MATLAB:

Part 1: Motor Control Algorithm

Task: Develop a MATLAB function to simulate/model any relevant motor in an EV system.

To simulate or model a motor for an Electric Vehicle (EV) system, let's choose a **Permanent Magnet Synchronous Motor (PMSM)**, as it is commonly used in EVs for its high efficiency and power density. Below is a MATLAB implementation for simulating the motor's behavior under varying load conditions.

MATLAB Function: Motor Control Algorithm

/MATLAB Drive/motorControlEV.m

```
function motorControlEV(sim_time, load_torque, voltage_input)


    J = 0.01;  % Moment of inertia of the rotor (kg.m^2)
    b = 0.001; % Motor viscous friction constant (N.m.s)
    Kt = 0.01; % Motor torque constant (N.m/A)
    Ke = 0.01; % Back EMF constant (V.s/rad)
    R = 1;     % Electric resistance (Ohms)
    L = 0.5;   % Electric inductance (H)

    % Simulation parameters
    dt = 0.001;                      % Time step (seconds)
    num_steps = round(sim_time / dt);% Number of simulation steps
    t = linspace(0, sim_time, num_steps); % Time vector

    % Initialize variables
    omega = zeros(1, num_steps);  % Angular velocity (rad/s)
    current = zeros(1, num_steps); % Current (A)
    torque = zeros(1, num_steps);  % Torque (N.m)


    for i = 2:num_steps
        % Calculate back EMF
        back_emf = Ke * omega(i-1);

        % Update current using differential equation
        dI_dt = (voltage_input - back_emf - R * current(i-1)) / L;
        current(i) = current(i-1) + dI_dt * dt;

        % Calculate motor torque
        torque(i) = Kt * current(i);

        % Update angular velocity using differential equation
```

/MATLAB Drive/motorControlEV.m

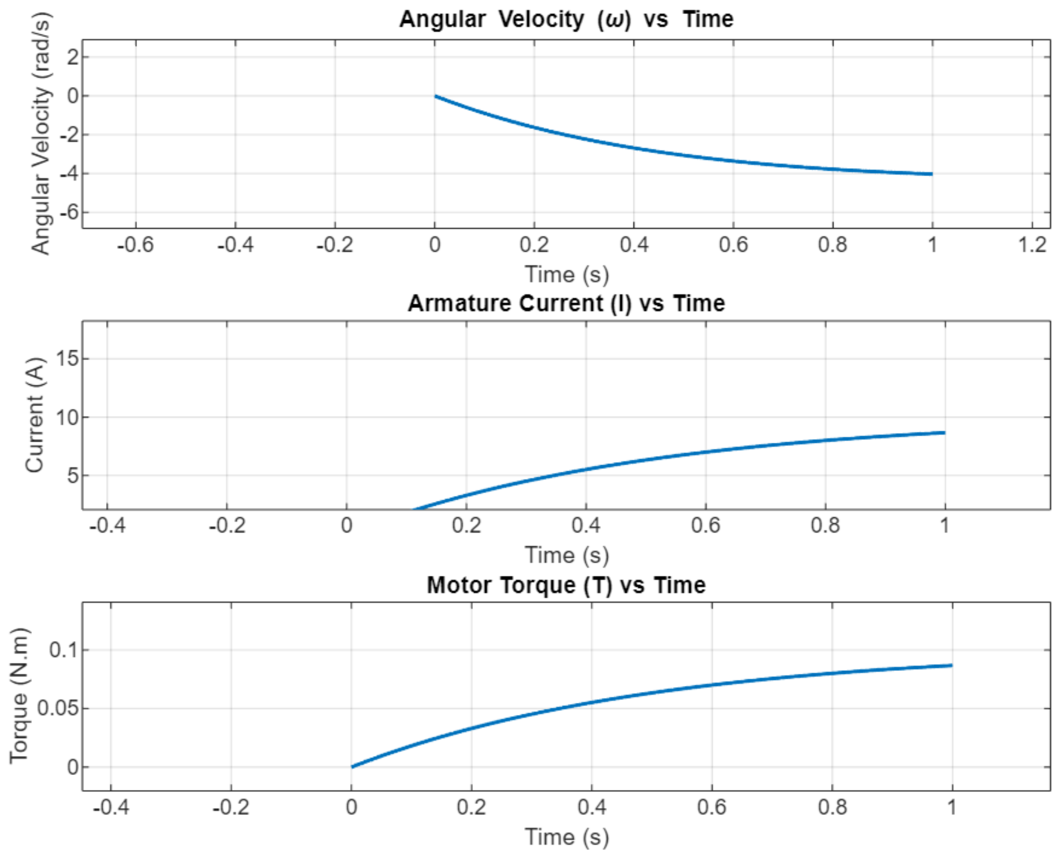
```matlab
            % Calculate motor torque
            torque(i) = Kt * current(i);

            % Update angular velocity using differential equation
            dOmega_dt = (torque(i) - load_torque - b * omega(i-1)) / J;
            omega(i) = omega(i-1) + dOmega_dt * dt;
        end

        % Plot results
        figure;
        subplot(3, 1, 1);
        plot(t, omega, 'LineWidth', 1.5);
        title('Angular Velocity (\omega) vs Time');
        xlabel('Time (s)');
        ylabel('Angular Velocity (rad/s)');
        grid on;

        subplot(3, 1, 2);
        plot(t, current, 'LineWidth', 1.5);
        title('Armature Current (I) vs Time');
        xlabel('Time (s)');
        ylabel('Current (A)');
        grid on;

        subplot(3, 1, 3);
        plot(t, torque, 'LineWidth', 1.5);
        title('Motor Torque (T) vs Time');
        xlabel('Time (s)');
        ylabel('Torque (N.m)');
        grid on;
    end
```

1.motorControlEV(1, 0.1, 10);

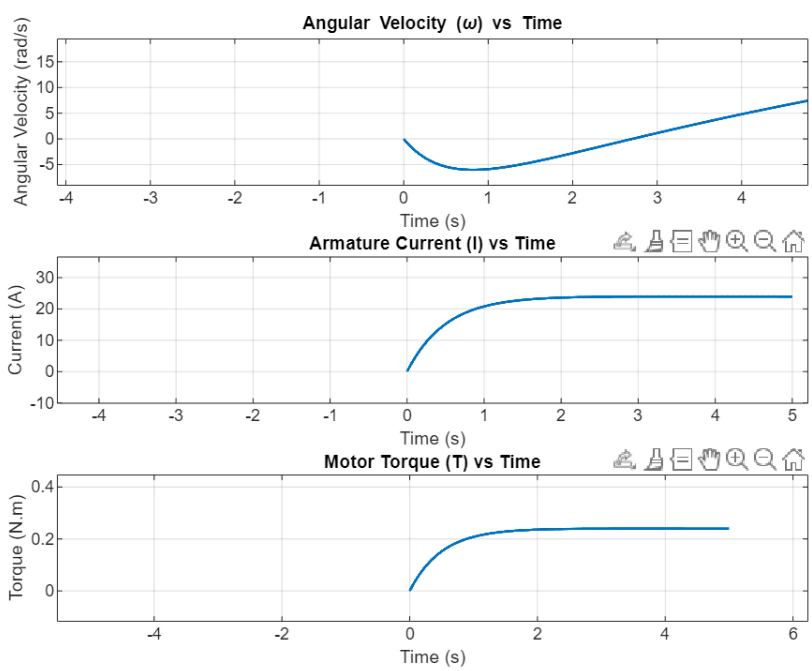
Command History

Search

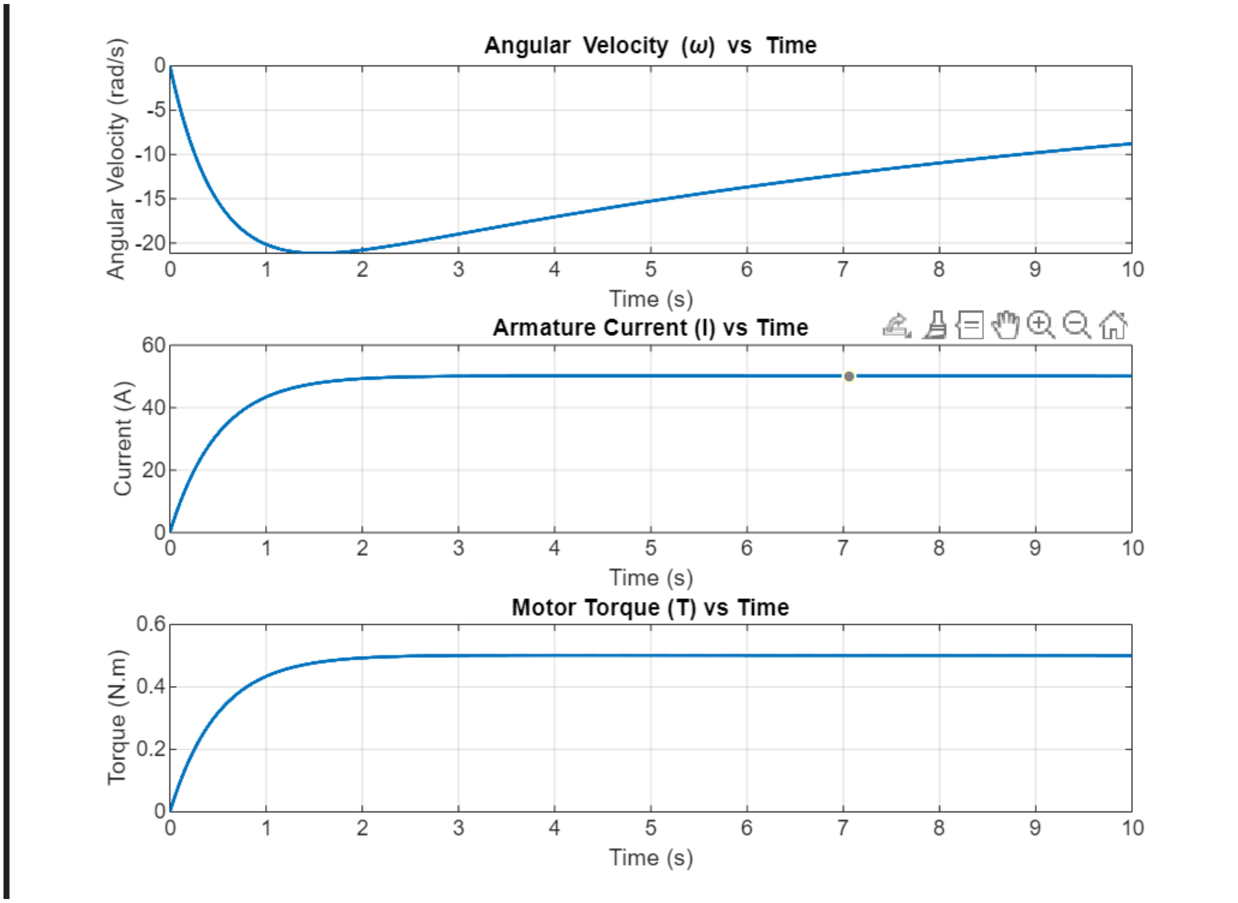
```matlab
motorControlEV(5, 0.2, 24);
motorControlEV(10, 0.5, 50);
motorControlEV(1, 0.1, 10);
new
untitled2
impz(cooling_power(end,end),'ctf'…
ylabel("cooling_power(end,end)");
title("cooling_power(end,end)");
legend("show");
2x boxchart(cooling_power(end,end));
ylabel("cooling_power(end,end)");
title("cooling_power(end,end)");
legend("show");
motorControlEV(5, 0.2, 24);
motorControlEV(1, 0.1, 10);
motorControlEV(10, 0.5, 50);
%-- 20/12/2024 11:27 pm --%
```



2.motorControlEV(5, 0.2, 24);



3.motorControlEV(10, 0.5, 50);



Part 2: **Thermal System Simulation**

Task: Write a MATLAB script to simulate/model the thermal management system of an EV

The thermal management system of an electric vehicle (EV) is crucial for maintaining optimal battery performance and longevity. It involves managing heat generation and dissipation to ensure the battery operates within a safe temperature range.

EV THERMAL SYSTEM

tempcheck.m × | motorControlEV.m × | untitled.m × | Figure 3 × | Figure 1 × | Figure 5 × | Figure 2 × | Figure 4

MATLAB Drive/tempcheck.m

```
% EV Thermal Management System Simulation

% Parameters
sim_time = 300;          % Total simulation time in seconds
dt = 1;                  % Time step in seconds
num_steps = sim_time / dt;

% Battery Parameters
battery_capacity = 50;   % Battery capacity in kWh
battery_efficiency = 0.9; % Efficiency of the battery
heat_gen_rate = 5;       % Heat generation rate (W per kW of load)

% Cooling System Parameters
cooling_power = 2000;    % Cooling system power in watts
ambient_temp = 25;       % Ambient temperature in °C
heat_transfer_coeff = 0.05; % Heat transfer coefficient (W/°C)

% Initial Conditions
battery_temp = 40;       % Initial battery temperature in °C
temperature = zeros(1, num_steps); % Array to store temperature over time
time = linspace(0, sim_time, num_steps); % Time vector

% Simulation Loop
for i = 1:num_steps
    % Heat generated by the battery
    heat_generated = battery_capacity * 1000 * (1 - battery_efficiency) * heat_gen_rate;

    % Heat removed by the cooling system
    heat_removed = cooling_power * heat_transfer_coeff * (battery_temp - ambient_temp);

    % Net heat in the system
    net_heat = heat_generated - heat_removed;
```

tempcheck.m × | motorControlEV.m × | untitled.m × | Figure 3 × | Figure 1 × | Figure 5 × | Figure 2 × | Figure 4 ×

/MATLAB Drive/tempcheck.m

```
time = linspace(0, sim_time, num_steps); % Time vector

% Simulation Loop
for i = 1:num_steps
    % Heat generated by the battery
    heat_generated = battery_capacity * 1000 * (1 - battery_efficiency) * heat_gen_rate;

    % Heat removed by the cooling system
    heat_removed = cooling_power * heat_transfer_coeff * (battery_temp - ambient_temp);

    % Net heat in the system
    net_heat = heat_generated - heat_removed;

    % Update battery temperature
    battery_temp = battery_temp + (net_heat / (battery_capacity * 3600)) * dt;

    % Store the temperature
    temperature(i) = battery_temp;
end

% Plot the results
figure;
plot(time, temperature, 'LineWidth', 1.5);
title('EV Battery Temperature Over Time');
xlabel('Time (s)');
ylabel('Battery Temperature (°C)');
grid on;

% Display final battery temperature
disp(['Final Battery Temperature: ', num2str(battery_temp), ' °C']);
```

battery_capacity : is the capacity of the battery in kWh.

initial_temperature : is the starting temperature of the battery.

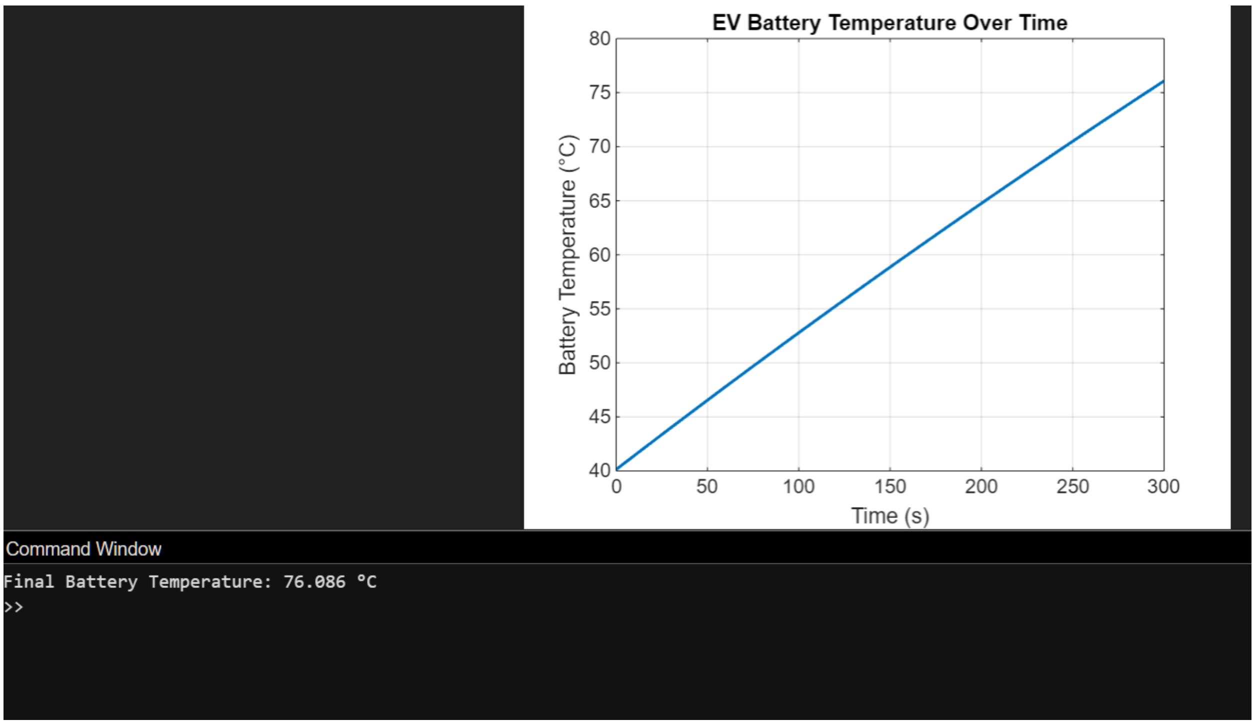
ambient_temperature : is the temperature of the surrounding environment.

heat_generation_rate :is the rate at which heat is generated by the battery.

cooling_rate : is the rate at which heat is dissipated by the cooling system.

time_step : is the time increment for the simulation.

simulation_time : is the total duration of the simulation.



Embedded Systems:

Part 1

Task: Design an embedded system for an automotive application that monitors and controls the temperature of an engine. The system should have the following specifications:

Embedded System Design for Monitoring and Controlling Engine Temperature

Key Components

Micro-controller/Processor

- Arduino

Temperature Sensors

- NTC thermistor
- DS18B20 (digital temp sensors)

Actuators

- Cooling Fans
- Electric Coolant Pump

Display & Indicators

- LCD Display
- Buzzrer or Led

Power Supply

- 12v dc supply

Vehicle Communication

- UART/12C/SPI
- CAN BUS

Programming Language and Platform

- C/C++
- Python

Development Platform

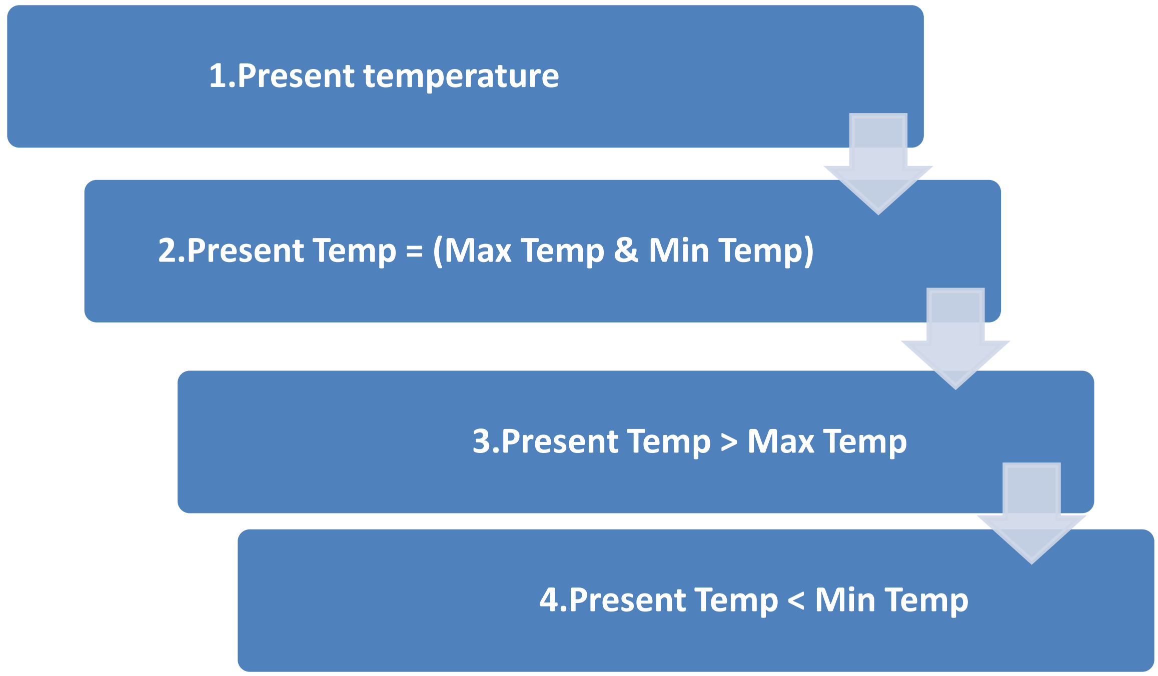
- STM32CubeIDE for STM32 microcontrollers.

Control Algorithm

Use a **PID (Proportional-Integral-Derivative**) controller to regulate engine temperature.

Logic:

- Compare current temperature with the setpoint.
- Adjust the fan speed or open/close the coolant valve proportionally.
- Beep the Buzzer or Blink the Warning LED if temperature exceeds critical limits.



Working   Embedded System

1. Sensor reads engine temperature **(Present Temperature).**
2. Compare Present Temperature with the desired range **(Max Temp & Min Temp)**
3. If (**Present Temp > Max Temp**), increase cooling fan speed or open coolant valve.
4. If (**Present Temp < Min Temp**), reduce cooling intensity or close the valve.
5. Log all events and provide visual/audible alerts for anomalie