Condensation Tracker Report

Nicole Damblon

Computer Vision

1 Condensation Tracker Algorithm

A recursive bayesian filter is implemented for tracking a specific object in a video. The object to track is represented by a probability distribution $p(x_t)$, where x_t is a state vector that contains coordinates and/or velocities. We are interested in the predictive distribution of the next time step given all measurements Z_{t-1} we obtained up to that point: $p(x_t|Z_{t-1})$, the *a priori distribution*. With the new measurement z_t we can then obtain the *a posteriori distribution* $p(x_t|Z_t)$. Given the *a posteriori distribution* from the previous iteration, the condensation tracker approximates the *a priori distribution* with a sampled set that is yielded by a noisy system model. This is the **prediction step**. In the **update step**, the information we gain from the new measurement z_t is used to update the prediction and approximate the new *a posteriori distribution* $p(x_t|Z_t)$ by re-weighting each sample in accordance to how 'close' it is to the the tracked object.

Compute the color histograms. The tracked object is represented as a bounding box (also called particle) with state vector $s = (x, y, x', y')^T$, where x and y are the center of the bounding box and x', y' the velocities in x and y directions. In order to determine a similarity between two different bounding boxes, to have a measure of 'closeness', color histograms over the bounding boxes are used. Three histograms per bounding box, one for each channel R,G,B, are computed with the function `numpy.histogram` and normalized by the total amount of pixel binned into the histogram.
Propagate the particles. Next, the particles are propagated, that means position (and velocity) evolve during time. This implements the prediction step. This assignment implements two models, one considers no motion resulting in a state vector only consisting of the coordinates $s = (x, y)$ and the second assumes constant velocity in x and y direction $s = (x, y, x', y')$. The system model that predicts the next position given, the current one (and the velocity) is given by

$$s_t = As_{t-1} + w_{t-1}, \tag{1}$$

where w is the system noise and the matrix A is mapping the state vector to its new position:
1. Model considers no motion

$$A = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \implies A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{2}$$

where $(x, y)^T$ is the state vector and A is the identity matrix because we assume no motion, hence the position has to stay the same. The change in position is only caused by the added noise, which is sampled from a univariate gaussian distribution with `numpy.random.randn` and given variance `sigma_position`.
2. Model assumes constant motion

$$A = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \\ x' \\ y' \end{bmatrix} = \begin{bmatrix} x + x' \\ y + y' \\ x' \\ y' \end{bmatrix} \implies A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3}$$

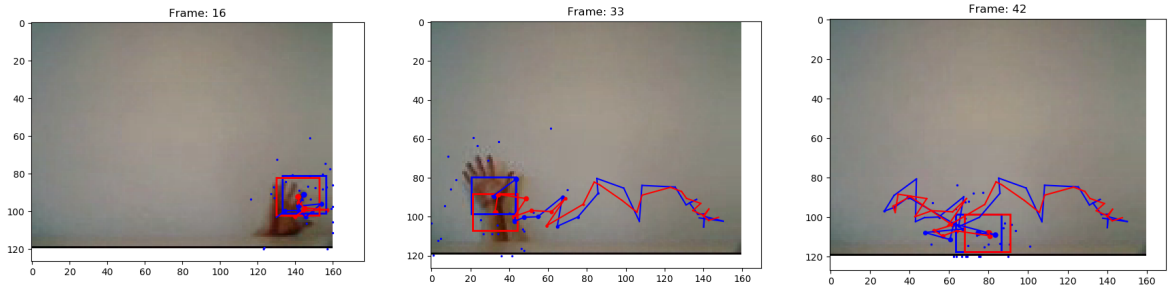
where the matrix A has to be derived, that maps the states x and y to their new states according to their velocities $x + x'$ and $y + y'$ and the motion x' and y' is constant, hence doesn't change. The solution for A achieves that. As compared to the no motion model, here we have to take into account the noise for position and velocity, which is achieved similarly to the first model. Finally, both models are updated according to equation 1.

Compute weights according to new observations. To obtain the weights per particle (bounding box), first the histograms for the propagated particles are computed. Next, the distance to the target histogram is computed. The last step is to compute the weights with a gaussian kernel function using the computed distances and the given measurement noise `sigma_observe`. The function returns the normalized weights of each particle. This function corresponds to the update step.
Estimate the mean state. The mean state is the weighted mean of the particles. For the 'no motion' model $s_{mean} = (x, y)$ is computed and for the constant motion model the mean state vector $s_{mean} = (x, y, x', y')$.
Resample particles. The last step is to resample the particles according to their 'importance'. Each weight quantifies how 'important' a sample is, a high weight indicates that the particle is close to the target A simple resampling method, multinomial resampling is implemented. First, each weight is associated with its 'importance' by cumulative summing up all the weights with `numpy.cumsum`. Visualized as a resampling wheel, each weight would have a length proportional to its value. Now, N (the number of particles) numbers between 0 and 1 are randomly generated each representing a 'pick' on the resampling wheel. A binary search finds the corresponding weight to the random number. The function `numpy.searchsorted(cumulative_sum, random_numbers)` returns the index of the picked weights, so the particles and weights can be resampled. The last step is to normalize the weights by dividing the weights by the sum over all weights.

2 Experiments and results

2.1 Video 1

The hand in video 1 is tracked with the original parameter set (table first row), once for model 0 and for model 1. The tracking performance looks good enough for both, see figure 1 and 2. The hand moves relatively slow and there are no occlusions or changing conditions. This 'easy' setting could be the reason why only 30 particles are enough to track the hand. Sampling less particles uses less computing power, so it could be a good idea to use little particles in those settings. However, the tracker in both models tends to follow the arm instead of the hand. The no motion model fails tracking the hand when the number of particles is to small num_particles = 15 and the measurement noise is assumed sigma_observe = 0.3, see fig 3. The particles are to sparse to capture the object.

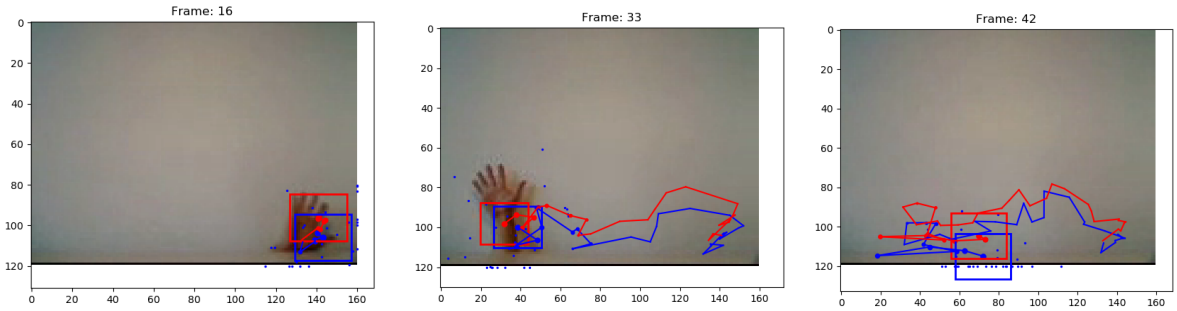


(a) (b) (c)

Fig. 1. Video 1 with original parameters, model 0.

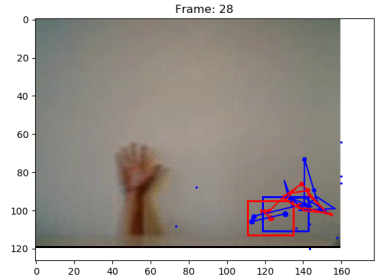
2.2 Video 2

In the first experiment the original parameters are used. Video 2 shows a faster moving hand with occlusion and background noise in form of a poster on the wall. Both models fail to track the faster moving hand with only 30 particles, fig. 4 and 5. Increasing the particles to 300 and the histogram



(a) (b) (c)

Fig. 2. Video 1 with original parameters, model 1.



(a)

Fig. 3. Video 1 with 15 particles, sigma_observe = 0.3, model 0.

count to 40 (to maybe get a more meaningful feature representation) only works for model 1, fig 5 (b). The tracking performance is good, only the occlusion confuses the tracker a bit. Model 0, assuming no motion, fails in tracking the hand with the same parameters, fig 4 (b). Increasing the number of particles to 500 leads to tracking the hand better, the tracker only looses track shortly at the poster fig (c). This experiments indicate, that if tracked object moves faster, a model with constant motion performs better and requires less particles than a model without motion.

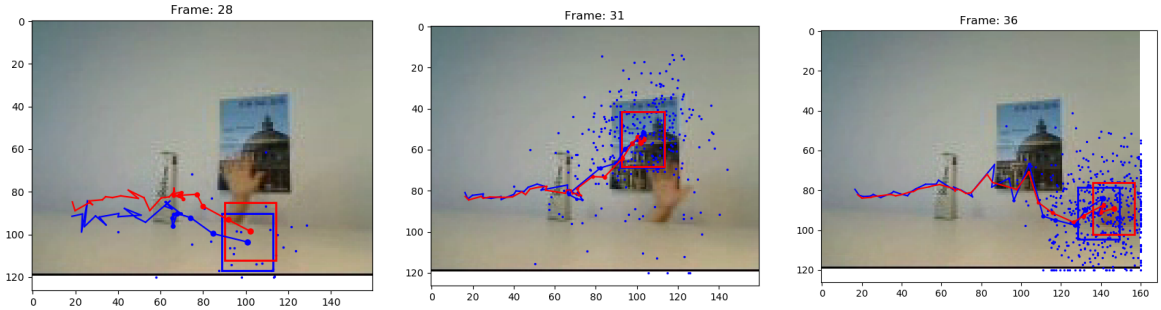
Increasing alpha from 0 to 0.5 the old target and mean state histogram are accounting for the same weight when updating the target histogram, see equation 4. Figure 6 (a) and 7 (a) show the effect of allowing appearance model updating, the no motion model gets stuck at the obstacle and after that fails to track the hand, however, model 1 has less problems when tracking the hand, the performance is still reasonably good. Model 1 fails in tracking the hand when increasing alpha to 0.8, the plot looks very similar to figure 6(a), where the no motion model gets stuck. This experiment shows, that both models fail eventually when alpha is too large, likely because the influence of the mean state histogram is to big on the update of the target histogram, so the background noise is confused with the target. However, with alpha = 0.1 the hand is tracked nicely with model 1 so an advantage of a non-zero alpha could be, that the tracker is more robust to a non-optimal old target histogram by considering also the mean state histogram. Interestingly, model 1 fails only with a very high alpha, so considering motion in that more complex and fast setting is an advantage.

Allowing more measurement noise, sigma_observe = 0.5 has the same effect on both models, first the trajectory is smooth and tracks the hand, but after a while the tracker gets too slow and fails keeping up with the hand. That indicates, that the assumption of the system noise should be rather small, see fig 6(b) for model 0 and 7(c) for model 1.

The next experiment varies the parameters for sigma_position (and sigma_velocity for the motion model), the two system noise parameters. Increasing both to sigma_position to 20 as shown in fig. 6(c) for the no motion model and fig 9 (a) for the motion model, has a similar effect on both models. In

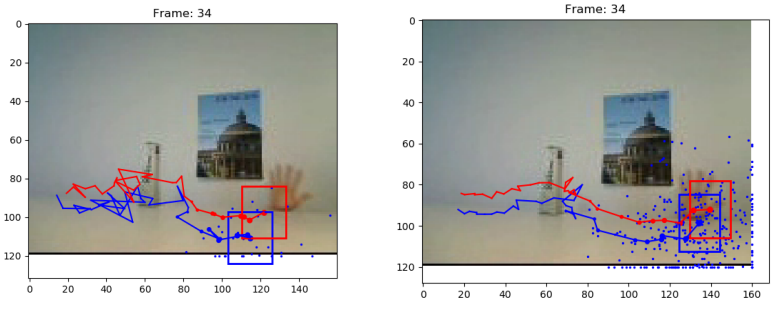
model 0, the tracker tracks the wall poster shortly before finding the hand again. In model 1, this also happens, and additionally the particles are clustered towards the edges of the frame, sigma_velocity = 5 in this experiment. Both results are not optimal, therefore lower values are tried next. Setting sigma_position = 10 and sigma_velocity = 2, as depicted in fig 8 for model 0 and fig 9 (b) for model 1, has a different effect: in model 0 the tracker is a bit too slow and shortly tracks the poster again, however the performance is much better than in the experiment before. Model 1 yields a very smooth trajectory but tilts down over time.
The last parameters to tune are the initial velocities for the motion model, fig 9 (c),(d). The tracker gets stuck at (0,120) and fails when increasing the velocity too much, when lowering it to inital_velocity = (5,5), the trajectory seems better, however it again tracks the poster shortly.
In summary, model 0 was less robust to parameter changes than model 1 resulting in failing completely, while the motion model was still tracking the hand. For a good tracking performance in this scenario, the motion model should be chosen with a large number of particles, here 300, as well as a low alpha, and sigma_observe. The histogram bins could be adjusted from 16 to 40, showing good results and the initial velocity shouldn't be too high.

$$CH_{target} = (1 - \alpha)CH_{target} + \alpha CH_{mean\_state} \tag{4}$$



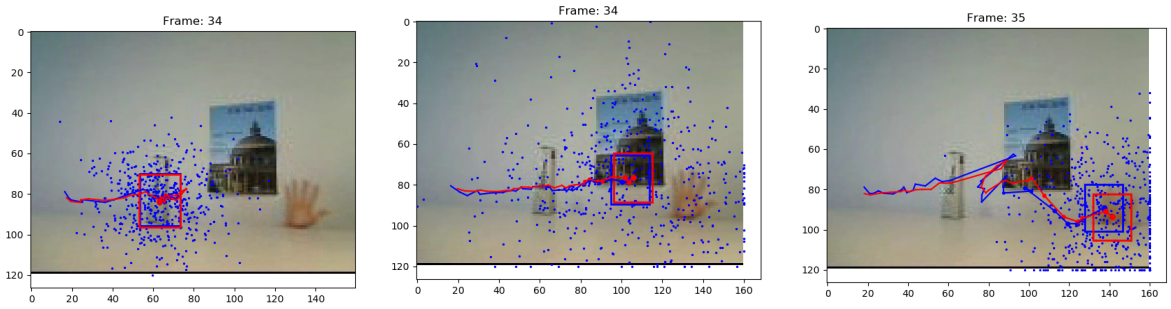
(a) (b) (c)

Fig. 4. Video 2, model 0, experiment 1,2,3 in table.



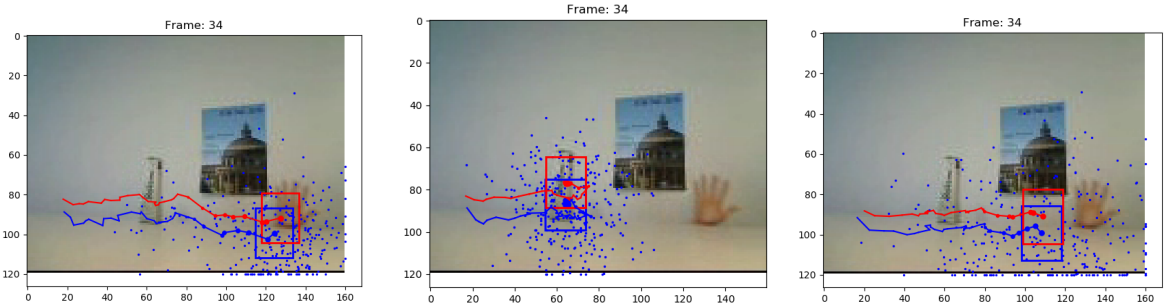
(a) (b)

Fig. 5. Video 2, model 1, experiments 1,2 in table.



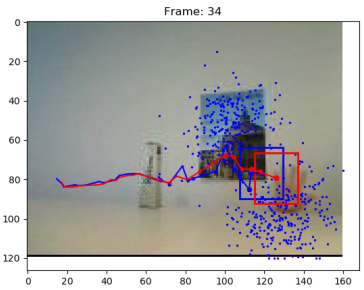
(a) (b) (c)

Fig. 6. Video 2, model 0, experiment 4,5,6 in table.



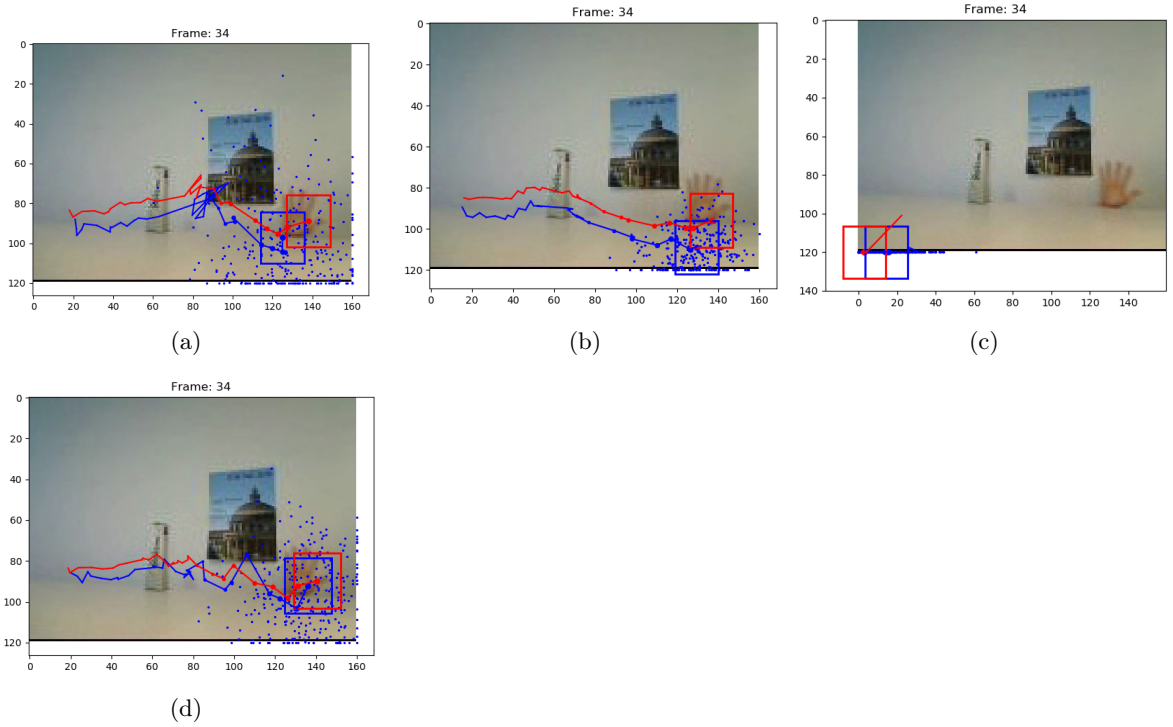
(a) (b) (c)

Fig. 7. Video 2, model 1, experiments 3,4,5 in table.



(a)

Fig. 8. Video 2, model 0, experiment 7 in table.



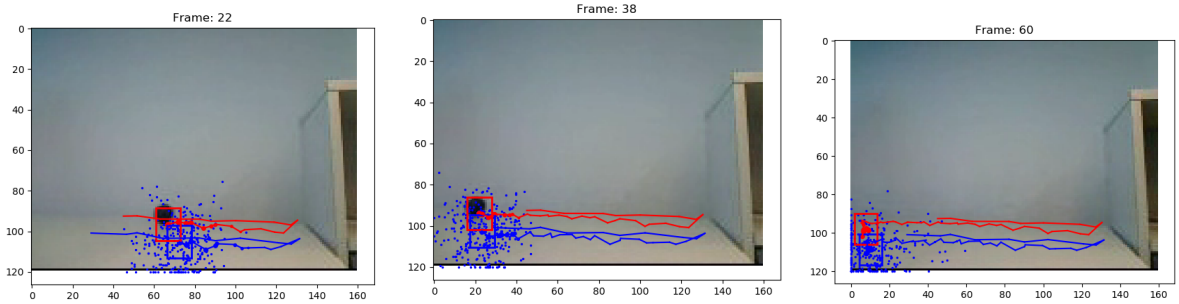
(a) (b) (c) (d)

Fig. 9. Video 2, model 1, experiments 6,7,8,9 in table.

Video 2								
fig.	model	hist_bin	alpha	sigma_obs	num_particles	sigma_pos	sigma_vel	initial_vel
4(a)	0	16	0	0.1	30	15	1	(1,10)
4(b)	0	**40**	0	0.1	**300**	15	1	(1,10)
4(c)	0	40	0	0.1	**500**	15	1	(1,10)
6(a)	0	40	**0.5**	0.1	500	15	1	(1,10)
6(b)	0	40	0	**0.5**	500	15	1	(1,10)
6(c)	0	40	0	0.1	500	**20**	1	(1,10)
8(a)	0	40	0	0.1	500	**10**	1	(1,10)
5(a)	1	16	0	0.1	30	15	1	(1,10)
5(b)	1	**40**	0	0.1	**300**	15	1	(1,10)
7(a)	1	40	**0.5**	0.1	300	15	1	(1,10)
7(b)	1	40	**0.8**	0.1	300	15	1	(1,10)
7(c)	1	40	0	**0.5**	300	15	1	(1,10)
9(a)	1	40	0	0.1	300	**20**	**5**	(1,10)
9(b)	1	40	0	0.1	300	**10**	**2**	(1,10)
9(c)	1	40	0	0.1	300	15	1	**(10,50)**
9(d)	1	40	0	0.1	300	15	1	**(5,5)**

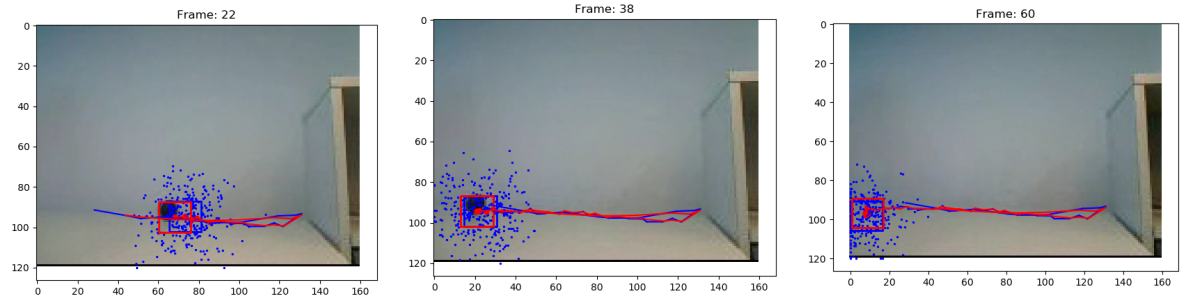
2.3 Video 3

Video 3								
fig.	model	hist_bin	alpha	sigma_obs	num_particles	sigma_pos	sigma_vel	initial_vel
10	1	40	0.1	0.1	300	10	2	(1,10)
11	0	40	0.1	0.1	300	10	2	(1,10)
-	0	40	**0.5**	0.1	300	10	2	(1,10)
-	0	40	0.1	0.1	**100**	10	2	(1,10)
-	0	40	0.1	0.1	**50**	10	2	(1,10)
12	0	40	0.1	**0.5**	**50**	10	2	(1,10)



(a) (b) (c)

Fig. 10. Video 3, model 1, best parameters (experiment 1 in table).

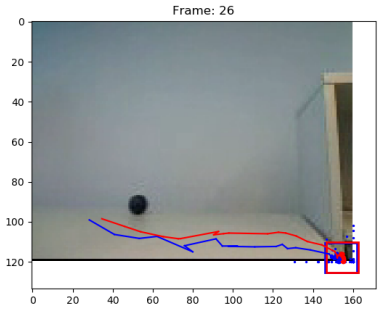


(a) (b) (c)

Fig. 11. Video 3, model 0, best parameters (experiment 2 in table).

The ball moves even faster than the hand but has less occlusions and no background noise. However it changes direction once. Using the best parameters of section 2, listed in the table for video 3 in the first row, leads to a very good tracking performance, see fig. 9. The same parameters with model 0 also lead to a very good result, a priori mean state and a posteriori mean state overlap much more, see fig 10. Interestingly, here a number of 300 particles is sufficient for model 0 to track the ball as compared to the hand in video 2, where 500 particles were necessary. A possible explanation could be, that occlusion and background patterns are a problem for the no motion model, but not mainly how fast the object moves. Since red and blue box overlap much more, it is no surprise that when setting alpha = 0.5, the no motion model still performs well, because a priori and a posteriori mean state are almost the same, so it doesn't matter how to weigh them in the update. In video 2 that led to a failure. Dereasing the number of particles to 100 (model 0) still tracks well, and even particles = 50. That leads to the

conclusion, a high number of particles is especially important in settings, where a lot of background clutter and occlusions are present. Model 0 fails with a sigma_observe = 0.5. When performing the same experiments with model 1, it shows similar behaviour as model 0. Only 50 particles and a high measurement noise of sigma_observe = 0.5 also leads to failure, it gets stuck in the dark corner on the right end of the frame, as shown in fig 12.



(a)

Fig. 12. Video 3, model 1, num_partices = 50, sigma_observe = 0.5.