思路一

面向动静态物体极目标的视觉SALM

设定好物体的属性：

车 人 自行车之类的为动态物体

路标，广告牌，路灯之类的为静态物体

正常做法都是将图像进行语义分割后，只取静态物体上的特征点进行定位和建图

但是如果是在地下停车场，或者路边停放的有静止的车的情况下，先验信息告诉我们不能用车上边的特征点，这样的话就有点浪费，会增加跟踪失败的可能性

如此，增加运动一致性检测就非常有必要

因此，在语义分割之后，进行一个运动一致性检验，目的是利用更多的静态特征点完成定位

（可以动态物体设置为 高动态 中动态 低动态） 如果判定该物体为动态物体了，通过运动一致性检验看是否为低动态物体，若是低 可以用里边的特征点

（可以将特征点提取加速融合进去，）

思路二

结合yolo 跟踪线程基于关键目标 使用光流法追踪代替特征点匹配 理论上是可以加速的，前提是动态物体的剔除部分一定要做好，否则很容易产生漂移

（对关键目标做一个特征点均匀化）

思路三

结合yolo先用静态特征点完成完成位姿初估计，然后利用运动一致性和几何约束，将动态框里的静态点拿出来，与之前的静态点一起对位姿进行优化 如图模型所示：

图示

描述已自动生成

用yolo检测两种属性的框 静态框，动态框，设两个个阈值

比如静态框里需要判定百分之80以上的特征点为动态点，才能认定 为动态物体

而动态属性的框 只需要有百分之40以上就可以认为是动态

判断静态也是如此

最终思路

突然有了一个感觉还不错的思路，有一篇论文是基于静态目标做的(皮家豪)，但是我有一个疑问，如果初始化关键目标达不到他的数量该如何是好？？？

应该是有把八对匹配点，最好有八个静态目标，每个目标里拿出一个匹配点，

这种情况可能只有在复杂路口，或闹市才能满足，平时的话或许精度也就一般了，因此我想可以在一般的路况下，就用简单的动态目标剔除，剩余点匹配求解位姿，一旦找到了满足要求的静态目标(比如设为8个，这可能比较难满足)，使用静态目标里的匹配点求解位姿，这样确实能规避一些动态物体的影响，但是静态目标要是绝对静的，不可能移动的物体！！！！

3.10.2023--(动态物体的后检验用局部光流(多视图几何)，如果满足静态物体的阈值)

下边我将开始对自己所看的博客，文献，以及任何资料都做一个记录，以免有时候忘记想要查看的内容

2023

为了实现这个功能，您需要在ORB-SLAM3中嵌入YOLOv5目标检测模型，并根据检测结果调整特征点匹配。下面是一个简化的实现步骤：

1. 首先确保你已经正确安装了**yolov5**和**PyTorch**。
2. 在ORB-SLAM3源代码中找到**System.cc**和**System.h**文件，然后将以下代码添加到**System.h**中，引入必要的头文件和添加新的成员函数。

#include "torch/script.h" // 包含PyTorch C++ API头文件

class System

{

public:

//...

std::vector<std::tuple<int, float, float, float, float, float>> DetectObjects(const cv::Mat &im);

private:

//...

torch::jit::script::Module yolo\_model\_; // 声明YOLOv5模型对象

};

1. 在**System.cc**的**System**构造函数中，加载YOLOv5模型：

System::System(const string &strVocFile, const string &strSettingsFile, const eSensor sensor)

{

//...

try {

// 加载YOLOv5模型

yolo\_model\_ = torch::jit::load("yolov5\_best.pt"); // 请将模型文件路径替换为实际路径

yolo\_model\_.to(at::kCUDA);

yolo\_model\_.eval();

}

catch (const c10::Error &e) {

cout << "Error loading the YOLOv5 model\n";

}

}

1. 在**System.h**中实现**DetectObjects**函数：

std::vector<std::tuple<int, float, float, float, float, float>> System::DetectObjects(const cv::Mat &im)

{

//...

}

1. 在**DetectObjects**函数中添加YOLOv5目标检测的实现：

std::vector<std::tuple<int, float, float, float, float, float>> System::DetectObjects(const cv::Mat &im)

{

std::vector<std::tuple<int, float, float, float, float, float>> detections;

// 将cv::Mat转换为torch::Tensor

cv::Mat img\_float;

im.convertTo(img\_float, CV\_32F, 1.0 / 255);

auto tensor\_img = torch::from\_blob(img\_float.data, {1, img\_float.rows, img\_float.cols, 3}).to(at::kCUDA);

tensor\_img = tensor\_img.permute({0, 3, 1, 2});

// YOLOv5目标检测

std::vector<torch::jit::IValue> inputs;

inputs.push\_back(tensor\_img);

auto output = yolo\_model\_.forward(inputs).toTuple()->elements()[0].toTensor().to(torch::kCPU);

// 解析YOLOv5输出

auto output\_data = output.accessor<float, 2>();

for (int i = 0; i < output.size(0); i++) {

int class\_id = static\_cast<int>(output\_data[i][5]);

float score = output\_data[i][4];

float x1 = output\_data[i][0];

float y1 = output\_data[i][1];

float x2 = output\_data[i][2];

float y2 = output\_data[i][3];

// 您可以根据需要对输出结果进行筛选，例如只保留分数较高的检测结果

if (score > 0.5) {

detections.emplace\_back(class\_id, score, x1, y1, x2, y2);

}

}

return detections;

}

6. 修改Tracking模块，使用`DetectObjects`函数过滤动态物体。在`Tracking.cc`文件中找到`Track()`函数，然后在特征点提取之后调用`DetectObjects`。

```cpp

void Tracking::Track()

{

//...

// 特征点提取

//...

// 调用DetectObjects函数获取目标检测结果

auto detections = mpSystem->DetectObjects(imGray);

// 过滤掉动态物体上的特征点

for (size\_t i = 0; i < mCurrentFrame.N; i++) {

cv::Point2f point = mCurrentFrame.mvKeys[i].pt;

for (const auto &det : detections) {

int class\_id;

float score, x1, y1, x2, y2;

std::tie(class\_id, score, x1, y1, x2, y2) = det;

// 如果特征点位于检测框内且物体属于动态物体类别（根据实际情况确定类别ID），则删除该特征点

if (point.x >= x1 && point.x <= x2 && point.y >= y1 && point.y <= y2) {

// 请根据实际情况确定动态物体类别ID

if (class\_id == /\*动态物体类别ID\*/) {

mCurrentFrame.mvbOutlier[i] = true;

break;

}

}

}

}

// 剩余的静态特征点将用于后续的位姿估计

//...

}