Microspin Internship Report - June 2019
Authors: Shritej & Pradeep

The first project assigned was to model the behaviour of the DC motor controlled by Pulse Width Modulation (PWM) using PI controller.

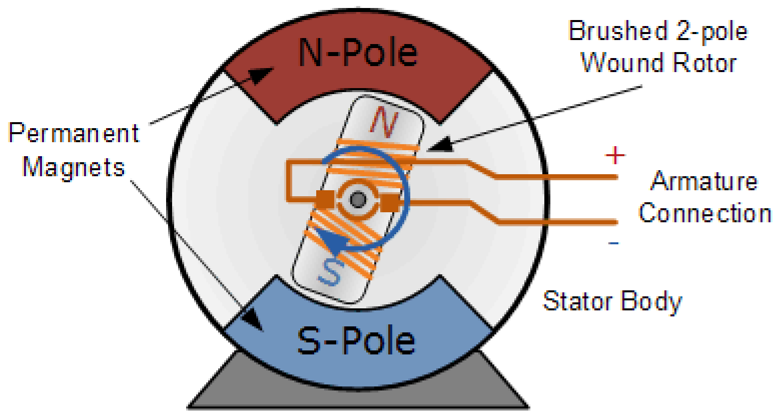
Theory :
A DC motor consists basically of two parts, stationary body of the motor called the "Stator" and the inner part which rotates producing the movement called the "Rotor". For D.C. machines the rotor is commonly termed the "Armature".

Generally in small light duty DC motors the stator consists of a pair of fixed permanent magnets producing a uniform and stationary magnetic flux inside the motor giving these types of motors their name of "permanent-magnet direct-current" (PMDC) motors.

The motors armature consists of individual electrical coils connected together in a circular configuration around its metallic body producing a North-Pole then a South-Pole then a North-Pole etc, type of field system configuration.

The current flowing within these rotor coils producing the necessary electromagnetic field. The circular magnetic field produced by the armatures windings produces both north and south poles around the armature which are repelled or attracted by the stator's permanent magnets producing a rotational movement around the motors central axis as shown.

2-Pole Permanent Magnet Motor



As the armature rotates electrical current is passed from the motors terminals to the next set of armature windings via carbon brushes located around the commutator producing another magnetic field and each time the armature rotates a new set of armature windings are energised forcing the armature to rotate more and more and so on.

So the rotational speed of a DC motor depends upon the interaction between two magnetic fields, one set up by the stator's stationary permanent magnets and the other by the armatures rotating electromagnets and by controlling this interaction we can control the speed of rotation.

The magnetic field produced by the stator's permanent magnets is fixed and therefore can not be changed but if we change the strength of the armatures electromagnetic field by controlling the current flowing through the windings more or less magnetic flux will be produced resulting in a stronger or weaker interaction and therefore a faster or slower speed.

Then the rotational speed of a DC motor (N) is proportional to the back emf (Vb) of the motor divided by the magnetic flux (which for a permanent magnet is a constant) times an electromechanical constant depending upon the nature of the armatures windings (Ke) giving us the equation of: $N \propto V/K_e\Phi$.

So how do we control the flow of current through the motor. Well many people attempt to control the speed of a DC motor using a large variable resistor (Rheostat) in series with the motor as shown.

While this may work, as it does with Scalextric slot car racing, it generates a lot of heat and wasted power in the resistance. One simple and easy way to control the speed of a motor is to regulate the amount of voltage across its terminals and this can be achieved using "**Pulse Width Modulation**" or PWM.

As its name suggests, pulse width modulation speed control works by driving the motor with a series of "ON-OFF" pulses and varying the duty cycle, the fraction of time that the output voltage is "ON" compared to when it is "OFF", of the pulses while keeping the frequency constant.

The power applied to the motor can be controlled by varying the width of these applied pulses and thereby varying the average DC voltage applied to the motors terminals. By changing or modulating the timing of these pulses the speed of the motor can be controlled, ie, the longer the pulse is "ON", the faster the motor will rotate and likewise, the shorter the pulse is "ON" the slower the motor will rotate. Pulse width modulation is a great method of controlling the amount of power delivered to a load without dissipating any wasted power.

PI (Proportional Integrator) controller characterized by something called negative feedback mechanism where the change in input voltage feed into the motor is directly proportional to the error between the desired and actual rpm produced and is given by the following equation.

$$\Delta V = K_p\epsilon(t) + K_i \sum_{t=0}^{t} \epsilon(t) \tag{1.1}$$

where,

$$\epsilon(t) = \omega_t(t) - \omega_o(t)$$

also,

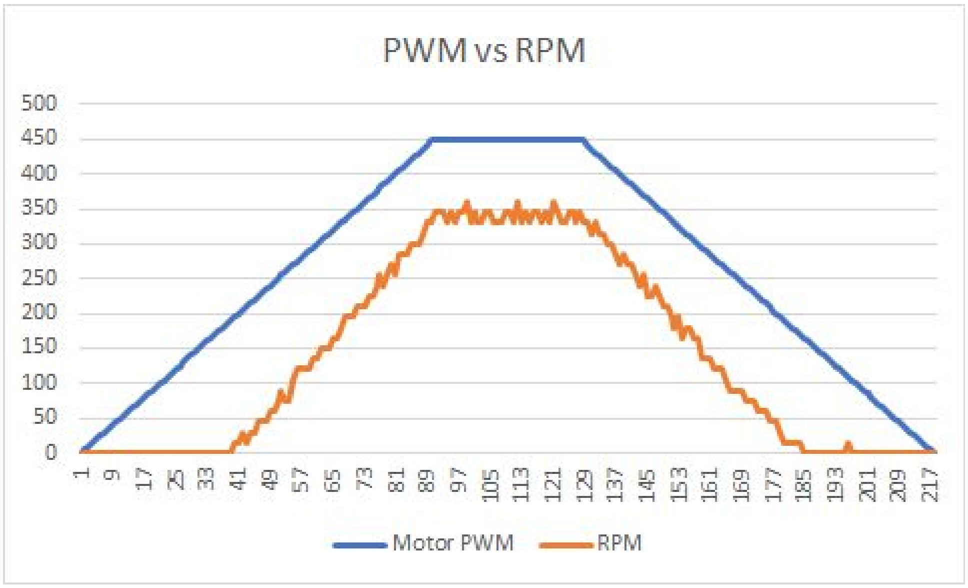
$\omega_t(t)$ - Target RPM of the motor

$\omega_o(t)$ - Actual Output RPM of the motor

K_p - Proportional Gain

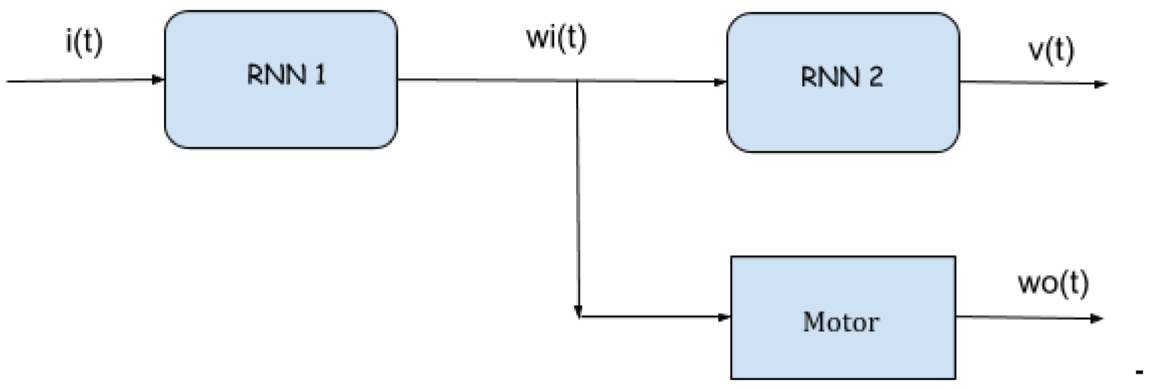
K_i - Integral Gain

The motor we tried to model was multiple pole permanent magnet DC motor. The relationship between the Motor PWM and RPM of the given motor can be seen in the below figure.



Model :

To predict the behaviour of the motor i.e to predict the future rpms given some 'n' past rpm becomes the problem of Recurrent Neural Network. So the following architecture of the model was suggested



Where, i(t) - Input RPM to the model
wi(t) - Input PWM to the motor and the RNN_2
v(t) - Output RPM from the RNN 2
wo(t) - Desired RPM output from the motor

The model was to be trained such that the following loss functions were minimized for respective RNN_1 and RNN_2 :

$$\sum_t (\omega_o(t) - v(t))^2 \tag{1.2}$$

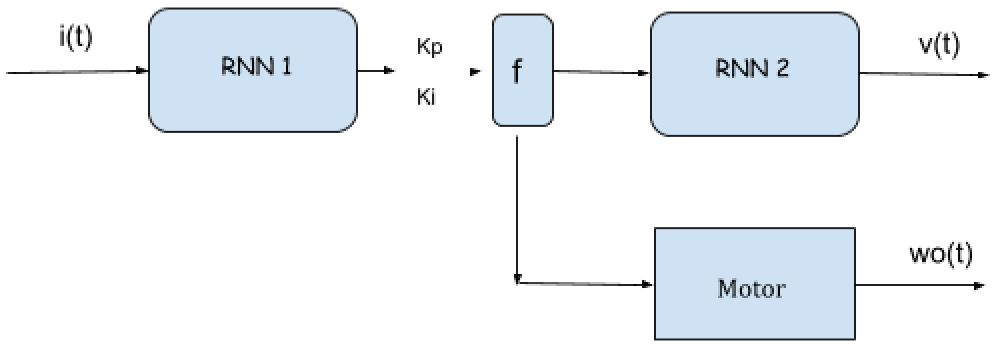
$$\sum_t (v(t) - i(t))^2 \tag{1.3}$$

The model includes 2 RNN's so capture the variation in motor characteristics over time like motor drift, etc.

Since the architecture of the model was proposed, the next question arose as to how to train the model. One idea was to train both RNNs simultaneously but so as to do that we needed to operate a motor during training which was not possible. Another idea was to train RNN_2 first and learned the weights of it and then train RNN_1 by freezing RNN_2 weights. We went with the latter one.
As we can see in the block diagram, the RNN_2 takes PWM as the input of the neural network and produces a corresponding RPM. In case of RNN we give a sequence of data, in this case we give timesteps of PWMs to predict the future RPM.

For coding purpose, python module Keras was used. Sequential model was initialized and two LSTM (Long Short Term Memory) layers were added to the neural network of 8 neurons and 1 neuron respectively. Also the timesteps for prediction was taken to be 3. The Loss function was taken to be the first one in the above set of equations and Optimizer was 'Adam'. The model was trained successfully on the given data which also contained negative values of PWM and RPM.

Now, the RNN_1 model was to be trained by freezing the learned weights of RNN_2 and by the use of the second cost function. But the future challenge was to perform the computation on chip which seemed like a long shot given the ability of the team. To cope with that we came up with the idea of redesigning the architecture of the model so as RNN_1 outputs just the values of Kp and Ki as shown in the PI control equation, which is then fed into the function which would calculate change in PWM which acts as an input to the RNN_2 as well as the motor. Using this architecture of the model, it would be easier to debug and hence less cumbersome.



Training:

As we can see in equations above the 2nd loss function is to be minimized for training RNN_1. To achieve that we have to backpropagate from the output layer of RNN_2 but since pretrained RNN_2 model was used we don't require to update the weights of the RNN_2.

Due to unstable training for the above method with two neural networks, it was decided to remove the RNN1 model. Instead, a few variations of the algorithm that involved using the RNN2 as a model for the motor and performing a least squares regression on the error equations thus obtained were analysed. Unfortunately, theoretical guarantees couldn’t be provided for the accuracy of these methods which made improvement of these algorithm performances difficult.