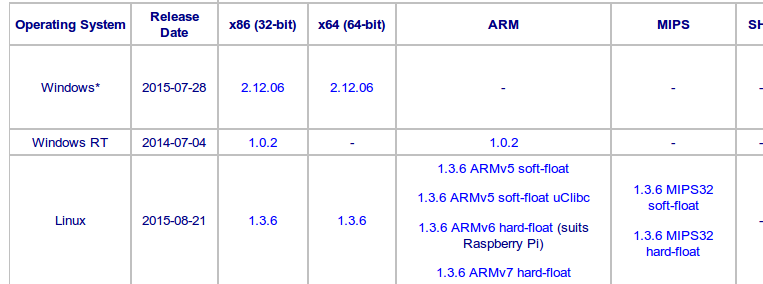
ubuntu下陀螺仪的安装

1.安装libftd2xx.so

<http://www.ftdichip.com/Drivers/D2XX.htm下载相关文件>

选择linux下x64 64bit 的1.3.6

解压后的相关操作看AN\_220\_FTDI\_Drivers\_Installation\_Guide\_for\_Linux .pdf

里面出现sudo rmmod ftdi\_si运行错误

sudo rmmod ftdi\_sio <ret>

sudo rmmod usbserial <ret>

rmmod: ERROR: Module ftdi\_sio is not currently loaded

首先检查设备连接是否有问题

然后输入检查是否load命令

lsmod | grep ftdi\_sio

如果仍然没出现相关设备信息输入 sudo modprobe ftdi\_sio

2.进行设备驱动安装

eb package installation

========================

libLpSensor depends on libbluetooth.so and libftd2xx.so. For installation of libftd2xx, please visit http://www.ftdichip.com/Drivers/D2XX.htm.

$ sudo dpkg -i liblpsensor-1.3.4-Linux.deb

$ dpkg -L liblpsensor

Compiling Sample programs

=========================

You can use libraries provided in ./lib to interface with our sensors directly. A simple example program is provided in ./sample demonstrating how to do this:

$ cd ./sample

$ mkdir build

$ cd build

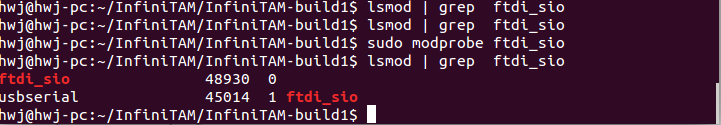
$ cmake ..

$ make

$ sudo ./LpmsSimpleExample

LpmsSimpleExample中的设备号一行命令改成我们自己的

LpmsSensorI\* lpms = manager->*addSensor*(DEVICE\_LPMS\_U, "A5022WD0");



// Copyright 2014-2015 Isis Innovation Limited and the authors of InfiniTAM

#pragma once

#include "../ITMLib/ITMLib.h"

#include <cstdio>

#include <thread>

#include "LpmsSensorI.h"

#include "LpmsSensorManagerI.h"

/\*namespace InfiniTAM

{

namespace Engine

{

class IMUSourceEngine

{

private:

static const int BUF\_SIZE = 2048;

char imuMask[BUF\_SIZE];

ITMIMUMeasurement \*cached\_imu;

void loadIMUIntoCache();

int cachedFrameNo;

int currentFrameNo;

public:

IMUSourceEngine(const char \*imuMask);

~IMUSourceEngine() { }

bool hasMoreMeasurements(void);

void getMeasurement(ITMIMUMeasurement \*imu);

};

}

}\*/

namespace InfiniTAM

{

namespace Engine

{

class IMUSourceEngine

{

private:

ImuData d;

// Gets a LpmsSensorManager instance

LpmsSensorManagerI\* manager;

// Connects to LPMS-B sensor with address 00:11:22:33:44:55

LpmsSensorI\* lpms;

ITMIMUMeasurement \*cached\_imu;

void loadIMUIntoCache();

int cachedFrameNo;

int currentFrameNo;

public:

IMUSourceEngine();

~IMUSourceEngine();

bool hasMoreMeasurements(void);

void getMeasurement(ITMIMUMeasurement \*imu);

};

}

}

// Copyright 2014-2015 Isis Innovation Limited and the authors of InfiniTAM

/\*#include "IMUSourceEngine.h"

#include "../Utils/FileUtils.h"

#include <stdio.h>

using namespace InfiniTAM::Engine;

IMUSourceEngine::IMUSourceEngine(const char \*imuMask)

{

strncpy(this->imuMask, imuMask, BUF\_SIZE);

currentFrameNo = 0;

cachedFrameNo = -1;

cached\_imu = NULL;

}

void IMUSourceEngine::loadIMUIntoCache(void)

{

char str[2048]; FILE \*f; bool success = false;

cached\_imu = new ITMIMUMeasurement();

sprintf(str, imuMask, currentFrameNo);

f = fopen(str, "r");

if (f)

{

size\_t ret = fscanf(f, "%f %f %f %f %f %f %f %f %f",

&cached\_imu->R.m00, &cached\_imu->R.m01, &cached\_imu->R.m02,

&cached\_imu->R.m10, &cached\_imu->R.m11, &cached\_imu->R.m12,

&cached\_imu->R.m20, &cached\_imu->R.m21, &cached\_imu->R.m22);

fclose(f);

if (ret == 9) success = true;

}

if (!success) {

delete cached\_imu; cached\_imu = NULL;

printf("error reading file '%s'\n", str);

}

}

bool IMUSourceEngine::hasMoreMeasurements(void)

{

loadIMUIntoCache();

return (cached\_imu != NULL);

}

void IMUSourceEngine::getMeasurement(ITMIMUMeasurement \*imu)

{

bool bUsedCache = false;

if (cached\_imu != NULL)

{

imu->R = cached\_imu->R;

delete cached\_imu;

cached\_imu = NULL;

bUsedCache = true;

}

if (!bUsedCache) this->loadIMUIntoCache();

++currentFrameNo;

}

\*/

#include "IMUSourceEngine.h"

#include "../Utils/FileUtils.h"

//#include <stdio.h>

using namespace InfiniTAM::Engine;

IMUSourceEngine::IMUSourceEngine()

{

currentFrameNo = 0;

cachedFrameNo = -1;

cached\_imu = NULL;

manager = LpmsSensorManagerFactory();

lpms = manager->*addSensor*(DEVICE\_LPMS\_U, "A5022WD0");

}

void IMUSourceEngine::loadIMUIntoCache(void)

{

bool success = false;

cached\_imu = new ITMIMUMeasurement();

//sprintf(str, imuMask, currentFrameNo);

//f = fopen(str, "r");

if (lpms->*getConnectionStatus*() == SENSOR\_CONNECTION\_CONNECTED &&

lpms->*hasImuData*())

{

d = lpms->*getCurrentData*();

cached\_imu->R.m00 = d.rotationM[0];

cached\_imu->R.m01 = d.rotationM[1];

cached\_imu->R.m02 = d.rotationM[2];

cached\_imu->R.m10 = d.rotationM[3];

cached\_imu->R.m11 = d.rotationM[4];

cached\_imu->R.m12 = d.rotationM[5];

cached\_imu->R.m20 = d.rotationM[6];

cached\_imu->R.m21 = d.rotationM[7];

cached\_imu->R.m22 = d.rotationM[8];

success = true;

}

if (!success) {

delete cached\_imu; cached\_imu = NULL;

printf("error reading\n");

}

}

bool IMUSourceEngine::hasMoreMeasurements(void)

{

loadIMUIntoCache();

return (cached\_imu != NULL);

}

void IMUSourceEngine::getMeasurement(ITMIMUMeasurement \*imu)

{

bool bUsedCache = false;

if (cached\_imu != NULL)

{

imu->R = cached\_imu->R;

delete cached\_imu;

cached\_imu = NULL;

bUsedCache = true;

}

if (!bUsedCache) this->loadIMUIntoCache();

++currentFrameNo;

}

IMUSourceEngine::~IMUSourceEngine()

{

// Removes the initialized sensor

manager->*removeSensor*(lpms);

// Deletes LpmsSensorManager object

delete manager;

}

// Copyright 2014-2015 Isis Innovation Limited and the authors of InfiniTAM

#include <cstdlib>

#include "Engine/UIEngine.h"

#include "Engine/ImageSourceEngine.h"

#include "Engine/OpenNIEngine.h"

#include "Engine/Kinect2Engine.h"

#include "Engine/LibUVCEngine.h"

using namespace InfiniTAM::Engine;

/\*\* Create a default source of depth images from a list of command line

arguments. Typically, @para arg1 would identify the calibration file to

use, @para arg2 the colour images, @para arg3 the depth images and

@para arg4 the IMU images. If images are omitted, some live sources will

be tried.

\*/

static void CreateDefaultImageSource(ImageSourceEngine\* & imageSource, IMUSourceEngine\* & imuSource, const char \*arg1, const char \*arg2, const char \*arg3, const char \*arg4)

{

const char \*calibFile = arg1;

const char \*filename1 = arg2;

const char \*filename2 = arg3;

const char \*filename\_imu = arg4;

printf("using calibration file: %s\n", calibFile);

if (filename2 != NULL)

{

printf("using rgb images: %s\nusing depth images: %s\n", filename1, filename2);

if (filename\_imu == NULL)

{

imageSource = new ImageFileReader(calibFile, filename1, filename2);

imuSource = new IMUSourceEngine();//gaidong

}

else

{

printf("using imu data: %s\n", filename\_imu);

imageSource = new RawFileReader(calibFile, filename1, filename2, Vector2i(320, 240), 0.5f);

imuSource = new IMUSourceEngine();//gaidong

}

}

if (imageSource == NULL)

{

printf("trying OpenNI device: %s\n", (filename1==NULL)?"<OpenNI default device>":filename1);

imageSource = new OpenNIEngine(calibFile, filename1);

if (imageSource->*getDepthImageSize*().x == 0)

{

delete imageSource;

imageSource = NULL;

}

}

if (imageSource == NULL)

{

printf("trying UVC device\n");

imageSource = new LibUVCEngine(calibFile);

if (imageSource->*getDepthImageSize*().x == 0)

{

delete imageSource;

imageSource = NULL;

}

}

if (imageSource == NULL)

{

printf("trying MS Kinect 2 device\n");

imageSource = new Kinect2Engine(calibFile);

if (imageSource->*getDepthImageSize*().x == 0)

{

delete imageSource;

imageSource = NULL;

}

}

// this is a hack to ensure backwards compatibility in certain configurations

if (imageSource == NULL) return;

if (imageSource->calib.disparityCalib.params == Vector2f(0.0f, 0.0f))

{

imageSource->calib.disparityCalib.type = ITMDisparityCalib::*TRAFO\_AFFINE*;

imageSource->calib.disparityCalib.params = Vector2f(1.0f/1000.0f, 0.0f);

}

}

int main(int argc, char\*\* argv)

try

{

const char \*arg1 = "";

const char \*arg2 = NULL;

const char \*arg3 = NULL;

const char \*arg4 = NULL;

int arg = 1;

do {

if (argv[arg] != NULL) arg1 = argv[arg]; else break;

++arg;

if (argv[arg] != NULL) arg2 = argv[arg]; else break;

++arg;

if (argv[arg] != NULL) arg3 = argv[arg]; else break;

++arg;

if (argv[arg] != NULL) arg4 = argv[arg]; else break;

} while (false);

if (arg == 1) {

printf("usage: %s [<calibfile> [<imagesource>] ]\n"

" <calibfile> : path to a file containing intrinsic calibration parameters\n"

" <imagesource> : either one argument to specify OpenNI device ID\n"

" or two arguments specifying rgb and depth file masks\n"

"\n"

"examples:\n"

" %s ./Files/Teddy/calib.txt ./Files/Teddy/Frames/%%04i.ppm ./Files/Teddy/Frames/%%04i.pgm\n"

" %s ./Files/Teddy/calib.txt\n\n", argv[0], argv[0], argv[0]);

}

printf("initialising ...\n");

ImageSourceEngine \*imageSource = NULL;

IMUSourceEngine \*imuSource = NULL;

CreateDefaultImageSource(imageSource, imuSource, arg1, arg2, arg3, arg4);

if (imageSource==NULL)

{

std::cout << "failed to open any image stream" << std::endl;

return -1;

}

ITMLibSettings \*internalSettings = new ITMLibSettings();

ITMMainEngine \*mainEngine = new ITMMainEngine(internalSettings, &imageSource->calib, imageSource->*getRGBImageSize*(), imageSource->*getDepthImageSize*());

UIEngine::Instance()->Initialise(argc, argv, imageSource, imuSource, mainEngine, "./Files/Out", internalSettings->deviceType);

UIEngine::Instance()->Run();

UIEngine::Instance()->Shutdown();

delete mainEngine;

delete internalSettings;

delete imageSource;

delete imuSource;//gaidong

if (imuSource != NULL) delete imuSource;

return 0;

}

catch(std::exception& e)

{

std::cerr << e.*what*() << '\n';

return EXIT\_FAILURE;

}