Simulation of N DC motors controlled in position, velocity and torque by a PID regulator

Michele Galullo, Yanet Vila

July 11, 2016

Course:	Real Time System
Professor:	G. Buttazzo

Contents

List of Figures

1 Introduction

In this document we present a simulation of N DC motors controlled in position, velocity and torque by PID regulator. We design a simple interface to provide set points of motors parameters and PID gains during the simulation run. We consider two frames on the simulations, one show the motor rotation, the second design the response of controlled variable (position, velocity and torque) as function of time. A demo was written to show three motor, where each one of them has a different control.
The project can be split in three objective:

First Objective
The design of algorithms to compute the control in position, velocity and torque by PID regulator;

Second Objective
The design for graphical interface to display the buttons interaction and controlled variables;

Third Objective The implementation a thread structure to simulate in real-time each motor;

In the following sections the motor modelling and software details will be explained.

2 Algorithms for control

2.1 Motor modelling

The basic principle of DC motor is the energy conversion that is produced when a current carrying conductor is placed in a magnetic field, then it experiences a torque and has a tendency to move. The electric equivalent circuit of the armature and the free-body diagram of the rotor are shown in the following figure:

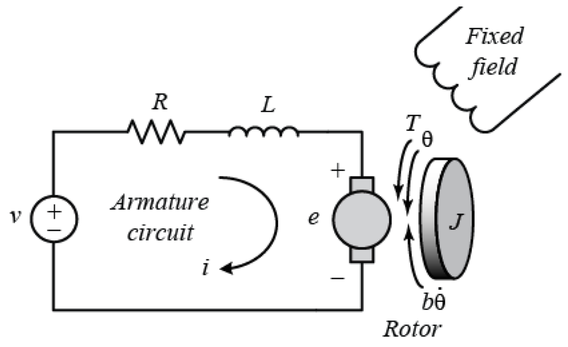


Figure 1: The electric equivalent circuit of the armature

The equations that we use for describe the motor are:

$$\begin{cases} V = Ri(t) + L\frac{di(t)}{dt} + e_b(t) \\ \\ \tau_m = J\ddot{\theta} + b\dot{\theta} + \tau_c \end{cases} \tag{1}$$

and the physical parameters that we need are:

- $J\ [kg \cdot m^2]$ Moment of inertia of the rotor.
- $b\ [\frac{kg}{s \cdot m}]$ Motor viscous friction constant.
- $Ke\ [\frac{V \cdot sec}{rad}]$ Electromotive force constant.
- $Kt\ [\frac{N \cdot m}{A}]$ Motor torque constant.
- $R\ [\Omega]$ Electric resistance.
- $L\ [H]$ Electric inductance.

According output that we chose, we obtained the following transfer functions in continuos-time for position, speed and torque:

$$\begin{cases} G_p(s) = \frac{Kt}{s((Js+b)(Ls+R)+KeKt)} \\ \\ G_v(s) = \frac{Kt}{(Js+b)(Ls+R)+KeKt} \\ \\ G_T(s) = \frac{Kt(Js+b)}{(Js+b)(Ls+R)+KeKt} \end{cases} \tag{2}$$

2.2 PID modelling

For the controller we used a simpler PID *Proportional-Integral-derivative* and the transfer function that we used is:

$$PID(s) = K_p + K_i \frac{1}{s} + K_d \cdot s \tag{3}$$

2.3 Discrete-time

For the simulation we convert the closed-loop transfer functions $Cl_i = \frac{PID(s) \cdot G_i(s)}{1+PID(s) \cdot G_i(s)}$ for each type of control in discrete-time using Tustin method we obtained the following digital equations:

$$\begin{aligned} Cl_p(z) = {} & n_{21}U(z) + n_{22}U(z)z^{-1} + n_{23}U(z)z^{-2} + n_{24}U(z)z^{-3} + \\ & n_{25}U(z)z^{-4} + n_{26}U(z)z^{-5} - d_{22}Y(z)z^{-1} - d_{23}Y(z)z^{-2} - \\ & d_{24}Y(z)z^{-3} - d_{25}Y(z)z^{-4} - d_{26}Y(z)z^{-5} \end{aligned} \tag{4}$$

$$\begin{aligned} Cl_v(z) = {} & n_{11}U(z) + n_{12}U(z)z^{-1} + n_{13}U(z)z^{-2} + \\ & n_{14}U(z)z^{-3} + n_{15}U(z)z^{-4} - d_{12}Y(z)z^{-1} - \\ & d_{13}Y(z)z^{-2} - d_{14}Y(z)z^{-3} - d_{15}Y(z)z^{-4} \end{aligned} \tag{5}$$

$$\begin{aligned} Cl_T(z) = {} & n_{31}U(z) + n_{32}U(z)z^{-1} + n_{33}U(z)z^{-2} + n_{34}U(z)z^{-3} + \\ & n_{35}U(z)z^{-4} + n_{36}U(z)z^{-5} - d_{32}Y(z)z^{-1} - d_{33}Y(z)z^{-2} - \\ & d_{34}Y(z)z^{-3} - d_{35}Y(z)z^{-4} - d_{36}Y(z)z^{-5} \end{aligned} \tag{6}$$

3 Software Architecture

The project is developed with *Linux Ubuntu* operative system and we used *allegro4.1* library for the graphics interface and *pthread* library for managing the thread structure.

3.1 Global structure

We created three type of structure like this:

$$struct \quad \textsf{PID\_controller} = \begin{cases} \mathit{float} & Ki \\ \mathit{float} & Kp \\ \mathit{float} & Kd \\ \mathit{float} & ref \end{cases} \quad struct \quad \textsf{select\_motor} = \begin{cases} \mathit{float} & L \\ \mathit{float} & R \\ \mathit{float} & J \\ \mathit{float} & b \\ \mathit{float} & ke \\ \mathit{float} & kt \end{cases} \tag{7}$$

$$\mathit{float} \quad out\_ctr \tag{8}$$

Then for each type of control we defined an *array* of *Nm* elements, so we can simulate simultaneously *Nm* motors.

3.1.1 File management

The global variables are defined and initialised in *global.c* and *global.h* and all functions are declared in *functions.h*.

3.2 Graphic Interface

We created a screen (1024x768) and split this panel in two principal section:

(a) Motor rotation display;

(b) Response time of controlled variable (trends);

(a) shows the behaviour of rotation motor after the action of controller: the position of this section is upside of screen. *(b)* shows the controlled variables as function of time: this section is arranged in the center of screen.
On the right bottom of screen we arranged the setting button configuration that allows users to interact during the simulation **Figure 2**.
The users can change the gains of PID controller, the reference signal or motor parameters by push the *set_PID* **Figure 3** and *set_mot* **Figure 4** buttons for each type of controlled motor (The value are casual in following pictures).

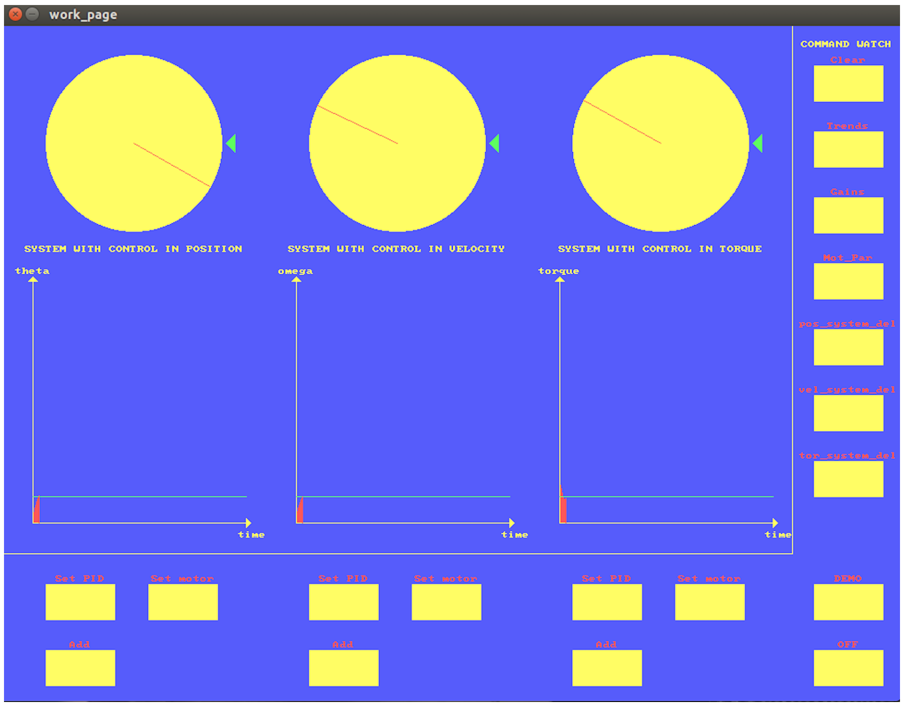
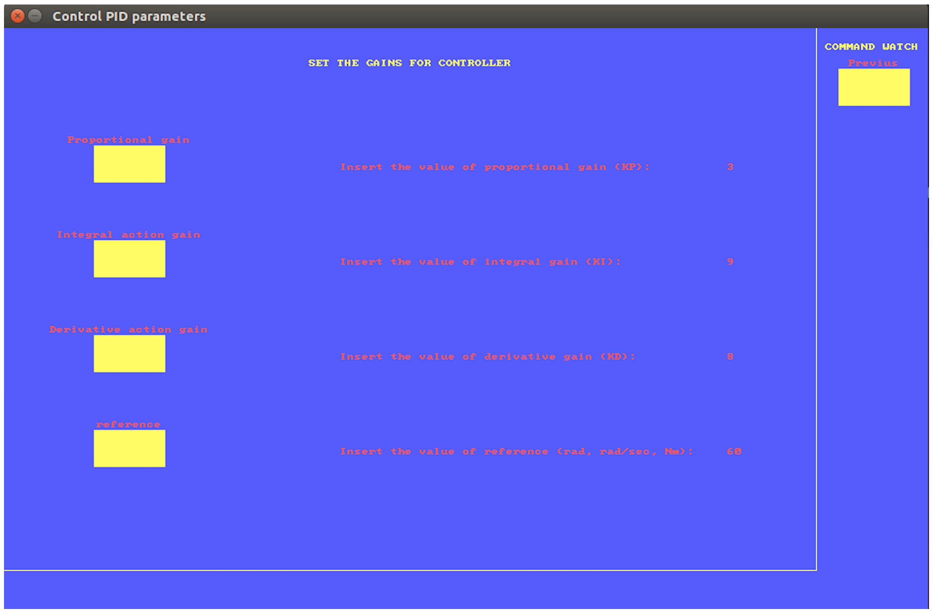


Figure 2: Work page interface

3.2.1 File management

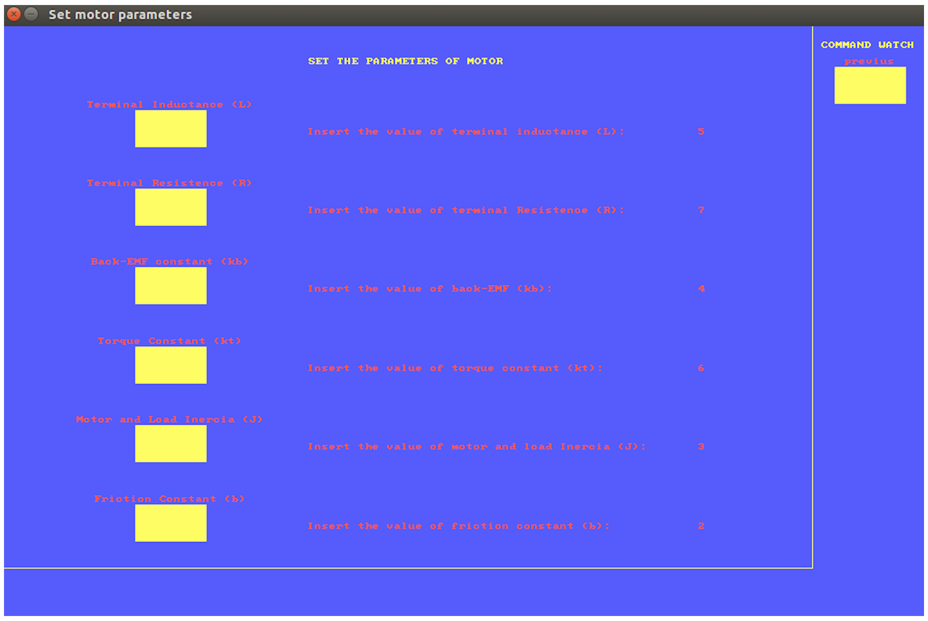
The code is split in many *file.c*:

- *layout_design.c*
  In this file we implemented the work page where is developed the basic interface that allow the users to watch the simulation;
- *motor_parameters.c*
  This file contains the code that implement the interface that appear after press of "set_mot" button and allow the users to change the motor parameters;
- *PID_parameters.c*
  This file implements the interface that appear after press of "set_PID" button and allows the users to change the gains of PID controller and the reference signal;
- *graphical_functions.c* and *utility_functions.c*
  Here are stored the simpler code functions that we used to make the graphical interface before explained and the other operation like press and release of mouse and others;



```
SET THE GAINS FOR CONTROLLER

Insert the value of proportional gain (KP):           3
Insert the value of integral gain (KI):               9
Insert the value of derivative gain (KD):             8
Insert the value of reference (rad, rad/sec, Nm):     60
```

Figure 3: PID gains set page



```
SET THE PARAMETERS OF MOTOR

Insert the value of terminal inductance (L):          5
Insert the value of terminal Resistence (R):          7
Insert the value of back-EMF (kb):                    4
Insert the value of torque constant (kt):             6
Insert the value of motor and load Inercia (J):       3
Insert the value of friction constant (b):            2
```

Figure 4: Motor param set page

3.3 Thread design

We decided to use two basic threads: one for task of users and the second to display the variation of controlled variable that manage the motor rotation and its response time. We used one single thread to display its because both actions operate and access to shared resources instantaneously. Then we created three threads for each type of control, that is the task, that computes the output control and put the results in *float* `out_contr`. For each motor we created a thread with respective task of control. How we can see the portion of memory allocate in *float* `out_contr` is a shared resources between the tread for graphics, that read, and the thread for control, that write, so we used a semaphore conveniently for mutual exlusion.
We choose the thread feature according the task that they do. To guarantee the functionality of the application we assign high priority to control task *(80)* and low priority to graphic task *(70)* because the last one depend of the values computed for the control task.
We set $100\,\mathrm{ms}$ the period of graphic threads and $90\,\mathrm{ms}$ for control threads. Then we set $88\,\mathrm{ms}$ for control thread and $98\,\mathrm{ms}$ graphics thread for the same argument of priority set. This threads are periodic thread.
The thread of mouse is aperiodic thread and for the correct operation we set the period at $50\,\mathrm{ms}$ and intermediate priority *(75)*.

3.3.1 File management

Also here the code is split in many *file.c*:

- *thread_control_task.c*
  In this file are implemented the task for control in position, velocity and torque;
- *thread_graphics_task.c*
  This file develop the display of rotation motor and controlled variable as function of time;
- *thread_mouse_task.c*
  Here is write the code that help the users for interaction with application;

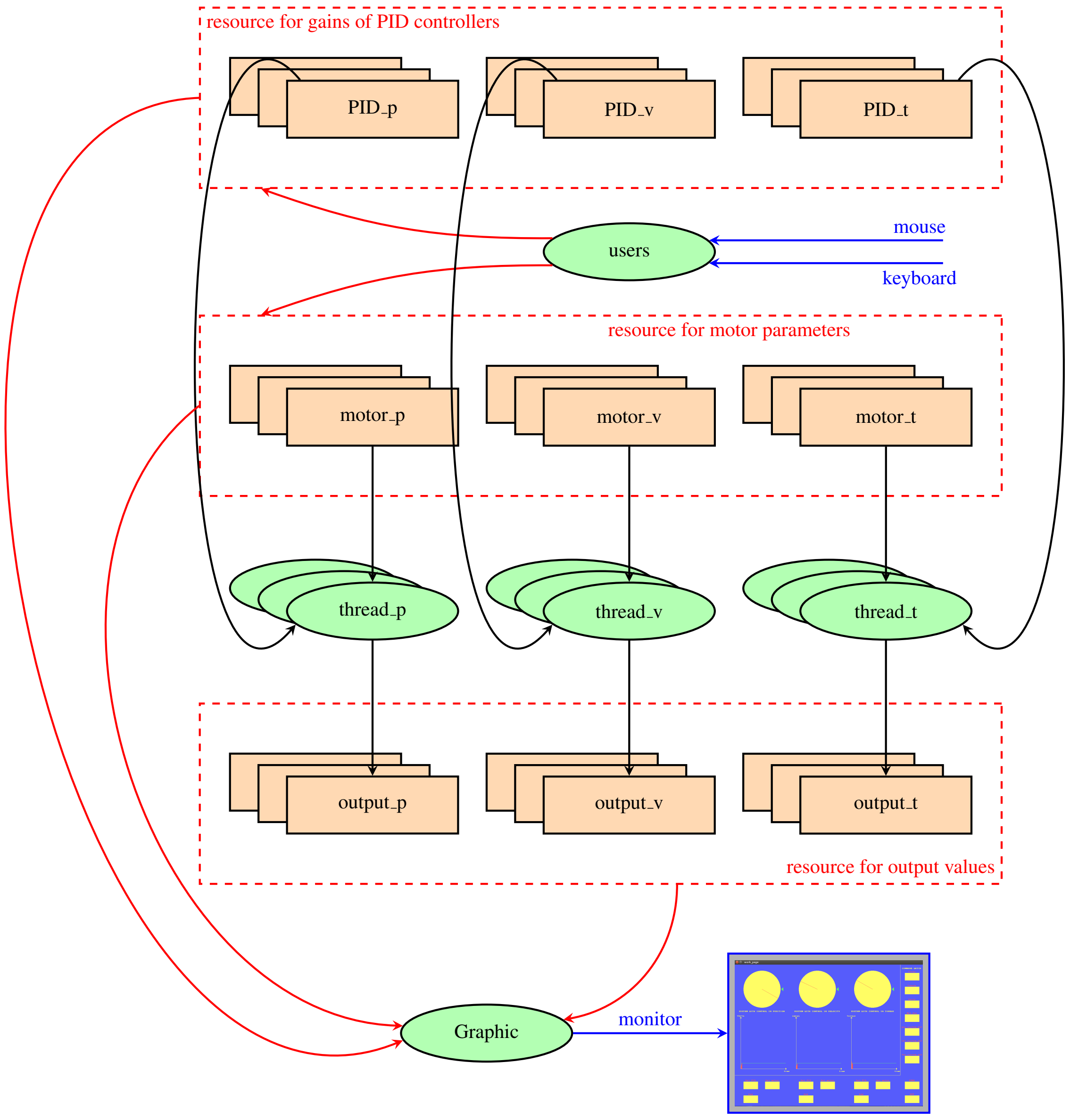


Figure 5: Flux diagram

4 Example Test

We created a “Demo” button that show the typically execution run of the application. The demo update the parameter of motor and the gains of PID by appropriate *file.txt*. We have the following parameters for PID controller:

Type of control	**Kp**	**Ki**	**Kd**
Control in velocity	16.6	49.2	0
Control in Position	49.2	2.99	16.8
Control in torque	200	100	10

and we have the motor with these features:

Motor parameter	**SI units**
J	$0.01\,\mathrm{kg\,m^2}$
b	$0.1\,\mathrm{N\,m^2}$
Ke	$0.01\,\mathrm{V\,rad^{-1}\,s}$
Kt	$0.01\,\mathrm{N\,m\,A}$
R	$1\,\Omega$
L	$0.5\,\mathrm{H}$

Then we simulated this motor and we obtain the good results as we can see in the next **Figure 6**.

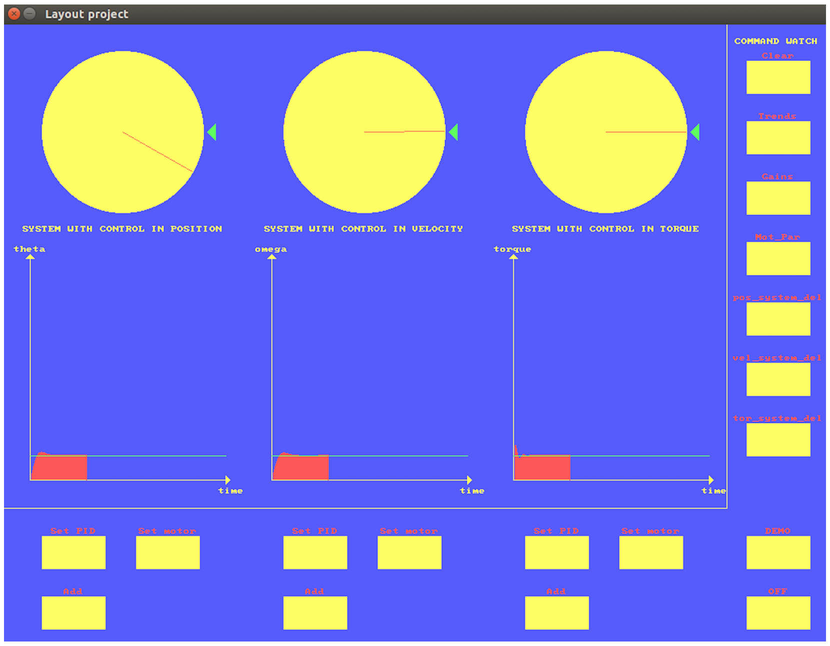
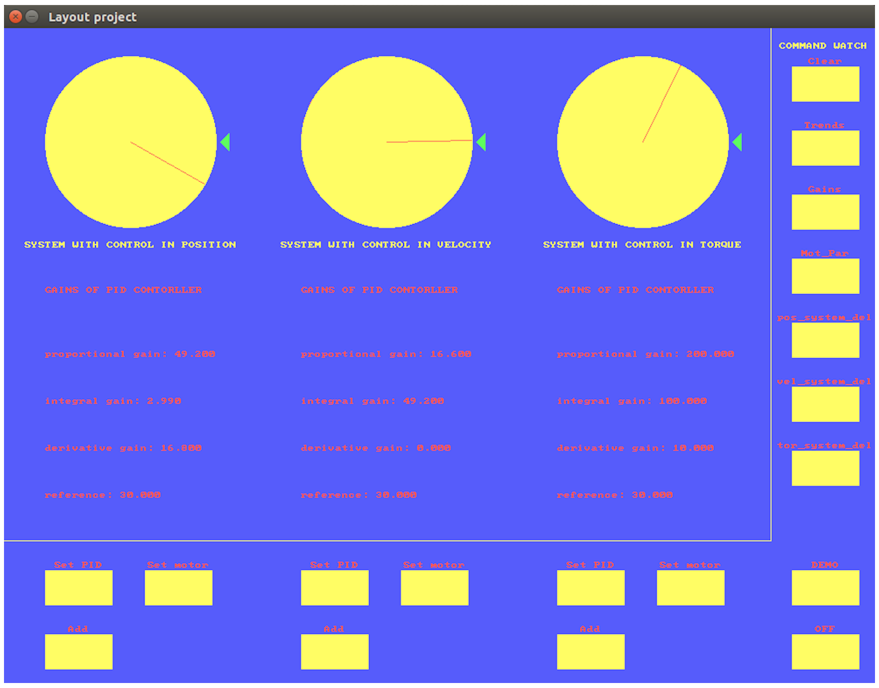
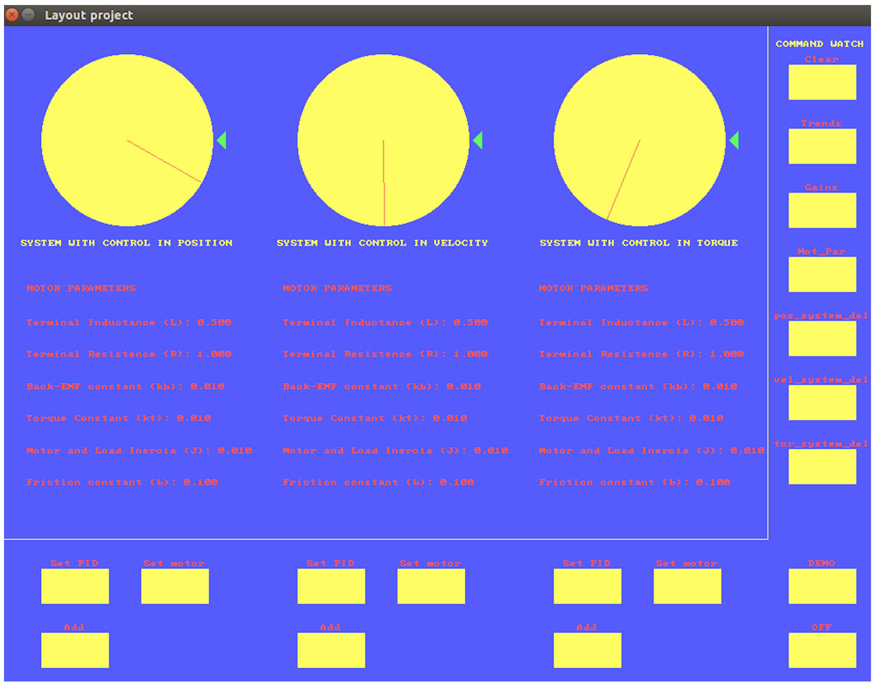


Figure 6: Step response



```
GAINS OF PID CONTORLLER          GAINS OF PID CONTORLLER          GAINS OF PID CONTORLLER

proportional gain: 49.200        proportional gain: 16.600        proportional gain: 200.000

integral gain: 2.990             integral gain: 49.200            integral gain: 100.000

derivative gain: 16.800          derivative gain: 0.000           derivative gain: 10.000

reference: 30.000                reference: 30.000                reference: 30.000
```

Figure 7: PID parameters during simulation



```
MOTOR PARAMETERS                            MOTOR PARAMETERS                            MOTOR PARAMETERS

Terminal Inductance (L): 0.500              Terminal Inductance (L): 0.500              Terminal Inductance (L): 0.500

Terminal Resistence (R): 1.000              Terminal Resistence (R): 1.000              Terminal Resistence (R): 1.000

Back-EMF constant (kb): 0.010               Back-EMF constant (kb): 0.010               Back-EMF constant (kb): 0.010

Torque Constant (kt): 0.010                 Torque Constant (kt): 0.010                 Torque Constant (kt): 0.010

Motor and Load Inercia (J): 0.010           Motor and Load Inercia (J): 0.010           Motor and Load Inercia (J): 0.010

Friction constant (b): 0.100                Friction constant (b): 0.100                Friction constant (b): 0.100
```

Figure 8: Motor parameters during simulation