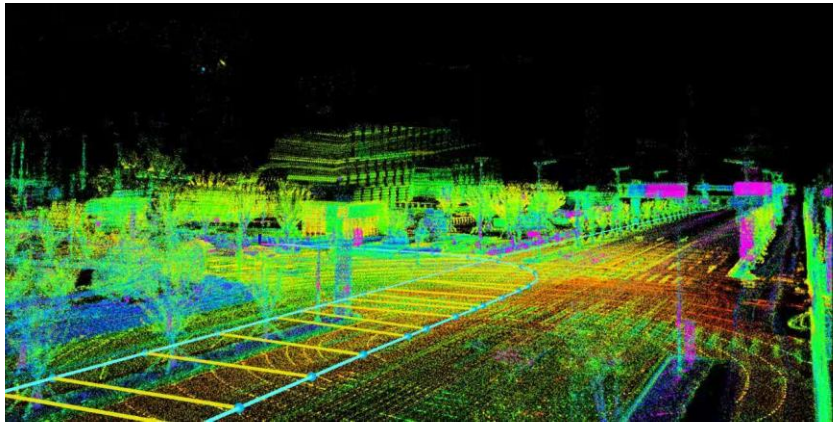
微信扫码加入星球

自动驾驶中实战课之多相机间的同步实战

Camera + LiDAR + Radar + IMU



主　讲　人：初　　　　　心

公　众　号：3D 视 觉 工 坊

内容

1.配置内容均是在docker 下进行，在docker下/home/下：

```
root@6dc69d3505b5:/home# ls
3rdparty  workspace
root@6dc69d3505b5:/home#
```

```
root@6dc69d3505b5:/home/workspace# ls
ad_sensor_fusion  data
root@6dc69d3505b5:/home/workspace#
```

2. 下载网盘数据practice_1_1_multi_camera_sync和3rdparty配置如下：

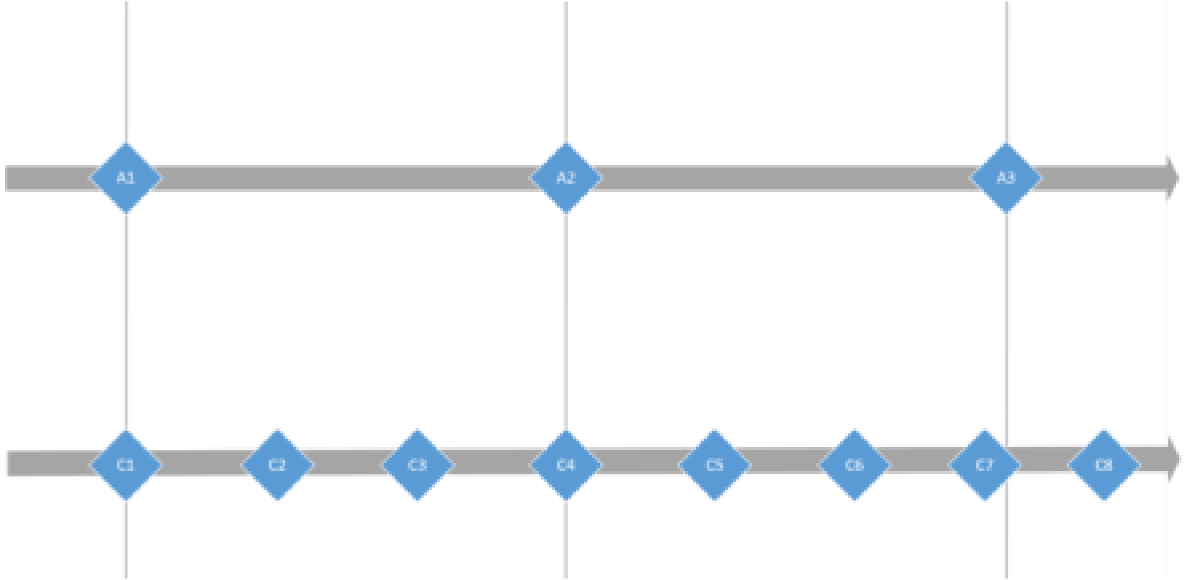
文件夹3rdparty, 这个opencv版本是为了特征点提取专门编译的，同时与其同级放置的还有workspace工作空间，在workspace下放置ad_sensor_fusion和data，后面我们的数据统一放置在data中，也请各位将本次课提供的数据文件夹practice_1_1_multi_camera_sync放置在该目录下：

```
practice_0_2_icp_pointclouds  practice_1_1_multi_camera_sync
root@6dc69d3505b5:/home/workspace/data# ls
practice_0_2_icp_pointclouds  practice_1_1_multi_camera_sync
root@6dc69d3505b5:/home/workspace/data#
```

3. 重新pull代码，更新到最新版本，查看log，确认添加了camera_camera_sync，修改其下面CMakeLists.txt：

```
message(STATUS "EIGEN3_VERSION: ${EIGEN3_VERSION}")
# 这里修改为自己的opencv路径
set(OpenCV_DIR "/home/3rdparty/share/OpenCV")
find_package(OpenCV REQUIRED)
message(STATUS "    libraries: ${OpenCV_LIBS}")
```

直接配准法，适合帧率具有整数倍数关系的传感器之间，以频率低的为基准，找出时间戳对应的数据即可。如右图，这种方法误差比较大，而且要求传感器之间的帧率是**整数倍**



遍历左右相机的文件夹，获取左右相机的文件队列。

以左相机图像序列中的顺序为基准，获取左相机序列中的文件名，并转换为对应的时间，然后与右相机图像序列中的文件对应时间做差，找出所有右侧相机序列中与左侧相机序列中的基准图像时间差小于100毫秒的图像。

采用SSIM相似性比较方法（这里前提是左右相机存在共视区），选取结构相似度较大的作为匹配对象（即验证过程）

重复执行，直到找出所有配对的图像

遍历左右相机的文件夹，获取左右相机的文件队列。

SSIM (Structural SIMilarity) 结构相似性

$SSIM$ 公式基于样本 x 和 y 之间的三个比较衡量：亮度 (luminance)、对比度 (contrast) 和结构 (structure)。

$$l(x,y)=\frac{2\mu_x\mu_y+c_1}{\mu_x^2+\mu_y^2+c_1}$$

$$c(x,y)=\frac{2\sigma_x\sigma_y+c_2}{\sigma_x^2+\sigma_y^2+c_2}$$

$$s(x,y)=\frac{\sigma_{xy}+c_3}{\sigma_x\sigma_y+c_3}$$

一般取 $c_3=c_2/2$。

- μ_x 为 x 的均值
- μ_y 为 y 的均值
- σ_x^2 为 x 的方差
- σ_y^2 为 y 的方差
- σ_{xy} 为 x 和 y 的协方差
- $c_1=(k_1L)^2, c_2=(k_2L)^2$ 为两个常数，避免除零
- L 为像素值的范围，2^B-1
- $k_1=0.01, k_2=0.03$ 为默认值

那么

$$SSIM(x,y)=[l(x,y)^\alpha\cdot c(x,y)^\beta\cdot s(x,y)^\gamma]$$

将 α,β,γ 设为 1，可以得到

$$SSIM(x,y)=\frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)}$$

每次计算的时候都从图片上取一个 $N\times N$ 的窗口，然后不断滑动窗口进行计算，最后取平均值作为全局的 SSIM。

二、时间同步-SSIM相似度度量


```cpp
double CameraCameraSync::evaluateImageTimeStampSync(cv::Mat orgImage, cv::Mat dstImage)
{
    //这里采用SSIM结构相似性来作为图像相似性评判
    double C1 = 6.5025, C2 = 58.5225;
    int width = orgImage.cols;
    int height = orgImage.rows;

    int width2 = dstImage.cols;
    int height2 = dstImage.rows;

    double mean_x = 0;
    double mean_y = 0;
    double sigma_x = 0;
    double sigma_y = 0;
    double sigma_xy = 0;
    for (int v = 0; v < height; v++)
    {
        for (int u = 0; u < width; u++)
        {
            mean_x += orgImage.at<uchar>(v, u);
            mean_y += dstImage.at<uchar>(v, u);

        }
    }
    mean_x = mean_x / width / height;
    mean_y = mean_y / width / height;
    for (int v = 0; v < height; v++)
    {
        for (int u = 0; u < width; u++)
        {
            sigma_x += (orgImage.at<uchar>(v, u) - mean_x)* (orgImage.at<uchar>(v, u) - mean_x);
            sigma_y += (dstImage.at<uchar>(v, u) - mean_y)* (dstImage.at<uchar>(v, u) - mean_y);
            sigma_xy += std::abs((orgImage.at<uchar>(v, u) - mean_x)* (dstImage.at<uchar>(v, u) - mean_y));
        }
    }
    sigma_x = sigma_x / (width*height - 1);
    sigma_y = sigma_y / (width*height - 1);
    sigma_xy = sigma_xy / (width*height - 1);
    double molecule = (2 * mean_x*mean_y + C1) * (2 * sigma_xy + C2);
    double denominator = (mean_x*mean_x + mean_y * mean_y + C1) * (sigma_x + sigma_y + C2);
    double ssim = molecule / denominator;
    return ssim;
}
```

```
void CameraCameraSync::getImageTimeStamp(std::string oriDirName, std::string dstDirName)
{
    //采用该函数遍历获得得队列不是顺序的，正好适合采用时间距离最近法来匹配
    getFiles(oriDirName, oriImageLists_);
    getFiles(dstDirName, dstImageLists_);
    if(oriImageLists_.size() != dstImageLists_.size())
    {
        std::cout << "the two image lists not equal!" << std::endl;
        ROS_ERROR_STREAM("the two image lists not equal!");
        return;
    }
}
```

```cpp
std::vector<std::pair<std::string, std::string> > CameraCameraSync::imageTimeStampSyncFuncion()
{
    std::vector<std::pair<std::string, std::string> > syncPairLists;

    double timeDifference;
    for(auto baseFileNames : oriImageLists_)
    {
        double maxSSIM = 0;
        std::string anchorFilenames;
        double baseImageTime = getbaseTime(baseFileNames, "png");

        for(auto candidateFileNames : dstImageLists_)
        {
            double candidateImageTime = getbaseTime(candidateFileNames, "png");
            timeDifference = std::abs(baseImageTime - candidateImageTime);
            if(timeDifference <= 0.1) // 100ms
            {
                cv::Mat orgImage = cv::imread(baseFileNames, cv::IMREAD_GRAYSCALE);
                cv::Mat dstImage = cv::imread(candidateFileNames, cv::IMREAD_GRAYSCALE);
                if( !orgImage.data || !dstImage.data )
                {
                    std::cout<< " --(!) Error reading images " << std::endl;
                    break;
                }
                double ssim = evaluateImageTimeStampSync(orgImage, dstImage);
                if (ssim > maxSSIM)
                {
                    maxSSIM = ssim;
                    anchorFilenames = candidateFileNames;
                }
            }
        }
        if(maxSSIM <=0){ continue;}
        std::pair<std::string, std::string> syncPair(std::make_pair(baseFileNames, anchorFilenames));
        syncPairLists.push_back(syncPair);
        std::cout << " Get the "<< baseFileNames << " time sync file is " << anchorFilenames << " and ssim is " << maxSSIM << std::endl;
    }

    return syncPairLists;
}
```

四、相机间的空间对齐

输入左右两幅图像

↓

分别提取特征点

↓

左右特征点匹配

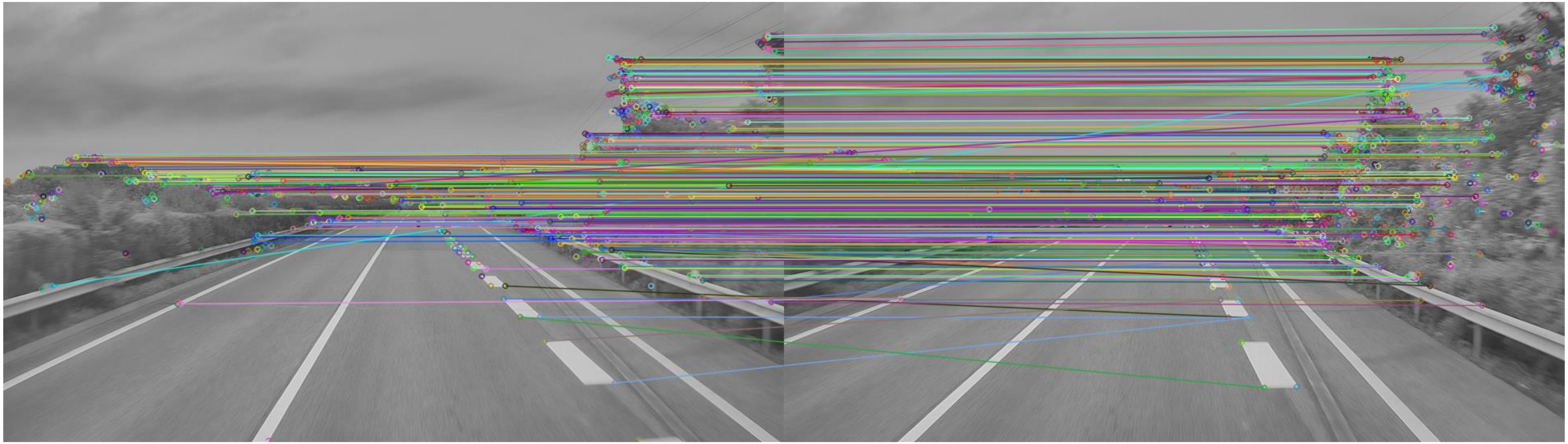
↓

利用左右匹配好的特征点构建非线性最小二乘问题，优化相机间的pitch 和roll角

↓

输出优化后的pitch和roll角

```
bool CameraCameraSync::synchronizePitchRoll(cv::Mat img_left, cv::Mat img_right)
{
    if(!img_left.data || !img_right.data )
    {
        ROS_ERROR_STREAM("no image data!");
        return false;
    }

    std::vector<cv::Point2f> left_pts, right_pts;
    findMatchPoints(img_left, img_right, left_pts, right_pts);
    std::cout << "find match points:size: left:" << left_pts.size() << " right: " << right_pts.size() << std::endl;

    // solve pitch and roll between cameras
    vector<vector<Point2f> > data = {left_pts, right_pts};
    Eigen::VectorXd x(2);
    x << 0., 0.;

    MeanFunctor functor(data);
    Eigen::NumericalDiff<MeanFunctor> num_diff(functor, 1e-6);

    Eigen::LevenbergMarquardt<Eigen::NumericalDiff<MeanFunctor>, double> lm(num_diff);
    int info = lm.minimize(x);

    std::cout << "current result: pitch & roll: " << x[0]/PI*180 << " " << x[1]/PI*180 << endl;

    pitch_cache_.push_back(x[0]);
    roll_cache_.push_back(x[1]);

    return true;
}
```



1. 请在camera_camera_sync文件下代码中添加合适的代码生成上述图像。
2. 调整合适的阈值使得左右特征点匹配相对较好。

购买该课程请扫描二维码

微信扫码加入星球

感谢聆听

Thanks for Listening