英特尔D435i深度相机和librealsense安装使用

分类: 机器视觉 个人专栏: 视觉感知 发布时间 2022.07.07 阅读数 2425 评论数 0

介绍

这是一款比较好用的相机。
D435i与D435多一个IMU，官方介绍见这里

在这张图片的相机中，从左到右有四个模组，其中，第一个和第三个为一组双目红外摄像头，这款相机通过双目立体视觉来获取深度，并且都是是红外摄像头而非RGBD，在光线比较暗时也有比较好的效果；第二个模组是一个红外发射器，可以理解为一个补光灯；最右侧为一个RGB相机。
关于相机的主要信息有：深度有效范围0.3-3m；双目全局快门，深度帧率90帧/秒，深度图像大小可到1280_720；RGB卷帘快门，帧率30帧/秒图像大小可到1920_1080；以及视场FOV信息。

特性	**使用环境** 室内/室外 **图像传感器技术:** 全局快门	**最大范围:** .3-3 米
深度	**深度技术:** 立体 **最小深度距离 (Min-Z):** ~0.28 米 **深度精度** <2% 位于2 米[1]	**深度视场 (FOV):** 87° × 58° **深度输出分辨率:** 高达 1280 x 720 **深度帧率:** 高达 90 帧/秒
RGB	**RGB帧分辨率:** 1920 × 1080 **RGB 帧率:** 30 帧/秒 **RGB 传感器技术:** 卷帘快门	**RGB 传感器 FOV (H × V):** 69° × 42° **RGB传感器分辨率:** 2 MP

其最大的好处就是，内参获取容易且内参精度比较高（通过程序接口读取到内参），数据流之间的对齐轻松（调用程序接口可以直接实现）。

sdk下载安装

参考sdk2介绍和说明文档和GitHub地址
我的系统是ubuntu18.04
首先安装依赖：

```
sudo apt-get install git libssl-dev libusb-1.0-0-dev pkg-config libgtk-3-dev
sudo apt-get install libglfw3-dev libgl1-mesa-dev libglu1-mesa-dev
```

下载librealsense，进入其源码目录下
udev规则设置：

```
sudo cp config/99-realsense-libusb.rules /etc/udev/rules.d/
sudo udevadm control --reload-rules
udevadm trigger
```

构建并应用修补的内核模块：

```
./scripts/patch-realsense-ubuntu-lts.sh
```

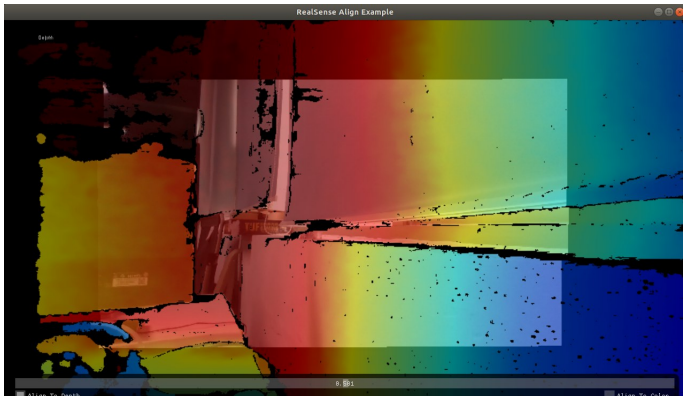
编译：

```
mkdir build
cd build
cmake ..
make -j8
sudo make install -j8
```

编译成功后在/build/examples文件夹下可以看到编译好的例程
运行前需要检查相机的数据线接到电脑的USB3.0接口
运行前需要检查相机的数据线接到电脑的USB3.0接口
运行前需要检查相机的数据线接到电脑的USB3.0接口

```
./build/examples/align/rs-align
```

效果如下：



在安装成功后，若要调用librealsense接口，CMakeList.txt内容参考examples/cmake/内的CMakeList.txt文件：

```
# License: Apache 2.0. See LICENSE file in root directory.
# Copyright(c) 2019 Intel Corporation. All Rights Reserved.
cmake_minimum_required(VERSION 3.1.0)

project(hello_librealsense2)

# Find librealsense2 installed package
find_package(realsense2 REQUIRED)

# Enable C++11
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED TRUE)

# Add the application sources to the target
add_executable(${PROJECT_NAME} hello_librealsense2.cpp)

# Link librealsense2 to the target
target_link_libraries(${PROJECT_NAME} ${realsense2_LIBRARY})
```

使用

这部分内容多参考示例说明，以及API文档，示例程序对应librealsense源码目录下的examples/内容。
源码目录下的examples/sensor-control/api_how_to.h封装了一些实用的例子：

Code Overview

Please follow the `api_how_to.h` for detailed documentation and explanations
This file contains detailed explanations and code on how to do the following:

- get a realsense device
- print device information
- get device name
- get sensor name
- get a sensor from a device
- get sensor option
- get depth units
- get field of view
- get extrinsics
- change sensor option
- choose a streaming profile
- start streaming a profile

借鉴我所看到的一些代码和官方的例子，我自己也做部分使用总结。

获取设备信息

```
rs2::device get_a_realsense_device()
{
    // 实例化context
    rs2::context ctx;

    // context的query_devices方法获取设备列表，返回一个device_list对象
    rs2::device_list devices = ctx.query_devices();

    //实例化device
    rs2::device selected_device;
    if (devices.size() == 0)
    {
        std::cerr << "No device connected!!!!!" << std::endl;
        exit(0);
    }
    else
    {
        //打印设备数量
        std::cout<<"device num: "<<devices.size()<<std::endl;
        //将第一个设备传给device实例化得到的selected_device
        selected_device = devices[0];
    }
    //至此，selected_device就代表我们的当前设备，将它返回
    return selected_device;
}
```

读取设备参数

在修改选项之前需要好好看看这个函数打印出来的信息。

```
void  get_sensor_option(const rs2::sensor& sensor)
{
    std::cout << "Sensor name: " << sensor.get_info(RS2_CAMERA_INFO_NAME) <<
std::endl;
    std::cout << "Sensor supports the following options:\n" << std::endl;
    // 轮循所有参数
    for (int i = 0; i < static_cast<int>(RS2_OPTION_COUNT); i++)
    {
        // 隐式转换，将数字转为枚举类型
        rs2_option option_type = static_cast<rs2_option>(i);

        // 首先判断传感器是否支持这个选项设置
        if (sensor.supports(option_type))
        {
            // 获取并打印对当前选项的描述
            const char* description = sensor.get_option_description(option_type);

            // 获取并打印当前选项的值
            float current_value = sensor.get_option(option_type);

            // 获取选项的取值范围、默认值、取值步长
            rs2::option_range range = sensor.get_option_range(option_type);
            float default_value = range.def;
```

```
            float maximum_supported_value = range.max;
            float minimum_supported_value = range.min;
            float difference_to_next_value = range.step;

            // 打印
            std::cout << "    " << i << ": "  << "<" << option_type  << ">" << std::endl;
            std::cout << "        Description   : " << description << std::endl;
            std::cout << "        Min Value     : " << minimum_supported_value << std::endl;
            std::cout << "        Max Value     : " << maximum_supported_value << std::endl;
            std::cout << "        Step          : " << difference_to_next_value << std::endl;
            std::cout << "        Default Value : " << default_value << std::endl;
            std::cout << "        Current Value : " << current_value << std::endl;
            std::cout << std::endl;
        }
        else
        {
            //std::cout << "        is not supported by this sensor" << std::endl;
        }
    }
    std::cout << std::endl;
    std::cout << std::endl;
}
```

这里注意，需要将rs2::device实例化得到的selected_device传递到rs2::sensor，但一个设备一般有好几个不同类型的传感器，比如D435i有Stereo Module、RGB Camera、Motion Module三种类型的传感器，所以函数的使用体现为

```
std::vector<rs2::sensor> sensors = selected_device.query_sensors();
for (rs2::sensor sensor : sensors){
    get_sensor_option(sensor);
}
```

关于详细参数列表参考rs_option.h：

```
/** \brief Defines general configuration controls.
These can generally be mapped to camera UVC controls, and can be set /
*/
typedef enum rs2_option
{
    RS2_OPTION_BACKLIGHT_COMPENSATION, /**< Enable / disable color bac
    RS2_OPTION_BRIGHTNESS, /**< Color image brightness*/
    RS2_OPTION_CONTRAST, /**< Color image contrast*/
    RS2_OPTION_EXPOSURE, /**< Controls exposure time of color camera.
    RS2_OPTION_GAIN, /**< Color image gain*/
    RS2_OPTION_GAMMA, /**< Color image gamma setting*/
    RS2_OPTION_HUE, /**< Color image hue*/
    RS2_OPTION_SATURATION, /**< Color image saturation setting*/
    RS2_OPTION_SHARPNESS, /**< Color image sharpness setting*/
    RS2_OPTION_WHITE_BALANCE, /**< Controls white balance of color ima
    RS2_OPTION_ENABLE_AUTO_EXPOSURE, /**< Enable / disable color image
    RS2_OPTION_ENABLE_AUTO_WHITE_BALANCE, /**< Enable / disable color
    RS2_OPTION_VISUAL_PRESET, /**< Provide access to several recommend
    RS2_OPTION_LASER_POWER, /**< Power of the laser emitter, with 0 me
    RS2_OPTION_ACCURACY, /**< Set the number of patterns projected per
    RS2_OPTION_MOTION_RANGE, /**< Motion vs. Range trade-off, with low
    RS2_OPTION_FILTER_OPTION, /**< Set the filter to apply to each dep
    RS2_OPTION_CONFIDENCE_THRESHOLD, /**< The confidence level thresho
    RS2_OPTION_EMITTER_ENABLED, /**< Emitter select: 0 – disable all e
    RS2_OPTION_FRAMES_QUEUE_SIZE, /**< Number of frames the user is al
    RS2_OPTION_TOTAL_FRAME_DROPS, /**< Total number of detected frame
    RS2_OPTION_AUTO_EXPOSURE_MODE, /**< Auto-Exposure modes: Static, A
    RS2_OPTION_POWER_LINE_FREQUENCY, /**< Power Line Frequency control
    RS2_OPTION_ASIC_TEMPERATURE, /**< Current Asic Temperature */
    RS2_OPTION_ERROR_POLLING_ENABLED, /**< disable error handling */
    RS2_OPTION_PROJECTOR_TEMPERATURE, /**< Current Projector Temperatu
    RS2_OPTION_OUTPUT_TRIGGER_ENABLED, /**< Enable / disable trigger t
    RS2_OPTION_MOTION_MODULE_TEMPERATURE, /**< Current Motion-Module T
    RS2_OPTION_DEPTH_UNITS, /**< Number of meters represented by a sin
    RS2_OPTION_ENABLE_MOTION_CORRECTION, /**< Enable/Disable automatic
    RS2_OPTION_AUTO_EXPOSURE_PRIORITY, /**< Allows sensor to dynamical
    RS2_OPTION_COLOR_SCHEME, /**< Color scheme for data visualization
    RS2_OPTION_HISTOGRAM_EQUALIZATION_ENABLED, /**< Perform histogram
    RS2_OPTION_MIN_DISTANCE, /**< Minimal distance to the target */
    RS2_OPTION_MAX_DISTANCE, /**< Maximum distance to the target */
    RS2_OPTION_TEXTURE_SOURCE, /**< Texture mapping stream unique ID *
    RS2_OPTION_FILTER_MAGNITUDE, /**< The 2D-filter effect. The specif
    RS2_OPTION_FILTER_SMOOTH_ALPHA, /**< 2D-filter parameter controls
    RS2_OPTION_FILTER_SMOOTH_DELTA, /**< 2D-filter range/validity thre
    RS2_OPTION_HOLES_FILL, /**< Enhance depth data post-processing wit
    RS2_OPTION_STEREO_BASELINE, /**< The distance in mm between the fi
    RS2_OPTION_AUTO_EXPOSURE_CONVERGE_STEP, /**< Allows dynamically ad
```

更改参数

```
void change_sensor_option(const rs2::sensor& sensor, rs2_option option_type, float requested_value)
{
    // 首先判断传感器是否支持这个选项设置
    if (!sensor.supports(option_type))
    {
        std::cerr << "option is not supported by sensor" << "<"  << sensor.get_info(RS2_CAMERA_INFO_NAME) << ">" <<std::endl;
        return;
    }
    else
    {

        // 使用set_option函数给选项赋新值
        sensor.set_option(option_type, requested_value);
        std::cout << "Change " << "<" << option_type  << ">" << " to " << ": " << requested_value << std::endl;
    }

}
```

与读取设备参数的操作一样，这里要注意一下用法，需要轮询前面得到的sensor：sensors，对不同类型的传感器进行各自的设置。

```
for (rs2::sensor sensor : sensors){
    ++index;
    if (index == 1) {// Stereo Module
        change_sensor_option(sensor, RS2_OPTION_ENABLE_AUTO_EXPOSURE, 1);
        change_sensor_option(sensor, RS2_OPTION_AUTO_EXPOSURE_LIMIT,50000);
        change_sensor_option(sensor, RS2_OPTION_EMITTER_ENABLED, 1);
    }
    if (index == 2){// RGB Camera
```

```
        change_sensor_option(sensor, RS2_OPTION_EXPOSURE,80.f);
    }
    if (index == 3){// Motion Module
        change_sensor_option(sensor, RS2_OPTION_ENABLE_MOTION_CORRECTION,0);
    }
    get_sensor_option(sensor);
}
```

获取深度信息的尺度

像素值乘以这个尺度因子就是深度信息。

```
float get_depth_units(const rs2::sensor& sensor)
{
    if (rs2::depth_sensor dpt_sensor = sensor.as<rs2::depth_sensor>())
    {
        float scale = dpt_sensor.get_depth_scale();
        std::cout << "Scale factor for depth sensor is: " << scale << std::endl;
        return scale;
    }
    else
        std::cout << "Given sensor is not a depth sensor" << scale << std::endl;
}
```

同样需要注意sensor类型，在D425i上应该归属Stereo Module这个传感器类型，不过这个scale的值一般都是0.001，不读也行。

获取图像

获取图像，包括RGB图、深度图、IR图，并转为opencv的格式。

```
    // 首先实例化pipeline和config
    rs2::pipeline pipe;
    rs2::config cfg;

    // RGB stream
    cfg.enable_stream(RS2_STREAM_COLOR, 640, 480, RS2_FORMAT_BGR8, 30);

    // Depth stream
    cfg.enable_stream(RS2_STREAM_DEPTH, 640, 480, RS2_FORMAT_Z16, 30);

    // IMU stream
    cfg.enable_stream(RS2_STREAM_ACCEL, RS2_FORMAT_MOTION_XYZ32F, 250);
    cfg.enable_stream(RS2_STREAM_GYRO , RS2_FORMAT_MOTION_XYZ32F, 400);

    // IR stream
    cfg.enable_stream(RS2_STREAM_INFRARED, 1, 640, 480, RS2_FORMAT_Y8, 30);
    cfg.enable_stream(RS2_STREAM_INFRARED, 2, 640, 480, RS2_FORMAT_Y8, 30);

    // 开启
    rs2::pipeline_profile profile = pipe.start(cfg);

    while(1){
        // 等待一帧数据流
        rs2::frameset frames = pipe.wait_for_frames();

        // 取得每一张影像
        rs2::depth_frame depth_frame = frames.get_depth_frame(); // 获取深度图像数据
        rs2::video_frame color_frame = frames.get_color_frame(); // 获取彩色图像数据

        rs2::frame irL_frame = frames.get_infrared_frame(1); // 左红外相机图像
        rs2::frame irR_frame = frames.get_infrared_frame(2); // 右红外相机图像

        // 转Opencv
        Mat color_image(Size(640, 480), CV_8UC3,
(void*)color_frame.get_data(),Mat::AUTO_STEP); // 彩色图像
        Mat depth_image(Size(640, 480), CV_16U ,
(void*)depth_frame.get_data(),Mat::AUTO_STEP); // 深度图像
        Mat irL_image  (Size(640, 480), CV_8UC1, (void*)irL_frame.get_data()
,Mat::AUTO_STEP); // 左红外相机图像
        Mat irR_image  (Size(640, 480), CV_8UC1, (void*)irR_frame.get_data()
,Mat::AUTO_STEP); // 右红外相机图像
        imshow("color", color_image);
        imshow("depth", depth_image);
        imshow("irL", irL_image);
        imshow("irR", irR_image);
        waitKey(1);
    }
```

获取imu数据

这里要注意，D435i的IMU只输出三轴加速度与三轴角速度数据，不会像T265一样输出姿态。
程序内容其实与获取图像一样，但是SDK没有给封装为完整函数，在这里会稍微加几句话。

```
// Get gyro measures
rs2::frame f_gyro = frames.first_or_default(RS2_STREAM_GYRO);
rs2_vector gyro_data = f_gyro.as<rs2::motion_frame>().get_motion_data();
std::cout << gyro_data << std::endl;

// Get accelerometer measures
rs2::frame f_accel = frames.first_or_default(RS2_STREAM_ACCEL);
rs2_vector accel_data = f_accel.as<rs2::motion_frame>().get_motion_data();
std::cout << accel_data << std::endl;
```

对比获取图像的函数封装：

```
/**
* Retrieve the first color frame, if no frame is found, search for the co
* \return video_frame - first found color frame.
*/
video_frame get_color_frame() const
{
    auto f = first_or_default(RS2_STREAM_COLOR);

    if (!f)
    {
        auto ir = first_or_default(RS2_STREAM_INFRARED);
        if (ir && ir.get_profile().format() == RS2_FORMAT_RGB8)
```

```
            f = ir;
        }
        return f;
    }
```

通过回调函数获取图像以及IMU数据

以上是在while(1)内获取图像以及IMU数据，这会堵塞程序。
使用回调的方式可以让程序更加灵活，主要就是回调函数的实现，以及将pipe.start(cfg);改为pipe.start(cfg, stream_callback);

```
void stream_callback(const rs2::frame& frame){

    static double frames_fs = 0;
    static double gyro_fs = 0;
    static double accel_fs = 0;
    std::cout << std::endl;
    std::cout << "frames_fs: " << frames_fs << std::endl;
    std::cout << "gyro_fs: " << gyro_fs << std::endl;
    std::cout << "accel_fs: " << accel_fs << std::endl;
    std::cout << std::endl;


    if(rs2::frameset frames = frame.as<rs2::frameset>()){

        //取得每一张图像
        rs2::depth_frame depth_frame = frames.get_depth_frame(); // 获取深度图像数据
        rs2::video_frame color_frame = frames.get_color_frame(); // 获取彩色图像数据
        rs2::frame irL_frame = frames.get_infrared_frame(1); // 左红外相机图像
        rs2::frame irR_frame = frames.get_infrared_frame(2); // 右红外相机图像

        // 图像转Opencv格式
        Mat color_image(Size(640, 480), CV_8UC3,
(void*)color_frame.get_data(),Mat::AUTO_STEP); // 彩色图像
        Mat depth_image(Size(640, 480), CV_16U ,
(void*)depth_frame.get_data(),Mat::AUTO_STEP); // 深度图像
        Mat irL_image  (Size(640, 480), CV_8UC1, (void*)irL_frame.get_data()
,Mat::AUTO_STEP); // 左红外相机图像
        Mat irR_image  (Size(640, 480), CV_8UC1, (void*)irR_frame.get_data()
,Mat::AUTO_STEP); // 右红外相机图像
        imshow("color", color_image);
        imshow("depth", depth_image);
        imshow("irL", irL_image);
        imshow("irR", irR_image);
        waitKey(1);

        // 计算、打印帧率
        static double frames_t_last = 0;
        // Get the timestamp of the current frame
        double frames_t_now = frames.get_timestamp();
        if(frames_t_last > 0){
            frames_fs = 1000/(frames_t_now - frames_t_last);
        }
        frames_t_last = frames_t_now;

    }else if(rs2::motion_frame m_frame = frame.as<rs2::motion_frame>()){
        if (m_frame.get_profile().stream_name() == "Gyro")
        {
            // Get gyro measures
            rs2_vector gyro_data = m_frame.get_motion_data();
            // std::cout << gyro_data << std::endl;

            // 计算、打印帧率
            static double gyro_t_last = 0;
            // Get the timestamp of the current frame
            double gyro_t_now = m_frame.get_timestamp();
            if(gyro_t_last > 0){
                gyro_fs = 1000/(gyro_t_now - gyro_t_last);
                std::cout << "gyro_fs: " << gyro_fs << std::endl;
            }
            gyro_t_last = gyro_t_now;
        }
        else if (m_frame.get_profile().stream_name() == "Accel")
        {
            // Get accelerometer measures
            rs2_vector accel_data = m_frame.as<rs2::motion_frame>
().get_motion_data();
            // std::cout << accel_data << std::endl;

            // 计算、打印帧率
            static double accel_t_last = 0;
            // Get the timestamp of the current frame
            double accel_t_now = m_frame.get_timestamp();
            if(accel_t_last > 0){
                accel_fs = 1000/(accel_t_now - accel_t_last);
                std::cout << "accel_fs: " << accel_fs << std::endl;
            }
            accel_t_last = accel_t_now;
        }
    }
}


int main()
{
    // 首先实例化pipeline和config
    rs2::pipeline pipe;
    rs2::config cfg;

    // RGB stream
    cfg.enable_stream(RS2_STREAM_COLOR, 640, 480, RS2_FORMAT_BGR8, 30);
    // Depth stream
    cfg.enable_stream(RS2_STREAM_DEPTH, 640, 480, RS2_FORMAT_Z16, 30);
    // IMU stream
    cfg.enable_stream(RS2_STREAM_ACCEL, RS2_FORMAT_MOTION_XYZ32F);
    cfg.enable_stream(RS2_STREAM_GYRO , RS2_FORMAT_MOTION_XYZ32F);
    // IR stream
    cfg.enable_stream(RS2_STREAM_INFRARED, 1, 640, 480, RS2_FORMAT_Y8, 30);
    cfg.enable_stream(RS2_STREAM_INFRARED, 2, 640, 480, RS2_FORMAT_Y8, 30);


```

```
    pipe.start(cfg, stream_callback);

    while(1){}
```

将其它图像与深度图像对齐

这属于是空间上的对齐而不是时间上的对齐，也就是像素点的一一对应。
由于这个对齐操作耗时久一点，所以不放在回调里，改到while(1)里。

```
double frames_fs = 0;
rs2::frameset fs_to_align;
int frames_point = 0;
void stream_callback(const rs2::frame& frame){

    // static double frames_fs = 0;
    static double gyro_fs = 0;
    static double accel_fs = 0;
    std::cout << std::endl;
    std::cout << "frames_fs: " << frames_fs << std::endl;
    std::cout << "gyro_fs: " << gyro_fs << std::endl;
    std::cout << "accel_fs: " << accel_fs << std::endl;
    std::cout << std::endl;

    if(rs2::frameset frames = frame.as<rs2::frameset>()){

        fs_to_align = frames;
        frames_point = 1;

    }else if(rs2::motion_frame m_frame = frame.as<rs2::motion_frame>()){
        if (m_frame.get_profile().stream_name() == "Gyro")
        {
            // Get gyro measures
            rs2_vector gyro_data = m_frame.get_motion_data();
            // std::cout << "gyro_data" << gyro_data << std::endl;

            // 计算、打印帧率
            static double gyro_t_last = 0;
            // Get the timestamp of the current frame
            double gyro_t_now = m_frame.get_timestamp();
            if(gyro_t_last > 0){
                gyro_fs = 1000/(gyro_t_now - gyro_t_last);
            }
            gyro_t_last = gyro_t_now;
        }
        else if (m_frame.get_profile().stream_name() == "Accel")
        {
            // Get accelerometer measures
            rs2_vector accel_data = m_frame.as<rs2::motion_frame>
().get_motion_data();
            // std::cout << "accel_data" << accel_data << std::endl;

            // 计算、打印帧率
            static double accel_t_last = 0;
            // Get the timestamp of the current frame
            double accel_t_now = m_frame.get_timestamp();
            if(accel_t_last > 0){
                accel_fs = 1000/(accel_t_now - accel_t_last);
            }
            accel_t_last = accel_t_now;
        }
    }
}


int main()
{
    // 首先实例化pipeline和config
    rs2::pipeline pipe;
    rs2::config cfg;

    // RGB stream
    cfg.enable_stream(RS2_STREAM_COLOR, 640, 480, RS2_FORMAT_BGR8, 30);
    // Depth stream
    cfg.enable_stream(RS2_STREAM_DEPTH, 640, 480, RS2_FORMAT_Z16, 30);
    // IMU stream
    cfg.enable_stream(RS2_STREAM_ACCEL, RS2_FORMAT_MOTION_XYZ32F);
    cfg.enable_stream(RS2_STREAM_GYRO , RS2_FORMAT_MOTION_XYZ32F);
    // IR stream
    cfg.enable_stream(RS2_STREAM_INFRARED, 1, 640, 480, RS2_FORMAT_Y8, 30);
    cfg.enable_stream(RS2_STREAM_INFRARED, 2, 640, 480, RS2_FORMAT_Y8, 30);


    // Create a rs2::align object.
    // rs2::align allows us to perform alignment of depth frames to others frames
    //The "align_to" is the stream type to which we plan to align depth frames.
    rs2::align align(RS2_STREAM_COLOR);


    pipe.start(cfg, stream_callback);



    while(1){

        if(frames_point == 1){

            // 计算、打印帧率
            static double frames_t_last = 0;
            // Get the timestamp of the current frame
            double frames_t_now = fs_to_align.get_timestamp();
            if(frames_t_last > 0){
                frames_fs = 1000/(frames_t_now - frames_t_last);
            }
            frames_t_last = frames_t_now;

            // Perform alignment here
            rs2::frameset aligned_frameset = align.process(fs_to_align);

            // 获取对齐后的RGB图像与深度图像
            rs2::video_frame aligned_color_frame =
```

```
aligned_frameset.get_color_frame();
            rs2::depth_frame aligned_depth_frame =
aligned_frameset.get_depth_frame();

            // 取得每一张图像
            rs2::depth_frame depth_frame = fs_to_align.get_depth_frame(); // 获取深
度图像数据
            rs2::video_frame color_frame = fs_to_align.get_color_frame(); // 获取彩
色图像数据
            rs2::frame irL_frame = fs_to_align.get_infrared_frame(1); // 左红外相机
图像
            rs2::frame irR_frame = fs_to_align.get_infrared_frame(2); // 右红外相机
图像


            // // 图像转Opencv格式
            Mat aligned_color_image(Size(640, 480), CV_8UC3,
(void*)aligned_color_frame.get_data(),Mat::AUTO_STEP); // 彩色图像
            Mat aligned_depth_image(Size(640, 480), CV_16U ,
(void*)aligned_depth_frame.get_data(),Mat::AUTO_STEP); // 深度图像
            Mat color_image(Size(640, 480), CV_8UC3,
(void*)color_frame.get_data(),Mat::AUTO_STEP); // 彩色图像
            Mat depth_image(Size(640, 480), CV_16U ,
(void*)depth_frame.get_data(),Mat::AUTO_STEP); // 深度图像
            Mat irL_image  (Size(640, 480), CV_8UC1, (void*)irL_frame.get_data()
,Mat::AUTO_STEP); // 左红外相机图像
            Mat irR_image  (Size(640, 480), CV_8UC1, (void*)irR_frame.get_data()
,Mat::AUTO_STEP); // 右红外相机图像
            imshow("aligned_color_image", aligned_color_image);
            imshow("aligned_depth_image", aligned_depth_image);
            imshow("color", color_image);
            imshow("depth", depth_image);
            imshow("irL", irL_image);
            imshow("irR", irR_image);
            waitKey(1);
            frames_point = 0;
        }
    }

    return 0;
}
```

程序的主要内容在于：

```
    // Create a rs2::align object.
    // rs2::align allows us to perform alignment of depth frames to others frames
    //The "align_to" is the stream type to which we plan to align depth frames.
    rs2::align align(RS2_STREAM_COLOR);


    // Perform alignment here
    rs2::frameset aligned_frameset = align.process(fs_to_align);
```

各传感器的内参获取

内参可以通过程序获取。

RGB、IR相机内参内容如下，深度图的内参与IR相机相同，包括图像的长宽、内参矩阵、相机模型、畸变系数：

```
/** \brief Video stream intrinsics. */
typedef struct rs2_intrinsics
{
    int           width;     /**< Width of the image in pixels */
    int           height;    /**< Height of the image in pixels */
    float         ppx;       /**< Horizontal coordinate of the principal point of the image, as a pixel offset from the left edge */
    float         ppy;       /**< Vertical coordinate of the principal point of the image, as a pixel offset from the top edge */
    float         fx;        /**< Focal length of the image plane, as a multiple of pixel width */
    float         fy;        /**< Focal length of the image plane, as a multiple of pixel height */
    rs2_distortion model;    /**< Distortion model of the image */
    float         coeffs[5]; /**< Distortion coefficients. Order for Brown-Conrady: [k1, k2, p1, p2, k3]. Order for F-Theta Fish-eye: [k1, k2, k3, k4, 0].
} rs2_intrinsics;
```

IMU传感器的内参内容如下，包含高斯白噪声、零偏、以及各自的协方差：

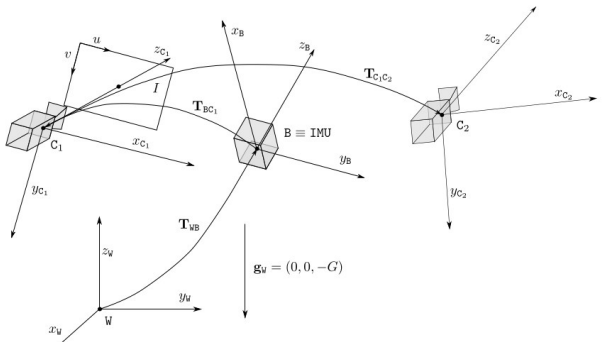
```
/** \brief Motion device intrinsics: scale, bias, and variances. */
typedef struct rs2_motion_device_intrinsic
{
    /* \internal
    * Scale X       cross axis  cross axis  Bias X \n
    * cross axis    Scale Y     cross axis  Bias Y \n
    * cross axis    cross axis  Scale Z     Bias Z */
    float data[3][4];           /**< Interpret data array values */

    float noise_variances[3];  /**< Variance of noise for X, Y, and Z axis */
    float bias_variances[3];   /**< Variance of bias for X, Y, and Z axis */
} rs2_motion_device_intrinsic;
```

```
void l_get_intrinsics(const rs2::stream_profile& stream)
{
    // A sensor's stream (rs2::stream_profile) is in general a stream of data with
no specific type.
    // For video streams (streams of images), the sensor that produces the data
has a lens and thus has properties such
    //  as a focal point, distortion, and principal point.
    // To get these intrinsics parameters, we need to take a stream and first
check if it is a video stream
    if (rs2::video_stream_profile video_stream =
stream.as<rs2::video_stream_profile>())
    {
        try
        {
            // 使用get_intrinsics方法获取相机内参
            rs2_intrinsics intrinsics = video_stream.get_intrinsics();
            // 处理一些结果
            auto principal_point = std::make_pair(intrinsics.ppx, intrinsics.ppy);
            auto focal_length = std::make_pair(intrinsics.fx, intrinsics.fy);
            rs2_distortion model = intrinsics.model;

            // 打印内参
            std::cout << "Principal Point         : " << principal_point.first <<
", " << principal_point.second << std::endl;
            std::cout << "Focal Length            : " << focal_length.first << ",
" << focal_length.second << std::endl;
            std::cout << "Distortion Model        : " << model << std::endl;
            std::cout << "Distortion Coefficients : [" << intrinsics.coeffs[0] <<
"," << intrinsics.coeffs[1] << "," <<
                intrinsics.coeffs[2] << "," << intrinsics.coeffs[3] << "," <<
intrinsics.coeffs[4] << "]" << std::endl;
        }
```

```
        catch (const std::exception& e)
        {
            std::cerr << "Failed to get intrinsics for the given stream. " << e.what() << std::endl;
        }
    }
    else if (rs2::motion_stream_profile motion_stream = stream.as<rs2::motion_stream_profile>())
    {
        try
        {
            // 使用get_motion_intrinsics方法获取IMU内参
            rs2_motion_device_intrinsic intrinsics = motion_stream.get_motion_intrinsics();

            // 打印内参
            std::cout << " Scale X      cross axis      cross axis  Bias X \n";
            std::cout << " cross axis    Scale Y        cross axis  Bias Y  \n";
            std::cout << " cross axis    cross axis     Scale Z     Bias Z  \n";
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 4; j++)
                {
                    std::cout << intrinsics.data[i][j] << "    ";
                }
                std::cout << "\n";
            }

            std::cout << "Variance of noise for X, Y, Z axis \n";
            for (int i = 0; i < 3; i++)
                std::cout << intrinsics.noise_variances[i] << " ";
            std::cout << "\n";

            std::cout << "Variance of bias for X, Y, Z axis \n";
            for (int i = 0; i < 3; i++)
                std::cout << intrinsics.bias_variances[i] << " ";
            std::cout << "\n";
        }
        catch (const std::exception& e)
        {
            std::cerr << "Failed to get intrinsics for the given stream. " << e.what() << std::endl;
        }
    }
    else
    {
        std::cerr << "Given stream profile has no intrinsics data" << std::endl;
    }
}

int main()
{
    // 首先实例化pipeline和config
    rs2::pipeline pipe;
    rs2::config cfg;

    // RGB stream
    cfg.enable_stream(RS2_STREAM_COLOR, 640, 480, RS2_FORMAT_BGR8, 30);
    // Depth stream
    cfg.enable_stream(RS2_STREAM_DEPTH, 640, 480, RS2_FORMAT_Z16, 30);
    // IMU stream
    cfg.enable_stream(RS2_STREAM_ACCEL, RS2_FORMAT_MOTION_XYZ32F);
    cfg.enable_stream(RS2_STREAM_GYRO , RS2_FORMAT_MOTION_XYZ32F);
    // IR stream
    cfg.enable_stream(RS2_STREAM_INFRARED, 1, 640, 480, RS2_FORMAT_Y8, 30);
    cfg.enable_stream(RS2_STREAM_INFRARED, 2, 640, 480, RS2_FORMAT_Y8, 30);

    rs2::pipeline_profile pipe_profile = pipe.start(cfg);

    std::cout << std::endl;
    std::cout << "Get RGB intrinsics :" << std::endl;
    l_get_intrinsics(pipe_profile.get_stream(RS2_STREAM_COLOR));

    std::cout << std::endl;
    std::cout << "Get ACCEL intrinsics :" << std::endl;
    l_get_intrinsics(pipe_profile.get_stream(RS2_STREAM_ACCEL));

    std::cout << std::endl;
    std::cout << "Get GYRO intrinsics :" << std::endl;
    l_get_intrinsics(pipe_profile.get_stream(RS2_STREAM_GYRO));

    std::cout << std::endl;
    std::cout << "Get IR intrinsics :" << std::endl;
    l_get_intrinsics(pipe_profile.get_stream(RS2_STREAM_INFRARED));

    std::cout << std::endl;
    std::cout << "Get DEPTH intrinsics :" << std::endl;
    l_get_intrinsics(pipe_profile.get_stream(RS2_STREAM_DEPTH));
}
```

各传感器之间的外参获取

默认的坐标系：



如图所示，相机坐标系向前为Z、向左为X、向下为Y;IMU坐标系向前为Z、向左为Y，向上为X;而我们常用的右手坐标系向上为Z、向左为Y、向后为X。

```
void l_get_extrinsics(const rs2::stream_profile& from_stream, const rs2::stream_profile& to_stream){

    // If the device/sensor that you are using contains more than a single stream, and it was calibrated
    // then the SDK provides a way of getting the transformation between any two streams (if such exists)
    try
    {
        // Given two streams, use the get_extrinsics_to() function to get the transformation from the stream to the other stream
        rs2_extrinsics extrinsics = from_stream.get_extrinsics_to(to_stream);
        std::cout << "Translation Vector : [" << extrinsics.translation[0] << "," << extrinsics.translation[1] << "," << extrinsics.translation[2] << "]\n";
        std::cout << "Rotation Matrix    : [" << extrinsics.rotation[0] << "," << extrinsics.rotation[3] << "," << extrinsics.rotation[6] << "]\n";
        std::cout << "                     [" << extrinsics.rotation[1] << "," << extrinsics.rotation[4] << "," << extrinsics.rotation[7] << "]\n";
        std::cout << "                     [" << extrinsics.rotation[2] << "," << extrinsics.rotation[5] << "," << extrinsics.rotation[8] << "]" << std::endl;
    }
    catch (const std::exception& e)
    {
        std::cerr << "Failed to get extrinsics for the given streams. " << e.what() << std::endl;
    }
}




int main()
{
    // 首先实例化pipeline和config
    rs2::pipeline pipe;
    rs2::config cfg;

    // RGB stream
    cfg.enable_stream(RS2_STREAM_COLOR, 640, 480, RS2_FORMAT_BGR8, 30);
    // Depth stream
    cfg.enable_stream(RS2_STREAM_DEPTH, 640, 480, RS2_FORMAT_Z16, 30);
    // IMU stream
    cfg.enable_stream(RS2_STREAM_ACCEL, RS2_FORMAT_MOTION_XYZ32F);
    cfg.enable_stream(RS2_STREAM_GYRO , RS2_FORMAT_MOTION_XYZ32F);
    // IR stream
    cfg.enable_stream(RS2_STREAM_INFRARED, 1, 640, 480, RS2_FORMAT_Y8, 30);
    cfg.enable_stream(RS2_STREAM_INFRARED, 2, 640, 480, RS2_FORMAT_Y8, 30);

    // Create a rs2::align object.
    // rs2::align allows us to perform alignment of depth frames to others frames
    //The "align_to" is the stream type to which we plan to align depth frames.
    rs2::align align(RS2_STREAM_COLOR);

    rs2::pipeline_profile pipe_profile = pipe.start(cfg);


l_get_extrinsics(pipe_profile.get_stream(RS2_STREAM_COLOR),pipe_profile.get_stream(RS

}
```

OVER

总结的代码我会放到我们码云上，地址

工欲善其事，必先利其器~

~

~

结尾玄月~

Realsense RGBD

打赏 0　点赞 1　收藏 0　分享

上一篇:从欧拉角到旋转位移矩阵再到变换矩阵

下一篇:单目相机内参标定与畸变矫正

为你推荐

事件相机(Event-based camera)模拟器功能介绍
阅读数 448 评论数 0

跟踪数据集汇总
阅读数 116 评论数 0

Tensorflow学习——结合ROS调用模型实现目标识别
阅读数 5431 评论数 0

激光雷达点云操作的工程化分享(三)—— RANSAC...
阅读数 771 评论数 0

数字图像处理(一)绪论
阅读数 3109 评论数 0

神器CLIP:连接文本和图像，打造可迁移的视觉模型
阅读数 816 评论数 0

评论(0)

您还未登录，请登录后发表或查看评论

阅读数 2451 评论数 0

图像缩放中的插值算法——双线性插值原理及c++实现
阅读数 306 评论数 0

热门泡泡

30积分 **失眠，聊聊自己搞ROS的心得体会吧**

ros学习路线

30积分 **TF_REPEATED_DATA ignoring data错误**

各位大佬，有什么ROS定位算法推荐吗

5积分 **ros中启动gazebo时报错**

5积分 **想买能用ROS2的开发套件。或者开发板**

TA的专栏

3篇
3篇
3篇
2篇
4篇

专栏导航 1 / 1

- 齐次坐标、对极约束、基本矩阵与本质矩阵、单应矩阵
- 单目相机内参标定与畸变矫正
- 英特尔D435i深度相机和librealsense安装使用