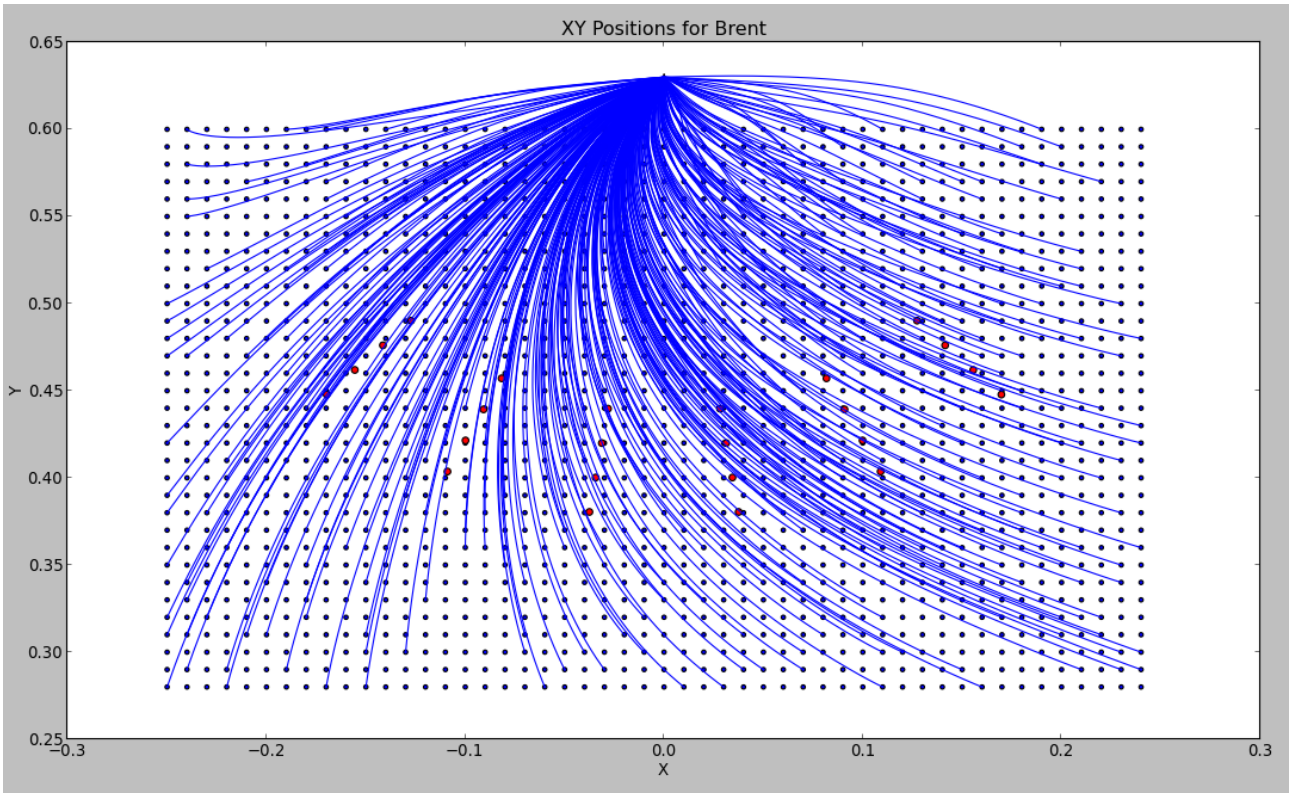
1 Data from the Brent controller used to train the RBFN controller

1 Training trajectories

The input of our process is a set of trajectories obtained from the model described in [Rigoux&Guigon]. There are some discrepancies between the protocol used to get these trajectories and the one used to train the RBFN controller. For training the Brent controller, the goal was to reach a small circle around the target from any direction, with a velocity close to zero. By contrast, for training the RBFN controller, the goal is to hit the target on a screen, so a high velocity is allowed and the trajectory stops when the ordinate of the end-effector becomes superior to that of the screen.

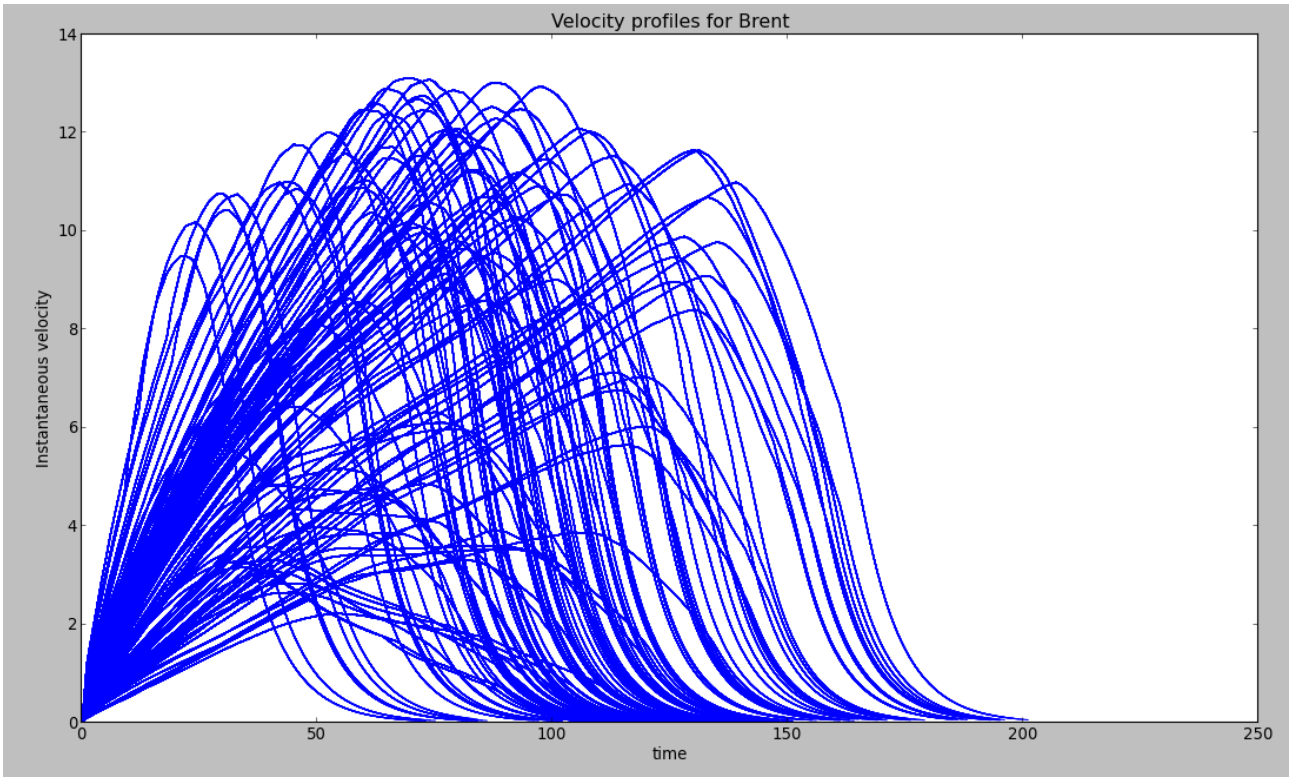
There are 1650 training trajectories starting from all the blue dots in the image.
For the sake of easier visualisation, only 20 % of these trajectories are displayed.



One can see that most trajectories starting from the upper right corner would hit the screen where the target is displayed far before reaching the target. This explains a strong bias towards the right when training the RBFN controller.

2 Velocity profiles

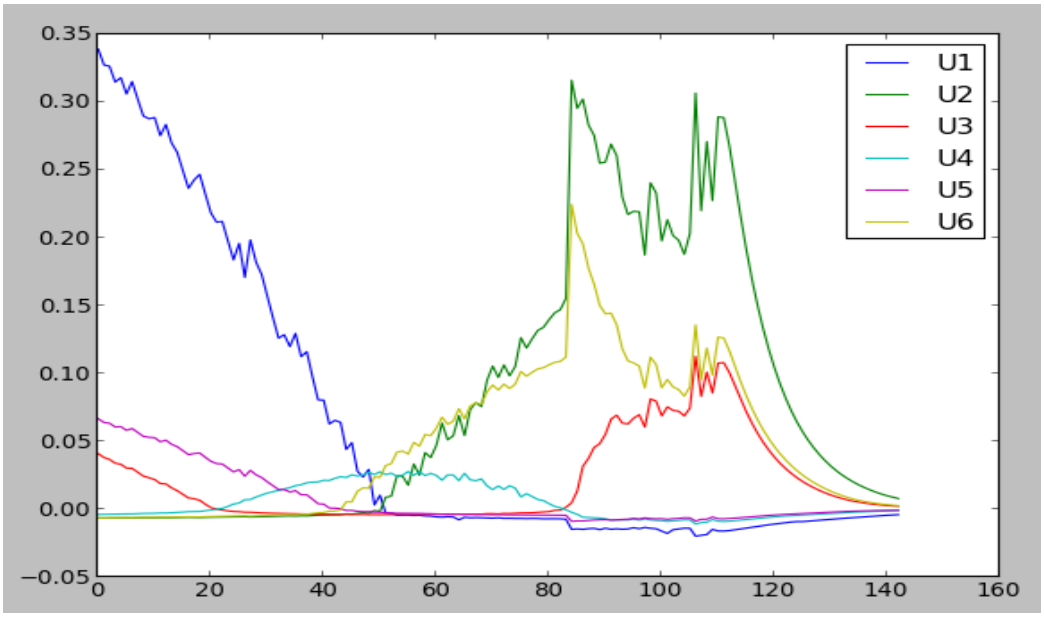
For the sake of easier visualisation, only 15 % of the available velocity profiles are displayed.



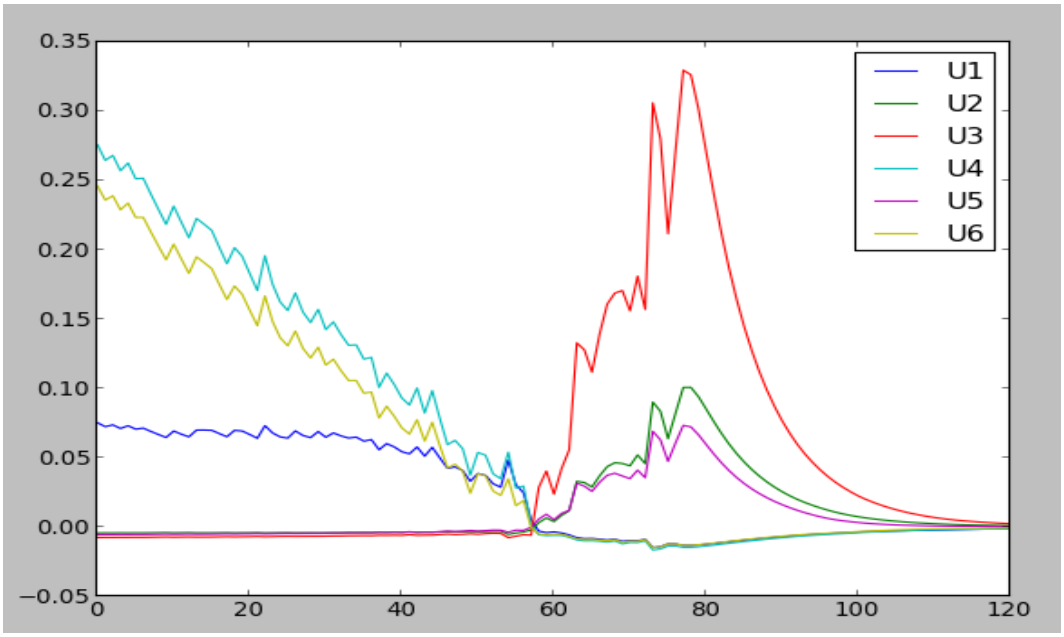
One can see that the constraint on close-to-null velocity at the end of trajectories is verified.

3 Muscular activation

In these displays, the motor noise is removed.



Typical muscular activations for a trajectory starting from the left (or is it the right ?) hand side. One can see that muscle 1 is activated first, then muscle 2.



Typical muscular activations for a trajectory starting from the other side.

One can see that muscles 4 and 6 are activated first, then muscle 6.

Note : it would be nice to comment on that, explain the strategy, compare to humans.

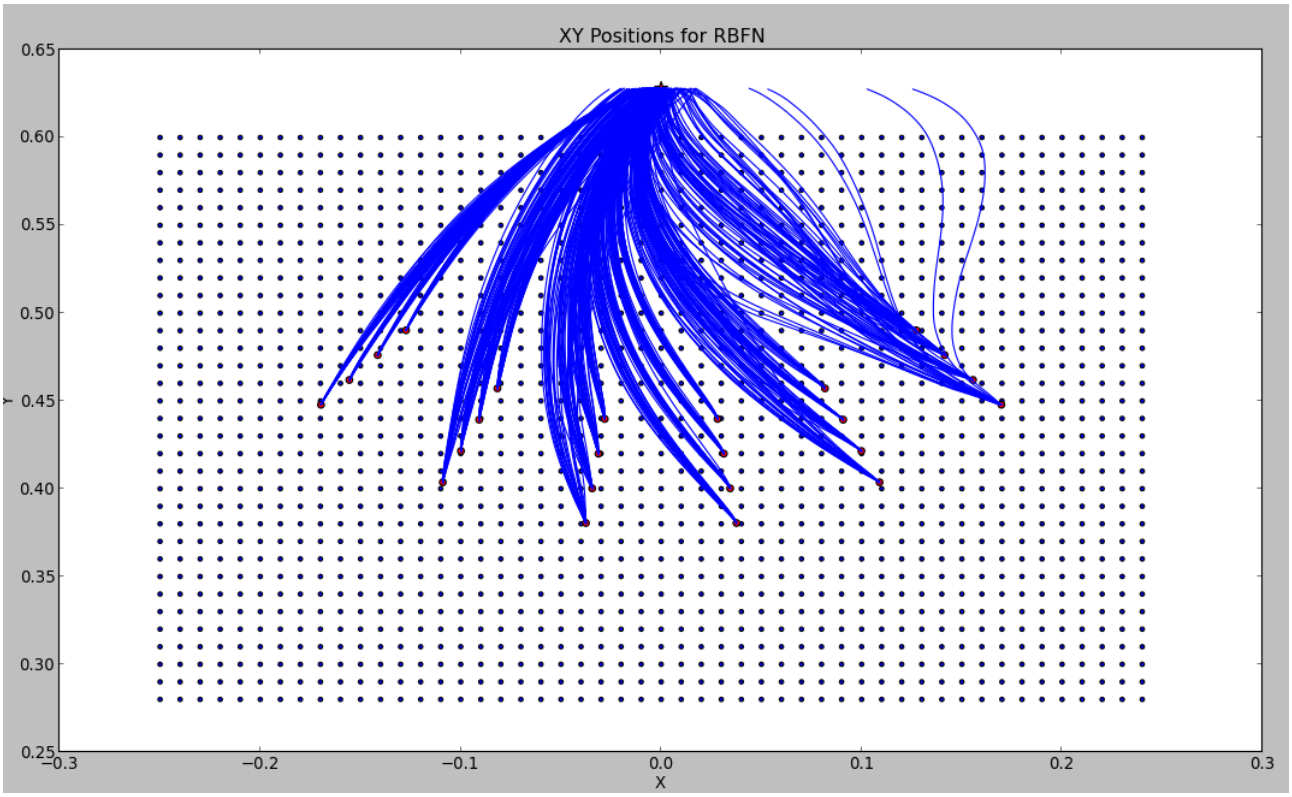
2 Data from the RBFN controller

The controller is trained from the previous data using a Radial Basis Function Network (RBFN). There are 3 Gaussian basis functions per dimension of the state space (dim = 4), thus 81 basis function per muscular function, thus 81x6 = 486 basis functions in total.

The Gaussian basis function are evenly spaced into the state space. The width constant of each Gaussian is set to 3 times the space between two Gaussian function centers.

1 Trajectories

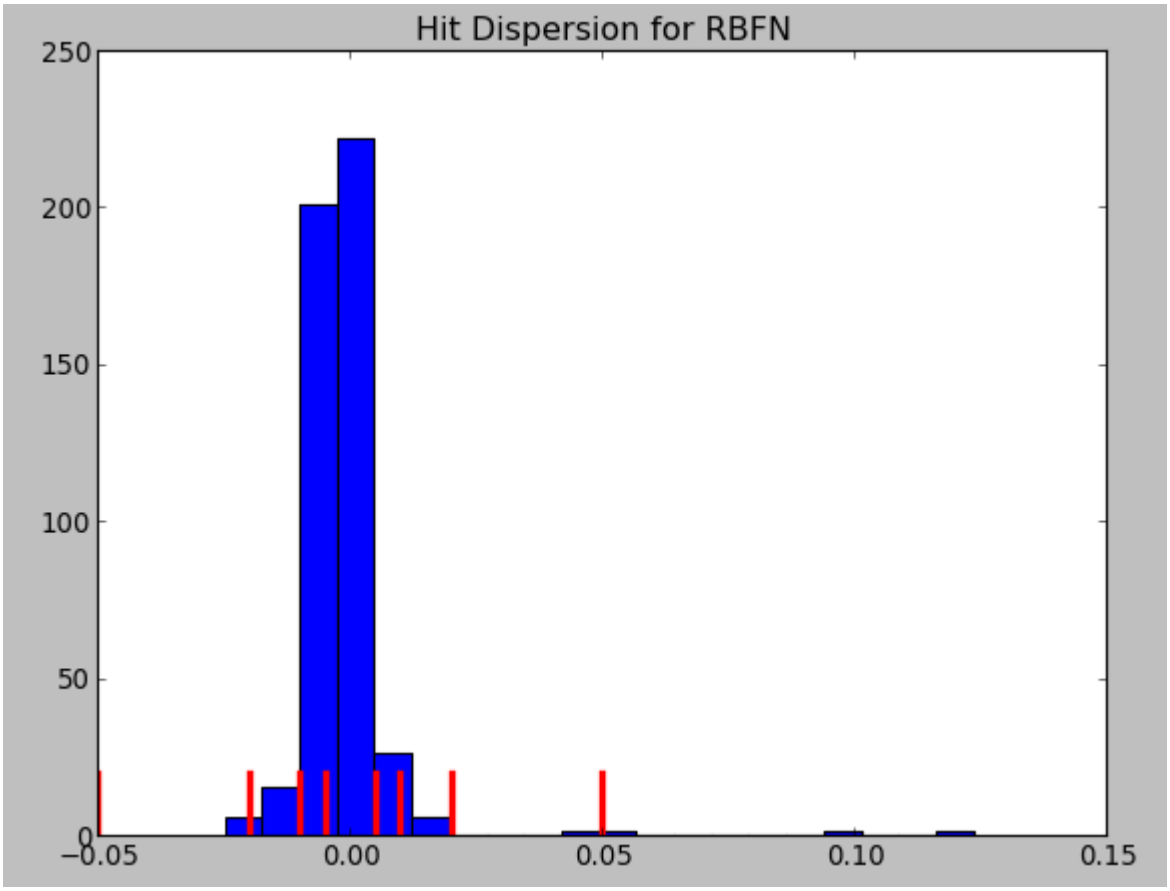
There are 20 trajectories for each **starting point of the protocol (explain)**



One can see that some trajectories starting from the right deviate to the right and that the hit dispersion when crossing the screen is quite large.

2 Hit dispersion

The hit dispersion that one can infer from the trajectories above is better seen with this image.

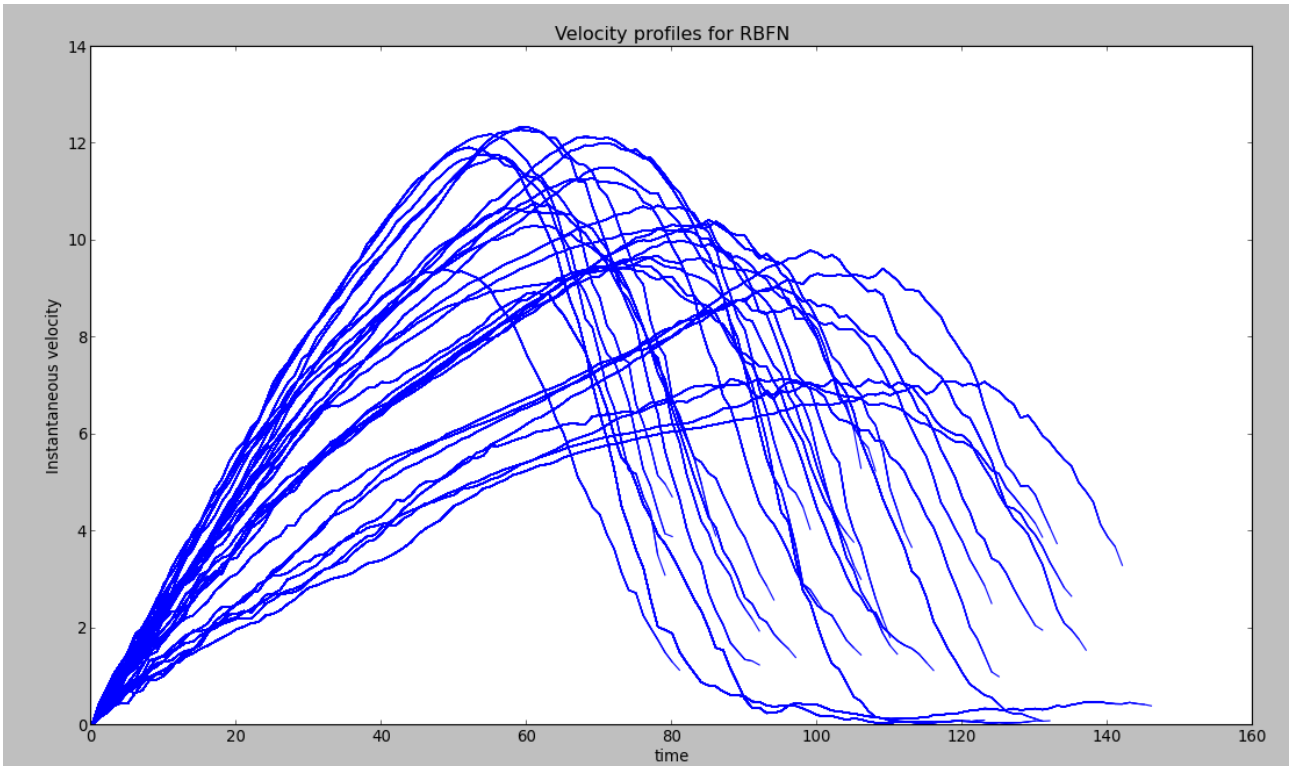


The red bars correspond to the boundaries of the 4 different targets. One can see that the controller is quite focused, but not enough to succeed on the smallest targets.

A key result in the paper will be to show that this hit dispersion has been optimized for each target size. More on that on monday !

3 Velocity profiles

As for the Brent controller, only 15 % of the available profiles are shown. Some finish later (at around 200 times steps) and with a close-to-null final velocity.



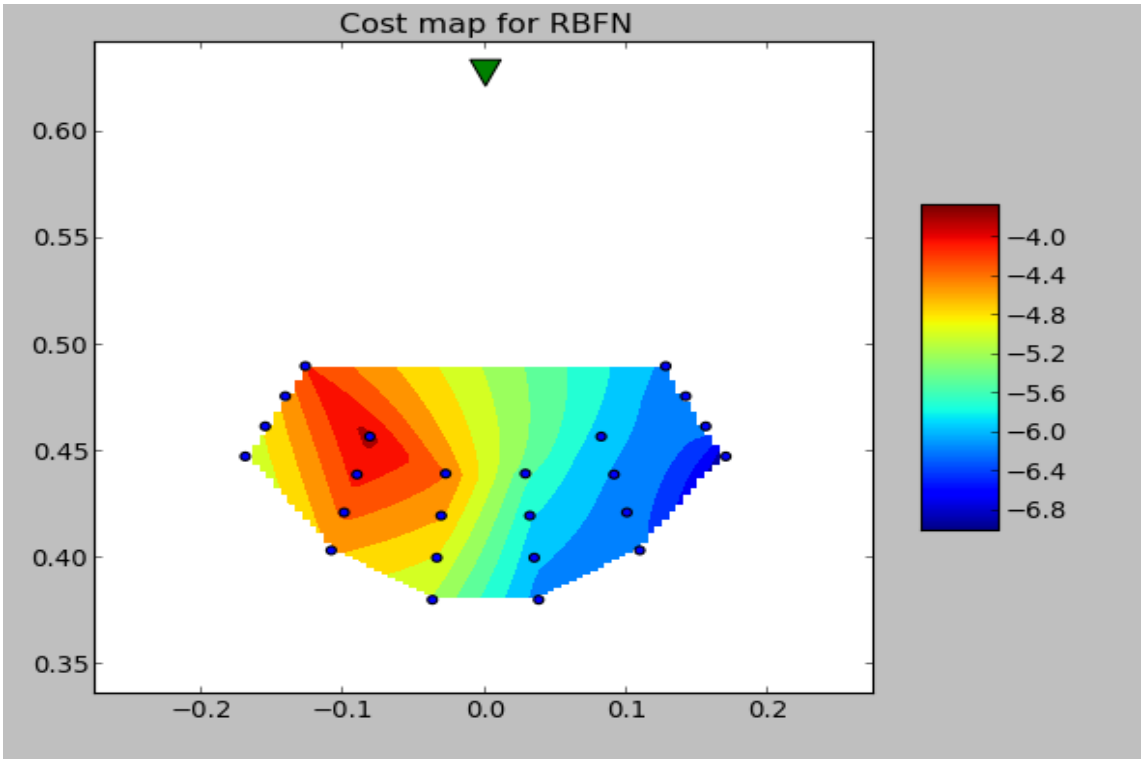
One can see that many trajectories stop abruptly while the velocity is decreasing, which corresponds to a fast hit on the screen. One can also see the effect of motor noise, which results in a « noisy » velocity profile.

4 Muscular Cost Map

This section is more there to convince us that the RBFN controller works.

Blue dots correspond to starting positions of the protocol.

The first image gives the global muscular cost, without taking the reward into account.

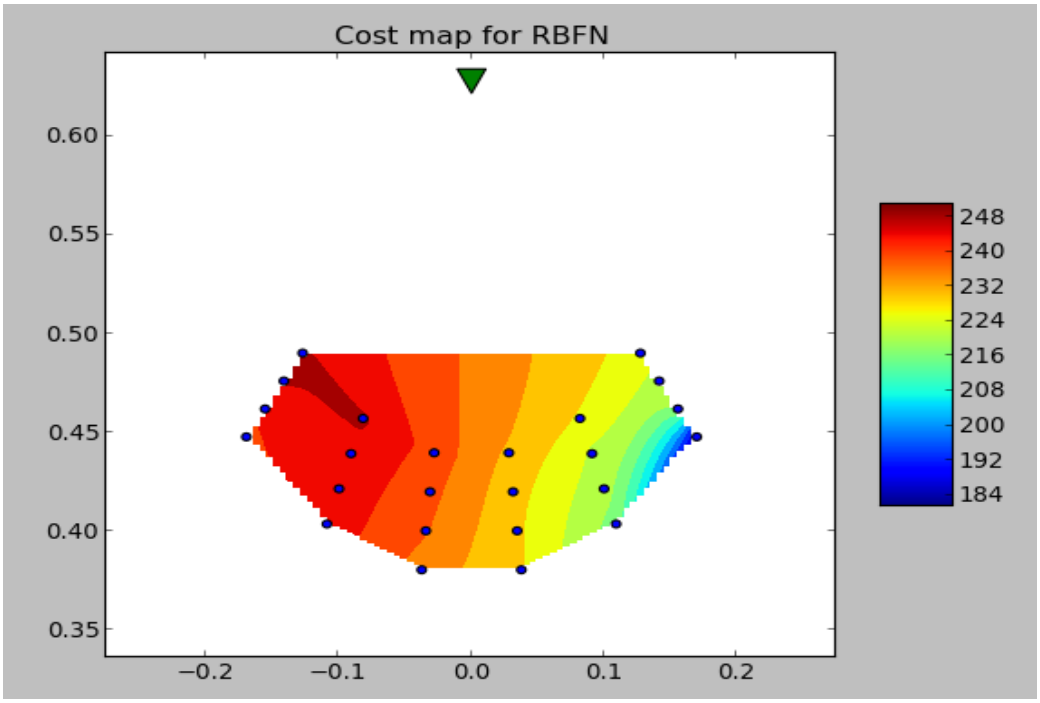


The second image gives the cost including the reward.

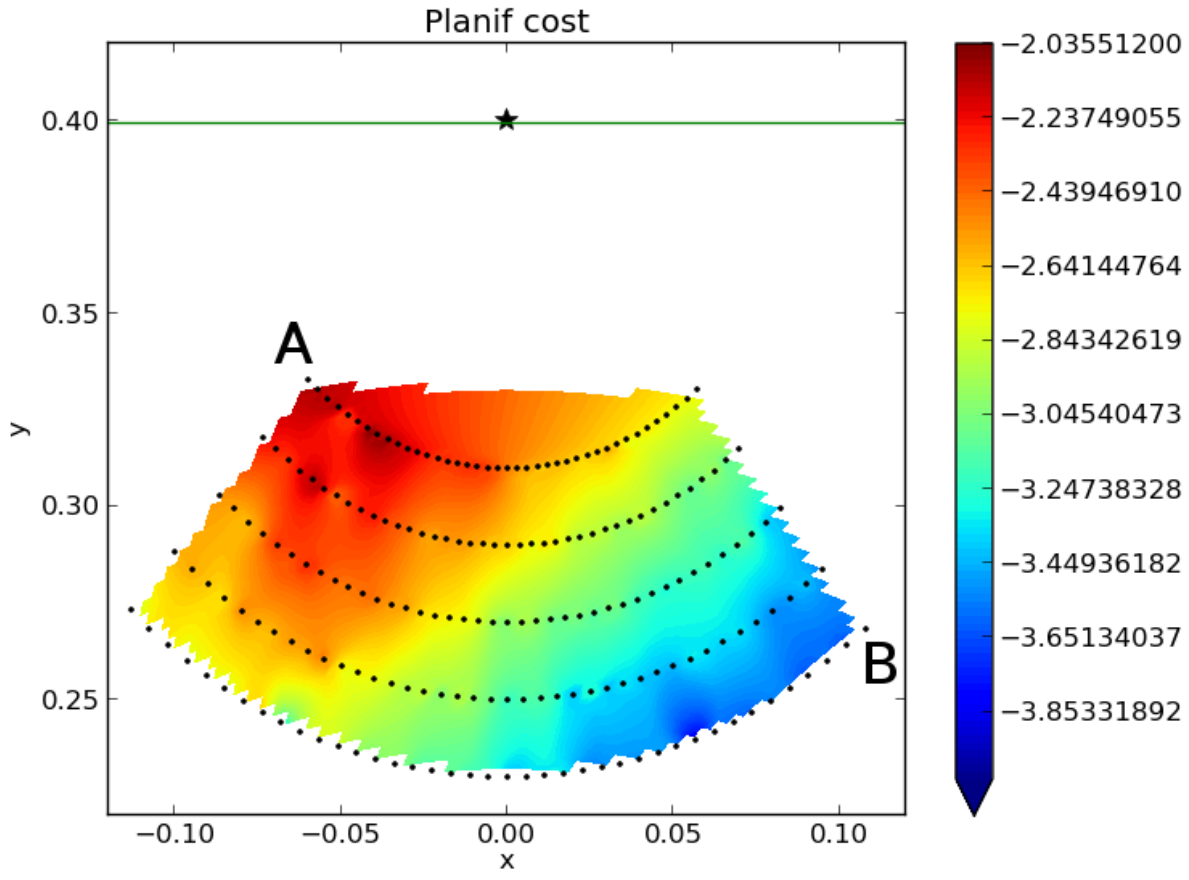
The value of the immediate reward is 300, the discount factor is 0.998, thus the highest reward a controller can get if it always succeeds in less than x steps is 300*pow(0.998,x). For x = 100, it is

245. Furthermore, the muscular cost is between -4 and -7, so the optimum cost for 100 time steps is around 240.

One can see on the second image that this optimum is close to being reached.

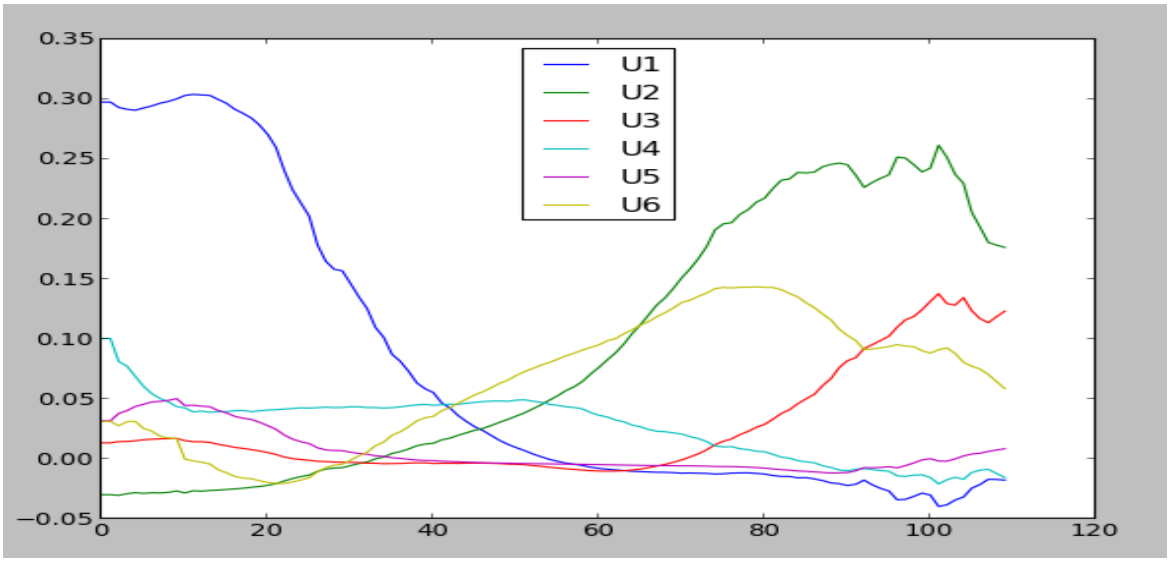


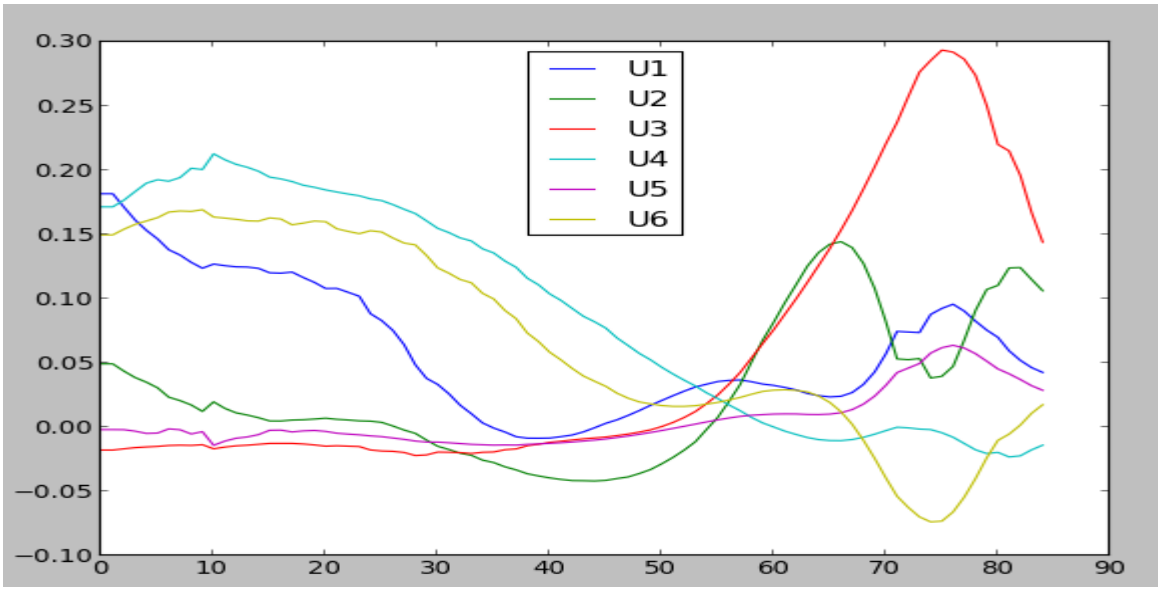
The third image is Fig4 from the previous paper, without taking the reward into account, it is there to show that it is similar. Note however that the scale of values has a little changed, due to a different tuning of the parameters. Note also that it is nicer, because the map is made out of more points. I'll probably improved the ones above.



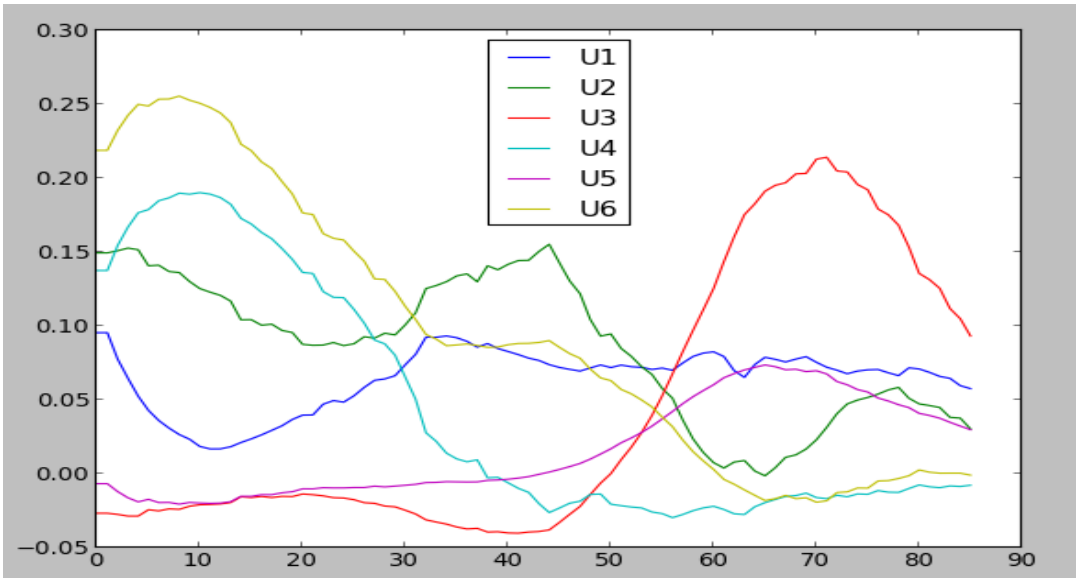
5 Muscular activations

We find again strategies that are similar to those of the Brent controller :





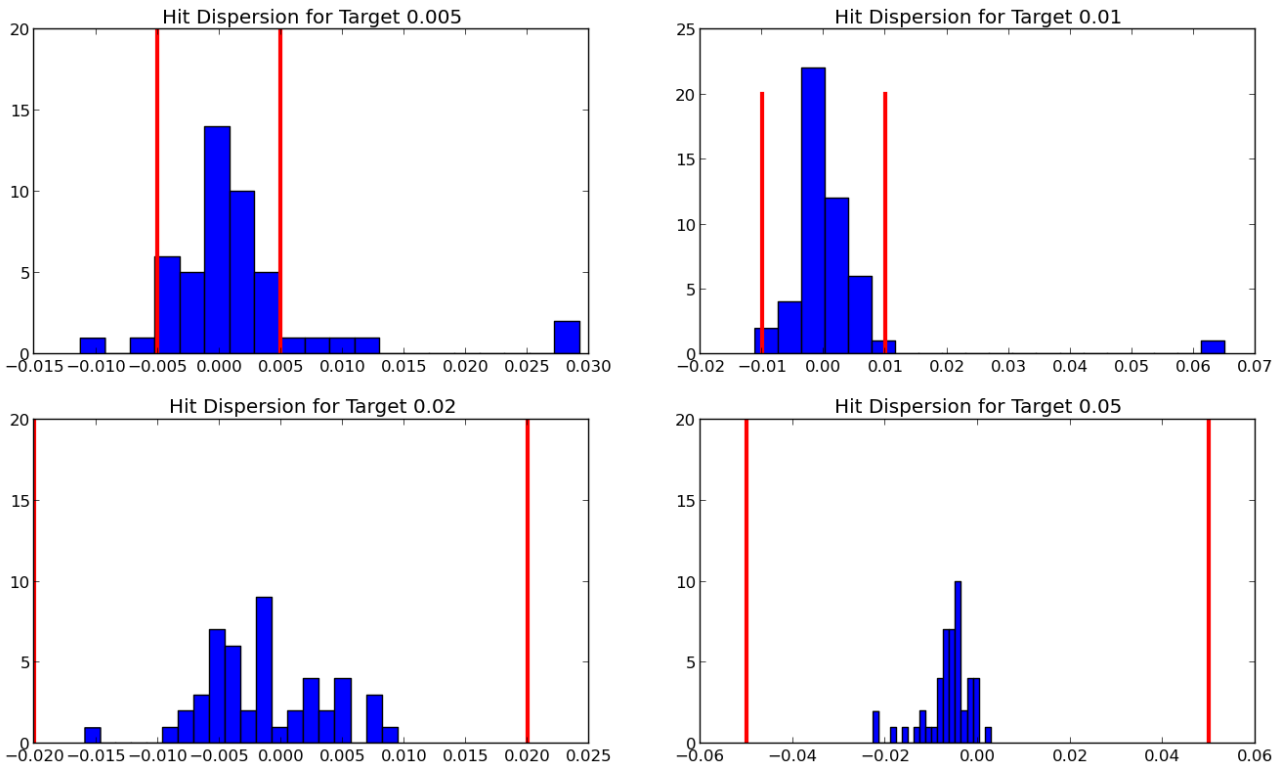
But some of them are just ugly :



3 Data from the 4 RBFN controllers optimized with CMA-ES

1 Trajectories

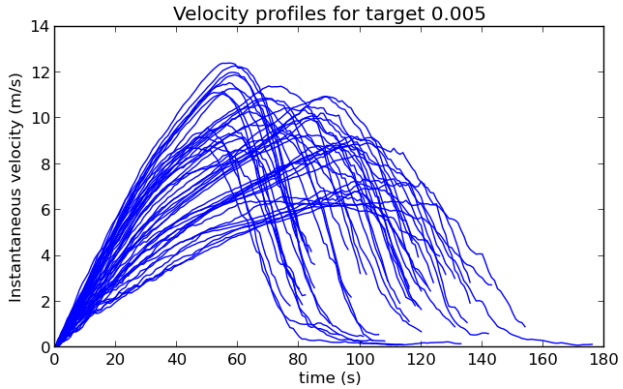
2 Hit dispersion

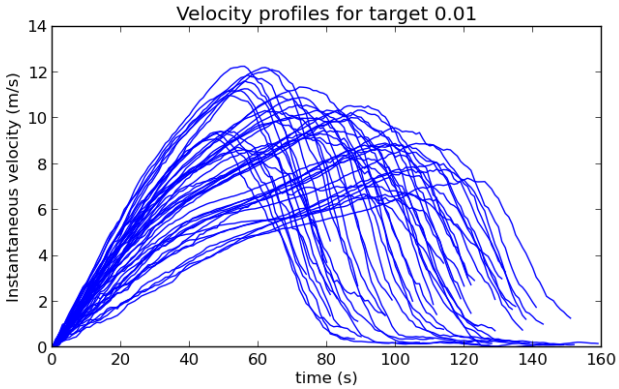


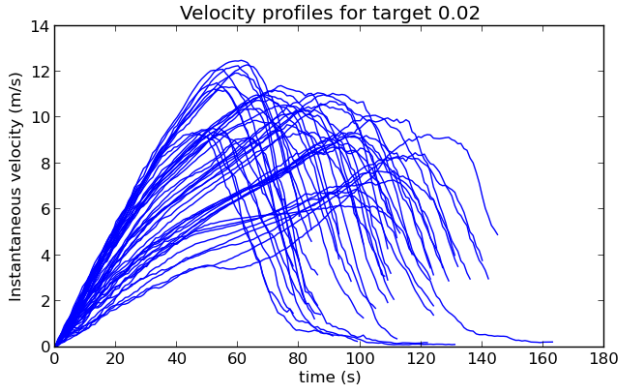
This one looks great : as expected, it succeeded in being much narrower for small targets

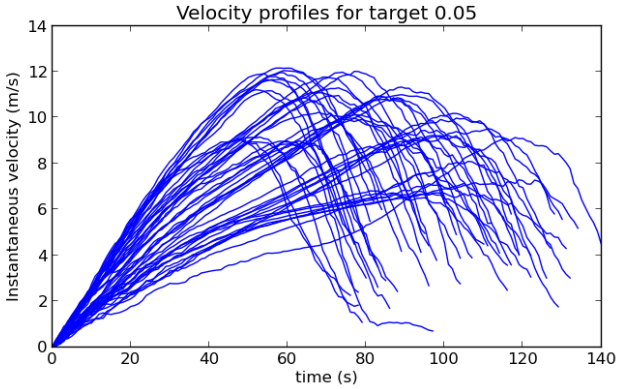
3 Muscular activations

4 velocity profiles



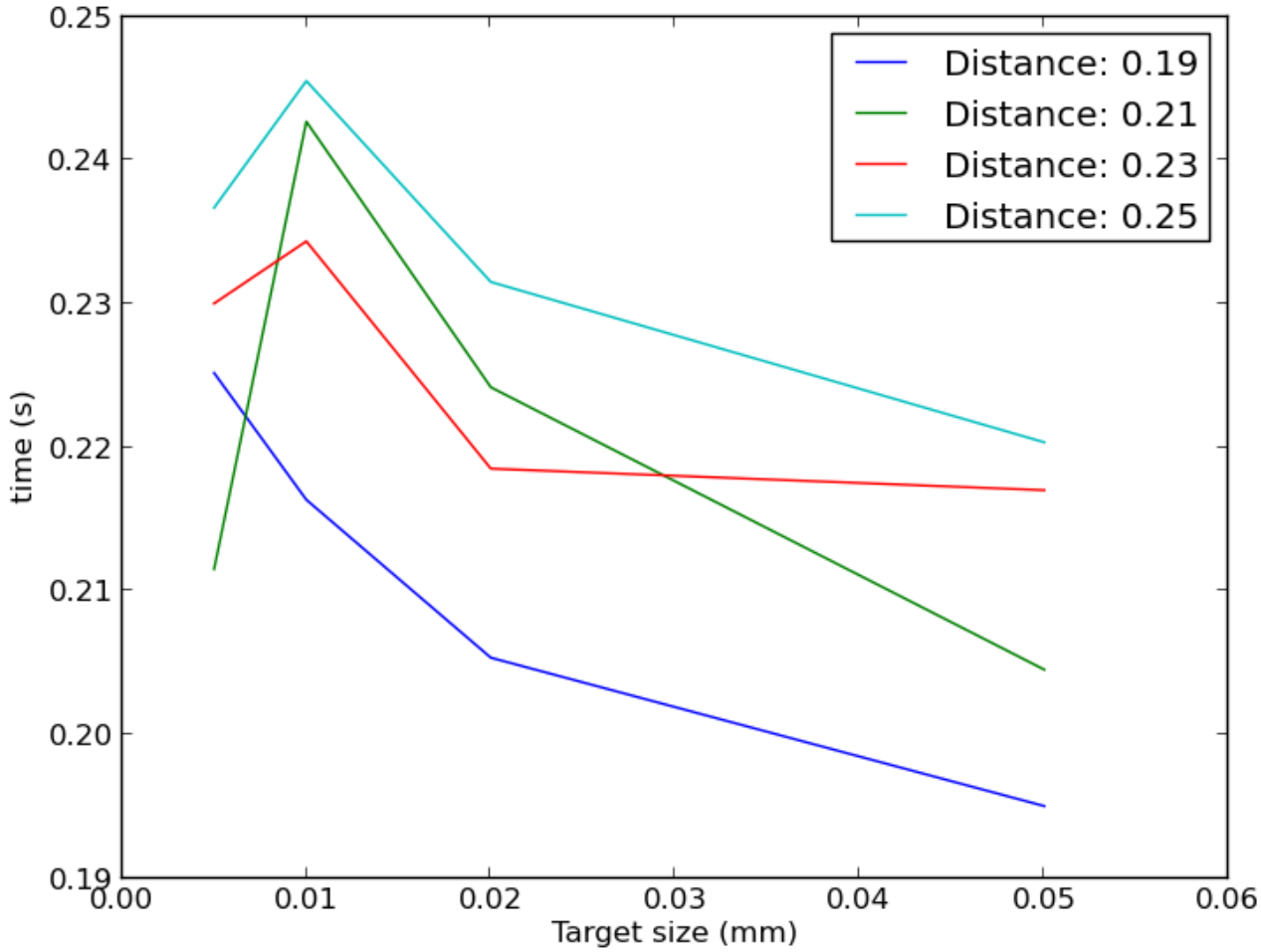




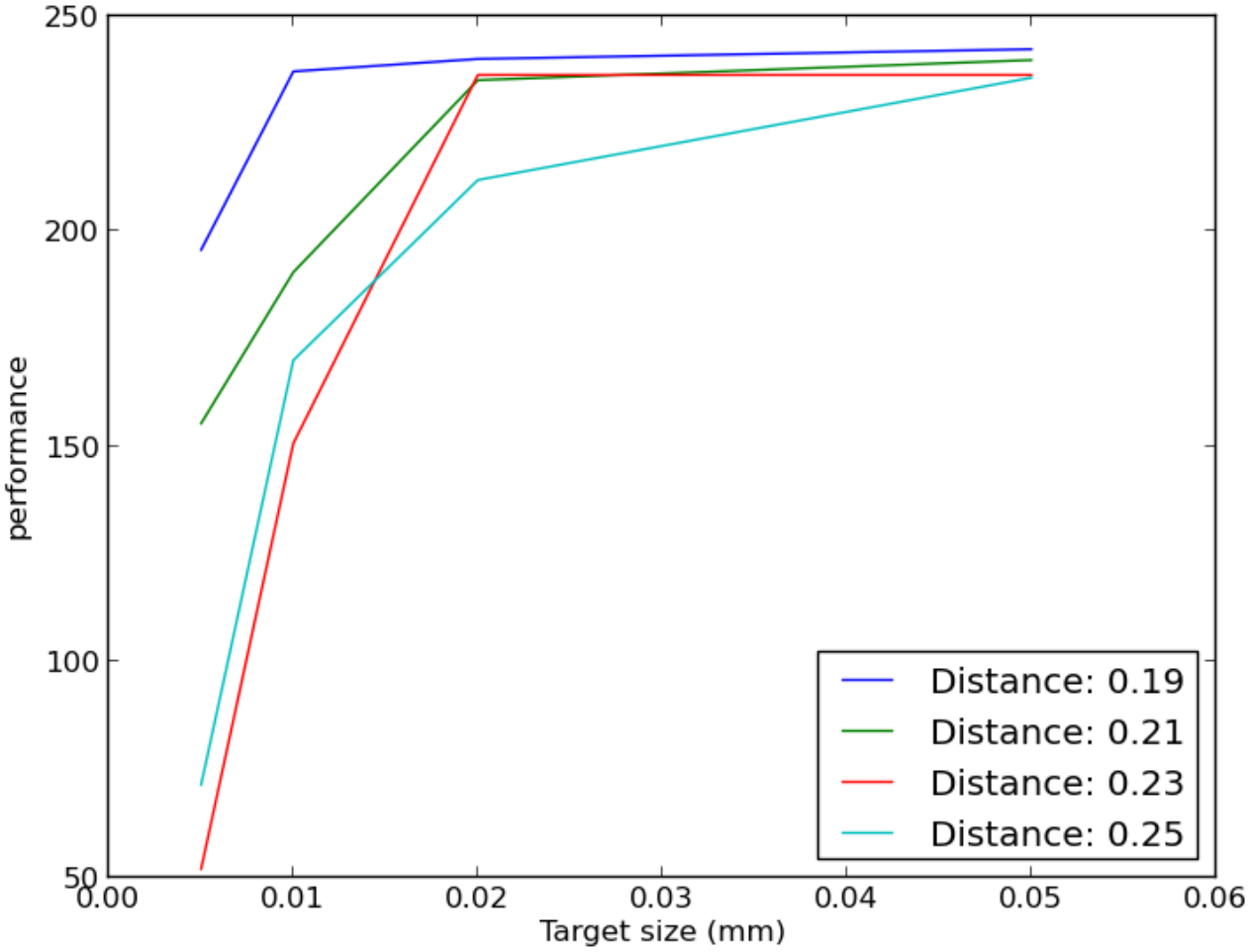


5 Muscular cost maps

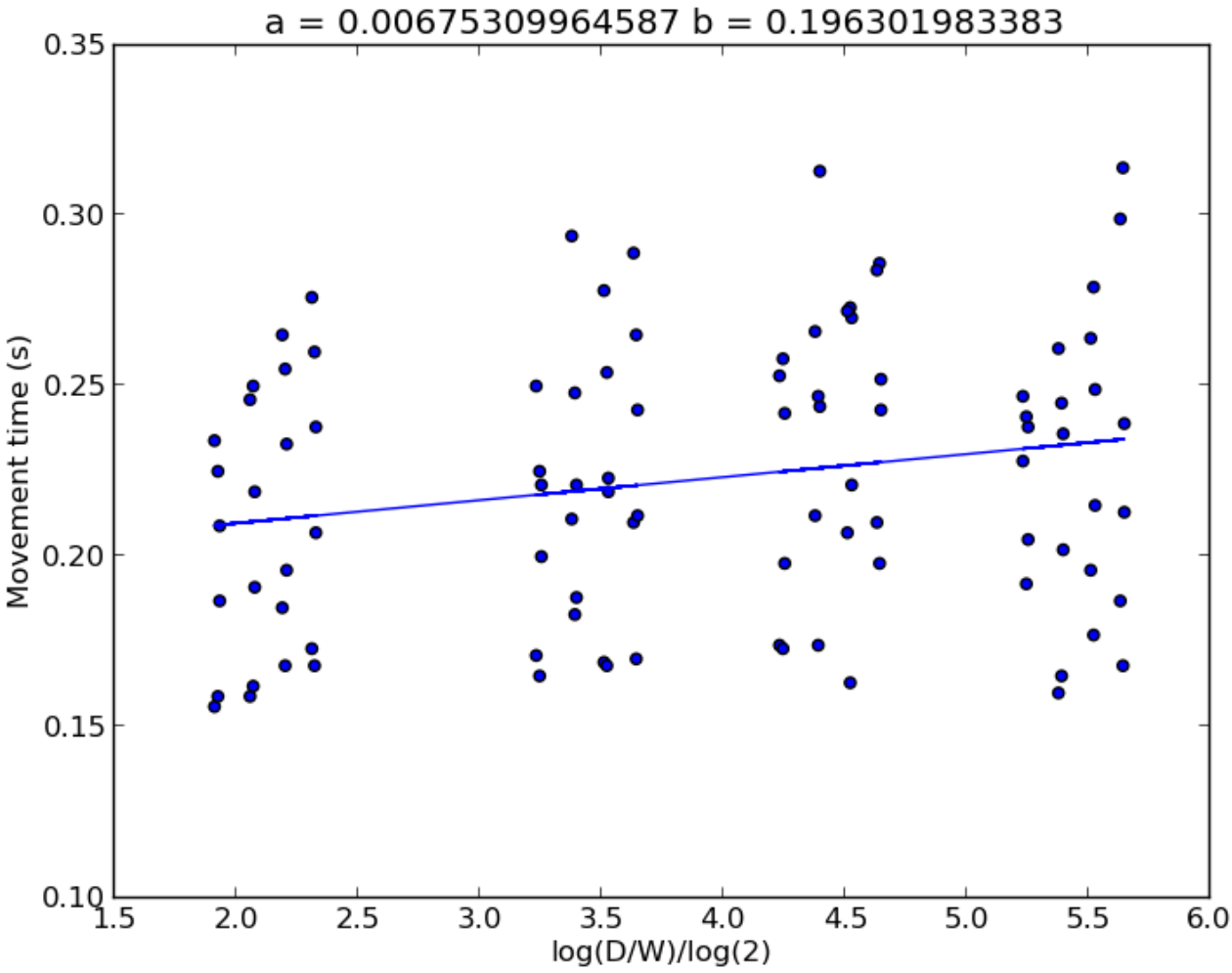
6 Time versus target size



7 Perf versus target size



8 Fitts' Law



4 Methods : state estimation

There are two sources of inaccuracy, corresponding to two critical parameters in the model: sensory delay (the corresponding parameter is a number of time steps) and motor noise (the corresponding parameter is a coefficient of propoprtional noise added to the muscular activation signal emitted by the brain).

If we follow the model of [Scott, 2004], the CNS receives an efferent copy of the muscular activation signal sent to the muscles before some motor noise is added to this signal.
Thus the best the CNS can do to compensate for the sensory delay is infer an estimated current state from the information of the noiseless muscular activation signals that where sent to muscles and from a known forward model of the musculo-skeletal system.

So our model simply stores the noiseless muscular activation signals that where sent to muscles backward in time up to sensory delay, and computes an estimated state from the knowledge of the delayed state and those muscular commands that were sent in between.

The model does not incorporate any kind of Kalman filtering, but our way to infer the estimated state corresponds to the best guess (it assumes that all noise signals will compensate for each other, which is what can expect on the limit if noise is centered on 0).

5 Remarks, thoughts

Note about the time of movement : all trajectories finish before 300 times steps, i.e. 600ms. This looks much faster than human movements, isn't it ?

I believe state estimation is critical to the model.

If state estimation is accurate, then the controller can reach quickly even to very small targets. So, if we want it to slow down for small target, we need that large speed generates a large deviation in state estimation and that slowing down improves this estimation.

To get a large deviation of the estimated state from the true state, one can either set the motor moise parameter to a large value, or consider a large delay.

The limit on how accurate the model can be is reached if it has a perfect estimation of the current state. If motor noise is too large, this limit might not be small enough for a small target: the deviation generated on one step by the noise would be too large to stay between the target boundaries.

But using a large delay is computationnally expensive, because the model has to infer a long chain of next states from the delayed state and stored noiseless muscular activations.

Thus, if a large deviation is required, a compromise has to be found between a large delay and a large motor noise.

We will now on monday whether the compromise was properly set !:)