Experiment M7
Robot Leg Part II

Procedure

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

1. Alter the physical characteristics of a mechanical system through feedback control.
2. Impedance control of an end-effector of a robot in Cartesian space
3. Trajectory following via impedance control

Overview:

In this lab, we will continue the impedance control of last experiment M6, now working with a 2 degree-of-freedom (DoF) robot leg. We will learn to control the end-effector (herein, the foot tip) of the robot moving in the Cartesian space, and also to control the stiffness and damping that is felt at the foot. For example, for a robot jumping onto a platform, you might not want to let the foot move laterally (to ensure it doesn't slip off the platform), but you'd likely want it to be soft in the vertical direction to ensure a soft landing. This (and other) use cases motivate us to consider how to programatically change the stiffness and the damping at the foot. The main challenge here is that the joints of the robot combine to together to determine these endpoint characteristics at the foot, so we'll have to learn how to make use of their coordinated effects.

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

Part 1. Impedance Control of 1 Dof

In the part 3 of the last experiment, we have controlled the motor so that its mechanical dynamics behave as a non-damped spring system (i.e., the friction is exactly compensated). We will now tune the full system to have a fast mechanical response without overshoot and oscillation via the impedance control strategy. Regardless of whether you successfully addressed the design challenge related to impedance control last week, it is essential to review it and confirm that the hardware is functioning correctly. To simplify the integration of the motor, encoder, current sensor, and motor drive, we will utilize a pre-soldered prototype PCB.

1. Wiring the experiment circuit as shown in Fig. 1:
   (a) Connect the motor drive shield with the FRDM micro controller board.
   (b) Make sure the power supply is off. Connect the 12V power supply (charging brick) through the inline kill switch to the shield.

(c) Connect the green, yellow and black wires from the pre-soldered prototype PCB to the motor drive ports of the shield. See Fig. 2 for detail instruction.

(d) Connect the prototype PCB with the motor shield by slotting-in on top of the shield. The dashed wires in the diagram will be connected to the FRDM board through the prototype PCB.

(e) Connect the FRDM board to your PC via Micro-USB and Ethernet cable. Note: do not connect the Micro-USB to the port with red X in the diagram.

(f) Connect the **single motor (not part of the leg)** as shown in the diagram. **Demonstrate the wiring to an instructor before proceeding.**

2. Code Setup

   (a) Open the Keil Cloud Stuido `https://studio.keil.arm.com/auth/login/`, navigate back to the last project Impedance_Control_1Dof that we have been using from last experiment. Set the project as active by right clicking on the project and selecting Set active project.

   (b) Please remember to store all files you utilized in the local folder named *AME30358_YourNames/M6-M7*, which was created last week.

   (c) Open the file `K64_Impedance_Control_matlab.m` in MATLAB from your local folder or download it again to your computer if you do not have it.

   (d) Again, 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 (V/rad·s^{-1})
   - K_p: Proportional gain for current control (V/A)

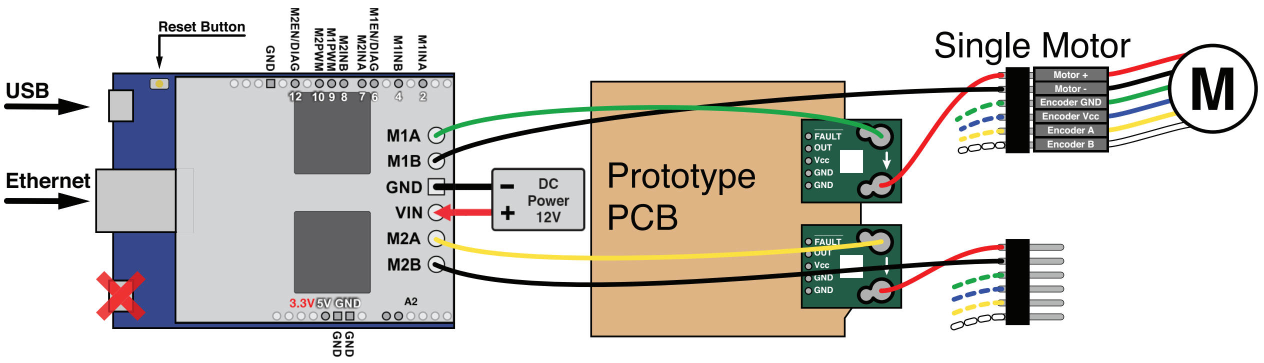
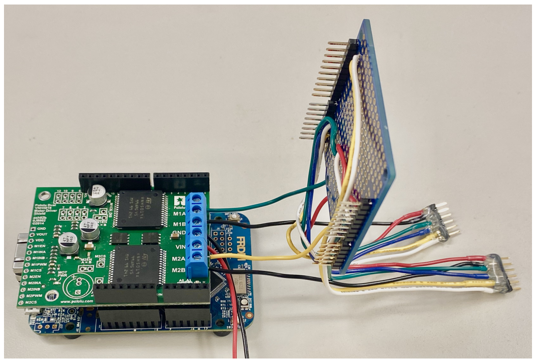


Figure 1: Wiring diagram of 1 Dof.


(a)

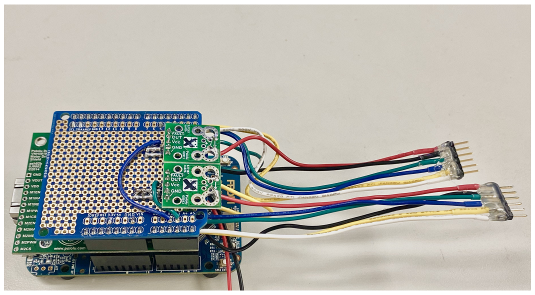

(b)

Figure 2: (a) Instruction of step 1(c): connect wires, (b) Instruction of step 1(d): slotting-in.

- 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 values from the previous lab for the blue variables above. The gains for the current controller are tuned properly, so it is not necessary to modify them. You will need to tune the stiffness K and damping D parameters in the following steps.

(e) In response, the FRDM board will send the following pieces of information back to Matlab

- t: Time (s)
- θ: Angle (rad)
- $\dot{\theta}$: Angular velocity (rad·s^{-1})
- V: Voltage (V)
- I: Current (A)
- I_d: Desired current (A)

3. As a reminder, the differential equation for the evolution of the joint angle is

$$k_\tau I + \tau_{\text{ext}} = J\ddot{\theta} + k_v\dot{\theta}, \tag{1}$$

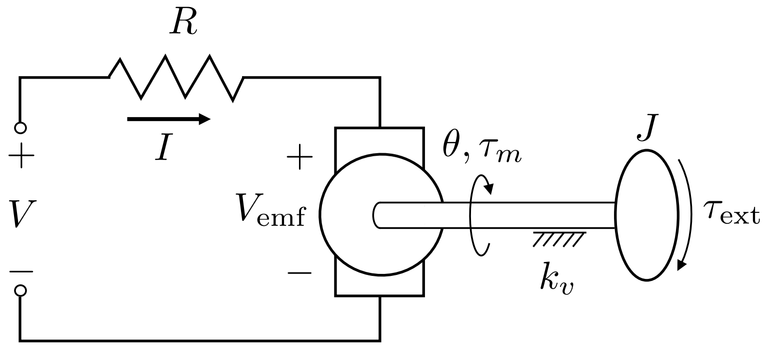


Figure 3: Model of DC motor with input voltage V, winding resistance R, current I, back-EMF voltage V_{emf}, output angle θ, motor torque τ_m, viscous friction coefficient k_v, rotational inertia J, and external torque τ_{ext}.

where k_τ is the torque constant (N·m/A) which equals to back-EMF constant k_b (V·s) under SI (*Système International*) units. J and k_v are joint inertia (Nm/rad·s^{-2}) and viscous friction coefficients (Nm/rad·s^{-1}) respectively.

4. Let us consider the case when the desired motor current is set as

$$i_d = \frac{\tau_d}{\hat{k}_b} = \frac{-K\,\theta - D\,\dot{\theta} + \hat{k}_v\,\dot{\theta}}{\hat{k}_b},$$

then, if all parameter estimates are ideal:

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

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

5. Modify on your code from M6 so that it commands torque on the motor as

$$i_d = \frac{\tau_d}{\hat{k}_b} = \frac{-K\,\theta - D\,\dot{\theta} + \hat{k}_v\,\dot{\theta}}{\hat{k}_b}$$

at line 126-127 of main.cpp within Keil Studio.

6. Flash the code onto FRDM board, and run the MATLAB script to run the experiment. *Recall:* You'll need to click the hammer icon to compile the code, and then you'll need to drag and drop the result onto the virtual flash drive for the FRDM board.

7. Turn on the power supply.

8. In MATLAB, run the program with different selections of K and D and physically interact with your closed-loop controller (e.g., by pushing on the link).

9. Fix a selection of $K = 0.1$ Nm/rad. Without experiments, make selections for D that passively would correspond to an underdamped, critically damped, and over damped system. Write these selections in your lab notebook.
10. Test these selections and sketch a plot of the position over time for each in your lab notebook. 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 credits.**

Part 2. Hardware verification of 2 Dof Leg

We will switch from the single motor to dual motor experiments. Before that, there are some logistic tasks to complete: verifying both the hardware and software are ready.

1. Setup hardware
   (a) Turn off the power supply. Then disconnect the single motor from the connector to the prototype PCB. Turn the test bench 180 degrees so that the tip is pointing out of table, mount the bench via two C-clamps. Ensure that the clamps do not obstruct the movement of the robot's leg.
   (b) Connect the hip and knee motors to the connectors of the prototype PCB correspondingly as shown in Fig. 4.
2. Clone the template code from `https://github.com/ND-AME-31358/Impedance_Control_2Dof` in the Keil via File → Clone.... Remember to set the FRDM-K64F as the target board.
3. Download the K64_Impedance_Control_2Dof_matlab.m and keypoints_leg.m MATLAB scripts from the Keil project to your local folder.

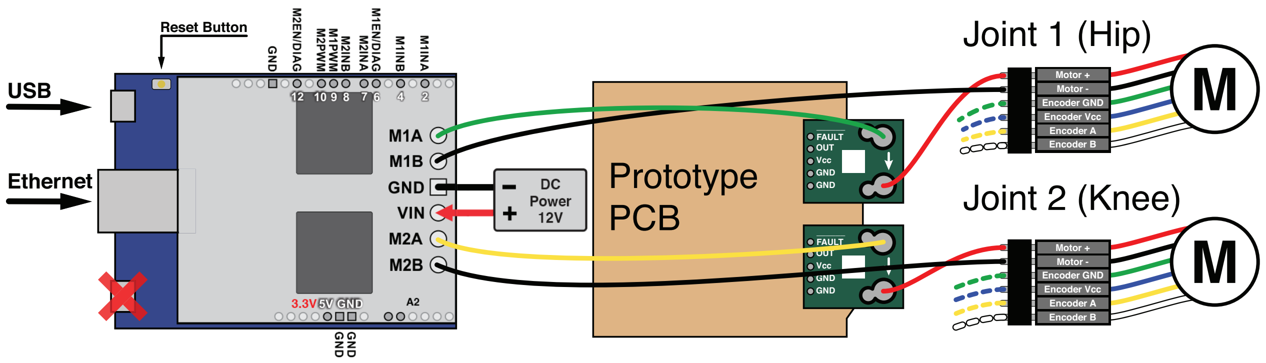


Figure 4: Wiring diagram of 2 Dof leg.

4. Take a few moments to familiarize yourself with both main.cpp and `K64_Impedance_Control_2Dof_matlab.m` code. Please note that while the code closely resembles that of the previous project, it has now been modified to accommodate dual motors, effectively doubling the components involved. The MATLAB script will send a set of experiment parameters to the FRDM board including:
   - Current control period in microseconds
   - Impedance control period in microseconds
   - Experiment time in seconds
   - $\hat{R}$: Motor winding resistance (Ohms)
   - $\hat{k}_b$: Back EMF Constant (V / (rad/s))
   - $\hat{k}_v$: Friction coefficient (Nm / (rad/s))
   - $\theta_{1,init}$, $\theta_{2,init}$: Initial angles for θ_1 and θ_2 (rad)
   - K_p: Proportional current gain (V/A)
   - K_i: Integration gain of current control (V/A·s)
   - K_{xx}, K_{yy}, K_{xy}: Foot stiffness (N/m)
   - D_{xx}, D_{yy}, D_{xy}: Foot damping (N/(m/s))
   - x_{set}, y_{set}: Foot position set points in x and y (m)
   - A: Amplitude of oscillation (m), used in Section Part 4
   - ω: Oscillation frequency (Hz), used in Section Part 4
   - Duty_max: Maximum PWM duty in range $[0.0, 1.0]$.
5. In response, the FRDM board will send the following pieces of information back to MATLAB : joint information (e.g., angle, current, etc.) of each joint and Cartesian position and velocity of foot tip. The MATLAB script takes the data and draw the robot leg on the screen.
6. Make sure the DC power is turned off. Flash the code to the FRDM board, and run the MATLAB script `K64_Impedance_Control_2Dof_matlab.m`. You should be able to see the digital twin of the cheetah leg on your screen. Move the leg by hand and notice the difference between the real robot and the visualization. Also, review the plots in Figures 1 through 3. **Demonstrate the result to a lab instructor for credit.**
7. Calibrate the encoders
   - The encoders that we use are called "incremental" encoders, in a sense that they only provide angles relative to their initial configuration. Thus, for our encoder

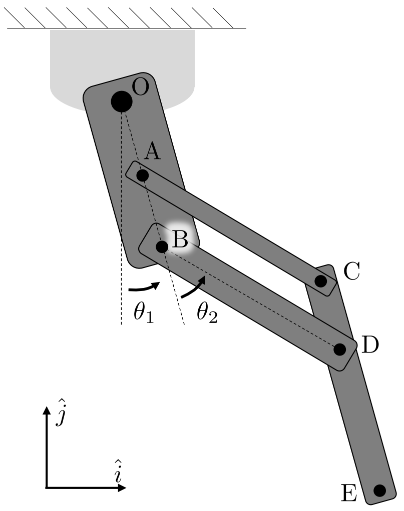


Figure 5: Illustration of the leg model.

readings to be meaningful, we need a well defined initial configuration for the leg. We will use the configuration with both joints rotated clockwise to their limits, as denoted on the diagram in the appendix.

- Determine values of the angles θ_1 and θ_2 (shown in Fig. 5) for this initial configuration. Record them on your lab notebook in both radian and degrees. Note that angles are measured as positive in the CCW direction. Once determined, you may pass values to the mbed using the input parameters `angle1_init` and `angle2_init` in MATLAB .

- Assuming your system always starts in this specified initial configuration, modify line 325-334 in main.cpp given angle1_init and angle2_init so that the variables angle1 and angle2 match the definitions of θ_1 and θ_2 in the figure.

- Verify that the line 180-185 in main.cpp reset the encoders and initialize the variables angle1_init and angle2_init upon receiving parameters from MATLAB upon the first run after the board is booted (or reset). For this code to function as intended, each time after you reset the FRDM board (by pressing the reset button) you will need to pull the leg to its initial configuration the first time you run the MATLAB script. This action will calibrate the incremental encoders based on the known initial configuration. On all subsequent runs (before another reboot), you can run the MATLAB script to start another experiment regardless the position of the foot since the encoders have been configured.

- Set the initial configuration properly, and move the system (by hand) to $\theta_1 = 0$ and $\theta_2 = \pi/2$ and verify your modifications using the visualization. **Demonstrate the result to a lab instructor for credit.**

Part 3. Impedance Control of 2 Dof Leg

Now we will expand the impedance control method to the robot leg. Instead of making the motor joint behave as a spring-damper, we will now make the foot tip behave like a virtual spring-damper system in 2D Cartesian space.

1. Using your result of your pre-lab exercise, complete the code at line 225-226 in main.cpp to compute the forward kinematics of the leg $\mathbf{r}(\boldsymbol{\theta}) = \begin{bmatrix} x \\ y \end{bmatrix}$, where $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$.
   - You can use variables `l_OA`, `l_OB`, `l_AC`, and `l_DE` to represent the length of OA, OB, AC, and DE respectively.
   - You should use `th1` and `th2` to represent the joint angle θ_1 and θ_2.
2. Using your result of your pre-lab exercise, complete code at line 231-234 in the main.cpp to compute the Jacobian of the leg:
$$\mathbf{J}(\boldsymbol{\theta}) = \frac{\partial \mathbf{r}}{\partial \boldsymbol{\theta}} = \begin{bmatrix} J_{x,\theta_1} & J_{x,\theta_2} \\ J_{y,\theta_1} & J_{y,\theta_2} \end{bmatrix}.$$
   - In the code, please use `Jx_th1` to represent J_{x,θ_1}, the same applies to the others.
3. Fix the line 238-239 to compute the foot tip velocity using Jacobian:
$$\dot{\mathbf{r}}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \mathbf{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} = \begin{bmatrix} J_{x,\theta_1} & J_{x,\theta_2} \\ J_{y,\theta_1} & J_{y,\theta_2} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$
   - In the code, `dth1` and `dth2` are used to represent $\dot{\theta}_1$ and $\dot{\theta}_2$.
   - Use `dx` and `dy` to represent $\dot{x}$ and $\dot{y}$.
4. Verify that the code at line 241-247 sets the desired foot position to the set point from MATLAB : $\mathbf{r}_d = [x_d, y_d]^\top = [x_{\text{set}}, y_{\text{set}}]^\top$, and the desired foot velocity to zero $\dot{\mathbf{r}}_d = [\dot{x}_d, \dot{y}_d]^\top = [0, 0]^\top$.
5. Given a desired foot position $\mathbf{r}_d$, and desired velocity $\dot{\mathbf{r}}_d$ complete the code at line 258-259 to compute the virtual force $[f_x, f_y]^\top$ based on a virtual spring damper at the foot.
$$\begin{bmatrix} f_x \\ f_y \end{bmatrix} = \begin{bmatrix} K_{xx}(x_d - x) + K_{xy}(y_d - y) + D_{xx}(\dot{x}_d - \dot{x}) + D_{xy}(\dot{y}_d - \dot{y}) \\ K_{xy}(x_d - x) + K_{yy}(y_d - y) + D_{xy}(\dot{x}_d - \dot{x}) + D_{yy}(\dot{y}_d - \dot{y}) \end{bmatrix}$$

6. Then, fix the code at line 264-265 to use the Jacobian to determine appropriate motor torques for your controller.

$$\tau = \mathbf{J}^T \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

7. We notice the robot leg has to start from an extreme configuration for encoder calibrations, and the set point in the later part would normally far away from the starting configuration. To avoid any extreme motion at the beginning of the experiment, a soft start parameter softStart is computed in line 205 which will increase from 0 to 1 in two seconds if it is the first run after reboot. Therefore, in line 268-269, the desired currents are given by $i_d = \beta(\tau_d/\hat{k}_b)$ where β is the softStart parameter.

   Show your completed code to a lab instructor.

8. Flash the code to your FRDM board, and turn on your power supply.

9. For the remainder of the tests, be sure to test your impedance controller first with `duty_max=0.2` as set in MATLAB . This will place a limit on the maximum duty factor used on the motor. After you have verified operation, you shall then increase this variable to 1 for the full performance.

10. Set `duty_max=0.2` in MATLAB . Test your Cartesian impedance from MATLAB with only horizontal stiffness $K_{xx} = 40$ N/m. The other Cartesian gains (K_{yy}, K_{xy}, D_{xx}, D_{yy}, and D_{xy}) should be zero. Be sure to put the leg in the correct initial configuration at the start of first experiment if you just flashed a new program, and hold onto the foot. Move the foot around and feel the stiffness in each direction.

    The foot should feel like there is a virtual spring preventing it from moving left and right, without any stiffness in the vertical direction. In MATLAB , the contours are showing the level sets for the energy of that virtual spring. (Denser contour lines indicate higher stiffness in that direction.)

    Note: besides the animation of the leg, the MATLAB script will also provide a set of plots with position and velocity (measured and desired), desired virtual force, and joint states (e.g., angle, current, etc.)

11. Test your Cartesian impedance with only horizontal stiffness $K_{xx} = 40$ N/m and vertical damping $D_{yy} = 10$ Ns/m. All other Cartesian gains should be zero. Be sure to put the leg in the correct initial configuration at the start of first experiment if you just flashed a new program.

12. Test your Cartesian impedance control with stiffness matrix

$$\begin{bmatrix} K_{xx} & K_{xy} \\ K_{xy} & K_{yy} \end{bmatrix} = \begin{bmatrix} 30 \text{ N/m} & 0 \text{ N/m} \\ 0 \text{ N/m} & 30 \text{ N/m} \end{bmatrix}$$

Damping gains should be zero. How do the stiffness characteristics at the end effector relate to the eigen vectors and corresponding eigen values for this matrix that you computed in your prelab.

13. Test your Cartesian impedance control with stiffness matrix

$$\begin{bmatrix} K_{xx} & K_{xy} \\ K_{xy} & K_{yy} \end{bmatrix} = \begin{bmatrix} 30 \text{ N/m} & -25 \text{ N/m} \\ -25 \text{ N/m} & 30 \text{ N/m} \end{bmatrix}$$

Damping gains should be zero. How do the stiffness characteristics at the end effector relate to the eigen vectors and corresponding eigen values for this matrix that you computed in your prelab. **Discuss and demonstrate the result to a lab instructor for credit.**

Part 4. Trajectory Tracking

Now, we will leverage the Cartesian space impedance control to accomplish a trajectory tracking task. The trajectory can be any arbitrary curve, we consider a straight line and a circle as two examples.

1. Straight Line Tracking
   (a) Modify the computation of desired x-axis foot position in line 243 so that it is oscillating around the foot position set point as

   $$x_d = A\sin(\omega t) + x_{\text{set}}$$

   where ω is the oscillation frequency (in rad/s) and A (in meters) is the amplitude of oscillation, both of them are sent from MATLAB to the FRDM board and stored in variables named `A` and `omega` in the C/C++ code.

   Tips: In the mbed code, you can use function `t.read()` *to get the time in seconds.*

   (b) Flash the code to FRDM and run the MATLAB script with $K_{xx} = K_{yy} = 20$N/m, all other Cartesian gains equal zero, and $A = 0.05$ m, $\omega = 3$ rad/s to verify your code. Check the tracking performance from the foot position plot in MATLAB .

   (c) Change the y-axis set point to -0.15 (m) for this part to have more interesting result.

   (d) Test with $A = 0.1$ m, $\omega = 3$ rad/s, observe the tracking performance.

   (e) Test with $A = 0.05$ m, $\omega = 12$ rad/s, observe the tracking performance by examining the plot of the x position vs. time in MATLAB . You can improve your tracking control by tuning the damping parameters D_{xx}, D_{yy}, and D_{xy}.

(f) What happens to the tracking performance as you increase the damping? Document the effect in your lab notebook.

(g) **Optimize the tuning of the damping parameters and demonstrate your results to a lab instructor for credit.**

2. Circle Tracking

(a) Similarly, modify your code of computation desired y-axis foot position so that it will also oscillate around the set point as

$$y_d = A\cos(\omega t) + y_{\text{set}}$$

(b) Flash the code and run the experiment with parameter: $K_{xx} = K_{yy} = 20\text{N/m}$ and your damping from the previous step. Use $A = 0.05$ m, and $\omega = 3$ rad/s. Observe the tracking performance both by looking at the leg, and by examining the plot of the y position in MATLAB . Make note the amplitude error and phase error in your lab notebook.

(c) Test with $\omega = 12$ rad/s, keep all gains unchanged. Observe the tracking error. Make note the amplitude error and phase error in your lab notebook.

(d) Now, increase the K_{xx} and K_{yy} to 30 N/m, test and observe the performance. Make note the amplitude error and phase error in your lab notebook.

(e) **Demonstrate your results to a lab instructor for credit.**

3. Tracking with velocity reference

(a) In line 246-247, you will see that the desired velocity of foot tip is always set to 0 which is inconsistent with our circle trajectory.

(b) Modify your code so that the desired velocity matches our desired trajectory. Then run the code again with $K_{xx} = K_{yy} = 20$ N/m, and using your optimized damping gains. Set $A = 0.05$ m, and $\omega = 12$ rad/s.

(c) How does the performance compare to the result in step 2c? What happens now when you increase the damping? Document the effect in your lab notebook.

(d) **Demonstrate your results to a lab instructor for credit.**

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. Single 12V DC motor with 18.75:1 gearbox and 64 CPR encoder
   `https://www.pololu.com/product/4751`
5. Assembly of the Junior Cheetah Leg with two above-mentioned motors
6. Two ACS711EX Current Sensor `https://www.pololu.com/product/2452`
7. Pre-soldered prototype PCB
8. USB-A to Micro-USB cable
9. USB-Ethernet adapter
10. Ethernet cable

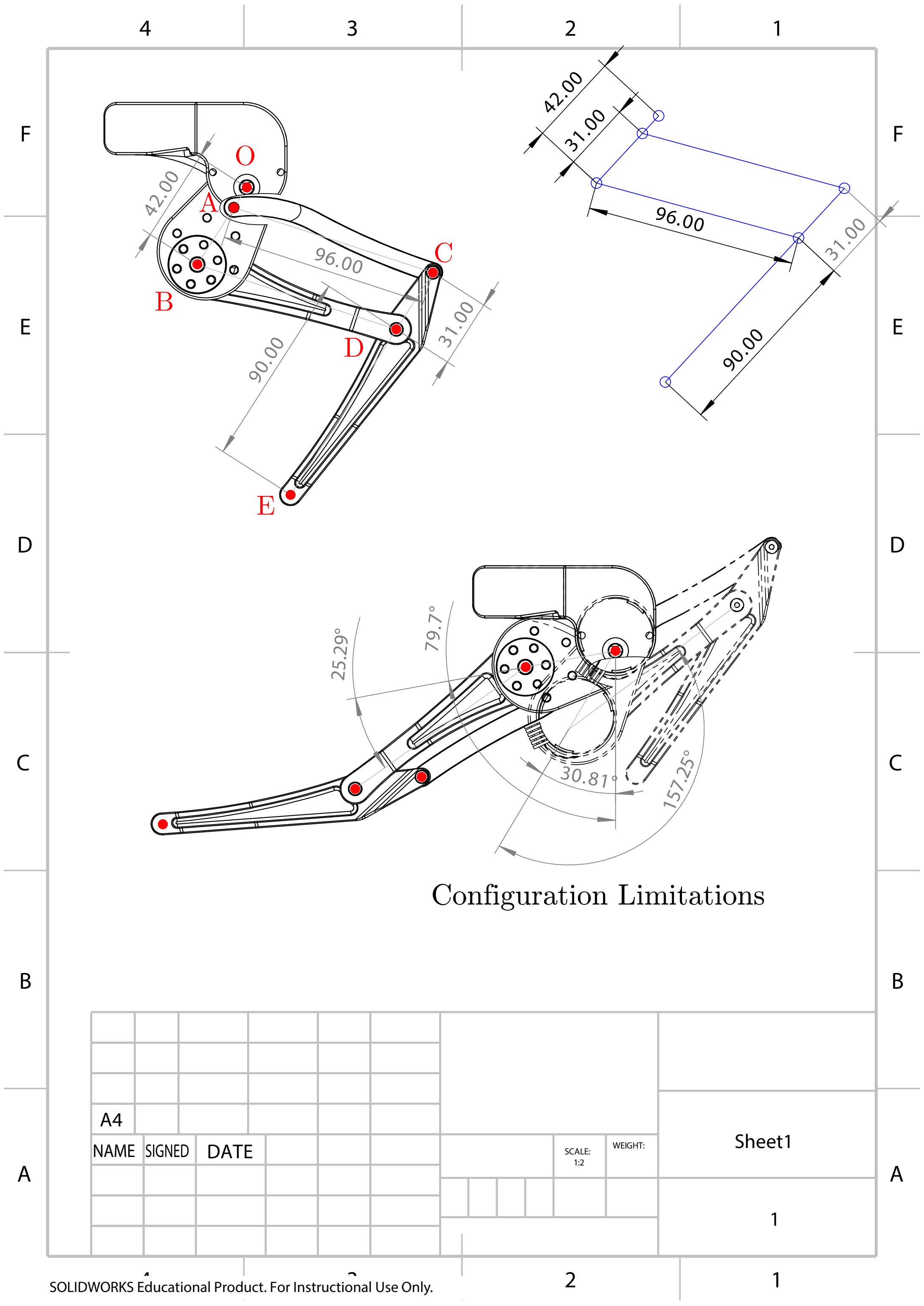


Figure 6: CAD drawing of the 2 Dof leg.