UNIVERSITÀ DEGLI STUDI
DI TRENTO

Laboratory of Embedded Control Systems
2012-2013

LEGO NXT
Unicycle Vehicle

Cavallari Sandro,
Sottovia Paolo,
Trentin Patrick

Abstract

Nowadays, one of the most trendy and intriguing aspects of applied computer science is the widespread diffusion of embedded systems in all sort of increasingly sophisticated devices. Day after day hardware components such as actuators, sensors and micro-controllers become cheaper and cheaper, while the internet provided the means for ease knowledge exchange and development through new form of communities and social interaction. These trends are giving rise to a paradigmatic shift in the customer mindset, which is not only increasingly aware of the benefits of a richer technological living experience, but also demanding for a more pervasive and creative applications that may improve his lifestyle.

It is therefore important to gather at least surface level knowledge of the instruments that allow to start facing this new market. The main goal of this report is exactly this: provide the basics of automated controls and embedded development by solving a simple task like following a wall surface at a certain distance within established satisfying performance parameters. In other words, the aim of this project is to develop a simple unicycle vehicle that follows a straight line.

To reach this goal we will use as hardware components the *LEGO NXT Kit*, provided by the University of Trento, programmed using *LEJOS OSEK* libraries and we will mainly rely

on *ScicosLab* and *Scicos* for all the part of design and digital simulation.

This report is divided into three main sections: the first one is devoted to identifying the Second Order System parameters underlying the dynamic of the *LEGO NXT MOTOR* by means of a sequence of experiments, the second part aims to design and implement a controller for the wheel that is proven to respect certain requirements and the last part will design a controller for the entire vehicle and glue together all components.

Contents

1 Assignment 1:
***LEGO NXT MOTOR* parameter identification 1**

1.1 Reference Model: Second Order System . . . 1
1.2 Experiment: Input Step Function . . . 1
1.2.1 Preparation of a new software platform . . . 2
1.2.2 Measurements . . . 3
1.2.3 Data Loading and Filtering . . . 3
1.2.4 Filtering Reviewed . . . 5
1.2.5 Data analysis and SOS parameters estimation . . . 5
1.2.6 Error Estimation . . . 6
1.2.7 Final Results . . . 7
1.3 Experiment: Sinusoid Input Function . . . 11
1.3.1 Measurements . . . 12
1.3.2 Data Loading and Filtering . . . 12
1.3.3 Frequency Response Identification . . . 13

2 Assignment 2:
***LEGO NXT MOTOR* controller design and implementation 16**

2.1 Review of Motor Identification . . . 16
2.2 Controller Design . . . 18
2.2.1 Output Signal Acceptable Characteristics . . . 18
2.2.2 Root Locus . . . 19
2.3 Analogic Model of the Closed Loop System . . . 21
2.4 Digital Model of the Closed Loop System . . . 24
2.5 *LEGO NXT MOTOR* Implementation . . . 27
2.5.1 *LEGO NXT MOTOR* execution results . . . 28

3 Assignment 3:
Unicycle Vehicle controller design and implementation 33

3.1 Review of *LEGO NXT MOTOR* Controller Design . . . 33
3.1.1 Anti Wind Up . . . 34
3.1.2 Smith Predictor . . . 36
3.2 Unicycle Model . . . 40
3.2.1 Kinematic Model . . . 41
3.2.2 Transfer function of a path following problem . . . 42

3.2.3 Estimation of the real distance 44
3.3 Vehicle Controller Design 45
3.3.1 Root Locus 45
3.3.2 Digital Model of Vehicle System 47
3.3.3 Vehicle Controller Simulation 50
3.3.4 Vehicle Controller Implementation 52
3.4 Performances Tuning and Verification 53
3.4.1 Tuning of Ultrasonic Sensor 53
3.4.2 Turbo Boost 58
3.4.3 Overall Observations 58

4 Conclusions **60**

1 Assignment 1: *LEGO NXT MOTOR* parameter identification

The *LEGO NXT MOTOR* is a closed technology, and as such it's internals are hidden to the user. In the first part of this report we will describe the experiments done in order to reconstruct a proper model for the Motor, which modelled as a Second Order System. The identification of a model for the *LEGO NXT MOTOR* can be done either in the Time Domain, by feeding the motor an input step function, or in the Frequency Domain, by using an input sinusoid function. We will cover both approaches and separately discuss the outcomes of our experimentations.

1.1 Reference Model: Second Order System

In this report we will consider the *LEGO NXT MOTOR* as a black-box system for which the inner structure its unknown, thus our aim will be that of identifying its frequency response.

Generally speaking, the behaviour of a motor can be generalized quite well using a Second Order System [SOS] model with a pole in 0, related to the step input function, and two complex conjugates roots with negative real part as a consequence of the BIBO stability of the system.

Note that the *LEGO NXT MOTOR* may, or may not, contain inner circuits, possibly with a simple feedback mechanism, that transforms the input signal given to the motor with the aim - for example - to force the system to behave much more like a First Order System [FOS].

The transfer function of a Second Order System has the following general formulation:

$$Y(s) = \frac{k}{\frac{s^2}{{\omega_n}^2} + \frac{2s\xi}{\omega_n} + 1} U\left(s\right) \tag{1}$$

where k is the gain, ξ is the damping factor and ω_n stands for the the natural pulsation of the system. The SOS response to an input step function in the time domain can be appreciated in figure 1.

The output is characterized by an initial ramp, ending with the overshoot, followed by a damped oscillation that stabilizes to the so called *steady state value* after a time t_s, known as *settling time*, with a percentile error lower than a chosen factor α.

1.2 Experiment: Input Step Function

In this experiment an input step function is fed to the *LEGO NXT MOTOR* and its response is evaluated in the *time domain*. The goal of the experiment is to estimate the parameters characterizing

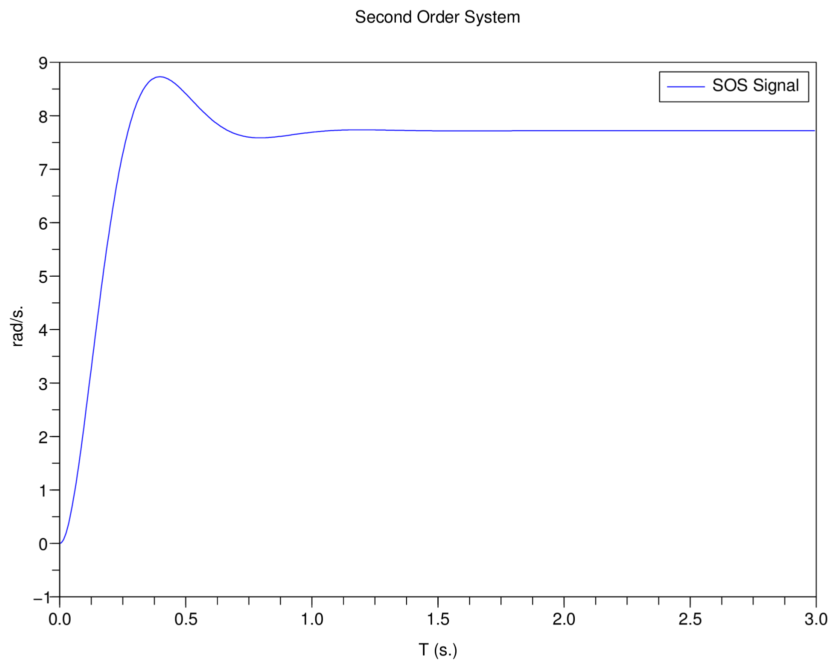

Figure 1: an example of time domain SOS response to an input step function, with $k : 1.7023353, \xi : 0.5570852, \omega_n : 14.139542;$

the *LEGO NXT MOTOR* from the filtered speed response of the system, assuming a Second Order System reference model.

1.2.1 Preparation of a new software platform

Before doing any real experiment we rewrote from scratch the *BRO_spam_client* sources so that:

- The input function (step or sinusoidal) now is hard coded in to the Brick, relieving us from the need of exchanging control packets with the pc;
- The Brick's clock now can be used to have a better estimation of the time at which the samples are taken;
- There is a new definition of the bluetooth packets exchanged, which now encloses five fields: clock_time, space, target_power, target_omega, is_valid_data. The packets are buffered before being sent out;
- Now the time interval of sampling can be safely lowered up to 2 milliseconds. Various tests at 2, 5 and 10 millisecond[ms] have been conducted to ensure that these options worked.

This allows more precise estimation of the response of the system. (As a side note, observe that to not loose bluetooth packets or get corrupted data smaller dimensions of the buffer must be preferred (5, 10) and it does help moving away from an environment that is noisy in the bluetooth or 802.11/b/g spectrum);

- Now the brick can be instructed to coordinate independently its own execution and collect an arbitrary number of experiments with user-defined options without the aid of human interaction;

The *BRO_fist* and *BRO_comm* sources have been modified as well so that the data is printed on *stderr* (which can be easily redirected to a file) instead of being sent to the *Scilab* application.

1.2.2 Measurements

Initially we tried to sample our data with 2 ms of sampling interval, however after some time we realized that the data was so oversampled that we were forced to proportionally redistribute the space covered during an interval of time even when the input step function did not have a so low power value.

Therefore, in the main experiment, we decided to fix our sampling interval to 5 ms and to feed the motor with an input step function with power ranging from 10% up to 100% at intervals of 5%. For each power we produced 50 tests, obtaining enough data for the next step of the identification procedure. All these tests have been made with the engine brake mode on.

1.2.3 Data Loading and Filtering

This phase has the goal to produce a filtered version of the speed response of the system that reassembles our target SOS model.

Once the file containing the data of a test is loaded, the vector of space is transformed into radians. The second step is to proportionally redistribute the covered space, if necessary, when an over-sampling phenomena occurs. This happens mostly when the sampling interval is low in respect to the capacity of the motor to rotate enough so that the Brick can detect this change. In such cases, omitting this step would give the wrong perception that the motor continuously switches from stationary periods to active states with a high instantaneous acceleration and an enormous speed. However we know by construction of the experiment that, excluding the initial adjustment time, the acceleration should be 0 and the velocity constant.

The next step is the computation of the *instantaneous speed* [IS] of the motor (*first derivative*) using the collected data:

$$\omega(t) = \frac{\Delta_\theta}{\Delta_t}$$

The IS is still raw and characterized by errors due to the quantization, sampling and noise effect. Therefore we need to apply a *low-pass filter* to eliminate the high frequencies.

We tried four different filters:

- **Moving Average Filter**, both simple and weighted, with a variable window with size ranging from 10 up to 50 samples;
- **Exponential Filter** with a single *forgetting factor* ranging from 0.05 to 0.4;
- **Double Exponential Filter** with a couple of *forgetting factors*, ranging again in the same discrete set of values;
- **Butterworth Filter**, with an order of 3 and various cut-off frequency values;

The first three operators produced a filtered signal that reassembled more the shape of a smoothed step with a high degree of noise rather than our target SOS model. The **butterworth** filter instead, designed to have a frequency response in the pass-band as flat as possible, revealed itself to produce a good filtered signal for our target SOS model. The only problem of this filter is that it introduces some delay, so that both the ramp and the over-shoot appear shifted in time, as shown in figure 2.

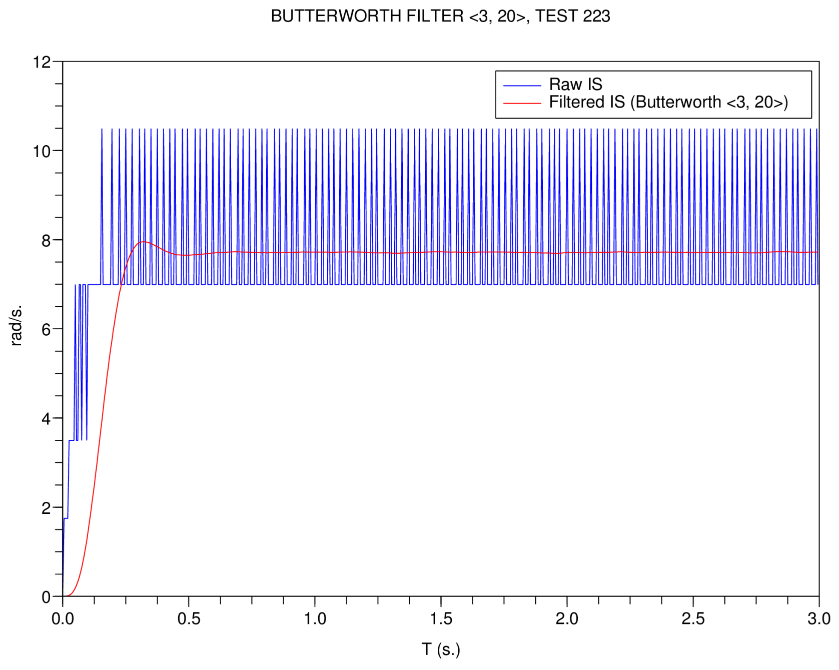

Figure 2: an example instantaneous speed data filtered with a butterworth operator

After some estimations with different cut-off frequencies, we found that a good calibration for this filter is to use an order of 3 with a cut-off frequency of 20. Higher cut-off frequencies would have been preferable, so that the over-shoot position could be anticipated, however this configuration introduces also a high degree of spurious noise on the filtered signal that would complicate the next phase of the SOS model parameters identification.

1.2.4 Filtering Reviewed

During the second assignment we realized that the estimation of SOS model obtained using the **butterworth** filter was rather poor, mainly due to the delay in rise and settling time it introduces, and that this problem would have affected the overall project experience if not immediately solved.

As reported at the beginning of the second assignment, we jumped back to the filtering techniques and tried a new approach based on **cubic splines**, a technique used for interpolation. The advantage of this choice is that now the filtered signal has a slope that is nearly overlapping the experimental data, meaning that it is no longer affected by the delay introduced with butterworth. This increasingly improved our estimation of ω_n.

As a consequence of this review work, from this point onward all the data, plots and analysis has been consistently corrected to reflect the newly found values.

1.2.5 Data analysis and SOS parameters estimation

As we have previously seen, the transfer function of a Second Order System has the general formulation of equation 1.1:

$$y(s) = \frac{k}{\frac{s^2}{{\omega_n}^2} + \frac{2s\xi}{\omega_n} + 1} u\left(s\right)$$

where k is the gain, ξ is the damping factor and ω_n stands for the natural pulsation of the system. These parameters are unknown, but can be computed from the *filtered signal* with a certain degree of confidence. We briefly summarize here the formulas that must be computed in order to proceed with the system identification:

- *Overshoot*, maximum oscillation in respect to the steady state value:

  $$O = \frac{\left|\omega_{max} - \omega\left(\infty\right)\right|}{\left|\omega\left(0\right) - \omega\left(\infty\right)\right|}$$

  where ω_{max} is the maximum oscillation value;

- $$k = \frac{\omega_\infty}{A}$$

where ω_∞ is the steady state value (q) and A is the amplitude of the input step signal (i.e. the power assigned to the motor). This formula derives from the final value theorem of a Laplace transform.

- *Damping Factor* (ξ):
$$\xi = \sqrt{\frac{\ln^2 O}{\pi^2 + \ln^2 O}}$$
- The *Settling Time* T_s, estimated as the moment from which the filtered signal value does not oscillate more than a percentile α ($= 5\%$) from its steady state q;
- *Natural Pulsation* (ω_n), which has three possible formulations:
$$\omega_n^1 = \frac{(\log\alpha - \log\overline{N})}{(-\xi * T)}$$
where $\overline{N}$ is computed as:
$$\overline{N} = \frac{1}{(1 - \sqrt[2]{1-\xi^2})}$$
$$\omega_n^2 = \frac{-\log\frac{\omega(t_0+T)-\omega_\infty}{\omega(t_0)-\omega_\infty}}{\xi * T}$$
$$\omega_n^3 = \frac{\pi}{T\sqrt[2]{1-\xi^2}}$$
where α ($= 5\%$) is the percentile used as a threshold below which the oscillating signal is considered stable, t_0 is a certain time instant and T is the period of the oscillations of the system. Note that in our experiment we used ω_n^3, since it has shown itself to result in the most numerically stable estimation of the ω_n variable.

1.2.6 Error Estimation

In order to determine with a good approximation the physical parameters of the Second Order System we tried to fit onto the *LEGO NXT MOTOR*, we selected the 20 tests with the best SOS models and computed the average of their ξ, ω_n and k estimations.

Here the notion of "better" SOS model is strictly connected to the following error estimators:

- *Integral Squared Error*:
$$ISE = \int_0^T (\omega_m(t) - \omega(t))^2\,dt$$

- *Integral Absolute Error*:
$$IAE = \int_0^T \|\omega_m(t) - \omega(t)\| \, dt$$
- *Integral Time Squared Error*:
$$ITSE = \int_0^T t(\omega_m(t) - \omega(t))^2 \, dt$$
- *Integral Time Absolute Error*:
$$ITAE = \int_0^T t\|\omega_m(t) - \omega(t)\| \, dt$$

We have decided to use a modified version of the *ITAE* error estimator to compare our models in which we have added to the denominator the area of the filtered signal. This change allowed us to compare tests taken using input step functions with different power.

We then validated our estimates of such parameters verifying that the *ITAE* error of all the tests was sensibly reduced using the newly estimated parameters. The improvement can be appreciated in figure 3, by simply comparing the green line (SOS model with best parameters) to the black line (original SOS model associated to this test data).

1.2.7 Final Results

In table 1 you can see, for each measured power, the estimations of the four main parameters of our model. As a consequence of the option *BRAKE = ON*, the steady state value shown in figure 4 grows linearly with the power. The **gain**, **damping factor** and **natural pulsation** are shown respectively in figures 5, 6 and 7. Note that the "high" variance in the estimation of the natural pulsation at very low power inputs could have been lowered by adjusting the **Cubic Spline** filtering parameters to the different input power levels. This has not been done in the displayed plot to maintain the homogeneity of the estimated values.

From these values we derived our final estimations of the general SOS model characterizing the *LEGO NXT MOTOR* by applying a simple averaging operation:

- $\mathbf{K} = 2.033620$
- $\omega_\mathbf{n} = 20.907025$
- $\xi = 0.730782$

These values will be at the basis for the next assignment devoted to the design of a **Controller** for the *LEGO NXT MOTOR*.

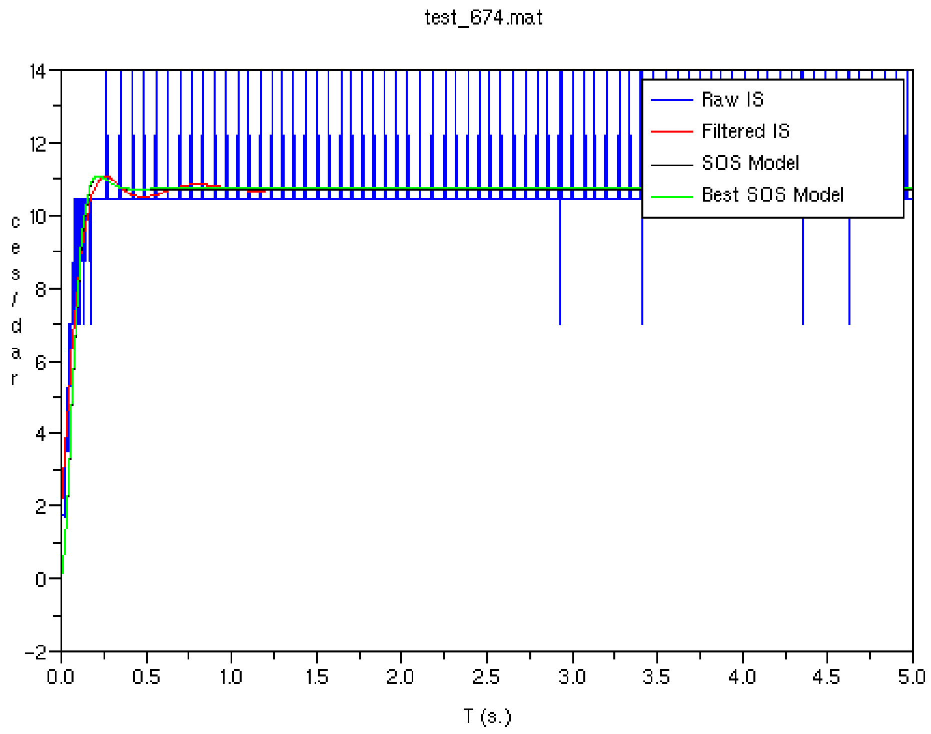

Figure 3: The response of the system: raw instantaneous speed (blue), filtered signal (red), sos model response for current experiment (black), best sos model response (green)

Power	Final Value (q)		Gain (k)		Damping F. (ξ)		Nat. Freq. (ω_n)	
%	avg	σ^2	avg	σ^2	avg	σ^2	avg	σ^2
10	1.34595	5e-05	1.81885	8e-05	0.73356	0.002	19.88264	5
15	2.11017	3e-05	1.90105	2e-05	0.71283	0.0005	19.57561	0.9
20	2.99757	3e-05	2.02538	1e-05	0.71225	0.0002	19.56728	0.3
25	3.76146	5e-05	2.03322	1e-05	0.70605	0.0001	19.27464	0.2
30	4.53271	4e-05	2.04176	8e-06	0.70764	0.0001	19.47343	0.3
35	5.30502	3e-05	2.04827	4e-06	0.70973	7e-05	19.64253	0.3
40	6.19232	0.0009	2.09200	0.0001	0.71359	5e-05	19.84787	0.1
45	6.88301	0.0002	2.06697	2e-05	0.71315	5e-05	19.89709	0.2
50	7.61985	0.0001	2.05942	7e-06	0.72138	4e-05	20.27311	0.1
55	8.36517	4e-05	2.05533	3e-06	0.72404	3e-05	20.50765	0.1

Table 1: Estimated parameters of the SOS characterizing the *LEGO NXT MOTOR* engine.

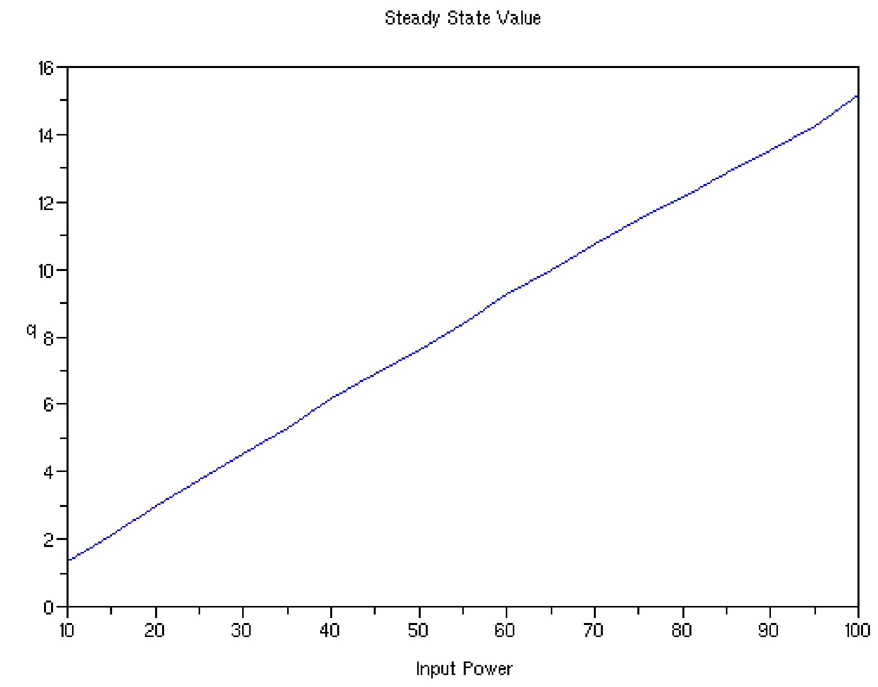

Figure 4: Evolution of steady state value at increasing power

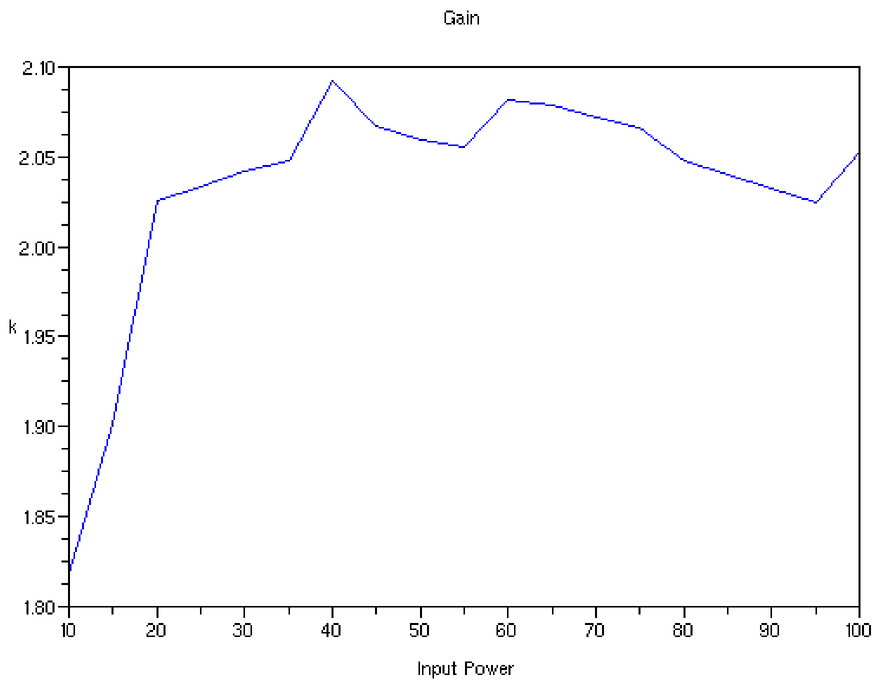

Figure 5: Evolution of the Gain value at increasing power

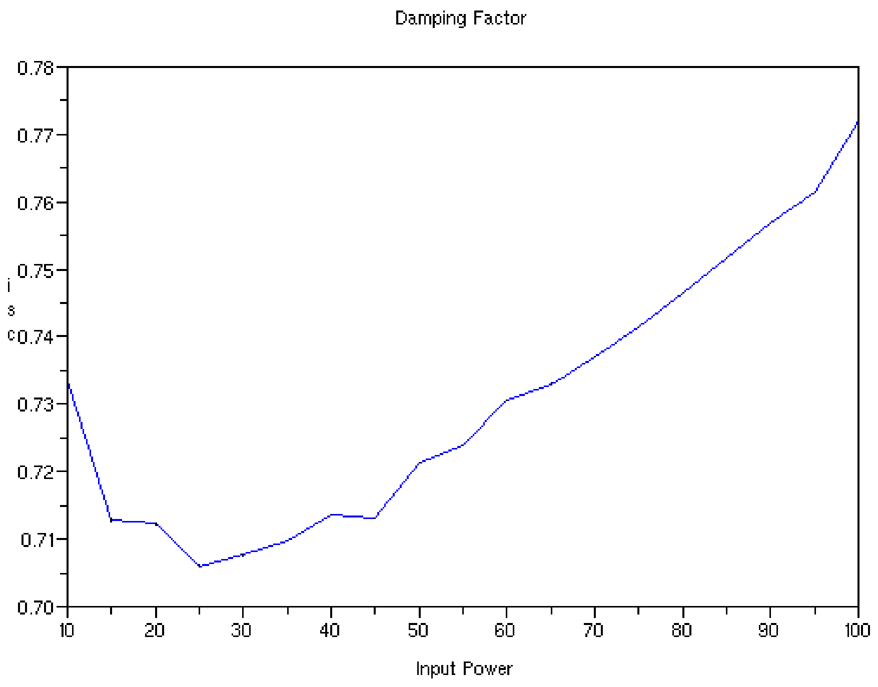

Figure 6: Evolution of damping factor value at increasing power

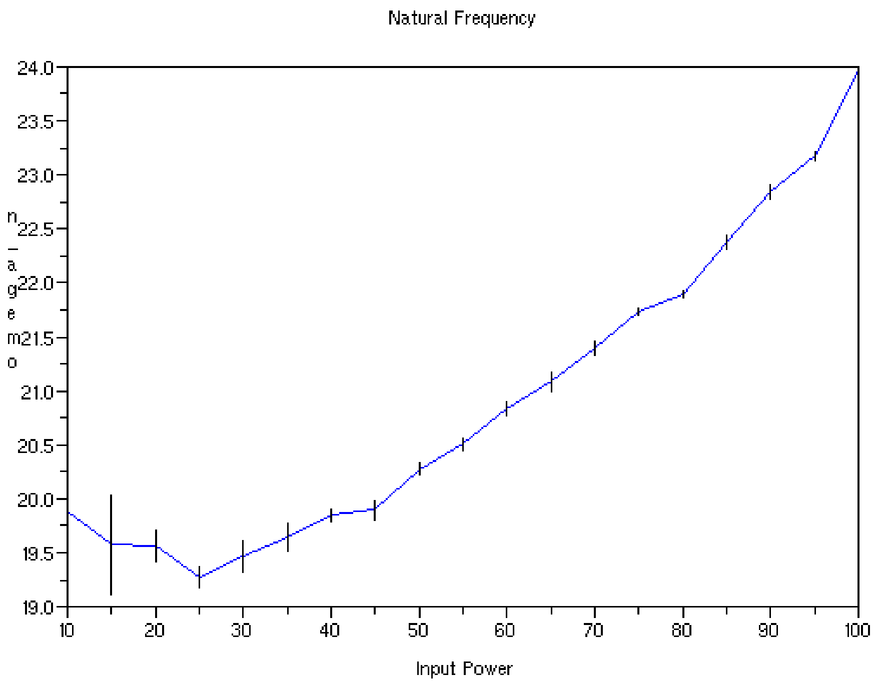

Figure 7: Evolution of natural frequency value at increasing power

We also tried to plot the frequency response of our SOS model with the aid of *scilab* functions, feeding as input to equation

$$G(s) = \frac{k}{\frac{s^2}{{\omega_n}^2} + \frac{2s\xi}{\omega_n} + 1}$$

the parameters identified, for example, with an input step function with power 50%, as shown in figure 8. The phase correctly decays of $180\,\deg$ since we have a couple of complex conjugate poles in our model, while the slope of the Magnitude appears to be of nearly $-40dB$ from 1 to 10 Hz.

$$p1 = -15.278478 + 14.271364i$$

$$p2 = -15.278478 - 14.271364i$$

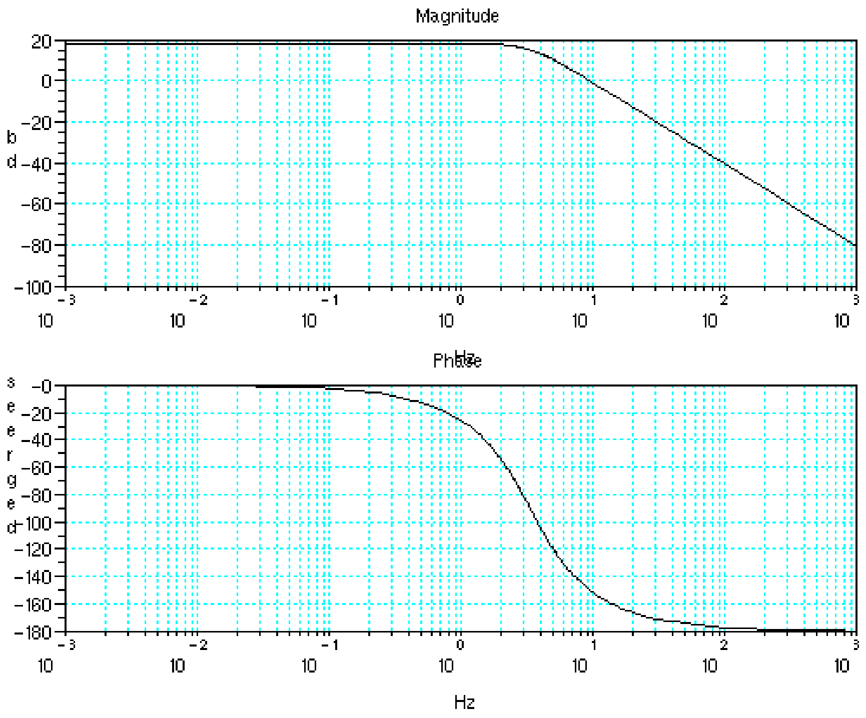

Figure 8: The magnitude and phase of the frequency response given by the parameters identified in the time domain for the input step function with 50% of power.

1.3 Experiment: Sinusoid Input Function

In this experiment we decided to use as input function for our *LEGO NXT MOTOR* a sinusoid signal with different frequency of oscillation and fixed amplitude. The goal of this experiment is determine the frequency response of the *LEGO NXT MOTOR* engine using a frequency approach.

1.3.1 Measurements

For this experiment we modified the code previously written so that it could now support a sinusoidal input signal with arbitrarily chosen amplitude.

In this experiment the data is sampled at intervals of $0.005s(t_{sampling})$, while each test lasts $60s(t_e)$. Therefore by the *Theorem of Nyquist-Shannon* we know that the maximum frequency we can properly sample and reconstruct is given by:

$$f_{max} = \frac{1}{t_{sampling} \cdot 2} = 100\ Hz$$

Since we experienced a certain degree of numerical instability in our estimation of *magnitude* and *phase*, in the actual case of our measurements we limited our frequency well below this threshold.

We fixed the sinusoid signal amplitude(A) to 50% of the motor supply power, while we made the frequency vary among several values computed using the following equation:

$$f = \frac{(test_n)^{1.5}}{t_e}$$

with $f < f_{max}$.

1.3.2 Data Loading and Filtering

Similarly to what we have done in the previous experiment with the time domain approach, we load from a file the sampled data of the system response plus the samples of the input signal fed to the system. Again, the space samples are transformed into radians and proportionally redistributed, when necessary. From these data we derive the *Instantaneous Speed*, over which we apply one of our *low pass* filtering techniques to eliminate the high frequencies typical of spurious errors and white noise.

The filters were configured as in the following description:

- **Moving Average Filter**, only simple (homogeneously weighted), with a variable window size equal to
$$max\Big(\frac{(N_{samples} \cdot T_{IS})}{(t_e \cdot 10)}, 3\Big)$$
- **Exponential Filter** with a single *forgetting factor* ranging from 0.05 to 0.4;
- **Double Exponential Filter** with a couple of *forgetting factors*, ranging again in the same discrete set of values;

- **Butterworth Filter**, with an order of 3 and various cut-off frequency values;

In this case the filter that had shown to behave better was *Moving Average Filter* since it does not introduce artificial delay in the *phase*, like the *Butterworth Filter*, nor smooth too much the *magnitude* thanks to the self adjusting window size.

In figure 9 you can appreciate the result of this filtering operation by looking at the line with red color.

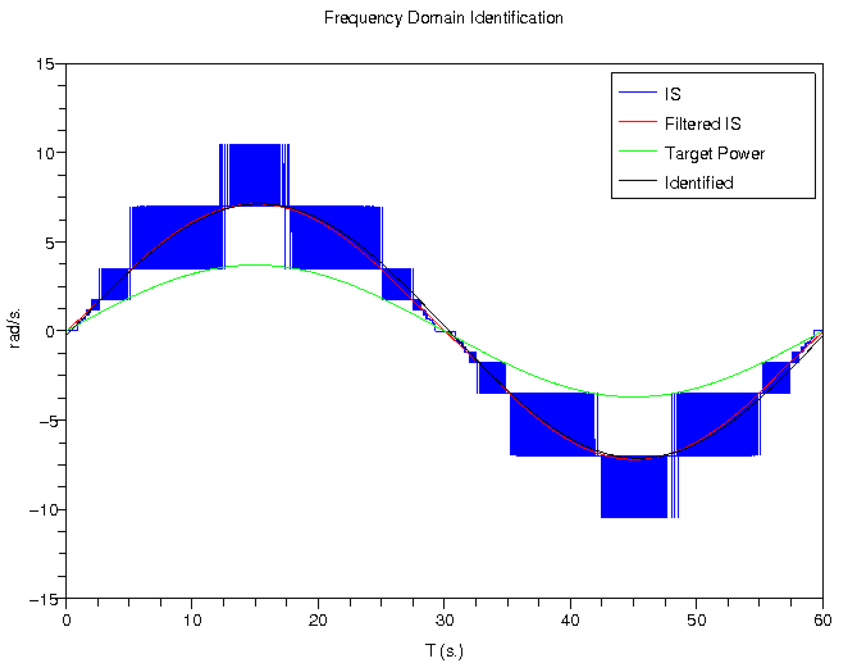

Figure 9: The response of the system: raw instantaneous speed (blue), filtered signal (red), reconstructed response signal (black) and best SOS model (green).

1.3.3 Frequency Response Identification

The identification in the frequency domains proceeds as follows: we feed to our asymptotically linear system $G(j\omega)$ a sinusoidal function with the general formulation

$$u(t) = A \cdot sin(\omega t)$$

and obtain an harmonic response of equation

$$y(t) = \|G(j\omega)\| A \cdot sin(\omega t + \angle G(j\omega))$$

The *Gain* is given by the ration between the peaks of the input and those of the outputs:

$$\|G(j\omega)\| = \frac{y_{max}}{u_{max}}$$

The *Phase* is estimated using the peak times t_y and t_y of the output and input signals respectively:

$$\omega t_y + \angle G(j\omega) = \frac{\pi}{2} + k_y 2\pi$$
$$\omega t_u + \angle G(j\omega) = \frac{\pi}{2} + k_u 2\pi$$

with k_y,$k_u \in \mathbb{Z}$.

Hence:

$$\angle G(j\omega) = \omega(t_u - t_y) + (k_y - k_u)2\pi, \ \angle G(j\omega) \in [-2\pi, 0]$$

By collecting samples of data at different input frequencies it's possible to reconstruct the frequency response shape, since it is uniquely characterized by its *magnitude* and *phase* values shown in the *Bode Diagram* 10. From the *phase* plot it's possible to see that the *LEGO NXT MOTOR* engine behaves like a Second Order System because we reach $-180\,\mathrm{deg}$. Like in the time domain identification, we observe that the slope in the *magnitude* plot has a decay of only $-10\,dB$.

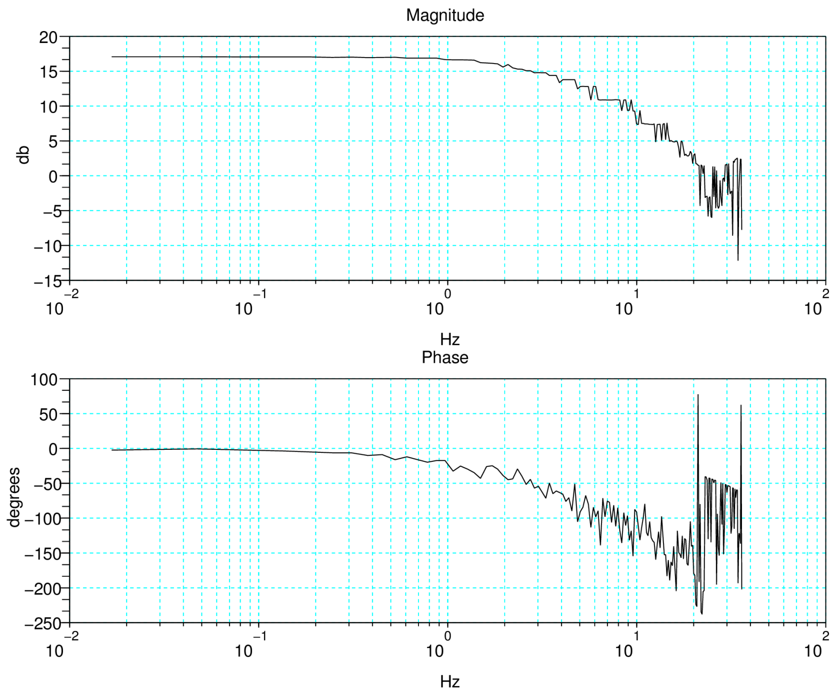

Figure 10: The magnitude and phase of the frequency response given in input a sinusoidal signal with amplitude power 50%.

In the *magnitude* plot we can't see the peak in correspondence to the resonance frequency as we would expect with a Second Order System. This may have been due to the fact the frequencies at which we sample have been selected using a function growing exponentially (with factor 1.5), so we may have missed to properly sample in the neighbourhood of such frequency. Therefore we have repeated the measurements in the range from $1Hz$ to $40Hz$, taking measurements for over 600 equidistant frequencies. This experiment confirmed the behaviour shown in figure 10.

2 Assignment 2: *LEGO NXT MOTOR* controller design and implementation

The second third of this report is devoted to the design of a Controller for the *LEGO NXT MOTOR* that respects a set of constraints defining the desired performances of the motor. After an initial review of the results obtained with the previous experimentation with the *LEGO NXT MOTOR*, we will use the *Root Locus* technique, combined with *Scicos* analogical and digital simulations, to lead the controller design phase up to acceptable performances. The controller will then be implemented on the *LEGO brick* and its performances verified against the initial constraints.

2.1 Review of Motor Identification

Since the very beginning of the Controller Design phase we realized the necessity of reviewing the results of the **First Report** dedicated to the Identification of Second Order System (SOS) parameters modelling the *LEGO NXT MOTOR*. This review was specifically aimed to strengthen our confidence toward the SOS parameters reliability as a starting point for the future work, therefore we conducted further experimentation with new signal filtering approaches that could approximate better approximate the key features of the raw data obtained from the *LEGO NXT MOTOR*.

In particular we decided to abandon the **butterworth** filtering technique which introduced heavy delay effects in between the experimental data and the filtered signal, thus leading to a wrong estimation of ω_n. In replacement of butterworth we adopted instead an approach based on **cubic splines**, a technique used for interpolation. In figure 11 it is possible to appreciate how the cubic splines allowed us to obtain a filtered signal with a slope nearly overlapping the experimental data, meaning that we are no longer affected by the delay introduced by butterworth and that now our estimate of ω_n can be more accurate.

Following the same methodology explained in the **First Report**, we estimated the new SOS parameters of the *LEGO NXT MOTOR* to be:

- $\mathbf{K} = 2.033620$
- $\omega_\mathbf{n} = 20.907025$
- $\xi = 0.730782$

We should mention that the **First Report** has been updated accordingly, so that it now displays the newly computed data and graphs.

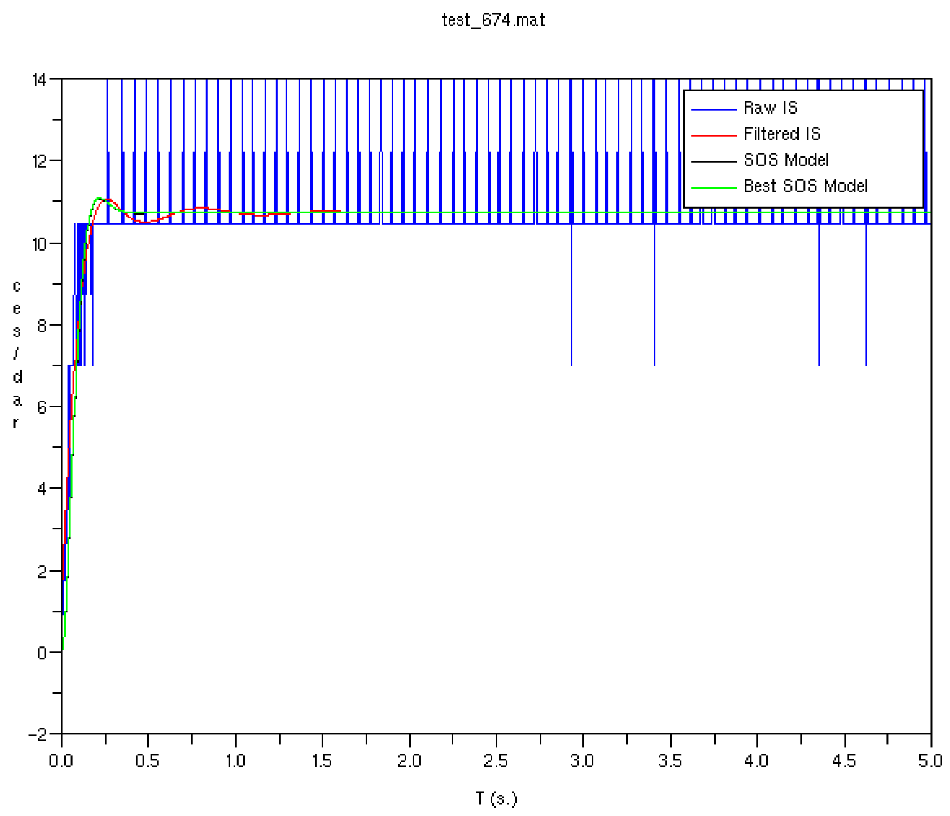

Figure 11: The response of the system: raw instantaneous speed (blue), filtered signal (red), reconstructed response signal (black) and best SOS model (green).

2.2 Controller Design

In this section we will cover the design of a *Controller* for the *LEGO NXT MOTOR* system with a *Closed Loop System* (CLS), a design architecture in which the output of the system is used as feedback reference for the controller itself.

2.2.1 Output Signal Acceptable Characteristics

The first step that has to be taken is to define what are the performance constraints that the output signal produced by the *Closed Loop System* must meet in order to be acceptable for our purposes. Generally speaking we aim to design the system so that it behaves well in the case of input step signals, which means having zero steady state error as well as an early settling time. Therefore the acceptance parameters have been set to:

- **Max Overshoot (**O_{max}**)**: 30%
- **Settling Time (**St_{max}**)**: $300ms$
- **Rise Time (**R_t**)**: $80ms$
- **Overshoot Time (**O_t**)**: $100ms$
- **Input Power (**IP**)**: 7.4 [V]
- **Steady State Value Tolerance (**α**)**: 5%

Recall from the **First Report** that the *Open Loop System* did not satisfy at all these requirements, in particular it had an average overshoot time of $200ms$ and settling time of $400ms$.

Using these parameters as reference one can easily derive the coefficients, namely ξ and ω_n, of the corresponding desired output signal using the following formulas:

$$\xi = \sqrt{\frac{\log(O_{max})^2}{\log(O_{max})^2 + \pi^2}} \tag{2}$$

$$\omega_n^1 = \frac{\log(0.05) - \log(\overline{N})}{-\xi \cdot St_{max}} \tag{3}$$

where $\overline{N}$ is computed as:$\overline{N} = \frac{1}{\sqrt[2]{1-\xi^2}}$

$$\omega_n^2 = \frac{-\log(\frac{(\omega_{max}-q)}{(\omega_{min}-q)})}{\xi \cdot O_t} \tag{4}$$

where $q = k \cdot IP$ and $\omega_{max} = (O_{max} \cdot q) + q$ and $\omega_{min} = 0$

$$\omega_n^3 = \frac{2 \cdot \pi}{2 \cdot O_t \cdot \sqrt{1 - \xi^2}} \tag{5}$$

$$\omega_n^4 = \frac{-log(\alpha)}{(St_{max} \cdot \xi)} \tag{6}$$

Note that of the several formulas available for ω_n we decided to adopt the third one, since by plotting the resulting signals we observed it to be the less error prone estimate available.

As a result of these computations we obtain $\omega_n = 33.64$ and $\xi = 0.36$. The corresponding SOS has been plotted and it proved itself to be within our acceptability constraints, since it has a $St_{max} = 230ms$ and $Ot_{max} = 100ms$.

Now it is possible to use the couple (θ, ω_n), where $\theta = asin(\xi) + \frac{pi}{2}$, on the complex plane to graphically show the region of all parameters that will satisfy our constraints over the signal. Given that by increasing values of ξ both the overshoot and Rt decrease while St_{max} increases, and that by increasing values of ω_n all the timing parameters (Rt, Ot and St_{max}) decrease, it follows that the acceptance region is the pink area shown in figure 12.

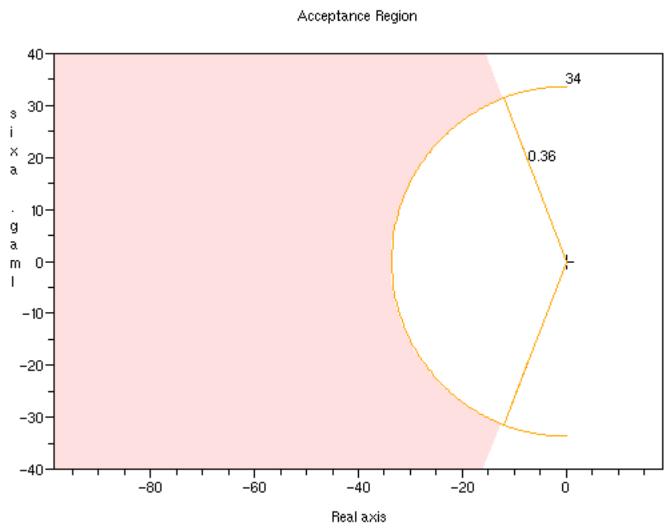

Figure 12: The pink area is acceptance region of the SOS system parameters satisfying the imposed constraints

2.2.2 Root Locus

The output of the system, considering that for a SISO system $P(s) \cdot C(s) = C(s) \cdot P(s)$, can be computed as

$$\begin{aligned}Y(s) =& P(s) \cdot U(s) \\ =& P(s) \cdot C(s) \cdot E(s) \\ =& P(s) \cdot C(s) \cdot (R(s) - Y(s))\end{aligned} \tag{7}$$

hence the *Closed Loop System* transfer function is

$$Tlf(s) = \frac{C(s) \cdot P(s)}{1 + (C(s) \cdot P(s))}$$

where the plant, instantiated with the previously seen parameters, is given by

$$P(s) = \frac{k}{\frac{s^2}{{\omega_n}^2} + \frac{2s\xi}{\omega_n} + 1} U\left(s\right)$$

We will determine the transfer function of the Controller so that, when the input signal is a step function, $Y(s)$ meets the acceptance constraints defined in the previous section and, as a consequence, it is stable. This can be done using the **Root Locus** analysis, a graphical technique that allows to understand how the behaviour of the *Closed Loop System* changes by placing the roots of the *Controller* in different positions of the complex plane.

The first thing to observe in the design of the Controller is that to obtain a zero steady state error with an input step function it is necessary to put a pole in the origin $(0+0i)$. More over, in order to attract the Plant poles into the acceptance region we eventually figured out that two complex and conjugated zeros placed at $(-22+15i)$ and $(-22-15i)$ reach the greatest effectiveness. Finally, to maintain the *causality* of the system and the asymptote more on the left as possible, we put another pole in $(-80+0i)$.

As it is possible to see in the figure 13, with the gain of the controller (kc) set to 3.5, we obtain a stable system that respects our acceptance constraints.

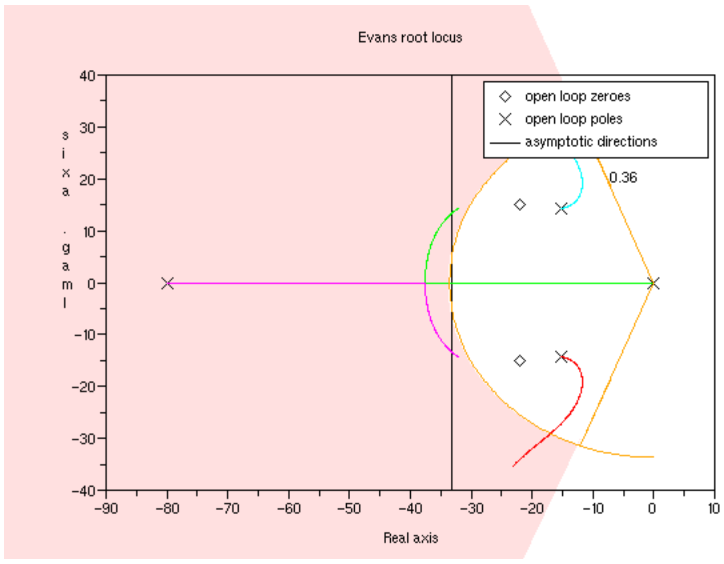

Figure 13: Root Locus of the Cloosed Loop System using the designed Controller

The resulting formulas of the Controller is :

$$C(s) = 3.5 \cdot \frac{(s^2 + 44s + 709)}{(s^2 + 80s)}$$

2.3 Analogic Model of the Closed Loop System

Once the design of the **Controller** has been established, we need to model our entire system on **Scicos** to check whether the *Closed Loop System*'s performances respect the desired one's as expected.

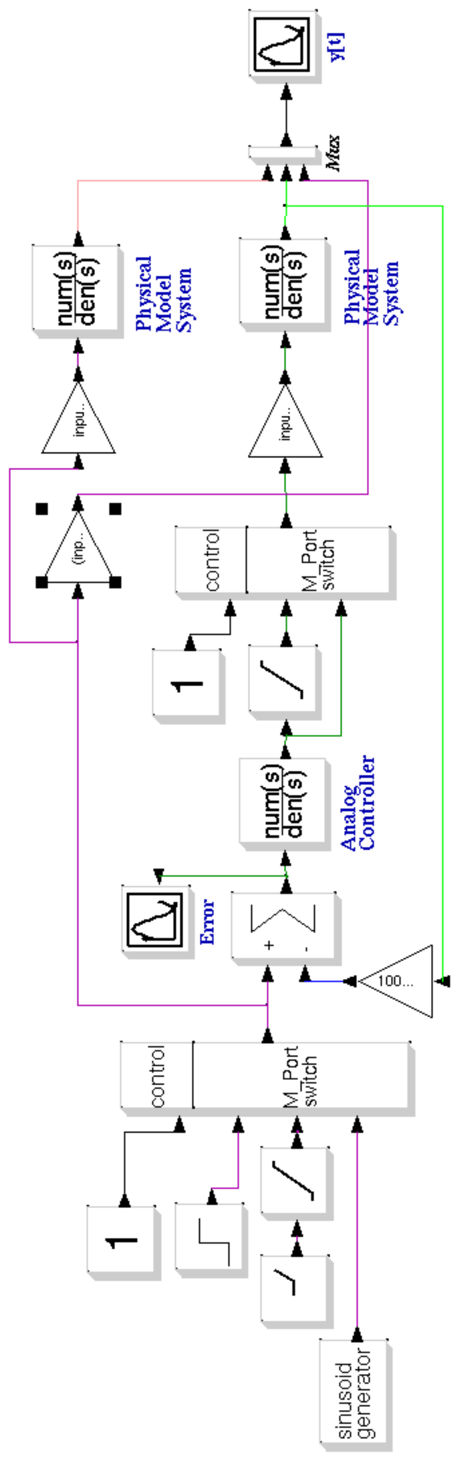

Figure 14: The Closed Loop System Circuit in the continuous time domain.

As it is shown in figure 14, we use an input step function signal for our system. In between the transfer function of the analogical controller and the motor system's transfer function we put a saturation block that accounts for non-linearities and the impossibility of setting the relative speed value of the physical motor outside the interval of values $[-100, 100]$ (a typical *windup* phenomenon).

The evolution of the output signal and of the error are shown respectively in figure 15 and 16.

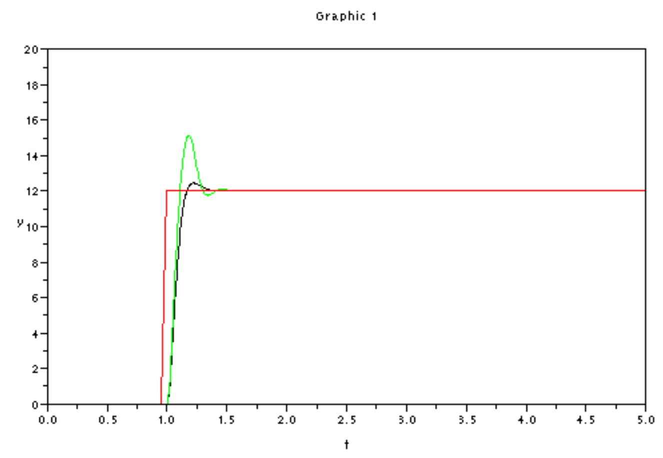

Figure 15: Input Step Signal (red), output signal in with Closed Loop System (green) and output signal with Open Loop System (black).

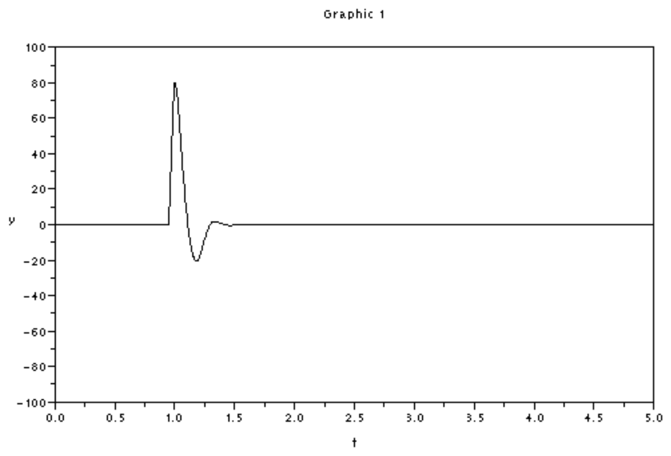

Figure 16: Evolution of the error of the Closed Loop System simulation in continuous time domain.

From the output plots of the continuous time domain simulation we can see that our *Closed Loop System* is stable and all the constraints are satisfied because:

- the Overshoot comes at nearly $100ms$ and its peak is not over 30% of the steady state value;

- the Error goes to zero as time flows, and the steady state value is stable around the right value;
- the Rise time comes at $80ms$ and the settling time is less than $300ms$;

2.4 Digital Model of the Closed Loop System

Once we get in the digital world we need to transform our continuous time Controller into a discrete time Controller, that is substituting the *Laplace* transfer function with the *Zeta-transform* in the *scicos block* of the Controller. One of the easiest and fastest approaches to accomplish this task is to use the **Bilinear Transform**. This method maps all positions along the $j\omega$ axis, with $Re(s) = 0$, of the *s-plane* to the unitary circle of the *z-plane*. Correspondingly, the entire left *s-plane* is mapped within that unitary circle, while the right half *s-plane* is mapped onto the values outside that boundary. Therefore one can easily conclude that a *zeta* transfer function is stable when all its poles are placed within the unitary circle.

Applying the **Bilinear Transform** is as simple as replacing all occurrences of s with the transform equation

$$s = \frac{2}{T} \cdot \frac{z-1}{z+1}$$

where T is the sampling period, in our case $5ms$. Applying this transform to the Controller transfer function gives us:

$$C(z) = 3.5 \cdot \frac{(0.7453594 - 1.6592812z + 0.9286927z^2)}{(0.6666667 - 1.6666667z + z^2)}$$

As expected the roots of the denominator fall within the unitary circle: 1 and 0.6666667.

To correctly model the behaviour of the *LEGO NXT MOTOR* we should also take into account the fact that on the brick we can only retrieve estimates of the covered space, which is quantized with 1 *degree* of precision. Therefore in spite of directly using the output values of the speed at which the motor is running, we should instead derive it from a sequence of quantized covered space estimations. The overall digital circuit is shown in figure 17.

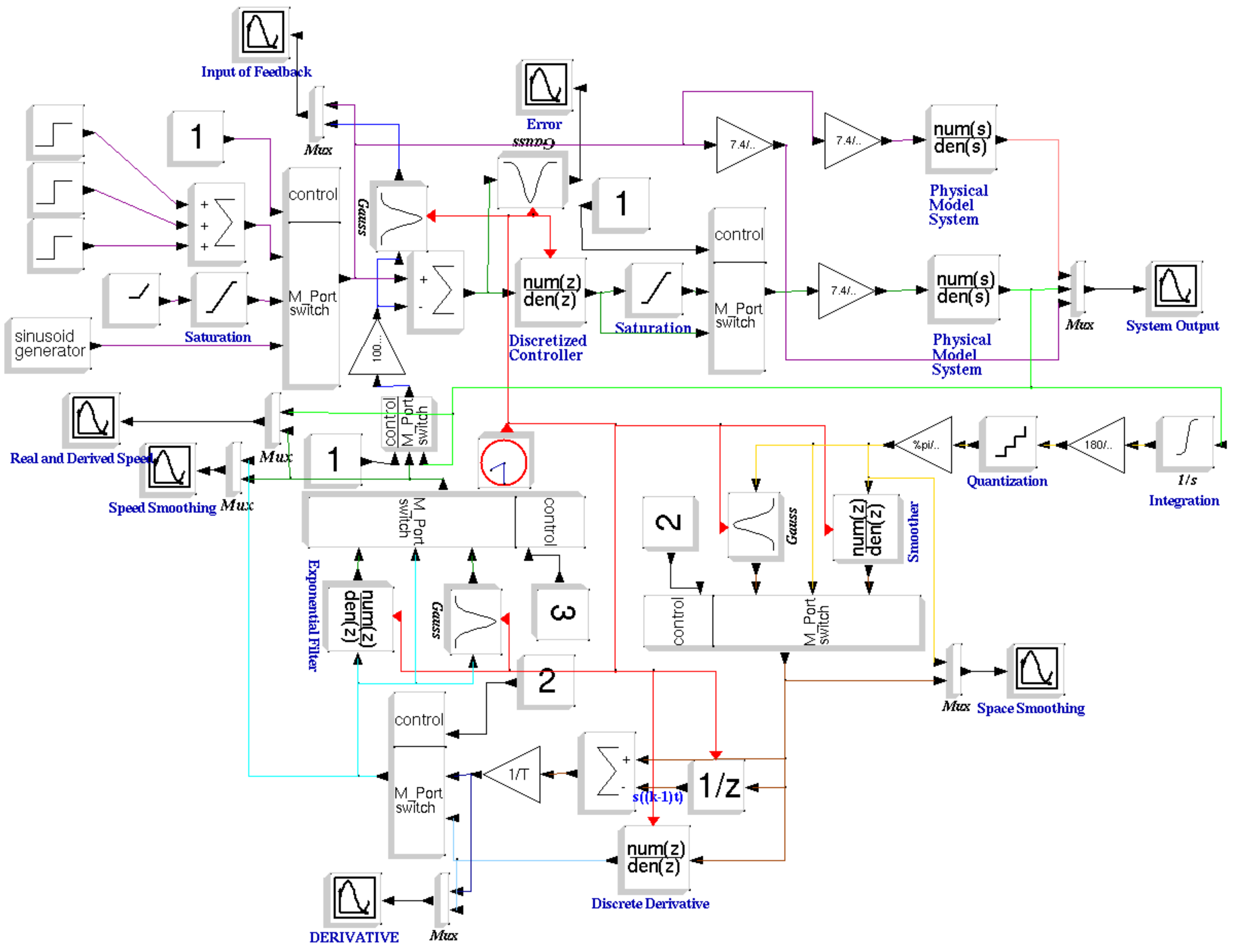

Figure 17: The Closed Loop System Circuit in the discrete time domain.

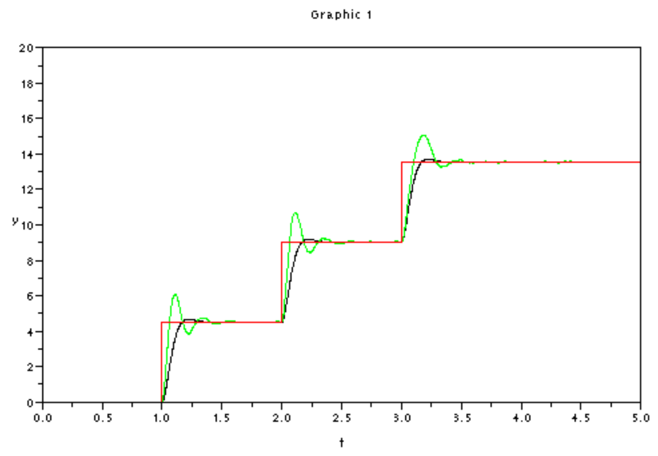

Figure 18: Input Step Signal (red), output signal with Closed Loop System (green) and output signal with Open Loop System (black).

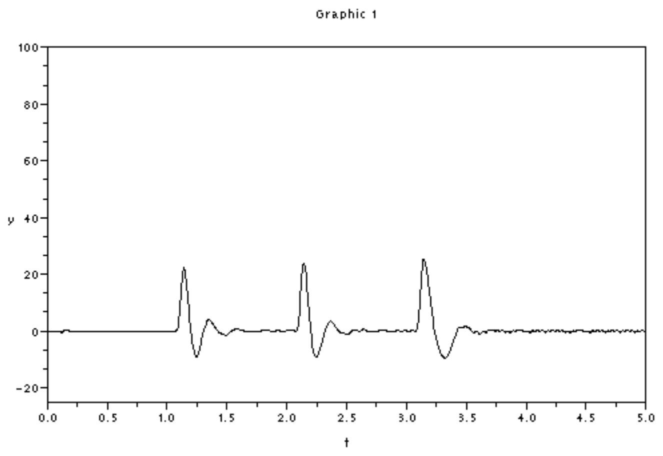

Figure 19: Evolution of the error of the Closed Loop System simulation in discrete time domain.

The output signal and the error evolution with the digital circuit simulation are shown respectively in figure 18 and 19. Once again, note that the error goes to zero as time lapses and that the output signal is compliant with the imposed requirements as expected.

Note that to achieve the results shown above we had to apply some filtering technique to ensure that noise and spurious results due to the quantization did not affect the reliability of the system. We decided to use the *exponential filter* with forget factor of 0.5 since it was both effective and simple enough to be implemented. In figure 20 it is possible to appreciate the improvement gained in the speed estimation by applying the filter.

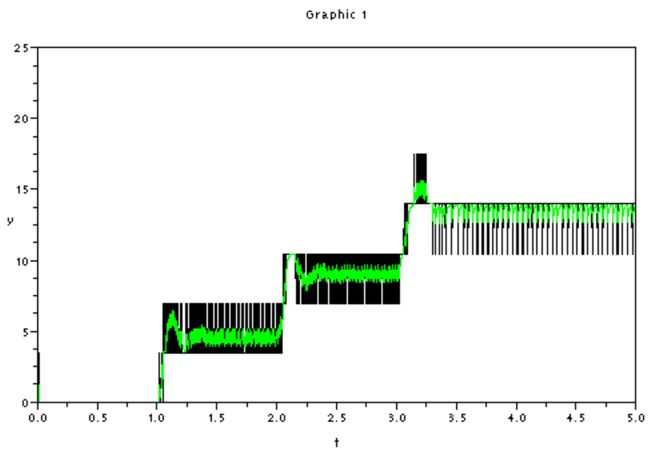

Figure 20: Filtered speed signal (green) obtained from the raw speed values (black) retrieved using the quantized space estimator.

2.5 *LEGO NXT MOTOR* Implementation

The last part of the assignment mandated the development of the controller inside the brick, thus concluding the work previously done on the root locus and the *Scicos* simulations. The discrete controller function implemented on the Brick has been computed from the *zeta-transform* of the Controller transfer function with the following steps, in which for simplicity the coefficients have been labelled:

$$\begin{aligned}
u(k) &= kc \cdot \frac{(e_2 + e_1 q + e_0 q^2)}{(u_2 + u_1 q + u_0 q^2)} \cdot e(k) \\
(u_2 + u_1 q + u_0 q^2) \cdot u(k) &= kc \cdot (e_2 + e_1 q + e_0 q^2) \cdot e(k) \\
u_2 u(k) + u_1 u(k) q + u_0 u(k) q^2 &= kc \cdot (e_2 e(k) + e_1 e(k) q + e_0 e(k) q^2) \\
u_2 u(k) + u_1 u(k+1) + u_0 u(k+2) &= kc \cdot (e_2 e(k) + e_1 e(k+1) + e_0 e(k+2)) \\
u(k+2) &= \frac{kc \cdot (e_2 e(k) + e_1 e(k+1) + e_0 e(k+2)) - u_2 u(k) - u_1 u(k+1)}{u_0}
\end{aligned}$$

Since we can not know the future values of the signal, we then applied a change of variable thus obtaining the equation:

$$u(k) = \frac{kc \cdot (e_2 e(k-2) + e_1 e(k-1) + e_0 e(k)) - u_2 u(k-2) - u_1 u(k-1)}{u_0}$$

For convenience, and coherency with the previous simulations, we decided to transform the angular velocity in the rate of power that can be fed to the motor using a proportional transformation. Part of the code running on the *brick* is shown in the *C* piece of code below, where you appreciate the simplicity of the control logic that maintains the desired performances for the *LEGO NXT MOTOR*.

```c
// >> CONTROLLER
// Accumulators
float uk_1 = 0;
float uk_2 = 0;
float ek_1 = 0;
float ek_2 = 0;
// Coefficients
float e_2 = 0.7453594;
float e_1 = -1.6592812;
float e_0 = 0.9286927;
float u_2 = 0.6666667;
float u_1 = - 1.6666667;
float u_0 = 1.0;
// KC
float Kc = 3.5;

/**
 * CONTROLLER:
 * Apply the controller computations based on incoming informations
 */
ek_0 = test_target_power - feedback_power_f; // current feedback error
uk_0 = (Kc*((e_2*ek_2)+(e_1*ek_1)+(e_0*ek_0))-(u_2*uk_2)-(u_1*uk_1))/u_0;
   // current output
ek_2 = ek_1; // Update Discrete Historical values
ek_1 = ek_0;
uk_2 = uk_1;
uk_1 = uk_0;
// Apply Saturation
if (uk_0 > 100)
        uk_0 = 100.0;
if (uk_0 < -100)
        uk_0 = -100.0;
```

2.5.1 *LEGO NXT MOTOR* execution results

In order to verify the correctness of our implementation and its capability of respecting the initial constraints, we have done a little of experimentation with our brick.

Step Function

The first experiment consisted into repeating the same kind of analysis that has been done during the **First Report**. Hence, we have run 50 tests for all speeds in the power rates interval $[10, 100]$, each one distanced by 5 from the others. We will not show all the graphs and data regarding this analysis here but only the final configuration that our *LEGO NXT MOTOR* system reaches:

- $\mathbf{K} = 2.033363$
- $\omega_{\mathbf{n}} = 26.656639$
- $\xi = 0.625341$

The plot 21 shows a typical sequence of power requests that the controller makes to the *LEGO NXT MOTOR* during a test with an input step function. The high variability of these values could have been decreased by extending the size of the average window that filters the feedback power values, although this would have also introduced a stronger delay in the controller capability of reacting to changes of its input signal.

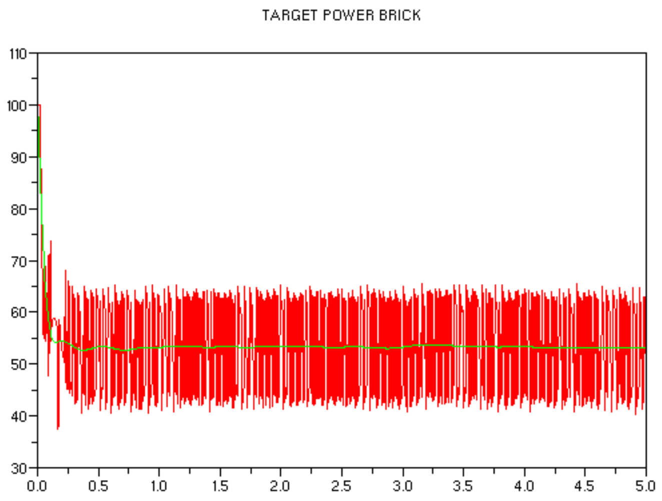

Figure 21: Controller power response with an input step function of 55%.

The plot of figure 22 shows instead the evolution of the feedback power, estimated using the space readings coming from the motor and already filtered. Note that, since the option $brake = ON$ makes the speed of the brick linearly proportional to the input power, this graphs represents also the evolution of the speed of the motor.

Although the results vary a bit using input step functions of different power level and that the filtering technique applied to the raw speed data introduces some delay in the process, it has been

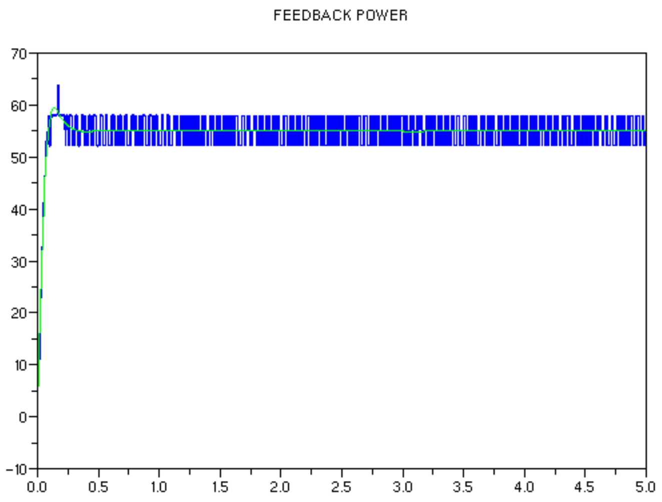

Figure 22: Feedback power, proportional to the motor speed, with an input step function of 55%.

possible to estimate that on average we meet our acceptance constraints: the overshoot is $\leq 30\%$, the $Ts \leq 200ms$, the $Rt \leq 80ms$. The only constraint that apparently is not met is the overshoot time Ot, that appears to be around $135ms$ instead of the requested $100ms$.

A little of delay, around $15ms$, is to be expected as a consequence of the average filtering technique implemented that slows down the capability of our controller to react to the input values. For the remaining part, this delay should be ascribed to the difficulty of properly filtering and identify the underlying SOS model that characterizes the output signal. This difficulty not only affects the current estimation of the parameters, but it has also affected the results obtained in the **First Report**. Therefore, it may be due to a slight imprecision in the estimation of the plant's poles that the effectiveness of the Controller is reduced in the real implementation.

After several experimentations with different Root Locus and filtering techniques we could not produce any significant improvement in the overshoot time of the output signal.

Step Function with Weight

We also wished to test our Controller against perturbations induced by the external environment on the motor freedom of rotating. For this reason we have developed an experiment that tests the behaviour of the Controller when one of its motors is repeatedly stressed with an oscillating acceleration. To accomplish this task we took advantage of the earth gravitation force, building ado *LEGO* block made by a couple of gears: one connected to the engine and the other to an external support. Then we attached an off-axis mass to the latter gear, so that the controller would have been forced to increase the motor power during the lift phase and decrease it during the descendant phase. The scheme of this implementation is shown in figure 23.

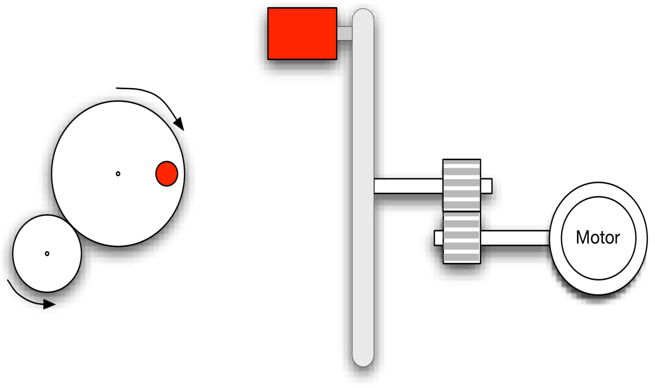

Figure 23: Simple model of the experiment

Although it could have been possible to estimate the gravitational force exerted on the mass and verifying that the resulting acceleration produced by the *Controller* was compliant to our experimental data, due to time constraints we decided to be content with a visual appreciation of the resulting signal. As expected, the controller adjusts the power requests imposed to motor to oppose the sinusoidal acceleration determined by the external environment, as it is shown in figure 24. On the converse, despite of the permanent acceleration stress imposed, the speed appears to be stably clanged to the target speed.

Note that this experiment lasted around $30s$ with samples every $5ms$, thereby it is natural that these plots appear to be densely populated.

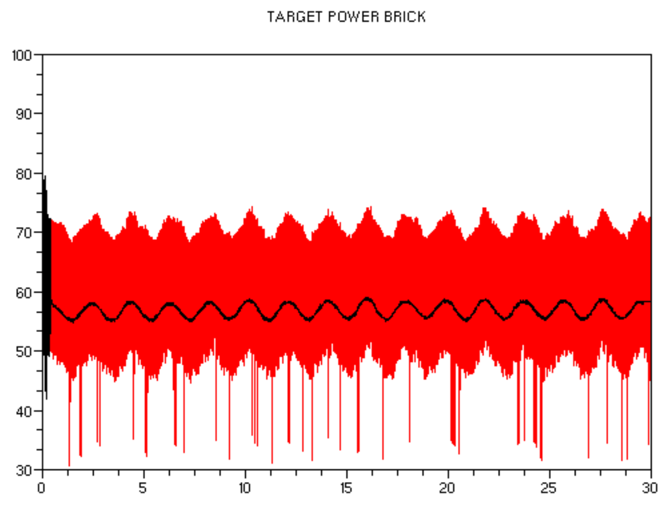

Figure 24: Controller power response with a off-axis mass perturbation afecting the motor.

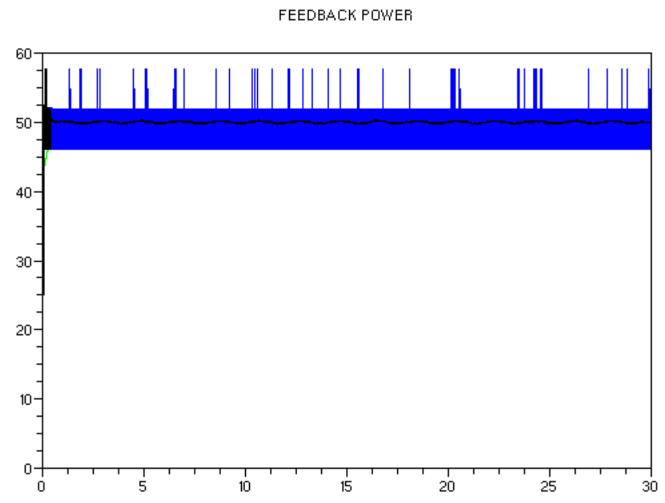

Figure 25: Feedback power, proportional to the motor speed, with an off-axis mass perturbation affecting the motor.

3 Assignment 3: Unicycle Vehicle controller design and implementation

The goal of the third and last part of this report is to design and implement a controller that allows the robot built with *LEGO NXT* to approach and follow a line using as reference the Unicycle Kinematic model. The first part of this report is a revision of the wheel controller design phase, followed by a brief explanation of the Unicycle Model. After that we go deeper into the details of the vehicle controller design and implementation, tackle some performance tuning issues and conclude by evaluating the overall performances of the robot in the real world environment.

3.1 Review of *LEGO NXT MOTOR* Controller Design

Before starting the final part of our laboratory project, we decided to review the work done in the second assignment. Specifically we wanted to tackle two issues still open:

- The **drift** introduced by the **saturation** block in the signal response to input step functions requesting speed values $\geq 90\%$ of the maximum speed, as shown in figure 26;

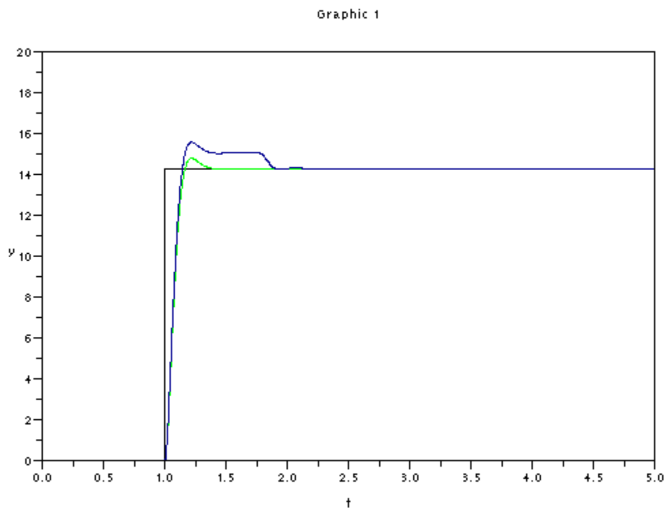

Figure 26: Digital Simulation output: input step signal (black), open loop response (green) and closed loop response (blue)

- The **delay** in the motor response detected using the actual data of the *LEGO NXT MOTOR* with our first

$$\begin{aligned}
u(k) &= kc \cdot \frac{(e_1 + e_0 q)}{(u_1 + u_0 q)} \cdot e(k) \\
(u_1 + u_0 q) \cdot u(k) &= kc \cdot (e_1 + e_0 q) \cdot e(k) \\
u_1 u(k) + u_0 u(k) q &= kc \cdot (e_1 e(k) + e_0 e(k) q) \\
u_1 u(k) + u_0 u(k+1) &= kc \cdot (e_1 e(k) + e_0 e(k+1)) \\
u(k+1) &= \frac{kc \cdot (e_1 e(k) + e_0 e(k+1)) - u_1 u(k)}{u_0}
\end{aligned}$$

implementation of the controller, which was estimated to be around $35ms$;

3.1.1 Anti Wind Up

When the **saturation** block constraints the values of the response of the controller, the estimated error may become significant. In fact as time lapses the integral action of the controller becomes more significant but the signal is already saturated. This situation folds itself into a vicious circle that diminishes the system performances, known as "**Wind Up**".

Common solutions to this problem are:

- Limitation of the reference signal: by avoiding sharp variations in the input signal (e.g. dividing it's power into a sequence of smaller step functions) the controller's response is less sharp and does not incur into saturation. However this choice degrades system performances;
- Conditional Integration: by turning off the integrator part in certain conditions (e.g. when signal is saturated and error/control variable have same value) it is possible to avoid increasing the error for a certain amount of time;
- Back Calculation: when the controller occurs in saturation the integral term is recomputed, so that its action is diminished by a value that is proportional to the saturation action.

We decided to implement the third option in our **Anti Wind Up** system since it appeared to be more than a hack to just make things work. To do so, the first step has been to compute the *partial fraction expansion* of our digital controller so to isolate its integrating component:

$$\begin{aligned}
c_1 &= \tfrac{8.8625}{s} \\
c_2 &= \tfrac{-44.8625}{80+s} \\
c_3 &= 1
\end{aligned}$$

Next we apply the bilinear transform to obtain the digital controller components in the *z-plane*:

$$
\begin{aligned}
c_1 &= \frac{8.8625+8.8625q}{-400+400q}\\
c_2 &= \frac{-44.8625-44.8625q}{-320+480q}\\
c_3 &= 1
\end{aligned}
$$

The next step is to put together the newly find controller components into a scicos circuit, where the integrating part is corrected with the derivative of the error produced by the saturating block. As shown in figure 27 our new implementation of the **wheel controller** contains both the old single controller and the new anti wind up system. In figure 28 it is possible to appreciate the improvement achieved: now the signal overshoot does no more stand to its peak value for some time but it immediately goes toward the desired final value as expected.

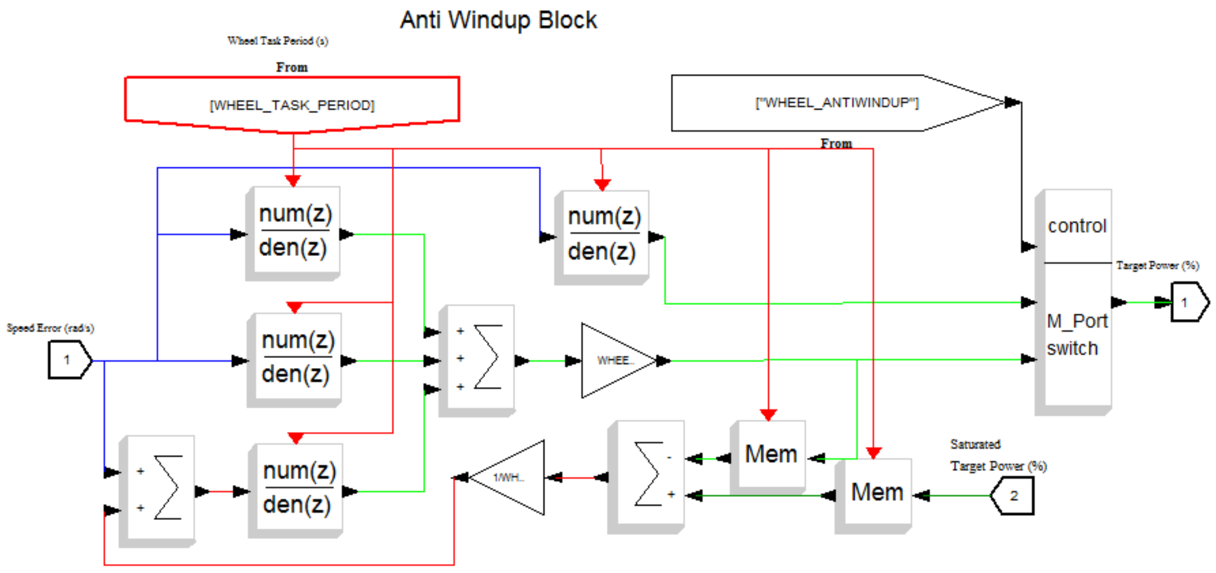

Figure 27: Scheme of Digital Anti Wind Up Controller

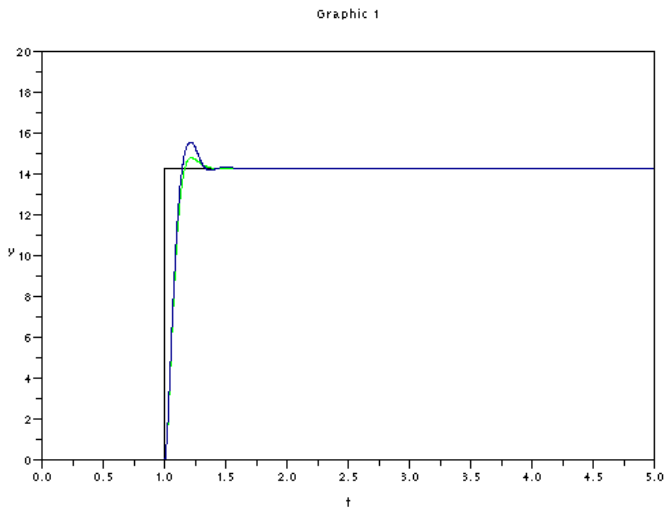

Figure 28: Digital Simulation output with Anti Wind Up: input step signal (black), open loop response (green) and closed loop response (blue)

Once verified that the digital simulation worked, the circuit has been implemented in the *LEGO brick*.

3.1.2 Smith Predictor

In order to compensate the delay of $35ms$ over the output signal that we detected after the implementation in the actual *LEGO brick*, we decided to use a predictive controller known as *Smith Predictor*.

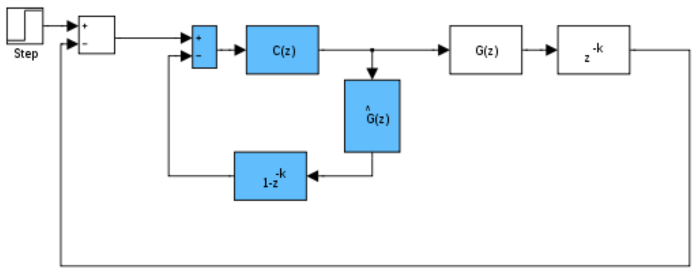

Figure 29: Smith Predictor Circuit [source: Wikipedia]

The *Smith Predictor* circuit is shown in figure 29. While the outer loop still feeds the output signal back to the input that is affected by a constant delay effect, an additional inner loop contains the predictor of what the unobservable output of the plant currently is. This circuit has been rearranged into the circuit shown in figure 30.

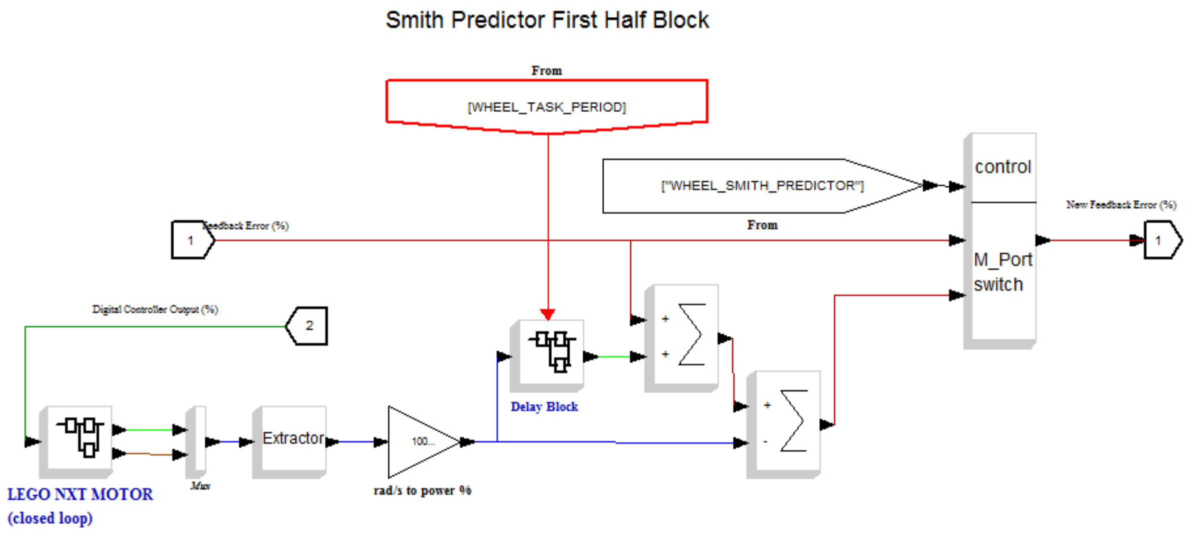

Figure 30: Smith Predictor Circuit in Scicos

Note that in order to make the simulation work, we had to add a signal delay effect to the feedback data. The circuit that adds the delay is depicted in figure 31.

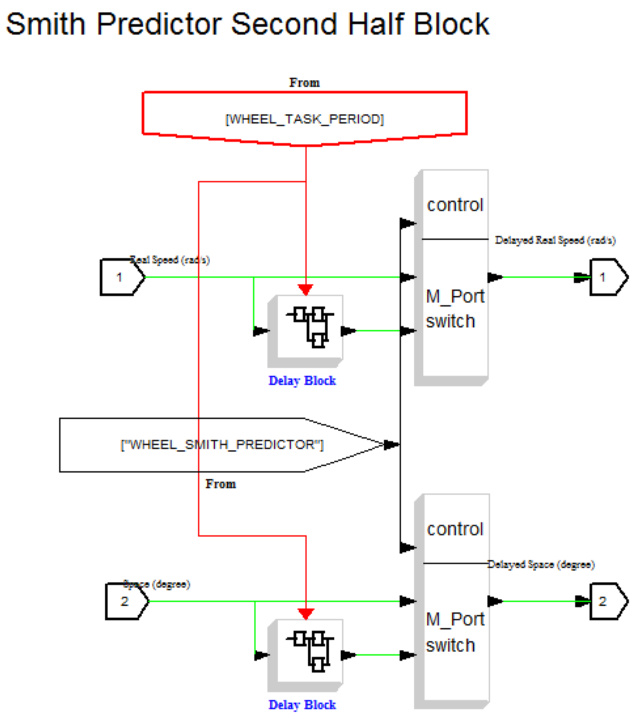

Figure 31: Delay Over the signal

The output signal obtained using the Smith Predictor is shown in figure 32, where the *blue* signal represents our delayed output signal and the *red* signal stands for the actual output signal that we are able to produce thanks to the predicting controller.

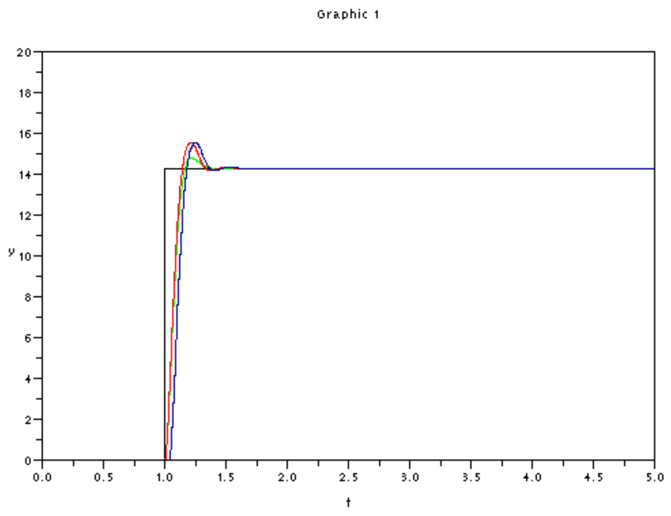

Figure 32: Digital Simulation Output with Smith Predictor: input step signal (black), open loop response (green), closed loop response (red) and delayed closed loop response (blue)

The final configuration of the Wheel Digital Circuit with the newly added components it is shown inf figure 33. As it is possible to see, some components already shown in the previous assignment have been encapsulated so that the overall picture could be more clear.

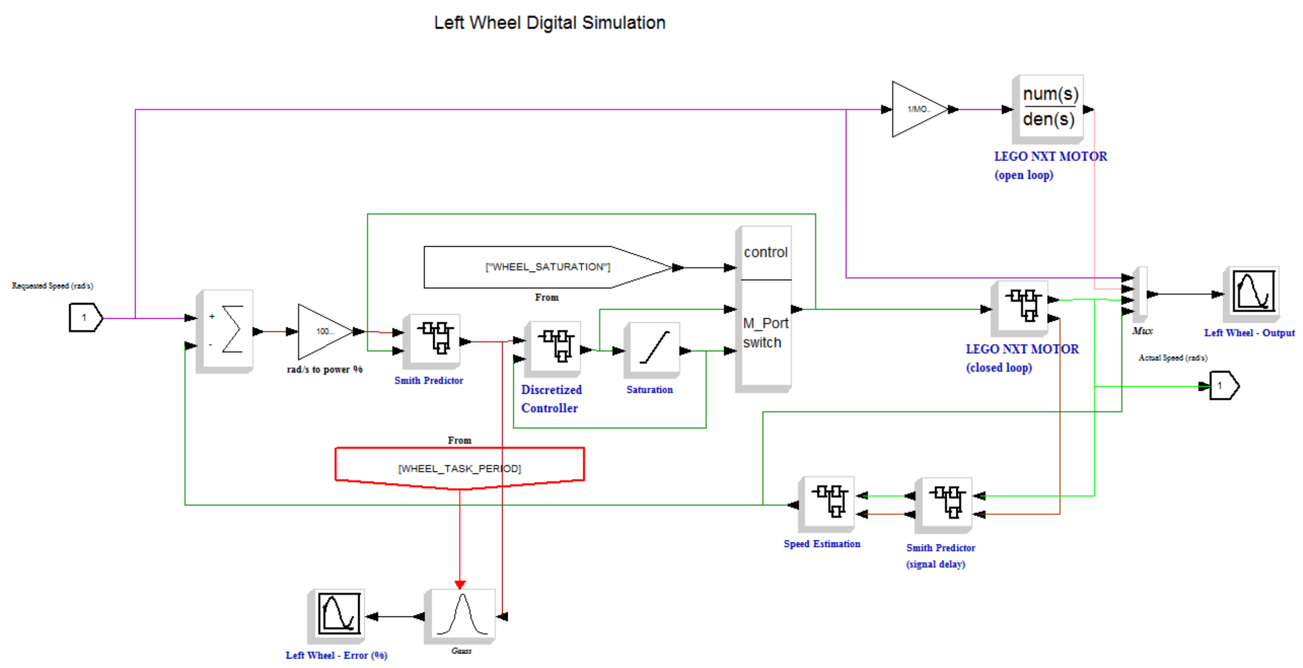

Figure 33: Digital Simulation Output with Smith Predictor: input step signal (black), open loop response (green), closed loop response (red) and delayed closed loop response (blue)

3.2 Unicycle Model

In this section we will model the behaviour of a *Unicycle Vehicle* using the differential drive. A unicycle type robot is in general a robot moving in a 2D world having some forward speed but zero instantaneous lateral motion. In other words it is a **non-holonomic system**. An example of *Unicycle Vehicle* is given in figure 34, where the vehicle is characterized by a couple of parallel driven wheels, with their own motor, and a third small caster wheel.

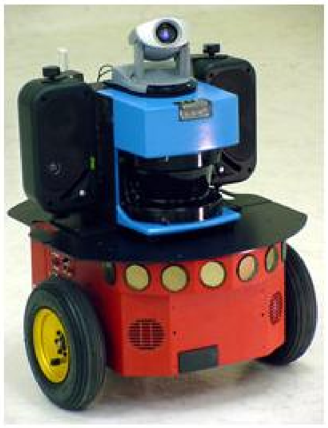

Figure 34: The Pioneer 3-DX8 typical unicycle vehicle

3.2.1 Kinematic Model

In order to control and drive this kind of vehicle we need a **Kinematic Model** that describes the trajectories that the mobile robots follows when subject to commanding speeds. The Kinematic of this system is simple because we have only *non-holonomic* limits. So all the configurations that can be reached by the robot can be described by the vector of coordinates

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

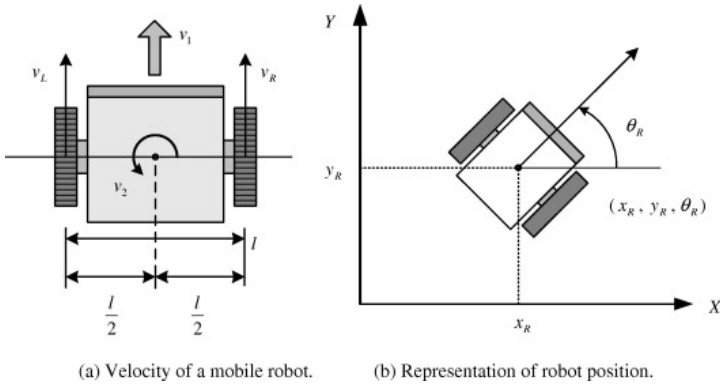

Figure 35: The kinematic model of an non holonomic system

In agreement with figure 35 the *Kinematic Model* can be described by the following non-linear system

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \end{cases}$$

where *v* is the speed of the vehicle and ω is the angular speed of the vehicle.

By knowing that *R* and *L* are respectively the radius of the wheels and the distance between the wheels, we can compute the angular speed of the left and right motor as

$$\begin{cases} v = \frac{R}{2}(\omega_r + \omega_l) \\ \omega = \frac{R}{L}(\omega_r - \omega_l) \end{cases}$$

3.2.2 Transfer function of a path following problem

Since our goal is to follow an imaginary line parallel to the wall we can simplify the Kinematic model by doing some assumptions:

- The robot will move forward with a constant speed;
- The X axis is parallel to the line that we want to follow;

Therefore we obtain the following model:

$$\begin{cases} \dot{y} = v \sin\theta \\ \dot{\theta} = \omega \end{cases}$$

It is possible to notice that the kinematic model is non-linear, so we have to linearise it as we want to use a linear controller to control the robot. By using the **notable limitation** $\lim_{x\to 0} \frac{\sin x}{x} = 1$ and assuming that $\sin\theta$ can be approximated by a straight line in contiguous instants (with low values of angular speed ω) we get the desired linear kinematic model:

$$\begin{cases} \dot{y} = v\theta \\ \dot{\theta} = \omega \end{cases}$$

Therefore a first loose approximation of the *transfer function* associated to the vehicle behaviour with our linear kinematic model is given by:

$$Y(s) = \frac{V_{speed}}{s^2} U(s)$$

To improve such an estimation, we may take into account the presence of the wheel engines for which we already computed a transfer function. We aim to find the wheels block contribution to the overall system, hence we look for an input/output relation of type $\omega^\star = G_M(s) \cdot \omega$. We may initially simplify our computations by imposing:

$$\begin{aligned} \omega_r &= \alpha \cdot V_{speed} + \beta \cdot \omega \\ \omega_l &= \alpha \cdot V_{speed} - \beta \cdot \omega \end{aligned}$$

Therefore, provided that $G_r(s) = G_l(s)$ and $\beta = \frac{2 \cdot R}{L}$:

$$\begin{aligned}
\omega^\star &= G_M(s) \cdot \omega \\
&= \left(\frac{R}{L}\right) \cdot (G_r(s) \cdot \omega_r - G_l(s) \cdot \omega_l) \\
&= \left(\frac{R}{L}\right) \cdot (G_r(s) \cdot (\alpha \cdot V_{speed} + \beta \cdot \omega) - G_l(s) \cdot (\alpha \cdot V_{speed} - \beta \cdot \omega)) \\
&= \left(\frac{R}{L}\right) \cdot (2G_r(s) \cdot \beta \cdot \omega) \\
&= G_r(s) \cdot \omega
\end{aligned}$$

So what we have found out is that the contribution of the entire wheel component is actually due only to the closed loop transfer function of such a system, which is $G_r(s) = G_l(s) = G_M(s) = W_{cltf}(s)$:

$$Y(s) = \frac{W_{cltf}(s) \cdot V_{speed}}{s^2} U(s)$$

3.2.3 Estimation of the real distance

Since we need the distance from the wall we have to use the *NXT ultrasonic sensor* witch is an hardware that can measure distances from some obstacle up to $255cm$ and with a precision of $\pm 3cm$.

The ultrasonic sensor is always perpendicular in respect to the robot and not in respect the wall, therefore when the robot is steering there is an angle β in respect to the wall that results in a wrong measure of the distance. This problem can compromise the performances of the robot, so it is useful to estimate the orientation of the robot in order to compensate the distance readings obtained from the ultrasonic sensor.

Under the assumption that the robot proceeds along a straight line among subsequent samples taken at intervals of $50ms$, we can use simple trigonometry rules in order to estimate the angle β in respect to the wall and subsequently compute the real distance d. From the configuration of the vehicle shown in figure 36 one can derive the following formulas:

$$\begin{aligned}
\Delta x &= v \cdot T_s \\
\beta &= \arctan\left(\frac{d_2 - d_1}{\Delta x}\right) \\
d_{real} &= \cos(\beta) \cdot d_2
\end{aligned}$$

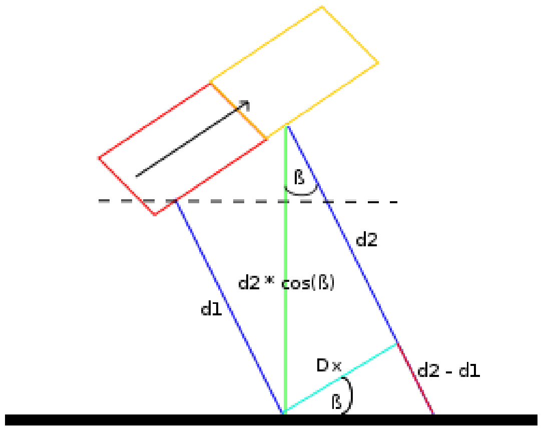

Figure 36: Model of the trigonometry used for the estimation of the real distance

3.3 Vehicle Controller Design

The vehicle controller design phase retraces the same steps taken for the wheel controller design, thus we will show very briefly all the essential steps without going too much deep into the theoretical details.

3.3.1 Root Locus

The **root locus** will be designed to support a vehicle speed $V_{speed} = 0.34m/s$, corresponding to 80% of the power that can be given to the motors. This value has been chosen since it leaves ample space for manoeuvre to the additional speed boost caused by the steering behaviour. The desired performances of the *Vehicle Closed Loop System* are given by:

- **Max Overshoot (O_{max})**: 0.01%
- **Settling Time (St_{max})**: $5s$
- **Rise Time (R_t)**: $5s$
- **Overshoot Time (O_t)**: $5s$
- **Steady State Value Tolerance (α)**: 5%

Using these values it's straightforward to compute the parameters of the acceptance region, now defined by the values $\omega_n = 0.56$ and $\xi = 0.83$. This time we accept a behaviour that is (a bit) slower than our requirements since we are interested into avoiding too sharp steering behaviours, therefore we consider the inner part of the slice to be our acceptance region.

The *Closed Loop System* transfer function is once again given by:

$$V_{cltf}(s) = \frac{V_C(s) \cdot V_P(s)}{1 + (V_C(s) \cdot V_P(s))}$$

As previously seen, $V_P(s)$ has two possible formulations:

$$\begin{aligned} V_P &= \tfrac{V_{speed}}{s^2} \\ V_P &= \tfrac{W_{cltf}(s) \cdot V_{speed}}{s^2} \end{aligned}$$

where $W_{cltf}(s)$ stands for the *Closed Loop System* transfer function of the motor.

In order to get a dominating first order behaviour, that reduces sharp steering behaviours, we add to the root locus a zero in -0.35 and a pole in -2.75. This time it is not necessary to add poles in zero to obtain zero steady state error tracking, since we already have two. The gain (kc) of the vehicle controller is set to 10. The final root locus is shown in figure 37, while its analogical formulation is:

$$V_C(s) = 10 \cdot \frac{(s + 0.35)}{(s + 2.75)}$$

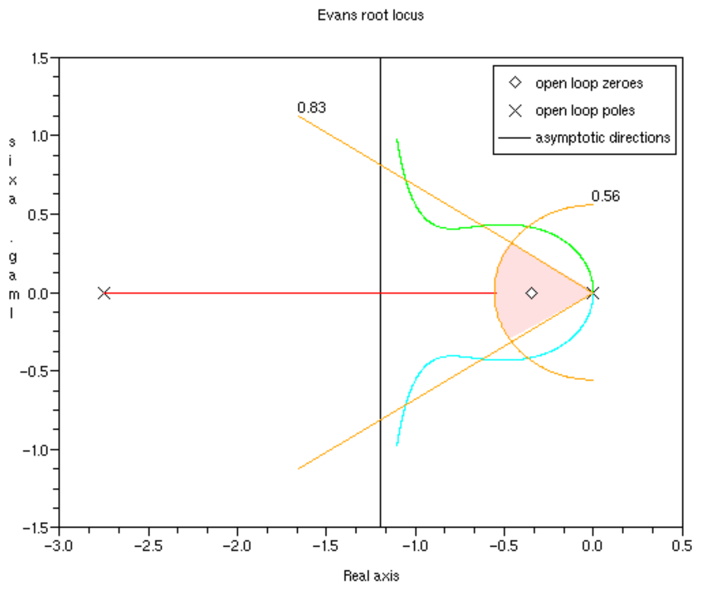

Figure 37: Root Locus for the Vehicle Controller with a first order dominating behaviour design

3.3.2 Digital Model of Vehicle System

At this stage of the work we are going to create the model that is representing the current situation. The model is always represented by a *Scicos* scheme.

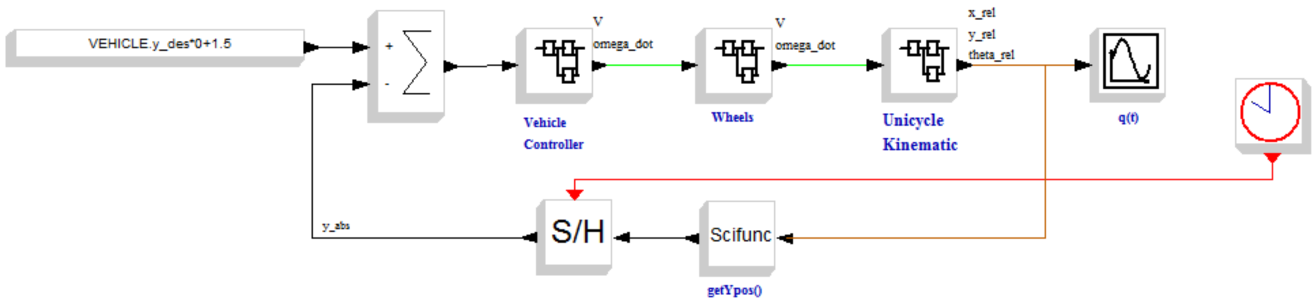

Figure 38: Complete digital model of the system

In figure 38 we can see the general vehicle model which contains the vehicle controller, the block of the engines and the block of the kinematic model. This digital model is used to simulate the behaviour of the robot which, with the aid of some *Scipad* functions, can be plotted on the Cartesian plane. Let's focus on the various blocks in the diagram to understand how the model components work and glue together in the overall picture.

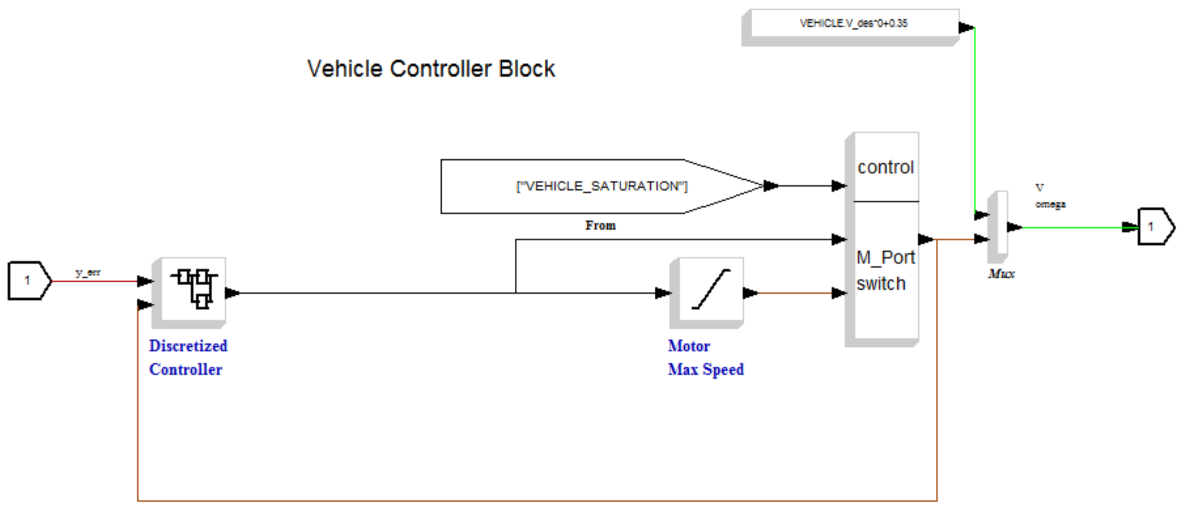

Figure 39: Controller of the vehicle

The first component we will drag our attention on is the controller of the vehicle, shown in figure 39. This super-block contains the actual vehicle controller obtained from the root locus phase and an optional saturation block. The controller receives as input the difference between the desired and the actual distance, while it returns as output the angular speed ω_{dot} of the vehicle. This value is then combined with the constant forward speed and passed to the wheels.

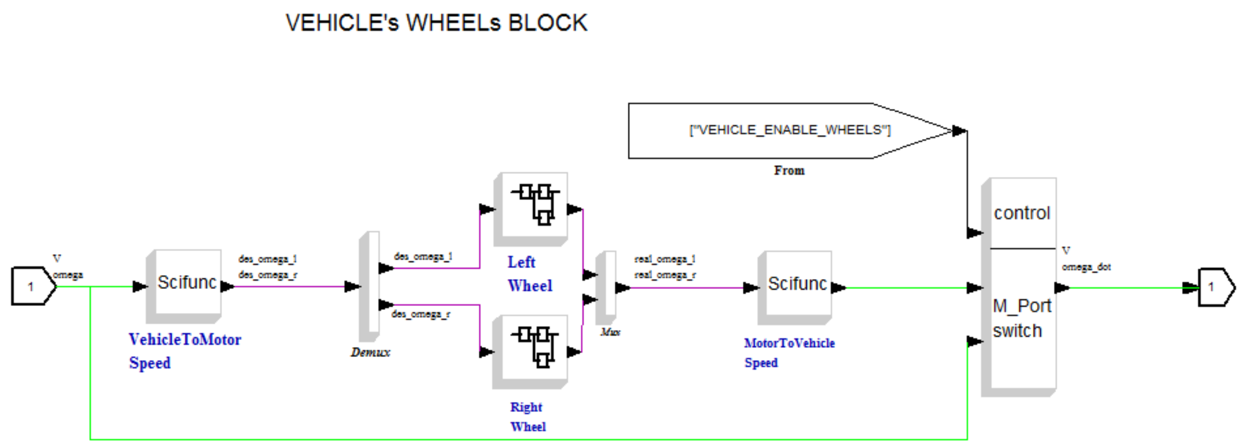

Figure 40: Model of the engines

The second block, shown in figure 40, simulates the behaviour of a wheel. In this diagram the first component on the left is a function called *VehicleToMotor-Speed* that maps the values of $< v, \omega_{dot} >$ describing the kinematic of the vehicle into the corresponding angular speeds of the wheels. The relation is given by the following equations:

$$\begin{aligned}\omega_l &= (2 \cdot v - VEHICLE.rod\_length \cdot \omega_{dot})/(2 \cdot VEHICLE.wheel\_radius);\\ \omega_r &= (2 \cdot v + VEHICLE.rod\_length \cdot \omega_{dot})/(2 \cdot VEHICLE.wheel\_radius);\end{aligned}$$

Since the wheels dynamic is completely transparent from the vehicle point of view, the feedback $< \omega_l, \omega_r >$ obtained by analysing the wheel rotation are then mapped back into the vehicle variables by inverting the previous equations:

$$\begin{aligned}v &= (\frac{VEHICLE.wheel\_radius}{2}) \cdot (\omega_r + \omega_l)\\ \omega_{dot} &= (\frac{VEHICLE.wheel\_radius}{VEHICLE.rod\_length}) \cdot (\omega_r - \omega_l);\end{aligned}$$

The internal look of the engine circuit, already shown in figure **??**, has been already explained in the part of the report devoted to the wheels controller development and will not be covered here again.

The last component we examine is the Unicycle Kinematic block that takes as input v and ω_{dot} and computes the values $< x, y, \theta >$ describing the current configuration of the vehicle in the Cartesian space.

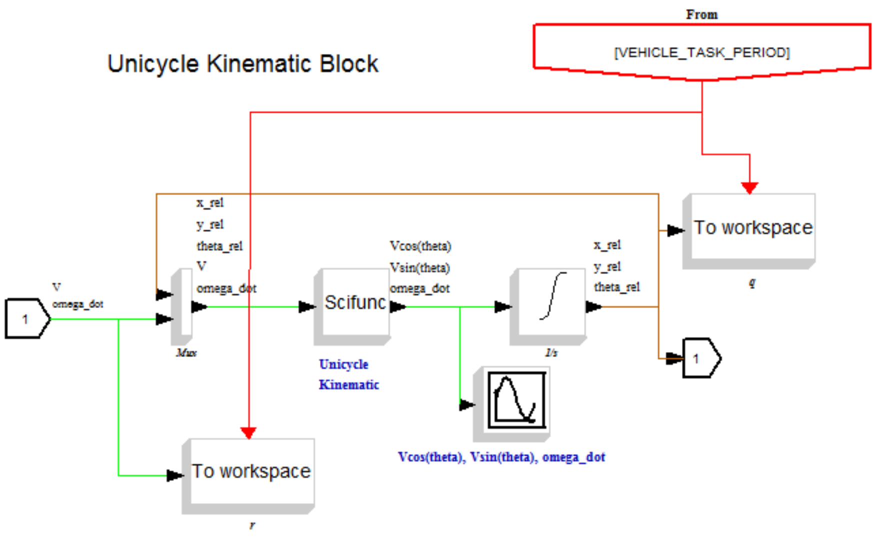

Figure 41: Model of one Unicycle Kinematic Block

3.3.3 Vehicle Controller Simulation

The behaviour of the overall system has been simulated in *Scicos* using both an *analogical* and *digital* circuit. However we will show here only the result of the digital simulation, since it is the most significant data before jumping in the implementation phase.

Since the *Sonar* can obtain new values at intervals of roughly $30/50ms$ and sometimes it returns spurious data, we decided to keep the controller time period set to $100ms$. This design choice is reinforced by the fact that at worst our wheels block may require up to $135ms$ to stabilize the wheels speed around the desired value. The presence of the speed is completely transparent to our vehicle model, hence we need to maintain the outer loop period big enough so that it is not affected by the feedback loop system of the wheels. Using the **bilinear transform** we obtain the *z-plane* transform of the vehicle controller:

$$V_C(s) = 10 \cdot \frac{(-0.8637363 + 0.8945055z)}{(-0.7582418 + z)}$$

As it is shown in figures 42 and 43 the overall behaviour of the vehicle respects the imposed requirements. The steering angle of the vehicle is always less than $0.418rad$, a value for which the linearising simplification of $sin(\alpha) = \alpha$ results in an intrinsic error of 3%.

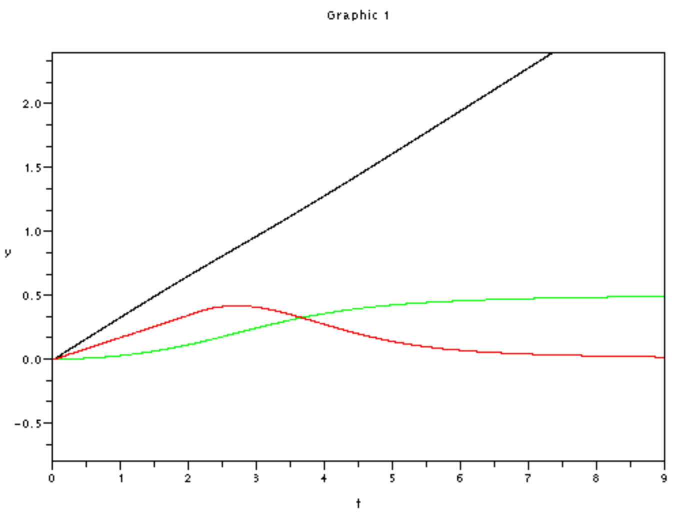

Figure 42: Digital simulation output showing the distance from the wall (green), the steering angle computed by the vehicle controller (red) and the covered space (black)

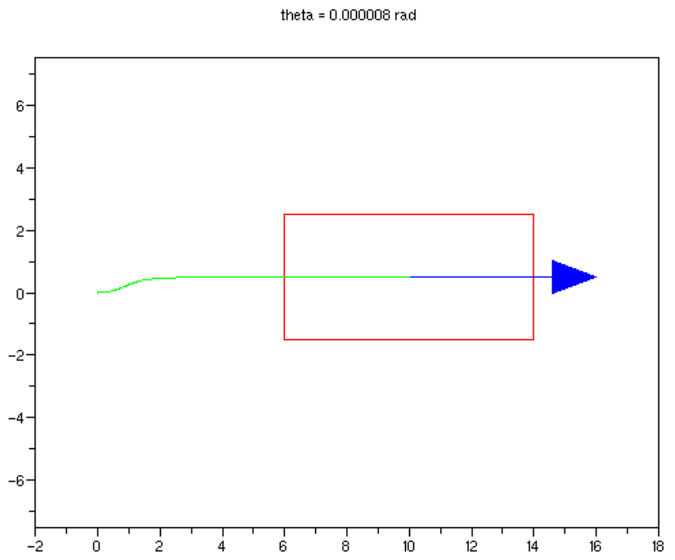

Figure 43: 2D plot of the vehicle behaviour over time

3.3.4 Vehicle Controller Implementation

The discrete controller function implemented on the Brick has been computed from the *zeta-transform* of the Controller transfer function with the following steps, in which for simplicity the coefficients have been labelled:

$$
\begin{aligned}
u(k) &= kc \cdot \frac{(e_1 + e_0 q)}{(u_1 + u_0 q)} \cdot e(k) \\
(u_1 + u_0 q) \cdot u(k) &= kc \cdot (e_1 + e_0 q) \cdot e(k) \\
u_1 u(k) + u_0 u(k) q &= kc \cdot (e_1 e(k) + e_0 e(k) q) \\
u_1 u(k) + u_0 u(k+1) &= kc \cdot (e_1 e(k) + e_0 e(k+1)) \\
u(k+1) &= \frac{kc \cdot (e_1 e(k) + e_0 e(k+1)) - u_1 u(k)}{u_0}
\end{aligned}
$$

Since we can not know the future values of the signal, we then applied a change of variable thus obtaining the equation:

$$u(k) = \frac{kc \cdot (e_1 e(k-1) + e_0 e(k)) - u_1 u(k-1)}{u_0}$$

The actual code of the implementation reassembles the one written for the wheels, with the only important difference that now we don't apply any saturation.

3.4 Performances Tuning and Verification

This section is devoted to the performance tuning of the vehicle, mainly involving the ultrasonic sensor, and a final review of how the vehicle actually performs when confronting against the real world.

3.4.1 Tuning of Ultrasonic Sensor

As it has been outlined in the previous sections, when the vehicle is steering the ultrasonic sensor measurements are affected by an error which is due to the angle of the vehicle moving direction in respect to the wall surface. This problem has been tackled with the aid of a simple triangulation mechanism that appropriately compensates the misreadings.

From time to time the ultrasonic sensor is also affected by spurious saturated measurements that may last even $300ms$, as it is depicted in figure 44.

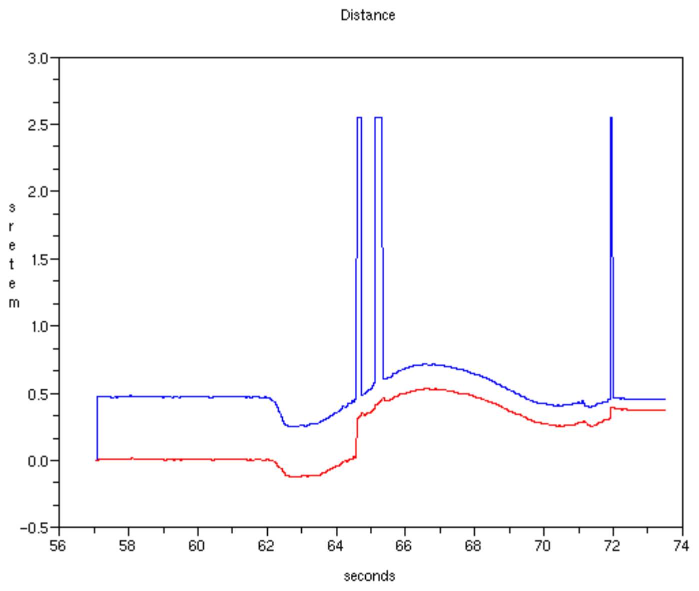

Figure 44: Distance data measured with the ultrasonic sensor (blue) and estimated distance based on feedback speed assuming distance zero as starting position (red)

Clearly the vehicle can not ground its kinematic over these wrong distance estimations, which make it blind to the real external environment for a certain amount of time. To overcome this problem we decided to implement the kinematic model of the vehicle inside the brick itself, as depicted by the following scrap of code:

```
void unicycle_kinematic(time_data* time)
{
    if ((time->current_time_f - time->start_time_f) < (SONAR_T*2.1)) {
        // At the beginning just use the data coming from the sonar
        oxy_rel.y = sonar_data.cur_estimated_distance;
        oxy_rel.theta = sonar_data.cur_alpha;
    } else {
        // Compute Current Vehicle Speed and Angular Speed
        oxy_rel.vehicle_forward_speed = (vehicle_wheel_radius / 2.0) * (
            dataA.feedback_speed + dataB.feedback_speed);
        oxy_rel.vehicle_angular_speed = (vehicle_wheel_radius /
            vehicle_rod_length) * (dataB.feedback_speed - dataA.
            feedback_speed);
        // Compute new position vector
        oxy_rel.x = oxy_rel.x + (cosf(oxy_rel.theta) * oxy_rel.
            vehicle_forward_speed * time->delta_time_f);
        oxy_rel.y = oxy_rel.y + (sinf(oxy_rel.theta) * oxy_rel.
            vehicle_forward_speed * time->delta_time_f);
        oxy_rel.theta = (oxy_rel.vehicle_angular_speed * time->delta_time_f
            );
    }
}
```

Note that the initial conditions of interest of the kinematic model are forcefully loaded from the sonar estimation procedure during the first cycles of activity. Since the digital model depends on data with a certain degree of intrinsic error that accumulates over time, the reference values must be periodically refreshed with the actual data obtained from the ultrasonic sensor. Although this procedure it is not quite sophisticated, it works well and deserves to have its tiny place in this report:

```
void sonar_fun ()
{
    update_timer(&sonar_timer);
    float current_measured_distance = ecrobot_get_sonar_sensor(SONAR_PORT)
        / 100.0; // meters
    // NOTE: theta estimate within the model is particularly BAD, better
        use last known angle
    oxy_rel.theta = sonar_data.cur_alpha;
    // NOTE: Every N samples force the simulation to sync to the sonar data
    if ((((int)(sonar_timer.current_time_f*1000)) / ((int)(SONAR_T*1000)) %
         20) == 0)
    {
        oxy_rel.y = sonar_data.cur_estimated_distance;
    }
    // Load new Values
    sonar_data.prev_measured_distance = sonar_data.cur_measured_distance;
    sonar_data.cur_measured_distance = current_measured_distance;
    if (sonar_timer.delta_time_f > 0.0) {
        if (sonar_data.cur_measured_distance < 2.40) { // use sonar value
            // Triangulation
            float delta_space = vehicle_target_speed * sonar_timer.
                delta_time_f; // meters
            sonar_data.cur_error_distance = sonar_data.
                cur_measured_distance - sonar_data.prev_measured_distance;
            sonar_data.cur_alpha = atanf(sonar_data.cur_error_distance /
                delta_space); // rad
            sonar_data.cur_trigon_distance = (float) (sonar_data.
                cur_measured_distance * cosf(fabsf(sonar_data.cur_alpha)));
            // Prepare for Filtering
            mem_dist_sonar[mem_sonar_index] = sonar_data.
                cur_trigon_distance;
            mem_time_sonar[mem_sonar_index] = sonar_timer.delta_time_f;
            mem_sonar_index = (mem_sonar_index + 1) % MEM_SONAR_LENGTH;
            // Apply Filtering
            sonar_data.cur_filtered_distance = moving_average_vector(
                mem_dist_sonar, mem_time_sonar, MEM_SONAR_LENGTH);
            // Choose one value
            sonar_data.cur_estimated_distance = sonar_data.
                cur_filtered_distance;
            // Update the Unicycle Kinematic Model
            unicycle_kinematic(&sonar_timer);
        } else { // Use simulation instead of sonar
            // Last computed distance should be reliable more or less
            oxy_rel.y = sonar_data.cur_estimated_distance;
            // Apply Unicycle Kinematic to predict distance according to
                model
            unicycle_kinematic(&sonar_timer);
            sonar_data.cur_estimated_distance = oxy_rel.y;
            sonar_data.cur_measured_distance = oxy_rel.y; // WARNING:
                approximation with tiny error
        }
    } else {
        sonar_data.cur_estimated_distance = current_measured_distance;
    }
}
```

The procedure starts with an initialization phase that updates the values of the timer associated to the sonar, takes the sonar distance and periodically synchronizes the kinematic model with the measured distance. If the ultrasonic sensor does not return potentially wrong distance measurements, we correct the distance estimates using the triangulation rules and apply a moving average filter of size 5. Otherwise we forcefully update the kinematic model with the last good estimate of the distance and use it to compute the new distance.

In figure 3.4.3 are shown the various (intermediate) distance estimations available to the robot collected during one of many experiments. The *blue* line stands for the actual measurements collected with the sonar, the *red* line results from the triangulation procedure while the *green* one results from the filtering. In this plot its not easy to spot when the sonar misread the distance, since in such a case the measured values are always overridden using the kinematic model that is the *black* line. As you can see the triangulation introduces a slight oscillation in its output values that is probably an error due to the usage of trigonometric functions, while from time to time introduces an instantaneously significant error that is by the way filtered out by the moving average.

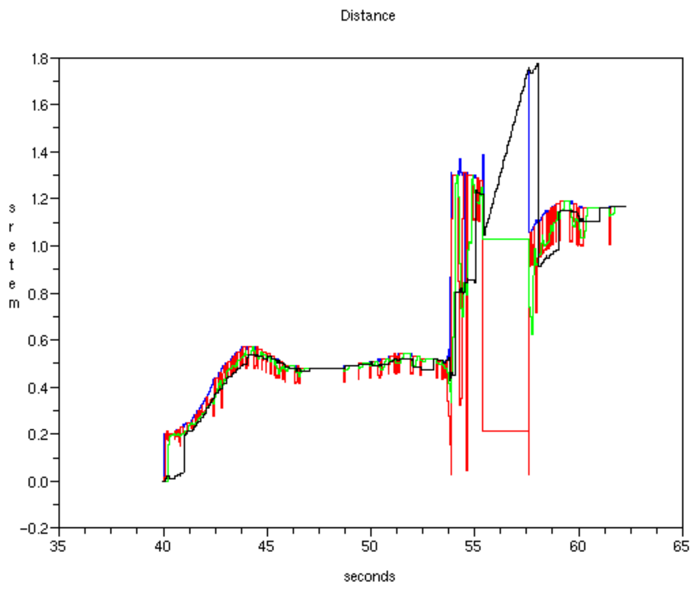

Figure 45:

Near the time of $54s$ the vehicle incurred into the end of the wall, hence it initially measured the new distance but as time went by it lost the wall reference for a few seconds due to the particular angle that the vehicle had in respect to the wall, which caused the ultrasonic wave to not be reflected back. Luckily the vehicle controller can recover from this particular situation by steering toward the reference wall. If the wall is actually there and the vehicle did not turn too much, then after some time it will follow the path after the corner, otherwise it will end up looping in a circle. The latter situation is the worst case example and can not be theoretically avoided with the limited instruments and sensors we have been provided for this project.

3.4.2 Turbo Boost

Since our vehicle is configured to proceed at a relatively modest speed, which is $0.337 m/s$, we decided to provide it with a *Turbo Boost* utility. This utility increases the general speed of the vehicle when it is following the wall at the right distance and with a nearly zero angle. This task is accomplished by the following simple piece of code:

```
// Turbo Boost Facility
if ((fabsf(sonar_data.cur_alpha) < 0.01)  && (fabsf(sonar_data.
    cur_estimated_distance - WALL_DISTANCE) < 0.035))
{
     vehicle_target_power = 95; // %
} else {
     vehicle_target_power = TARGET_POWER; // 80%
}
vehicle_target_speed = ((k * input_power * vehicle_target_power *
    vehicle_wheel_radius) / 100.0);
```

3.4.3 Overall Observations

When the vehicle starts up to $0.5m$ far away from the desired path but parallel to the wall, as it was the case for the measurement data shown in figure , it is able to reach the desired distance from the wall within $5-6s$. These performances are acceptable both in respect to the outcome of the digital simulation in *Scicos* and to our initial requirements.

The vehicle can also start with an initial angle that is up to 45 degrees in respect to the wall, in which case it obviously requires more time to converge. In our experimentations the vehicle performed better when it was initially placed near the wall rather than far from it, probably due to the fact that in the latter case the ultrasonic measure uncertainty connected to the angle in respect to the wall is bigger.

Although the vehicle immediately reacts to obstacles placed in front of the wall, their length must be of at least 1.5 meters in order to let the vehicle approach the new reference path given the high speed it is moving forward.

4 Conclusions

This laboratory experience provided us with a simple, but yet complete, methodical procedure to develop a functioning robot. Starting from the instruments that enabled us to model an actuator behaviour using standard mathematical notions, we were then given the basic notions needed to develop an controller with a feedback loop that guarantees the actuator matches the desired performances of the system. At last we developed further on these building blocks by studying how a kinematic behaviour can be formalized into a model and then linearised through a sequence of simplifications. At this point all the building blocks could be finally glued together to “magically” result in a fully functional Unicycle Robot that matches all our expectations.

This experience has been particularly satisfactory and enjoying in all its phases, although it has been displeasure that during the lectures there wasn’t enough time to dedicate into investigating further and more deeply more advanced techniques of the field. From our perspective the part that up to now is more affected by the lack of advanced techniques is the spatial localization of the vehicle, for which we had to apply some ad-hock hacks.

One of the main problem that we found during the carrying out of the assignment is the instability of *Sicoslab* on OS different than Linux. In our prospective using a different program can be easier and allow a faster execution of the exercise.