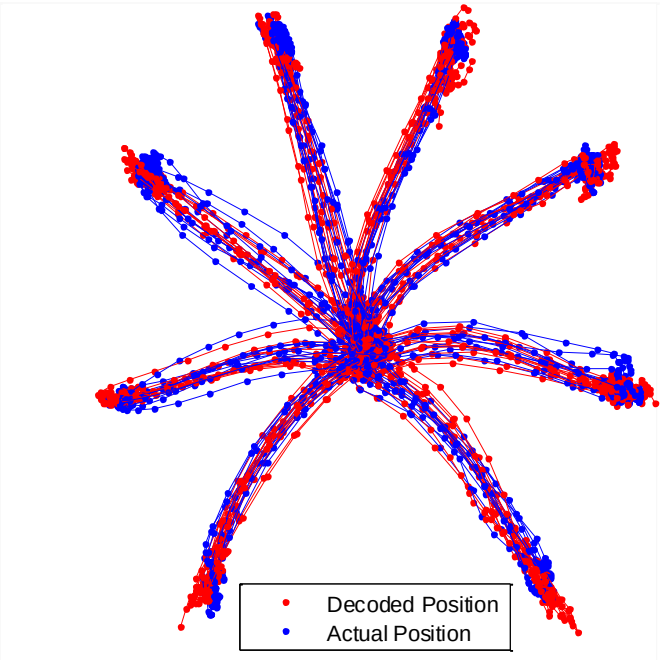
Monkey spirit

Antoine Lizée

29/04/2012



This brief report presents several points in the quest of decoding that has been undertaken by the group “Mashed Potatoes”. Some general points are not detailed in order to focus on more advanced, specific issues. Part of the detailed discussion is held in the Annexes in an attempt to keep the report under the word limit.

1. Introduction*

1.I. Neural decoding is a crucial step of BMI

I won't expose the usefulness of decoding algorithms in general, and applied to brain machine interfacing in particular. These heavily computed approaches have undergone a huge leap forward recently, enabling development of efficient neuro-prostheses recalling some Science-fiction movies or books. This is interesting - and fun – to try by ourselves to develop an actual real time decoder. The overall final performance of this short time project proves the potential of such an approach to brain machine interfacing and promises huge achievements in neuro-prostheses or derivatives in the next few years – provided that the monitoring hardware follows.

1.II. Description of our exercise

The decoding exercise is split in two tasks. The first is a simple *classification*; the second one is a *continuous decoding*. Both of these tasks need to be undertaken under the light of our biophysical problem: we have data collected by an array of 100 intracortical electrodes from the motor and premotor area of a monkey performing a 1-among-8 target tracking task in a 2D plane. We are provided with the digitalized spike activity in 1ms-bins along with the recorded position of the monkey's hand. The data expands from 300ms before the start of the movement to 100ms after the end of it. The development time was about one month, with an intense period toward the end for the completion of the second task.

1.III. Teamwork

We did our best to share the workload, and to communicate and explain between us regarding the different issues. Nevertheless, due to huge differences in competences – mainly due to difference in people's background, the development has actually been carried out by a minority.

2. The discrete neural decoder

2.I. Description of the problem – development strategies

The goal is to create a discrete decoder that will anticipate which target the monkey is aiming thanks to the recorded activity from $t = -300ms$ to $t = 0$. To make the problem more difficult, we are limited to the activity recorded from ten neurons, which add a dimension to the classification problem: we need to select the best 10 neurons from a set of 98 neurons recorded.

Thus, the development of the decoder had two different parts: first the design of a good classification algorithm, then a process of choosing the right set of 10 neurons. Although these two parts have both an impact on performance, the set of neurons was homogeneous enough to enable us to "split" the tasks accordingly. Indeed, it appeared that the difference of optimal sets for each algorithm was small, and a generic set of 10 "good" neurons was a good benchmark to compare our algorithms. To make this comparison easy between different algorithms, we chose from the beginning a versatile architecture for our code which enables us to simply write a new file for each new type of algorithm that we want to test, and switch easily thanks to a method

* Image of first page is reproduced and presented in the performance analysis (Annex 2), as Figure 7.

argument. Once the complete framework for the training, decoding and testing was set up (including a complete cross validation function) implementation and testing of a new algorithm was a matter of a few lines.

2.II. Algorithms

This section describes the algorithms that we have tested for the classifier.

2.II.a K-nearest neighbours

Because of the high dimensionality of the set (dim=10 at least) and the relatively low number of points (100 $trials * 8\ angles = 800\ points$), we dismissed this method. We were to later regret this choice since it finally turned out that the most efficient classifier features a mixture of classifiers including a k-nn.

2.II.b Population vector decoding

We tried at first to look at this "classical" algorithm for decoding neural activity. The main hypothesis is that the direction of the aimed trajectory is encoded at the brain level through a virtual vector, and that the activity of the neurons encoding the direction information is proportional to the scalar product between this virtual vector and a direction vector which is a characteristic of each neuron. There are many things to say concerning these decoding algorithms, and many things to do to improve their meaningfulness and performance. The concurrent development of the generative algorithm had soon proven that even in its simplest form (Gaussian hypothesis) it will outperform by far any version of the population vector decoding unless a miracle; we thus stopped the development of this algorithm before implementing advanced features. I will only describe what we implemented. The (more) interesting discussion about the potential enhancement of the algorithm will be held in the Annexes.

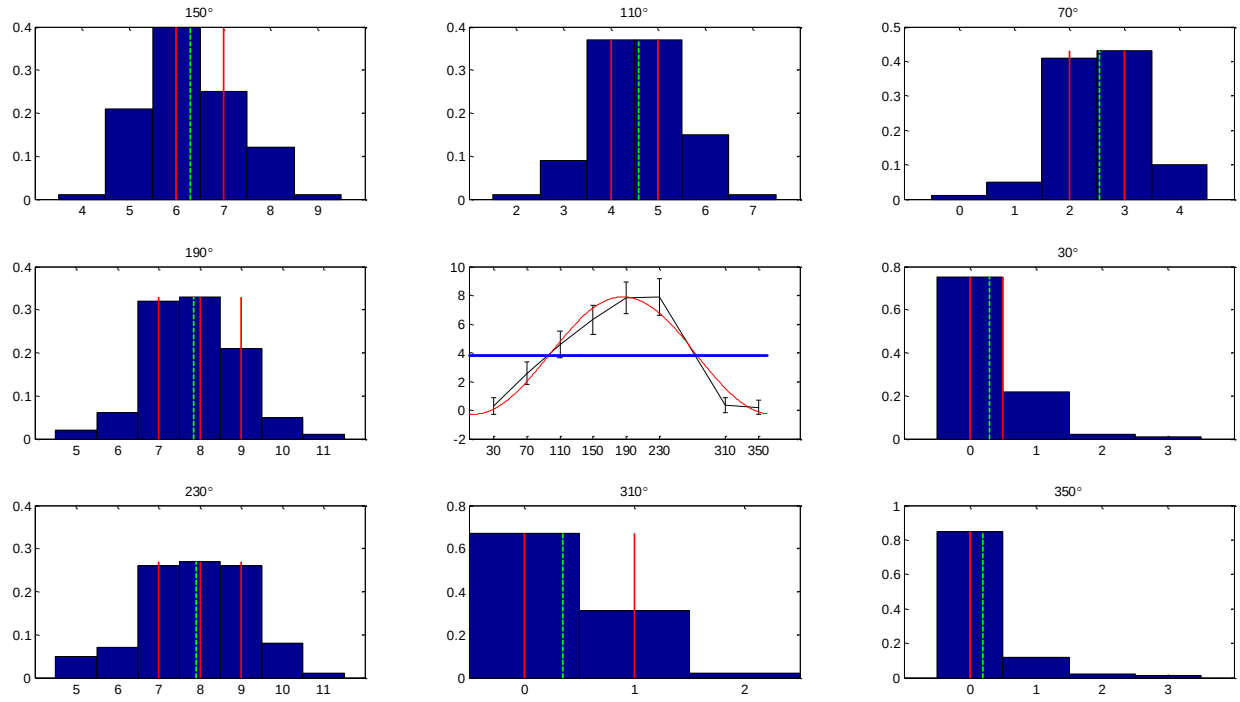


Figure 1. Complete tuning curve representation of neuron 33 based on the total spiking activity of the neuron on the monitoring period (from $\boldsymbol{t = -300ms}$ to $\boldsymbol{t = 0}$) The global tuning curve is fitted with a cosine for the population vector decoding, error bars display the standard deviation. "Sub-tuning" curves features the mean in green.

The first basic version of the decoder had for main hypothesis the scalar product. This is actually a very strong hypothesis on the tuning curves of the neurons, which need to have a cosine shape of one period across the whole range of direction, as observed in Figure 1. The parameters of the offset, amplitude and phase of the cosine are fitted during the training by a least square regression. For the decoding, we first compute the projection values for each neuron, and we then do an inversion that will give one orientation from these 10 projections†.

2.II.c Generative algorithm

The generative algorithm uses more information than just the tuning curves by taking into account the actual distribution $p_a^n(r)$ of values of activity r for each angle a. This approach gives more flexibility concerning the hypothetic response of a neuron and especially does not make any assumption on the shape of the tuning curve. Our first attempt was the simplest. During the training of the algorithm, the distributions $p_a^n(r)$ are fitted with Gaussians. During the decoding, an estimation of the likelihood of each class (here the 8 target angles) is computed using the Bayes rule:

$$p(a|r_{tot}) \quad \alpha \quad p(r_{tot}|a) = \prod_{n=1..10} p_a^n(r)$$

For each angle a, with r_{tot} the 10 dimensional response vector, and with "α" denoting proportionality. This is true only because the probabilities of occurrence of all angles are equal. Having computed all the likelihoods, our generative algorithm is a simple likelihood maximisation which selects for the decoded angle:

$$a_d = arg\max_a p(a|r_{tot})$$

There are two main ways of improving the algorithm performance. First, improve the fitting of the distributions, second, modifying the choice of a_d, through a cost function for instance. The latter has not been implemented because it seemed that the maximum likelihood was the good approach considering our system. Concerning the fitting, it is hard to find a better "fits-all" distribution than the Gaussian with adjustable mean and standard deviation. The best would be a sort of Gaussian with additional "tuning" of the third and maybe fourth moment of the distribution, in order to fit a potential asymmetry and "skinniness". Without any simple solution provided by MATLAB, and under the time constraint, we only tried to fit a "kernel" distribution that extrapolates the histogram into a smoothed distribution. This has given slightly worse results, mainly because of over-fitting.

2.II.d Multi-fit generative algorithm

Gaussian distributions are quite powerful models for our problem but they suffer one principal limitation being their symmetry, whereas they are fitted to values of activities that are bounded to be positive. This means that for small values of activities, Gaussian distributions are not appropriate for fitting $p_a^n(r)$, and this is very important since the information provided by a neuron is often concentrated in this low response along some directions. Gaussian distributions tend to attenuate this specificity as it can be seen in the example in Figure 2 for the directions 1, 7 and 8. For these directions, the activity is really low compared to the other ones, but the Gaussian fit doesn't catch it. The hypothesis of a Gaussian distribution is rejected with a very small value, whereas for all other direction it cannot be rejected (p>5%). From these

†The details are presented in Annex 1

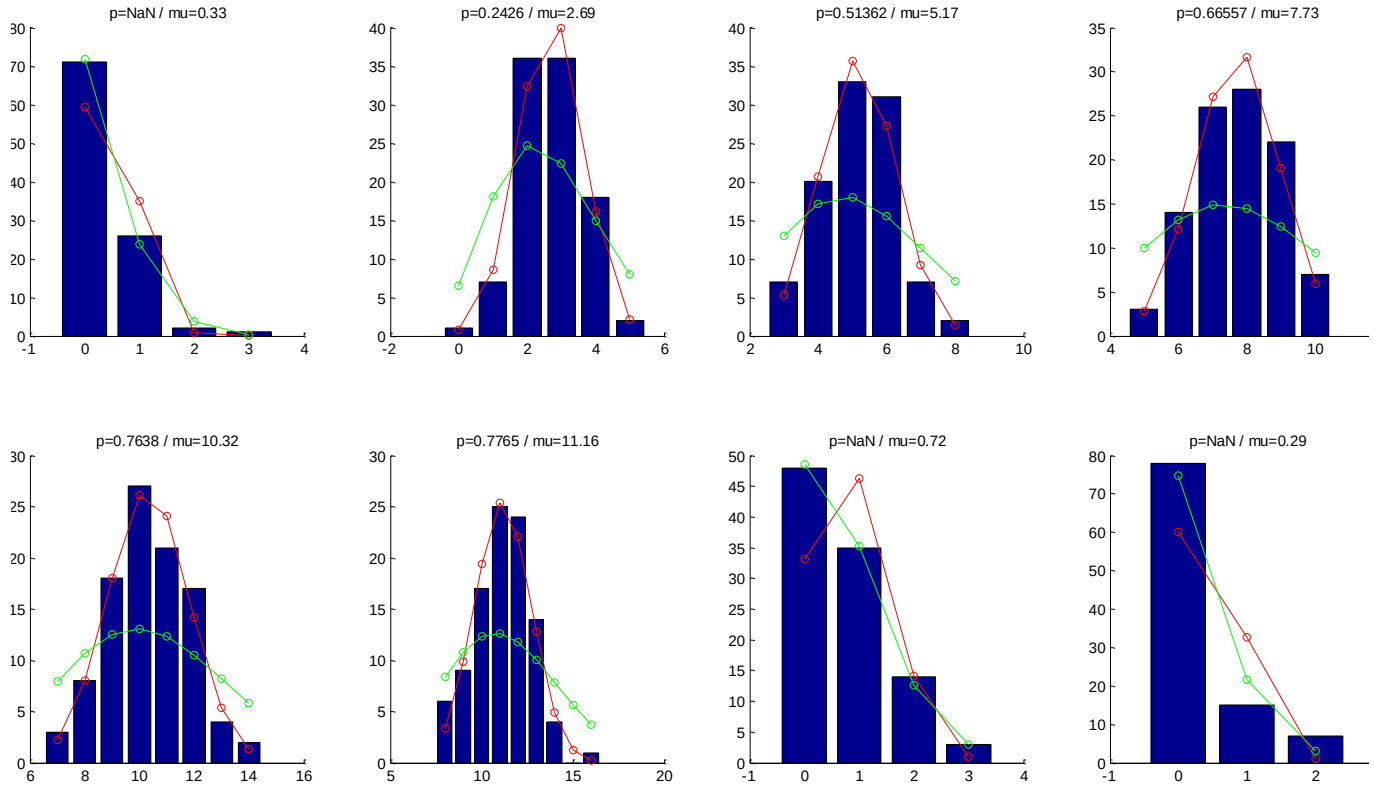


Figure 2. Distributions of activity in all 8 directions for neuron 69. Experimental distributions are fitted with Gaussian distributions (red) and binomial distributions (green). Mu is the mean of activity, p is the p-value of a constrained chi-square test to test the hypothesis of generated discrete data from a Gaussian distribution against data generated from another distribution. The Gaussian fit cannot be ruled out except for low activity, where the binomial fit is more efficient.

observations, we decided to fit a binomial distribution to any distribution of activity that would have a mean below 1, enhancing the accuracy of fitting and the decoding performances. Final performance on the inner testing was more than 96%, which is a really good result considering the fact that we have only access to the data of 10 neurons, *prior* to the movement offset.

2.II.e Multivariate Bayesian algorithm

Instead of considering the activity of each neuron independent of the others (by considering the product of the 10 probabilities as the total probability), we tried to fit a multivariate Gaussian to the 10-dimensional distribution of activity for every angle. We need to avoid over-fitting from adding some 45 additional covariance values in addition to the 20 mean and variance parameters, for each angle. Thus, we tried two things: first to keep only the neurons that seemed not independent, by setting to 0 all covariance values under a certain threshold set as follows:

$$\sigma_{n_1 n_2}^{max} = \gamma * \sigma_{n_1} \sigma_{n_2}$$

Then, the final covariance matrix was a linear combination of such a truncated matrix Σ_γ and a diagonal matrix with only the variances:

$$\Sigma_{\text{final}} = \lambda * [\sigma_{ii}]_i + (1 - \lambda) * \Sigma_\gamma$$

I implemented this multivariate approach in addition to the multi-fit: the "binomial" neurons for each angle were left out in the multivariate fitting process. This approach has been really disappointing as there was only a decrease in performance for any value of (γ, λ) departing from

$(0^+, 1^-)$. The uncorrelated Gaussian hypothesis proved out to be more robust than any dose of cross correlated views. I haven't found any persuasive explanation to the lack of additional information provided by significant cross-correlation!

2.III. Selection of neurons

We developed many approaches to select the best ten neurons for each algorithm. This is a non-trivial problem, since brute force testing is immediately excluded: the number of set of 10 neurons among 98 is of the order of magnitude of 100^{10}. The main challenge is then to induce the performance of a neuron in a set of ten from its characteristics, depending on the set. We used this approach to make a first quantification of the "good" and "bad" neurons, relying on the cosine fitting[‡] and later on an algorithm relying on ANOVA test[§]. We then developed consecutively two different machine-learning algorithms: the last and most efficient one was a really simple genetic algorithm[‡].

The first automated algorithm enabled us to further the investigations and to find our first "good" sets of 10 neurons that have been then used in the GA as a "pool". This algorithm is inspired by complex systems in Physics, and features a probabilistic temperature-related behaviour to "wander" on the energetic landscape of the performance of the 10-neuron set. To avoid the lengthy cross-validation at each step, the algorithm relied on an estimate of the real performance of the 10-neuron set, computed from a bank of cross validation values for all the couples of neurons. Figure 3 is a graph that tries to sum up the different steps that defines the evolution of the decaying system. Further testing of the algorithm and tuning of its aging parameters has been stopped in favour of the Genetic Algorithm which was more easy to use and to implement, although more demanding in computational power (but this has been mitigated by implementation of parallel processing).

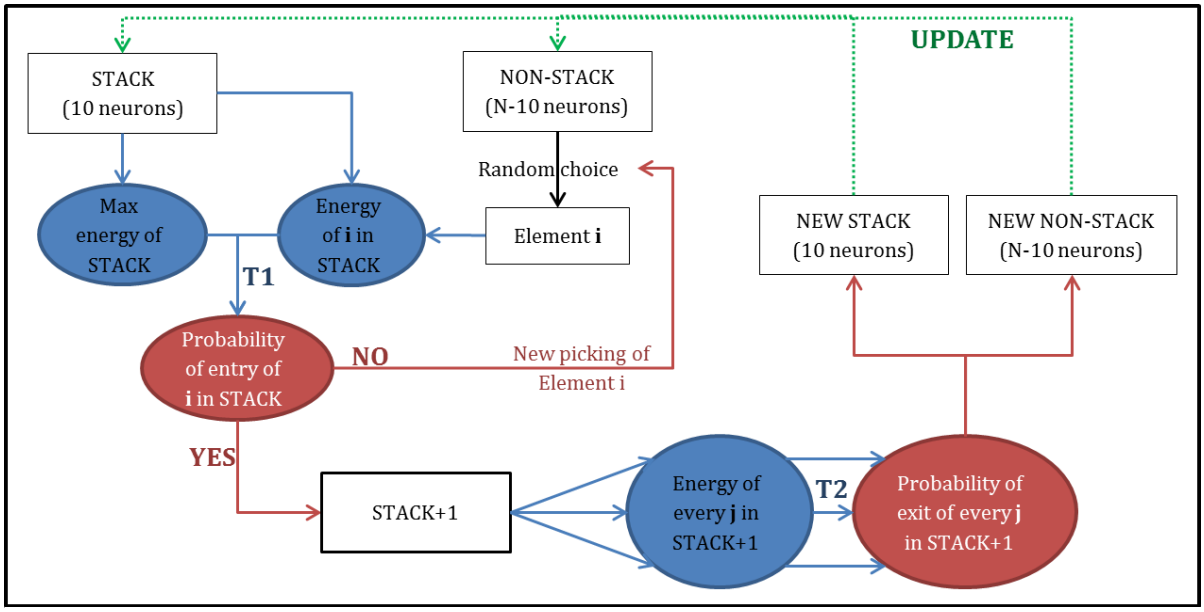


Figure 3. Graph illustrating the aging step of our decaying algorithm. The aim is to find an optimal stack of 10 neurons by repetition of this aging step. Blue arrows denote computation steps, red arrows denote choice after probability testing. T1 and T2 are the entry and exit temperatures.

[‡] See the report of Octave Etard for more details on the first selection and on the generic algorithm.
[§] This second selection process based on ANOVA is not detailed. It provided us a good way to quantify the discrimination power of the neurons along each direction, and thus to detect "good" neurons.

3. Trajectory decoder

3.I. Algorithm

Due to time constraints, we did not operate a comparison of algorithms as for the first task. We implemented a classical Kalman filter, enhanced by a Mixture model. To be more relevant, we were limited in our exercise by the knowledge of only the prior spiking activity, which started 100ms before the actual start of the movement.

3.I.a Classical Kalman filter

The decoder had a trajectory model and an observation model. The trajectory model $\widehat{\boldsymbol{T}}$ gives an estimate of the position from the *physical description* of the previous trajectory (position, velocity, acceleration…). The observation model $\widehat{\boldsymbol{O}}$ links the spiking information to the current state of trajectory. Both models are then put together to give, step by step, an estimate of the total trajectory following this process for each step[**]:

1. From state S_{t-1}, gives an estimate $\widetilde{S_t}$ of S_t through the trajectory model : $\widetilde{S_t} = \widehat{\boldsymbol{T}}(S_{t-1})$
2. Compute an estimated observation : $\widetilde{o_t} = \widehat{\boldsymbol{O}}^{-1}(\widetilde{S_t})$
3. Compute the final estimate of the state with both models thanks to a linear gain $\boldsymbol{K}$, and the real observation o_t : $S_t = \widetilde{S_t} + \boldsymbol{K} * (\widetilde{o_t} - o_t)$
4. Update and store the gain.

We used a linear centred Gaussian model, with a describing state including up to the 4th order derivative of the position for the best results. The incremental time $\Delta t = 20ms$ between each estimation step was chosen to match the testing procedure of our algorithm for more simplicity.

3.I.b Mixture model

On the top of the Kalman filter, we took advantage of our classifier to train 8 different sets of parameters for the Kalman filter, one for each possible trajectory, and use an additional classifier algorithm to determine which Kalman filter should be used during decoding. There were two phases:

1. Up to the tenth estimation step (i.e. before 200ms into the trajectory), we incremente all our 8 sub-models separately. The output position follows a Bayesian approach:
$$(x,y)_{final} = \sum_{i_a=1}^{8} LL(i_a) * (x,y)_{i_a}$$
With $LL(i_a)$ being the Likelihood computed by our discrete classifier

2. From the 11th step till the end of the trajectory, we choose the most likely sub-model of the 10th step, and use only this one for the end of the trajectory estimation

The final version of our Kalman filter reached a RMSE of $22cm$ in internal cross-validation, and $50cm$ in the external testing. The graphic output is on the front page of the report, and commented in the performance analysis carried out in the annexes.

[**] Technical details can be found in the literature mentioned in the bibliography section.

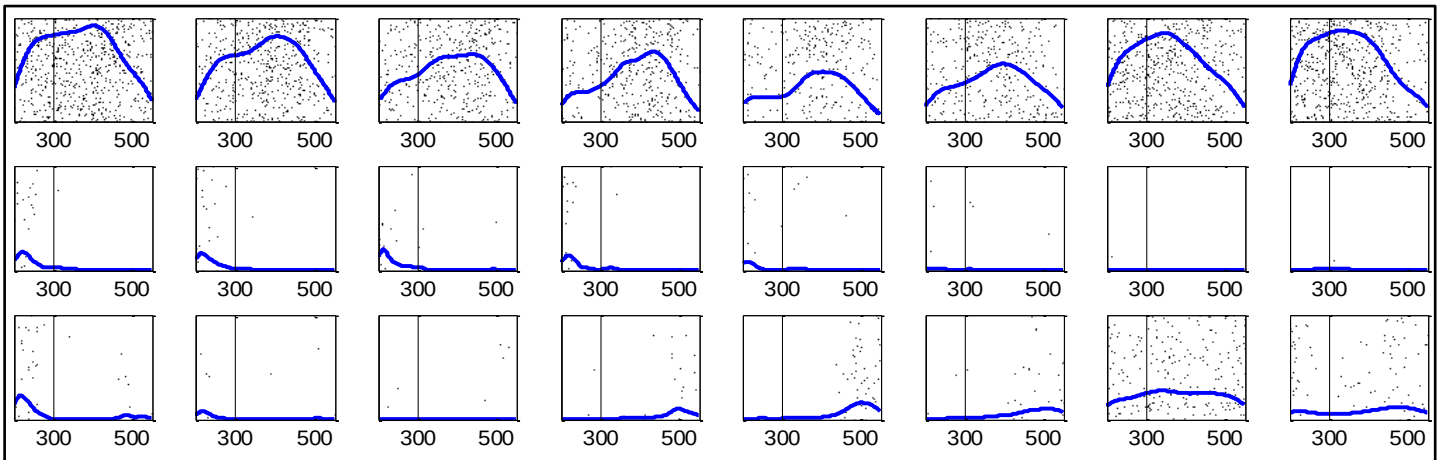


Figure 4. Enhanced Raster plots of neurons 81,44 and 90. The blue line is a smoothed representation of the firing rate, on the top of a Raster plot over the 100 trials. The line denotes the onset of movement. The first neuron is a "good" one, both for the classifying task (difference of activity between the angles during the first half) and the Kalman filter. The second one is a "useless" neuron, and the last one is an example of a neuron that could have stayed in sub-selections for angles 5 to 8 but has been dismissed in our case because of its silence in the other directions.

3.II. Selection of neurons

3.II.a …for the Kalman Filter

The main issue regarding the implementation of the classical Kalman filter was the inversion of the observation model when updating the gain. Indeed, the observation of the spiking activity is highly redundant for two reasons: some of the neurons of the given data are really similar, and may be in reality the same neuron measured by several electrodes (error of clustering during spike sorting), and some neurons are just silent. This redundancy is a real problem to invert the observation model. We simply dismissed by hand any neuron that could be "dangerous" for this inversion for any angle thanks to an enhanced Rasterplot (see Figure 4). Thus, we only kept neurons that had information on every angle, for all the period. We ended up with a set of 40 neurons. This approach has been carried out hastily and is non-optimal; in particular we could have chosen 8 different sets of neurons for each sub-model.

3.II.b …for the Classifier

We used the multi-fit version of our discrete decoder, but the optimal set was different from the first one. Indeed, the data available to compute the likelihoods was on a different timespan: $T_2 = [-100; 200]$ at most, against $T_1 = [-300; 0]$ for the first task. Moreover, we were not limited to a set of 10 neurons anymore. We just did a pre-selection with the Raster plots, then a quick optimisation with the genetic algorithm. Again, our choice may not be optimal at all††.

4. Conclusion

In reality, an impaired patient would want to reach a target which in not among eight predefined ones. This complicates the problem because all the classification part has to be adapted both in the first task and for the trajectory decoder which will lose much of its accuracy without the mixture model. Nevertheless, the success of our approach shows the overall efficiency of neural decoders, all the more as many ways to improve our results have not been implemented – because of shortage of our most precious resource: time.

†† For the anecdote, this choice of neurons put our group into the first position of the competition on the last moment, about 10 minutes before the final deadline.

Annexes

1. Annex 1: more about the population vector decoding

I think it is important to detail a bit more what could have been done regarding this algorithm, because it could be very useful in the relevant case of continuous decoder for the angle inference.

1.I. The main decoding step: Inversion of the projection values

Let $\boldsymbol{P}$ be the vector of the n projections along the n vectors v_i of the n neurons (here n=10). Let $\boldsymbol{U}$ be the projection matrix, defined by the coordinates of the n vectors v_i in the orthonormal base of the 2D plan. Let $\boldsymbol{S}$ be the stimulus virtual vector in this base. The ideal population vector encoding is defined by:

$$\boldsymbol{P} = \boldsymbol{U} * \boldsymbol{S}$$

i.e.:

$$\begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} = \begin{pmatrix} x_{v_1} & y_{v_2} \\ \vdots & \vdots \\ x_{v_n} & y_{v_n} \end{pmatrix} * \begin{pmatrix} x_r \\ y_r \end{pmatrix} = \begin{pmatrix} \cos(\theta_1) & \sin(\theta_1) \\ \vdots & \vdots \\ \cos(\theta_n) & \sin(\theta_n) \end{pmatrix} * \begin{pmatrix} \cos(\theta_s) \\ \sin(\theta_s) \end{pmatrix}$$

If we explicit the matrices and let θ denotes the angles of the considered vectors, which are here taken of norm 1. $\theta_1 .. \theta_n$ are given by the training.

When decoding, the first step is to get the projection values p_i from the model, in the first and simplest case of the cosine fitting, we get $p_i = \frac{r_i - b_i}{a_i}$ from the fitted parameters a_i and b_i. The main step is to *invert* the relation above to get an estimated stimulus vector $\widehat{\boldsymbol{S}}$ from these projection values. If $n > 2$, which is almost always the case, the system is over-constrained. It means that we have "too many" projection values to get one unbiased estimate (provided that the projections are not linearly dependant, i.e. the measures do not perfectly "agree" on the angle). We have chosen the classical Moore-Penrose pseudo inverse of $\boldsymbol{U}$ as a first try:

$$\widehat{\boldsymbol{S}} = \boldsymbol{U}^{\dagger} * \boldsymbol{P}$$

1.II. Enhancing the inversion

Nevertheless, it is not the most meaningful choice, as it minimise the square error:

$$\left|\boldsymbol{P} - \widehat{\boldsymbol{P}}\right|^2 = \left|\boldsymbol{P} - \boldsymbol{U} * \widehat{\boldsymbol{S}}\right|^2$$

It means that the algorithm gives the same "importance" to all the projection measured, in term of error quantification, whereas the error comes from many source (linear noise added to the neuron activity, "bad" projection) and we would like to weight differently some projection than others. For instance, it seems that the neurons whose characteristic vectors are closely aligned with the stimulus vector measure more accurately the projection values; this is not taken into account by the pseudo-inverse inversion. What is more, the preferred directions of the neurons in the 2D plane are not necessarily uniformly distributed, and we know that the stimulus direction, for experimental reasons, is not uniformly distributed as well. The former is not taken into account by the inversion, and could easily been implemented to evenly distribute along all directions the potential error made when estimating the final vector. The latter is prior knowledge that has not been taken into account at all in this algorithm so far.

All these remarks suggest that we could as a first step tweak the inversion of the matrices to counter some asymmetrical effect of the pseudo-inverse inversion, but they suggest also that the best inversion would not be linear, and in particular would depends on the estimated stimulus direction.

1.III. Enhancing the projection

The major hypothesis of the algorithm so far is the cosine shape of the tuning curves. This is absolutely not what has been observed in reality, and some neurons have very good discrimination power without this cosine shape (and therefore are not well used by our algorithm). It would be easy to change the tuning curve fitting and then do a mapping of the value found for the activity to a corresponding projection. The only constraint is that we need to keep a symmetrical shape fitting in order to avoid any ambiguity. This would enable us to fit some curve with a less regular shape than the cosine. Nevertheless, pushing to far this kind of approach will end up developing a continuous generative algorithm based on the tuning curves!

2. Annex 2: Performance analysis

2.I. Comparison of all algorithms

Here is a table that summaries the performance for several of our algorithms for the internal cross-validation performed on 15 different sets of 10 neurons:

Algortihm	Mean performance (%)
Gaussian	93,96
Kernel fit (smoothed estimation of distribution)	92,17
Multi-fit (gaussian+binomial)	94,67
Multivariate (with $\lambda = 0.5$ and $\gamma = 0.18$)	94,13
Population decoding	46,65

Next page is a figure that shows in detail the performance of the best four algorithms for all the 15 example sets. One can see that performance ranking of the algorithms is quite homogeneous. In particular, we have not found any optimal set for the multivariate multi-fit algorithm that would overcome the performance of the optimal set of the basic multi-fit algorithm.

Below are two example of confusion matrices. They have been really useful for the fine tuning of our algorithms and choice of neurons. Unfortunately, they have not been implemented early enough to reveal the crude assymetry in the performance of our population vector algorithm: there is room for improvement!

48	2	0	0	0	0	0	50
55	42	2	0	0	0	0	1
0	29	66	5	0	0	0	0
0	2	39	58	1	0	0	0
0	0	1	58	31	7	3	0
0	0	0	0	55	44	0	1
0	0	0	0	0	62	38	0
2	0	1	0	0	2	35	60

97	1	0	0	0	0	0	2
1	99	0	0	0	0	0	0
0	2	92	6	0	0	0	0
0	0	3	93	4	0	0	0
0	0	0	4	91	5	0	0
0	0	0	0	5	95	0	0
0	0	0	0	0	0	100	0
2	0	0	0	0	0	1	97

Figure 5. Confusion Matrixes for the population vector and the multi-fit algorithms. Values are in percentage, each angle of tested data is a line.

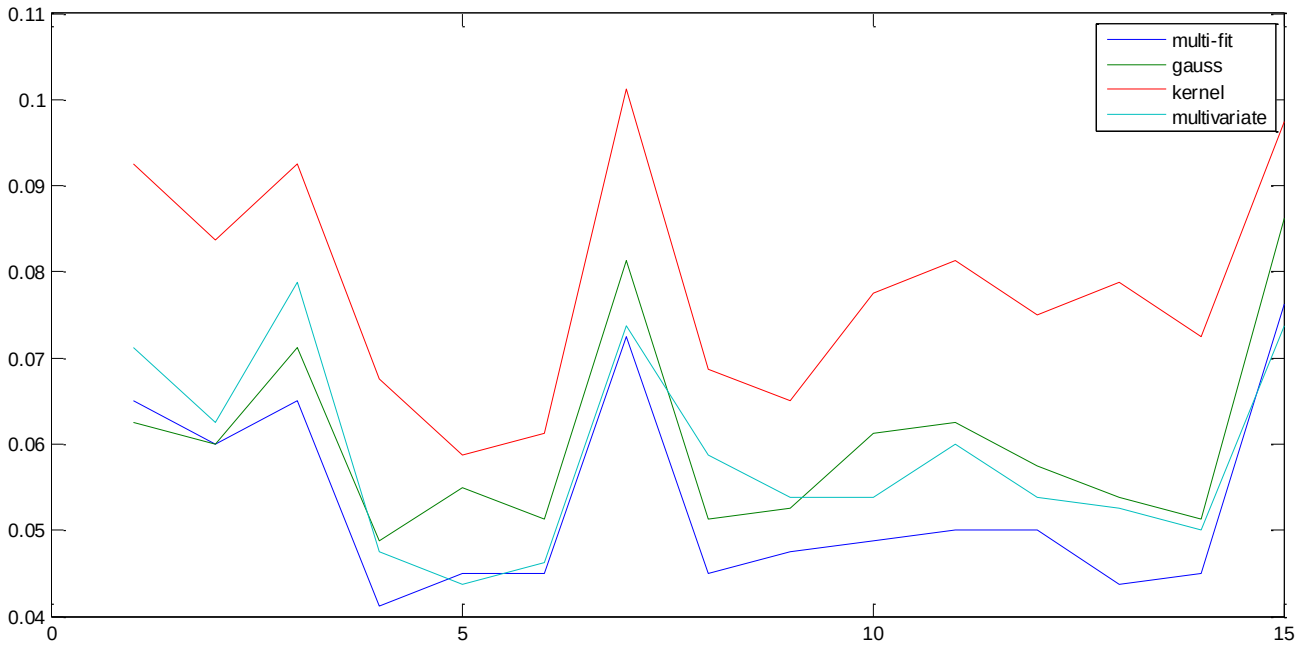


Figure 6. Error for the 4 best algorithms on a set of 15 groups of ten neurons. The multi-fit version is almost always the best. The multivariate algorithm has been tested with $\boldsymbol{\lambda = 0.5}$ and $\boldsymbol{\gamma = 0.18}$.

2.II. Performance of the trajectory decoder

Here is reproduced the figure of the front page. One can see that the estimated trajectories are less variable than the real ones. This comes from the strength of the trajectory model, only "tuned" by the predictions of the observation model. This raise an important question about our decoder: what if there was no clustering of trajectories, and no trajectory means of sub-models to follow? This is more likely the case in reality, where a user could move along intermediate trajectories.

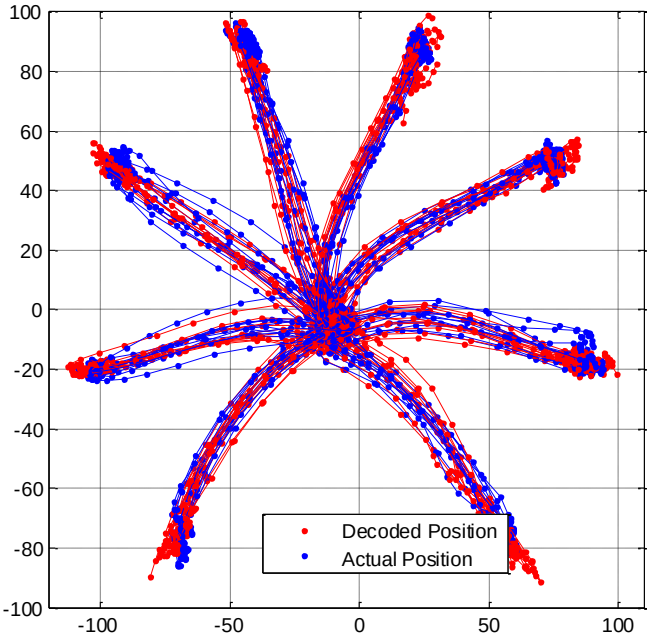


Figure 7. Results of the trajectory decoder. The eight possible types of trajectories constrain the movement, and help the decoder. Performance is really high, with a RMSE of only 13 cm.

2.III. Remark on the marking of algorithms

It occurred that, for all groups performing this exercise, and in particular for all algorithms that we provided, there is a significant difference (about 4 points of percentage) in performance when testing with our cross validation type "leave one out" and the test of algorithm, after training on our 100 given trials and evaluation of performance on the 48 remaining ones. The only explanation to this loss of performance is a non-equivalence between the set of training and inner testing (100 trials) and the set of final testing (the 48 remaining ones). A quick enquiry on the trial IDs corroborates this fact since all the IDs are grouped in a range that do not spans the $8 * 188 = 1456$ minimum span of a randomised set over all the trials. If this is true, performance could have been improved by taking into account this "sorting", and by giving an importance of the trial number.

2.IV. How to quantify the activity?

This is one of the main questions, and it has been eluded since the beginning. We systematically summed up all the spikes during the considered period, but this is not necessary the good approach. Many other representations of the activity could have been used, one of the first idea being a weighting of the spiking activity regarding the time of firing. We tried very briefly to consider such an alternative during task 1, with a shorter period of integration of the spiking activity that would concentrate the information, but it failed to give better results. To enhance the generative classifier, we could "cut" simply the activity along two or more time windows, and consider these several values to be fitted instead of one big sum. On the other hand, the Kalman filter could be enhanced by convoluting the firing rate of each neuron with a dealyed Gaussian, to "smooth" the time binning of the observation model. This could give more information than limiting this count to a $20ms$ bin with a delay.

Bibliography

- In general, for decoding tasks:

C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

- Concerning Population decoding:

F. Gabbiani and S. Cox, *Mathematics for Neuroscientists*. Academic Press, Elsevier Academic Press, 2010.

- For the Kalman Filter:

B. Yu, C. Kemere, G. Santhanam, A. Afshar, S. Ryu, T. Meng, M. Sahani, and K. Shenoy, “Mixture of trajectory models for neural decoding of goal-directed movements” *Journal of Neurophysiology*, vol. 97, no. 5, pp. 3763-3780, 2007.

W. Wu, Y. Gao, E. Bienenstock, J. P. Donoghue, and M. J. Black, “Bayesian population decoding of motor cortical activity using a kalman fillter" *Neural Comput.*, vol. 18, pp. 80-118, Jan. 2006.