

ROBOTICVISION 文档



【版权声明】

版权所有©百度在线网络技术（北京）有限公司、北京百度网讯科技有限公司。 未经本公司书面许可，任何单位和个人不得擅自摘抄、复制、传播本文档内容，否则本公司有权依法追究法律责任。

【商标声明】



和其他百度系商标，均为百度在线网络技术（北京）有限公司、北京百度网讯科技有限公司的商标。 本文档涉及的第三方商标，依法由相关权利人所有。未经商标权利人书面许可，不得擅自对其商标进行使用、复制、修改、传播等行为。

【免责声明】

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导。 如您购买本文档介绍的产品、服务，您的权利与义务将依据百度智能云产品服务合同条款予以具体约定。本文档内容不作任何明示或暗示的保证。

目录

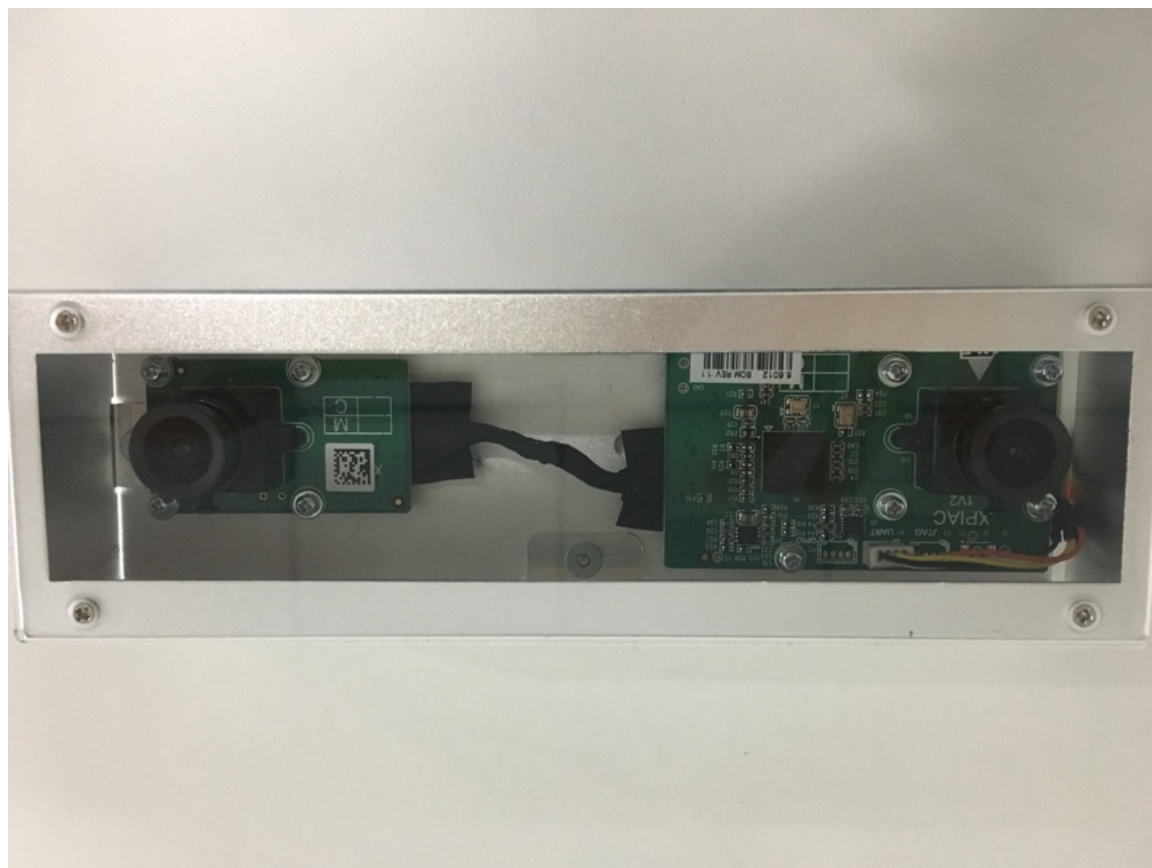
目录	2
产品介绍	3
产品简介	3
硬件构成	3
产品性能	3
环境安装	4
系统要求	4
环境配置	4
模组校正	4
采集数据	4
简介	6
验证效果	6
预验覆盖	8
校正程序	8
范例程序	9
准备工作	9
功能说明	9
循迹导航	9
深度图+障碍物提示：	10
基本SLAM功能	11
识别功能	12
路径规划导航	13
SLAM的存储和读取地图：	14
编译开发	15
接口调用	15
编译程序	15
常见问题	16
常见问题	16
识别接口	16
使用机器人视觉SDK调用百度AI平台在线API服务	16
接口说明	17
建立机器人视觉应用	17

产品介绍

产品简介

百度机器人视觉平台的立体惯性相机模组具有三维空间定位，障碍物检测，和物体识别功能。产品具有定位准确度高，延迟低，障碍物检测距离远，不受阳光照射影响，物体检测种类多，准确度高等优点。该专利产品由世界顶尖研发人员历时多年开发，应用了传感器融合，深度学习，和硬件优化等核心技术。立体惯性相机可以支持主流的PC、ARM平台。如果选配合百度机器人视觉平台的计算芯片，软件系统无需占用上位机资源。立体惯性相机已有多个应用案例，可以应用于机器人领域做导航避障，或者应用于可穿戴设备做头部跟踪。

硬件构成



上图是

由百度及其硬件供应商提供的试用模组：

- 模组的硬件包含两个摄像头（可换）和一个六轴IMU
- 基线（两摄像头间距）可调6-20cm（试用模组已固定）
- 内置驱动，Linux下无需额外安装 照相机与IMU时钟硬件同步，精度10us

产品性能

功能模块	技术指标
照相机	90 – 170 FOV, Global Shutter, VGA
惯性传感器	Gyroscope +- 1000 DPS Accelerometer +- 8 G
驱动接口	USB 3.0 或 2.0 支持 x86 或 ARM 平台
使用场景	居家场景、办公场所、商场等
三维空间定位	6自由度，延迟小于10毫秒
空间定位误差	无已知地图时小于 1% 行走距离 有已知地图时小于 0.5% 地图尺寸
障碍物检测范围	0最近0.2米，最远 50米
障碍物检测误差	障碍物到相机距离的1%
物体、文字、人脸识别	连接百度API

环境安装

系统要求

- 硬件平台：x86或者ARM
- 操作系统：Linux 14.04/16.04

环境配置

- 试用模组对应的软件包含两个压缩包，其中命名为3rdparty_lib_lean-MM-DD-YY.tar.gz是环境安装包，打开命令行
- 运行以下命令行以解压，创建目录和安装第三方库：

```
mkdir ~/XP_release
tar -xvf [3rdparty_lib_lean_mmdyyy.tar.gz] -C ~/XP_release
source 3rdparty_lib_lean/update.sh
```

注意：此处路径需要固定为 ~/XP_release，因为后续脚本全部都会去此处找第三方库。若想使用识别功能，请在运行update.sh时加参数 -enable_recognition

- （非必需）如果需要之后在本地编译我们提供的范例程序，此时还应该装如下库：

```
sudo apt-get install build-essential cmake libgflags-dev libgoogle-glog-dev
```

- 另一个压缩包命名为XX.YY.ZZ.tar.gz，这是我们的程序及sdk包，其中XX.YY.ZZ为版本号。可以把程序包也解压到~/XP_release路径下。

模组校正

采集数据

校正的第一步是采集数据，数据的采集决定了校正的质量，请按步骤细心进行。

- 进入之前解压的程序所在的文件夹内，将模组接到PC，运行如下命令：

```
./pc_apps/xp_sensor_logger -record_path /tmp/record1 -sensor_type XP2 -spacebar_mode
```

xp_sensor_logger 为我们的采集／验证程序，其中：

- -record_path 为数据采集存放的路径

- `-sensor_type` 为我们的硬件版本
- `-spacebar_mode` 代表此模式下每按一下空格键采集一张图片
如需查看所有参数及其使用方法，可加 `-helpshort`
- 用A4纸打印一张程序包根目录下的ApritTag5x7.png图片，将打印好的纸放在平坦的桌面上，并用尺子测量途中任一个二维码的矩形边长。（根据打印纸张以及页边距的不同，不同打印机打印出来的边长稍有差异，请仔细测量，并精确到毫米，一般为3cm左右）
- 拿起模组，按照以下方式采集图像（按一次空格采集一张）：
 - **A类图**：二维码图纸所有二维码均在左右镜头中，约30张。



- **B类图**：二维码图纸中的部分二维码在镜头外（镜头的各个边缘和角落都要被覆盖采集过），可以只保留部分二维码在镜头中，视情况约50张



- 采集的目的：要尽量使左右镜头中的每一个部分均被二维码覆盖采集过，以保证之后在使用时整个镜头的各个部分的特点都能被有效还原，所以采集图片数多多益善，80-90张为宜
- 注意：采集过程中不要让镜头中出现显示屏，否则二维码会在图片中二次成像，导致整个校正失败！
- 采集完成后，按ESC键退出

简介

摄像头的标定以及参数校正是至关重要的一步，匹配的参数才能保证程序运行时的良好效果。由于我们的试用模组固定在铁盒中，且在寄出试用套装时已经对每一个模组进行过校正，因此大部分情况下拿到后可以直接引入我们提供的校正文件使用。但如果出现磕碰或是镜头被触碰过，或者发现效果不够好，就需要进行重新校正以达到最佳效果（也建议每隔一段时间校正一次）。

验证效果

- 运行完校正运算程序后，需要验证校正的结果，运行如下命令：

```
./pc_apps/xp_sensor_logger -calib_verify -calib_yaml ./tmp/calib.yaml -sensor_type XP2
```

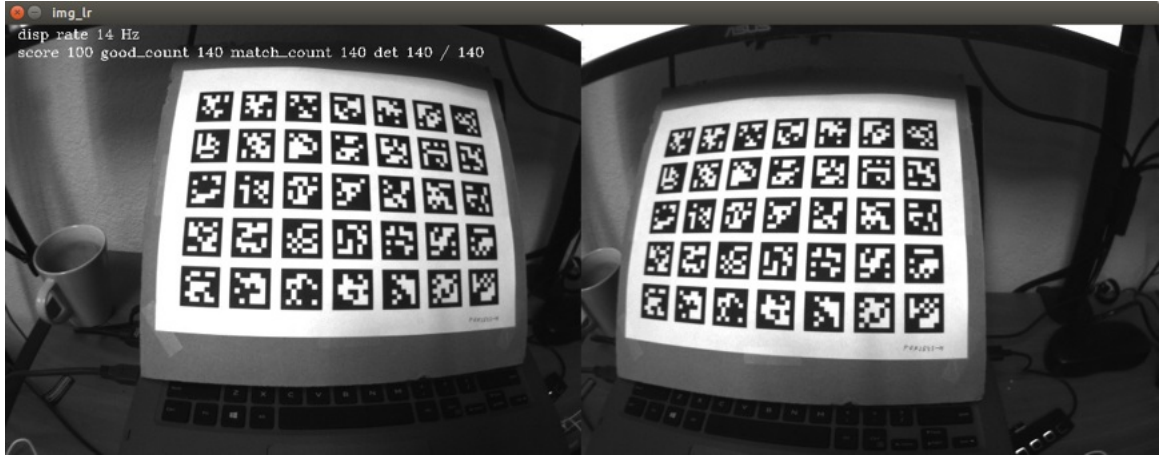
xp_sensor_logger 为我们的采集/验证程序，其中：

- -calib_verify 表示这是验证模式
- -calib_yaml 为要验证的校正文件
- -sensor_type 为我们的硬件版本

如需查看所有参数及其使用方法，可加 -helpshort

- 拿起模组，将模组指向二维码图纸，可以看到左上角会有分数（score），如下图显示：
 - 分数满分为100分
 - 若图纸在镜头视角中间时不是100分，需要重新采集并校正

- 图纸在镜头边缘时分数可能会略低，但若大面积出现小于60分时也需重新采集并校正



- 若对校正结果满意，打开新生成的校正文件（上例为/tmp/calib.yaml），将下图中的“Camera:”（在angv_noise_var下面）及以下的部分全部复制，并粘贴覆盖旧的校正文件的对应位置。注意：yaml文件对格式要求极其严格，拷贝过程要保证数据和原来一模一样（对齐），否则会出错。

```

g)   - [1, 0, 0]
      - [0, 1, 0]
      - [0, 0, 1]
      gyro_bias: [0, 0, 0]
g)   accel_noise_var: [0.0016, 0.0016, 0.0016]
ac   angv_noise_var: [0.0001, 0.0001, 0.0001]
at Camera:
me   D_T_C_l:
D     - [1, 0, 0, 0]
      - [0, 1, 0, 0]
      - [0, 0, 1, 0]
      - [0, 0, 0, 1]
D     D_T_C_r:
      - [0.9999724, -0.00616791, 0.004147161, 0.03012888]
      - [0.006058447, 0.999646, 0.02590854, 0.0002995041]
      - [-0.004305494, -0.0258827, 0.9996557, 0.0003964022]
      - [0, -0, -0, 1]
ca   cameraK_l:
      - [382.1099, 0, 375.2052]
      - [0, 383.2338, 232.6493]
      - [0, 0, 1]
ca   cameraK_r:
      - [383.5515, 0, 352.2229]
      - [0, 384.6034, 243.0525]
      - [0, 0, 1]
d:   dist_coeff_l:
      - [0.04890894]
      - [-0.1207129]
      - [-0.0009581916]
      - [-0.000267568]
      - [-0.003174605]
      - [0.4985427]
      - [-0.1919671]
      - [-0.03025227]
      dist_coeff_r:
      - [0.955914]
      - [0.1636111]
      - [-1.717751e-05]
      - [-0.0001284909]
      - [0.002328827]
      - [1.405757]
      - [0.4892114]
      - [0.02735064]

```

- 恭喜！至此，整个校正过程已完成。日常使用时注意保护模组，以防磕碰等行为。若今后使用过程发现效果变差，请重复整个校正过程。

预验覆盖

采集完成后，首先我们来预验覆盖率。

- 运行如下命令：

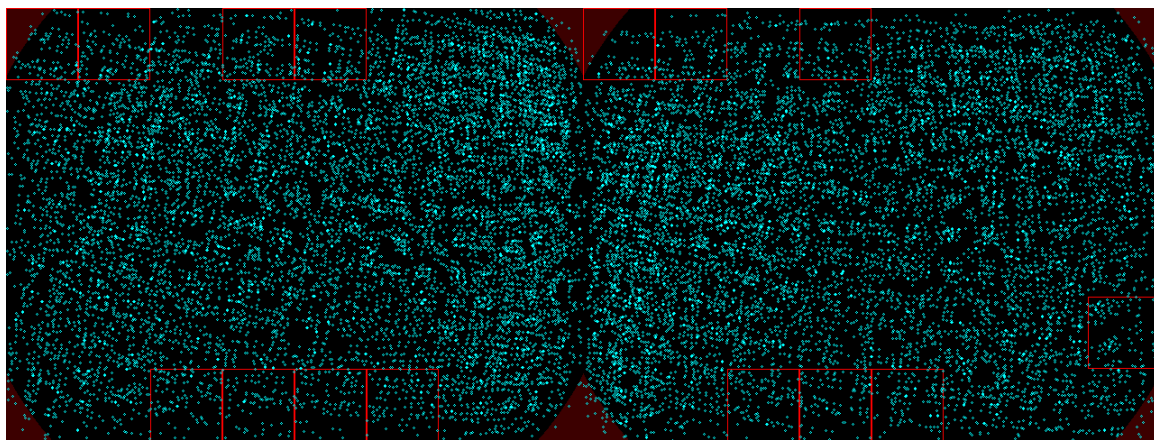
```
./cam_calibration -folder_path /tmp/record1/ -square_size X -show_coverage
```

cam_calibration 为我们预验和校正程序：

- folder_path 为之前采集的图像的目录
- square_size 为上一步中测量的二维码的变长，此处单位为米，例如0.032
- show_coverage 为根据采集的图片快速显示镜头视角的二维码覆盖率

如需查看所有参数及其使用方法，可加 -helpshort

- 运行完后，程序会读取采集的图片，生成覆盖率图片（生成第一张后，按任意键生成第二张）如下：



- 这里需要看点的覆盖率，如果图像采集的好的话（镜头视角的各个部位都被二维码覆盖过），应该看到点均匀地密布在结果图的所有角落
- 若出现某些区域的点不够密，则程序输出的图上会有红色方框（如上图所示），此时需要重复 采集数据 步骤，在该区域补充二维码拍照（无需删除已有照片），直至没有任何红框。
- 再次按任意键退出程序

校正程序

- 覆盖率验证完成后，运行如下命令：

```
./pc_apps/cam_calibration -folder_path /tmp/record1/ -square_size X -save_calib_yaml /tmp/calib.yaml -show_reproj
```

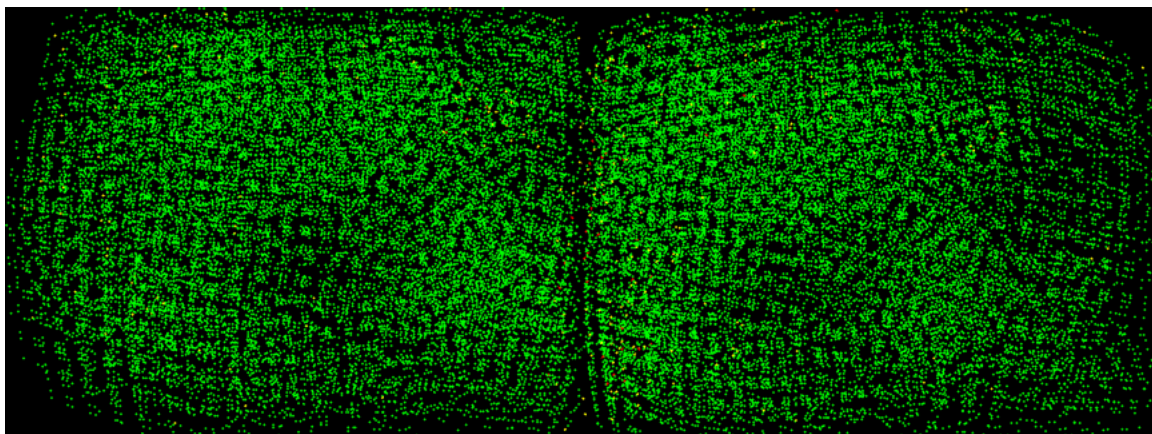
cam_calibration 为我们的校正运算程序：

- folder_path 为之前采集的图像的目录
- square_size 为上一步中测量的二维码的变长，此处单位为米，例如0.032
- save_calib_yaml 指定程序生成的校正文件（yaml格式）的路径和名称
- show_reproj 为显示采集的图像二维码覆盖面

需查看所有参数及其使用方法，可加 -helpshort

- 根据数据采集的质量，校正程序运行时间从3-20分钟不等，完成后可以看到左右镜头的图像采集的二维码覆盖面，如下图。

- 绿色代表效果好的点，黄色代表效果不是很好的点，红色 代表效果很差的点。若黄色和红色点太多，代表采集的照片很差（一般因为晃动、模糊、光照等原因），则需删除原校正文件夹，重新校正
- 除了看点的颜色以外，还需看点的覆盖率，如果图像采集的好的话（镜头视角的各个部位都被二维码覆盖过），应该看到点均匀地密布在结果图上，如下图所示



- 当两张结果图都出现后，代表程序运行结束，按ESC退出

范例程序

准备工作

每次打开新的命令行窗口时，在执行安装包内的其他程序前，需要先执行一次以下命令来设置环境变量（一个窗口只需一次）：

```
source config_environment.sh
```

功能说明

- 在校正完成后，就可以运行我们的范例程序 `./app_tracking`
- `app_tracking` 最基本的功能是SLAM算法，在模组移动过程中能够根据环境的特征点绘制出运动轨迹，并且能够在模组移动到起点时闭环
- `app_tracking` 还具有深度检测的功能，能够展示深度图和障碍物提示
- `app_tracking` 能够根据环境亮度进行自动曝光
- `app_tracking` 能够将走过的路径存储下来，并且能够读取之前的路径，以达到更精确的效果以及实现重定位
- 结合 `./guide_receiver` 程序，能够在已存的路径上进行导航

循迹导航

范例程序`app_tracking`和`guide_receiver`配套使用可以实现循迹导航功能（按录制地图时的路线行走），由`app_tracking`进行轨迹存储、调用，`guide_receiver`负责接收导航信号，源代码均提供在程序包中。

- 轨迹存储命令与记录地图相同：

```
./app_tracking -sensor_type XP2 -cam_calib_path xxxx.yaml -pb_save /tmp/room.pb
```

- 如果需要在轨迹中存储路径点，可在对应的位置按下字母键A、B、C、D、E（最多5个，用于从任意地点走向对应路径点），若同一个路径点的字母被第二次按到，将会开启路径闭合，认为第一次按下的位置和第二次按下的位置处于同一个地点。强制路径闭合有利于帮助建立更准确的轨迹地图。（若不在录制过程中设置路径点，则之后导航时系统默认走完整个轨

迹)

- 录制完轨迹后，按ESC键退出。
- 轨迹导航的使用可分为两种:
- 使用guide_receiver通过UDP接收导航信息
- 使用app_tracking直接对步科Dilili底盘控制导航

☞ 使用guide_receiver进行导航

- 开启另一个命令行窗口，运行

```
./guide_receiver
```

- guide_receiver 是命令行的接收客户端（也是利用traker.h里面的函数实现），通过UDP传输的方式从app_tracking处接收导航信息，并打印在命令行窗口中。
- guide_receiver 用于接收UDP协议的默认端口是8889，IP地址没有限制，收到的信息格式为： guide message from [IP]: [PORT][a, b, c]
- A为当前位置到下一个特征点的距离
- B为导航指示的偏离角度（从-180度到+180度，左负右正，正中间为0度）
- C为导航状态（0为失败，1为正确，2为停止，3为到达目的地结束）。
- 对应的发送端app_tracking命令如下：

```
./app_tracking -sensor_type XP2 -cam_calib_path xxxxx.yaml -pb_load /tmp/room.pb -path_follower walk
```

- 程序启动后，若录制地图时没有设置路径点，则程序会自动进行定位，并按照预设路径进行导航，若为闭合轨迹，则会无限循环直至退出。若录制地图时设置过路径点，则需先按一下所需要行走的路径点对应字母，系统会给出导航，行走至对应点后会进入待机状态。

☞ 使用app_tracking直接控制步科底盘

- 步科Dilili智慧桌是一款采用双轮驱动的实验机器人平台。其控制系统已经集成到了程序包中。
- 首先将步科底盘控制USB线连接到PC端，修改读写权限:

```
sudo chmod a+rw /dev/ttyUSB0
```

- 然后启动app_tracking程序:

```
./app_tracking -sensor_type XP2 -cam_calib_path xxxxx.yaml -pb_load /tmp/room.pb -path_follower robot
```

- 程序启动后，若录制地图时没有设置路径点，则程序会自动进行定位，机器人底盘会按照预设路径行走，若为闭合轨迹，则会无限循环直至退出。若录制地图时设置过路径点，则需先按一下所需要行走的路径点对应字母，机器人底盘会按照导航行走，行走至对应点后会进入待机状态。
- 机器人在行走的过程中，会启动避障模式，即如果有物体进入视觉模组前方0.75米范围内，则机器人会停止行走并停止特征点跟踪以防SLAM结果漂移。

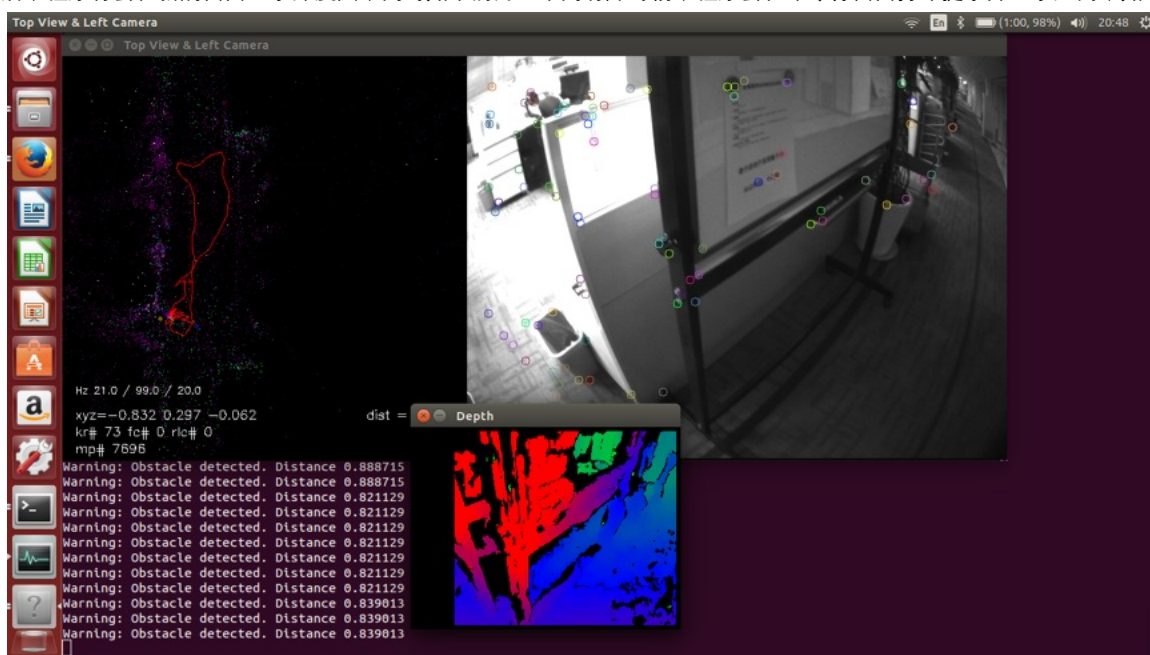
深度图+障碍物提示：

- 开启深度图和障碍物提示，只需要在基本的app_tracking后加-show_depth参数即可

```
./app_tracking -sensor_type XP2 -cam_calib_path xxxxx.yaml -show_depth
```

- 开启后，程序将会在新的窗口显示深度图，同时如果前方一米内有障碍物，程序会在命令行窗口打印提示并显示距离，如下

图。



基本SLAM功能

- 运行：

```
./pc_apps/app_tracking -sensor_type XP2 -cam_calib_path xxxxx.yaml
```

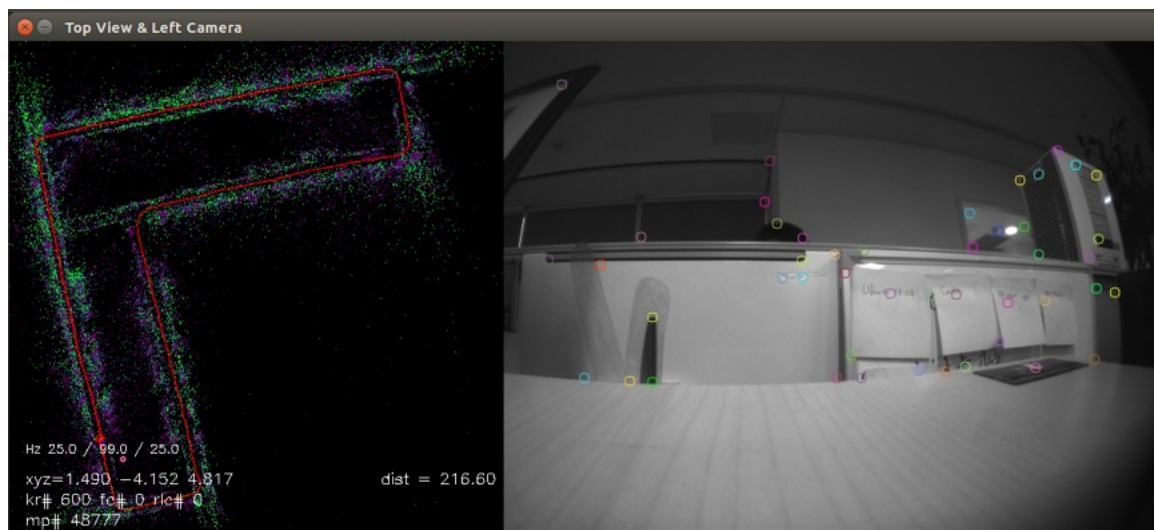
- sensor_type 为模组硬件型号
- cam_calib_path 为校正好的校正文件

如需查看所有参数及其使用方法，可加 -helpshort

- 程序启动后，出现两个窗口如下图所示，此时可以带着模组走动，左窗口显示的红线为运行轨迹的俯视图，右窗口显示左摄像头的呈像，其中的彩色小圆圈为算法标记的特征点。
- 行走过程中程序会针对周围环境的明暗自适应调整曝光值，以达到最佳效果。如果校正文件失效或是不再适用，行走一小段后会在左窗口出现BAD CALIBRATION的警告，此时应重新进行校正。



- 若行走一圈，轨迹回到原点时，程序会尝试闭环（需要几秒钟），闭环后的效果如下图：



识别功能

范例程序app_tracking还具有识别功能，目前支持文字、物体、人脸属性检测、人脸识别，识别是通过在线调用百度智能云端API实现，若需要识别功能，请先去[百度智能云控制台](#)注册，每个账号每天有一定的免费调用额度。

- 注册完后将会得到三串字符：
- id: 用户的应用id:
- ak: 用户的api key
- sk: 用户的api secret
- 在~/XP_release文件夹下创立/keys文件，将上述密钥用以下格式放在文件中：

```
id=xxxx
ak=yyyy
sk=zzzz
```

- 启动识别，运行以下命令：

```
./app_tracking -sensor_type XP2 -cam_calib_path xxxxx.yaml -ocr -face_attribute -face_recognition
```

- -ocr 为激活文字识别
- -face_attribute 为激活人脸属性检测
- -face_recognition 为激活人脸识别
- 请根据需要加上上述识别的选项，程序启动后等约5秒钟待初始化完成后，在键盘按下字母R键，将调用已激活的识别功能。
- 人脸属性检测和人脸识别的结果会显示在一个新弹出的窗口中，文字识别结果会显示在命令行窗口中。每按一次R键程序会并行调用一次所有已激活的API，并在所有结果返回后刷新一次结果（不用担心一直刷流量）
- 因窗口有限，范例程序只显示了部分结果，完整的结果和调用说明请参考范例程序和官方文档（见下页）



路径规划导航

除了循迹导航，我们还提供路径规划导航，与循迹导航不同的是，用户可以通过在录制好的地图上预先设置多个中继点（目前只支持预设轨迹），程序会根据地图来计算任意两个中继点的最短路径，从而生成规划好的路径，然后再由app_tracking读取此路径文件来进行导航。规划步骤有如下几步

- 准备步骤：将之前录好的地图（pb后缀）放到一个文件夹内，假设此文件夹为 /tmp/navigation, 生成的pb为final.pb。注意：为保证地图规划的准确性，录制地图时应遍历所有之后规划时可能走的路径，录制地图时未走过的路径，规划时将不会采用。录制此地图时不添加中继点，中继点会在后续编辑地图时加入。录制地图时，在起点位置附近让镜头360度全方位旋转以记录周围特征点，如有条件尽量录制回环轨迹。

- 运行以下命令，生成waypoints.csv:

```
./dump_map_info -map_pb /tmp/navigation/final.pb >> /tmp/navigation/waypoints.csv
```

- 运行以下命令，根据waypoints.csv生成地图map_occupancy.png和yml文件map_occupancy.yml：

```
./waypoints_generate_map -waypoints_file /tmp/navigation/waypoints.csv -output_folder /tmp/navigation/
```

- 运行以下命令，生成waypoints_graph.txt:

```
./waypoint_graph -waypoints_file /tmp/navigation/waypoints.csv -waypoints_only_mode -map_occupancy_file /tmp/navigation/map_occupancy.png -map_occupancy_specs_file /tmp/navigation/map_occupancy.yml -waypoint_graph_output /tmp/navigation/waypoints_graph.txt
```

- 运行以下命令设置轨迹中继点:

```
./trajectory_maker -waypoints_file /tmp/navigation/waypoints.csv -map_occupancy_file /tmp/navigation/map_occupancy.png -map_occupancy_specs_file /tmp/navigation/map_occupancy.yml -waypoint_graph_file /tmp/navigation/waypoints_graph.txt
```

程序运行后，会弹出以下窗口：



其中左边为可操作区域，右边为程序生成的轨迹。操作方式如下：

- 用鼠标左键单击左边地图轨迹上的任一点，然后按下键盘字母键P，此时会在此处出现蓝色点，如下图所示



- 重复上一步，将设置下一个中继点，并且会在右边的窗口绘制从上一个中继点到此中继点的路径，如下图所示



- 可如此继续若干次，直至所有的中继点都标记完后，按ESC退出
- 因为以上步骤众多，建议在此时确认/tmp/navigation文件夹内包含所有所需文件

- waypoints.csv
- map_oppcupancy.png
- map_oppcupancy.yml
- waypoints_graph.txt
- trajectory_planned.txt

- 运行app_tracking，载入整个/tmp/navigation文件夹

```
./app_tracking -sensor_type XP2 -cam_calib_path xxxxx.yaml -navigation_folder /tmp/navigation -path_follower robot/walk -pb_load /tmp/navigation/final.pb
```

- 注意：若用的是步科底盘，-path_follower 参数后接的是robot，且可以在上述指令后附加-controller PD参数获得更平滑的地盘行走效果（-controller PD 模式下的运动参数在XP/config/algorithm_param.yaml文件中的Navigation字段下的v_max~kp_acc_w行调节）。若是普通的UDP导航模式，-path_follower 后接walk，并且需要加参数-guid_port 8889 -guide_ip 127.0.0.1，在UDP导航模式下请不要加入-controller PD 参数

SLAM的存储和读取地图：

- 软件的另一个功能是记录轨迹和地图，用来给下一次行走提供重定位和导航功能。例如，可以执行一下命令将本次行走的信息记录在 /tmp/room.pb文件中，运行如下命令：

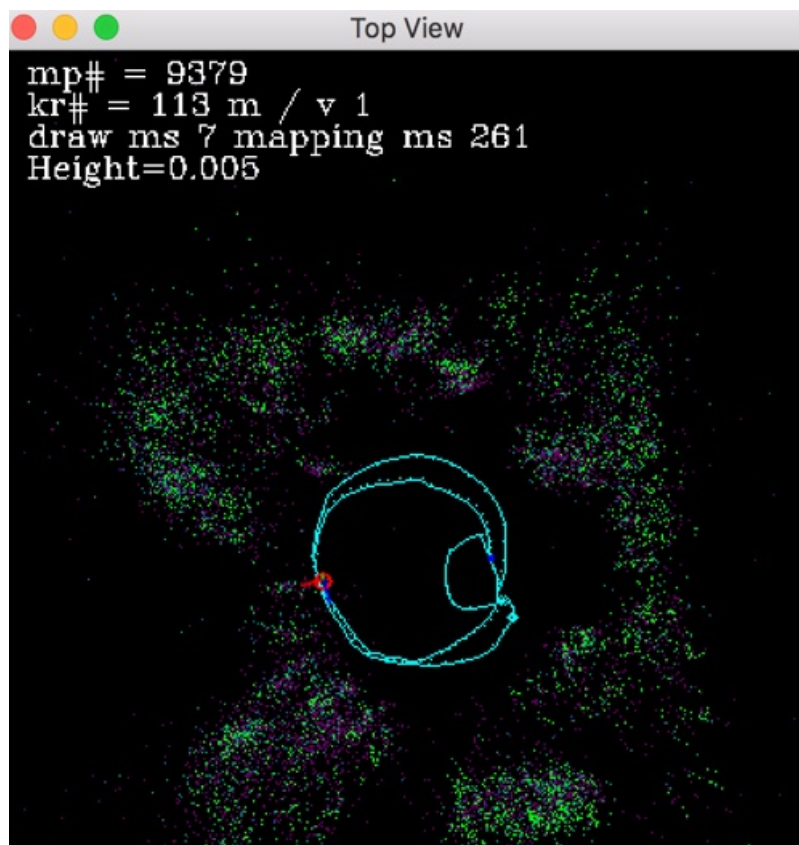
```
./pc_apps/app_tracking -sensor_type XP2 -cam_calib_path xxxxx.yaml -pb_save /tmp/room.pb
```

- -sensor_type 为模组硬件型号
- -cam_calib_path 为校正好的校正文件
- -pb_save 为储存地图的路径和文件名
- 地图的记录是实时的，按ESC键后会自动保存到文件中。
- 如果房间很大，需要离线处理（例如超过2000平米），请联系我们协助操作。

- 记录好的地图可以用来给下一次行走提供重定位和导航功能。如果地图所在文件为/tmp/room.pb，则可以执行：

```
./app_tracking -sensor_type XP2 -cam_calib_path xxxxx.yaml -pb_load /tmp/room.pb
```

- pb_load 为读取地图的路径和文件名
- 地图文件中记录的轨迹会通过浅蓝色线表示，当地图文件被加载后，程序在启动时会自动做重新定位，如下图所示。
- 重新定位完成时，可见到红绿蓝三色坐标出现在地图中，代表照相机位置。如果使用已经记录的地图，程序则只会测量照相机现在的运动，但并不会增加新的轨迹线



编译开发

接口调用

- 程序包还带了SDK并开放了接口供开发者自行调用，所有的头文件都在程序包根目录下的include文件夹内。
- 开发者可以参考app_tracking的源程序app_tracking.cpp 来作为xp_tracker.h内的各个函数的调用范例，以开发自己的程序。或者改动app_tracking.cpp，并按照上述的方式进行编译。

编译程序

- 软件包内除了已经编译好的范例程序以外，还提供了app_tracking的源代码供开发者调试和参考，以及sdk供开发者使用。
- app路径下的源代码都是可以编译的，现在我们来编译app_tracking
 - 首先，我们需要一些额外的库，详情请参考环境配置下的（非必需）
 - 然后运行以下命令行进行编译：

```
cd app
mkdir build
cd build
cmake ..
make
```

- 编译好的 `app_tracking` 将会在 `build` 文件夹里面，用法和程序包自带的一样

常见问题

常见问题

- 我采集了好几次图像，为什么校正时都失败了？
 - 如果重新采集图像，为了避免之前失败图像的影响，需要先删除原先采集的图像所在文件夹，否则只会往里面添加图片
- 我新生成的校正文件不见了？
 - 范例命令中的校正文件存储路径为 `/tmp`，这个目录下的文件会随着机器的重启而清空，校正完后务必及时备份校正文件
- 我运行校正程序的时候出错了？
 - 校正程序 `cam_calibration` 在采集的图像质量很差或者数量不够的时候会出错，此时应删除之前的图像采集文件夹，重新采集图像数据。
 - 二维码图纸需要放置在平坦的平面上，手拿着或者凹凸不平都会造成结果的不准确，建议将图纸贴在一块平坦的白板上。
- 我运行校正程序后显示的点图全都集中在中间？
 - 采集图像时请多采集二维码在镜头成像边缘的情况，一部分的二维码出了边框是可以的。
 - 采集的图片数量尽量多一些，这样能提高校正的精度。
- 我校正好了模组，运行 `app_tracking` 的时候出错了？
 - 是否忘记运行 `source config_environment`？
 - 运行 `ls /dev/video*` 看模组有没有被系统识别，可以尝试插拔模组，或者换个USB口
 - 如果出现了 `core dump`，或者程序崩溃，可在运行 `app_tracking` 时添加参数 `-record_path /tmp/record`，此时会把此次运行的所有帧图片全部存储在对应路径下，然后可以发给我们来跟踪问题。

识别接口

使用机器人视觉SDK调用百度AI平台在线API服务

百度机器人视觉SDK已经集成了百度AI平台的C++ SDK，其中包含了调用相关识别服务在线API的接口能力，您可以包含相关头文件并使用这些函数接口。

🔗 使用方法

🔗 如果您在百度机器人视觉SDK中使用

1. 按需引入相应服务头文件，使用机器人视觉服务应用的 **API Key** 和 **Secret Key** 初始化相应服务的客户端类，使用客户端类调用具体接口，具体使用可以参考**样例代码**部分文档以及SDK中附带的样例代码。

如果您想单独使用C++ SDK

1. 百度AI平台依赖库curl openssl jsoncpp(>1.6.2版本，0.x版本将不被支持)
2. 编译工程时添加 C++11 支持 (gcc/clang 添加编译参数 -std=c++11), 添加第三方库链接参数 -lcurl, -lcrypto, -ljsoncpp。
3. 按需引入相应服务头文件，使用机器人视觉服务应用的 **API Key** 和 **Secret Key** 初始化相应服务的客户端类，使用客户端类调用具体接口

百度AI能力头文件列表

百度AI调用接口以.h文件形式集成于SDK中，实现源码也都包含在.h文件中，开发者只需要引入相应头文件即可使用这些服务提供的接口。

```
XP
├── util
│   └── aip
│       ├── base
│       │   ├── base.h          // 请求客户端基类
│       │   ├── base64.h       // base64加密相关类
│       │   ├── http.h         // http请求封装类
│       │   └── utils.h        // 工具类
│       ├── ocr.h              // 文字识别
│       ├── image_classify.h    // 图像识别
│       ├── face.h             // 人脸识别
│       └── speech.h           // 语音识别，语音合成
```

接口说明

包含在SDK中的百度AI平台在线API能力有文字识别，图像识别，人脸识别，语音识别以及语音识别。各个API的调用方式以及调用参数，返回详细数据请参考相应的百度AI平台API C++文档：

- [文字识别](#)
- [图像识别](#)
- [人脸识别](#)
- [语音识别](#) [语音合成](#)

建立机器人视觉应用

使用百度AI平台的在线API之前，首先需要在 [百度智能云控制台](#) 中建立机器人视觉的应用。

1. 请先前往百度AI开放平台下载机器人视觉SDK [SDK下载](#)
2. 注册账号并登录，并创建一个**机器人视觉服务应用** [百度智能云控制台](#)
3. 查看所创建应用的 **API Key** 和 **Secret Key**，他们用途是获取百度AI平台在线API服务授权的access_token，在线授权相关问题可以参考 [授权机制](#)。