实时平面目标跟踪

201800130086　徐鹏博

2020 年 12 月 25 日

内容提要

任务要求

- 完成一个如以下视频所示的平面跟踪演示程序：
  平面模板可以提前制作好并存贮在文件里，避免每次运行都要交互框选

任务要求

- 完成一个如以下视频所示的平面跟踪演示程序：
  平面模板可以提前制作好并存贮在文件里，避免每次运行都要交互框选

- 要求:
  - 速度达到实时 (25 帧/秒 +)
  - 跟踪稳定，不要有明显的错误和抖动
  - 尝试结合连续特征跟踪（KLT 方法，cv::calcOpticalFlowPyrLK）改善速度和稳定性
  - 详细记录实验过程中针对遇到问题的解决方案和结果

基本思想

- 基于光流法
  - 基本假设条件
    - 亮度恒定不变
    - 时间连续或运动幅度微小
    - 保持空间一致性

基本思想

- 基于光流法
  - 基本假设条件
    - 亮度恒定不变
    - 时间连续或运动幅度微小
    - 保持空间一致性
  - 原理
    - 对一个连续的视频帧序列进行处理
    - 针对每一个视频序列，利用一定的目标检测方法，检测可能出现的前景目标
    - 如果某一帧出现了前景目标，找到其具有代表性的关键特征点
    - 对之后的任意两个相邻视频帧而言，寻找上一帧中出现的关键特征点在当前帧中的最佳位置，从而得到前景目标在当前帧中的位置坐标

基本思想

- 基于光流法
  - 基本假设条件
    - 亮度恒定不变
    - 时间连续或运动幅度微小
    - 保持空间一致性
  - 原理
    - 对一个连续的视频帧序列进行处理
    - 针对每一个视频序列，利用一定的目标检测方法，检测可能出现的前景目标
    - 如果某一帧出现了前景目标，找到其具有代表性的关键特征点
    - 对之后的任意两个相邻视频帧而言，寻找上一帧中出现的关键特征点在当前帧中的最佳位置，从而得到前景目标在当前帧中的位置坐标
- 主要优点
  稀疏光流并不对图像的每个像素点进行逐点计算。它通常需要指定一组点进行跟踪，这组点最好具有某种明显的特性，例如 Harris 角点等，那么跟踪就会相对稳定和可靠。稀疏跟踪的计算开销比稠密跟踪小得多，实时计算速度效果较好

实验流程

- 主要步骤
  - 根据模板利用 SIFT 选择初始帧特征点
  - 开始跟踪:
    - 通过函数 cvCalcOpticalFlowPyrLK 跟踪特征点
    - 删除损失特征点
    - 对对应角点进行筛选，抛弃表现不好的点
    - 替换前后帧，即将后帧作为下一帧的前帧
  - 当遇到角点过少时:
    重新与模板进行匹配得到该帧角点

实验流程

- 主要步骤
  - 根据模板利用 SIFT 选择初始帧特征点
  - 开始跟踪:
    - 通过函数 cvCalcOpticalFlowPyrLK 跟踪特征点
    - 删除损失特征点
    - 对对应角点进行筛选，抛弃表现不好的点
    - 替换前后帧，即将后帧作为下一帧的前帧
  - 当遇到角点过少时:
    重新与模板进行匹配得到该帧角点

- 跟踪可视化
  - 将每次的特征点转换为 KeyPoint 格式，然后通过 SIFT 计算特征向量和描述子并匹配
  - 利用 drawMatches() 函数对每帧与模板对应的特征点进行匹配连线

具体过程

- 先完成根据模板利用 SIFT 选择特征点:

```
1  //其中KeyPoint1,2;good_matches,matches,matcher都设置成了全局变量.cnt是比
       例因子
2  void GetMatch(const Mat& img1, const Mat& img2){
3      Ptr<SIFT> detector = SIFT::create();
4      detector->detect(img1, KeyPoint1); detector->detect(img2, KeyPoint2);
5      detector->detectAndCompute(img1, Mat(), KeyPoint1, MatL);
6      detector->detectAndCompute(img2, Mat(), KeyPoint2, MatR);
7      FlannBasedMatcher matcher;
8      if (MatR.type() != CV_32F || MatL.type() != CV_32F) {
9          MatR.convertTo(MatR, CV_32F);
10         MatL.convertTo(MatL, CV_32F);
11     }
12     matches.clear();    good_matches.clear();
13     matcher.match(MatL, MatR, matches);
14     double minDistance = 2000, maxDistance = 5.0;
15     for (int i = 0; i < MatL.rows; i++) {
16         double distance = matches[i].distance;
17         if (distance > maxDistance) maxDistance = distance;
18         if (distance < minDistance) minDistance = distance;
19     }
20     sort(matches.begin(), matches.end());
21     int ptsPairs = std::min(50, (int) (matches.size() * cnt));
22     for (int i = 0; i < ptsPairs; i++)
23         good_matches.push_back(matches[i]);
24 }
```

具体过程

- 光流法，初始化跟踪特征点

```
1  //调用了刚才写的GetMatch()函数
2  void Initial(){
3      cvtColor(frame, gray, COLOR_BGR2GRAY);
4      cvtColor(temp_img,temp_gray,COLOR_BGR2GRAY);
5      GetMatch(temp_gray, gray);
6      for (size_t t = 0; t < good_matches.size(); t++)
7          initial.push_back(KeyPoint2[good_matches[t].trainIdx].pt);
8      points[0].clear();
9      points[0].insert(points[0].end(), initial.begin(), initial.end());
10     gray.copyTo(gray_prev);
11     calcOpticalFlowPyrLK(gray_prev, gray, points[0], points[1], status,
           err);
12     output.copyTo(temp_out);
13     int k = 0;
14     for (size_t i = 0; i < points[1].size(); i++) {
15         if (acceptTrackedPoint(i)) {
16             initial[k] = initial[i];
17             points[1][k++] = points[1][i];
18         }
19     }
20     points[1].resize(k);
21     initial.resize(k);
22     DrawMatch(temp_img,frame,points[1],initial);
23     imshow(window_name, output);
24     swap(points[1], points[0]);
25     swap(gray_prev, gray);
26 }
```

具体过程

- 光流法跟踪中一些其他函数

```
1  bool Point_in_Rect(const Rect& rect,const Point2f& point)//判断点在区域中
2  {
3      if (rect.x<=point.x && point.x<=rect.x+rect.width && rect.y<=point.y
           && point.y<=rect.y+rect.height)  return true;
4      else   return false;
5  }
6  bool acceptTrackedPoint(int i)//判断是否保留特征点
7  {
8      return  status[i] && Point_in_Rect(Temp,points[1][i]);
9  }
```

具体过程

- 光流法，过程跟踪

```
1  void tracking() {
2      cvtColor(frame, gray, COLOR_BGR2GRAY);
3      if(points[0].size()<20){
4          cvtColor(temp_img,temp_gray,COLOR_BGR2GRAY);
5          GetMatch(temp_gray, gray);  points[1].clear();
6          for (size_t t = 0; t < good_matches.size(); t++)
7              points[1].push_back(KeyPoint2[good_matches[t].trainIdx].pt);
8          initial.insert(initial.end(),points[1].begin(),points[1].end());
9          points[0].insert(points[0].end(), initial.begin(), initial.end())
               ;
10     }
11     points[1].clear();int k = 0;
12     calcOpticalFlowPyrLK(gray_prev, gray, points[0], points[1], status,
           err);
13     for (size_t i = 0; i < points[1].size(); i++) {
14         if (status[i] ) {
15             initial[k] = initial[i];
16             points[1][k++] = points[1][i];
17         }
18     }
19     points[1].resize(k);
20     initial.resize(k);
21     DrawMatch(temp_img,frame,points[1],initial);
22     imshow(window_name, output);
23     swap(points[1], points[0]);// 把当前跟踪结果作为下一此参考
24     swap(gray_prev, gray);
25 }
```

具体过程

- 鼠标交互截出模板

```
1  void onMouse(int event,int x,int y,int flag,void* param) {
2      if (event == EVENT_LBUTTONDOWN) {
3          flag = true;
4          P1 = Point(x, y);
5          P2 = P1;
6      }
7      if (event == EVENT_MOUSEMOVE && flag) {
8          tmp = frame.clone();
9          P2 = Point(x, y);
10         if (P1 != P2) {
11             rectangle(tmp, P1, P2, Scalar(255, 0, 0), 2);
12         }
13         imshow(window_name, tmp);
14     }
15     if (event == EVENT_LBUTTONUP) {
16         flag = false;
17         Temp=Rect(P1, P2);
18         temp_img=tmp(Temp);
19         Initial();
20         get_temp = true;
21     }
22 }
```

具体过程

- main() 中主要循环函数

```
1   while (true) {
2       if (!flag) {
3           capture >> frame;
4           frame.copyTo(output);
5       }
6       if (get_temp) {
7           tracking();
8           imshow(window_name, output);
9           if (!get_size) {
10              siz = Size(int(output.cols), int(output.rows));
11              writer.open(filename, codec, fps, siz, true);
12              get_size = true;
13          }
14          writer << output;
15      } else {
16          imshow(window_name, output);
17      }
18      int c = waitKey(25);
19      if (c == 27) break;
20      if (c == 'q') waitKey(0);
21  }
22  capture.release();
```

总结分析

- 1. 最开始没有使用光流法，单独用 SIFT/SURF 对每一帧和模板进行匹配，实时性不好，并且每一帧特征选择之间关联不大，猜想是前后帧之间没有有效联系。

总结分析

- 1. 最开始没有使用光流法，单独用 SIFT/SURF 对每一帧和模板进行匹配，实时性不好，并且每一帧特征选择之间关联不大，猜想是前后帧之间没有有效联系。
- 2. 第二次修改时，只使用光流法 KLT 跟踪时，初始化特征点是利用自带的 goodFeaturesToTrack，跟踪过程中只进行特征点的剔除。发现效果也不是很好，特征点会慢慢变少；或者当出现偏离后，之后一段时间内都保持偏离状态。猜想是由于之后每一帧都没有与模板进行适当联系。

总结分析

- 1. 最开始没有使用光流法，单独用 SIFT/SURF 对每一帧和模板进行匹配，实时性不好，并且每一帧特征选择之间关联不大，猜想是前后帧之间没有有效联系。
- 2. 第二次修改时，只使用光流法 KLT 跟踪时，初始化特征点是利用自带的 goodFeaturesToTrack，跟踪过程中只进行特征点的剔除。发现效果也不是很好，特征点会慢慢变少；或者当出现偏离后，之后一段时间内都保持偏离状态。猜想是由于之后每一帧都没有与模板进行适当联系。
- 3. 第三次，在光流法基础上，为每次跟踪时发现跟踪点较少时都重新与模板进行匹配以增加初始化特征点集。效果出现明显提高，但是出现许多被匹配到的外点连线，猜想是由于比例因子过大。

总结分析

- 1. 最开始没有使用光流法，单独用 SIFT/SURF 对每一帧和模板进行匹配，实时性不好，并且每一帧特征选择之间关联不大，猜想是前后帧之间没有有效联系。
- 2. 第二次修改时，只使用光流法 KLT 跟踪时，初始化特征点是利用自带的 goodFeaturesToTrack，跟踪过程中只进行特征点的剔除。发现效果也不是很好，特征点会慢慢变少；或者当出现偏离后，之后一段时间内都保持偏离状态。猜想是由于之后每一帧都没有与模板进行适当联系。
- 3. 第三次，在光流法基础上，为每次跟踪时发现跟踪点较少时都重新与模板进行匹配以增加初始化特征点集。效果出现明显提高，但是出现许多被匹配到的外点连线，猜想是由于比例因子过大。
- 4. 之后又对特征点选择的比例因子进行调整，找到比较良好的参数范围。

总结分析

- 1. 最开始没有使用光流法，单独用 SIFT/SURF 对每一帧和模板进行匹配，实时性不好，并且每一帧特征选择之间关联不大，猜想是前后帧之间没有有效联系。
- 2. 第二次修改时，只使用光流法 KLT 跟踪时，初始化特征点是利用自带的 goodFeaturesToTrack，跟踪过程中只进行特征点的剔除。发现效果也不是很好，特征点会慢慢变少；或者当出现偏离后，之后一段时间内都保持偏离状态。猜想是由于之后每一帧都没有与模板进行适当联系。
- 3. 第三次，在光流法基础上，为每次跟踪时发现跟踪点较少时都重新与模板进行匹配以增加初始化特征点集。效果出现明显提高，但是出现许多被匹配到的外点连线，猜想是由于比例因子过大。
- 4. 之后又对特征点选择的比例因子进行调整，找到比较良好的参数范围。
- 5. 此时对于平移，旋转，倾斜等情况都能匹配到，但是仍然有因为幅度较大以及平面光照不均匀、时有反光等因素导致的匹配错误。

实验效果

视频效果请直接看 avi 文件!

谢　谢!