapter 12

## C/C++ 核心库 (libdmcam) 参考

### 12.1 核心 API

#### 12.1.1 dmcam.h

SmartToF SDK 中的所有核心 API 都在 dmcam.h 中说明，详见本章内容。

DM's camera device API.

Detail Decsription starts here

#### Functions

—API void **dmcam\_init**(const char \* *log\_fname*)

Init the DM camera layer. It should be called before any dmcam API is invoked.

##### Parameters

- *log\_fname*: [in] specified log file name of dmcam layer. if NULL, the default log (dmcam\_YYYYMMDD.log) is used.

—API void **dmcam\_uninit**(void)

Uninit the DM camera layer.

—API void **dmcam\_log\_cfg**(dmcam\_log\_level\_e *console\_level*, dmcam\_log\_level\_e *file\_level*, dm-

cam\_log\_level\_e *usb\_level*)

Set the logging configuration for dmcam layer.

##### Parameters

- *console\_level*: [in] specified dmcam\_log\_level\_e, the console log whose log level bellow this value will be suppressed.

- **file\_level:** [in] specified dmcam\_log\_level\_e, the file log whose log level bellow this value will be suppressed.
- **usb\_level:** [in] specified dmcam\_log\_level\_e, the usb log whose log level bellow this value will be suppressed.

— API void **dmcam\_path\_cfg**(const char \* *path*)

Setting where to save calibration data

**Return** — API void

**Parameters**

- **path:**

— API char\* **dmcam\_path\_get**(void)

Getting calibration data path

**Return** — API char\*

— API const char\* **dmcam\_error\_name**(int *error\_code*)

covert specified error code into error string

**Return** const char\*

**Parameters**

- **error\_code:**

— API int **dmcam\_dev\_list**(dmcam\_dev\_t \* *dev\_list*, int *dev\_list\_num*)

list the dmcam device and fill into dmcam\_dev\_t array.

**Return** int [out] number of dmcam device found

**Parameters**

- **dev\_list:** [out] device list array to be filled.
- **dev\_list\_num:** [in] capacity of device list

— API dmcam\_dev\_t\* **dmcam\_dev\_open**(dmcam\_dev\_t \* *dev*)

open specified dmcam device. if the device is not specified, it' ll try to open the first dmcam device

**Return** dmcam\_dev\_t\* NULL = open device failed.

**Parameters**

- **dev:** [in] specified dmcam device which is usually get from dmcam\_dev\_list. if Null, the first dmcam device will be opened.



```
__API bool dmcam_reg_batch_write(dmcam_dev_t * dev, dmcam_dev_reg_e tar-
get, uint32_t reg_base, const uint32_t * reg_vals,
uint16_t reg_vals_len)
```

Batch write registers of specified target on the device.

**Return** bool [out] true = write ok.

#### Parameters

- **dev:** [in] dmcam device handler
- **target:** [in] specified target defined in dmcam\_dev\_reg\_e
- **reg\_base:** [in] base address of the registers
- **reg\_vals:** [in] register values to be written. All register value is denoted as UINT32
- **reg\_vals\_len:** [in] count of values in reg\_vals

```
__API bool dmcam_reg_batch_read(dmcam_dev_t * dev, dmcam_dev_reg_e tar-
get, uint32_t reg_base, uint32_t * reg_vals,
uint16_t reg_vals_len)
```

Batch read registers of specified target on the device.

**Return** bool [out] true = read ok.

#### Parameters

- **dev:** [in] dmcam device handler
- **target:** [in] specified target defined in dmcam\_dev\_reg\_e
- **reg\_base:** [in] base address of the registers
- **reg\_vals:** [out] register values to be filled. All register value is denoted as UINT32
- **reg\_vals\_len:** [in] count of values in reg\_vals

```
__API bool dmcam_param_batch_set(dmcam_dev_t * dev, const dmcam_param_item_t
* param_items, int item_cnt)
```

Batch write generic parameters to specified device.

**Return** bool [out] true = operation is ok.

#### Parameters

- **dev:** [in] dmcam device handler
- **param\_items:** [in] dmcam\_param\_item\_t is used to denotes generic parameter:
  - param\_id[in]: defined in dmcam\_dev\_param\_e to identify the parameters.
  - param\_vals[in]: denotes the generic value (max = 16bytes)
  - param\_vals\_len[in]: denotes the length of value.

- `item_cnt`: [in] count of params in `param_items`

`__API bool dmcam_param_batch_get(dmcam_dev_t * dev, dmcam_param_item_t * param_items,  
int item_cnt)`  
Batch read generic parameters from specified device.

**Return** bool [out] true = operation is ok.

#### Parameters

- `dev`: [in] dmcam device handler
- `param_items`: [in/out] `dmcam_param_item_t` is used to denotes generic parameter:
  - `param_id`[in]: defined in `dmcam_dev_param_e` to identify the parameters.
  - `param_vals`[out]: denotes the generic value (max = 16bytes) filled by this function
  - `param_vals_len`[out]: denotes the length of value filled by this function.
- `item_cnt`: [in] count of params in `param_items`

`__API bool dmcam_cap_config_set(dmcam_dev_t * dev, const dmcam_cap_cfg_t * cfg)`  
Set specified capture configuration for specified device. This api is available from v1.58 to replace  
`dmcam_cap_set_frame_buffer`

**Return** bool [out] true = set OK.

#### Parameters

- `dev`: [in] specified dmcam device
- `cfg`: [in] specified capture configuration

`__API void dmcam_cap_config_get(dmcam_dev_t * dev, dmcam_cap_cfg_t * cfg)`  
Get capture configuration of specified device

#### Parameters

- `dev`: [in] specified dmcam device
- `cfg`: [out] capture configuration to be filled

`__API bool dmcam_cap_set_frame_buffer(dmcam_dev_t * dev, uint8_t * frame_buf,  
uint32_t frame_buf_size)`  
set frame buffer during capturing.

**Return** bool [out] return true = set ok.

#### Parameters

- `dev`: [in] dmcam device handler

- `frame_buf`: [in] framebuffer assigned by user. if null, the frame\_buf will be allocoed internally
- `frame_buf_size`: [in] frame buffer size, which will be rounded to frame size boundary.

— API void `dmcam_cap_set_callback_on_frame_ready(dmcam_dev_t * dev, dmcam_cap_frdy_f cb)`  
register frame ready callback function

#### Parameters

- `dev`: [in] dmcam device handler
- `cb`: [in] callback function in following format: void (`dmcam_cap_frdy_f`)(`dmcam_dev_t`, `dmcam_frame_t`)

— API void `dmcam_cap_set_callback_on_error(dmcam_dev_t * dev, dmcam_cap_err_f cb)`  
register error callback function. It's invoked when some error occurs during the capturing process.

#### Parameters

- `dev`: [in] dmcam device handler
- `cb`: [in] callback function in following format: void (`dmcam_cap_err_f`)(`dmcam_dev_t`, int `errno`);

— API bool `dmcam_cap_snapshot(dmcam_dev_t * dev, uint8_t * frame_data, uint32_t frame_dlen, dmcam_frame_t * frame)`  
Take a snapshot and fill frame data into specified frame. If the device is capturing, the snapshot will return the latest image{} or it'll auto start/snapshot/stop

**Return** bool return true = ok

#### Parameters

- `dev`: [in] dmcam device handler
- `frame_data`: [out] frame data
- `frame_dlen`: [in] frame buffersize should be large enough to containing one frame.
- `frame`: [out] `frame_t` filled during snapshot. it can be null

— API bool `dmcam_cap_is_ongoing(dmcam_dev_t * dev)`  
Check whether the device is in capturing state.

**Return** bool [out] true = device in capturing state

#### Parameters

- `dev`: [in] dmcam device handler

---

— API bool **dmcam\_cap\_start**(dmcam\_dev\_t \* *dev*)  
start device capturing.

**Return** bool return true = ok

#### Parameters

- *dev*: [in] dmcam device handler

— API bool **dmcam\_cap\_stop**(dmcam\_dev\_t \* *dev*)  
stop device capturing.

**Return** bool return true = ok

#### Parameters

- *dev*: [in] dmcam device handler

— API int **dmcam\_cap\_get\_frames**(dmcam\_dev\_t \* *dev*, uint32\_t *frame\_num*, uint8\_t  
\* *frame\_data*, uint32\_t *frame\_dlen*, dmcam\_frame\_t  
\* *first\_frame\_info*)

Get specified number of frames into specified user buffer. This function may be blocking wait on the frame stream. if enough frames data are collected or any error happens, it'll returns.

**Return** int [out] return the number for ready frames collected. On error the errno is returned. (errno < 0)

#### Parameters

- *dev*: [in] dmcam device handler
- *frame\_num*: [in] number of frames to be captured.
- *frame\_data*: [out] frame data filled during capturing.
- *frame\_dlen*: [in] frame\_data buffer size in bytes.
- *first\_frame\_info*: [out] first frame attributes. It can be NULL

— API int **dmcam\_cap\_get\_frame**(dmcam\_dev\_t \* *dev*, uint8\_t \* *frame\_data*,  
uint32\_t *frame\_dlen*, dmcam\_frame\_t \* *frame\_info*)  
get one frame into specified buffer. this function is non-blocking, if no frame is ready, it returns 0

**Return** int return 0 if not frame is ready, else return 1

#### Parameters

- *dev*: [in] dmcam device handler
- *frame\_data*: [out] frame data to be filled, it can be NULL
- *frame\_info*: [out] frame attributes. It can be NULL

— API int **dmcam\_firmware\_upgrade**(dmcam\_dev\_t \* *dev*, uint8\_t *type*, uint16\_t *version*, const char \* *file\_name*)  
Firmware upgrade for different type target.

**Return** int

#### Parameters

- *dev*[in]: dmcam: device handler
- *type*[in]: firmware: type
- *version*[in]: firmware: version
- *file\_name*[in]: firmware: name

— API int **dmcam\_data\_download**(dmcam\_dev\_t \* *dev*, char \* *name*, uint8\_t *type*, uint16\_t *version*, uint32\_t *addr*)

— API int **dmcam\_data\_upload**(dmcam\_dev\_t \* *dev*, uint8\_t *type*, const char \* *file\_name*)

— API int **dmcam\_frame\_get\_distance**(dmcam\_dev\_t \* *dev*, float \* *dst*, int *dst\_len*, uint8\_t \* *src*, int *src\_len*, const dmcam\_frame\_info\_t \* *finfo*)  
alias for dmcam\_frame\_get\_dist\_f32

— API int **dmcam\_frame\_get\_dist\_f32**(dmcam\_dev\_t \* *dev*, float \* *dst*, int *dst\_len*, uint8\_t \* *src*, int *src\_len*, const dmcam\_frame\_info\_t \* *finfo*)  
convert to distance data to float32 from raw frame data.

**Return** int [out] return the number for distance points in dst

#### Parameters

- *dev*: [in] specified dmcam device
- *dst*: [out] distance buffer. The unit of distance is in meters (float32)
- *dst\_len*: [in] distance buffer length in number of float
- *src*: [in] raw frame data buffer
- *src\_len*: [in] raw frame data length in byte
- *finfo*: [in] raw frame information

— API int **dmcam\_frame\_get\_dist\_u16**(dmcam\_dev\_t \* *dev*, uint16\_t \* *dst*, int *dst\_len*, uint8\_t \* *src*, int *src\_len*, const dmcam\_frame\_info\_t \* *finfo*)  
convert to distance data in uint16 from raw frame data.

**Return** int [out] return the number for distance points in dst

#### Parameters

- *dev*: [in] specified dmcam device
- *dst*: [out] distance buffer. The unit of distance is in millimeter (uint16)

- **dst\_len:** [in] distance buffer length in number of uint16
- **src:** [in] raw frame data buffer
- **src\_len:** [in] raw frame data length in byte
- **finfo:** [in] raw frame information

API int dmcam\_frame\_get\_gray(dmcam\_dev\_t \* dev, float \* dst, int dst\_len, uint8\_t \* src, int src\_len, const dmcam\_frame\_info\_t \* finfo)  
alias for dmcam\_frame\_get\_gray\_f32

API int dmcam\_frame\_get\_gray\_f32(dmcam\_dev\_t \* dev, float \* dst, int dst\_len, uint8\_t \* src, int src\_len, const dmcam\_frame\_info\_t \* finfo)  
get gray data in float32 from raw frame data.

**Return** int [out] return the number for gray points in dst

#### Parameters

- **dev:** [in] specified dmcam device
- **dst:** [out] gray buffer. The gray value denotes the amplitude. (float32 in [0, 2048.0] )
- **dst\_len:** [in] distance buffer length in number of float
- **src:** [in] raw frame data buffer
- **src\_len:** [in] raw frame data length in byte
- **finfo:** [in] raw frame information

API int dmcam\_frame\_get\_gray\_u16(dmcam\_dev\_t \* dev, uint16\_t \* dst, int dst\_len, uint8\_t \* src, int src\_len, const dmcam\_frame\_info\_t \* finfo)  
get gray data in uint16\_t from raw frame data.

**Return** int [out] return the number for gray points in dst

#### Parameters

- **dev:** [in] specified dmcam device
- **dst:** [out] gray buffer. The gray value denotes the amplitude. (uint16\_t in [0, 2048])
- **dst\_len:** [in] distance buffer length in number of uint16\_t
- **src:** [in] raw frame data buffer
- **src\_len:** [in] raw frame data length in byte
- **finfo:** [in] raw frame information

```
— API int dmcam_frame_get_pcl(dmcam_dev_t * dev, float * pcl, int pcl_len, const float * dist,
                                int dist_len, int img_w, int img_h, const dmcam_camera_para_t
                                * p_cam_param)
```

get point cloud data from distance data. The distance data is usually calcuated using dmcam\_frame\_get\_dist\_f32.

**Return** int [out] return number of points in point cloud buffer. Note: n points means 3\*n floats. N should be img\_w \* img\_h

#### Parameters

- **dev:** [in] specified dmcam device
- **pcl:** [out] point clound buffer. each 3 element consists a (x,y,z) point, output is in (w,h,3) demension. point in value (0,0,0) is invalid
- **pcl\_len:** [in] point cloud float element count
- **dist:** [in] distance image data buffer. The unit of distance is meter (float)
- **dist\_len:** [in] distance image data count (in sizeof(float))
- **img\_w:** [in] distance image width in pixel
- **img\_h:** [in] distance image height in pixel
- **p\_cam\_param:** [in] user specified camera lens parameter. if null, the internal camera parameter is used.

```
int dmcam_frame_get_pcl_xyzd(dmcam_dev_t * dev, float * pcl, int pcl_len, const float * dist,
                               int dist_len, int img_w, int img_h, bool pseudo_color, const dm-
```

cam\_camera\_para\_t \* p\_cam\_param)

get point cloud data from distance data. The distance data is usually calcuated using dmcam\_frame\_get\_distance.

**Return** int [out] return number of points in point cloud buffer. Note: n points means 4\*n floats. N should be img\_w \* img\_h

#### Parameters

- **dev:** [in] specified dmcam device
- **pcl:** [out] point clound buffer. each 4 element consists a (x,y,z,d) point. (x,y,z) is coordinate, d is distance or pseudo-color. output is in (w,h,4) demension. point in value (0,0,0) is invalid
- **pcl\_len:** [in] point cloud float element count
- **dist:** [in] distance image data buffer. The unit of distance is meter (float)
- **dist\_len:** [in] distance image data count (in sizeof(float))
- **img\_w:** [in] distance image width in pixel

- `img_h`: [in] distance image height in pixel
- `pseudo_color`: [in] if true, `d` is pseudo uint32 rgb color value; if false, `d` is the distance in meter
- `p_cam_param`: [in] user specified camera lens parameter. if null, the internal camera parameter is used.

`__API int dmcam_filter_enable(dmcam_dev_t * dev, dmcam_filter_id_e filter_id, dmcam_filter_args_u * filter_arg, uint32_t reserved)`  
Enable filter controller setting for raw data processing

**Return** int 0 = OK, otherwise failed.

#### Parameters

- `dev`: [in] dmcam device handler
- `filter_id`: [in]:defined in `dmcam_filter_id_e` to identify the filter
- `filter_arg`: [in] filter control args
- `reserved`: [in] reserved for future use. User should set to 0

`__API int dmcam_filter_disable(dmcam_dev_t * dev, dmcam_filter_id_e filter_id)`  
Disable filter controller setting for raw data processing

**Return** int 0 = OK, otherwise failed.

#### Parameters

- `dev`: [in] dmcam device handler
- `filter_id`: [in] defined in `dmcam_filter_id_e` to identify the filter

`__API int dmcam_cmap_dist_f32_to_RGB(uint8_t * dst, int dst_len, const float * src, int src_len, dmcam_cmap_outfmt_e outfmt, float range_min_m, float range_max_m)`  
convert dist\_f32 image (pixel in meter) to pesudo-RGB points with specified pixel format

**Return** int [out] the count of pseudo RGB points

#### Parameters

- `dst`: [out] pseudo-RGB point buffer
- `dst_len`: [in] point buffer size in bytes
- `src`: [in] float points buffer
- `src_len`: [in] count of float points
- `outfmt`: [in] pixel format of the pseudo-RGB

- `min_val`: [in] minimum range of source point
- `max_val`: [in] max range of source point

```
__API int dmcam_cmap_float(uint8_t * dst, int dst_len, const float * src, int src_len,
                           dmcam_cmap_outfmt_e outfmt, float range_min_m,
                           float range_max_m)
```

```
__API int dmcam_cmap_dist_u16_to_RGB(uint8_t * dst, int dst_len, const uint16_t * src,
                                       int src_len, dmcam_cmap_outfmt_e outfmt,
                                       uint16_t range_min_mm, uint16_t range_max_mm)
```

convert dist\_u16 image (pixel in milimeter) to pesudo-RGB points with specified pixel format

**Return** int [out] the count of pseudo RGB points

#### Parameters

- `dst`: [out] pseudo-RGB point buffer
- `dst_len`: [in] point buffer size in bytes
- `src`: [in] dist\_u16 image buffer
- `src_len`: [in] count of u16 points
- `outfmt`: [in] pixel format of the pseudo-RGB
- `min_val`: [in] minimum range of source point
- `max_val`: [in] max range of source point

```
__API int dmcam_cmap_gray_u16_to_IR(uint8_t * dst, int dst_len, const uint16_t * src, int src_len,
                                      int balance)
```

convert gray\_u16 image to IR image whose pixel is in [0~255]

**Return** int [out] the count of IR image

#### Parameters

- `dst`: [out] IR image buffer
- `dst_len`: [in] IR image buffer size in bytes
- `src`: [in] gray\_u16 image
- `src_len`: [in] count of u16 points in gray\_u16 image
- `balance`: [in] [-1024, 1024] -> [darkest, brightest]

```
__API int dmcam_cmap_gray_f32_to_IR(uint8_t * dst, int dst_len, const float * src, int src_len,
                                      int balance)
```

convert gray\_f32 image to IR image whose pixel is in [0~255]

**Return** int [out] the count of IR image

**Parameters**

- **dst**: [out] IR image buffer
- **dst\_len**: [in] IR image buffer size in bytes
- **src**: [in] gray\_f32 image
- **src\_len**: [in] count of f32 points in gray\_f32 image
- **balance**: [in] [-1024, 1024] -> [darkest, brightest]

— API int **dmcam\_file\_open**(const char \* *fname*, const char \* *mode*)  
 open specified file and get file descriptor for dmcam\_frame\_save\_xxx apis.

**Return** int [out] file descriptor. < 0 = failed

**Parameters**

- **fname**: [in] specified filename

— API void **dmcam\_file\_close**(int *fd*)  
 close specified file descriptor

**Parameters**

- **fd**: [in] specified file descriptor

— API bool **dmcam\_frame\_save\_raw**(int *fd*, dmcam\_frame\_save\_fmt\_t *save\_fmt*, const uint16\_t  
 \* *raw*, int *raw\_len*, int *img\_w*, int *img\_h*, int *dcs\_cnt*, const  
 char \* *raw\_tag*)  
 save specified raw data (in uint16\_t) with specified pixcel width and height to file with specified saving  
 format.

**Return** bool [out] true = save raw frame ok, false = fail

**Parameters**

- **fd**: [in] specified file handler
- **save\_fmt**: [in] file saving format defined in dmcam\_frame\_save\_fmt\_t. only followin format  
 is supported: DMCAM\_FRAME\_SAVE\_UINT32 DMCAM\_FRAME\_SAVE\_UINT16
- **raw**: [in] raw data
- **raw\_len**: [in] number of raw data (in count of uint16\_t)
- **img\_w**: [in] dist data pixel width
- **img\_h**: [in] dist data pixel height
- **dcs\_cnt**: [in] dist data dcs sub-frame count

- **raw\_tag**: [in] any string. if want to used by replay, specify (dmcam\_t\*)dev->product string here.

— API bool **dmcam\_frame\_save\_distance**(int *fd*, dmcam\_frame\_save\_fmt\_t *save\_fmt*, const float \* *dist*, int *dist\_len*, int *img\_w*, int *img\_h*)  
save specified distance data (in float32, unit: meter) with specified pixcel width and height to file with specified saving format.

**Return** bool [out] true = save distance frame ok, false = fail

#### Parameters

- **fd**: [in] specified file handler
- **save\_fmt**: [in] file saving format defined in @ dmcam\_frame\_save\_fmt\_t. only followin format is supported: DMCAM\_FRAME\_SAVE\_FLOAT32 DMCAM\_FRAME\_SAVE\_UINT32 DMCAM\_FRAME\_SAVE\_UINT16
- **dist**: [in] distance data (in float32, unit: meter)
- **dist\_len**: [in] number of distance data (in count of float)
- **img\_w**: [in] dist data pixel width
- **img\_h**: [in] dist data pixel height

— API bool **dmcam\_frame\_save\_gray**(int *fd*, dmcam\_frame\_save\_fmt\_t *save\_fmt*, const float \* *src*, int *src\_len*, int *img\_w*, int *img\_h*)  
save specified gray data (in float32) with specified pixcel width and height to file with specified saving format.

**Return** bool [out] true = save distance frame ok, false = fail

#### Parameters

- **fd**: [in] specified file handler
- **save\_fmt**: [in] file saving format defined in dmcam\_frame\_save\_fmt\_t. only followin format is supported: DMCAM\_FRAME\_SAVE\_UINT16 DMCAM\_FRAME\_SAVE\_UINT8
- **src**: [in] gray data (in float32)
- **src\_len**: [in] number of distance data (in count of float)
- **img\_w**: [in] dist data pixel width
- **img\_h**: [in] dist data pixel height

— API int **dmcam\_frame\_load\_raw**(int *fd*, uint16\_t \* *dst*, int *dst\_len*, int \* *dst\_w*, int \* *dst\_h*, int \* *dst\_dcsn*, char \* *dst\_tag*, int *dst\_tag\_len*)  
load one raw frame from specified file fd.

**Return** int [out] length of loaded raw data (in count of sizeof(uint16))

**Parameters**

- **fd:** [in] specified data file fd. The fd related file is always saved by *dmcam\_frame\_save\_raw* api
- **dst:** [out] raw
- **dst\_len:** [in] dst buffer length (in count of sizeof(uint16\_t))
- **dst\_w:** [out] raw frame pixel width
- **dst\_h:** [out] raw frame pixel height
- **dst\_dcsn:** [out] raw dcs cnt per frame
- **dst\_tag:** [out] raw data tag string
- **tag\_len:** [in] raw data tag buffer size

API int **dmcam\_frame\_load\_distance**(int *fd*, float \* *dst*, int *dst\_len*, int \* *dst\_w*, int \* *dst\_h*)  
load one distance frame from specified file fd.

**Return** int [out] length of loaded distance data (in count of sizeof(float))

**Parameters**

- **fd:** [in] specified data file fd. The fd related file is always saved by *dmcam\_frame\_save\_distance* api
- **dst:** [out] distance in float (unit: meter)
- **dst\_len:** [in] dst buffer length (in count of sizeof(float))
- **dst\_w:** [out] distance frame pixel width
- **dst\_h:** [out] distance frame pixel height

API int **dmcam\_frame\_load\_gray**(int *fd*, float \* *dst*, int *dst\_len*, int \* *dst\_w*, int \* *dst\_h*)  
load one gray frame from specified file fd.

**Return** int [out] length of loaded gray data (in count of sizeof(float))

**Parameters**

- **fd:** [in] specified data file fd. The fd related file is always saved by *dmcam\_frame\_save\_gray* api
- **dst:** [out] gray in float (unit: meter)
- **dst\_len:** [in] dst buffer length (in count of sizeof(float))
- **dst\_w:** [out] gray frame pixel width
- **dst\_h:** [out] gray frame pixel height

## 12.2 模组参数和滤波类型说明

SmartToF 中主要通过模组参数设置和滤波功能设置来控制模组采集时的功能配置，所有的模组参数项和滤波功能项定义都在 SDK 中的 dmcam.h 头文件中。

### 12.2.1 模组参数说明

模组参数枚举类型定义：

```
typedef enum {
    PARAM_DEV_MODE = 0,
    PARAM_MOD_FREQ,

    PARAM_INFO_VENDOR,
    PARAM_INFO_PRODUCT,
    PARAM_INFO_CAPABILITY,
    PARAM_INFO_SERIAL,
    PARAM_INFO_VERSION,      //HW&SW info
    PARAM_INFO_SENSOR,       //part version, chip id, wafer id

    PARAM_INFO_CALIB,        //get calibration info
    PARAM_ROI,               //ROI set/get
    PARAM_FRAME_FORMAT,      //frame information,eg.dcs1for gray,4 dcs for distance
    PARAM_ILLUM_POWER,        //illumination power set/get
    PARAM_FRAME_RATE,         //frame rate set/get
    PARAM_INTG_TIME,          //integration time set/get
    PARAM_PHASE_CORR,         //phase offset correction
                            //PARAM_SWITCH_MODE, /*>s with modeu

    ↵use [gray,3d]*/
    PARAM_TEMP,                //Get camera temperature-----
    PARAM_HDR_INTG_TIME,      //Setting HDR integration time param
    PARAM_SYNC_DELAY,          //delay ms for sync use
    PARAM_ENUM_COUNT,
}dmcam_dev_param_e;
```

Table 1: 模组参数枚举类型说明

参数名	取值范围	说明
PARAM_DEV_MODE	不用设置	用户无需设置此参数
PARAM_MOD_FREQ	12M、24M、36M 单位 Hz	频率要对应校准数据
PARAM_INFO_VENDOR	不用设置	默认 Digital Miracle
PARAM_INFO_PRODUCT	不用设置	模组名称, 如 TC-E2-1.0
PARAM_INFO_CAPABILITY	不用设置	
PARAM_INFO_SERIAL	不用设置	3 个 uint32_t 类型数字
PARAM_INFO_VERSION	不用设置	包括软件和硬件版本信息
PARAM_INFO_CALIB	不用设置	包含校准相关信息
PARAM_ROI	320*240 320*160	
PARAM_FRAME_FORMAT	2,4	默认模式是 2, 运动模式 1 需设置为 4
PARAM_ILLUM_POWER	暂不用设置	
PARAM_FRAME_RATE	1-36	通常取值 5、10、20、30
PARAM_INTG_TIME	0us-1500us	如果超过最大范围, 对模组可能造成损坏
PARAM_PHASE_CORR	暂不支持设置	
PARAM_TEMP	只读	分别读取传感器上下面左右部分温度和灯板的温度
PARAM_HDR_INTG_TIME	0-1500	取值同 PARAM_INTG_TIME, 设置不为 0 时则开启 HDR 模式
PARAM_SYNC_DELAY	0ms-6ms	

## 12.2.2 模组滤波类型说明

模组滤波类型包括深度滤波、幅值滤波、自动曝光、运动模式等, 具体定义如下:

```
typedef enum {
    DMCAM_FILTER_ID_LEN_CALIB,      /**>lens calibration*/
    DMCAM_FILTER_ID_PIXEL_CALIB,    /**>pixel calibration*/
    DMCAM_FILTER_ID_MEDIAN,         /**>Median filter for distance data*/
    DMCAM_FILTER_ID_RESERVED,       /**>Gauss filter for distance data*/
    DMCAM_FILTER_ID_AMP,            /**>Amplitude filter control*/
    DMCAM_FILTER_ID_AUTO_INTG,      /**>auto integration filter enable : use sat_ratio
    ↵to adjust */
    DMCAM_FILTER_ID_SYNC_DELAY,     /**> sync delay */
    DMCAM_FILTER_ID_TEMP_MONITOR,   /**>temperature monitor */
    DMCAM_FILTER_ID_HDR,           /**>HDR mode */
}
```

(continues on next page)

(continued from previous page)

```

DMCAM_FILTER_ID_OFFSET,           /**> set offset for calc distance */
DMCAM_FILTER_ID_SPORT_MODE,     /**> set sport mode */
//-
DMCAM_FILTER_CNT,
}dmcam_filter_id_e;

```

Table 2: 模组滤波类型说明

滤波功能 ID	说明
DMCAM_FILTER_ID_LEN_CALIB	镜头校准 ID
DMCAM_FILTER_ID_PIXEL_CALIB	像素校准 ID
DMCAM_FILTER_ID_MEDIAN	深度滤波 ID
DMCAM_FILTER_ID_AMP	最小幅值滤波 ID
DMCAM_FILTER_ID_AUTO_INTG	自动曝光 ID
DMCAM_FILTER_ID_SYNC_DELAY	软件多模组串扰 ID
DMCAM_FILTER_ID_HDR	HDR 功能 ID
DMCAM_FILTER_ID_OFFSET	距离偏移功能 ID
DMCAM_FILTER_ID_SPORT_MODE	运动模式功能 ID 非运动模式: 全分辨率, 4xDCS, 18ms 模糊运动模式 0: 垂直分辨率减半, 4xDCS, 6ms 模糊运动模式 1: 垂直分辨率减半, 2xDCS, 无模糊, 噪声高 (竖条纹) 运动模式 2: 全分辨率, 2xDCS, 6ms 模糊, 噪声高 (竖条纹)

## 12.3 参数和滤波代码示例

模组参数和滤波类型说明 已经介绍了模组相关的参数和滤波功能 ID, 下面将说明如何设置及获取模组参数已经使能关闭模组的相关滤波功能。

### 12.3.1 模组参数设置和读取

下面以设置和读取模组的曝光时间为例:

- 进行参数设置:

```

dmcam_param_item_t wparam;
uint16_t intg_time = 100;                                //表示设置积分的大小 范围为 0-1500
memset(&wparam, 0, sizeof(wparam));
wparam.param_id = PARAM_INTG_TIME;                         //表示设置的参数为积分时间
wparam.param_val_len = sizeof(intg_time);

```

(continues on next page)

(continued from previous page)

```
wparam.param_val.intg.intg_us = intg_time;
assert(dmcam_param_batch_set(dev,&wparam,1)); //调用 API 进行单个参数设置

dmcam_param_item_t wparam;
uint16_t intg_hdrtime = 700; //表示设置 HDR 的积分时间的大小
memset(&wparam,0,sizeof(wparam));
wparam.param_id = PARAM_HDR_INTG_TIME; //表示设置的参数为积分时间
wparam.param_val_len = sizeof(intg_time);
wparam.param_val.intg.intg_us = intg_hdrtime;
assert(dmcam_param_batch_set(dev,&wparam,1)); //调用 API 进行单个参数设置
```

**Tip:** 如果 hdr 的积分时间设置了不为 0，则开启了模组的 HDR 模式。

- 进行参数的读取：

```
dmcam_param_item_t rparam;
memset(&rparam,0,sizeof(rparam));
rparam.param_id = PARAM_INTG_TIME; //表示要读取的参数项为积分时间
assert(dmcam_param_batch_get(dev,&rparam,1)); //调用 API 获取单个参数
```

其他模组参数的设置获取参照上面积分时间的设置获取。

### 12.3.2 滤波功能开启和关闭

- 像素校准，开启后用于深度数据的矫正：

```
dmcam_filter_args_u witem;
dmcam_filter_id_e filter_id = DMCAM_FILTER_ID_PIXEL_CALIB; //像素校准
dmcam_filter_enable(dev,filter_id,&witem,sizeof(dmcam_filter_args_u)); //开启像素校准
dmcam_filter_disable(dev,DMCAM_FILTER_ID_PIXEL_CALIB); //关闭像素校准
```

- 深度滤波，开启后用于对深度数据滤波：

```
dmcam_filter_args_u witem;
dmcam_filter_id_e filter_id = DMCAM_FILTER_ID_MEDIAN; //深度滤波
witem.median_ksize = 3; //深度滤波通常设置值
dmcam_filter_enable(dev,filter_id,&witem,sizeof(dmcam_filter_args_u)); //开启深度滤波
dmcam_filter_disable(dev,DMCAM_FILTER_ID_MEDIAN); //关闭深度滤波
```

- 幅值滤波，开启后过滤质量差的点：

```

dmcam_filter_args_u witem;
dmcam_filter_id_e filter_id = DMCAM_FILTER_ID_AMP;           //最小幅值滤波
witem.min_amp = 30;      //设置的最小幅值滤波的阈值
dmcam_filter_enable(dev,filter_id,&witem,sizeof(dmcam_filter_args_u)); //开启最小幅值滤波
dmcam_filter_disable(dev,DMCAM_FILTER_ID_AMP); //关闭最小幅值滤波

```

- HDR 模式，设置一大一小两个积分时间，确保同一个模组在测量远近不同的物体时不会过曝：

```

dmcam_filter_args_u witem;
dmcam_filter_id_e filter_id = DMCAM_FILTER_ID_HDR;           //HDR 模式
witem.intg.intg_3d = 100;          //HDR 模式时小的曝光时间
witem.intg.intg_3dhdr = 700;     //HDR 模式时大的曝光时间
dmcam_filter_enable(dev,filter_id,&witem,sizeof(dmcam_filter_args_u)); //开启 HDR 模式

```

**Tip:** 开启 HDR 模式的另一种方法是可以直接通过设置 HDR 的曝光时间不为 0，如上面设置积分的样例。

- 自动积分时间，开启后根据被测物体的距离自动调整曝光时间大小：

```

dmcam_filter_args_u witem;
dmcam_filter_id_e filter_id = DMCAM_FILTER_ID_AUTO_INTG;       //自动曝光
witem.sat_ratio = 5; //自动曝光时设置的值
dmcam_filter_enable(dev,filter_id,&witem,sizeof(dmcam_filter_args_u)); //开启自动曝光
dmcam_filter_disable(dev,DMCAM_FILTER_ID_AUTO_INTG); //关闭自动曝光

```

- 多模组串扰消除，开启消除或者减小多模组同时开启时的串扰：

```

dmcam_filter_args_u witem;
dmcam_filter_id_e filter_id = DMCAM_FILTER_ID_SYNC_DELAY;        //串扰延时
witem.sync_delay = 0; //延时时间设为 0 时为随机时间
dmcam_filter_enable(dev,filter_id,&witem,sizeof(dmcam_filter_args_u)); //开启串扰延时
dmcam_filter_disable(dev,DMCAM_FILTER_ID_SYNC_DELAY); //关闭串扰延时

```

- 运动模式 0，帧格式要设置为 2：

```

dmcam_filter_args_u witem;
dmcam_filter_id_e filter_id = DMCAM_FILTER_ID_SPORT_MODE;        //运动模式 0
dmcam_param_item_t wparam;
uint32_t set_format = 2; //表示要设置的帧格式为 2
memset(&wparam,0,sizeof(wparam));

```

(continues on next page)

(continued from previous page)

```
wparam.param_id = PARAM_FRAME_FORMAT; //表示设置的参数为帧格式
wparam.frame_format.format = set_format; //设置的帧格式为 2
wparam.param_val_len = sizeof(set_format);
assert(dmcam_param_batch_set(dev,&wparam,1)); //调用 API 进行帧格式参数设置
witem.sport_mode = 0; //设置运动模式为 0
dmcam_filter_enable(dev,filter_id,&witem,sizeof(dmcam_filter_args_u)); //开启运动模式
0
dmcam_filter_disable(dev,DMCAM_FILTER_ID_SPORT_MODE); //关闭运动模式 0
```

- 运动模式 1, 帧格式要设置为 4:

```
dmcam_filter_args_u witem;
dmcam_filter_id_e filter_id = DMCAM_FILTER_ID_SPORT_MODE; //运动模式 1
dmcam_param_item_t wparam;
uint32_t set_format = 4; //表示要设置的帧格式为 4
memset(&wparam,0,sizeof(wparam));
wparam.param_id = PARAM_FRAME_FORMAT; //表示设置的参数为帧格式
wparam.frame_format.format = set_format; //设置的帧格式为 4
wparam.param_val_len = sizeof(set_format);
assert(dmcam_param_batch_set(dev,&wparam,1)); //调用 API 进行帧格式参数设置
witem.sport_mode = 1; //设置运动模式为 1
dmcam_filter_enable(dev,filter_id,&witem,sizeof(dmcam_filter_args_u)); //开启运动模式
1
//关闭运动模式 1 时, 帧格式要先恢复到 2
set_format = 2; //帧格式要恢复设置为 2
wparam.frame_format.format = set_format; //设置的帧格式的值为 2
wparam.param_id = PARAM_FRAME_FORMAT; //表示设置的参数为帧格式
wparam.param_val_len = sizeof(set_format);
assert(dmcam_param_batch_set(dev,&wparam,1)); //调用 API 进行帧格式参数设置 2
dmcam_filter_disable(dev,DMCAM_FILTER_ID_SPORT_MODE); //关闭运动模式 1
```

## 12.4 录像生成与读取回放代码说明

在模组的算法评估中, 评估前的一个主要准备工作就是录像文件的获取, 在 SmartToFViewer 中已经介绍如何获取通过 SmartToFViewer 进行录像以及播放, 这里介绍如何通过程序进行录像和读取录像文件, 以及主要 API 的介绍。

### 12.4.1 录像文件的设置

是否保存录像文件以及录像文件保存何种数据主要通过在程序中设置 `dmcam_cap_cfg_t` 的数据结构，具体代码如下

```
/* set capture config */
dmcam_cap_cfg_t cap_cfg = {
    .cache_frames_cnt = FRAME_BUF_FCNT, /* FRAME_BUF_FCNT frames can be cached in frame buffer*/
    .on_error = NULL, /* No error callback */
    .on_frame_ready = NULL, /* No frame ready callback */
    .en_save_replay = false, /* false save raw data stream to replay file */
    .en_save_dist_u16 = false, /* disable save dist stream into replay file */
    .en_save_gray_u16 = false, /* disable save gray stream into replay file */
    .fname_replay = NULL, /* replay filename */
};
```

上述结构体参数中跟录像文件相关的主要有 `en_save_replay` `en_save_dist_u16` `en_save_gray_u16` 这几个参数，这几个参数的具体说明如下：

- 支持 SmartToF SDK 标准回放

如果只要支持 SmartToF SDK 标准的回放，只要将上述的 `en_save_replay` 设置为 `true`，其他两个设置为 `false`。

- 兼容 OpenNI 工具的回放（如 NiViewer）

支持 OpenNI 工具的回放，如 NiViewer 的播放，这时需要将 `en_save_replay` 设置 `false`，`en_save_dist_u16` 和 `en_save_gray_u16` 都设置为 `true` 或者任意一个设置 `true`，如果都设置为使能，则表示存储深度图和灰度图，其中一个使能则使能 `en_save_dist_u16` 为深度图，使能 `en_save_gray_u16` 则表示保存的为灰度图。

### 12.4.2 录像文件的读取

SmartToF 的录像文件“xxx.oni”可以被模拟成标准的 DMCAM 设备，可以通过 `dmcam_dev_open_by_uri` 函数打开，如文件名为“test.oni”，则读取文件名代码如下

```
dev = dmcam_dev_open_by_uri("test.oni") //or file://test.oni
```

**Caution:** 模拟 dmcam 设备和真实 dmcam 设备的区别

- 模拟设备时，凡是 SDK 中基于原始 DCS 数据的处理均可调节和叠加，例如：深度滤波、最小幅值滤波、像素校准、镜头校准等。

- 模拟设备时，采集 DCS 原始数据的相关参数再模拟设备中调节是无效的，例如曝光时间，HDR 功能等。



# Chapter 13

## Python 扩展

### 13.1 dmcam python 扩展概述

#### 13.1.1 python 扩展的安装

dmcam 提供了基于标准 python wheel 的扩展，该扩展可以通过 pip 直接进行安装，支持 Windows 和 Linux 的 32 位和 64 位环境，详见 [Pypi 项目主页](#)。安装命令为：

```
pip install -U dmcam
```

#### 13.1.2 python API 说明

Python 中的模组 API 和 C 库中 **dmcam.h** 中定义的 API 基本一一对应。

- python 扩展中默认的包名为： *dmcam* 。可通过 *import* 直接导入：

```
import dmcam
```

- API 命名映射关系 (C->Python): **dmcam\_xxxxxx(...)** -> **dmcam.xxxxxx(...)** 。例如 *dmcam\_dev\_open* 被映射为 *dmcam.dev\_open*
- 结构体映射关系 (C->Python): **dmcam\_xxxxxx -> dmcam.xxxx()** 。结构体被映射为 Python 中的一个类。例如通过如下方式创建一个 *dmcam\_frame\_info\_t* 结构体：

```
finfo = dmcam.frame_info_t()
```

- *NULL* 被映射为 *None* 。例如：

```
dmcam.init(None) # dmcam_init(NULL)
```

- 以下 Python 接口和 C 的 API 有差异，需要注意。

*dmcam.dev\_list()* Python 中对 C 接口进行了简化，可以直接通过下列方式获取设备列表。其返回值为 *dmcam.dev\_t()* 的列表。使用样例如下：

```
devs = dmcam.dev_list()
if devs is None:
    print(" No device found")
else:
    print("found %d device" % len(devs))
    print(" Device URIs:")
    for i, d in enumerate(devs):
        print("#%d: %s" % (i, dmcam.dev_get_uri(d, 256)[0]))
```

*dmcam.param\_batch\_set(dev, dict)* Python 中对 C 接口进行了简化，直接传递一个 ‘dict‘，而不必构造比较复杂的 *dmcam\_param\_item\_t* 结构体及其长度参数。使用样例如下：

```
wparams = {
    dmcam.PARAM_FRAME_RATE: dmcam.param_val_u(),
    dmcam.PARAM_INTG_TIME: dmcam.param_val_u(),
}
wparams[dmcam.PARAM_FRAME_RATE].frame_rate.fps = 15
wparams[dmcam.PARAM_INTG_TIME].intg.intg_us = 1000

if not dmcam.param_batch_set(dev, wparams):
    print(" set parameter failed")
```

*dmcam.param\_batch\_get(dev, list)* Python 中对 C 接口进行了简化，直接传递需要获取参数的 ‘list‘，而不必构造比较复杂的 *dmcam\_param\_item\_t* 结构体及其长度参数。使用样例如下：

```
# get intg from device
param_vals = dmcam.param_batch_get(dev, [dmcam.PARAM_INTG_TIME]) # type: ↴
                                                               ↴ list[dmcam.param_val_u]
param_intg_us = param_vals[0].intg.intg_us
```

*dmcam.set\_callback\_on\_frame\_ready* 和 *dmcam.set\_callback\_on\_error* 由于 Python 回调函数和 C 的差异。关于采集过程中回调函数的设置，目前，python 只支持通过上述两个接口分别设置 *frame\_ready* 和 *error* 两种类型的回调。不支持通过 *dmcam.cap\_config\_set(dev, cap\_cfg\_t)* 中的 *cap\_cg\_t* 进行回调函数的设置。使用样例如下：

```
def on_frame_rdy(dev, f):
    print("cap: idx=%d, num=%d" % (f.frame_fbpos, f.frame_count))
```

(continues on next page)

(continued from previous page)

```

def on_cap_err(dev, errnumber, errarg):
    print("caperr: %s" % dmcam.error_name(errnumber))

cap_cfg = dmcam.cap_cfg_t()
cap_cfg.cache_frames_cnt = 10 # frame buffer = 10 frames
cap_cfg.on_frame_ready = None # callback should be set by dmcam.cap_set_
    ↪callback_on_frame_ready
cap_cfg.on_cap_err = None # callback should be set by dmcam.cap_set_
    ↪callback_on_error
cap_cfg.en_save_dist_u16 = False # save dist into ONI file: which can be
    ↪viewed in openni
cap_cfg.en_save_gray_u16 = False # save gray into ONI file: which can be
    ↪viewed in openni
cap_cfg.en_save_replay = False # save raw into ONI file: which can be
    ↪simulated as DMCAM device
cap_cfg.fname_replay = os.fsencode("replay_dist.oni")

dmcam.cap_config_set(dev, cap_cfg)

dmcam.cap_set_callback_on_frame_ready(dev, on_frame_rdy)
dmcam.cap_set_callback_on_error(dev, on_cap_err)

```

下表列出了一些常用的 API 接口对比：

C 库核心函数	python 调用函数
dmcam_init	dmcam.init
dmcam_dev_list	dmcam.dev_list
dmcam_dev_open	dmcam.dev_open
dmcam_dev_close	dmcam.dev_close
dmcam_cap_config_set	dmcam.cap_config_set
dmcam_cap_set_callback_on_error	dmcam.cap_set_callback_on_error
dmcam_param_batch_set	dmcam.param_batch_set
dmcam_cap_get_frames	dmcam.cap_get_frames
dmcam_frame_get_distance	dmcam.frame_get_distance
dmcam_frame_get_gray	dmcam.frame_get_gray

## 13.2 python 中参数设置和滤波相关

这里介绍在 python 中相关参数的设置读取和滤波功能的开启关闭。

### 13.2.1 python 下参数设置和读取

- 单个参数设置，如设置帧格式：

```
wparams_fmt = {dmcam.PARAM_FRAME_FORMAT: dmcam.param_val_u()}
wparams_fmt[dmcam.PARAM_FRAME_FORMAT].frame_format.format = 2
if not dmcam.param_batch_set(dev, wparams_fmt):
    log.error(" frame format failed")
```

- 单个参数读取，如读取积分时间：

```
param_val = dmcam.param_batch_get(dev, [dmcam.PARAM_INTG_TIME])
param_intg_us = param_val.intg.intg_us
```

### 13.2.2 python 下滤波功能开启和关闭

- 像素校准，用于深度数据校正：

```
drnu_param = dmcam.filter_args_u()
drnu_param.case_idx = 0 # 12MHz calibaration
dmcam.filter_enable(dev, dmcam.DMCAM_FILTER_ID_PIXEL_CALIB, drnu_param, 0) # 使能像素校准

dmcam.filter_disable(dev, dmcam.DMCAM_FILTER_ID_PIXEL_CALIB) # 关闭像素校准
```

- 深度滤波，用于深度数据滤波：

```
filter_param = dmcam.filter_args_u()
filter_param.median_ksize = 3 # 深度滤波通常设置值
dmcam.filter_enable(dev, dmcam.DMCAM_FILTER_ID_MEDIAN, filter_param, sys.getsizeof(filter_param)) # 使能深度滤波

dmcam.filter_disable(dev, dmcam.DMCAM_FILTER_ID_MEDIAN) # 关闭深度滤波
```

- 幅值滤波，用于过滤质量差的点：

```
amp_min_val = dmcam.filter_args_u()
amp_min_val.min_amp = 30 # 设置最小幅值滤波的阈值
dmcam.filter_enable(dev, dmcam.DMCAM_FILTER_ID_AMP, amp_min_val, sys.getsizeof(amp_min_val)) # 使能幅值滤波

dmcam.filter_enable(dev, dmcam.DMCAM_FILTER_ID_AMP) # 关闭幅值滤波
```

- 自动积分时间，开启模组根据被测物自动调整曝光时间：

```

intg_auto_arg = dmcam.filter_args_u()
intg_auto_arg.sat_ration = 5      # 自动曝光设置的值
dmcam.filter_enable(dev, dmcam.DMCAM_FILTER_ID_AUTO_INTG, intg_auto_arg, sys.
    ↪getsizeof(intg_auto_arg))  # 开启自动曝光

dmcam.filter_disable(dev, DMCAM_FILTER_ID_AUTO_INTG)                      # 关闭自动曝光

```

- 运动模式 0，帧格式设置 2：

```

dmfilter = dmcam.filter_args_u()

wparams_fmt = {dmcam.PARAM_FRAME_FORMAT: dmcam.param_val_u()}
wparams_fmt[dmcam.PARAM_FRAME_FORMAT].frame_format.format = 2  # 要设置帧格式为 2
dmcam.param_batch_set(dev, wparams_fmt)                         # 进行帧格式参数设置

dmfilter.sport_mode = 0           # 设置运动模式为 0
dmcam.filter_enable(dev, dmcam.DMCAM_FILTER_ID_SPORT_MODE, dmfilter, 0)      # 开启运动模式 0

dmcam.filter_disable(dev, dmcam.DMCAM_FILTER_ID_SPORT_MODE)                 # 关闭运动模
式 0

```

- 运动模式 1，帧格式设置 4：

```

dmfilter = dmcam.filter_args_u()

wparams_fmt = {dmcam.PARAM_FRAME_FORMAT: dmcam.param_val_u()}
wparams_fmt[dmcam.PARAM_FRAME_FORMAT].frame_format.format = 4 # 要设置的帧格式为 4
dmcam.param_batch_set(dev, wparams_fmt)  # 进行帧格式设置

dmfilter.sport_mode = 1           # 设置运动模式为 1
dmcam.filter_enable(dev, dmcam.DMCAM_FILTER_ID_SPORT_MODE, dmfilter, 1)      # 开启运动模式 1

# 关闭运动模式 1 时，帧格式要先恢复到 2
wparams_fmt = {dmcam.PARAM_FRAME_FORMAT: dmcam.param_val_u()}
wparams_fmt[dmcam.PARAM_FRAME_FORMAT].frame_format.format = 2 # 恢复帧格式为 2
dmcam.param_batch_set(dev, wparams_fmt)  # 进行帧格式设置

dmcam.filter_disable(dev, dmcam.DMCAM_FILTER_ID_SPORT_MODE)                 # 关闭运动模
式 1

```

(continues on next page)

(continued from previous page)

# Chapter 14

## C# 扩展说明

### 14.1 dmcam C# 扩展概述

dmcam 提供的 C# 扩展可方便基于.net C# 的开发人员基于 SmartToF 相机进行快速的二次开发。本文档主要描述 C# 扩展的安装以及 API 和 C 的关联和区别。

#### 14.1.1 C# 扩展的安装

- Window
  - 支持的系统:
    - \* Windows 7/8/10 32bit/64bit
    - \* .Net framework >= 3.5
  - C# 扩展动态库包括:
    - \* dmcam\_csharp.dll: C# 扩展动态库, 可导入 C# 工程.
    - \* dmcam\_csharp\_adapter.dll: C# dmcam 的扩展适配器
    - \* libdmcam.dll: dmcam core lib
  - 安装方式
    - \* 将以上 dll 加入 PATH 路径或复制到执行文件目录。
    - \* C# 工程中引用 *dmcam\_csharp.dll*
- Linux
  - 支持的系统:
    - \* Linux 64bit (Ubuntu 14.04/16.04 tested)
    - \* Mono

- C# 扩展动态库包括:
  - \* dmcam\_csharp.dll: C# 扩展动态库, 可导入 C# 工程.
  - \* dmcam\_csharp\_adapter.so: C# dmcam 的扩展适配库
  - \* libdmcam.so: dmcam core lib
- 安装方式
  - \* 设置 `LD_LIBRARY_PATH` 包含上述 so/dll 目录, 或将动态库放入系统库目录下, 例如:  
`/usr/local/lib/`

### 14.1.2 C# API 说明

C# 中的模组 API 和 C 库中 `dmcam.h` 中定义的 API 基本一一对应。

- C# 扩展默认的名字空间为: `com.smarttof`。可通过 `using` 方式导入:

```
using com.smarttof;
namespace sampleBasic {
public class sampleBasic{
    public static void Main(string[] argv) {
        dmcam.init(null);
        /* ... */
        dmcam.uninit();
    }
}
```

- API 命名映射关系 (C->C#): `dmcam_xxxxxx(...)` -> `dmcam.xxxxxx(...)`。例如 `dmcam_dev_open` 被映射为 `dmcam.dev_open`
- 结构体映射关系 (C->C#): `dmcam_xxxxxx -> xxxx()`。结构体被映射为 C# 中的一个类。例如通过如下方式创建一个 `dmcam_frame_info_t` 结构体:

```
finfo = new frame_info_t()
```

- `NULL` 被映射为 `null`。例如:

```
dmcam.init(null) // dmcam_init(NULL)
```

- 以下 C# 接口和 C 的 API 有差异, 需要注意。

`dmcam.dev_list()` C# 中通过如下方式获取设备列表。其返回值为就绪设备数量。使用样例如下:

```
dmcamDevArray devs = new dmcamDevArray(16);
int cnt = dmcam.dev_list(devs.cast(), 16);
```

(continues on next page)

(continued from previous page)

```
Console.WriteLine("found {0} device\n", cnt);
```

*dmcam.param\_batch\_set()* C# 中设置参数相对 C 比较复杂一些, 需要构造 param\_item\_t 实例。  
具体使用样例如下:

```
param_item_t p_fps = new param_item_t();
p_fps.param_id = dev_param_e.PARAM_FRAME_RATE;
p_fps.param_val.frame_rate.fps = 15;

param_item_t p_intg = new param_item_t();
p_intg.param_id = dev_param_e.PARAM_INTG_TIME;
p_intg.param_val.intg.intg_us = 1000;

dmcamParamArray wparams = new dmcamParamArray(2);
wparams.setitem(0, p_fps);
wparams.setitem(1, p_intg);

if (!dmcam.param_batch_set(dev, wparams.cast(), 2)) {
    Console.WriteLine(" set param failed\n");
}
```

*dmcam.param\_batch\_get(dev, list)* C# 中设置参数相对 C 比较复杂一些, 需要构造 param\_item\_t 实例。具体使用样例如下:

```
param_item_t r_fps = new param_item_t();
r_fps.param_id = dev_param_e.PARAM_FRAME_RATE;
param_item_t r_intg = new param_item_t();
r_intg.param_id = dev_param_e.PARAM_INTG_TIME;

dmcamParamArray rparams = new dmcamParamArray(2);
rparams.setitem(0, r_fps);
rparams.setitem(1, r_intg);

if (!dmcam.param_batch_get(dev, rparams.cast(), 2)) {
    Console.WriteLine(" get param failed\n");
} else {
    Console.WriteLine("fps = {0}, intg = {1}",
        (int)rparams.getitem(0).param_val.frame_rate.fps,
        (int)rparams.getitem(1).param_val.intg.intg_us);
}
```

*dmcam.set\_callback\_on\_frame\_ready* 和 *dmcam.set\_callback\_on\_error* C# 扩展中不支持回调函数。采集时，可以参考如下设置：

```
cap_cfg_t cfg = new cap_cfg_t();
cfg.cache_frames_cnt = 10;
cfg.on_error= null;
cfg.on_frame_ready= null;
cfg.en_save_replay= 0;
cfg.en_save_dist_u16= 0;
cfg.en_save_gray_u16= 0;
cfg.fname_replay= null;

dmcam.cap_config_set(dev, cfg);
```

# Chapter 15

## Java 扩展说明

### 15.1 dmcam Java 扩展概述

dmcam 提供的 Java 扩展可方便基于 Java 的开发人员基于 SmartToF 相机进行快速的二次开发。本文档主要描述 Java 扩展的安装以及 API 和 C 的关联和区别。

#### 15.1.1 Java 扩展的安装

- Window 平台
  - 支持的系统:
    - \* Windows 7/8/10 32bit/64bit
    - \* JDK >= 1.8
  - Java 扩展动态库包括:
    - \* dmcam.jar: Java 扩展动态库, 可导入 Java 工程.
    - \* dmcam\_java.dll: Java dmcam 的扩展适配库
    - \* libdmcam.dll: dmcam core lib
  - 安装方式
    - \* 将以上 dll 加入 PATH 路径或复制到执行文件目录。
    - \* Java 工程中引用 *dmcam.jar*
- Linux
  - 支持的系统:
    - \* Linux 64bit (Ubuntu 14.04/16.04 tested)
    - \* Open JDK >= 7

- Java 扩展动态库包括:
  - \* dmcam.jar: Java 扩展动态库, 可导入 Java 工程.
  - \* libdmcam\_java.so: Java dmcam 的扩展适配库
  - \* libdmcam.so: dmcam core lib
- 安装方式
  - \* 设置 `LD_LIBRARY_PATH` 包含上述 so/dll 目录, 或将动态库放入系统库目录下, 例如:  
`/usr/local/lib/`

### 15.1.2 Java API 说明

Java 中的模组 API 和 C 库中 `dmcam.h` 中定义的 API 基本一一对应。

- Java 扩展默认的名字空间为: `com.smarttof.dmcam`。可通过 *import* 方式导入:

```
import com.smarttof.dmcam.*;

public class sampleBasic{
    public static void main(String[] args) {
        dmcamDevArray devs = new dmcamDevArray(16);
        int cnt = dmcam.dev_list(devs.cast(), 16);
    }
}
```

- API 命名映射关系 (C->Java): `dmcam_xxxxxx(...)` -> `dmcam.xxxxxx(...)`。例如 `dmcam_dev_open` 被映射为 `dmcam.dev_open`
- 结构体映射关系 (C->Java): `dmcam_xxxxxx -> xxxx()`。结构体被映射为 Java 中的一个类。例如通过如下方式创建一个 `dmcam_frame_info_t` 结构体:

```
finfo = new frame_info_t()
```

**Caution:** Java 扩展加载时候会自动调用 `dmcam.init(null)`, 释放的时候会自动调用 `dmcam.uninit()`。因此不必在程序中使用 `dmcam.init()` 和 `dmcam.uninit()`

- 访问结构体的成员变量: `obj.field -> obj.getField()/setField`。成员变量的访问需通过成员变量名对应的 *get/set* 方法进行。例如:

```
cap_cfg_t cfg = new cap_cfg_t();
cfg.setCache_frames_cnt(10); // cfg.cache_frame_cnt = 10
cfg.setOn_error(null); // cfg.on_error = NULL
/* ... */
```

- *NULL* 被映射为 *null*。例如:

```
dmcam.dev_open(null) // dmcam_dev_open(NULL)
```

- 以下 Java 接口和 C 的 API 有差异, 需要注意。

*dmcam.dev\_list()* Java 中通过如下方式获取设备列表。其返回值为就绪设备数量。使用样例如下:

```
dmcamDevArray devs = new dmcamDevArray(16);
int cnt = dmcam.dev_list(devs.cast(), 16);

System.out.printf("found %d device\n", cnt);
```

*dmcam.param\_batch\_set()* Java 中设置参数相对 C 比较复杂一些, 需要构造 param\_item\_t 实例。具体使用样例如下:

```
int pwr_percent = 100;
param_item_t wparam = new param_item_t();
dmcamParamArray wparams = new dmcamParamArray(1);

wparam.setParam_id(dev_param_e.PARAM_ILLUM_POWER);
wparam.getParam_val().getIllum_power().setPercent((short) pwr_percent);

wparams.setitem(0, wparam);
if (!dmcam.param_batch_set(dev, wparams.cast(), 1)) {
    System.out.printf(" set illu_power to %d %% failed\n", pwr_percent);
}
```

*dmcam.param\_batch\_get(dev, list)* Java 中获取参数相对 C 比较复杂一些, 需要构造 param\_item\_t 实例。具体使用样例如下:

```
param_item_t r_fps = new param_item_t();
r_fps.setParam_id(dev_param_e.PARAM_FRAME_RATE);
param_item_t r_intg = new param_item_t();
r_intg.setParam_id(dev_param_e.PARAM_INTG_TIME);

dmcamParamArray rparams = new dmcamParamArray(2);
rparams.setitem(0, r_fps);
rparams.setitem(1, r_intg);

if (dmcam.param_batch_get(dev, rparams.cast(), 2)) {
    System.out.printf("fps = %d, intg = %d",
        (int)rparams.getitem(0).getParam_val().getFrame_rate().getFps(),
```

(continues on next page)

(continued from previous page)

```
(int)rparams.getItem(1).getParam_val().getIntg().getIntg_us());  
}
```

*dmcam.set\_callback\_on\_frame\_ready* 和 *dmcam.set\_callback\_on\_error* Java 扩展中不支持回调函数。采集时，可以参考如下设置：

```
cap_cfg_t cfg = new cap_cfg_t();  
cfg.setCache_frames_cnt(10);  
cfg.setOn_error(null);  
cfg.setOn_frame_ready(null);  
cfg.setEn_save_replay((short)0);  
cfg.setEn_save_dist_u16((short)0);  
cfg.setEn_save_gray_u16((short)0);  
cfg.setFname_replay(null);  
  
dmcam.cap_config_set(dev, cfg);
```

# Chapter 16

## Android 扩展说明

### 16.1 dmcam Android 扩展概述

dmcam 提供的 Android 扩展可方便 Android 开发人员基于 SmartToF 相机进行快速的二次开发, 本文档主要描述 Android 扩展的使用以及 API 和 C 的关联和区别。

#### 16.1.1 Android 扩展的安装

- 支持的系统:
  - Android 4.4.2
- Android 扩展动态库包括 (ARM V5 和 ARM V7 架构)
  - libdmcam.so: dmcam core lib
  - libdmcam\_java.so: Java dmcam 的扩展适配库
  - dmcam\_android.jar: android 扩展动态库, 可导入 Android 工程
  - libusb1.0.so: usb 设备驱动库
- 安装方式
  - 将以上的所有 so 库加入到 Android 工程下的 libs
  - Android 工程下再加入 *dmcam.jar*

#### 16.1.2 Android API 说明

Android 的 app 开发是基于 Java 语言的, 所以 Android 中的主要 API 说明和 Java 中的 API 说明基本一致, 都和 C 库中的 **dmcam.h** 中定义的 API 基本一一对应。

- Android 中扩展的默认名字空间为: *com.smartttof.dmcam*。可通过 *import* 方式导入:

```

import com.smarttof.dmcam.*;

public boolean devSetFrameRate(dev_t dev, int fps){
    if (dev == null)
        return false;

    param_item_t wparam = new param_item_t();
    dmcamParamArray wparams = new dmcamParamArray(1);

    wparam.setParam_id(dev_param_e.PARAM_FRAME_RATE);
    wparam.getParam_val().getFrame_rate().setFps(fps);

    wparams.setitem(0, wparam);
    logUI("DMCAM",
          String.format("set fps = %d\n", wparams.getitem(0)
                        .getParam_val().getFrame_rate().getFps()));
    if (!dmcam.param_batch_set(dev, wparams.cast(), 1)) {
        logUI("DMCAM", String.format(" set fps to %d failed\n", fps));
        return false;
    }
    return true;
}

```

- Android 中的调用的 API 和 Java 中调用的 API 基本相同，映射关系都如: `dmcam_xxxxxx(...)` -> `dmcam.xxxxxx(...)`。例如 `dmcam_dev_close` 被映射为 `dmcam.dev_close`
- 结构体映射关系: `dmcam_xxxxxx -> xxxxxx()`。结构体被映射为 java 中的一个类。例如通过如下方式创建一个 `dmcam_param_item_t` 结构体:

```
wparam = new param_item_t();
```

**Caution:** 在 Android 中打开设备时调用的 API 为 `dmcam.dev_open_by_fd`, 不是其他环境下的 `dmcam.dev_open`

- 访问结构体的成员变量: `obj.field -> obj.getField()/setField`。成员变量的访问需通过成员变量名对应的 `get/set` 方法进行。例如:

```

cap_cfg_t cfg = new cap_cfg_t();
cfg.setCache_frames_cnt(10); // cfg.cache_frame_cnt = 10
cfg.setOn_error(null); // cfg.on_error = NULL

```

(continues on next page)

(continued from previous page)

```
/* ... */
```

- *NULL* 被映射为 *null*。例如:

```
cfg.setOn_error(null);
```

- 以下是在 Android 中调用 API 与 C 中的一些区别, 需要注意。

*dmcam.dev\_open\_by\_fd()*

在 Android 下打开设备, 需要调用专为 Android 设计的 API *dmcam.dev\_open\_by\_fd()*, 具体使用样例如下:

```
UsbDeviceConnection connection = usbManager.openDevice(usbDevice);
fd = connection.getFileDescriptor();
dev = dmcam.dev_open_by_fd(fd);
```

*dmcam.param\_batch\_set()*

需要构造 *param\_item\_t* 实例, 具体样例如下:

```
param_item_t wparam = new param_item_t();
dmcamParamArray wparams = new dmcamParamArray(1);

wparam.setParam_id(dev_param_e.PARAM_INTG_TIME);
wparam.getParam_val().getIntg().setIntg_us(expoUs);

wparams.setitem(0, wparam);
if (!dmcam.param_batch_set(dev, wparams.cast(), 1)) {
    logUI("DMCAM",
          String.format(" set exposure to %d us failed\n"
            , expoUs));
    return false;
}
```

*dmcam.param\_batch\_get()*

获取参数也要构造实例, 具体样例如下:

```
param_item_t r_fps = new param_item_t();
r_fps.setParam_id(dev_param_e.PARAM_FRAME_RATE);

dmcamParamArray rparam = new dmcamParamArray(1);
rparam.setitem(0, r_fps);
```

(continues on next page)

(continued from previous page)

```
if (dmcam.param_batch_get(dev, rparam.cast(), 1)) {
    logUI("DMCAM",
          String.format(" get frame_rate %d fps\n", (int)rparam.
←getitem(0).getParam_val().getFrame_rate().getFps()));
}
```

*dmcam.set\_callback\_on\_frame\_ready* 和 *dmcam.set\_callback\_on\_error* Android 扩展中不支持回调函数。采集时，可以参考如下设置：

```
cap_cfg_t cfg = new cap_cfg_t();
cfg.setCache_frames_cnt(10);
cfg.setOn_error(null);
cfg.setOn_frame_ready(null);
cfg.setEn_save_replay((short)0);
cfg.setEn_save_dist_u16((short)0);
cfg.setEn_save_gray_u16((short)0);
cfg.setFname_replay(null);

dmcam.cap_config_set(dev, cfg);
```

# Chapter 17

## ROS 扩展

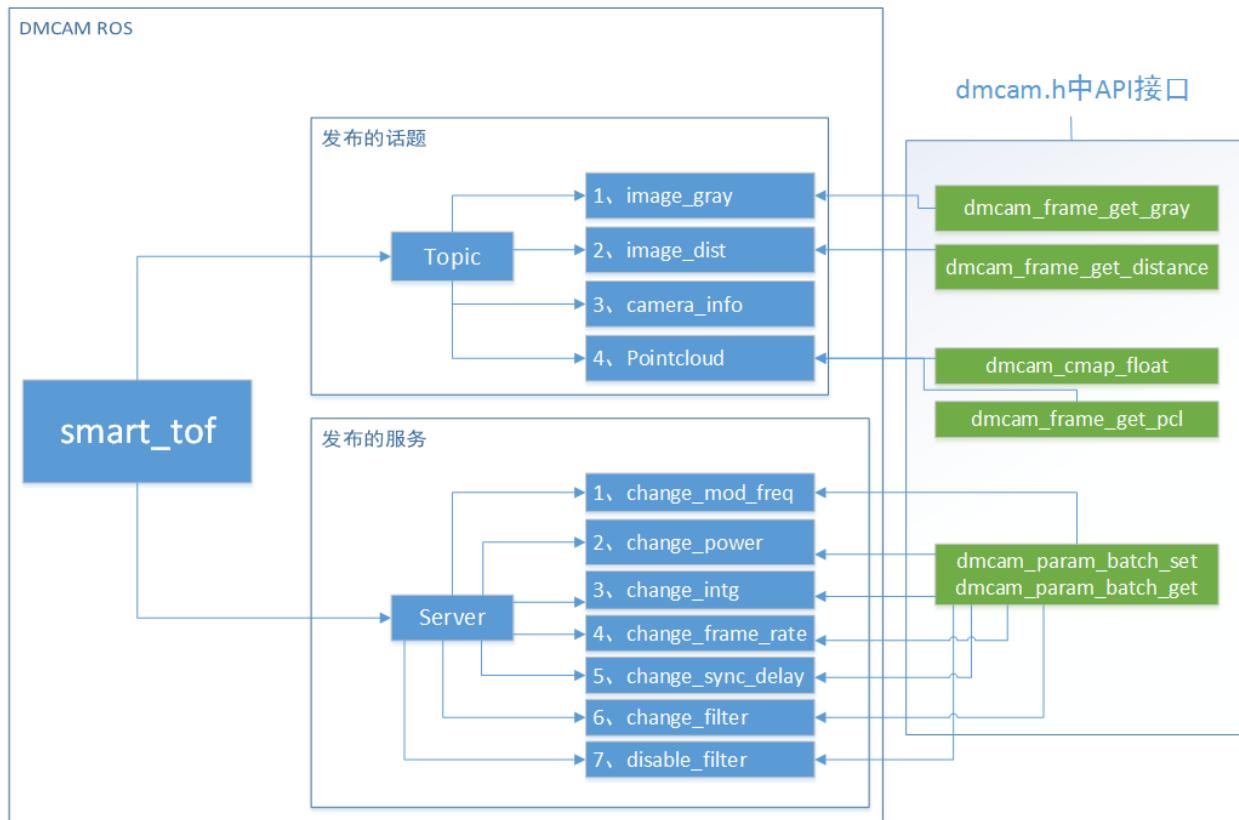
### 17.1 ROS 设计介绍

#### 17.1.1 ROS 概述

SDK ROS 中提供的功能是在 ROS 的基础上对 Dmcam API 的封装，当我们调用 ROS API 的时候，会相应的通过 dmcam 的 api 来与模组进行交互。

#### 17.1.2 ROS 框架

ROS 中的 API 与 dmcam 中的 API 对应关系图如下：



## 17.2 ROS API 说明

### 17.2.1 dmcam\_ros 发布的话题

1. `/smarttotof/image_dist`

使用命令	<code>rosrun image_view image_view image:=/smarttotof/image_dist</code>
功能描述	从 <code>image_dist</code> 发布的话题中获取深度数据

2. `/smarttotof/image_gray`

使用命令	<code>rosrun image_view image_view image:=/smarttotof/image_gray</code>
功能描述	从 <code>image_gray</code> 发布的话题中获取灰度数据

3. `/smarttotof/camera_info`

使用命令	<code>rosrun image_view image_view image:=/smarttotof/camera_info</code>
功能描述	从 <code>camera_info</code> 发布的话题中打印摄像头的信息

4. /smarttof/pointcloud

使用命令	<code>rosrun image_view image_view image:=/smarttof/pointcloud</code>
功能描述	从 rviz 中显示通过 pointcloud 发布的话题中的点云数据

### 17.2.2 dmcam\_ros 发布的服务

1. /smarttof/change\_power

使用命令	<code>rosservice call /smarttof/change_power "power_value:&lt;value&gt;"</code>
功能描述	动态修改 PARAM_ILLUM_POWER 的值
函数参数	value 值默认为 0

2. /smarttof/change\_intg

使 用 命 令	<code>rosservice call /smarttof/change_intg "intg_value:&lt;value&gt;"</code>
功 能 描 述	动态修改 PARAM_INTG_TIME 的值, PARAM_INTG_TIME 为积分时间
函 数 参 数	积分时间的 value 值范围为 0-1500

3. /smarttof/change\_mod\_freq

使 用 命 令	<code>rosservice call /smarttof/ change_mod_freq "mod_freq_value:&lt;value&gt;"</code>
功 能 描 述	动态修改 PARAM_MOD_FREQ 的值, PARAM_MOD_FREQ 为时钟频率
函 数 参 数	value 值目前固定为 12MHz

4. /smarttof/change\_frame\_rate

使 用 命 令	<code>rosservice call /smarttof/ change_frame_rate "frame_rate_value:&lt;value&gt;"</code>
功 能 描 述	动态修改 PARAM_FRAME_RATE 的值, PARAM_FRAME_RATE 为帧率
函 数 参 数	vallue 的范围为 10-30

5. `/smarttof/change_sync_delay`

使 用 命 令	<code>rosservice call /smarttof/ change_sync_delay "sync_delay_value:&lt;value&gt;"</code>
功 能 描 述	动态修改 PARAM_SYNC_DELAY 的值, PARAM_SYNC_DELAY 为同步延时时间
函 数 参 数	value 值 0 为自动, 1-10 为指定范围

6. `/smarttof/change_filter`

使 用 命 令	<code>rosservice call /smarttof/change_filter "filter_id: '&lt;id&gt;' filter_value:&lt;value&gt;"</code>
功 能 描 述	打开 filter_id 中指定 id 值的滤波功能
函 数 参 数	<p>filter_id 中的 id 值可以设置为</p> <ul style="list-style-type: none"> <li>DMCAM_FILTER_ID_LEN_CALIB //镜头校准</li> <li>DMCAM_FILTER_ID_PIXEL_CALIB //像素校准</li> <li>DMCAM_FILTER_ID_RESERVED //暂不支持</li> <li>DMCAM_FILTER_ID_AMP //幅值滤波器</li> <li>DMCAM_FILTER_ID_AUTO_INTG //积分时间</li> <li>DMCAM_FILTER_ID_SYNC_DELAY //暂不支持</li> <li>DMCAM_FILTER_ID_TEMP_MONITOR //暂不支持</li> <li>DMCAM_FILTER_ID_HDR //HDR 模式</li> <li>DMCAM_FILTER_ID_OFFSET //距离偏移</li> <li>DMCAM_FILTER_ID_SPORT_MODE //运动模式</li> <li>DMCAM_FILTER_ID_SYS_CALIB //暂不支持</li> <li>DMCAM_FILTER_ID_AMBIENT_LIGHT_CALIB //暂不支持</li> </ul> <p>目前仅 DMCAM_FILTER_ID_AMP 中需要设置 filter_value 的 value 值, 范围为 0-100</p> <p>其它 filter_value 中的 value 值默认为 0 即可</p>

## 7. /smarttof/disable\_filter

使用命令	<code>rosservice call /smarttof/disable_filter "filter_id: '&lt;id&gt;'"</code>
功能描述	关闭 filter_id 中 id 值的滤波功能
函数参数	filter_id 中的 id 值可以设置为
	DMCAM_FILTER_ID_LEN_CALIB //镜头校准
	DMCAM_FILTER_ID_PIXEL_CALIB //像素校准
	DMCAM_FILTER_ID_RESERVED //暂不支持
	DMCAM_FILTER_ID_AMP //幅值滤波器
	DMCAM_FILTER_ID_AUTO_INTG //积分时间
	DMCAM_FILTER_ID_SYNC_DELAY //暂不支持
	DMCAM_FILTER_ID_TEMP_MONITOR //暂不支持
	DMCAM_FILTER_ID_HDR //HDR 模式
	DMCAM_FILTER_ID_OFFSET //距离偏移
	DMCAM_FILTER_ID_SPORT_MODE //运动模式
	DMCAM_FILTER_ID_SYS_CALIB //暂不支持
	DMCAM_FILTER_ID_AMBIENT_LIGHT_CALIB //暂不支持



# Chapter 18

## OpenNI2 扩展

### 18.1 OpenNI2 驱动说明

为了支持 OpenNI2 中的框架实现对 SmartToF 模组的调用，SDK 中提供了相应支持的驱动库 samrtttof.dll，该驱动依循 OpenNI2 在 OniDriverAPI.h 中的定义，主要包括了 DriverService、DriverBase、DeviceBase、StreamBase 这几个类别，smartttof.dll 的驱动源码在 SDK 中一同发布。

#### 18.1.1 SmartToF 模组的设置

SmartToF 中的所有相关的参数设置和滤波功能的使用都通过 smarttofStream 类中的 setProperty 函数执行，根据 setProperty 中的 propertyId 进行对应得设置。

下表列出了 property 的属性 ID：

功能 ID	说明
PROPERTY_ID_PARAM_SET	模组参数设置 ID
PROPERTY_ID_PARAM_GET	模组参数获取 ID
PROPERTY_ID_FILTER_LEN_CALIB_ENABLE	镜头滤波使能
PROPERTY_ID_FILTER_LEN_CALIB_DISABLE	镜头滤波关闭
PROPERTY_ID_FILTER_PIXEL_CALIB_ENABLE	像素滤波使能
PROPERTY_ID_FILTER_PIXEL_CALIB_DISABLE	像素滤波关闭
PROPERTY_ID_FILTER_AMP_CALIB_ENABLE	幅值滤波使能
PROPERTY_ID_FILTER_AMP_CALIB_DISABLE	幅值滤波关闭
PROPERTY_ID_FILTER_AUTO_INTG_ENABLE	自动曝光使能
PROPERTY_ID_FILTER_AUTO_INTG_DISABLE	自动曝光关闭
PROPERTY_ID_FILTER_TEMP_MONITOR_ENABLE	温度监控使能
PROPERTY_ID_FILTER_TEMP_MONITOR_DISABLE	温度监控关闭
PROPERTY_ID_FILTER_HDR_ENABLE	HDR 功能使能
PROPERTY_ID_FILTER_HDR_DISABLE	HDR 功能关闭

## OpenNI2 下模组设置示例代码

OpenNI2 下可以通过 `setProperty` 将以上列表中的属性 ID 设置到 SmartToF 模组中去，具体设置的代码示例可以参考下面的设置积分时间代码：

```
dmcam_param_item_t wparam; //setting integration time
wparam.param_id = PARAM_INTG_TIME;
wparam.param_val.intg.intg_us = intg;
wparam.param_val_len = sizeof(wparam.param_val.intg.intg_us);
depth.setProperty(PROPERTY_ID_PARAM_SET, (void *)&wparam, sizeof(wparam));
```

获取模组设置的代码如下：

```
dmcam_param_item_t rparam; //getting framerate
rparam.param_id = PARAM_FRAME_RATE;
rparam.param_val_len = sizeof(rparam.param_val.frame_rate.fps);
depth.getProperty(PROPERTY_ID_PARAM_GET, &rparam);
printf("frame rate:%d fps\n", rparam.param_val.frame_rate.fps);
```

# Chapter 19

## Indices and tables

- genindex
- modindex
- search



# Index

## D

`dmcam_cap_config_get (C function)`, 77  
`dmcam_cap_config_set (C function)`, 77  
`dmcam_cap_get_frame (C function)`, 79  
`dmcam_cap_get_frames (C function)`, 79  
`dmcam_cap_is_ongoing (C function)`, 78  
`dmcam_cap_set_callback_on_error (C function)`,  
    78  
`dmcam_cap_set_callback_on_frame_ready (C`  
    *function*), 78  
`dmcam_cap_set_frame_buffer (C function)`, 77  
`dmcam_cap_snapshot (C function)`, 78  
`dmcam_cap_start (C function)`, 78  
`dmcam_cap_stop (C function)`, 79  
`dmcam_cmap_dist_f32_to_RGB (C function)`, 83  
`dmcam_cmap_dist_u16_to_RGB (C function)`, 84  
`dmcam_cmap_float (C function)`, 84  
`dmcam_cmap_gray_f32_to_IR (C function)`, 84  
`dmcam_cmap_gray_u16_to_IR (C function)`, 84  
`dmcam_data_download (C function)`, 80  
`dmcam_data_upload (C function)`, 80  
`dmcam_dev_close (C function)`, 75  
`dmcam_dev_get_uri (C function)`, 75  
`dmcam_dev_list (C function)`, 74  
`dmcam_dev_open (C function)`, 74  
`dmcam_dev_open_by_fd (C function)`, 74  
`dmcam_dev_open_by_uri (C function)`, 75  
`dmcam_dev_reset (C function)`, 75  
`dmcam_error_name (C function)`, 74  
`dmcam_file_close (C function)`, 85  
`dmcam_file_open (C function)`, 85  
`dmcam_filter_disable (C function)`, 83  
`dmcam_filter_enable (C function)`, 83  
`dmcam_firmware_upgrade (C function)`, 79  
`dmcam_frame_get_dist_f32 (C function)`, 80  
`dmcam_frame_get_dist_u16 (C function)`, 80  
`dmcam_frame_get_distance (C function)`, 80  
`dmcam_frame_get_gray (C function)`, 81  
`dmcam_frame_get_gray_f32 (C function)`, 81  
`dmcam_frame_get_gray_u16 (C function)`, 81  
`dmcam_frame_get_pcl (C function)`, 81  
`dmcam_frame_get_pcl_xyzd (C function)`, 82  
`dmcam_frame_load_distance (C function)`, 87  
`dmcam_frame_load_gray (C function)`, 87  
`dmcam_frame_load_raw (C function)`, 86  
`dmcam_frame_save_distance (C function)`, 86  
`dmcam_frame_save_gray (C function)`, 86  
`dmcam_frame_save_raw (C function)`, 85  
`dmcam_init (C function)`, 73  
`dmcam_log_cfg (C function)`, 73  
`dmcam_param_batch_get (C function)`, 77  
`dmcam_param_batch_set (C function)`, 76  
`dmcam_path_cfg (C function)`, 74  
`dmcam_path_get (C function)`, 74  
`dmcam_reg_batch_read (C function)`, 76  
`dmcam_reg_batch_write (C function)`, 75  
`dmcam_uninit (C function)`, 73