Identification of the model of a simulated robot

Overview We want to estimate the system model of Bubblebot, a V-REP simulated robot using ROS. We assign Bubblebot to move on a plane ground, over only one axis. We do not know anything about the physics of Bubllebot so we chose a black box method for the model construction. We chose the ARX and FIR model as model structure and estimate the optimum model parameters with the maximum likelihood method and Matlab's System Identification, and prove it efficiency via validation data. We first build a model assuming there is no noise neither on the robot nor the measurement. We then study several cases of noise inputs. For each estimation, we adopt the following method: process the data points, chose the model structure, estimate the model structure parameters and then run BubbleBot model estimation. We present our graphical and numerical results only for the ARX model since FIR is a particular case or the ARX model. Still, we find get to the same conclusion as for the ARX method.

Environment In the simulation, the robot named BubbleBot is a non-linear MIMO system. It is controlled in velocity through the joystick: the system compares the actuel velocities to the desired velocities and returns system inputs that make the actual velocities more like the desired ones.

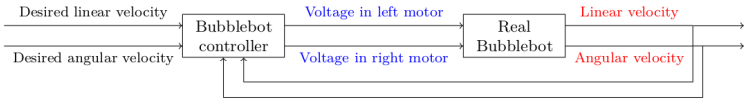


Figure 1: Robot MIMO model

The controlled system {Bubblebot + controler} has therefore the following inputs and outputs :

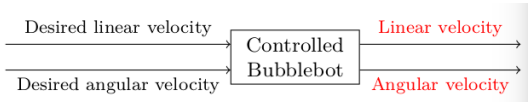


Figure 2: Controlled system MIMO model

We choose to identify the linear velocity channel which is SISO and linear contrary to the previous model. it has the following input and output:

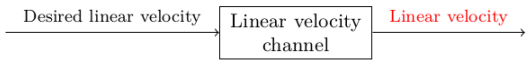


Figure 3: Controlled system SISO model

Data collection protocol We feed Bubblebot various forward and backward linear velocity commands and record both this input signal and the corresponding output for further analysis. To publish desired linear velocities to Bubblebot, we will use a joystick and the teleoperation package you made

for Bubblebot available in the git repository: **learning_joy**. To record the input and output signals of the linear velocity channel, we will use ROS data recording capability.

The desired and actual velocities are provided by the simulator under the topics /vrep/twistCommand and /vrep/twistStatus, respectively. Twists are generalized velocities: three linear and three angular components, in this order.

We allow Bubblebot to move only backward or forward to avoid polluting the data with accidental angular joystick movements. We set the publication rate of input and at the output to be the same to avoid resampling work. We use ROS data recording capabilities to wollect two binary data sets: one for identification one for validation. We convert the data sets to csv formats to analyze them using Matlab.

We collect two other noise-augmented data sets using the same protocol :

- Process noise B_{pn}: We simulate disturbances affecting the robot by adding a pseudorandom uniform noise in $[0, 1]$ to its motors.
- Measurement noise B_{mn}: We add centered uniform noise to the sensor's output.

The beginner's mistake When we naively try to build a model on the collected data, we observe that when we maximize the model complexity we get a good estimation of the estimation data but the validation test is always a failure. We experiment with a set of 700 inputs and 700 outputs and plot the monotony of the fit with respect to the number of parameters (Na,Nb) for the ARX model and Nb for the FIR model. A naive would be to chose the model complexity that maximizes the fit.

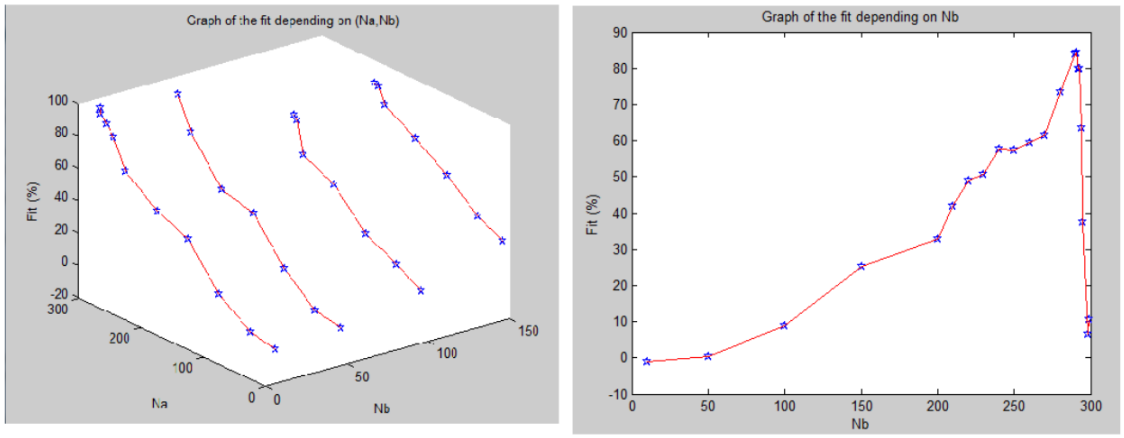


Figure 4: Model fitting (%) with respect to the number of parameters for ARX model (left) and FIR model (right)

ARX model We observe that given a model complexity, the fit is lower when Nb is lower. So we would chose the couple with the lowest Nb and the biggest fit : (Na,Nb) = (290,40). FIR model : we want to localize the model complexity from which the fist grows less quickly. In our experiment there is no such complexity. The maximum fit is 85.63% for Nb = 291 and the Fit = 81.83% for nb = 290 so we chose Nb = 290. We thought we found the best possible with a fit of 93,56% (81.83%) on estimation data value until validating our model and getting a fit of only 52,27% (-30,86%) for the validation data for ARX (FIR). This high gap is not only due to the high complexity of our model which is vulnerable to overfitting with data changes but also to the fact that our data were not synchronous. A given input data point was supposed to be mapped to an output data point at a different time so even if we had a good fit, the estimation could only be irrelevant for a given time.

This error showed us the importance of processing the data we want to model and that this task is at leat as important as choosing and estimating the best model. We process the data sets to get a sequence of synchronized inputs and outputs:

$$u(0), u(1), u(2), ..., u(N)$$
$$y(0), y(1), y(2), ..., y(N)$$

These two sequences have the same number of elements and are sampled at identical rates and corresponding times (times closer from each other than half the sampling interval).

To do so, we clean the csv files from any non numerical values and import the file in Matlab using `csvread`. We bring the timestamps from nanoseconds to seconds and shift them by the same amount so that the input start at time zero. Ig the input and output vector are not the same length (for example, if we end the sample just after recording an input) we crop the longest vector to the dimension of the shortest. We remove outliers using the Matlab's `find` function: it locates the outlier in the output signal and we replace them by the mean of the nearest neighbors.

Model choice We choose to study the ARX and FIR models and use the Matlab's GUI identification system tool to determine the complexity model and try to minimize Na+Nb and get an acceptable fit. There is no exact method for choice of the model so the GUI use a try-and-error method. We plot the estimation error with respect to the nupmber of parameters $N_a + N_b$ i.e. the part of our system that the model does not predict.

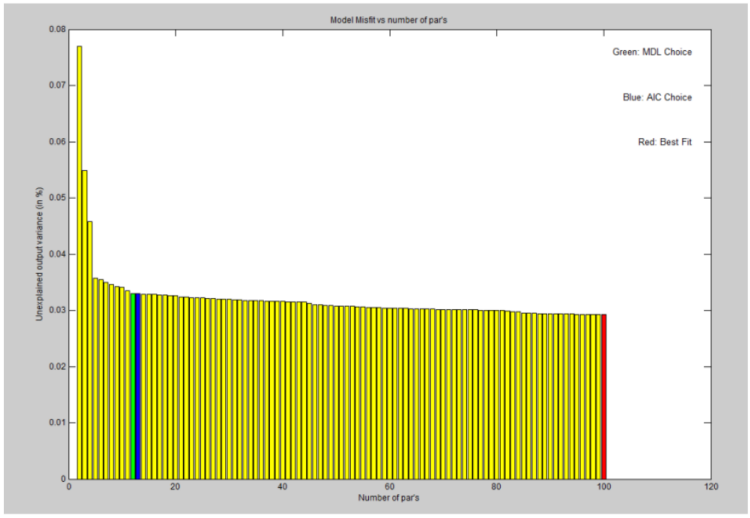


Figure 5: Estimation error (%) with respect to the number of parameters

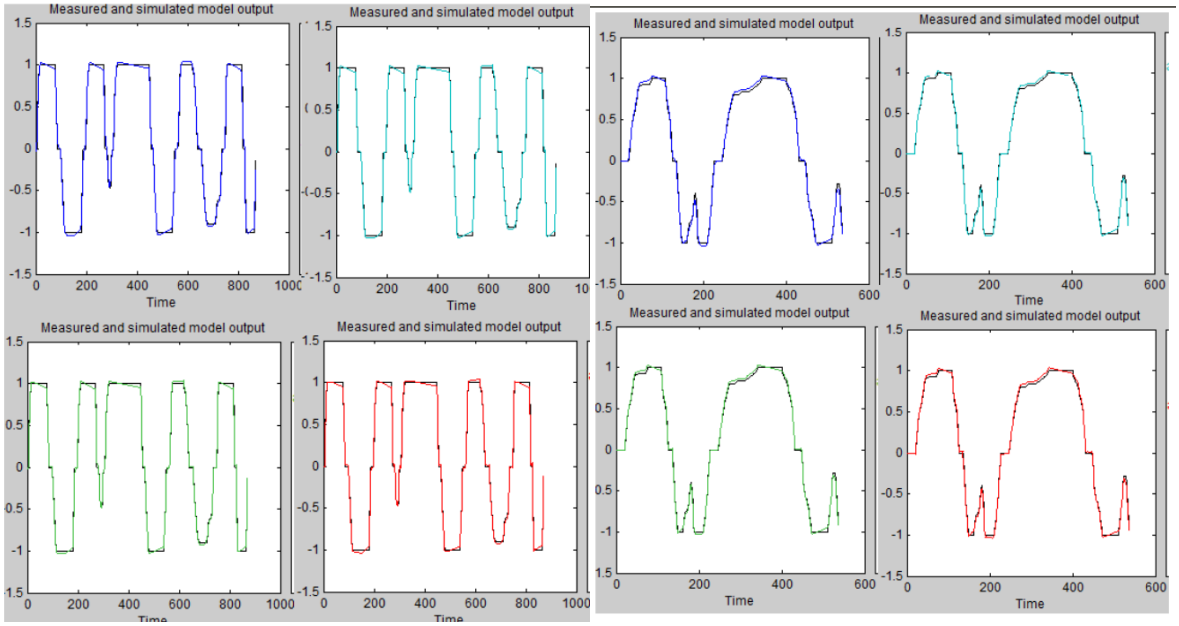
As expected, the error decreases with the number of parameter and we also observe that the speed of the decrement ecreases. For a model complexity $Na + Nb = 200$ we get a misfit of 0.062% but since the decrement is low, we can decrease the model complexity without increasing a lot the misfit. That is why we will choose to have a low complexity model even if it means a higher misfit. The first acceptable value of misfit is 0.052876 % for $N_a = 1$ and $N_b = 1$ with a fit of 96.39%. Conclusion: We are aware that the choice of Na and Nb is easy only because our model is easy. In a more complex case, we would have to define what is an acceptable fit for our model and how much fit we can sacrifice for decreasing model complexity.

The model seems simple even though we already have a high fit. Still we want to observe if increasing the model complexity is relevant or not. We compute the ARX model estimation for the following parameters 2.

Table 1: Training error with respect to the number of parameters

Color	N_a	N_b	Training data fit(%)	Validation data fit (%)
Black (measured output)	X	X	X	X
Blue	1	1	95.32	95.79
Turquoise	1	2	96.53	96.72
Green	1	4	96.74	96.68
Red	5	7	96.88	96.46

We plot the recorded output data and the estimated output for each model and both training and validation data sets.



(a) Training data set

(b) Validation data set

Figure 6: Real output and estimated output plot

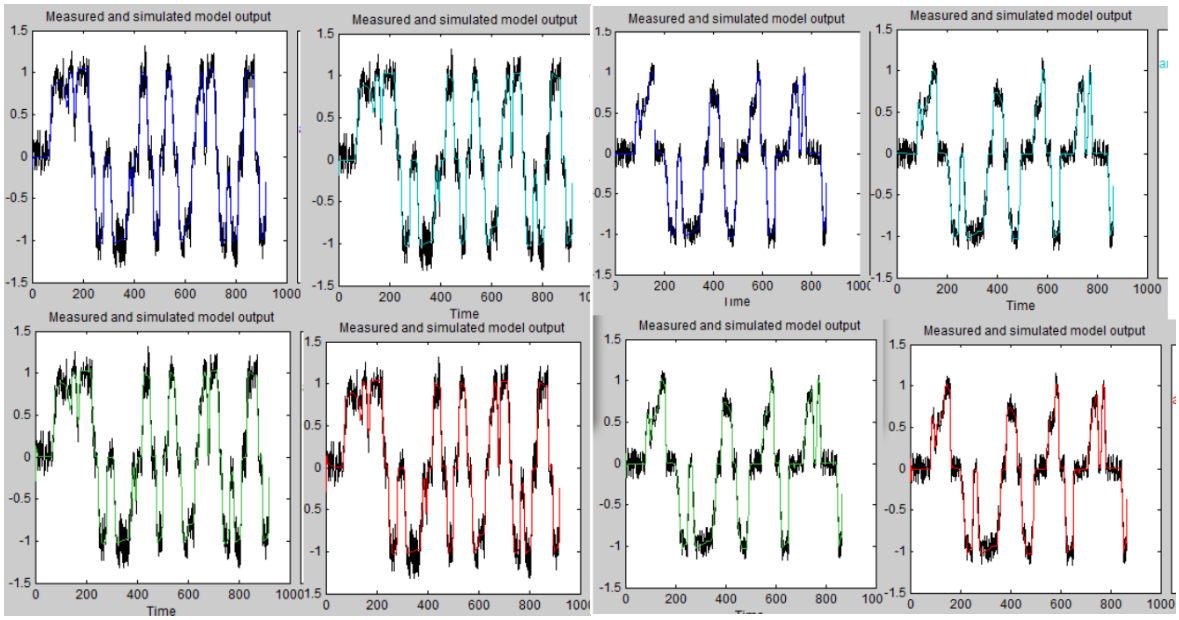
Noisy data sets The test on our model is conclusive : the model estimator fits the data with a fit rate of 95.79% and presents a difference of only 0.87% with most complex model. It is even less that the difference on estimation model. The validation test was positive but running only one validation test on one validation set of data is not trustful enough. We should run it on several set and average the obtained fit to get a real conclusion. Due to a lack of time, we did not go through this method. Still we run validation tests on noised data set and still get a positive test. We show our results in the table **??**. The noise on the measurements is a random uniform noise with a 0 mean and variance of 1. The noise on the robot is a random uniform noise with a 0 mean and variance of 0.2. Once again, the simplest model does not show disadvantages compared to higher models since the fit gap is equal to 0.54%.

We plot the recorded output data and the estimated output for each model and both training and

Table 2: Validation error with respect to the number of parameters

Color	N_a	N_b	B_{pn} data fit(%)	B_{mn} data fit (%)
Black (measured output)	X	X	X	X
Blue	1	1	82.93	80.16
Turquoise	1	2	83.54	80.87
Green	1	4	83.48	80.77
Red	5	7	83.38	80.7

validation data sets.



(a) Training data set (b) Validation data set

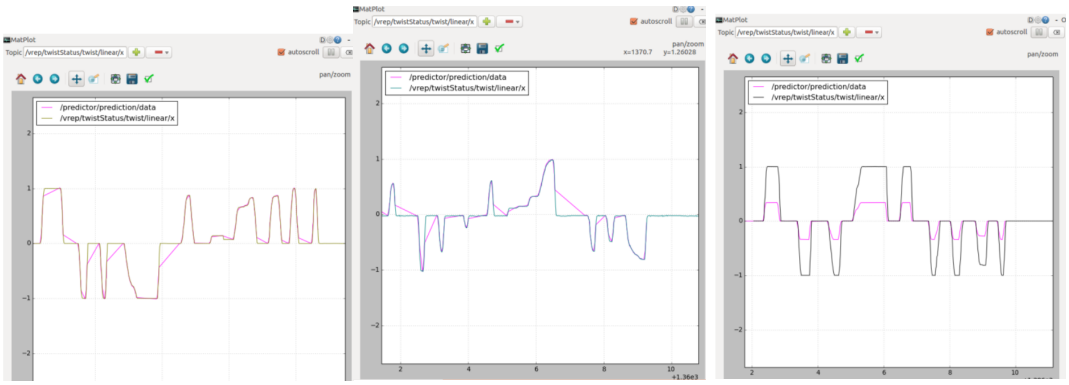
Figure 7: Real output and estimated output plot

Conclusion All along the modelization, we have to make choices on the accuracy of the model : model complexity, acceptable fit for estimation, acceptable fit for validation. In this particular case, we don't have any landmark to decide is a 95% fit is good or not. On more practical cases, these model parameters should be defined depending on the future use of the model and before the beginning of the experiment.

Disturbance detection We use the model we identified to realize disturbance detection on the Bubblebot: we use the knowledge of the model to compare the actual output and the simulated one.

- If they are reasonably the same, we can deduce that the model is well adapted to the current behavior of the Bubblebot.
- If they are suddenly substantially different, it means that the model has become ill-adapted to the system: there might have been a disturbance, or the system might have changed.

We induce changes in Bubblebot by substantially changing its mass mid-simulation. We plot the estimated and measured outputs and then modify Bubblebot's mass.



(a) weight = 10 (b) weigth = 100 (c) weight = 0.001

Figure 8: Real and estimated output plot

For a weight of 10, which is the weight when we created our model, the published data is almost always equal to estimated one, which is logic since our model is valid. We will take this plot as a reference to judge of the prediction gaps.

For a higher weight, 100, our model is still valid : we can interpret this result physically. Since the robot has a higher weight, it has a higher inertia and is then less sensitive to the motors noise. There is still the the noise on the measurements but it is not high enough to perturb the prediction.

With a weight of 0.001, the model is no longer valid : as we did before, we can interpret this with a physical model. The robot is sensitive to any light disturbance which makes its trajectory moves since it has less inertia.

Conclusion This observations show us that our model estimation must be revised as the robot parameters change. We can still say that out model is satisfying since even when the system parameters change, the prediction stays close the measured output.