由于预处理过程会产生一些非常不好的中间结果，所以并没有将所有的函数放到一个脚本中统一执行。本说明旨在解释各函数的执行顺序及一些注意事项，在某些函数执行之后，还需要手动去删除一些不良结果。由于文件比较独立，所以可能某些文件开头部分有重复的读取视频的信息，在执行过程中，依具体情况自行注释掉相关部分。

1. Tracking.m 对一段监控视频进行活动物体检测、跟踪及分割。运行一个特定视频的时候，注意更改文件路径。
2. Generate\_Objects.m 将上一步的结果转换成一个个活动物体的数据，包含bounding 位置等信息。
3. Delete\_Objects.m 删除存在时间不足两帧的物体，预处理过程会产生很多细小的tube片段，大部分是背景，也可能是物体tube的一部分（Tube的概念请参考视频浓缩相关论文）。所以不建议使用这个函数来做优化，效果最好的办法是将tube生成单个视频，然后手动去删除数据。具体见后面。
4. Generate\_Tube.m 生成tube数据，实质是将属于每个物体的segmentation mask串联起来。注意更改文件的路径。背景要挑一帧原视频中没有活动物体的，在代码中更改参数。
5. SoloTestID.m 为每个tube生成相应的视频，这样通过察看视频，就可以找到那些tube中含有活动物体，哪些是由于背景噪声产生的。记录下不良tube的相关信息，如ID编号、在数据中所处的行数等，再将其手动从数据中删除。（是在第2步产生的object数据中进行删除）
6. changeID.m 当删除不良数据后，object的ID变号就不再连续，所以这个函数是重新组织活动物体的编号使之连续。
7. 利用第4步，重新生成tube，到这里就会生成较为完好的tube信息了。
8. Estimate\_Density.m 判断视频在各个时间段的拥挤程度，得到满足拥挤条件的帧序号。由于前面的Tracking.m 是matlab自带的程序，并没有直接在那个模块里进行判断，而是单独写了个文件。使用时注意文件路径。
9. Separate\_crowded\_frames.m 原视频中拥挤程度可能是随时间变化的，因此拥挤视频片段也应该是一段一段的。此函数将上步得到的拥挤视频的帧序号进行分组，如果两个帧号相差不足指定的阈值，那么这两个帧序号之间的视频也同样被当作是拥挤视频的一部分。最终得到的结果是frame\_group，这个数据的每行包括两个数，一个是拥挤视频片段的开始帧号，另一个是结束帧号。数据有多少行，证明原视频包含多少个独立的拥挤视频片段。
10. DeleteObjectinCrowd.m 我们的方法是将拥挤时段的tube用video clip替换，所以这个函数就是删除拥挤时段出现的object。
11. 利用第6步的changeID.m 来梳理object的编号，并利用第4步的Generate\_Tube.m生成新的tube数据，之前生成的tube可以用来做对比实验，如不需要就可以直接删掉。
12. NewMergeObjects.m 对objects进行分组，详细算法步骤可参考论文。
13. Group\_adjustment.m 合并objects分组，为tube与拥挤视频片段的组合做准备。（对拥挤视频片段的提取并删除了相应时段的objects，那么本来应该属于同一组的objects之间的联系可能就被切断，继而它们会被分到了两个组里，这里利用object和视频片段的时间关系来合并潜在的上述分组）
14. Calculate\_group\_duration.m 计算每个组的持续时间、结束时间等基本信息。
15. MergeObjectandFrame.m 对objects和拥挤视频片段进行合并（逻辑上的合并，将相关的objects序号和拥挤片段的序号及其他一些信息进行合并）。
16. Sort\_Group.m 每个group内部按开始时间，对成员排序。
17. SortStructByDuration.m 各个group按它们的历时长短进行排序，历时长的排前面，为之后的group的出场顺序优化做准备。
18. Optimization.m 进行优化，安排每个group、每个成员在浓缩视频中的出场时间。
19. Stitch.m 根据前面所有的计算结果，合成浓缩视频。

没有介绍到的函数可能是一些小的功能性函数或是一些供测试使用的函数及脚本，不在算法流程的重点之内，这里为了使流程清晰，故没有介绍。