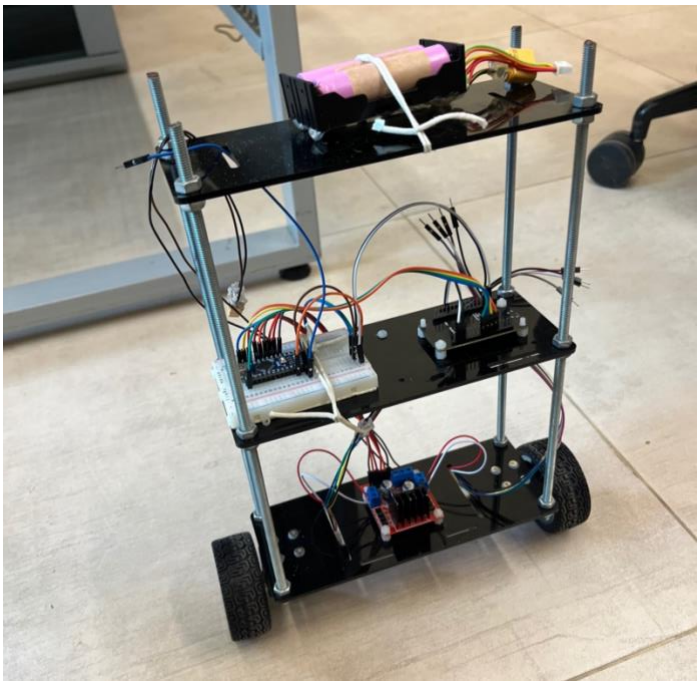
Lebanese American University

MCE 411– Mechatronics System Design II

Instructor: Dr. Noel Maalouf

PROJECT: Self-Balancing Robot.



Ziad Saoud

Roy Sawaya

Chadi Kallas

Date of submission: Saturday, 14 May

Table of Contents

System Modeling

Mechanical System:

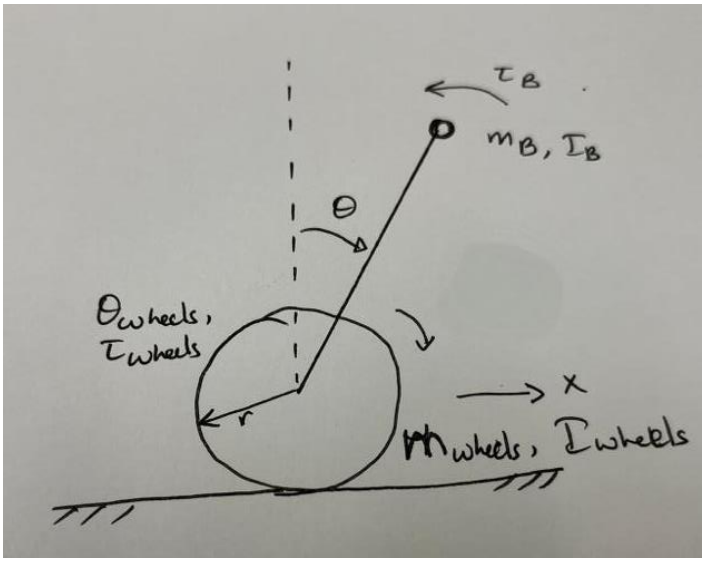


Figure 1 Free body diagram of the Robot's mechanical System

The no slip condition of movement is given by

$$\boldsymbol{x = r\theta_{wheel}};$$

The movement of robot's body COG is given by

$$\boldsymbol{x_{COG} = x + lsin\theta}$$

$$\boldsymbol{z_{COG} = lcos\theta}$$

$$\boldsymbol{v_{COG} = \sqrt{\dot{x}_{COG} + \dot{z}_{COG}}}$$

Kinetic energy of the body is given by

$$\boldsymbol{E_{kB} = \frac{1}{2}m_b{v_{COG}}^2 + \frac{1}{2}I_b\dot{\theta}^2_{Wheels} = \frac{1}{2}m_b\left(\dot{x} + 2\dot{x}lcos\theta \times \dot{\theta} + l^2\dot{\theta}^2\right) + \frac{1}{2}I_b\dot{\theta}^2}$$

Kinetic energy of the wheel is given by

$$\boldsymbol{E_{kW} = \frac{1}{2}m_W\dot{x}^2 + \frac{1}{2}I_{Wheels}\theta^2 = \frac{1}{2}m_W\dot{x}^2 + \frac{1}{2}\frac{I_{Wheels}}{r}\dot{x}^2}$$

Potential energy of the body is given by

$$\boldsymbol{E_{pB} = m_bglcos\theta}$$

Potential energy of the wheel is given by $E_{pw} = 0$;

The Lagrangian of the system is defined as the difference between the above kinetic and potential energy:

$$\boldsymbol{L = E_{kB} + 2E_{kW} - E_{pB} - 2E_{pw}}$$

The Lagrange second order equations are defined as:

$$\boldsymbol{\frac{d}{dt}\left(\frac{\delta L}{\delta \dot{q}i}\right) - \frac{\delta L}{\delta qi} = Qi}$$

where qi represents generalized coordinates and Qi represents generalized forces. Generalized forces are torques from both wheels:

$$\boldsymbol{Qi = 2\tau_{Wheels}}$$

Position x and angular position θ were chosen as system coordinates Substituting system coordinates into generalized coordinates, we can rewrite the Lagrange second order equations and get equations of motion. For the x coordinate we obtain:

$$\boldsymbol{\left(m_B + 2m_{wheels} + \frac{2I_{wheels}}{r^2}\right)\ddot{x} \;\; + m_B l cos\theta \times \ddot{\theta} \; - m_B l sin\theta \dot{\theta}^2 = \frac{2\tau_{Wheels}}{r}}$$

and for the θ coordinate the equation is

$$\boldsymbol{(m_B l^2 + I_B)\ddot{\theta} \; - mgl sin\theta + m_B l cos\theta \times \ddot{x} \; = 0}$$

Also,

$\boldsymbol{sin\theta = \theta;}$

$\boldsymbol{cos\theta = 1}$;

$\boldsymbol{\dot{\theta}^2 = 0}$;

The linearized equations become:

$\boldsymbol{\left(m_B + 2m_{wheels} + \frac{2I_{wheels}}{r^2}\right)\ddot{x} \;\; + m_B l\ddot{\theta} \; = \frac{2\tau_{Wheels}}{r};}$

$\boldsymbol{(m_B l^2 + I_B)\ddot{\theta} \; - mgl\theta + m_B l\ddot{x} \; = 0;}$

Actuator Subsystem

The brushed DC gearmotors are used as the actuator subsystem of the robot. The subsystem directly provides rotary motion and coupled with the wheels allows the robot to make movement. The input to this subsystem is voltage source Vs applied to the motor's armature, while the output is the rotational speed ω and torque τ_W of the gearbox shaft.

The torque τ generated by a DC motor is in general proportional to the armature current i and strength of the magnetic field. Assume that the magnetic field is constant. In that case the motor torque is proportional to the armature current by constant factor kt:

$\boldsymbol{\tau = K_t i;}$

While the shaft is rotating the back emf voltage V_{emf} is generated and is proportional to the angular velocity of the shaft by constant factor K_e:

$$\boldsymbol{v_{emf} = k_e \omega};$$

From the electrical equivalent circuit of the armature, we can derive the equation:

$$\boldsymbol{v_s - Ri - \frac{Ldi}{dt} - v_{emf} = 0};$$

From the free-body diagram of the rotor we can derive the equation:

$$\boldsymbol{I_M \ddot{\theta}_M + b_m \dot{\theta}_M + \tau_L = \tau};$$

By putting the together, we obtain the equation describing the overall DC motor characteristics

$$\boldsymbol{I_M \ddot{\theta}_M + (b_m + \frac{K_t K_e}{R}) \dot{\theta}_M + \tau_L = \frac{K}{R} v_s};$$

then by using the gear aspect ratio n we obtain:

$$\boldsymbol{\dot{\theta}_M = n \dot{\theta}_W};$$

$$\boldsymbol{\tau_L = \frac{\tau_w}{n}};$$

Substituting the above equations into and denoting by I_G the internal inertia of the gear and by b_G the internal damping of the gear, we obtain the equation of the actuator subsystem:

$$\boldsymbol{(I_M n^2 + I_G) \ddot{\theta}_W + ((b_m + \frac{K_t K_e}{R}) n^2 + b_G) \dot{\theta}_W + \tau_W = \frac{Kn}{R} v_s};$$

Overall Model

DC gearmotors are located between the wheels and the robot's body.

$$\boldsymbol{\theta_{Wheels} - \theta = \theta_W}$$

Substituting the equations all together:

$$\boldsymbol{(\frac{m_b r^2}{2} + m_W r^2 + I_W + I_M n^2 + I_G) \ddot{x} + ((b_m + \frac{K_t K_e}{R}) n^2 + b_G) \dot{x} + (m_b l r^2 / 2 - r(I_M n^2 + I_G)) \ddot{\theta} - r((b_m + \frac{K_t K_e}{R}) n^2 + b_G) \dot{\theta} = \frac{r K_t n}{R} v_s;}$$

$$\boldsymbol{m_b l \ddot{x} + (m_b l^2 + I_b) \ddot{\theta} - m_b g l \theta = 0;}$$

The above equations can be rewritten into a matrix form:

$$\begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -E^{-1}G & 0 & -E^{-1}F & \end{bmatrix} \begin{bmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ E^{-1}H \end{bmatrix} Vs$$

System Control:

To control the tilt angle of our robot we used the MPU6050 sensor it has a built-in accelerometer and gyroscope. It is known that the gyroscope angular velocity readings are not noisy however estimating the tilt angle only using the gyroscope will cause the angle value to drift significantly and on the other side, the accelerometer readings are noisy but do not drift. Hence, we used a dedicated Kalman filter library to use the accelerometer and gyroscope values to get a very good and reliable readings of the tilt angle.

Fuzzy Logic Control

In this part of the report we will show how the fuzzy controller was build and tested on MATLAB before implementing in on hardware.

Construction of Membership functions

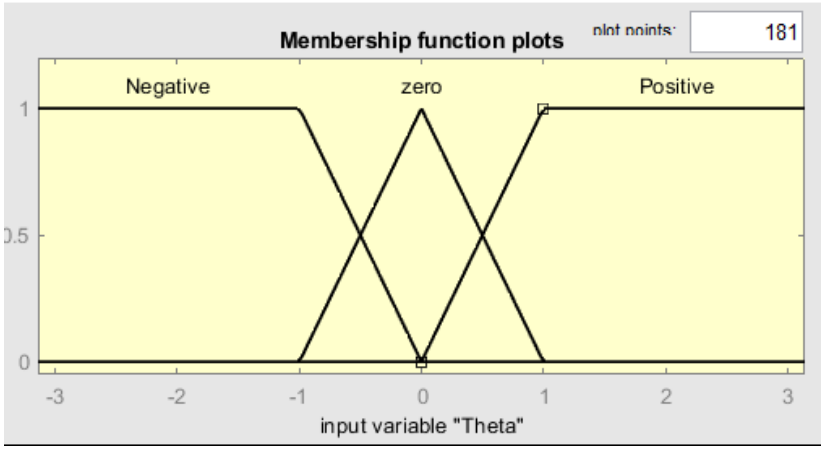
To control the system, we first had to get our inputs from the state variable vectors which are:
θ
$\dot{\theta}$

For the angle θ (our first input x_1) we decided to choose a universe of discourse of

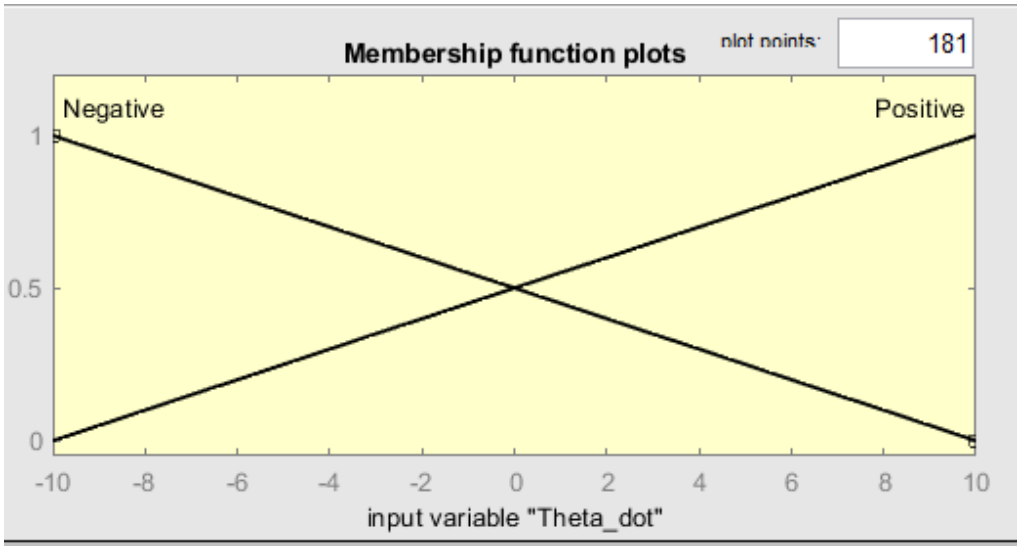
$$-\pi \leq x_1 \leq \pi$$

and for the angular speed $\dot{\theta}$ (second input x_2) we finally settled for a universe of
$-10\ rad/s \leq x_2 \leq 10\ rad/s$

Next for the first input we constructed 3 membership functions each representing the state of the robot at a given time and that can be Positive, Zero, or Negative as shown in figure below:



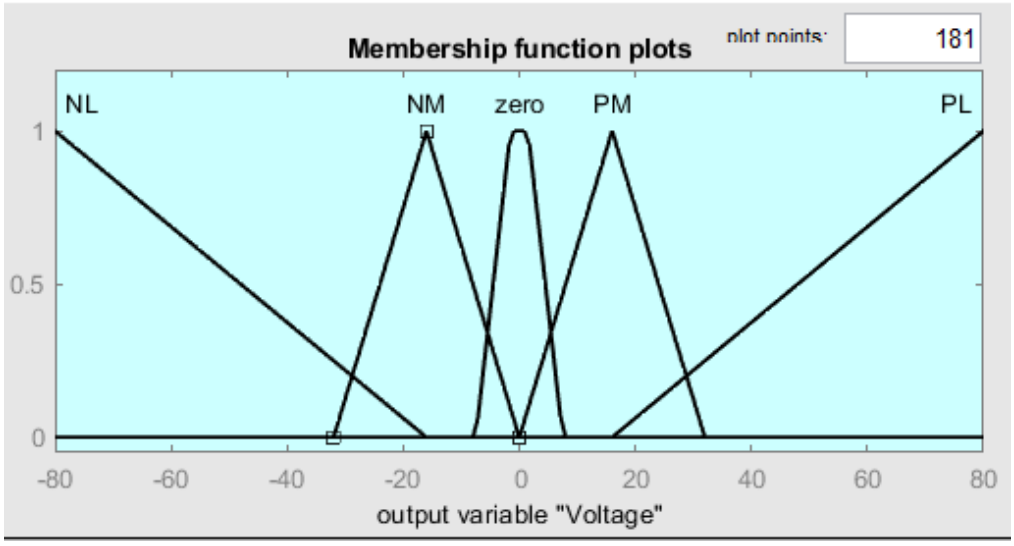
Then, we construct two membership functions for x_2 on its universe, that is, for when the angular speed is either increasing or decreasing: P (positive), N (negative).



Finally, to partition our control space output u which is the voltage supplied to DC motors, we constructed five membership function for u on its universe which is

$$-80V \leq u \leq +80V.$$

The membership functions were divided as follows: NL (negative large), NM (negative), Zero, PM (positive), and PL (positive large).



It is important to note that at the beginning of our simulation we did not have a zero-membership function for our output, however after we added it, we visualized a much better performance of the robot. Additionally, we decided to go one step further and have this membership function as trapezoid ta have a bigger range at 0, this also caused an improvement in the balancing of the robot.

Constructing the rule base

After the stage of constructing membership functions, we construct a Fuzzy Associative Memory (FAM) Table for the rule base. According to the FAM table our rule base can be expressed as:

R1: $if\ (x_1 = N)\ and\ (x_2 = N)\ then\ u = NL$

R2: $if\ (x_1 = P)\ and\ (x_2 = P)\ then\ u = PL$

R3: $if\ (x_1 = P)\ and\ (x_2 = N)\ then\ u = NM$

R4: $if\ (x_1 = N)\ and\ (x_2 = P)\ then\ u = PM$

R5: $if\ (x_1 = zero)\ and\ (x_2 = N)\ then\ u = zero$

R6: $if\ (x_1 = zero)\ and\ (x_2 = P)\ then\ u = zero$

Additionally, the FAM table for our robot can be seen below:

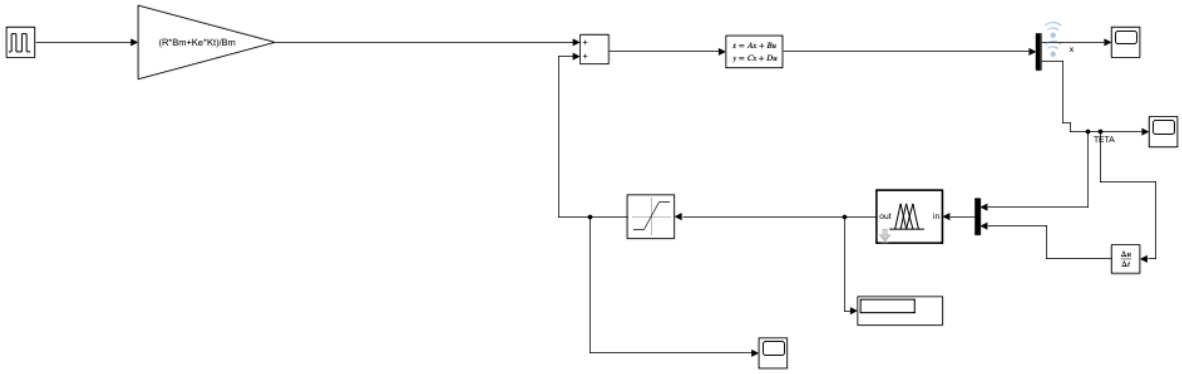
x_1 \ x_2	Neg	Pos
Neg	NL	PM
Pos	NM	PL
Zero	Zero	Zero

Next, we needed to determine both the fuzzification and defuzzification methods. We opted for both the inference system with Mamdani method, and with the centroid method respectively.

Finally, we did a couple of runs of MATLAB fuzzy to test our fuzzy system and to determine if any change was needed before moving to Simulink

MATLAB implementation and Results

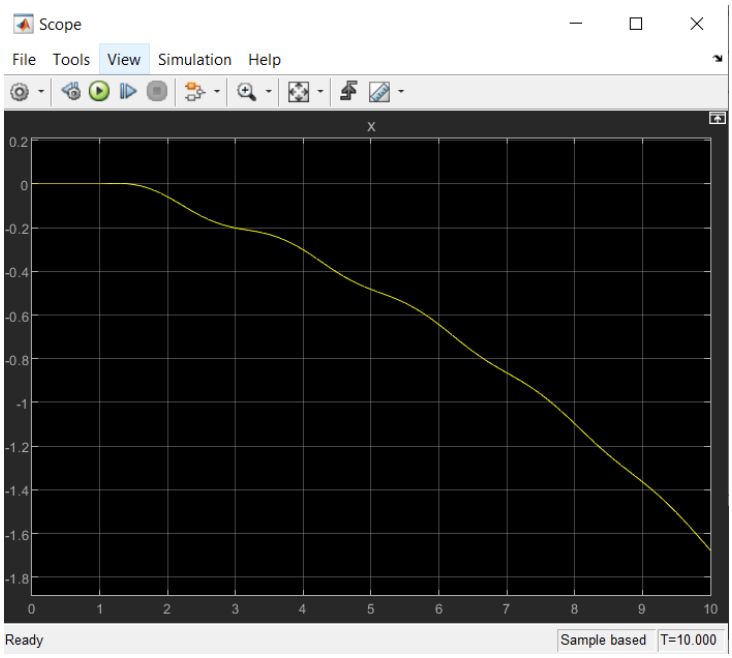
After checking our results on MATLAB fuzzy, we decided to link both our fuzzy controller with our plant that we created earlier.

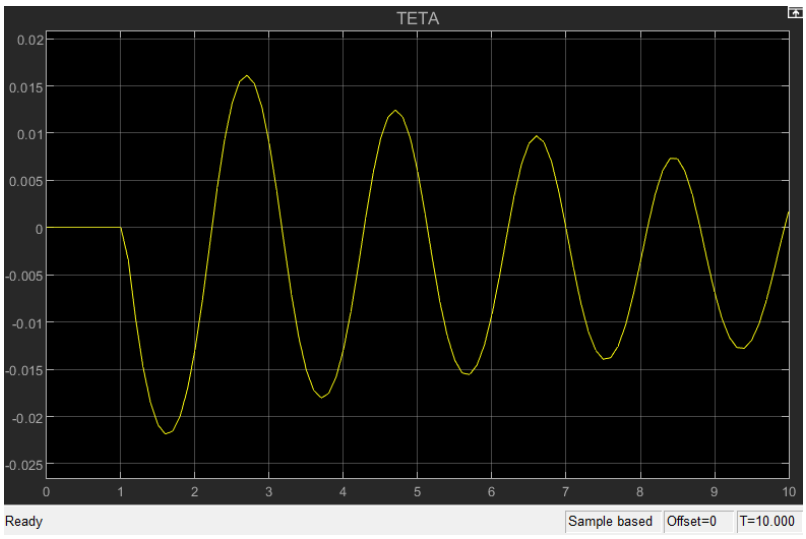


As shown in the picture above, the output of my system is the state space which will give 4 state variables, however we only needed two to be the input to our fuzzy controller which are
θ
$\dot{\theta}$

After both these inputs passes through the fuzzy controller, it will output the voltage V that needs to be fed back to the plant. Moreover, we decided to simulate an external disturbance and this can also be viewed in the image above. This disturbance will be an impulse set by the PWM block that will shock my system every 10sec with a 1% pulse period and with an amplitude of 0.2. The saturation block placed at the output of the fuzzy logic will make sure that the voltage that reached the plant will always be between -12V and 12V.

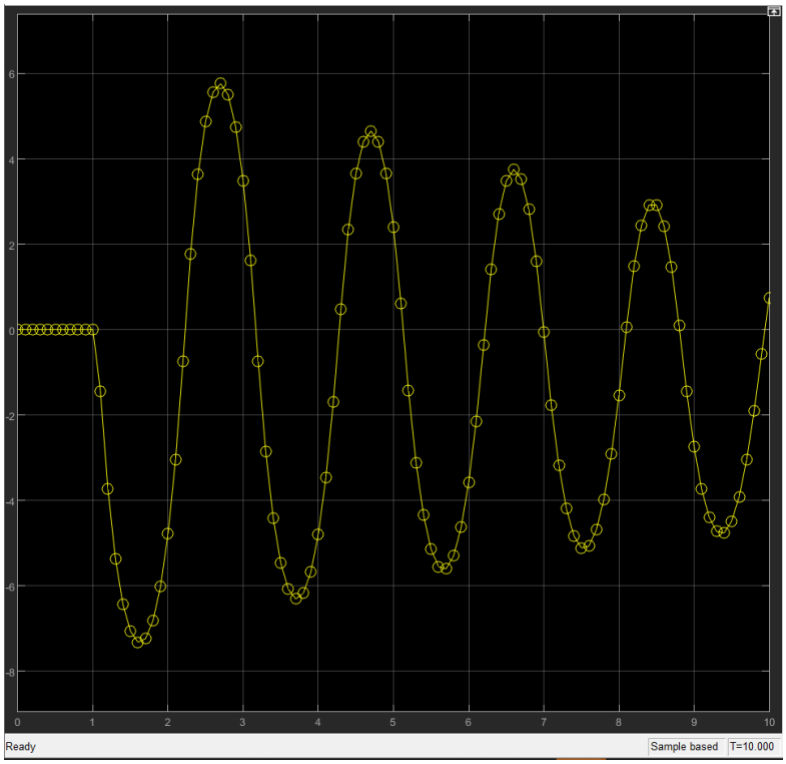
Additionally, we also wanted to view the change in position and angle of our system so we added a scope to both these variables and after running the system for 10 seconds we observed the following results which can be seen below:





For the results, we can observe that the angle θ, it is slightly varying between 0.02 rad and -0.02 rad. This actually makes sense since the robot, due to the fuzzy controller will be actively trying to balance itself after being physically pushed (disturbance in the Simulink model). As for the position x, in order to balance itself, the robot will be moving towards the direction where he was pushed to stay balanced. If position control was added then, the position should have oscillated around 0, similar to the angle θ.

We also added a scope to visualize my voltage output that is being fed back to the plant:

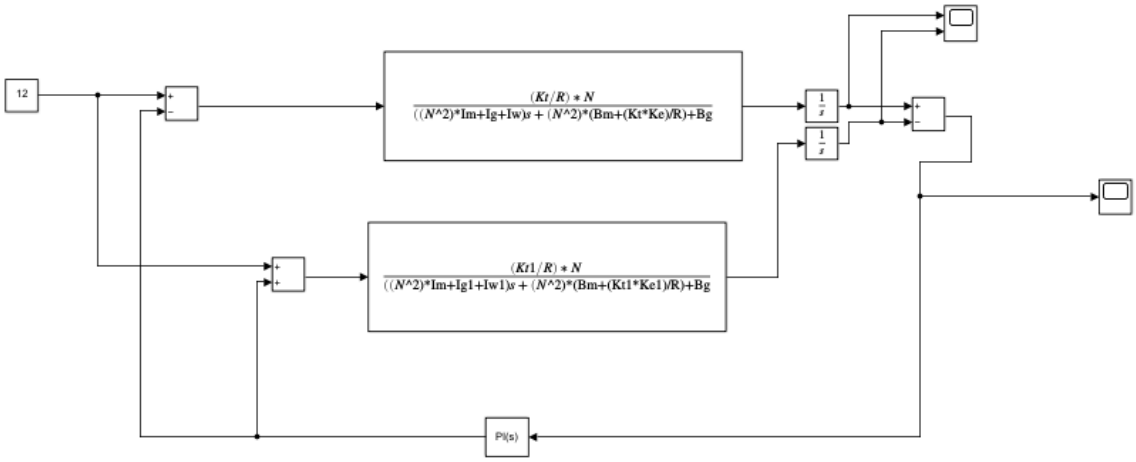


We can observe that it is following a similar path to the angle θ. This is due to the fact that when θ is at its lowest (robot is falling on the left side), the wheels of the robot need to rotate in the counterclockwise direction to adjust its position, then the voltage supplied to the motors need to be negative and with a high amplitude, and vis versa.

Wheel synchronizer

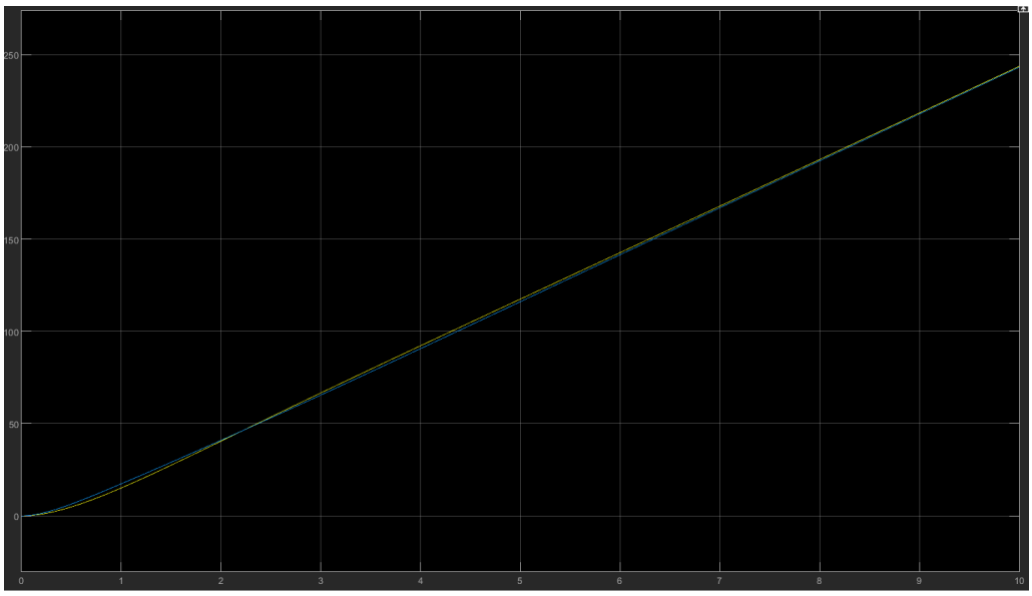
After many runs, we realized that the motors did not have the same speed and this affected the performance of the balancing as it can be viewed in the video that we submitted. To fix this issue, we decided to do add a PI controller, which will synchronize both wheels to have the same speed.

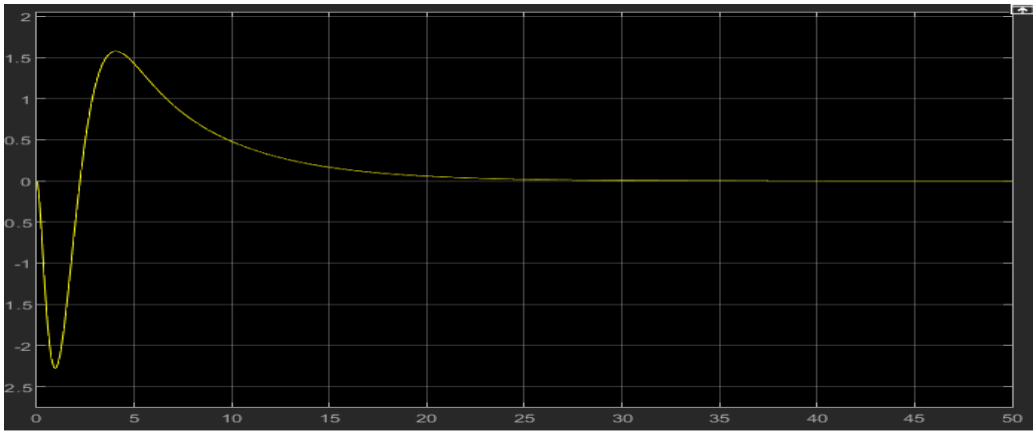
The model of the system can be seen below:



On MATLAB, to simulate different motor speeds, we changed some parameters of the second motor block, we increased a bit the Kt and Iw parameter and this resulted in a slower output speed. The input of the PI will be the difference of the speed, as it can be considered the error to be fixed. On the other side, the output of the PI will once be added to the reference voltage of 12V for the slower motor, and will be subtracted from the fastest, the it will be respectively fed back to each motor. After auto tuning the PI we got Ki to be 0.225 and Kp to be 0.038.

The results of the scopes can be shown below:



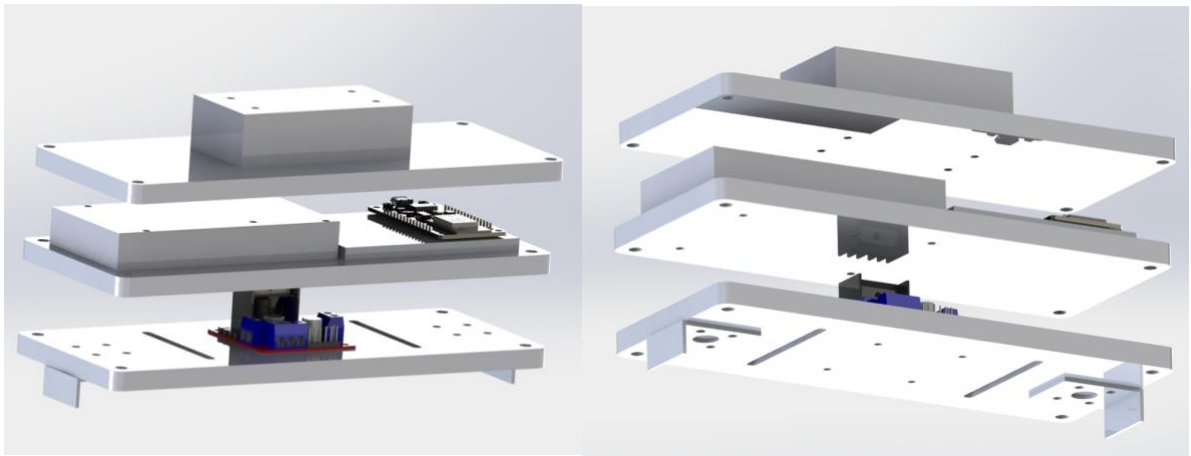


We can observe that after some time the error between the motors become negligible and the yellow and blue line, which each represent the speed of each motor in the first figure, will follow each other.

However, in the end, due to the limitation of pins and due to time constraint, we decided not to implant this step on the hardware.

SolidWorks Design

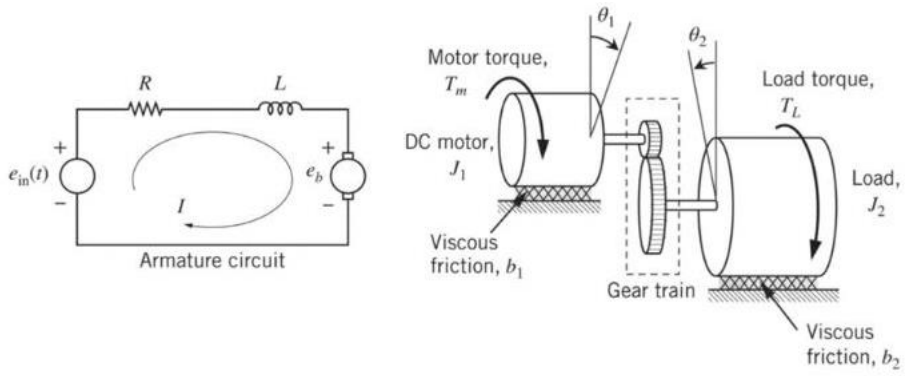
During this project, we had to design the whole robot from the ground up. We first started by getting the exact measurements of DC motor and the wheels. Then we went and designed the first layer whom the motor will be attached to and where the h-bridge will stay. Then we decided that all the batteries and the power bank, which are the heaviest part of the robot, should be place on the top layer so that the robot model will be as close as possible to a pendulum. Finally, a middle layer was created to hold both microcontrollers (the NodeMCU and the Arduino Nano).



At the end we opted to carve the three layers using Plexiglas material and not to 3D print them. The Plexiglas material used was 3mm thick and it turned out to be stronger and much cheaper.

DC Motor Parameter Estimation with MATLAB:

First, we derived the transfer function of the DC motor according to the following image.



When performing the parameter estimation experiment our load (Robot wheel) was not touching any surface hence we assumed that the viscous friction is negligible. Also, in the modeling of the motor we neglected the inductance L of the motor. The aim of estimating the parameters where to get a transfer function on MATLAB that behaves as similar as possible to our motor in real life, that will allow us to have a precise model of our robot and will give us a better idea if the control algorithm we are going to implement will actually work or not on real hardware.

Available parameters of the motor:

N: Gears ratio.

I_w: Wheel inertia.

Unknown parameters:

k_t: Torque constant.

k_e: Back EMF constant.

I_m: DC motor rotor inertia.

R: DC motor armature resistance.

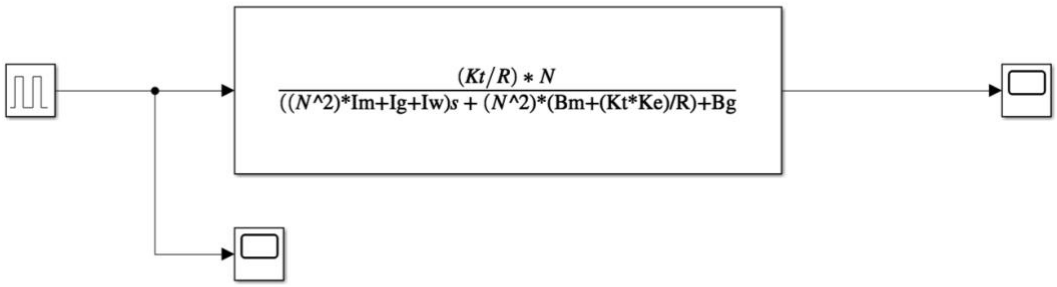
b_m: DC motor damping coefficient.

b_g: DC motor Gearbox damping coefficient

I_g: DC motor Gearbox inertia

Estimation Experiment:

First, we created a transfer function block of the DC motor on Simulink.



The input of the block is a voltage and the output is the speed of the motor in rad/s. We apply a voltage pulse on the motor and we measure the speed of the motor in rad/s. In order to collect the speed readings of the motor our motor featured a magnetic quadrature encoder that has a resolution of 11pulses/rotation of the motor shaft. We used the 2 available pins on the Arduino that feature external interrupts to capture the pulses of the encoder, also our program was configured to trigger an interrupt on both rising and falling edges of the two signals A and B of the encoder. In other words, we were using our encoder in X4 resolution mode and therefore we will get a resolution of 44pulses/rotation of the motor shaft. In order to solve for the velocity of the motor our program will get the number of pulses that occurred in a certain time interval and using the following formula to get the velocity.

$$W_{wheel} = \frac{60 \times \#Pulses}{Resolution \times N \times T_{sampling}} (rpm)$$

After plotting the data of the encoder, we noticed a clear noise in the data and it is caused due to the discrete counts provided by the encoder hence we implemented a digital low pass filter to reduce the magnitude of the noise and hence get a better estimate of the real value of the angular velocity of the wheels.

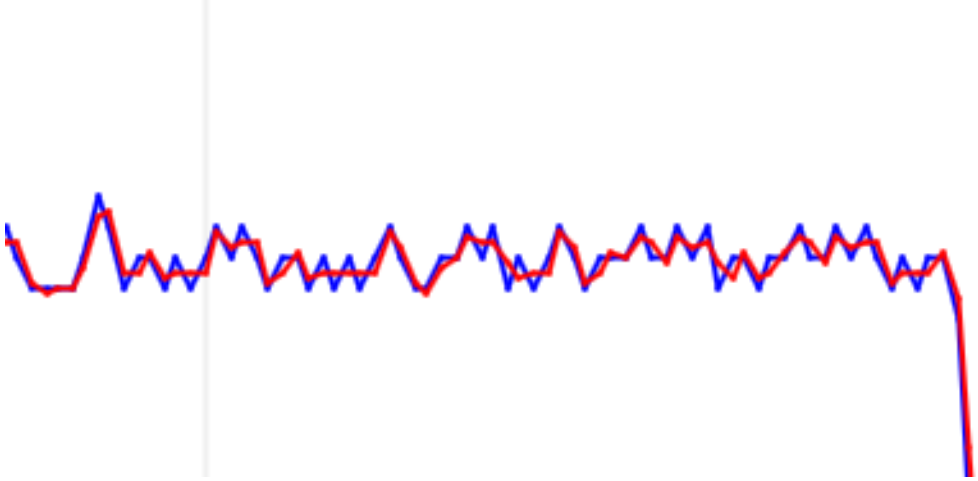
Low pass filter transfer function:

$$\frac{W_{filtered}(s)}{W(s)} = \frac{2PIf_0}{s + 2PIf_0}$$

Using the Bilinear Euler Discretization:

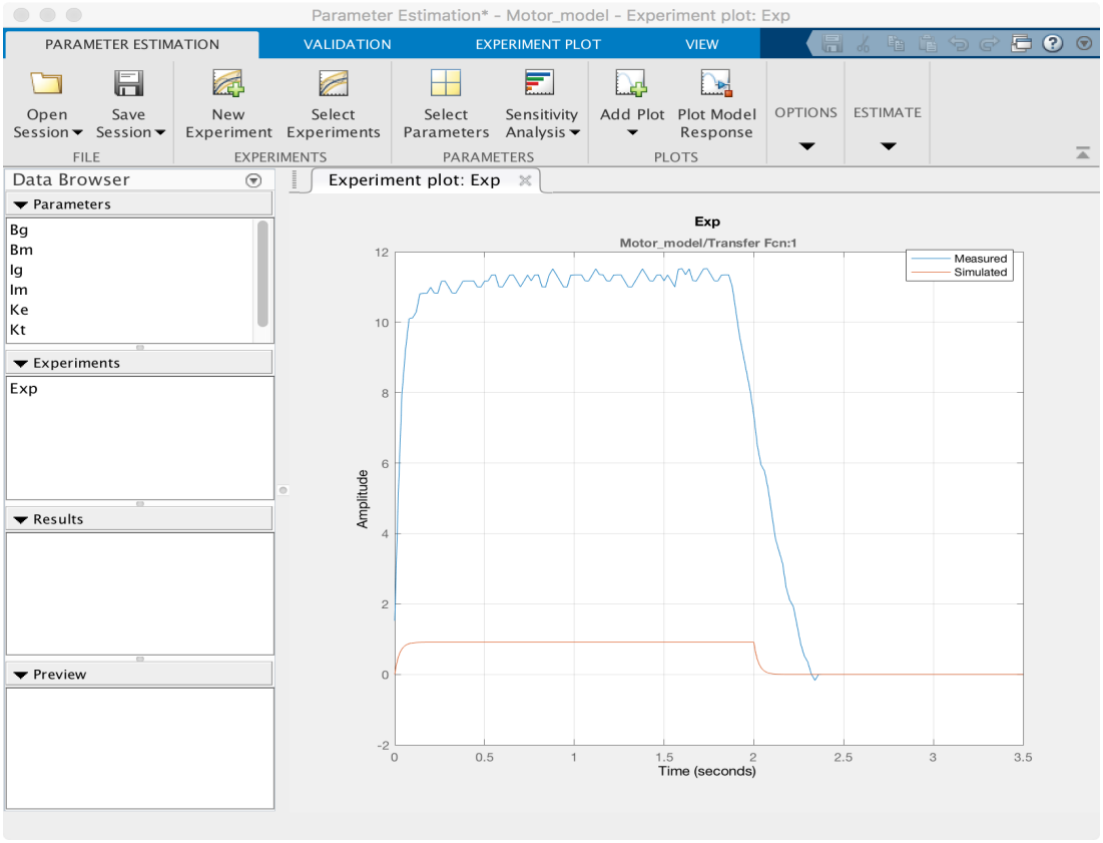
$$W_f(K) = \frac{1}{PIf_0T_s + 1}(-(PIf_0T_s - 1)W_f(k-1) + PIf_0T_s(W(k) + W(k-1))$$

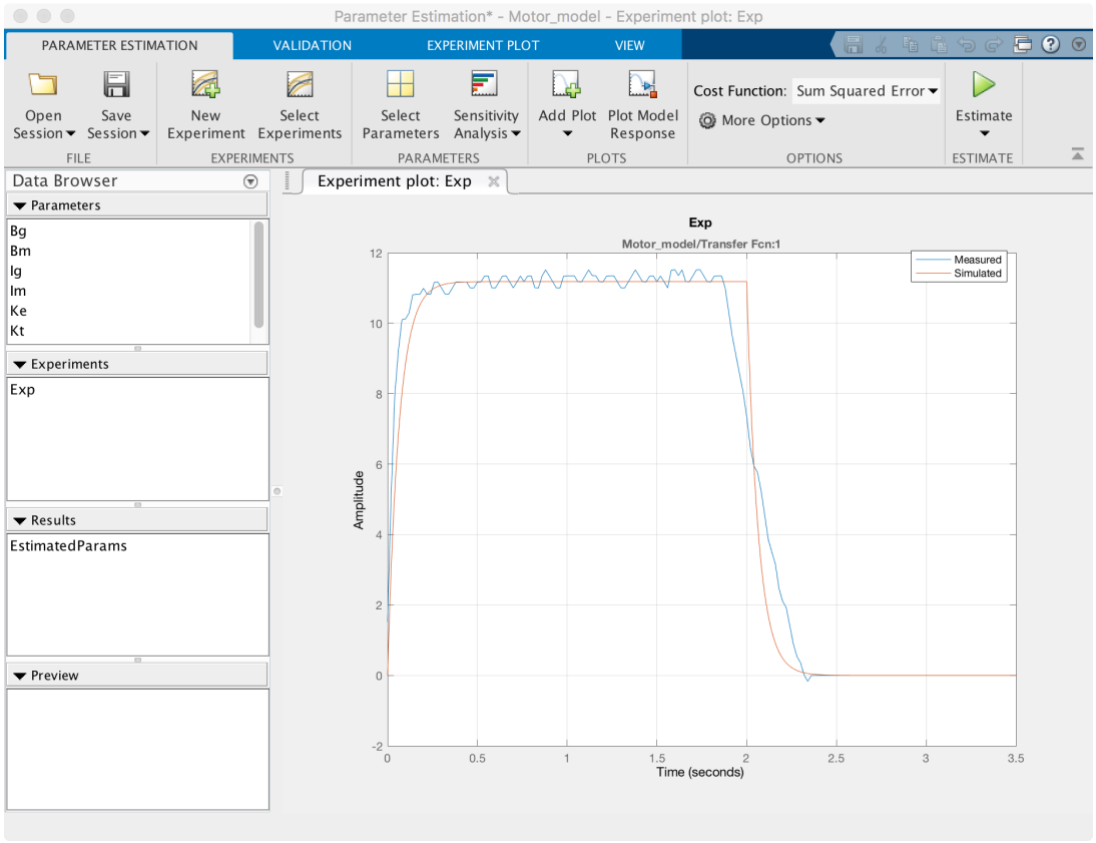
The following picture shows the unfiltered signal in blue and the filtered signal in red.



We tried different variations of the filter with different sampling time and cutoff frequencies and we found best that a cutoff frequency $f_0 = 15Hz$ and a sampling Time $T_s = 20ms$ the filtered signal tracks the real signal with minimal delay as seen in the above figure.

Now moving on to the parameter estimation on MATLAB the following picture we can see the collected data from the experiment plotted in Blue and the motor response with initial common known values for the DC motor parameters. We can clearly see the big difference in the response of the real motor and the Simulink block of the transfer function of the motor.

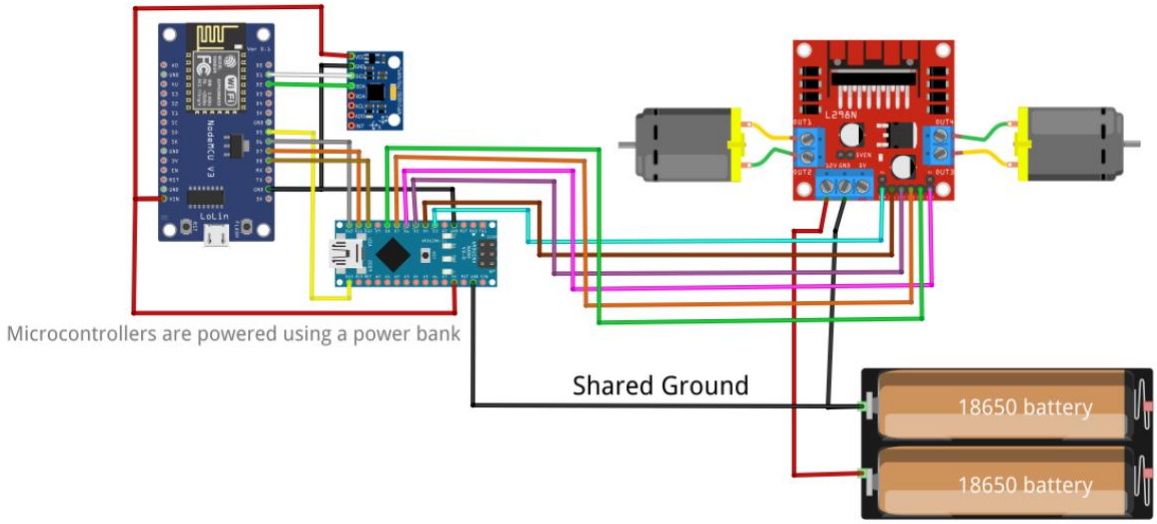




As we can clearly see in the above picture after estimating the parameters of the DC motor the transfer function has minimal error with the real model.

Circuit Design:

For the circuit design our design is simple we had 2 microcontrollers taking over SPI (serial peripheral interface). The ESP8266 microcontroller where the fuzzy controller and Kalman filter are programed it is a fast microcontroller running at 80MHz however it does not feature PWM pins and for PWM generation for controlling the speed and direction of the DC motors we used an Arduino Nano microcontroller that receives the data over SPI from the ESP.



Useful Links:

Demo video Link:

https://youtu.be/-Y4fEU-bn3E

GitHub Repository for the project:

https://github.com/ZiadSaoud/Self-Balancing-Robot