Motor Control Simulation

Abdullah Alawfi | ELCT 350 | 12/08/2021

Introduction

The aim of this project is to create a discrete time simulation of a simple PID control system to control the rotation speed of a PMDC motor. The simulation was attempted in c++ and via Simulink tool. In order to simulate the system, a mathematical description of the DC motor's transfer function is necessary. Modified Nodal Analysis (MNA) is also used in this project to find the voltages and currents across and through the components of the systems. The closed-loop system is composed of a reference value (desired output), negative feedback, PID controller, model of the PMDC motor and added torque, and controller voltage source. The overall block diagram is shown in figure 1.

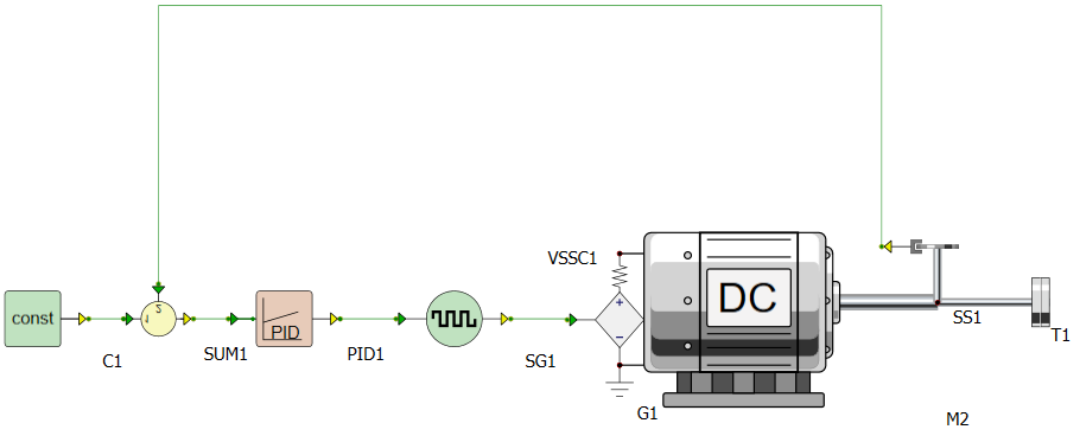

Figure 1. Block diagram of the closed-loop control system

Model Identification

In order to simulate the system, one must have an understanding of the output (the reaction) of the motor at different inputs. The mathematical model that relates the motor's input and output is the transfer function. A Permanent Magnet DC motor (PMDC) converts electrical energy into mechanical energy. The current flowing through the motor's coil generates a magnetic field that would be in a constant motion of repelling and attracting with the permanent magnet surrounding the coil. It is essential for the simulation to have an idea of the characteristics of the electrical and mechanical energy conversion. The moto's electrical circuit diagram is shown in figure 2.

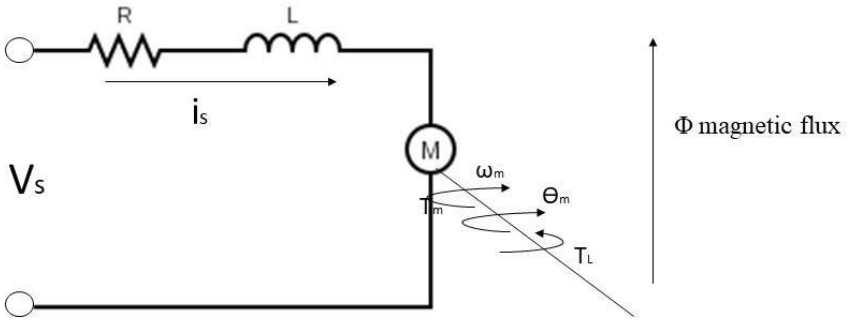

Figure 2. Circuit diagram of a PMDC motor. R and L are inherited characteristics of the motor.

A KVL analysis can be obtained from the schematic in figure 2, the resulting expression of the KVL analysis is given in equation 1.

$$V_s(t) = i_s R + \frac{L di_s}{dt} + V_m \quad \text{(eq. 1)}$$

V_m is the voltage drop across the motor, while R and L are physical characteristic of the motor. Similarly, an analysis of the motor's rotational force using newton's second law of rotation can be obtained. Newton's law states that the sum of all torques exerted on a rigid body is equal to the object's moment of inertia multiplied by its angular acceleration. This yields the expression in equation 2.

$$J_m \ddot{\theta} = T_m - B_m \omega_m \quad \text{(eq. 2)}$$

Where J_m is the motor's inertia, T_m is the motor's torque, T_L is the load's torque, B_m is viscous friction, and ω_m is angular velocity. Assume a constant K_m that sets the relationship between motor's voltage drop and the motor's rotational velocity, then V_m in eq.1 can be expressed as

$$V_m = K_m \omega_m \quad \text{(eq. 3)}$$

So by combining the three equations,

$$J_m \ddot{\theta} + B_m \omega_m = K_m i(t) \quad \text{(eq. 4)}$$

$$L \frac{di}{dt} + Ri(t) = V - K_m \omega_m \quad \text{(eq. 5)}$$

Then, apply Laplace transform on equation 4 and 5, to arrive at the transfer function (equation 8)

$$s(Js + b_m)\theta(s) = K_m I(s) \quad \text{(eq. 6)}$$

$$(Ls + R)I(s) = V(s) - K_m s\theta(s) \quad \text{(eq. 7)}$$

$$G(s) = \frac{\dot{\theta}(s)}{V(s)} = \frac{\omega_m(s)}{V(s)} = \frac{K_m}{JLs^2 + s(JR + bL) + bR + K_m^2} \quad \text{(eq. 8)}$$

Modified Noda Analysis (MNA)

The traditional signal flow approach require the examiner to track the input and outputs of each node for the circuit. Some limitation of that approach is that it requires a new derivation of equations for each node or output. The current expression can only be applied to the current circuit topology. Any change on the circuit will require new derivation of output equations. It is also difficult to track the signal when the system is cascaded with another system. An example of the traditional signal-flow approach is the nodal analysis performed on the circuit of the DC motor in the previous section.

A major feature of MNA is that it allows to create a solver method that eliminates the need for manual analysis. This mean each block of the circuit (each component) will have characteristics information that does not correlate to the other blocks or components of the overall system. Then, the algorithm takes into account all the characteristics matrices of the blocks and combine them to create an overall system matrix. This often yield larger set of expressions, however, it is easier to automate the process of solving the circuit with a computer.

The MNA standard form is given equation 9. Here, I is referred to as the through vector, G is the Jacobian matrix, V is the across vector. In case of electrical circuits, this would be the current and the voltage respectively.

$$\vec{I} = G\vec{V} - B \qquad \text{(eq. 9)}$$

The Jacobian matrix and the intercept matrix hold a signature description of an MNA block. That signature, plays a major role in determining the through vector and the across vector. The size of the Jacobian matrix is n×n where n is the number of ports of the block. The vectors will also have n number of rows. For instance, figure 3 shows an MNA block of a resistor. As labled, the resistor have two ports, P and N.

Figure 3. A resistor MNA block

Hence, the size of the Jacobian matrix is 2×2, and the remaining vectors will have two rows. Since this example deals with an electrical component, the through vector represent the current through the resistor and the across vector represents the voltage across the resistor. The current through the resistor is obtained by the equation below. The current at port N is flowing out of the resistor, hence the direction is opposite to the current flowing in (the value will be negated).

$$I_p = \frac{V_p - V_n}{R} \qquad \text{(eq. 10.1)}$$

$$I_N = -\frac{V_p - V_n}{R} \qquad \text{(eq. 10.2)}$$

Based on the set of equation above, the overall matrix can be obtained using equations 9, 10.1, 10.2

$$\begin{bmatrix} I_p \\ I_N \end{bmatrix} = \begin{bmatrix} \frac{1}{R} & -\frac{1}{R} \\ -\frac{1}{R} & \frac{1}{R} \end{bmatrix} \begin{bmatrix} V_p \\ V_n \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{(eq. 10.3)}$$

Similarly, for the purpose of the simulation, each block of the system will have its MNA stamp. The MNA stamps needed for this project are MNA stamp of the DC motor, Torque load, and controlled voltage source. This is because the calculations of rotational velocity, current, and voltage are dependent on these components only.

MNA STAMP OF CONTROLLED VOLTAGE SOURCE

A controlled voltage source have a series resistor connected to it. Figure 4 shows a diagram of an MNA block of the controlled voltage source

Figure 4. MNA block of a controlled voltage source with a dc resistance

Similar to the example shown previously, the stamp of this differ from the resistor block only at the intercept since we added a voltage source to the resistance. Previously, the voltage source was zero hence it does not contribute to the system matrix.

$$\begin{bmatrix} I_p \\ I_N \end{bmatrix} = \begin{bmatrix} \frac{1}{R} & -\frac{1}{R} \\ -\frac{1}{R} & \frac{1}{R} \end{bmatrix} \begin{bmatrix} V_p \\ V_n \end{bmatrix} - \begin{bmatrix} \frac{V_s}{R} \\ -\frac{V_s}{R} \end{bmatrix} \quad \text{(eq. 11)}$$

MNA STAMP OF DC MOTOR

The DC motor (figure 5) will also have an MNA stamp with its own signature. The MNA stamp will be related to the equations (1-5)above. However, these equations need to be discretized since they were given in continuous time. Backward Euler numerical integration method was used to discretize the equations. This is necessary since the tools used in this project operate in discrete time.

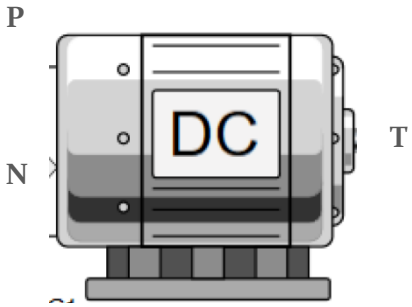

Figure 5. MNA block of DC motor with labeled MNA ports P,N, and T.

Continuous-time to discrete-time

The mathematical description of the DC motor and the load are given in continuous time. Hence it is necessary to discretize them first. Backward Euler equation is given below. In discrete time, the next value is equal to the current value in addition to the time step multiplied by the derivative of the next value.

$$x[n+1] = x[n] + \Delta t(\dot{x}[n+1]) \qquad \text{(eq .12)}$$

By applying backward Euler (equation 12) to the motor's equation (equation 5),

$$\frac{di}{dt} = \frac{1}{L}V(t) - \frac{1}{L}i(t)R - k_m\omega_m(t) \qquad \text{(eq .13.1)}$$

$$I[n+1] = I[n] + \Delta t * \dot{I}[n+1] \qquad \text{(eq. 13.2)}$$

$$I[n+1] = I[n] + \frac{\Delta t}{L} * (V[n+1] - R(I[n+1] - K_m\omega[n+1]) \qquad \text{(eq. 13.3)}$$

$$I[n+1] = \frac{L}{L+R*\Delta t} * I[n] + \frac{\Delta t}{L+R\Delta t}V[n+1] - \frac{K_m\Delta t}{L+R\Delta t}\omega[n+1] \qquad \text{(eq. 13.4)}$$

Equation 13.4 provide a discrete-time description of the through vector. From this equation, a expression for the through and across values were derived in terms of their respective ports (P,N).

$$I_p[n+1] = \frac{L}{L+R*\Delta t} * I_p[n] + \frac{\Delta t}{L+R\Delta t}(V_p[n+1] - V_N[n+1]) - \frac{K_m\Delta t}{L+R\Delta t}\omega[n+1] \qquad \text{(eq. 13.5)}$$

$$I_N[n+1] = \frac{-L}{L+R*\Delta t} * I_p[n] - \frac{\Delta t}{L+R\Delta t}\left(V_p[n+1] - V_N[n+1]\right) + \frac{K_m\Delta t}{L+R\Delta t}\omega[n+1] \qquad \text{(eq. 13.6)}$$

The third port shown in figure 5 is T. The through expression of T is given in equation 4 in continuous time. The discretizing process is shown below.

$$T[n+1] = \frac{J+\Delta t}{\Delta t} B_m \omega_m[n+1] - k_m * I[n+1] - \frac{J}{\Delta t}\omega[n] \quad \text{(eq. 14.1)}$$

From current in equation 13.4,

$$T[n+1] = \frac{J+\Delta t}{\Delta t} B_m \omega_m - k_m \frac{L}{L+R*\Delta t} * I[n] - \frac{k_m \Delta t}{L+R\Delta t} V[n+1] + \frac{K_m^2 \Delta t}{L+R\Delta t}\omega[n+1]) - \frac{J}{\Delta t}\omega[n] \text{(eq. 14.2)}$$

Now, since all equations are in discrete time, once may develop the MNA stamp of the DC motor include the torque. The through values are I_p, I_N, and T. The across values are V_P, V_N, ω_m. The stamp is provided in equation 15.

$$\begin{bmatrix} I_P \\ I_N \\ T \end{bmatrix} = \begin{bmatrix} \frac{\Delta t}{L+R\Delta t} & \frac{-\Delta t}{L+R\Delta t} & \frac{-k_m \Delta t}{L+R\Delta t} \\ \frac{-\Delta t}{L+R\Delta t} & \frac{\Delta t}{L+R\Delta t} & \frac{k_m \Delta t}{L+R\Delta t} \\ \frac{-k_m \Delta t}{L+R\Delta t} & \frac{k_m \Delta t}{L+R\Delta t} & \frac{k_m^2 \Delta t}{L+R\Delta t} + \frac{J+B\Delta t}{\Delta t} \end{bmatrix} \begin{bmatrix} V_p \\ V_N \\ \omega_m \end{bmatrix} - \begin{bmatrix} \frac{-L}{L+R\Delta t} I_p[n] \\ \frac{L}{L+R\Delta t} I_p[n \\ \frac{1}{\Delta t}\omega[n+1] + \frac{k_m L}{L+R\Delta t} I_p \end{bmatrix} \quad \text{(eq. 15)}$$

MNA STAMP OF TORQUE LOAD

The torque load in figure 6 also require an MNA block since it affects the rotational velocity ω_m. This MNA component only have one port. Therefore, the through vector, across vector, intercept, and Jacobian will all be of size 1.

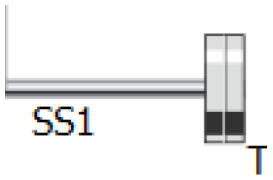

Figure 5. Torque load MNA block with only one port (S).

Using the given equation for the torque load in expression 16.1, MNA representation of the torque load. It is noted that the behavior of the ω_m is nonlinear. Hence one must find the partial derivative of the rotation velocity with respect to the across value. This is based on the generic formula provided in the appendix.

$$T_L(t) = K_d \omega^2(t) \quad \text{(eq.16.1)}$$

$$[T] = [2K_d \omega_m][\omega_m] - [K_d \omega_m^2] \quad \text{(eq.16.2)}$$

Control System and Simulation

After obtaining the transfer function of the plant (DC motor) one must design a Proportional Integral Derivative (PID) controller. In time domain, the expression of the PID controller is given in equation 17.1. Each term of the controller contribute to the step response characteristics (rise time, settling time, steady-state error, etc.) of the output at given e(t). Where e(t) is the difference between the actual output and the reference point or desired output.

$$y(t) = K_p e(t) + k_i \int_0^t x(t) + K_d \frac{de(t)}{dt} \quad \text{(eq. 17.1)}$$

To find an expression in s domain, take Laplace transform. This yield,

$$\frac{Y(s)}{E(s)} = \frac{K_d s^2 + K_p s + K_i}{s} \quad \text{(eq .17.2)}$$

The PID controller in s domain is given in the block diagram below.

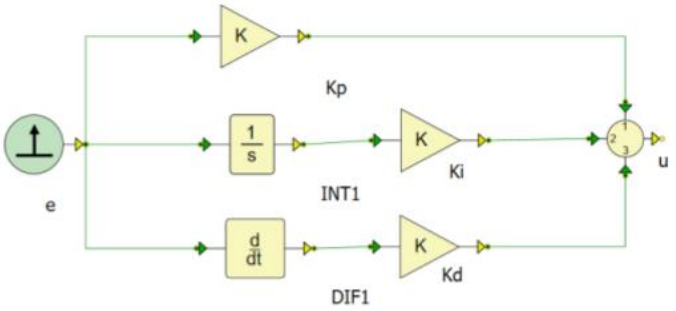

Figure 6. Block diagram of a PID controller

The design of the gains K_p, K_i, K_d require an accurate knowledge of the plant. There are various PID tuning methods that could be used. In this project, optimPID function in MATLAB was used to design the controller gains. The plant is the transfer function of the PMDC motor provided in equation 8. To design the closed loop gain, one must find the forward path of the control system. The feedback is assumed to be unity gain (H=1). The PID function generate optimal gain values based on the system order and coefficients. The MATLAB code is provided in the appendix. The Kp gain was found to be 0.6 Ki was 6.54. Kd was 0.017. This was under assumption of transfer function coefficients. MATLAB code provided in the appendix.

The control system was simulated in Simulink. The figure below shows the simulation block diagram that includes the reference point, PID controller, negative feedback, PWM generator, Plant of the DC motor and the load.

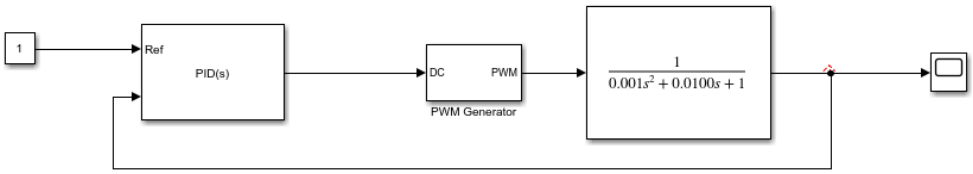

Figure 7. Simulink block diagram of the simulated block

The control system was simulated under the assumption of the transfer function coefficients and load. Those assumptions were used to evaluate the transfer functions parameters and to obtain the PID gains under the same assumption. The transfer function shown in the Simulink block diagram above include these assumptions of DC motor characteristics and load. The simulation results is shown in the figure below.

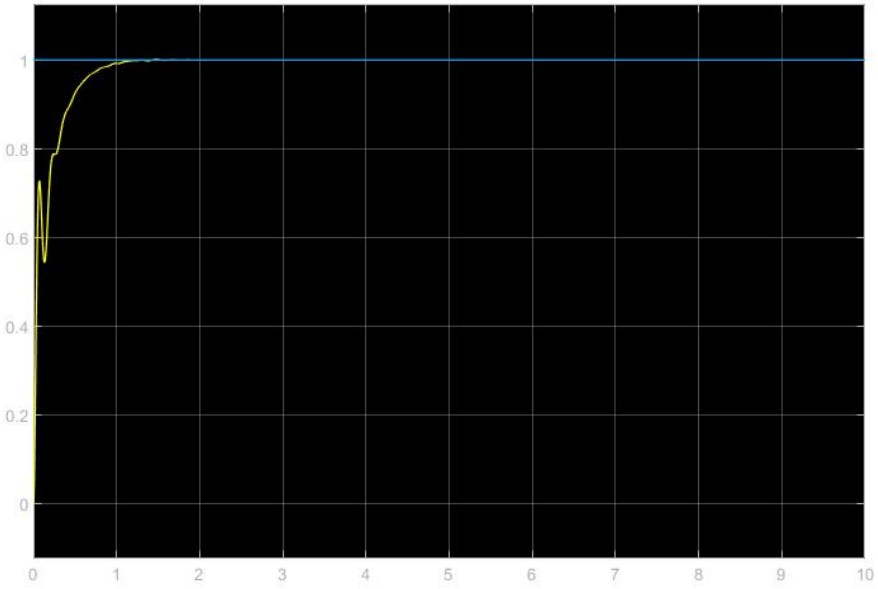

Figure 8. Simulink simulation results.

Description of C++ Simulation

The C++ code is an application of building a system matrix based on their individual MNA signatures or stamps. The independent blocks will be combined in the solver to find the across and through values of the system matrix. Along with MNA blocks, signal blocks are also implemented in c++ to simulate the PID controller, square wave generator, summing junctin , and the constant reference. The is done through a base class “Block” that signal blocks and MNA block virtually inherited from.

DEFENITION OF A BLOCK

A block is the base class for both MNA block and a signal block. It holds a number of parameters that determines the block’s size (how many inputs and outputs) and values of these parameters. This will act as a base (parent) class for the MNA and the signal blocks.

```
1.  Block::Block(size_t numberOfParameters) : _numberOfParameters(0), _parameters(nullptr)
2.  {
3.    _numberOfParameters = numberOfParameters;
4.  }
5.
6.  Block::~Block()
7.  {
8.    if (_parameters)
9.    {
10.             _parameters = nullptr;
11.             _numberOfParameters = 0;
12.   }
13. }
14.
15. void Block::setParameterValue(size_t index, double value)
16. {
17.   checkParameterIndex(index);
18.   _parameters = new double();
19.    (_parameters[index]) = value;
20. }
21.
```

DEFENITION OF A SIGNAL BLOCK

A signal block is a block with at least one output and input. An example of a signal block is the constant reference block. This block takes no inputs but only provides an output of the it parameter. The signal block contains two arrays of type input port and output port. When the a signal block is constructed, the number of input ports and output ports can be determined through the constructor argument.

```
1.  Block::Block(size_t numberOfInputPorts, size_t numberOfOutputPorts)
2.    : _numberOfInputPorts(0),
3.    _numberOfOutputPorts(0),
4.    _inputPorts(nullptr),
5.    _outputPorts(nullptr)
6.  {
7.    _inputPorts = new InputPort[numberOfInputPorts];
8.    _outputPorts = new OutputPort[numberOfOutputPorts];
```

```
9.    _numberOfInputPorts = numberOfInputPorts;
10.   _numberOfOutputPorts = numberOfOutputPorts;
11.
12. }
13.
```

INPUTS AND OUTPUTS PORTS

The input and output ports of a signal block are array pointers of types input port output port respectively. They inherit form class port which has a getter and a setter of the port's value. The input port class has a connect function, copy value, and dependencies check function. The input port also contains an array pointer of type output port. Whenever a signal block is connected to another block, it's output is the input if the second block. Hence, the input port value cannot be set directly but instead it will be copied form the connected port

```
1.  void InputPort::connect(const OutputPort& port)
2.  {
3.      _connectedPort = &port; // Set the connection
4.      copyValue(); // import values form connected port
5.  }
6.  void InputPort::copyValue() {
7.
8.      // Set the value of the input port to the value of the output port
9.      if (areDependenciesSatisfied()); // if we are ready
10.     {
11.         this->Port::setValue(_connectedPort->getValue());
12.     }
```

DEFENITION OF AN MNA BLOCK

An MNA block inherits from base class block and it includes the intercept and Jacobian stamps of the block's component. The block also has a set of ports that can be connected to other nodes in the system.

PORTS

An MNA port hold data of the across and through values at that port. It has an index that gives it a unique label in the block, and a true or false statement that determine if the port is grounded. The port also has a shared pointer of node.

NODES

An MNA node has a unique index in the system and it has a set ports that represent the ports connected to it. Along with setters and getters of the across and index values, the node has a special function called joinNode. In the join node method two nodes become one node. All the ports form the previous node are now connected to the current node.

```
1.  void Node::joinNode(std::shared_ptr<Node>& oldNode) {
2.
3.    for (auto aPort : oldNode->_ports) {
4.              //aPort->setNode(*this);
5.              this->_ports.insert(aPort);
6.    }
```

```
7.    oldNode->setIndex(this->getIndex());
8.  };
9.
```

FUNTIONS

Other function of an MNA block are getters and setters of Jacobian matrix, intercept, and a computer through function. The compute through function solves for the through value for each port based on equation 9.

```
1.  void Block::computeThroughValues()
2.  {
3.      Math::Vector temp(_jacobian.getNumberOfRows(),0.0);
4.      for (size_t i = 0; i < _jacobian.getNumberOfRows(); i++)
5.          temp[i] = _ports[i].getAcross();
6.      for (size_t i = 0; i < _jacobian.getNumberOfRows(); i++) {
7.          double through =+ _jacobian.getRow(i).dotProduct(temp) - _intercept[i];
8.          _ports[i].setThrough(through);
9.      }
10. }
11.
```

SIGNAL AND MNA SOLVERS

The Signal solver is responsible of connecting to signal blocks together and update their input and output on each time step. This is done by calling the connect function discussed previously, and then adding the block to a set. The blocks in the set can be called on each time step to update their signals.

```
1.  void Solver::connect(const Block& outputBlock, size_t outputPortIndex,
2.                       Block& inputBlock, size_t inputPortIndex)
3.  {
4.      _blocks.insert(&inputBlock);
5.      inputBlock.connect(inputPortIndex, outputBlock, outputPortIndex);
6.  }
7.  void Solver::step()
8.  {
9.      for (auto ablock : _blocks)
10.     {
11.         ablock->signalStep(_timeStep, _time);
12.     }
13.     _time =_time + _timeStep;
14. }
15.
```

Similarly, the MNA solver steps all the blocks in the set, and build the overall system matrix by collecting the stamps of independent MNA blocks in the set. It then solves for the across value at every node and update the across of ports connected to each node.

Problems encountered

The developer encountered problems with the MNA solver that impeded the delivery of simulation results by the required deadline. The problems were 1) failure to locate the grounded ports in the overall system matrix to delete the row and column from the Jacobian matrix and intercept matrix 2) failure to find the derivative of the error term in Newton-Raphson method 3) inability to build the .cpp of the controller voltage source MNA block by the deadline.

Expected Differences in C++ and Simulink Simulations

Since both simulations are conducted in discrete-time, the step response characteristics would be the same as long as the motor's parameters and controller gains are equal. The time step in both simulations need to be equal as well for a fair comparison. Due to memory allocation, the results of the C++ simulation is expected to generate quicker. This is also due to Simulink running processes in the background that slows it down.

Appendix

Generic formula of an MNA stamp

$$\begin{bmatrix} I_p \\ I_N \end{bmatrix} = \begin{bmatrix} \frac{\partial I_P}{\partial V_P} & \frac{\partial I_P}{\partial V_N} \\ \frac{\partial I_N}{\partial V_P} & \frac{\partial I_N}{\partial V_N} \end{bmatrix} \begin{bmatrix} V_P \\ V_N \end{bmatrix} - \begin{bmatrix} \frac{\partial I_P}{\partial V_P}.V_P + \frac{\partial I_P}{\partial V_N}.V_N - I_P \\ \frac{\partial I_N}{\partial V_P}.V_P + \frac{\partial I_P}{\partial V_N}.V_N - I_N \end{bmatrix}$$

MATLAB code

```
J = 1.0;
L = 0.001;
R = 0.01;
b = 0.01;
num = [K_m];
denom = [J*L (J*R+b*L) K_m^2]
G_motor = tf(num,denom,'InputName','u', 'OutputName', 'delta')
G_Feedback = tf(1,...
    'InputName','y', 'OutputName', 'y_d'); %Feedback path gain
G_forward = G_Feedback*G_motor
G_c = optimPID(G_forward(1),3,4)
G_c.InputName = 'e';
G_c.OutputName = 'u';
G_open = G_forward*G_c

```