**Question 5**:

The algorithm was executed on frames 1-40 of PETS2014-0101 video, and scaled down by 50% for making our computer able to process it.

1. Short description of the algorithm:

The algorithm perform tracing on objects by looking for their histograms in every frame. The objects-to-track (= **models**) initial tracking details are inserted manually (their location, and in what frames they appear), and also their **histogram** for the first frame is calculated. In every consecutive frame, we look for the model in some **region** around it by doing exhaustive search for **candidate-model-histogram** in that region. Using the region's integral histogram, the exhaustive search is quickly performed. In every exhaustive-search-round, the algorithm tests candidates-models with 75%, 100% and 125% size of the original model in order to support scaling (this is available due to normalization of the histograms). When the best candidate is chosen. If after certain consecutive frames the object is not found on the screen, it's no longer tracked (if the gap between the original histogram and the new is larger than a threshold, for X consecutive frames, the algorithm no longer tracks the object).

1. Tracking results:

We tested our video on the PETS2014-0101 video from the given DATA folder. The algorithm managed to track objects (motion is from left to right):



However when arriving to the conus in the last frame above, the algorithm found that the conus should be part of the pedestrians we track, but was able to recover few frames later:



This is due to the color of the wall behind the pedestrian, which has similar colors to the conus, and when there's less wall, there'll be more conus.

There's another aspect that affects the algorithm performance: resolution.

There are 2 more pedestrians that the algorithm was told to track on (gray & yellow pedestrians):



Let's look at them a little closer:

 

The histogram of the left pedestrian is very similar to other histograms around it, such as where the algorithm "detects" him in the next frame:



And is very hard to detect. The other pedestrian is very different from his environment and therefore the algorithm manages to track him:



The algorithm didn't notice the gray guy at all, since he looks like his environment.

To summarize up the results: the algorithm can effectively track objects and is sensitive to the model's histogram changes. However the resolution is a major player especially here, since as the resolution is lower it is hard to detect the tracked objects.

1. Continue with Ben

**Question 6:**

The algorithm was executed on frames 360-460 of Walk2.mpeg.

1. For tracking objects we will use the algorithm in question 5, using integral histograms. The method for determining when people are getting close to each other is by looking at their **search region** around them. A search region is the area around the tracked object, that we will exhaustively-search it in the next frame. When 2 search regions are overlapping each other 🡪 the objects may be close to each other in the next frames 🡪 we draw a red line between them.

The method will absolutely fail for objects that are close on X & Y axes but far on the Z axis (when one object is more "deeper" than the other), because the algorithm do not support the Z axis. The method will also fail when at the 1st frame the two objects are near each other, but in the next frames are getting far from each other. So until their regions won't overlap anymore, even though they are getting far from each other – the algorithm will mark them as "getting close".

1. Implemented in integralHistogramTracking.m. Depend on parameter 'check\_proximity'.
2. Implemented in q6\_c.m.

The following is a snapshot of the result. When 2 objects are close to each other, a red line is stretched between them (motion is from left to right):

