

Experiment M6

Robot Leg Part I (a.k.a., DC Motor Control)

Procedure

Lab objectives: After completing this lab, students should be able to:

1. Tune a PID controller based on experimental outcomes.
2. Identify the electrical parameters of a DC motor model from experimental data.
3. Implement a feedforward controller for current with a DC electric motor.
4. Employ multi-rate control systems to simplify the control of complex systems.
5. Identify the physical parameters of a DC motor model from experimental data.
6. Apply knowledge of second-order systems to analyze the behavior of dynamic systems.
7. Alter the physical characteristics of a mechanical system through feedback control.

Overview:

In this lab, we will learn to control the behavior (broadly defined) of a DC electric motor model. A diagram of our DC electric motor is given below. The motor is given a voltage input V . This voltage produces current I in the motor, which places torque τ on the motor shaft, and then causes a change in the output angle θ of the motor.

In Part 1, we will first look at how to control the output angle of the motor directly, by programmatically setting the voltage input based on a Proportional Integral Derivative (PID) controller. If we want to control the angle of the output, this control strategy is a great option.

However, if we want to control the physical feel of the output (e.g., how stiff it is to a user that may come in contact with it), then the above doesn't do the job. To change the physical characteristics at the output (stiffness and damping), we will proceed through three main steps (Parts 2-4). In Part 2, we will run experiments to identify an electrical model of the

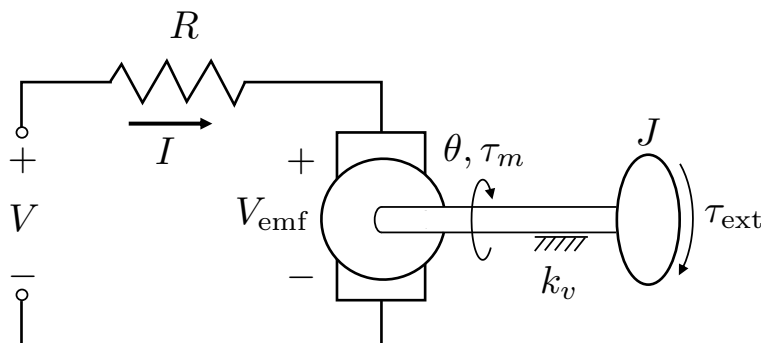


Figure 1: Model of DC motor with input voltage V , winding resistance R , current I , back-EMF voltage V_{emf} , output angle θ , motor torque τ_m , viscous friction coefficient k_v , rotational inertia J , and external torque τ_{ext} .

motor by looking at the relationship between input voltage and measured current. This model will then allow us to control the current flowing through the motor, which is proportionally related to the motor torque.

Once we can control the torque on the motor shaft, in Part 3, we will then run experiments to identify the physical model of the motor (i.e., to identify the rotational inertia of the motor and the damping on the motor shaft). Finally, in Part 4, we use this information to programmatically change how stiff or damped the motor shaft will feel to the user. That is, even without a spring attached to the motor shaft, we will be able to make a user feel like a spring is present, and we'll be able to adjust how stiff it feels.

Throughout, we will denote physical parameters (that we can't control) in purple (e.g., J, k_b), our estimates of them in blue with hats (e.g., \hat{J}, \hat{k}_b), and user-selected tuning parameters in green (e.g., K_p, K_i).

In the next lab, we'll build on this to adjust how stiff a multi-jointed leg feels at the its foot.

Deliverables: Checked score sheet. Other deliverables listed at the end of each part.

Part 0. Getting Started

The Arduino family that we have been using until now provided a wide variety of user friendly micro-controller platforms to easily prototype mechatronic systems with minimal required knowledge. However, to accomplish our tasks in the following experiments, we need a more powerful platform with higher CPU frequency for fast math calculations and advanced connectivity options to transfer more data.

In this experiment, we will use a far superior micro-controller platform: FRDM-K64F (pronounced freedom) which is equipped with a 120MHz ARM CPU (same architecture as your phone) and an Ethernet port. We will take a few steps to familiarize ourselves with the new platform.

1. Locate your code library that you created in the first experiment on the desktop: *AME30358_YourNames*. Create a sub-directory and name it as *M6-M7*. We are going to put **all local files** from M6 and M7 in this folder.
2. If you are unfamiliar with C++, take a few moments to familiarize yourself with some of the available documentation.
 - Tutorial: <http://www.tutorialspoint.com/cplusplus/index.htm>
 - Reference: <http://www.cplusplus.com/reference>
3. You should have your **Keil Studio Cloud** (hereinafter 'Keil') account created during the pre-lab quiz. Open **Keil** and log in with your account at <https://studio.keil.arm.com/auth/login/>.

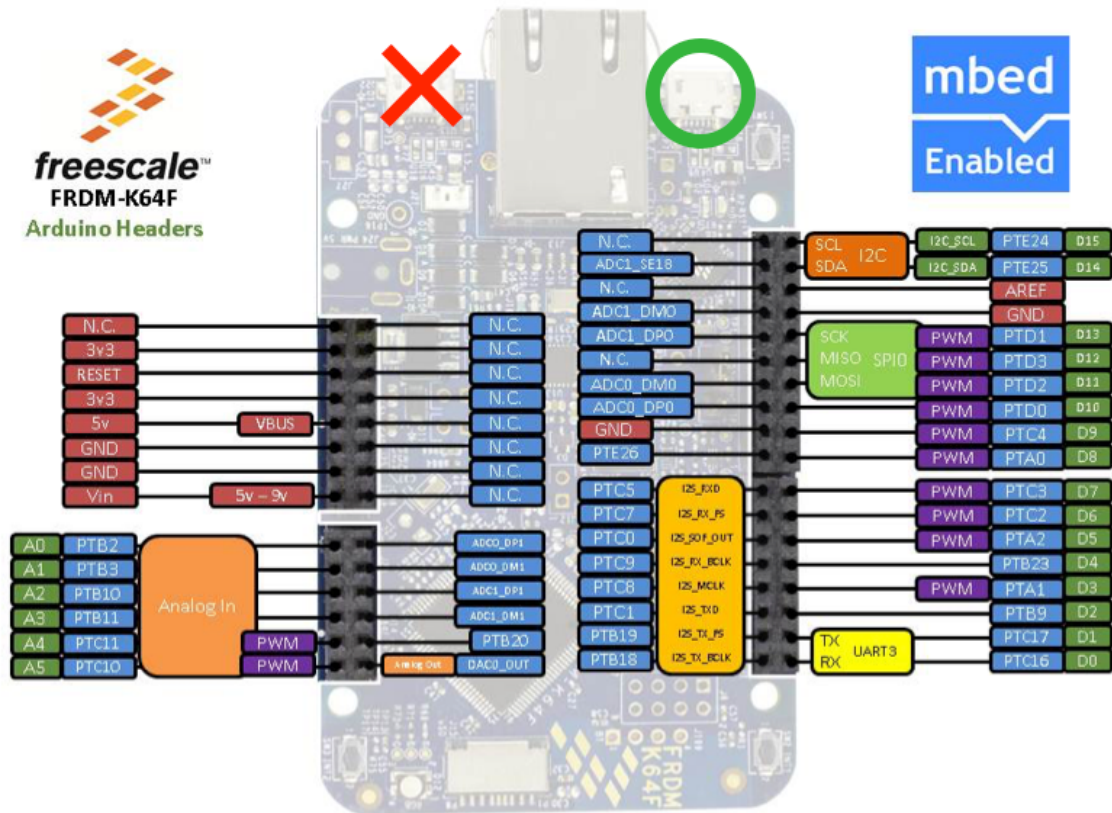



Figure 2: FRDM-K64F micro-controller (the outer rows are compatible with Arduino UNO).

4. Create new project with template
 - (a) Create a new project through [File → New → Mbed Project](#)
 - (b) Choose the [mbed-os-example-blinky](#) template. (Change the project name with your preference is welcomed but not required.)
 - (c) Keep the [Make this the active project](#) selected
 - (d) For this test, we don't need the Git repository. Please uncheck the item [Initialize this project as a Git repository](#).
 - (e) Click the [Add project](#) button to create. Review the example code.
5. Add FRDM-K64F to the build target (Click [build target](#) on the left, then search).
6. Connect the FRDM-K64F controller board to your computer through the right micro-USB port on board as shown in Fig. 2.
7. Verify that a flash drive named FRDM-K64F shows up within your file explorer.

8. Navigate back to [Keil](#), and flash program to controller via download
 - (a) Compile code using the [Build](#) button (hammer ) – this will download a compiled binary file to your computer.
 - (b) Download the generated binary file by dragging it from download folder to FRDM-K64F flash drive. If a new “AutoPlay” window shows up, it means the file was successfully flashed.
 - (c) In order to make sure the new code is running, press reset on FRDM board (the button next to the USB port used to connect the device).
9. Verify that the red LED is blinking. If so, the flashing process worked successfully.
10. Change the `BLINKING_RATE_MS` in `main.cpp` file so that the LED is blinking in 0.5Hz. Flash the program to the controller following the instruction above. **Demonstrate the working result to the lab instructor to obtain points on the score sheet.**

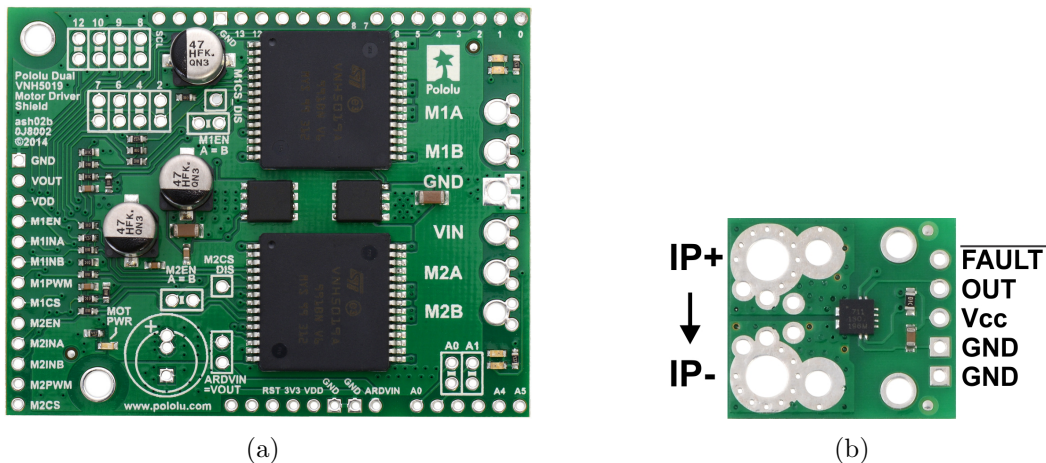


Figure 3: (a) Pololu dual VNH5019 motor driver shield for Arduino, (b) ACS711EX Current Sensor.

Color	Function
Red	Motor power
Black	Motor power
Green	Encoder GND
Blue	Encoder Vcc (+5V)
Yellow	Encoder A Out
White	Encoder B Out

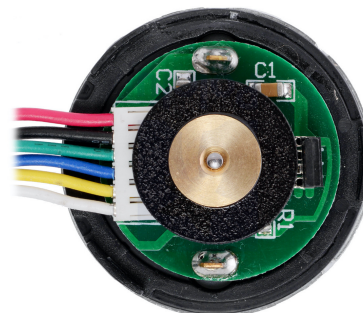


Figure 4: 12V DC Motor with 18.75:1 gearbox and encoder pinout.

Part 1. DC Motor Setup & Position Control

1. Measure the winding resistance of the motor using DMM (measure across red and black wires on the motor). It should be a few Ohms. Record the value in your lab notebook.
2. Wiring with Dupont pin wires.
 - (a) Put the motor drive shield (Fig. 3(a)) on the FRDM board. Note that the shield is designed for Arduino UNO and FRDM board has compatible pinouts.
 - Further documentations can be found in appendix
 - (b) Connect the analog current sensor (Fig. 3(b)) and the motor to the FRDM/shield assembly following the instruction of Fig.5. The sensor output connects to pin “A2” on the FRDM board.
 - (c) Connect the single motor on your test bench and its encoder (Fig. 4) to the FRDM/shield assembly as well.
 - (d) Finally connect the power input port on motor shield to the power supply (the charging brick) through the inline kill switch. **Make sure the power is turned off while you connect it!**

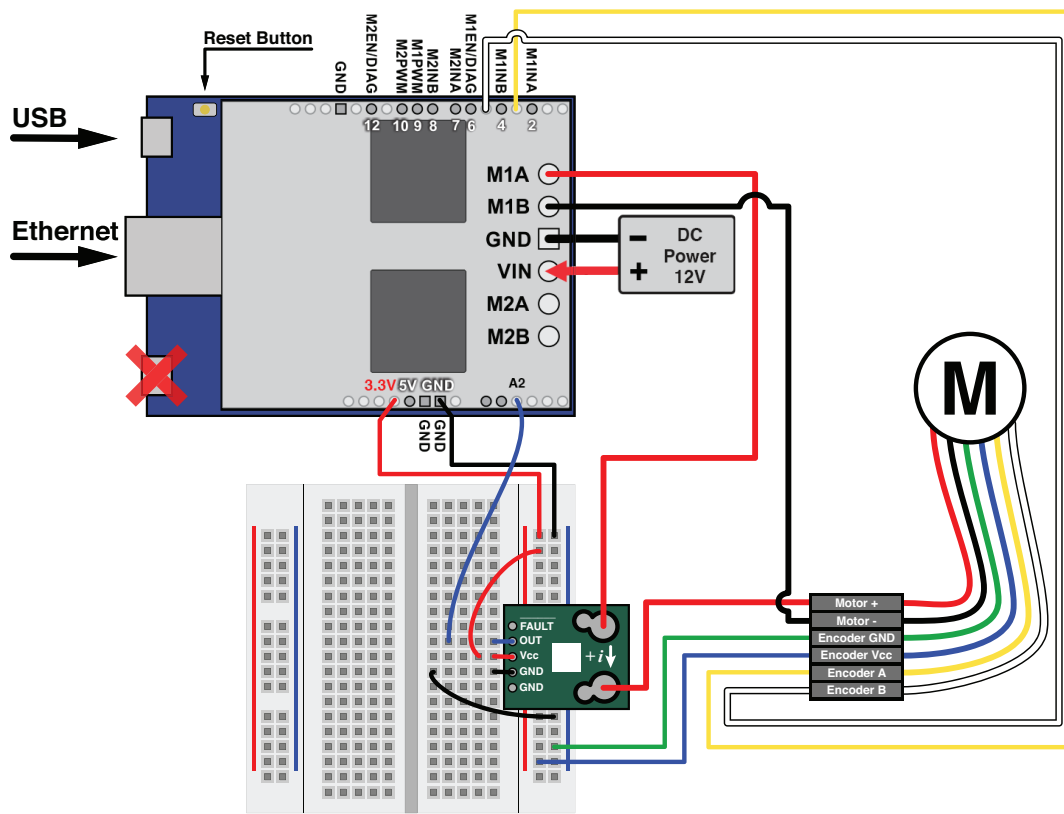



Figure 5: Wiring diagram.

3. Import Base Code for Motor Interface

- (a) Clone the motor interface base code from github through [File → Clone...](#)
 - i. Copy and paste the url <https://github.com/ND-AME-31358/MotorInterfaceBase> into the [URL](#) box in the popup window. Keep the [Project name](#) or choose a reasonable one.
 - ii. Make sure to check [Make this the active project](#) or manually set it active after import. Click [Add project](#) button to clone.
- (b) Select the [FRDM-K64F](#) as the [build target](#) again as in [Part 0. Step \(5\)](#). Keil would frequently demand manual adjustment of the target board settings, often following a project import and occasionally after altering the active project.
- (c) Take a few moments to familiarize the [main.cpp](#) example code, enter the corresponding pin name for motor control and current sensor to the code block [line 34-43](#).
- (d) Flash the code to the FRDM board. Sometimes red LED on board would blink to indicate the flashing is done and demand reset.
- (e) Next we will connect a debugging interface via serial.
 - i. Open the device manager by searching in the start menu. Check the enumeration names under the  [Ports \(COM & LPT\)](#) category. Write down the COM number (e.g. [COM16](#)) with the name `mbed serial` in your lab notebook.
 - ii. Open the serial monitor application PuTTY, select the [Serial](#) checkbox. Enter the COM number you got and enter 115200 as the baud rate. Click open.
 - iii. By pressing the reset button on the FRDM board, you should be able to see the messages from FRDM board. Confirm the message as

```
=====
Starting Server
...Intializing Ethernet
...Connecting
```

4. Interface with MATLAB

- (a) Data from the FRDM board will be sent to the computer via an Ethernet connection, so it can be viewed in real time via MATLAB.
- (b) Connect the FRDM board and your computer through the Ethernet cable and the USB-Ethernet adapter. Ask the lab instructor to confirm the Ethernet connection is configured as follows:
IP Address: 192.168.1.200

Subnet mask: 255.255.255.0
Router/Gateway: 192.168.1.1

- (c) Now navigate back to serial monitor in PuTTY, press the reset button on the FRDM board and wait for a moment. **New Experiment** message as detailed below should appear. Once you confirm the message **Waiting for parameters...**, the board would be ready to communicate with MATLAB. A green LED on board would light up at the same time to indicate it is ready.

```
=====  
Starting Server  
...Intializing Ethernet  
...Connecting  
...Ethernet IP Address is 192.168.1.100  
...Opened  
...Listening on Port 11223  
  
=====  
New Experiment  
...Waiting for parameters...
```

Note: this message and the green LED only indicate the UDP service we used to transport data is ready. If the IP address in last step was not properly configured, no data would be successfully transported in the following steps.

- (d) Download or copy the [RunExperiment.m](#) file from the [RunExperimentMatlab](#) folder within your [Keil project](#) onto your computer. Ideally, you should copy this file to your experiment folder from [Part 0. Step \(1\)](#). This file contains a base function shared by all MATLAB scripts in this experiment.
- (e) Download the [ApplyVoltage.m](#) code from the [Keil project](#) to the same folder, and open it in MATLAB.
- (f) **DO NOT turn on the power supply.** Run [ApplyVoltage.m](#) on your machine. A plot includes motor current (should be 0 amps) and encoder readings should pop up in MATLAB. Please turn the motor roughly 180 degrees and verify it on the plot. **Demonstrate the working result to the lab instructor for credit.**

5. Current sensing

- (a) Navigate back to Keil, fix the current measurement code at [line 99](#) of [main.cpp](#) by typing in the calibration equation from pre-lab quiz so that the variable `current` gives the sensed current, measured in amps. Note the following:
- The current sensor output sensitivity is 90 mV/A for our setup where the current sensor is connected to a 3.3V pin of the microcontroller.

- Also note that since the current sensor is bidirectional, a 1.65V output (half of 3.3V) represents 0 current. Higher than 1.65V means positive current and lower means negative current.
 - The sensor voltage [0.0V, 3.3V] is mapped to [0.0, 1.0] as the CS variable in the code.
 - Tip: add a 'f' after each number or a decimal point to ensure C++ consider it as a float number (e.g. 3f, 3.0, or both 3.14f).
- (b) Now connect and turn on the power supply to 12V. Run `ApplyVoltage.m` again, the motor should now rotate under default voltage: 3V. **Demonstrate the working result to the lab instructor for credit.**
6. Closed-loop motor control (PID)
- (a) Import the template PID control code by cloning (“Ctrl+Alt+M”) from https://github.com/ND-AME-31358/K64F_PID_Control as above.
 - (b) Set the **build target** to the FRDM-K64F board as [Part 0. Step \(5\)](#).
 - (c) Take a few moments to familiarize yourself with the `main.cpp` file (particularly the [line 79 to 115](#)).
 - (d) In `main.cpp`, complete the code on [lines 40-43](#) and [line 101](#) for the pin assignment and calibration formula using your code from the previous parts.
 - (e) Using this interface, modify the [line 114](#) of `main.cpp` in the Keil code for the FRDM board to implement a proportional-integral-derivative (PID) controller for the motor shaft angle.
 - As controlled by [line 128](#), we are running under control loop rate of 1 KHz.
 - Given a shaft angle $\theta[k]$ at time step k , a desired angle θ_d , and angle velocity $\dot{\theta}_d$, the error $e[k] = \theta_d - \theta[k]$, $\dot{e}[k] = \dot{\theta}_d - \dot{\theta}[k]$ should be related to the commanded voltage through:
$$V[k] = K_p e[k] + K_d \dot{e}[k] + K_i \sum_{j=0}^k e[j]$$
- For your information on later steps: In your program, it uses:
Inputs to FRDM board: θ_d , $\dot{\theta}_d$, K_p , K_i , K_d , Experiment Time
Outputs from FRDM board: $t[k]$, $\theta[k]$, $\dot{\theta}[k]$, $V[k]$, $I[k]$
- (f) To run an experiment, you should first flash the FRDM board with your code.

- (g) Similarly, download the `K64_PID_matlab.m` file included in the project to your computer, open it, take a few moments to familiarize yourself with it. Then run `K64_PID_matlab.m` on your computer. The same plots as in the previous step will appear, with the addition of the desired position being displayed as a red line.
- (h) Once working, and without tuning the gains in the MATLAB script, provide a graph of your closed-loop step response, showing position, velocity, voltage input, and current of motor over time.
- (i) Now, tune the gains in the MATLAB script so that the position of the motor reaches a desired position of 1 radian in under **150 milliseconds**, with overshoot less than **0.05 rad**. Guidelines are given below. **Demonstrate the working result to the lab instructor for credit.**
- You should start by tuning K_p , starting from a small value and increasing as you make better guesses. Note that K_p has units of volts per radian, so you can come up with a reasonable initial guess by considering how many volts of input you would want to apply for 1 radian of angle error. You should increase K_p until the response starts to display significant oscillations. (For this system, increasing K_p directly increases the natural frequency of the response.)
 - After tuning K_p , you can increase K_d to remove oscillations. The constant K_d has units of volts per rad/s. Based on the magnitude of velocities you are observing, and the fact that the voltage is limited to 12V, you should be able to make a reasonable guess to begin. If the motor moves too slow, you can decrease K_d . If it still displays oscillations, you should increase K_d . (For this system, increasing K_d directly increases the damping ratio of the response.)
 - After tuning, you can continue to iterate between tuning K_p and K_d to further improve response. If you observe steady-state error once the motor settles, then you can increase K_i to remove it.

Part 2. Electrical Model and Current Control

Consider the DC motor electrical dynamics equation

$$k_b \dot{\theta} + RI + LI = V \quad (1)$$

where $k_b, \dot{\theta}, R, I, L, V$ are back-EMF constant, angular velocity, winding resistance, winding inductance, and armature voltage (applied motor voltage). Assume the winding inductance is negligible ($L \approx 0$), we are going to estimate other constant parameters of the DC motor.

1. Resistance identification

- (a) In Keil, set the `MotorInterfaceBase` project to active, make sure the FRDM-K64F board is the `build target`, and flash the program to the FRDM board again.
- (b) Now, you can run the command `ApplyVoltage(volt)` to apply `volt` to the motor.
- (c) Run experiments with a number of voltage values from 3 to 12 V, while holding the motor in different stall positions such that $\dot{\theta} = 0$. (Hold tight.) Record the data of each time.
 - Note that the current sensor is not accurate under 1A so we start from 3V.
- (d) Based on Eq. (1), with $\dot{\theta} = 0$ and $L = 0$ estimate the winding resistance \hat{R} from your experimental values. Record the value in your lab notebook. Does the estimated resistance match the value you measured via a DMM in [Part 1. Step \(1\)](#)? **Show your estimated resistance to the lab instructor for credit.**

2. Back-EMF identification

- (a) Run experiments with a number of voltage values from 3 to 12 V again, but let the motor be free spinning (not stall) this time. Record a table in your lab notebook with a column for applied voltage, a column measured speed, and a column for measured current.
- (b) Similarly, based on Eq. (1), estimate the back-EMF \hat{k}_b from your experiments using measured values $I, \dot{\theta}$ and estimated \hat{R} .
- (c) **Show your estimated \hat{k}_b to the lab instructor for credit. Make sure the units are correct.**

3. Current Control: we will now use the parameters \hat{R} and \hat{k}_b to create a feedforward/feed-back controller.

- (a) Import the template current control code by cloning from https://github.com/ND-AME-31358/K64F_Current_Control as cloning procedure above.
- (b) In the `main.cpp` file, the code shares similarities with previous sections, but introduces a new function named `currentLoopFunc`, defined in [line 138-164](#). This

function implements the current controller and is invoked by a ticker, which is configured in [line 110](#).

- (c) Again, in `main.cpp` of your new project, complete the code on [lines 40-43](#) and [line 148](#) for the pin assignment and current sensor calibration formula using your code from the previous parts.
- (d) Consider a desired current I_d . Complete the code at [line 159](#) of the proportional current controller in the `main.cpp` file to match the formula below:

$$V = \hat{R} I_d + \hat{k}_b \dot{\theta} + K_p (I_d - I)$$

The constant K_p here has units of volts per amp, and should be a different value from what you used in the previous PID control of the motor angle. Note that this current controller is set up to run at a rate of 5kHz.

- (e) Flash the code to the FRDM board. Rest the board if necessary.
- (f) Download the `K64_Current_Control_matlab.m` in the project and open with MATLAB. Set the variables `R_motor` equals to the parameter you estimated (leave `kb` equals to zero). Note that this script plots the desired current as a red line in the figure.
- (g) Conduct experiments to test the current control while again holding the motor such that $\dot{\theta} = 0$. Tune K_p . Once satisfied, provide a step response for two different currents (one positive, one negative). Record the error on your lab notebook.
- (h) Investigate how your controller performs when you do not hold the output (i.e. when $\dot{\theta} \neq 0$). **Sketch the current v.s. time in your lab notebook. Write down the steady state value of the current.**
- (i) Set the variable `kb` equals to the parameter you estimated so that we have the back-EMF compensation. **Sketch the current v.s. time in your lab notebook. Write down the steady state value of the current. Demonstrate the results to the instructor.**

Part 3. Mechanical Model and Friction compensation

Consider the mechanical model of a DC motor:

$$k_\tau I + \tau_{\text{ext}} = J\ddot{\theta} + k_v\dot{\theta}, \quad (2)$$

where torque constant k_τ has the same numeric value as the back-EMF k_b when adopting standard metric units, k_v and J are the viscous friction coefficient ($\text{Nm}/\text{rad}\cdot\text{s}^{-1}$) and rotational inertia ($\text{Nm}/\text{rad}\cdot\text{s}^{-2}$), respectively. We consider the output torque $\tau_{\text{ext}} = 0$ when the motor is free spinning.

We will build from the results of the previous section, where the current controller was running at 5kHz (electrical dynamics are fast!) and now will include an outer controller for the physical dynamics (the mechanical dynamics here are slower) with the block diagram given below.

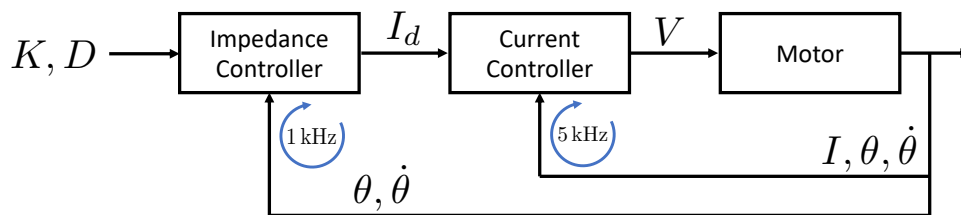


Figure 6: Multi-rate control diagram.

1. Setup

- Clone the template code from https://github.com/ND-AME-31358/Impedance_Control_1Dof.git. You will use this code for both this section and next section "Impedance Control".
- Again, in `main.cpp` of your new project, complete the code on [lines 40-43](#), [line 165](#), and [line 181](#) for the pin assignment, calibration formula, and current control using your code from the previous parts. Note that we have introduced an integral control term in the current controller, please leave it untouched.
- Download the file `K64_Impedance_Control_matlab.m` to your MATLAB folder.
- Take a moment to familiarize yourself with both pieces of code.
- With this example, MATLAB will send over a set of parameters to the FRDM board as follows:
 - \hat{R} : Estimated motor resistance (ohms)
 - \hat{k}_b : Estimated back-EMF constant ($\text{V}/\text{rad}\cdot\text{s}^{-1}$)
 - K_p : Proportional gain for current control (V/A)

- K_i : Integral gain for current control (V/A·s)
- \hat{k}_v : Estimated viscous friction coefficient (Nm/rad·s⁻¹)
- K : A virtual stiffness parameter (Nm/rad)
- D : A virtual damping parameter (Nm/rad·s⁻¹)
- t_{exp} : Time to run the experiment (s)

Note: Please fill in your estimated results of the blue variables above to the MATLAB script. The gains for the current controller are already tuned properly, so, it is not necessary to modify them. You will need to tune the stiffness K and damping D parameters later in Part 4.

- (f) In response, the FRDM board will send the following pieces of information back to MATLAB with an addition of desired current for plotting.
- t : Time (s)
 - θ : Angle (rad)
 - $\dot{\theta}$: Angular velocity (rad·s⁻¹)
 - V : Voltage (V)
 - I : Current (A)
 - I^d : Desired current (A)
2. Viscous friction identification: we will use feedback on the measured angle data to create an artificial spring torque. Thus the system will behave as a damped harmonic oscillator, and the viscous friction coefficient k_v can be determined from the damping ratio.
- (a) Assuming an ideal current controller, we will first attempt to identify the viscous friction coefficient k_v .
- (b) Note that using standard units, $k_b = k_\tau$. We can achieve a desired output shaft torque τ_d by setting $i_d = \tau_d / \hat{k}_b$ for the current controller.
- (c) If we assume exact tracking for a desired output shaft torque τ_d , and we set $\tau_d = -K\theta + \hat{k}_v \dot{\theta}$, then the physical angle will follow the differential equation:

$$J \ddot{\theta} + (k_v - \hat{k}_v) \dot{\theta} + K \theta = 0$$

- (d) Recall that any differential equation of the form:

$$A\ddot{y} + B\dot{y} + Cy = 0$$

will have a solution of the form:

$$y(t) = C_0 e^{-\zeta\omega_n t} \cos(\omega_d t + \phi)$$

for some constants C_1 and ϕ , where $\omega_n = \sqrt{C/A}$ is the natural frequency, $\zeta = \frac{B}{2A\omega_n}$ is the damping ratio, and $\omega_d = \omega_n \sqrt{1 - \zeta^2}$ is the damped natural frequency.

Note that when we set $\hat{k}_v = k_v$, the damping ratio will be zero, and the natural frequency will be $\omega_n = \sqrt{K/J}$.

With this in mind, modify your C++ code and conduct an experiment as follows:

- Check the [line 125-130](#) and verify that it will set the desired current for your current controller to achieve the desired torque $\tau_d = -K\theta + \hat{k}_v \dot{\theta}$.
- Flash the program to the FRDM board.
- In MATLAB, set the `R_motor` and `kb` equal to parameters you estimated. Keep the current controller gains K_p and K_i . Then firstly tune K so that you observe an oscillation of a frequency of roughly 2Hz while $\hat{k}_v = 0$. *Tip: You would need to physically interact the system by pushing the link about 1-2 rad to trigger the oscillation.*
- Then, increase \hat{k}_v until you observe a pure undamped sinusoid for $\theta(t)$. Once you have that, the value of \hat{k}_v gives an estimate for k_v . Use the oscillation period T of the response to estimate the motor inertia J via:

$$\frac{2\pi}{T} = \omega_n.$$

Write down the estimated values \hat{k}_v and \hat{J} in your lab notebook. Show them to a lab instructor for credit.

Part 4. Design Challenge: Impedance Control

We will now tune the full system to have a fast mechanical response without overshoot and oscillation. Note that the current control loop will be nested in the mechanical control loop, running at a much faster rate.

1. Let us consider the case when $\tau_d = -K\theta - D\dot{\theta} + \hat{k}_v \dot{\theta}$, then, if all parameter estimates are ideal:

$$J\ddot{\theta} + D\dot{\theta} + K\theta = \tau_{\text{ext}}$$

2. That is, we can implement a virtual spring K (units Nm/rad) and damper D (units Nm/rad·s⁻¹) at the output.

3. Modify on your code at [line 126-127](#) of `main.cpp` within [Keil Studio](#) so that it produce the torque as $\tau_d = -K\theta - D\dot{\theta} + \hat{k}_v\dot{\theta}$.
4. Make different selections of K and D and interact with your closed loop controller.
5. Fix a selection of K . Without experiments, make selections for D that passively would correspond to an underdamped, critically damped, and over damped system.
6. Test these selections and provide a graph of the position over time for each. Do they behave as expected? If not, offer suggestions about what may be the cause of the inconsistency. **Demonstrate the results of your experiment and discuss your findings with a lab instructor for final sign off.**

A. List of Equipment

1. 12V DC power supply with kill switch
2. FRDM-K64F micro-controller <https://os.mbed.com/platforms/FRDM-K64F/>

Note: All FRDM boards in this lab have been properly flashed with the latest firmware and bootloader. Please follow the instruction on the production page to update the firmware and bootloader if you are going to connecting a brand new board to a Windows machine. Otherwise, some storage service in Windows might 'brick' your board. See <https://os.mbed.com/blog/entry/DAPLink-bootloader-update/>

3. Pololu Dual VNH5019 Motor Driver Shield for Arduino
<https://www.pololu.com/product/2507>
4. 12V DC motor with 18.75:1 gearbox and 64 CPR encoder
<https://www.pololu.com/product/4751>
5. ACS711EX Current Sensor <https://www.pololu.com/product/2452>
6. Breadboard
7. USB-A to Micro-USB cable
8. USB-Ethernet adapter
9. Ethernet cable