Lab 3

This week we will start on the real tracker. We we'll start implementing some improvements to the brute force tracker. The goal is a Mean-Shift algorithm is described in Kernel-Based Object Tracking *by Dorin Comaniciu, Visvanathan Ramesh and Peter Meer. It is advisable to have read this paper before continuing.*

Kernel based histograms

The first step towards a *Mean-Shift* tracker is implementing kernel weighted histograms. This means that when you create a histogram of your target or candidate, you weigh the pixel values according to their distance from the centre. The rationale of this is that pixels further from the centre of an object are more likely to be occluded or otherwise transformed during the tracking. Another reason is that the kernel creates a smooth landscape of histogram distances; the smoothness is needed to perform the mean-shift step later on. One way to implement this is by creating a mask of the weight function. This is a 2D array, from which you can easily retrieve the weight for each pixel position. This weight is added to the corresponding bin in the histogram. The mean-shift tracker uses the Epanechnikov kernel. The formula is in the paper, but you will need these constants: d= 2, c_d=∏. The kernel can be calculated beforehand and store the values in a matrix the size of the target region. This way you can look up the weight of a pixel with one look in the kernel matrix.

For this you need a function that can take the histogram at a certain point (x,y coordinates). The nicest way is to use the centre of the target as reference (origin), but you can also adjust the coordinates to get the same result. By making the mask the same dimensions as the target, you can map the locations one-to-one, making it easy to retrieve the correct weights.

I think it is best that you just start to program and develop by trial and error. In the **data-directory** you'll find an archive with many (200) frames. You can probably still track if you skip some frames at each step.

Here is some information on how to easily read all the frames. Since Matlab requires a huge amount of memory for reading all the frames, your computer will completely freeze is you try to read all the frames at once. Therefore, you should read all the frames incrementally: when you need a frame, read it in. The frames in the files MoreFrames_part_1.zip, MoreFrames_part_2.zip,MoreFrames_part_3.zip, MoreFrames_part_4.zip in the **data directory** are numbered from 0085 to 0285. So, you can process all the images in a for-loop. During the tracking, you may like to save the frames into a movie-file. There are several parameters that can be tuned, you can play around with these. This can be done like this (be warned that the resulting movie can become quite large!):

```
path = your_path_to_the_images;

all_frames =[];
for next_frame_nr = 85:285

        next_frame_name = [path '/frame' num2str(next_frame_nr,'%04d') '.png'];

        next_frame = im2double(imread(next_frame_name));



        % *your tracking-code here*



        all_frames = [all_frames getframe(gca)];
```

```
end


save_movie(all_frames, 'your_movie.avi', 15, 100, 'Cinepak');

function save_movie(frames, movie_name, fps, quality, compression)

                    curr_movie = avifile(movie_name, 'fps', fps, ...

                   'quality', quality, ...

                   'Compression', compression);

                    curr_movie = addframe(curr_movie, frames);

                    curr_movie = close(curr_movie);

end
```