UNIVERSITY OF TRENTO - Italy

Information Engineering and Computer Science Department

Laboratory of Applied Robotics

Modelling and simulation of Lego Mindstorms Motors

Elena Tumanov, Antonio Sciarretta, Davide Todeschi

January 20, 2016

1 Introduction

1.1 Project goals and steps of implementation

We had to implement Modelling and simulation of Lego Mindstorms Motors in 3 steps:

1. Identification of parameters and control of SISO systems for Lego Motors;
2. Control Design and Digital Implementation of the controller;
3. Modelling and simulation of the larger system, application on the comprehensive model and implementation of digital controller for the whole system.

1.2 General information about LEGO MINDSTORMS Education

LEGO MINDSTORMS Education is the next generation in educational robotics, enabling students to discover Science, Technology, Engineering and Mathematics in a fun, engaging, hands-on way. By combining the power of the LEGO building system with the LEGO MINDSTORMS Education technology, teams of students can design, build, program, and test robots. Working together on guided and open-ended engineering projects, the team members develop creativity and problem-solving skills along with other important mathematics and science knowledge. Students also become more skilled in communication, organization and research, which helps prepare them for future success in higher levels of schooling and in the workplace.
LEGO MINDSTORMS Education features an advanced 32-bit computer controlled NXT brick, Interactive Servo Motors, Sound, Ultrasonic and other sensors, Bluetooth communication and multiple downloading capabilities. The icon-based LEGO MINDSTORMS Education NXT Software is built on the LabVIEW™ software from National Instruments, an industry standard with applications in many engineering and research fields. [1].

[1] Lego Mindstorms NXT - Wikipedia, the free encyclopedia, `https://en.wikipedia.org/wiki/Lego_Mindstorms_NXT`

2 Used tools

2.1 nxtOSEK

nxtOSEK is an open source platform for LEGO MINDSTORMS NXT. nxtOSEK consists of device driver of leJOS NXJ C/Assembly source code,TOPPERS/ATK (Automotive Kernel, formerly known as TOPPERS/OSEK) and TOPPERS/JSP Real-Time Operating System source code that includes ARM7 (ATMEL AT91SAM7S256) specific porting part, and glue code to make them work together.
item nxtOSEK can provide:

- ANSI C/C++ programming environment by using GCC tool chain;
- C API for NXT Sensors, Motor, and other devices;
- C++ API for NXT Sensors and Motor which include many third party sensors;
- TOPPERS/ATK provided real-time multitasking features proven in automotive industry;
- TOPPERS/JSP provided real-time multitasking features complied with Japan original open RTOS specification ITRON 4.0;
- Fast execution and less memory consumption (nxtOSEK program is executed natively on the ARM7 and nxtOSEK itself consumed totally just about 10Kbytes). [2]

2.2 ScicosLab

ScicosLab is a free open-source software package for scientific computation. ScicosLab includes a fork of Scilab, based on Scilab 4, the modeling and simulation tool Scicos and a number of other toolboxes.
Scilab is an interpreted language specifically developed for matrix based numerical computations. It includes hundreds of general purpose and specialized functions for numerical computation, organized in libraries called toolboxes that cover such areas as simulation, optimization, systems and control, and signal processing. These functions reduce considerably the burden of programming for scientific applications. [3]

[2]nxtOSEK, `http://lejos-osek.sourceforge.net/whatislejososek.htm`

[3]Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4, Stephen L. Campbell, Jean-Philippe Chancelier and Ramine Nikoukhah, Second Edition

3 Collecting data from motor

We used Bluetooth connection to collect data from motor.

3.1 NXT Brofist directory changes

At the beginning, we set command Set Power to return the current values of the counter from the motor (degrees). In the server part, we sent an input step function and after the power, which was 50, we send more than 1000 commands for multiple times (from power 30 to 70).
Because Bluetooth connection is very slow, we chose the following way:

- We created a buffer in the client part, which has the capacity to store 1000 packets/commands (Set power commands).
- Stored simultaneously all set power commands and after elaborated all at once.
- Between the execution of two successive commands we set a wait time of 2 ms.
- After the execution, we sent back all the results.
- Set power instructions returned the current values.

We modified the packets to add the information about the time.

3.2 Collecting data through Bluetooth

Client side modifications:
Added Buffer to store a huge number of instructions. This let us to use Bluetooth only twice per experiment.
Implemented the interface to return also the current measure NXT timestamp.
Modified Set power commands, to return information about current counter and time.
Server side modifications:
Implement the interface to include also the current measure NXT timestamp.
Created Set power commands, which gave a step function with Power 50.

1. Send message from PC via Bluetooth to brick that define motor power(that determines the speed).
2. Waited for the elaboration.

3. Receive message from brick with time, input power, tachometer count.
4. Elaborated the speed.

$$Speed = \frac{\text{Count-PrevCount}}{\text{Time–PrevTime}} * \frac{\pi}{180} \tag{1}$$

5. We saved data in .csv files that contains information received from the brick. We collected the following information: time, input power, tachometer count, speed(1).

Code is here: Collect Data Implementation
Collected data files are here: Files with Data

4 Filtering data and estimating the parameters

To estimate the parameters we filtered the data using Chebyshev 2 filter and after this we estimated the parameters using a regular method proposed during the course.

4.1 Filtering

4.1.1 Chebyshev 2 filter

We used Chebyshev 2 filter of order 1 and a cut-off frequency of 0.04. In the following fig. 1., you can see the comparison between Raw Data and Filtered Data for the Power 40.

4.1.2 Chebyshev instead of Butterworth

Chebyshev equal ripple magnitude have a better rate of attenuation beyond the pass-band than Butterworth but is considerably more ringing in step response than Butterworth. This filter response has the steeper initial rate of attenuation beyond the cutoff frequency than Butterworth.This advantage comes at the penalty of amplitude variation(ripple) in the pass-band. Unlike Butterworth and Bessel response, which have 3dB attenuation at the cutoff frequency,Cebyshev cutoff frequency is defined as the frequency at which the response falls below the ripple band. For even-order filters, all riple is above the dc-normalized passband gain response, so cutoff is at 0dB. For odd-order filters, all riple is below the dc-normalized passband gain response, so cutoff is at -(ripple) dB. For a given number of poles, a steeper cutoff can be achieved by allowing more pass-band

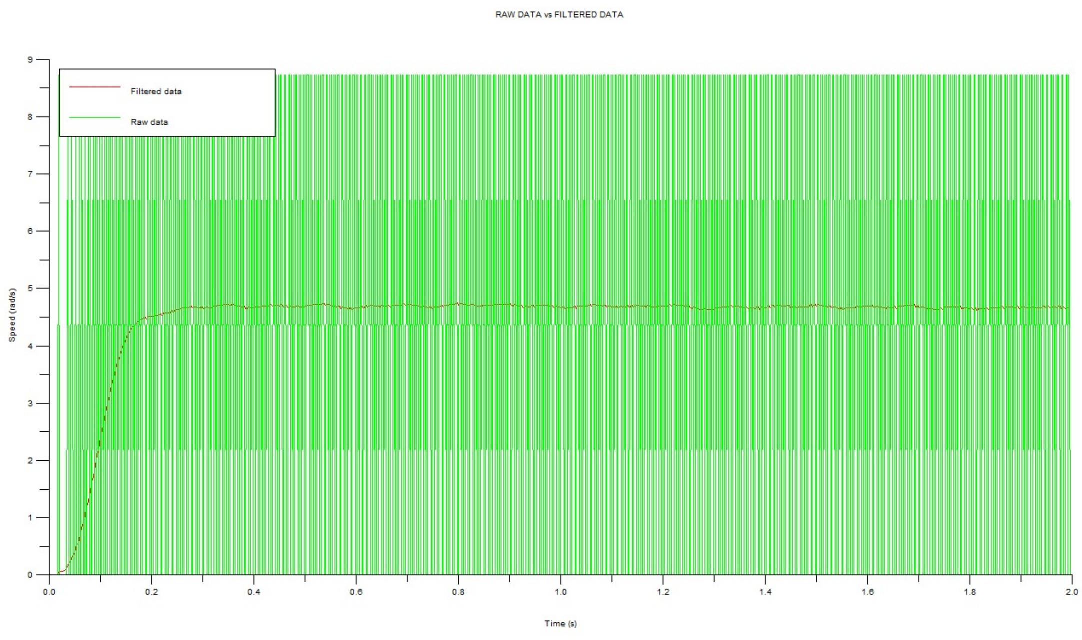


Figure 1: Raw Data and Filtered Data at Power 40

ripple. The Chebyshev has more ringing in its pulse response than the Butterworth - especially for high-ripple designs.
Butterworth filter have some overshoot and ringing in step response. This filter has the flattest possible pass-band magnitude response. Attenuation is -3dB at the design cutoff frequency. Attenuation beyond the cutoff frequency is a moderately steep -20dB/decade/pole. The pulse response of the Butterworth filter has moderate overshoot and ringing.
The Butterworth filter is completely defined mathematically by 2 parameters: Cutoff frequency and number of poles. The Chebyshev filter has a third parameter: Passband Ripple.

4.2 Parameter Estimation

We need to estimate 3 parameters q, ω_n, ξ. You can find code in Deliverable 1

We used the following formulas:

$$q = \text{Last speed value} \tag{2}$$

$$\xi = \sqrt{\frac{\log(\text{overshot})^2}{\pi^2 + \log(\text{overshot})^2}} \tag{3}$$

$$\omega_n = \frac{\log(\frac{alpha}{100}) - \log(\frac{1}{\sqrt{1-\xi^2}})}{-\text{settling time} * \xi} \tag{4}$$

We started evaluation from Steady state values of the filtered functions (f - filtered function).

1. Steady state value estimation = k_{est};

   $$k_{est} = \frac{\text{f}}{inputsignal} \tag{5}$$

   We evaluated the damping factor ξ, called ξest.

   $$\xi_{est} = \sqrt{\frac{\log(\text{overshot})^2}{\pi^2 + \log(\text{overshot})^2}} \tag{6}$$

2. Natural frequency ω_n which is called ω_{nest}.

   $$\omega_{nest} = \frac{\log(\frac{alpha}{100}) - \log(\frac{1}{\sqrt{1-\xi^2}})}{-\text{settling time} * \xi_{est}} \tag{7}$$

   Where: alpha= 3;

3. Made the estimations of our system using the following estimation formula:

   $$G_{est} = \frac{.k_{est}}{\frac{s^2}{\omega_n^2} + 2 * \frac{\xi_{est}}{\omega_{nest}} * s + 1} \tag{8}$$

   We evaluated all previous values for 40 times from Power 30 to Power 70 and we made the mean of the values.

   Result values are:

   $$k_{est} = 0.1531618 (Steadystatevalue) \tag{9}$$

   $$\xi_{est} = 0.8209478 (Dampingfactor) \tag{10}$$

   $$\omega_{nest} = 16.515118 (Naturalfrequency) \tag{11}$$

   In the following fig. 2., you can see the difference between Estimation and Test for Power of 40, 50 and 60.

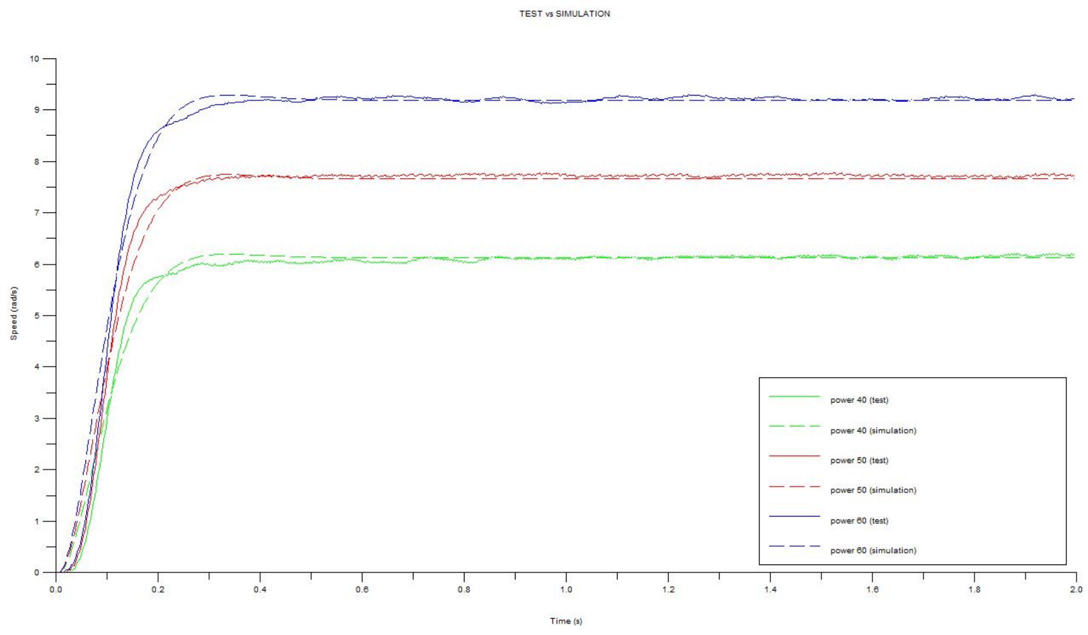


Figure 2: Test vs Simulation (Power 40, 50, 60)

4.3 Performance indices to control system

A metric is needed to measure distances between observed and predicted data and, hence, to assess model fit. As a metric, we adopted some standard indices used as performance indices for control systems.
The indices formulas:

Integral squared error (ISE) = $\int_0^T (y_m(t) - y(t))^2 dt$;

Integral absolute error (IAE) = $\int_0^T |y_m(t) - y(t)| dt$;

Integral time squared error (ITSE) = $\int_0^T t(y_m(t) - y(t))^2 dt$;

Integral time absolute error (ITAE) = $\int_0^T t|y_m(t) - y(t)| dt$.

We tested our system three times with Power 40, Power 50 and Power 60.

Got the following errors calculating the average value of the above formulas:
Power 40: AverageError = 38.2590
Power 50: AverageError = 53.2616
Power 60: AverageError = 45.9473

Mean squared error:
Power 40: MSE = 0.0259 (rad/s)^2
$Power50 : MSE = 0.0324(rad/s)^2$
$Power60 : MSE = 0.0408(rad/s)^2$

Used filter caused the big error at the beginning of the axis in the following figure because it have a big delay.
In the following fig. 3., you can see the difference between Output Estimations and Input Estimations for Power of 40, 50 and 60.

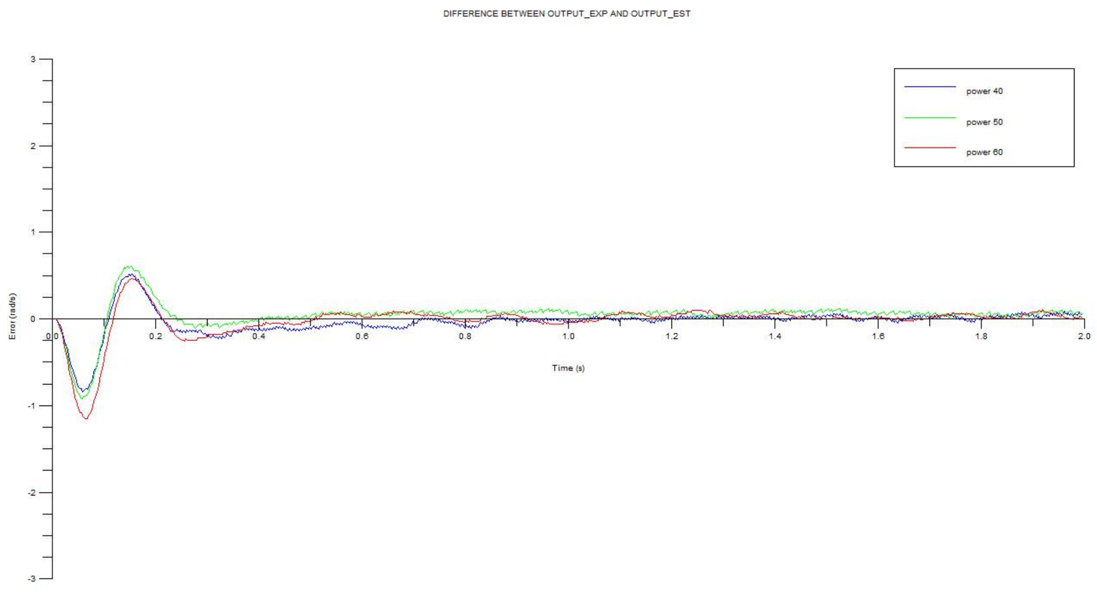


Figure 3: Difference between Input and Output estimations

5 General description of Control Design

Control Design and Digital implementation of controller for the Lego NXT motor. The second task was to show our designed controller, describe it properties and digital implementation.

5.1 General information about Control design for the motor

Root locus: In control theory and stability theory, root locus analysis is a graphical method for examining how the roots of a system change with variation of a certain system parameter, commonly a gain within a feedback system. This is a technique used as a stability criterion in the field of control systems developed by Walter R. Evans which can determine stability of the system. The root locus plots the poles of the closed loop transfer function in the complex S plane as a function of a gain parameter (see pole–zero plot). [4]

Closed-loop transfer function: in control theory is a mathematical expression (algorithm) describing the net result of the effects of a closed (feedback) loop on the input signal to the circuits enclosed by the loop. [5] [6]

In such a case, we have the plant P(s) and an additional block C(s): the controller. The objective was to determine C(s) in order to satisfy certain closed loop performance.

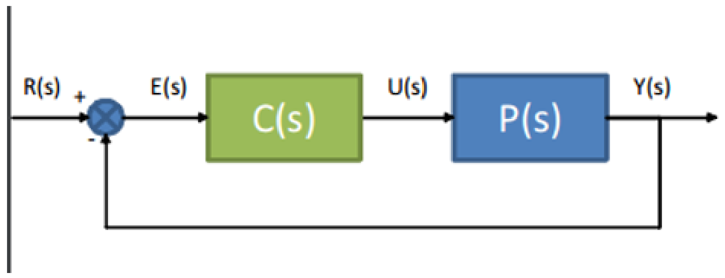


Figure 4: Closed loop

6 Control design for the motor

6.1 Needed performance

- Steady state tracking error = 0

[4]Root Locus,https://en.wikipedia.org/wiki/Root_locus

[5]Closed loop transfer function,https://en.wikipedia.org/wiki/Closed-loop_transfer_function

[6]Root locus course,http://disi.unitn.it/~palopoli/courses/ECL/RootLocus.pdf

- Overshot $< 20\%$
- Settling time < 0.2s

Overshot requirement on root locus plot is shown by the following formula:

$$\frac{-\pi\xi}{\sqrt{1-\xi^2}} <= \frac{log^2 0.2}{\pi} = \alpha \tag{12}$$

6.1.1 Closed loop system

(a) Closed loop system:

$$Gcl = \frac{K_c * C * G}{1 + K_c * C * G} \tag{13}$$

(b) Set

$$K_c = 1; C = 1 \tag{14}$$

(c) Added a pole

$$C = 1/s \tag{15}$$

to obtain 0 steady state error, possible due to Internal model principle.

6.2 Our Motor Design

$$C(s) = \frac{(s+10)^2}{s*(s+31)} \tag{16}$$

$$K_c = 12 \tag{17}$$

Root locus is illustrated in fig. 5, and the response to the step function in fig. 6. Results of Scicoslab simulation fig. 7 are shown in the following figures:

- Ω in fig. 8
- Power in fig. 9

Code is in this directory.[7].

[7]`https://github.com/etumanov/AppliedRoboticsUNITN/tree/master/DeliverableNII`

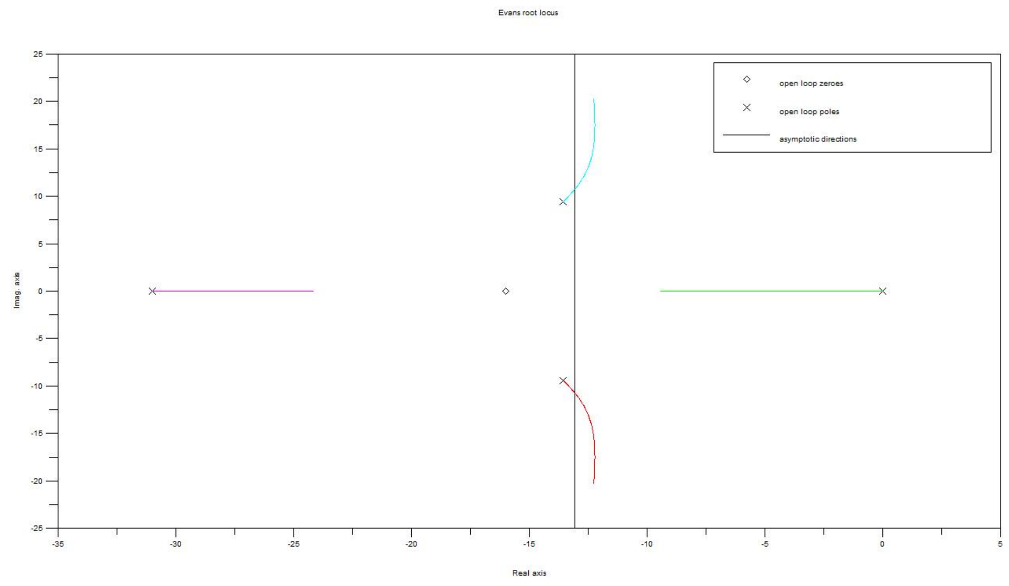


Figure 5: Root locus

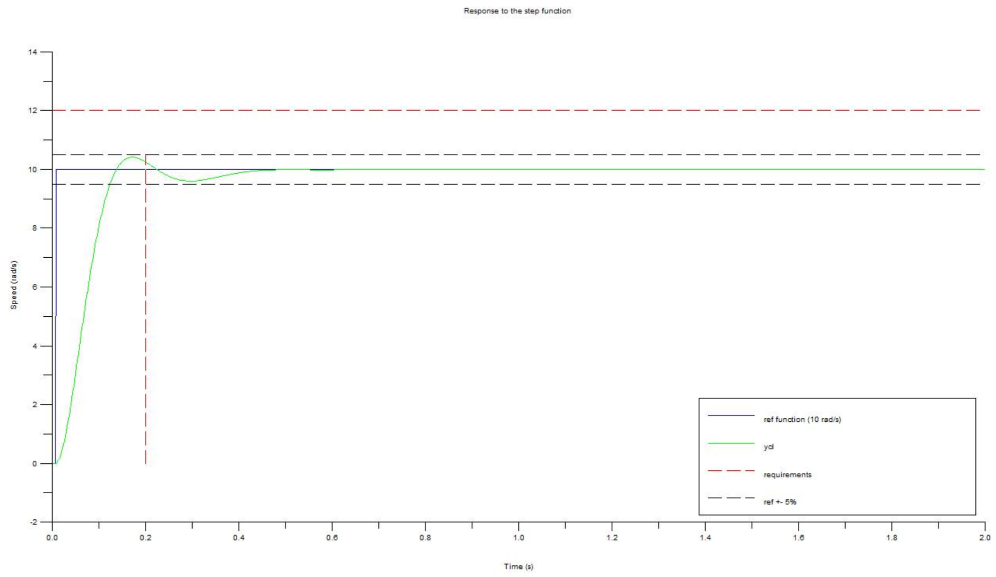


Figure 6: Response to the step function

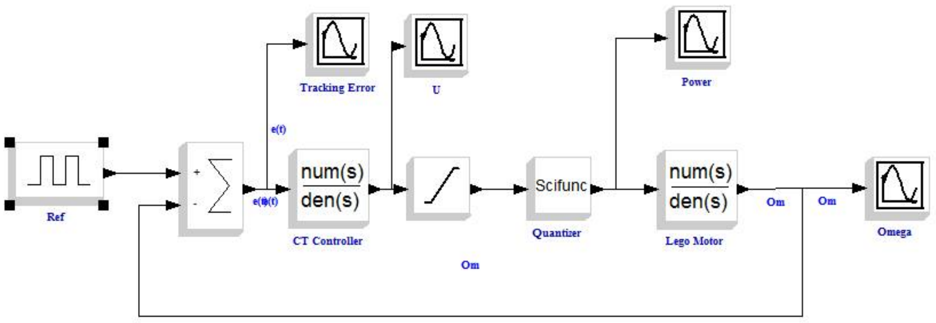


Figure 7: Controller Design

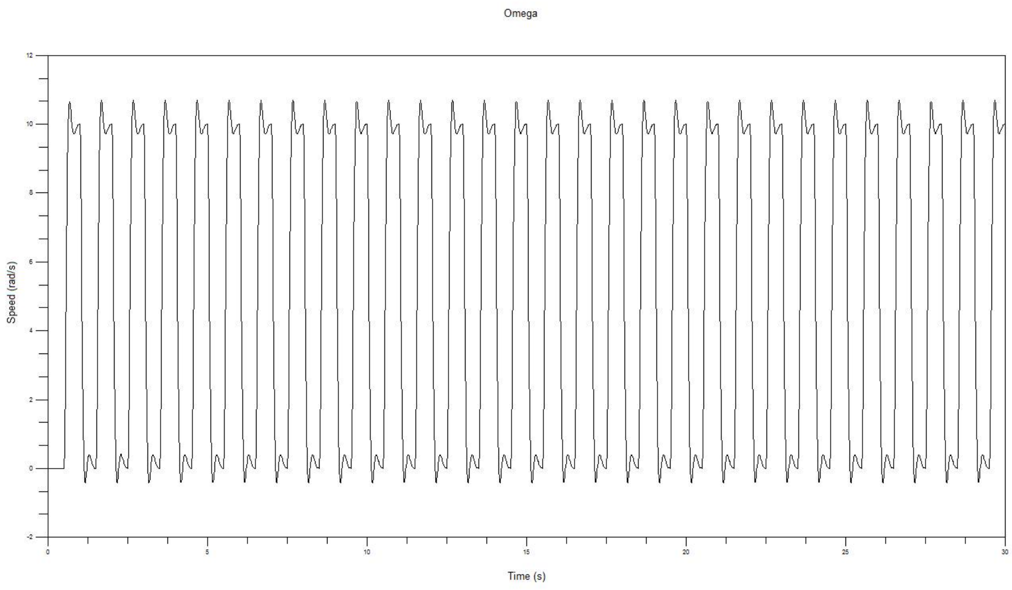


Figure 8: Speed Ω

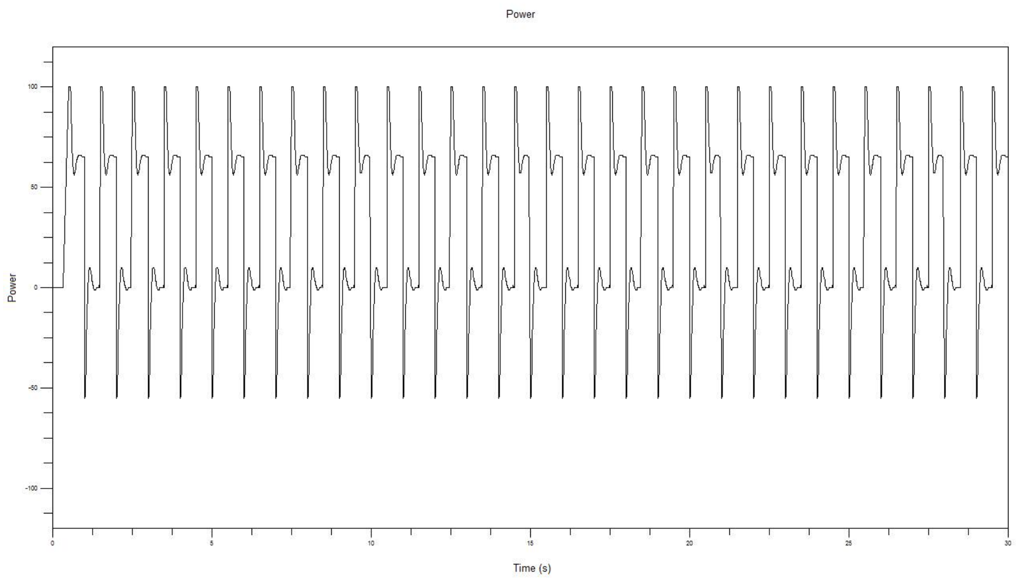


Figure 9: Power

7 Digital implementation of the controller

What we have done is fig. 10:

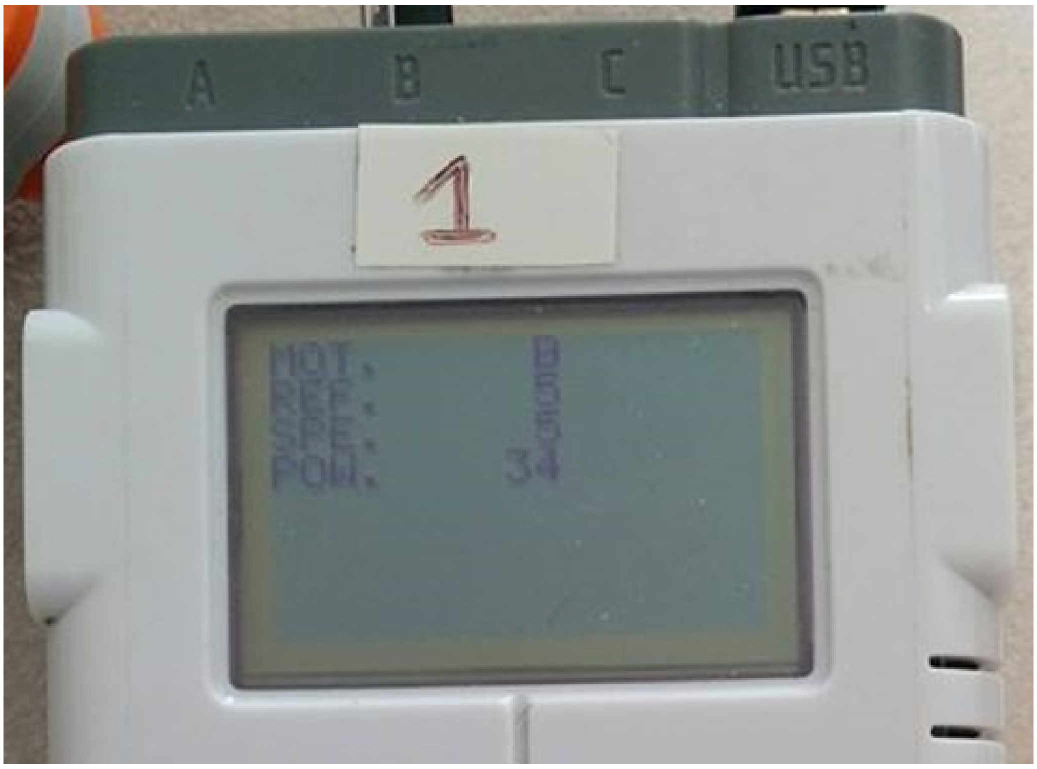
- Set a speed we want to reach (REF)
- Evaluated the current speed (SPE)
- Calculated the power we need (POW)

$$y_{k+2} = \frac{1}{-31 * sT - 2}(-4*y1+(-31*sT+2)*y0+gain*(u2*(-128*sT*sT-32*sT-2) + u1*(4-256*sT*sT) + u0*(-128*sT*sT+32*sT-2))) \quad (18)$$

Speed estimated using Exponential moving average: $S_1 = Y_1, t > 1$
$S_t = \alpha \cdot Y_t + (1-\alpha) \cdot S_{t-1}$(19)

Code is available in a shared folder[8].

[8]https://github.com/etumanov/AppliedRoboticsUNITN/tree/master/DeliverableNII



```
MOT.      B
REF.      5
SPE.      5
POW.     34
```

Figure 10: Brick with REF, SPE and POW values

8 Modelling and simulation of the larger system

Our last task was to Model and simulate a larger system:

1. 2 motors;
2. Moving straight(no deviations) without any visual or position control, but only using the the implementation of a digital controller for two motors;
3. Implementing an additional controller which is evaluating the error of the speed from the first and second motor in order to adjust them;
4. Test on the Lego Mindstorms Robot for 3 different distances and 3 different speeds.

Our actions:

1. Modified the packets which contain the information about the distance;
2. In the client we modified the command set speed;
3. Duplicating the command and controller to the both the motors;
4. Add the control on the distance (for the distance you can add the formula, the one in the code);
5. Add the controller on the angle omega which has to be 0.

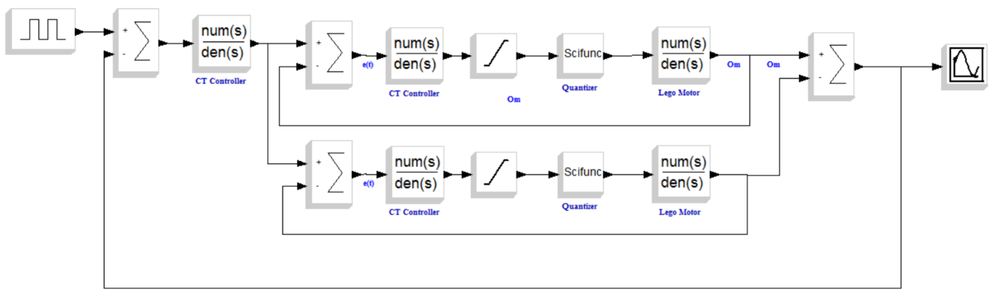


Figure 11: Complete System

To evaluate speed of each motor we used this function:

```
double evaluate_speed(motor_t * motor, double previousSpeed) {
  double space_rad = (motor->revolutions[0] - motor->revolutions[1]) * (PI / 180.0);
  double speed_rad = space_rad/ (0.002);
  speed_rad = speed_rad * alpha + (1-alpha) * previousSpeed;
  return speed_rad;
```

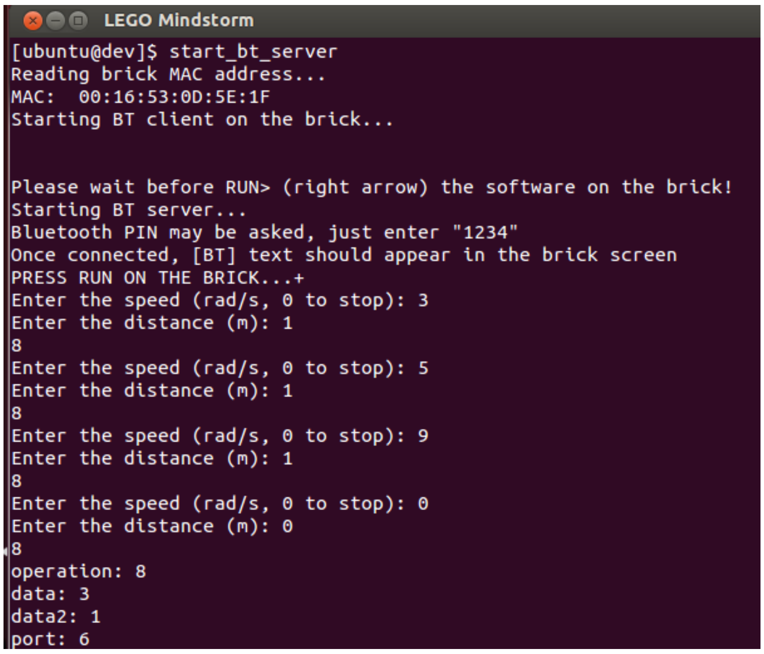
To control the speed from the both motors we implemented next function:

```c
void updateMotors(engines_t * motors){
  motor_t *motor;
    motor_t *otherMotor;
  for(int m = 0; m < 2; m++){
    switch (m){
      case 0: motor = &(motors->first); otherMotor = &(motors->second); break;
      case 1: motor = &(motors->second); otherMotor = &(motors->first); break;
    }
    for (int i = SAMP_NUM; i > 0; i--){
      motor->revolutions[i] = motor->revolutions[i-1];
      motor->speeds[i] = motor->speeds[i-1];
    }
    motor->revolutions[0] = nxt_motor_get_count(motor->port);
    motor->times[0] = ecrobot_get_systick_ms();
    motor->speeds[0] = evaluate_speed(motor, motor->speeds[1]);

    if(motor->speed_control_type == NOT_USING)
      continue;

    if(motor->speed_control_type == PID_CONTROLLED){

      double distance = (((motor->revolutions[0] + otherMotor->revolutions[0])/2.0) * PI/180) * 0.028;

      if(abs(distance) >= motor->distance_ref[0]){
        for (int i = 0; i < BUFFER_SIZE - 1; i++){
          motor->speed_ref[i] = motor->speed_ref[i+1];
          motor->distance_ref[i] = motor->distance_ref[i+1];
          otherMotor->speed_ref[i] = otherMotor->speed_ref[i+1];
          otherMotor->distance_ref[i] = otherMotor->distance_ref[i+1];
        }
        motor->speed_ref[BUFFER_SIZE-1] = 0;
        motor->distance_ref[BUFFER_SIZE-1] = 0;
        otherMotor->speed_ref[BUFFER_SIZE-1] = 0;
        otherMotor->distance_ref[BUFFER_SIZE-1] = 0;
      }

      if(distance < motor->distance_ref[0]){
        double error = motor->speed_ref[0] - motor->speeds[0];
        double rotationError = rotationController(motors);

        if(m == 0) {
          error = error - rotationError/2;
```

```
            motor->powers[0] = controller(error);

         }
         if(m == 1) {
            error = error + rotationError/2;
            motor->powers[0] = controller2(error);
         }

      }
   }

   nxt_motor_set_speed(motor->port, motor->powers[0], 1);
  }
}
```

Full implimentation of the system can see in [9].

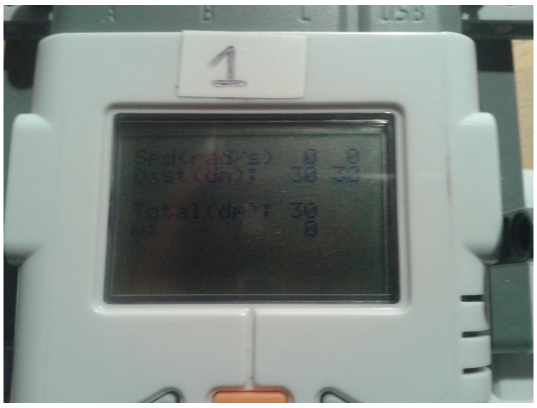
Bellow is the result from one test:



```
LEGO Mindstorm
[ubuntu@dev]$ start_bt_server
Reading brick MAC address...
MAC:  00:16:53:0D:5E:1F
Starting BT client on the brick...


Please wait before RUN> (right arrow) the software on the brick!
Starting BT server...
Bluetooth PIN may be asked, just enter "1234"
Once connected, [BT] text should appear in the brick screen
PRESS RUN ON THE BRICK...+
Enter the speed (rad/s, 0 to stop): 3
Enter the distance (m): 1
8
Enter the speed (rad/s, 0 to stop): 5
Enter the distance (m): 1
8
Enter the speed (rad/s, 0 to stop): 9
Enter the distance (m): 1
8
Enter the speed (rad/s, 0 to stop): 0
Enter the distance (m): 0
8
operation: 8
data: 3
data2: 1
port: 6
```

Figure 12: Introduced Speeds and Distances from terminal

[9] `https://github.com/etumanov/AppliedRoboticsUNITN/tree/master/DeliverableNIII`



```
Spd(rad/s)   0  0
Dist(dm):   30 30

Total(dm): 30
w:          0
```

Figure 13: Results on the brick, w = 0 as we expected

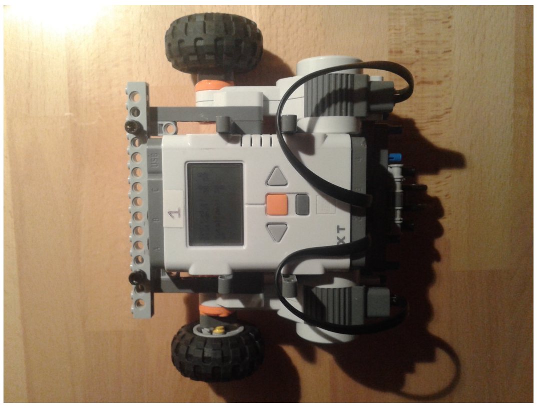


Figure 14: Our Robot Construction

9 Conclusions

While working on this project, we applied a lot of knowledge learned at the courses of Laboratory of Applied Robotics. Furthermore, we gained a lot of experience in programming an embedded system. It was very interesting and exciting to implement a real Robot and see the expected results in simulation and SciLab Graphs.
The scope of this project was to get familiarity with the overall process of design a complete robotic system.
We have subdivide our work in 3 steps: identification phase, controller design and system realization.
Above we illustrated our method of identification of parameters from Lego Mindstorms motors based on the collected data. In this phase, we had to solve a lot of different problems; some of them were more technical problems, like understand-

ing and working with NXT C API, bluetooth slowness, communication protocol definition and so on, while others were more theoretical, like model definition, data filtering, estimates evaluation, etc.
After some processing, it was possible to compare the predicted behavior of the system with our observation.
For the second phase, our goal was to design a controller for the motor and to realize a digital implementation, given some requirements. Here we faced some trouble due to errors in the previous phase: we didn't convert properly some data's units of measurement, therefore we had to recompute all values from the first step.
With the theory provided by the study of the root locus, we developed a system which matches the requirements we had. Above we showed how our implementation is properly working.
Our last deliverable was to present a complete system which is a tricycle, moving straight. Our system is composed by two independent motors that move as many wheels, and a third free wheel for balancing. In order to implement this we made a stable construction with 2 wheels in front and the third one in the back. We realized two controllers, one for each independent motor, and a third to control the direction. The last one is simply verifying that the overall speed of the two motors is the same (this is the big condition to move straight.
This is all the work we had to do in order to realize our simple robot system. Thanks to this project, now we are more conscious about the work that is behind such systems and we have some good intuitions about how to realize them.