

Apollo&SVL联合仿真（二）标定

1 简介

在实车中，由于安装误差以及传感器各自的参数不同导致这些传感器参数均需要我们进行标定来获取，构建出一个符合汽车自身的TF树。相机的内参标定包含畸变系数以及焦距大小，以深蓝学院的小车为例，传感器之间的外参包含Lidar-Imu, Camera到Lidar之间的变换。拿到这些外参后，方便我们后续的定位建图以及感知融合等，以下图为例，这是外参好坏所建立的NDT地图的差异，场景-腾讯众创空间A座停车场附近。

1 启动标定工具

深蓝学院公布的开源标定工具[calibration kit](#)，其所需要的配置环境可能与自身电脑并不十分匹配，导致需要额外的环境配置问题，这里通过Docker将标定工具所需的环境进行打包，在Docker建立的容器中即可运行标定工具。

1.1 安装Docker

请参考[Docker官方文档](#)完成 Docker 环境的安装

安装完成后, 请务必进行如下操作, 以保证可以无脑跟随后续文档进行操作:

将当前用户加入Docker Group

为了能在非 `sudo` 模式下使用 `Docker`, 需要将当前用户加入 `Docker Group`.

1. 执行命令:

```
2. sudo usermod -aG docker $USER
```

1.2 安装Docker-Compose

`Docker-Compose` 是基于Docker解决方案的Orchestrator.

请参考[Docker Compose官方文档](#)完成 `Docker-Compose` 环境的安装

1.3 获取镜像

在安装完成 `Docker` 以及 `Docker-Compose` 之后，需要拉取所需镜像.

注意: 执行第1条命令时，**需要输入密码!!!需要输入密码!!!需要输入密码!!!**, 密码请看!!!

```
# 1. login to Docker registry
docker login --username=Jiahao031008
# 2. then download images:
docker pull jiahao031008/calib-tools:v0
```

1.4 克隆项目

```
$ git clone https://github.com/calibtoolkit/calibration_kit.git
$ cd calibration_kit
```

1.5 启动实例

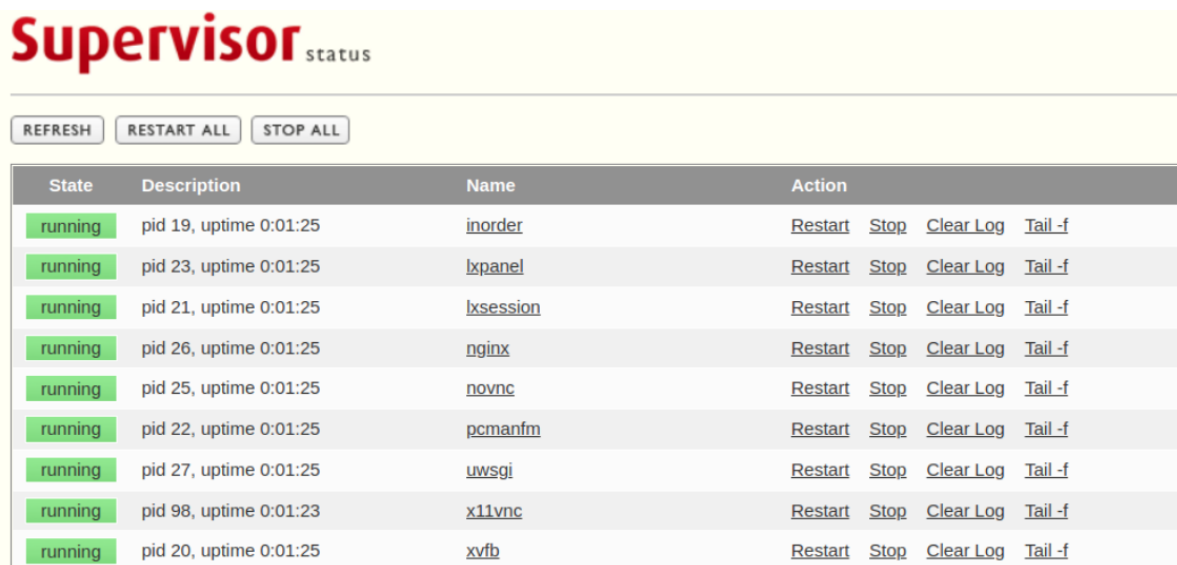
在当前Repo 根目录下(即`docker-compose.yml`所在的那个文件夹), 启动Terminal, 执行命令, 启动 Docker Workspace:

```
docker-compose down && docker-compose up
```

1.6 Service Health Check

然后打开 Chrome 浏览器, 访问URL `http://localhost:49001/`, 默认账号/密码为 `sensorfusion/sensorfusion`, 确保所有服务成功启动.

若所有服务成功启动, 系统状态如下图所示:



The image shows the Supervisor web interface. At the top, there's a header with the Supervisor logo and 'status'. Below the header are three buttons: 'REFRESH', 'RESTART ALL', and 'STOP ALL'. The main content is a table with four columns: 'State', 'Description', 'Name', and 'Action'. There are 10 rows of services, all in a 'running' state. Each row includes a green 'running' status, a description with PID and uptime, the service name, and a set of action links: 'Restart', 'Stop', 'Clear Log', and 'Tail -f'.

State	Description	Name	Action
running	pid 19, uptime 0:01:25	inorder	Restart Stop Clear Log Tail -f
running	pid 23, uptime 0:01:25	lxpanel	Restart Stop Clear Log Tail -f
running	pid 21, uptime 0:01:25	lxsession	Restart Stop Clear Log Tail -f
running	pid 26, uptime 0:01:25	nginx	Restart Stop Clear Log Tail -f
running	pid 25, uptime 0:01:25	novnc	Restart Stop Clear Log Tail -f
running	pid 22, uptime 0:01:25	pcmanfm	Restart Stop Clear Log Tail -f
running	pid 27, uptime 0:01:25	uwsgi	Restart Stop Clear Log Tail -f
running	pid 98, uptime 0:01:23	x11vnc	Restart Stop Clear Log Tail -f
running	pid 20, uptime 0:01:25	xvfb	Restart Stop Clear Log Tail -f

1.6 访问工作空间

接着在 Chrome 浏览器中, 访问URL `http://localhost:40081/vnc.html?autoconnect=1&autoscale=1&quality=16`, 默认登录密码为 `shen1an`, 访问 Docker Workspace

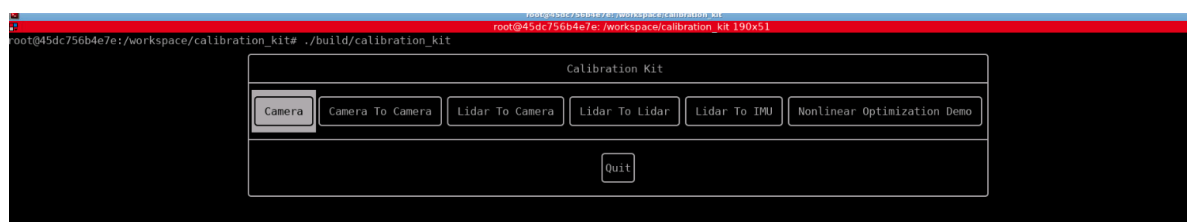
1.7 编译标定程序

在容器中打开终端

```
$ cd /worksapce/calibration_kit/  
$ cmake -S . -B build -DCMAKE_BUILD_TYPE=Debug  
$ cmake --build build --parallel 4
```

1.8 运行标定工具

```
$ ./build/calibration_kit
```



1 Camera内参标定

在SVL仿真中，由于无法进行标定，这里以真实环境中采集的数据为例对camera内参标定工具的使用进行介绍，标定板采用的是7×6_50×50大小的棋盘格，采集包含各种人脸造作以及远近不等的旋转图像。

1.1 图像采集

启动ros图像采集程序，按 `空格` 进行图像采集，图像文件保存在

1.1.1 启动camera

```
$ cd /workspace/catkin_ws && catkin_make
$ roslaunch usb_cam usb_cam.launch
```

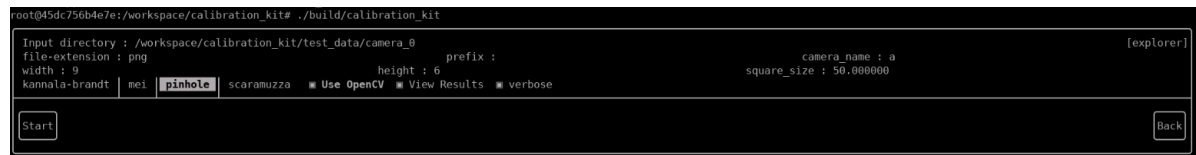
1.1.2 运行图像采集程序

```
# 新建终端
$ roslaunch cam_collect cam_collect.launch
```

按 `空格` 键即可将当前帧图像.png按顺序保存在data/camera_0文件夹下

1.2 开始标定

根据窗口提示，确定 `Input_directory`, `file-extension`, `camera_name`, `width:7`, `height:6`, `pinhole`, 点击 `start` 开始进行标定



```
[a] # INFO: Final reprojection error: 0.540 pixels
[a] # INFO: Camera Parameters:
      model_type PINHOLE
      camera_name a
      image_width 1920
      image_height 1080
Distortion Parameters
      k1 -0.554392
      k2 0.270977
      p1 -0.000271773
      p2 0.00529771
Projection Parameters
      fx 1965.28
      fy 1975.21
      cx 862.282
      cy 472.485

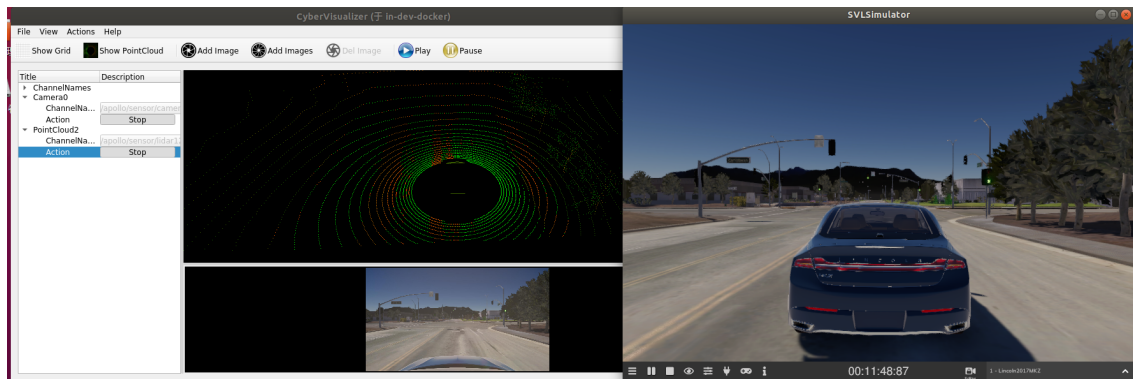
# INFO: Calibration took a total time of 11.784 sec.
# INFO: Wrote calibration file to a_camera_calib.yaml
```



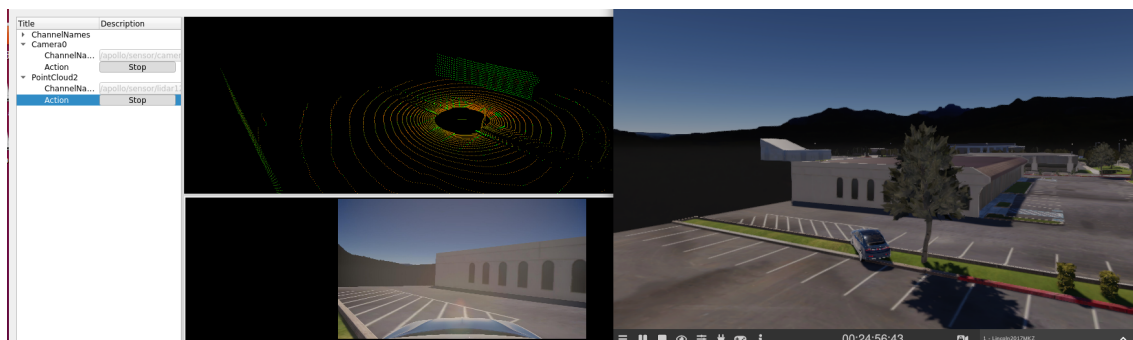
2 Lidar-Camera标定

2.1 传感器数据收集

- 参照文档（一）启动apollo和svl_simulator在cyber_visualizer下检查数据接受是否正常



- 寻找标定场地，将汽车开至有墙角的建筑物附近来进行标定，如下图所示



- 录制数据包，录制几秒钟后Ctrl+C结束，*.record.*文件将保存在apollo根目录下

```
cyber_recorder record -a
```

在 `\apollo\data\bag` 目录下新建 `calibration` 文件夹，将录制好的bag包文件粘贴进来。

2.2 数据解析

2.2.1 图像数据解析

数据解析用到了 `apollo` 自带的解析程序，将我们的录制包中的 `image` 保存成 `jpeg` 格式的图片

- 修改数据解析的参数配置文件，在 `/apollo/modules/tools/record_parse_save` 文件目录下，打开 `parser_params.yaml` 文件
 - `filepath`: 改为我们存放数据包的文件夹 `/apollo/data/bag/calibration`
 - `parse`: 选为 `camera`
 - `camera` 的 `channel_name`: 选为我们实际压缩图像的 `topic` 名，如 `/apollo/sensor/camera/front_6mm/image/compressed`



- 执行数据解析程序

```
./bazel-bin/modules/tools/record_parse_save/record_parse_save
```

解析完成后，会将图像保存在我们数据包的上一级目录下文件夹下 `data_camera_6mm_front`。

2.2.2 Lidar数据解析

lidar的数据解析需要借助MSF下的数据解析程序，由于仿真数据发过来的数据跟其消息的大小不一致，我们需要对apollo内部程序稍作更改即可

- 打开 `/apollo/modules/localization/msf/local_tool/data_extraction/pcd_exporter.cc`，在程序的第61行 `void PCDExporter::WritePcdFile(const std::string &filename, const drivers::PointCloud &msg)` 函数内，将 `cloud.width`, `cloud.height` 做如下改写。

```
bool bsimulator = true;
if(bsimulator == true)
{
    std::cout << "msg.size: " << msg.point_size() << "msg.width() " <<
msg.width() << " msg.height() " << msg.height() << std::endl;
    cloud.width = msg.point_size();
    cloud.height = 1;
}
else{
    cloud.width = msg.width();
    cloud.height = msg.height();
}
```

重新编译apollo程序

```
./apollo.sh build_opt localization
```

- 添加数据解析脚本

在/apollo/scripts文件夹下新建 `lidar_parse.sh` 脚本文件，copy以下程序

```
#!/bin/bash
if [ $# -lt 3 ]; then
    echo "Usage: msf_simple_map_creator.sh [records folder][output folder]
[extrinsic_file] [lidar_type]"
    exit 1
fi

DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
cd "${DIR}/.."

source "${DIR}/apollo_base.sh"

GNSS_LOC_TOPIC="/apollo/localization/msf_gnss"
LIDAR_LOC_TOPIC="/apollo/localization/msf_lidar"
FUSION_LOC_TOPIC="/apollo/localization/pose"
ODOMETRY_LOC_TOPIC="/apollo/sensor/gnss/odometry"

GNSS_LOC_FILE="gnss_loc.txt"
LIDAR_LOC_FILE="lidar_loc.txt"
FUSION_LOC_FILE="fusion_loc.txt"
ODOMETRY_LOC_FILE="odometry_loc.txt"

IN_FOLDER=$1
OUT_MAP_FOLDER=$2
EXTRINSIC_FILE=$3
LIDAR_TYPE=${4:-lidar128}

PARSED_DATA_FOLDER="$OUT_MAP_FOLDER/parsed_data"
CLOUD_TOPIC="/apollo/sensor/lidar128/compensator/PointCloud2"

function data_exporter() {
    local BAG_FILE=$1
    local OUT_FOLDER=$2
    /apollo/bazel-
bin/modules/localization/msf/local_tool/data_extraction/cyber_record_parser \
    --bag_file $BAG_FILE \
    --out_folder $OUT_FOLDER \
    --cloud_topic $CLOUD_TOPIC \
    --gnss_loc_topic $GNSS_LOC_TOPIC \
    --lidar_loc_topic $LIDAR_LOC_TOPIC \
    --fusion_loc_topic $FUSION_LOC_TOPIC \
    --odometry_loc_topic $ODOMETRY_LOC_TOPIC
}

function poses_interpolation() {
    local INPUT_POSES_PATH=$1
    local REF_TIMESTAMPS_PATH=$2
    local EXTRINSIC_PATH=$3
    local OUTPUT_POSES_PATH=$4
    /apollo/bazel-
bin/modules/localization/msf/local_tool/map_creation/poses_interpolator \
    --input_poses_path $INPUT_POSES_PATH \
    --ref_timestamps_path $REF_TIMESTAMPS_PATH \
    --extrinsic_path $EXTRINSIC_PATH \
    --output_poses_path $OUTPUT_POSES_PATH
}
```

```

}

cd $IN_FOLDER
mkdir -p $OUT_MAP_FOLDER
mkdir -p $PARSED_DATA_FOLDER
for item in $(ls -l *.record* | awk '{print $9}'); do
    SEGMENTS=$(echo $item | awk -F'.' '{print NF}')
    DIR_NAME=$(echo $item | cut -d . -f ${SEGMENTS})
    DIR_NAME="${PARSED_DATA_FOLDER}/${DIR_NAME}"
    mkdir -p ${DIR_NAME}

    data_exporter "${item}" "${DIR_NAME}"
    poses_interpolation "${DIR_NAME}/pcd/${ODOMETRY_LOC_FILE}"
"${DIR_NAME}/pcd/pcd_timestamp.txt" "${EXTRINSIC_FILE}"
"${DIR_NAME}/pcd/corrected_poses.txt"

done

echo "Done."

```

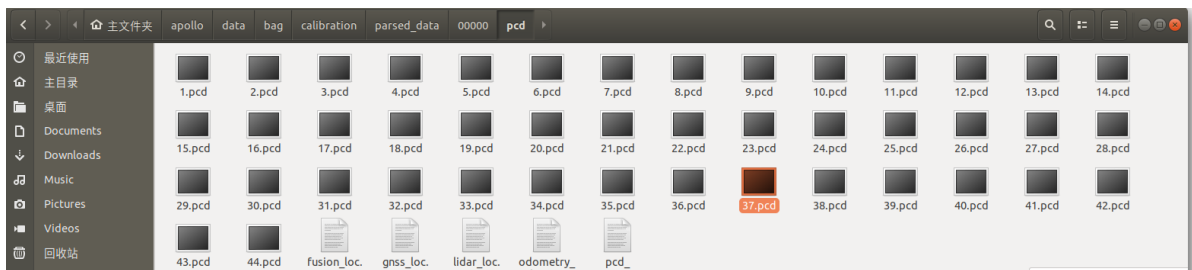
- 运行数据解析脚本文件

```

bash scripts/lidar_parse.sh /apollo/data/bag/calibration
/apollo/data/bag/calibration
/apollo/modules/calibration/data/Lincoln2017MKZ/velodyne_params/velodyne128_n
ovatel_extrinsics_example.yaml lidar128

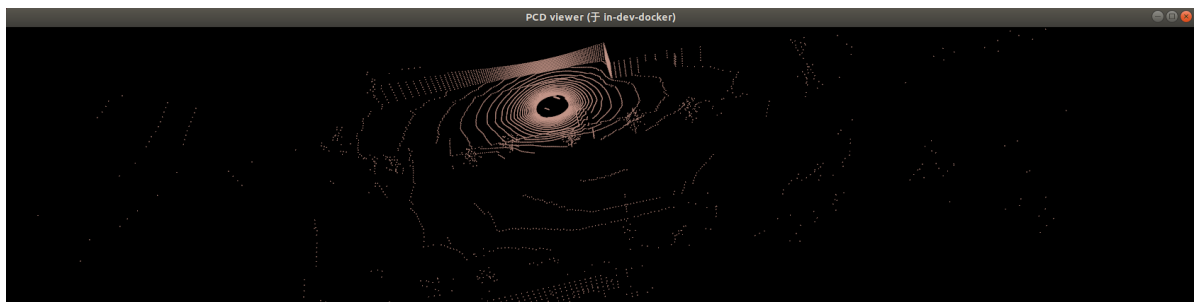
```

在 calibration 文件夹下会生成 parsed_data，里面存放了Lidar 点云的pcd文件，用于进行标定。



在apollo的docker容器内运行pcl_viewer对点云数据进行查看

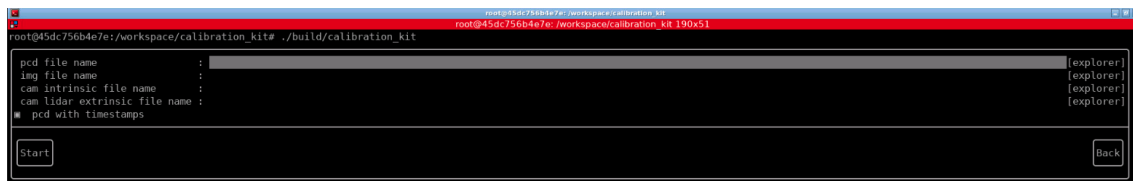
```
pcl_viewer data/bag/calibration/parsed_data/00000/pcd/1.pcd
```



2.2.3 lidar-camera标定

由于数据采集过程中，我们是对着墙角静止采集，故我们从解析的 `image` 和 `lidar_pointcloud` 中任选一帧用于标定，标定工具深蓝学院开源的标定工具进行标定，其基本原理为将Lidar采集的点云通过 Lidar->Imu之间的外参进行变换后，经camera的内参变换投影至图像上，观察点云与图像之间的匹配度。

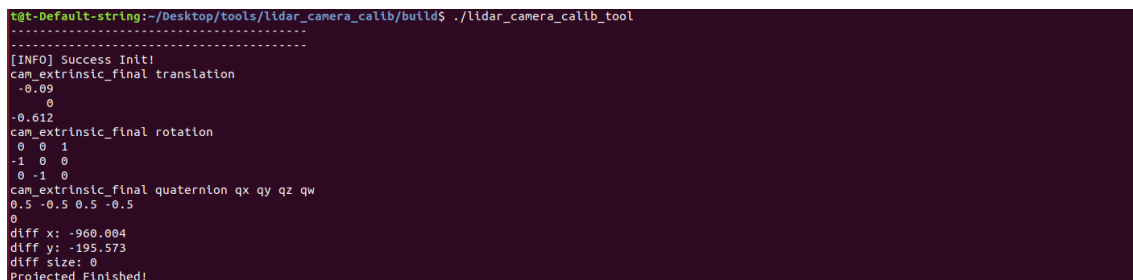
- 任选一帧采集数据，拷贝至标定工具下的 `test` 文件夹下，并重命名为 `1.jpeg` 和 `1.pcd`
- 将 `/apollo/modules/calibration/data/Lincoln2017MKZ/camera_params/` 下的相机参数拷贝至 `test` 文件夹下
- 在标定工具窗口点击 `Lidar To Camera` 后，如下图：



通过点击 `explorer` 确定lidar和image的相关文件地址后，点击start



如图所示，Lidar的点云强度到墙的边为止，我们可以拿墙的边界和窗户在作为他们之间的参考，来进行对选旋转的 `yaw`、`pitch`、`row` 以及平移 `x`、`y`、`z` 进行调节，最终标定结果会在终端进行打印。



3 Lidar-Gnss标定

Lidar到Ins的标定，同样可以理解为点云拼接的过程，我们将代码写

在 `/apollo/modules/localization/msf/local_tool/map_creation/` 目录下，大约200行的一个程序 `lidar_camera_calib.cc`

3.1 传感器数据收集

- 参照文档（一）启动apollo和svl_simulator在cyber_visualizer下检查数据接受是否正常
- 寻找标定场地，将汽车开至有墙角的建筑物附近来进行标定，如下图所示
- 绕建筑物墙体周围转一圈，进行数据采集


```
cyber_recorder record -a -k /apollo/sensor/camera/front_12mm/image/compressed  
/apollo/sensor/camera/front_6mm/image/compressed -i 600
```

3.2 数据解析

- 在 `/apollo/data/bag/` 目录下新建 `lidar_ins_calibration` 文件夹，将录制的数据包粘贴进去。
- 采用2.3.2的lidar数据解析的脚本文件进行数据解析来获取点云的pcd文件已经通过插值来获得对应的位姿。

```
bash scripts/lidar_parse.sh /apollo/data/bag/calibration  
/apollo/data/bag/lidar_ins_calibration lidar128
```

3.3 Lidar-Ins标定

- 在 `apollo` 的 `docker` 容器内运行我们的标定工具

```
./bazel-bin/modules/localization/msf/local_tool/map_creation/lidar_ins_calib --  
pcd_folder /apollo/data/bag/lidar_ins_calibration/parsed_data/000000/pcd --  
pose_file  
/apollo/data/bag/lidar_ins_calibration/parsed_data/000000/pcd/corrected_poses.txt  
--extrinsic_path  
/apollo/modules/calibration/data/Lincoln2017MKZ/velodyne_params/velodyne128_novate  
l_extrinsics_example.yaml
```

note：这里有用到IMU进行插值，IMU发过来的measurement_time存在延迟的情况，不过还好，在进行插值的时候采用的是odometry的时间戳，他的时间戳没有问题，大家放心使用

在标定过程中，建议大家最好是绕着一堵墙进行旋转，目的就是在进行标定过程中，将墙体作为参照物，尽可能的将墙体的点云拼的越薄越好，成一条线。在实车测试中我们可以通过尺子进行测量来他们之间的平移变换，在标定工具中来对旋转进行标定，主要是yaw角。

标定完成后，结果会在终端进行显示。

```
Timu_lidar translation
0.26792
0.980729
2.312
Timu_lidar rotation
-6.85389e-08      -1  0.000785398
      1  2.22045e-16  8.72665e-05
-8.72664e-05  0.000785398      1
Timu_lidar quaternion qx qy qz qw
0.000246827 0.000308534 0.707107 0.707107
```