Royale 0.9.5.32

Introduction

Version 0.9.5.32

0.9.5.32
Created: 03.09.2015

Contents

1 Introduction

The **Royale** software package provides a light-weight camera framework for time-of-flight (ToF) cameras. While being tailored to PMD cameras, the framework enables partners and customers to evaluate and/or integrate 3D TOF technology on/in their target platform. This reduces time to first demo and time to market.

Royale contains all the logic which is required to operate a ToF based camera. The user need not care about setting registers, but can conveniently control the camera via a high-level interface. The Royale framework is completely designed in C++ using the C++11 standard.

Royale officially supports the **CamBoard pico flexx** camera.

on the following platforms:

- Windows 7/8
- Linux (tested on Ubuntu 14.04)
- OS X (tested on Yosemite 10.10.4)
- Android (tested on Android 4.4.2, currently not running on all Android 5 images)

2 Getting Started

Plug in your supported PMD camera to a free USB port of your target device. For some Android devices you may need a powered USB hub for proper usage. For Windows-based systems, please install the low-level camera driver which is part of the delivery package. For Linux-based systems, make sure that you have proper permissions to the USB device. The installation package contains a proper rules file which can be used. For more details, please see the drivers/udev/README file (for Linux distributions).

Once the camera is attached to a free USB port, and the drivers are in place, you may start the **royaleviewer** application which gives you a first indication, if the camera is working on your target system. The royaleviewer displays a 2D or a 3D representation of the captured depth data. By clicking on the *Start* button, the capture mode is started and you should see a colored depth map.

Depending on the camera type, you have a set of operation modes which can be chosen in the application. An **operation mode** is a pre-defined set of parameters which represents a certain use case. Please refer to the section “Use Case” further below for detailed information.

The operation mode encodes three different type of information:

- number of raw frames being captured for generating the depth image
- number of frames per seconds
- the maximal exposure time for a raw frame capture event

A *raw frame* contains a 12-bit image which is returned from the ToF imager using a single phase measurement. The user typically does not need to set the internals of the ToF imager since Royale is providing this low-level information.

The operation mode is denoted in the following structure:

MODE_S1_S2FPS_S3

where

S1 is the number of raw images used to generate a depth image, *S2* is the number for frames per seconds, and *S3* is the exposure time in microseconds.

An example is *MODE_9_10FPS_1000* which captures 9 raw frames (8 modulated plus one intensity image) with 10 frames per seconds, and a maximal integration time of 1000 microseconds.

2.1 Access Levels

Royale offers three different access levels, each one representing a specific user base.

> For eye-safety reasons access for devices labelled as `Laser Class 1` is only permitted on `Level 1` and `Level 2`.

- **Level 1** users are *standard users* who operate the camera in a default configured way. Only a limited set of functions are exposed which allow the change the behaviour of the camera module. This level is typically used by application developer who want to integrate Royale into their applications serving high quality depth data.
- **Level 2** For experienced users who can also handle raw data in addition to depth data. Users are also allowed to alter internal processing parameters in order to evaluate the depth data generation and optimise for certain use cases. This level is typically used by users who want to evaluate and optimise depth but also raw data.
- **Level 3** Users are highly professional who are working on bringing up new camera modules, and also optimise the internal parameters of the camera module. This mode must be run with caution since it may damage the hardware but also harm your health. This level is restricted to internal employees only and will not be exposed to customers.

2.1.0.1 Level Activation

The different user level can be activated via the `ICameraDevice` Interface by entering the correct access codes on construction via the `CameraDeviceManager`. `Level 1` is activated by default, no code is required. `Level 2` and `Level 3` can only be activated with a special activation code (two different codes, one for each level). The higher level also inherits all functionality of the lower levels.

2.2 Use Cases

For 3D sensing a set of key applications and corresponding use cases have been determined and a corresponding set of processing parameters and camera options have been derived. Those values can only be changed on the 2nd and 3rd level of the `ICameraDevice` interface.

> Please note that these settings are initial proposals. When investigating your specific application do not hesitate to try a different operation modes, in order to verify whether it provides more beneficial data.

Nr	Use Cases	Operation Modes	Frequencies	Range [m]	Framerate	Time (us)
1	Indoor room reconstruction	MODE_9_5FPS_2000	80 & 60 MHz	1 - 4.0	5 fps	2000
2	Room scanning indoor navigation	MODE_9_10FPS_1000	80 & 60 MHz	1 - 4.0	10 fps	1000

Nr	Use Cases	Operation Modes	Frequencies	Range [m]	Framerate	Time (us)
3	3D object reconstruction	MODE_9_15FPS_700	80 & 60 MHz	0.5 - 1.5	15 fps	700
4	Medium size object recognition, face reconstruction	MODE_9_25FPS_450	80 & 60 MHz	0.3 - 2.0	25 fps	450
5	Remote collaboration, step by step instruction, table-top gaming	MODE_5_35FPS_600	60 MHz	0.3 - 2.0	35 fps	600
6	Small object/product recognition, Hand tracking	MODE_5_45FPS_500	60 MHz	0.1 - 1.0	45 fps	500

- **Indoor room reconstruction** **PMD** sensors are a viable solution to locate objects or people inside large environments, such as buildings. This Use-Case is optimized for long range scanning at a maximum data quality. By making use of multiple frequencies the ambiguity range of the sensor signal can be increased by several magnitudes. At the same this sampling methods leads to an increase in data confidence and applications with very high demands in data quality can be realized.
- **Room scanning, indoor navigation** For mapping applications demanding an enhanced situational awareness **PMD** quick response time are a necessity. These demands are met by increasing the framerate at a minimum cost in data quality.
- **3D object reconstruction** Scanning and Reproduction of man-sized objects in close proximity demands high data confidence equal to environmental mapping. Since in general the objects of interest are in closer proximity, the range requirements and necessary integration time could be lowered in favor of faster scanning speed.
- **Medium size object Recognition, face reconstruction** In general the quality demands of applications in the field of pattern and object recognition are less demanding than metrological applications. On the other hand, a quick system response time is mandatory. Therefore the integration time and correspondingly the data quality is lowered in favor of faster framerates.
- **Remote collaboration, step by step instruction, table-top gaming** For modern gaming and collaborative applications a quick system response is even more important. Since the range requirement can be lowered and the noise performance of **PMD** sensors is directly related to the object distance, higher framerates at equal data quality can be realized.
- **Small object/product recognition** For hand-size objects and products the necessary range requirements can be further limited and only one scanning frequency is sufficient. Therefore the framerate could be almost doubled and vice versa the overall scanning speed.
- **Hand tracking** The precise detection and recognition of hand gestures in 3D space is very demanding, both in data quality and processing speed. Hence a special Operation Mode has been devised offering optimum setting for this special application.

3 SDK and Examples

Besides the royaleviewer.exe application, the package also provides a Royale SDK which can be used to develop applications using the PMD camera.

There are two samples included in the delivery package which should give you a brief overview about the exposed API. The *doc* directory offers a detailed description of the Royale API which is a good starting point as well.

The Royale framework currently offers capturing in free running mode only. During capturing the exposure time can be altered up to the maximal specified values defined by the operation mode.

The data is provided using a callback based interface. The consumer must register a listener to the camera device and retrieves the depth data. The high-level API consists of the following interfaces:

- **ICameraDevice** Main interface which represents the physical camera. The user will mostly work with this interface to talk to the physical camera device.
- **CameraManager** The main component which can query for connected/supported cameras and is able to generate the proper ICameraDevice.
- **CameraStatus** Gives information about the internal state of the system. Most methods will return a proper error code to the caller.
- **IDepthDataListener** Interface which needs to be implemented in order to retrieve the depth data.
- **DepthData** Contains the processed depth data which can be consumed by the user. Furthermore it contains the exposureTimes for all frames within one DepthData

3.1 Compilation

The package contains all necessary headers and libraries including two samples which should provide a first get-in-touch. The samples also contain a CMakeLists.txt file which allows to compile the code on different platforms.

Please note that you need a C++11 compliant compiler:

- Windows: Visual Studio 2013 or higher
- Linux: gcc 4.8 or higher
- OS X: gcc 4.8 or higher or clang 6.1 or higher

3.1.1 Linux

Open up your favorite console:

```
> cd samples/sample1
> mkdir build
> cd build
> cmake ..
> make
```

3.1.2 Windows

Open up your development shell (e.g. Visual Studio command line):

```
> cd samples/sample1
> mkdir build
> cd build
> cmake ..
```

Then open up your generated solution file with your Visual Studio IDE.

For Debug builds you also have to set the /MD flag in the properties of your project. Otherwise you might experience crashes. For the samples and the royale-config.cmake this is already set.

3.1.3 OS X

Open up your favorite console:

```
> cd samples/sample1
> mkdir build
> cd build
> cmake -G Xcode ..
```

Then open up your generated project file with Xcode IDE.

The package also provides a cmake config file for properly building Royale applications. The cmake file can be found under the share directory.

3.1.4 Requirements

The package requires to have the libusb available under Linux-based systems.

3.2 Examples

The following examples demonstrate the current API and its usage.

3.2.1 Create a CameraDevice

The physical depth camera module is represented by the `ICameraDevice`. This object is automatically generated by the `CameraManager`. During initialization, the correct configuration parameters for the camera module are loaded and the module/imager is initialized. Here is an example for how to instantiate a `ICameraDevice`:

```
std::unique_ptr<ICameraDevice> camera;
CameraManager manager;

auto connectedCameras = manager.getConnectedCameraList();

// get the first found camera, assuming one camera was found
camera = manager.createCamera(connectedCameras[0]);
```

3.2.2 Set an OperationMode

The camera can be operated in pre-defined operation modes. The following code snippet shows, how to set a certain operation mode:

```
camera->setOperationMode(camera->getOperationModes()[0]);
```

Since the operation mode provides a maximal exposure time, the time can be set during runtime by using the setExposureTime() method. The pre-defined operation modes make sure that eye safety is guaranteed for cameras using laser illumination.

```
camera->setExposureTime(600);
```

3.2.3 Start Capture Mode

After setting up the capture mode, the camera can start capturing. Therefore, a listener needs to be registered:

```
camera->registerDataListener(listener);
```

The listener implementation needs to be derived from the following interface:

```
class IDepthDataListener
{
public:
    virtual ~IDepthDataListener() {}
    virtual void onNewData (const DepthData * data) = 0;
};
```

After registering a listener, the camera capture mode can be started:

```
camera->startCapture();
```

The camera capture mode can be stopped again by the following code:

```
camera->stopCapture();
```

3.2.4 Data Structure

3.2.4.1 Point Cloud Data

The Royale framework delivers 3D point cloud data with the following information for each voxel:

- X/Y/Z coordinates in object space [m]
- Grayscale information
- Depth noise
- Depth confidence

The point cloud data is a dense map of those values in order to maintain neighbourhood information. The following data structure is proposed for the 3D point cloud data:

```
struct DepthPoint
{
    float x;                 //!< X coordinate [meters]
    float y;                 //!< Y coordinate [meters]
    float z;                 //!< Z coordinate [meters]
    float noise;             //!< noise value [meters]
    uint16_t grayValue;      //!< 16-bit gray value
    uint8_t depthConfidence; //!< value 0 = bad, 255 = good
};

struct DepthData
{
    int                         version;         //!< version number of the data format
    std::chrono::milliseconds timeStamp;         //!< timestamp in milliseconds precision
                                                 //   (time since epoch 1970)
    uint16_t                    width;           //!< width of depth image
    uint16_t                    height;          //!< height of depth image
    std::vector<uint32_t>       exposureTimes;   //!< exposureTimes retrieved from
                                                 //   CapturedUseCase
    std::vector<DepthPoint>     points;          //!< array of points
};
```

4 Reference

FAQ: http://pmdtec.com/picoflexx/

5 License

See royale_license.txt. Parts of the software covered by this License Agreement (royale_license.txt) is using libusb under LGPL 2.1, QT5.4 under LGPL 3.0 and zlib under zlib license.