Nonlinear Control System – Final Project

The Cascaded PID-PI Controller for Speed and Current Regulation of DC Motor with Ziegler-Nichols Stability Limit Tuning Method

Bramantio Yuwono, EPIC 2022-2023, 15 June 2023

This work presents the design and implementation of a cascade Proportional-Integral-Derivative (PID) and Proportional-Integral (PI) controller for simultaneous control of motor speed and armature current in a DC motor. The Ziegler-Nichols stability limit tuning method is used to determine the initial controller gains. The cascade control strategy allows for improved performance by decoupling the control loops and addressing the interaction between speed and current control. The effectiveness of the proposed controller is evaluated through Software-in-the-loop via serial communication between a PC and Arduino Uno.

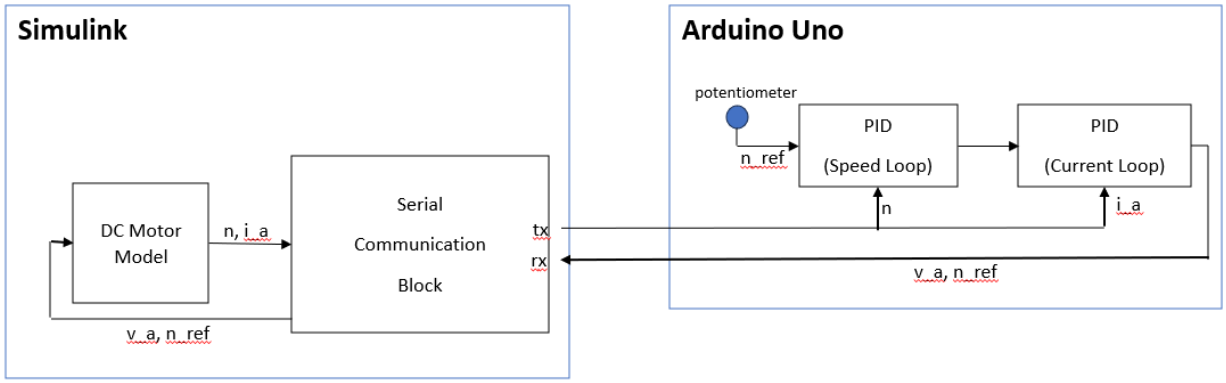

Figure 1 Software-in-the-loop diagram

The DC Motor is modelled in Simulink while the cascaded PID-PI controller is implemented on Arduino Uno. The communication between the Simulink DC motor model and Arduino is ensured by utilizing serial communication. The overview of the Software-in-the-loop system is shown in the figure above.

1. DC Motor Modelling

The accurate modelling of a DC motor is crucial for understanding its dynamic behaviour and designing effective control strategies. This section will jump into the mathematical derivation and explanation of the DC motor model. By capturing the electrical and mechanical dynamics of the motor, we aim to develop a comprehensive representation that enables accurate prediction of its response to different inputs and operating conditions.

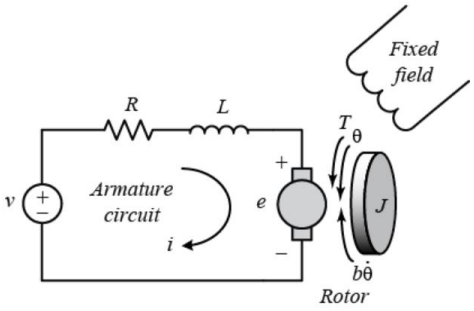

Figure 2 DC Motor Modelling

The DC motor model consists of two primary components: the electrical and mechanical subsystems. The electrical subsystem describes the relationship between the motor's input voltage and current, while the mechanical subsystem characterizes the motor's rotational dynamics, including speed, torque, and inertia. The mechanical(1) and electrical(2) subsystem equations are presented below.

$$J\dot{\omega} + b\omega + T_L = K_t i = T_e \quad \text{...(1)}$$

$$L\dot{i} + Ri = v - K_e\omega = v - e \quad \text{...(2)}$$

The mechanical subsystem accounts for factors such as rotor inertia, friction, and load torque. By applying Newton's laws of motion and the principles of rotational dynamics, equation (1) is obtained. It describes the motor's torque-speed relationship and its response to external loads. The electrical subsystem involves the armature circuit, which comprises the armature resistance, inductance, and back electromotive force (EMF). By considering Kirchhoff's voltage law and the basic principles of electromagnetism, equation (2) that govern the electrical behaviour of the motor is derived. This equation will give an understanding of the interaction between the armature current and the applied voltage.

$$\dot{\omega} = \frac{K_t}{J} i - \frac{b}{J}\omega - \frac{T_L}{J} \quad \text{...(3)}$$

$$\dot{i} = \frac{1}{L} v - \frac{K_e}{L}\omega - \frac{R}{L} i \quad \text{...(4)}$$

Equations (1) and (2) can be implemented by taking the derivative term to one side and integrating all the other terms as shown above. The block implementation is shown in the figure below.

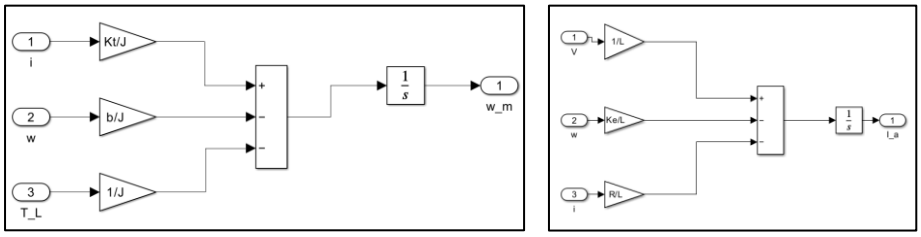

Figure 3 Implementation of mechanical equation(left) and electrical equation(right) of DC motor on Simulink

With the electrical and mechanical subsystems implemented on Simulink, we will combine them to form a comprehensive DC motor model. This integrated model will enable us to capture the interplay between the electrical and mechanical dynamics, providing a holistic understanding of the motor's behavior. The whole DC motor model on Simulink is shown below.

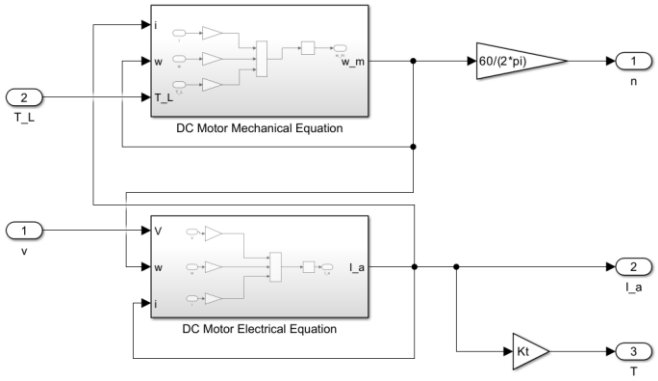

Figure 4 The complete DC motor model on Simulink

In the software-in-the-loop system, the model of the DC motor should be connected to the serial block of Simulink to ensure the data transfer from the model to the Arduino Uno. The implementation of software-in-the-loop system and the serial block is the following.

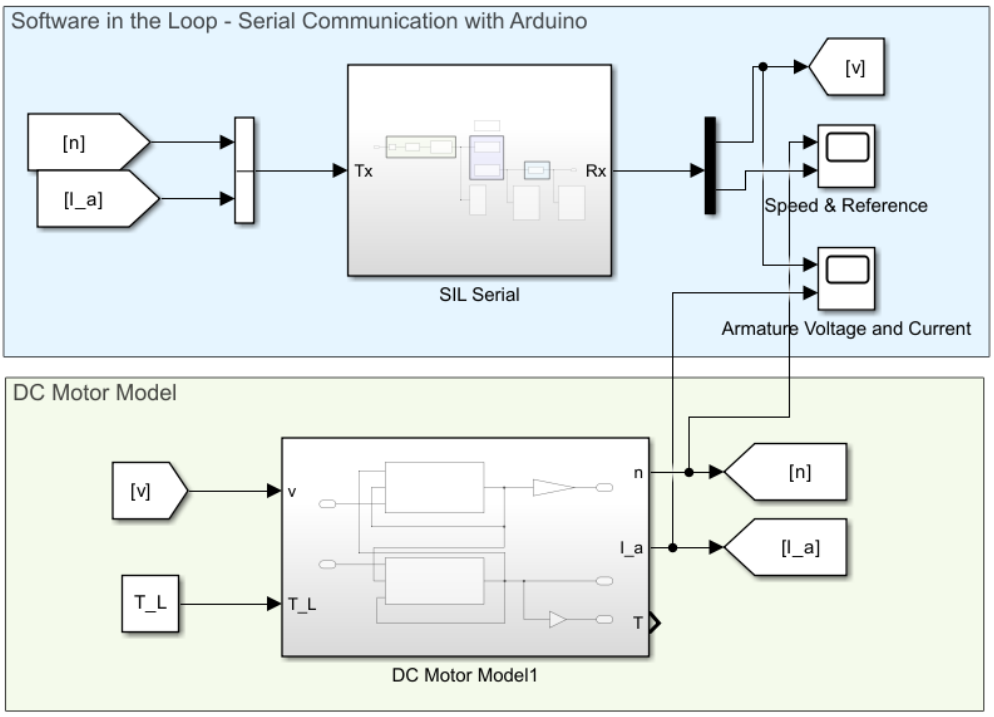

Figure 5 the software-in-the-loop implementation on Simulink

The parameter values used in this work is summarized in the following table.

Table 1 DC Motor Model Parameter Values

No	Parameters	Value	Unit	Remark
1	J	0.01	kg.m^2	Moment inertia of the rotor
2	B	0.1	N.m.s	Motor viscous friction constant
3	Ke	0.367	V/rad/sec	EMF constant
4	Kt	0.367	N.m/Amp	Motor torque constant
5	R	0.1	ohm	Armature resistance
6	L	0.5	H	Armature coil inductance
7	T_L	0	N.m	Load torque
8	vb	0	V	Voltage drop in brush

To determine the required sampling time for the cascaded PID-PI controllers, the step response time should be measured and that value will be the sampling time of the controllers. It can be observed from the figure below that the measured response time is around 1 second when a step-signal(12V) is applied to the DC motor model. Hence, we can set the sampling time to the same value of the step response time.

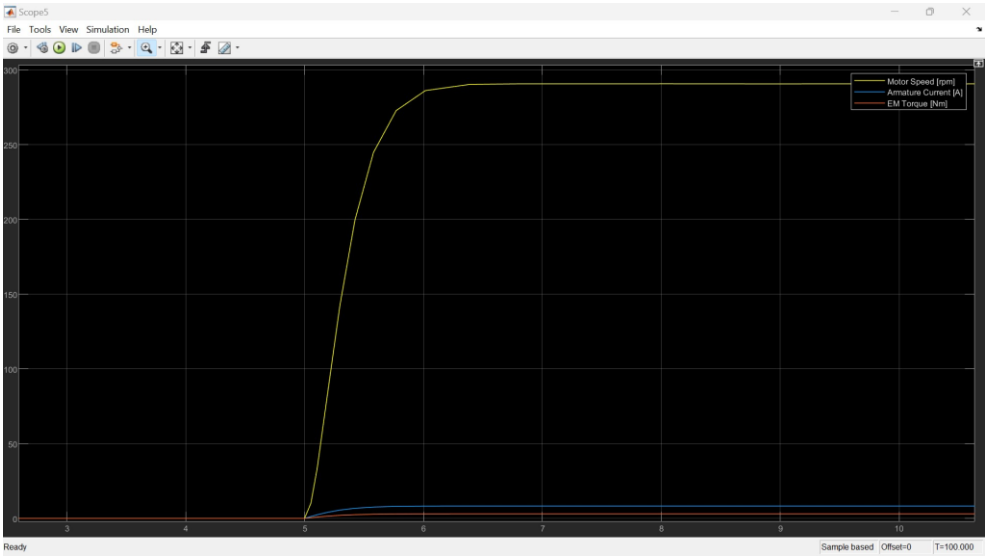

Figure 6 Step response of the DC motor model

2. PID Controller Tuning: Ziegler-Nichols Stability Limit Method

PID (Proportional-Integral-Derivative) control is a widely used and versatile control algorithm employed in various industries and applications. It is a feedback control mechanism that aims to regulate an output of a system by continuously adjusting a control variable based on the error between the desired setpoint and the actual measured value. The PID controller consists of three main components: the proportional term, which provides a control action proportional to the error; the integral term, which integrates the accumulated error over time to eliminate steady-state error; and the derivative term, which accounts for the rate of change of the error to improve transient response and stability. By intelligently combining these three components, PID control offers a robust and adaptable solution for a wide range of control problems, enabling accurate and efficient regulation of systems in the presence of disturbances, nonlinearities, and uncertainties.

The command given by a PID controller is explained in the following equation. In this work, constant Kr, Ti, and Td of equation (5) is tuned and adjusted during tuning process, while in the implementation of PID calculation on Arduino Uno, equation (6) is used.

$$u_k = K_r.\left[\varepsilon_k + \frac{T}{T_i}\sum_{i=0}^{k}\varepsilon_i + \frac{T_d}{T}(\varepsilon_k - \varepsilon_{k-1})\right] \quad \ldots(5)$$

$$u_k = K_p\varepsilon_k + K_i\sum_{i=0}^{k}\varepsilon_i + K_d(\varepsilon_k - \varepsilon_{k-1}) \quad \ldots(6)$$

One of the well-known PID tuning methods is the Ziegler-Nichols Stability Limit method. This method is a popular and widely used heuristic approach for tuning PID controllers. It provides a systematic procedure to determine the Kr, Ti, and Td constants based on the system's stability characteristics. The Ziegler-Nichols method is primarily focused on achieving system stability while sacrificing some performance aspects like overshoot and settling time. The tuning process is done in the following way:

1. Start by disconnect proportional, integral, and derivative term from the model.
2. Increase Kr until the system exhibit sustained oscillation. The obtained Kr is the critical gain value (Ko)
3. Measure also the period of the sustained oscillation (To)
4. The Ziegler-Nichols method provides different tuning rules depending on the control response :
   - P controller:
     Kr = 0.5*Ko
   - PI controller:
     Kr = 0.4*Ko
     Ti = 0.8*To
   - PID controller:
     Kr = 0.6*Ko
     Ti = 0.5*To
     Td = 0.12*To

It's important to note that while the Ziegler-Nichols method provides a starting point for tuning PID controllers, it may not always yield optimal or robust performance for all systems. The method tends to result in overshoot and slower response times due to the conservative tuning.

The tuning procedure for the cascaded PID-PI controllers begins by initially tuning the inner loop, which controls the armature current. This sequential approach ensures that the inner loop achieves a stable response, minimizing disturbance to the outer loop. Subsequently, the outer loop, responsible for motor speed control, is tuned. Both PID controllers are tuned using the Ziegler-Nichols stability limit method, as mentioned earlier.

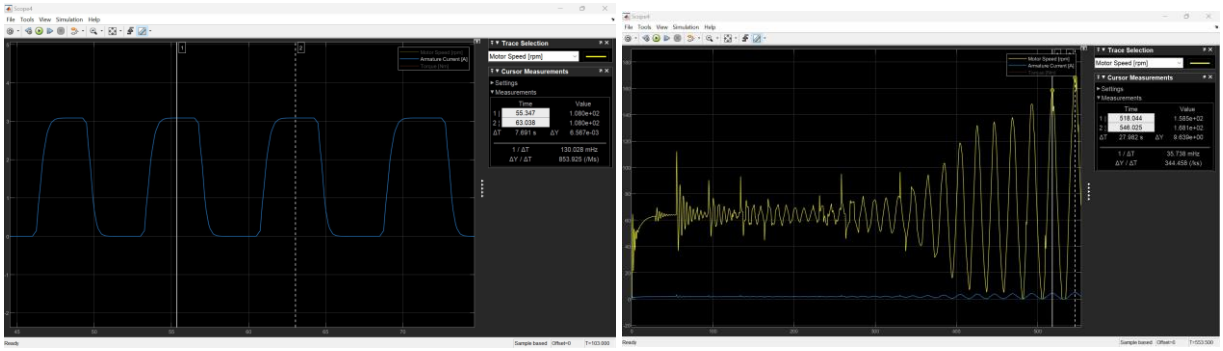

Figure 7 Oscillation responses for inner loop(left) and outer loop(right)

The figure above presents the obtained oscillation response when Ko=2.23 and To=7.691 for the inner loop and Ko=50.5e-3 and To=27.982. With these values, we can calculate Kr, Ti, and Td based on the Ziegler-Nichols rule.

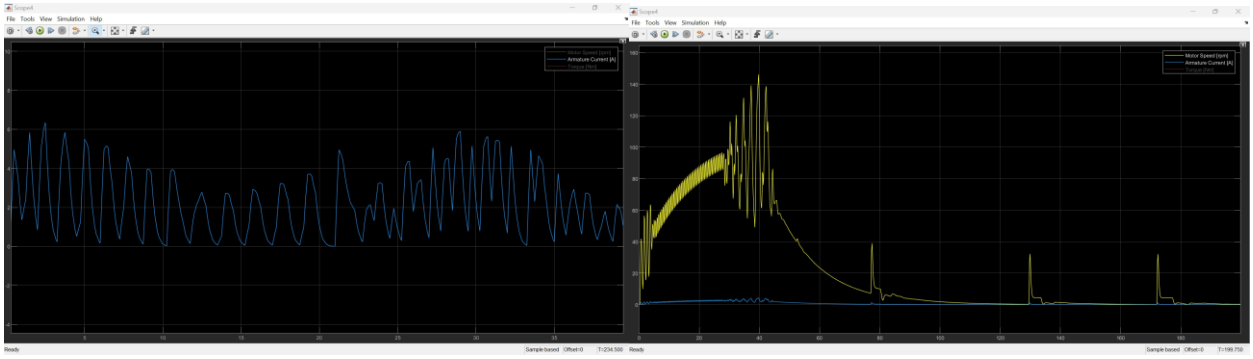

Figure 8 Response of inner loop(left) and outer loop(right) with Kr, Ti, Td values conformed with the Ziegler-Nichols rule

The Ziegler-Nichols rule provides values for Kr, Ti, and Td that result in an unstable response with significant oscillations. These values do not meet the criteria for a robust and stable controller for a DC motor. Therefore, it is necessary to adjust these Ziegler-Nichols rule-conformed values. The adjusted Kr, Ti, and Td values for both loops are presented below.

Table 2 Adjusted Kr, Ti, and Td values for inner loop and outer loop

The inner loop (Armature Current Controller)		The outer loop (Motor Speed Controller)	
Constants	Value	Constants	Value
Kr	0.446	Kr	2.50E-03
Ti	3.8455	Ti	1.3991
Td	0	Td	0.083946

3. The Cascaded PID-PI Controllers Implementation Result

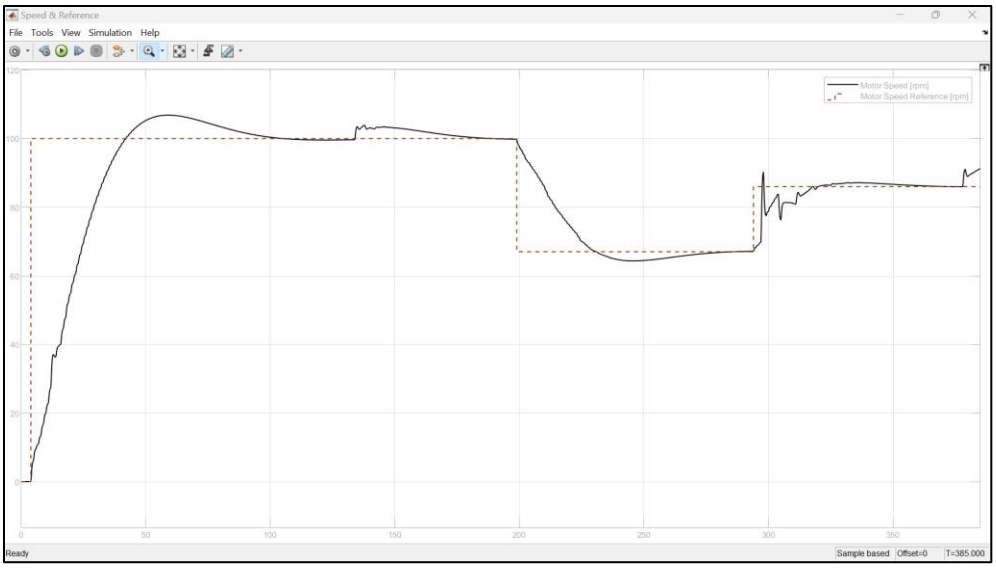

Figure 9 The response of motor speed with adjusted Kr, Ti, and Td values for both loops

Figure 9 shows that the cascaded PID-PI controllers yield a robust and stable response. The controllers are able to handle small ripples occurred. It shows also that the controllers can tract the reference change, although with a considerable amount of overshoot. However, it comes with the trade-off: slow response time (around 40s).

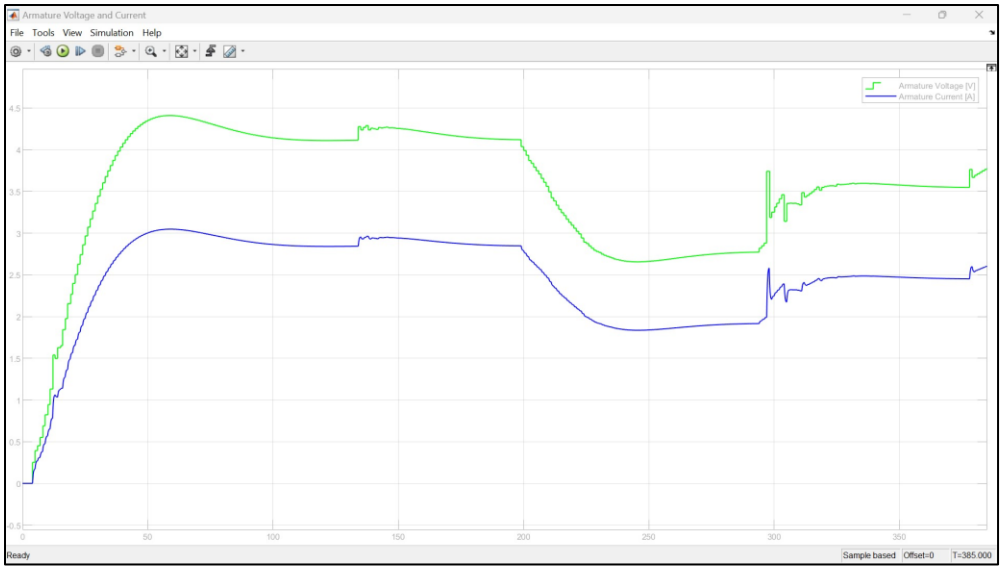

Figure 10 The armature voltage and current with adjusted Kr, Ti, and Td values for both loops

From the figure above, it can be observed that the controllers can regulate the armature current very well. The armature current and voltage never reach the imposed limit, 6A and 12V respectively. It is true even with the motor speed reference changes occur during the observation.

4. Conclusion

From this work, the following key points can be highlighted:

1. The tuning process for cascaded PID-PI controllers has to start from the inner loop that has a faster response time so it ensures the stability of the response of the inner loop. Afterwards, with the obtained Kr, Ti, Td values of the inner loop, the tuning process of the outer loop can be done.
2. The Ziegler-Nichols method can be used as a starting point for the tuning process. Rule-based constant values are not necessarily producing a good response so adjustments to the constant values are still required.
3. The cascaded PID-PI configuration performs really well in terms of stability and robustness for DC motor applications. However, it produces a really slow response.
4. The software-in-the-loop setup can be used for verifying the control algorithm implemented on the microcontrollers. It can prevent the possibilities of control algorithm giving commands that can harm the real actuators/plants.
5. Due to the bottleneck of slow serial communication, the software-in-the-loop using serial communications can only be utilized with process that has slow response time.

APPENDIX A –ARDUINO CODE

Serial Communication from/to Simulink

FLOATUNION_t to store variablesto be received/transmitted

```
//Union for sending/receiving float to/from Simulink
typedef union {
  float fval;
  uint8_t bytes[4];
} FLOATUNION_t;

FLOATUNION_t yk1, yk2, rk, uk;
```

readFromMatlab() to read two float data from Simulink

```
void readFromMatlab( FLOATUNION_t* f1, FLOATUNION_t* f2 ){
  int count = 0;
  FLOATUNION_t f;
  bool allReceived = false;
  uint8_t buffer;
  int idx = 0;

  // read the incoming bytes:
  int rlen = Serial.readBytesUntil('\n', buff, BUFFER_SIZE);

  for( int i=0;i<4;i++){
    f1->bytes[i]=buff[i];
  }
  for( int i=4;i<8;i++){
    f2->bytes[i-4]=buff[i];
  }
}
```

writeToMatlab() to write two float data to Simulink

```
void writeToMatlab( FLOATUNION_t fnumber, FLOATUNION_t fnumber2 ) {

  // Print header: Important to avoid sync errors!
  Serial.write('A');

  // Send each byte of uk and rk respectively
  for (int i=0; i<4; i++){
      Serial.write(fnumber.bytes[i]);
  }
  for (int i=0; i<4; i++){
      Serial.write(fnumber2.bytes[i]);
  }

  // Delimiter
  Serial.print('\n');

}
```

PID Control

struct PID to construct a PID loop component

```
//Struct for PID Controller Component
struct PID{
  float kp;
  float ki;
  float kd;
  float rk;
  float yk;
  float uk;
  float uk_1;
  float e;
  float e_sum;
  float e_1;
};

struct PID n; // PID for motor speed control
struct PID i; // PID for armature current control
```

Initialize_pid_param() to initialize PID parameters based on the given Kp, Ti, Td and sampling time T

```
void initialize_pid_param( struct PID* pid, float p, float i, float d, float T )
{
  // Initialize all PID component parameters
  // Set Kp = Kr
  pid->kp     = p;

  // Zero-value Ti protection
  // Set Ki = Kp*T/Ti if Ti is not 0
  if (i != 0.0)
    pid->ki   = p*T/i;
  else
    pid->ki   = 0;

  // Set Kd = Kp*Td/T
  pid->kd     = p*d/T;

  // Set the remaining PID parameters to 0.0
  pid->rk     = 0.0;
  pid->yk     = 0.0;
  pid->uk     = 0.0;
  pid->uk_1   = 0.0;
  pid->e_sum  = 0.0;
  pid->e_1    = 0.0;

}
```

PID_control() to update the PID parameters based on the obtained yk, ek, imposed yk limit, and sampling time

```
void PID_control( struct PID* pid, float ref, float out, float lim_top, float lim_btm, float dt )
{
  // Assign the reference value and output value from the model
  pid->rk = ref;
  pid->yk = out;

  // Calculate the error
  float error = ref - out;
  pid->e = error;

  // Calculate the proportional term
  float proportional = pid->kp * error;

  // Calculate the integral term
  pid->e_sum += error;
  float integral = pid->ki * pid->e_sum;

  // Calculate the derivative term
  float derivative = pid->kd * (error - pid->e_1);

  // Calculate the command output
  pid->uk = proportional + integral + derivative;

  // Command Limit
  if(pid->uk >= lim_top)
    pid->uk = lim_top;
  if(pid->uk <= lim_btm)
    pid->uk = lim_btm;

  // PID memory update
  pid->uk_1 = pid->uk;
  pid->e_1 = error;

}
```

Overall Code

setup()

```
void setup() {

  // Set up serial communication
  Serial.begin(115200);

  // Initialize motor speed PID loop component
  initialize_pid_param( &n, kr_n, ti_n, td_n, sample_time );

  // Initialize armature current PID loop component
  initialize_pid_param( &i, kr_i, ti_i, td_i, sample_time );

  // Set motor speed reference to 0 initially
  n.rk = 0.0;

  // Capture start timing for reference reading with potentiometer
  prev_time = millis();

}
```

loop()

```
void loop() {

  // Read potentiometer voltage every T_REF_UPDATE
  if ( millis() - prev_time > T_REF_UPDATE ) {
    potVal = analogRead(analogInPin);
    n.rk = map(potVal, 0, 1023, 0, N_MAX);
    prev_time = millis();
  }

  readFromMatlab(&yk1, &yk2);

  // Assigned the received data
  n.yk = yk1.fval;
  i.yk = yk2.fval;

  //---------------------------------------------------------//
  // PID Control for DC Motor Speed Loop
  //---------------------------------------------------------//
  PID_control( &n, n.rk, n.yk, I_MAX, I_MIN, sample_time );

  //---------------------------------------------------------//
  // PID Control for DC Motor Armature Current Loop
  //---------------------------------------------------------//
  PID_control( &i, n.uk, i.yk, V_MAX, V_MIN, sample_time );

  // assign calculated armature voltage to uk and reference speed to rk
  uk.fval = (float) i.uk;
  rk.fval = (float) n.rk;

  // Transmit uk and rk via serial to SIL
  writeToMatlab(uk, rk);

  // Delay for sampling time
  delay((int)sample_time*1000);

}
```