Department of Electrical Engineering
Indian Institute of Technology Madras

Control Experiments using LEGO Mindstorms

S.R.Manikandasriram
Guide: Dr. Bharath Bhikkaji
July 10, 2014

This report analyses the scope of using the LEGO Mindstorms NXT Kit as a platform for teaching Control Engineering concepts for both Undergraduate and Graduate students. A set of simple experiments using only a LEGO DC Motor and the "NXT Intelligent Brick" have been analysed. The experiments cover the introductory concepts in Control Engineering like Transfer Function, PID Control, System Identification and Bode Plots and is thus ideal for an Undergraduate Laboratory course on Control Systems. The later half of the report deals with building a 2-wheel Self-balancing Robot using LEGO Mindstorms kits and a HiTechnic Gyro sensor. The mathematical model of the *Segway* type robot is explained in detail and a Servo PID control for stabilising the *Inverted Pendulum* is studied. This would be ideal as a Course Project for first year Masters students.

Acknowledgement

There are a number of publications on the use of LEGO Mindstorms kits as an educational platform for undergraduate engineering students which provided valuable insights for this analysis. In particular, the first set of experiments discussed in this report is an extension of the paper *A LEGO based Undergraduate Control Systems Laboratory* by Sabiha A Wadoo and Rahul Jain. Also, the 2-wheel self-balancing robot project discussed in this report is an adaptation of the paper *NXTway-GS Model based design* by Y.Yamamoto in 2008. The Pendulum on a Cart system is an adaptation of the work done by Peter J. Gawthorp and Euan McGookin in 2006.

1 Introduction

The use of LEGO Mindstorms series of kits as a platform for education has received widespread acceptance since the release of its first edition - *Robotics Invention System* - in 1998. The LEGO Mindstorms kit is relatively cheap, robust, customizable, reprogrammable and induces enthusiasm and creativity in students. The official software for LEGO NXT is a NI LabView based programming language called NXT-G. However, enthusiasts have extended the software and hardware in various ways to make the LEGO NXT kits as a favourable Rapid prototyping platform for academic and research activities. In this report, we present a few experiments/projects using these LEGO Mindstorms kits for teaching Control Engineering.

The LEGO EV3 kit is the third generation release in the Mindstorms series. It comes with the *EV3 intelligent brick* which is the "brain" of the kit. The *Intelligent brick* has an ARM9-based Sitara AM1808 processor runnning Linux and has optional WiFi support via USB dongle. More details can be found here.[**?**].

The high reconfigurability of the LEGO Technic parts combined with the Programmable Linux brick, a wide variety of sensors and an active community of students, teachers, hobbyists and researchers makes the LEGO Mindstorms kits as an ideal platform for teaching various engineering concepts ranging from learning a programming language to verifying reinforcement learning algorithms on a physical system.[**?**][**?**]. Various control experiments can be devised and implemented in the Mindstorms Kit.

The first section discusses a series of simple laboratory experiments for analysing the LEGO DC motor. This is ideal for an introductory course in control engineering. The next section discusses a course project using LEGO Mindstorms kit - Two wheeled self-balancing robot. The last section discusses the design and development of a Pendulum on a Cart system for minimizing the sway of the pendulum during linear motion of the cart.

2 Analysing the LEGO DC Motor

This section explains relatively simple control experiments for analysing a DC Motor system. These experiments have been adapted and extended from [**?**]. The analysis is divided into 4 parts - 1. Observing the internal PID Controller, 2. Writing custom PID controller to meet design requirements, 3. Determining and verifying the Transfer Function of the DC Motor and 4. Designing PID controller using the derived Transfer Function. In order to avoid students from having to get used to a new programming language, ROBOTC - a C-like programming language developed by Carnegie Mellon University's Robotics Academy - will be used as software for the first two experiments and MATLAB/Simulink will be used for the later two experiments in this section.

2.1 Observe PID Controller in action

In this first experiment, the students observe the functioning of the internal PID Controller available in ROBOTC. The system under analysis is comprised of an *Interactive Servo motor* which is controlled by Pulse Width Modulation by the Intelligent Brick. The feedback is provided by the *rotary encoder* present inside the *Interactive Servo motor* which gives 360 counts per single revolution of the motor shaft (i.e. a precision of 1 deg). The block diagram for the PID controller is shown in Figure 2.1. Using the feedback from the encoders, the *Intelligent Brick* calculates the instantaneous angular speed of the motor. The formula used for angular speed calculation is

$$ang_spd = \frac{\Delta\theta}{\Delta t}\text{degrees/sec} \tag{2.1}$$

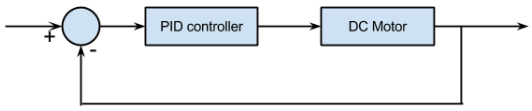

Figure 2.1: Block Diagram of the internal closed loop PID Controller

The LEGO motors are assigned a power rating between -100 and 100 with a negative value denoting reverse direction. Under ideal conditions, the maximum speed of the LEGO motors is 1000 degrees per second. But this value drops with decrease in battery voltage and increase in motor load.

The commands for enabling the internal PID Speed Controller and set the maximum regulated speed is:

```
nMotorPIDSpeedCtrl[motorB] = mtrSpeedReg;
nMaxRegulatedSpeedNxt = 750;
```

The internal PID controller provides consistent speed by continuously adjusting the raw power sent to the motor.

The LEGO Intelligent Brick has a *Datalog* feature which allows values to be stored in memory as `(Key,Value)` pairs, which can later be exported to a PC for post-processing. In this experiment, the encoder counts and Motor PWM level are logged every Δtms. The ROBOTC code for this experiment is given below as the motive of this experiment is to get the students acquainted with the software and hardware interfaces.

```
task main()
{
  nMaxRegulatedSpeedNxt = 500;
  // Reset the Motor Encoder
  nMotorEncoder[motorB] = 0;
  nMotorPIDSpeedCtrl[motorB] = mtrSpeedReg;

  int motorRAWpower, motorDegrees;
  motor[motorB] = 50;    // 50/100
  time1[T1] = 0;         // Timer

  // Allocate memory for datalog
  nDatalogSize = 1600;

  while (time1[T1] < 10000) {
    motorDegrees = nMotorEncoder[motorB];
    motorRAWpower = motorPWMLevel[motorB];
    // store value to Datalog
    AddToDatalog(1,motorPIDdegrees);
    AddToDatalog(2,motorPIDpower);
    wait1Msec(50);
  }
  motor[motorB] = 0;   // Stop the motors
```

Figure 2.2: Experimental setup for observing internal PID Controller

```
  SaveNxtDatalog();
}
```

In order to observe the PID control action, the students are asked to run the above program and apply friction on the motor. The internal PID controller will then step in and increase the motor PWM level in order to maintain the speed. This can also be quantitatively verified from the log file. The datalog file (which would be named as *DATAnnnn.rdt*) can be exported to a PC using the *File Management Utility* available in the ROBOTC Development Environment under *Robot→NXT Brick→File Management Utility*. The *Spreadsheet Upload* feature transfers the datalog file to the PC and additionally converts the *.rdt* file into a *.csv* file which can then be processed using any Spreadsheet Processors.

As mentioned earlier, the data is stored as *Key, Value* pairs which constitute the two columns in the exported CSV file. The datalog sequentially stores data with every call to the `AddToDatalog(<index>,<data>);` creating a new entry in the file with a *key* which is proportional to the *index* and the *data* stored as a 16−bit unsigned integer. Due to this implicit type casting, negative values would get "wrapped" around. Hence appropriate post-processing has to be done to handle the same.

The students can now plot the speed in degrees per second and motor PWM level in a single graph to observe the PID action. One such plot is shown in Figure 2.3.

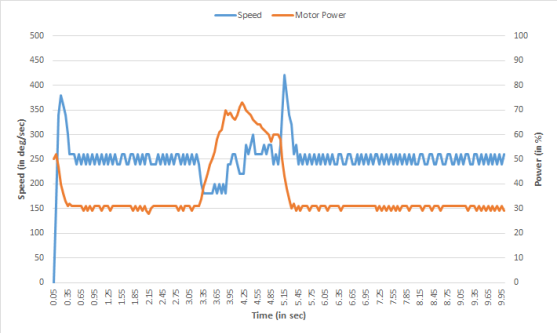

Figure 2.3: Plot of Speed vs PWM level of the LEGO motor

As can be seen from the plot, there is a dip in the speed of the motor between 3 and 5 seconds which caused an increase in the PWM power level applied to the motor. When the friction is removed, the speed shoots up due to high power level, before settling down to the desired speed of 250 deg /s.

2.2 Design and tuning of PID controller

The internal PID controller provided by ROBOTC is sufficient for general purpose use. But in order to understand the characteristics of the DC motor system, writing a custom PID controller would be insightful. In order to implement a custom PID controller, the internal PID controller has to be disabled. The command for disabling the internal PID controller is:

```
nMotorPIDSpeedCtrl[motorB] = mtrNoReg
```

Once the internal speed regulation is disabled, you can write your own controller to regulate the speed in order to satisfy desired system characteristics. In this section, you will write a position controller which will regulate the speed so that the motor turns and stops at the desired *setpoint*. The pseudo-code for writing your own PID controller in ROBOTC is given below:

```
setpoint = distance in degrees
while(true){
new err = setpoint-new reading
diff err = new err-old err
calculate proportional, integral and derivative constants
correction = proportional constant + integral constant + derivative constant
motor PWM = correction
old err = new err
store pwm level and current reading to datalog
wait for few ms
}
```

Students can follow the provided pseudo-code to design and test custom PID controllers. As shown in the previous experiment, the datalog feature can be used to log data during the experiment and spreadsheet processors can be used to measure the system response. In order to achieve desired system characteristics, the PID parameters need to be tuned. Since we don't have a complete mathematical model of the system, the PID constants cannot be determined through analytic methods. But since we know that DC servo motor is a 3rd order system, we can use Zeigler-Nichols method for calculating the initial value of PID parameters.

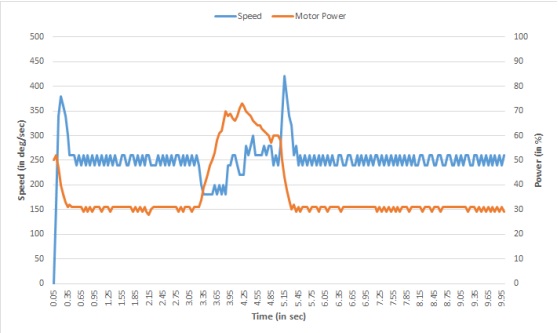

Figure 2.4: Zeigler-Nichols tuning for achieving desired system performance.

Students need to understand that a practical DC motor system needs a minimum voltage in order to start rotating. Also, the difference in static and dynamic friction coefficients will have an effect in the minimum voltage required. While employing Zeigler-Nichols tuning, the setpoint must be chosen to be large enough to produce a

significant voltage output. Also, the calculated PWM level must be constrained to the range $[-100, 100]$ to avoid unexpected behaviours due to "wrapping".

For more details on Zeigler-Nichols method, refer [**?**]. The initial PID parameters obtained using Z-N method may not satisfy the design characteristics. Therefore, the PID parameters are tuned through "informed trial and error" to achieve the desired system performance. See Figure [**?**] for an example.

2.3 System Identification using MATLAB

System Identification is the process of deriving a mathematical model of the system under consideration using experimental data. In the previous experiment, we adopted "trial and error" methods for calculating the parameters for implementing a closed-loop PID controller. In this experiment, we will focus on deriving a complete mathematical model of the DC motor system so that analytical methods can be employed for calculating the PID parameters and verified on the real-world system.

MATLAB provides System Identification toolbox which can be used to estimate transfer function models of an experimental system using the input/ouput data. System Identification is in itself a vast field and hence we will explain only the features required for this experiment. In brief, system identification involves a 3-step loop - 1. Data acquisition, 2. Estimate Mathematical model and 3. Verify the derived model. For this and the next part of this section, MATLAB and Simulink will be the used as the programming environment. This requires updating the LEGO kit firmware with the "enhanced standard firmware". Refer Appendix[**?**] for detailed instructions.

2.3.1 Data Acquisition

The first step in system identification is acquiring "*good data*". This is done by exciting the system using a well-designed input signal and measuring the system response. A well-designed input signal should excite the system with all frequencies. The estimated mathematical model can only be as accurate as the acquired data. But there are fundamental constraints in measuring data which are introduced by the quantizations in the optical position encoder of the motor, time delay in motor response, discrete time data acquisition etc... For the lego DC motor system, the dominant constraint is the low resolution of the rotary encoder which has a least count of 1 deg.

In order to obtain"*good data*", the system needs to excited using a wide bandwidth signal around the natural operating frequency. This can be achieved by using the `Signal Builder` block in Simulink. A *Pseudorandom noise* signal with a frequency of $0.5Hz$ is used to control the LEGO motor. The output of the system is obtained using the `Encoder` block which provides the total degrees rotated by the motor. This time domain input-output data is exported into workspace by using the `scope` block. Refer Figure[**?**] for the complete Simulink block diagram. While the input to the `Motor` block in Simulink toolbox is PWM duty ratio, the actual input to the system is `voltage`. Due to various non-idealities, the PWM to voltage mapping is not linear, especially as the battery voltage does not remain constant throughout the operation. Hence, emperical

results are used to calculate the maximum voltage supplied to the lego motors for a given battery voltage.[**?**]. This Simulink model is then run in *external* mode in order to let the `scope` log the data from the `Intelligent Brick` in real-time. Due to various reasons, the data-logging begins only after a few hundred milliseconds and hence the input is given after $1sec$.

2.3.2 Estimate Mathematical Model

The next step in system identification is to fit a mathematical model to the acquired data. This process is simplified by using System Identification Toolbox in MATLAB. In this experiment, we will estimate the transfer function model of the DC motor system using the measured signal data. The code snippet for achieving this is given below. For more details on transfer function estimation, refer MATLAB's tutorial on *Estimating Transfer Function Models for a Heat Exchanger*.

```
# Assuming scope data is stored in motorResponse variable.
u1 = motorResponse(:,2); % Voltage
y1 = motorResponse(:,3); % Speed
data = iddata(y1,u1,Ts);
sysTF = tfest(data,2,0,nan);
```

The quality of fit can be analysed using `compare` and `residue` commands. It is recommended to have 70%-30% split of the acquired data. The larger portion is used to estimate the transfer function while the smaller portion is used to validate the model. This helps in identifying *over-fitting*.

2.3.3 Verification

The final step in system identification is to verify the model in real-time. This can be achieved by simultaneously exciting the system and the transfer function model with a pulse input and comparing the outputs. The `idmodel` block can be used in Simulink to import the transfer function model from the workspace. The complete block diagram for the verification is shown in Figure [**?**].

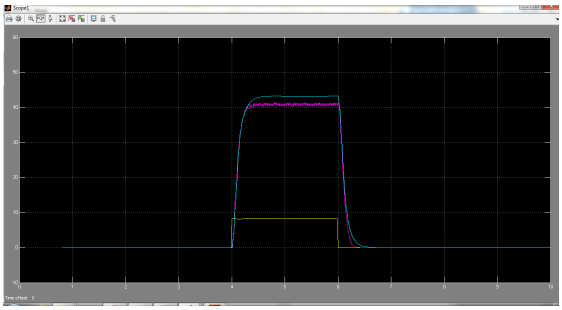

Figure 2.5: Simulink block diagram for verifying the estimated mathematical model.

2.4 PID controller design using mathematical model

Closed loop controller design PID controller design

3 Conclusions for Analysis of LEGO DC Motor

Concepts learned: Block Diagram, Feedback control, Open loop and closed loop control, PID controller, Zeigler-Nichols tuning, transfer function, system identification and verification, bode plots etc...

4 NXTway-GS - A two wheeled self-balancing robot

The second half of the report details the building of a 2-wheel Self-balancing robot using the LEGO NXT kit and a HiTechnic Gyro Sensor. The 2-wheel Self-balancing robot can be modelled as a *2D Inverted pendulum* system which is a well studied classic problem in Control Engineering. This makes it ideal for a 1 month course project in a first year Masters programme in Control Engineering. The analysis is divided into 3 parts - 1. Mathematical Modelling, 2. Servo PID Controller design and 3. Experimental Results. MATLAB provides official Simulink support package for LEGO Mindstorms NXT hardware which allows the students to use the various toolboxes available in Simulink and deploy the model to the LEGO NXT hardware.

4.1 Modelling the 2-wheel Self-balancing Robot

The 2-wheel Self-balancing Robot can be modelled as a 2-wheeled Inverted Pendulum system as shown in Figure 4.1. The mathematical model of such a system is derived here from first principles. The physical parameters used in the model with their nominal

Figure 4.1: 2-wheeled Inverted Pendulum

values have been tabulated in Table **??** in Appendix B.

The equations of motion for the 2-wheeled inverted pendulum can be derived using the Lagrangian method based on the coordinate axis adopted in Figure 4.1. If direction of two-wheeled inverted pendulum is positive x-axis at $t = 0$, then:

$$(\theta, \phi) = (\frac{\theta_l + \theta_r}{2}, \frac{R(\theta_r - \theta_l)}{W}) \tag{4.1}$$

Parameter	Nominal Value	Remark
g	$9.81m/sec^2$	Accel. due to Gravity
m	$0.03kg$	Wheel weight
R	$0.04m$	Wheel radius
J_w	$\frac{mR^2}{2}kgm$	Wheel inertia moment
M	$0.6kg$	Body weight
W	$0.14m$	Body width
D	$0.04m$	Body depth
H	$0.144m$	Body height
L	$\frac{H}{2}m$	Dist. of CoM from wheel axle
J_ψ	$\frac{ML^2}{3}kgm^2$	Body pitch inertia moment
J_ϕ	$\frac{M(W^2+D^2)}{12}kgm^2$	Body yaw inertia moment
J_m	$1\tilde{A}\mathring{U}10\hat{a}\acute{L}\check{S}5kgm^2$	DC motor inertia moment
R_m	6.69Ω	DC motor resistance
K_b	$0.468Vsec/rad$	DC motor back EMF constant
K_t	$0.317Nm/A$	DC motor torque constant
n	1	Gear ratio
f_m	0.0022	Friction coeff. b/w body and DC motor
f_W	0	Friction coeff. b/w wheel and floor

4.2 Controller Design

Linearization about $\psi = 0$ State space representation Servo controller Feedback gain calculations - LQR concepts Integral gain to regulate steady state errors.

4.3 Simulation and Verfication

Simulation results experimental results graphs and diagrams

4.4 Future work

Possible extensions to the project

5 Conclusions for two-wheeled self-balancing robot

Summary of concepts learned through this project. Time frame for the project. Possible alterations

MATLAB and Simulink provides many *hardware support* packages for developing standalone algorithms which can be directly deployed in the target hardware of choice from Simulink. A wide variety of platforms are supported which include Arduino, Raspberry Pi and LEGO Mindstorms NXT and EV3 as well. The *Hardware Support* packages can be installed using `supportPackageInstaller` command in MATLAB command window. Alternatively, the LEGO NXT kit can be controlled from MATLAB using RWTHMindstormsNXT toolbox. This MATLAB toolbox largely developed by members of RWTH Aachen University, provides matlab scripts for communication with the LEGO NXT Intelligent brick and to interface with the motors and sensors that are attached to the Intelligent brick.

- Set Ki and Kd to zero.
- Slowly increase Kp to a value Ku at which sustained oscillations - constant amplitude and periodic - are observed.
- Note period of oscillation
- Refer table below for initial values

Z-N Model	Kp	Ki	Kd
P controller	0.5*Ku	0	0
PI controller	0.455*Ku	0.833*Tu	0
PID controller	0.588*Ku	0.5*Tu	0.125*Tu

Since Z-N method usually results in aggressive tuning, alternatively Tyreus-Luyben method can be adopted.

T-L Model	Kp	Ki	Kd
PI controller	0.312*Ku	2.2*Tu	0
PID controller	0.454*Ku	2.2*Tu	0.159*Tu

8.3 Servo Control (PID type)

8.4 Updating firmware of LEGO Intelligent Brick