Amanda Ghassaei, Matt Carney, Eric VanWyk
Lab 3 Writeup

```
#include "mbed.h"
#include "rtos.h"
#include "EthernetInterface.h"
#include "ExperimentServer.h"
#include "QEI.h"

#define NUM_INPUTS 21
#define NUM_OUTPUTS 17
#define PULSE_TO_RAD (2.0f*3.14159f / 1200.0f)

Serial pc(USBTX, USBRX);    // USB Serial Terminal
ExperimentServer server;    // Object that lets us communicate with MATLAB
Timer t;                    // Timer to measure elapsed time of experiment

// Variables for q1
float current1;
float current_des1;
float angle1;
float angle_des1;
float velocity1;
float velocity_des1;
float duty_factor1;
float angle1_init;

// Variables for q2
float current2;
float current_des2;
float angle2;
float angle_des2;
float velocity2;
float velocity_des2;
float duty_factor2;
float angle2_init;

// Fixed kinematic parameters
const float l_OA=.010;
const float l_OB=.040;
const float l_AC=.095;
const float l_DE=.095;

// Timing parameters
float pwm_period_us;
float current_control_period_us;
float impedance_control_period_us;
float exp_period;

// Control parameters
float K_xx;
float K_yy;
float K_xy;

float D_xx;
float D_xy;
float D_yy;

float current_gain;

float xDesFoot;
float yDesFoot;


// Model parameters
float R;
float k_emf;
float nu1, nu2;
float supply_voltage;
float duty_max;


DigitalOut motorFwd1(D7);
DigitalOut motorRev1(D6);
AnalogIn currentSense1(A0);
PwmOut   pwmOut1(D5);

DigitalOut motorFwd2(D13);
DigitalOut motorRev2(D12);
AnalogIn currentSense2(A1);
PwmOut   pwmOut2(D11);

QEI encoder1(D3,D4 , NC, 1200 , QEI::X4_ENCODING); // Pins D3, D4, no index, 1200 counts/rev, Quadrature encoding
QEI encoder2(D9,D10, NC, 1200 , QEI::X4_ENCODING); // Pins D3, D4, no index, 1200 counts/rev, Quadrature encoding
Ticker currentLoop;
```

```
float prev_current_des1 = 0;
float prev_current_des2 = 0;
void CurrentLoop()
{
    motorFwd1 = current_des1 >= 0;
    motorRev1 = current_des1 <  0;

    current1  = currentSense1.read()*3.3f / 0.14f;   //measure current
    if (prev_current_des1 < 0)
        current1*=-1;

    duty_factor1=(current_des1*R + current_gain*(current_des1-current1) + k_emf*velocity1)/supply_voltage;
    motorRev1 = duty_factor1< 0;
    motorFwd1 = duty_factor1>=0;
    float absDuty1 = abs(duty_factor1);
    if (absDuty1 > duty_max) {
        duty_factor1 *= duty_max / absDuty1;
        absDuty1= duty_max;
    }
    pwmOut1.write(absDuty1);
    prev_current_des1 = current_des1;

    motorFwd2 = current_des2 >= 0;
    motorRev2 = current_des2 <  0;

    current2  = currentSense2.read()*3.3f / 0.14f;   //measure current
    if (prev_current_des2 < 0)
        current2*=-1;

    duty_factor2=(current_des2*R + current_gain*(current_des2-current2) + k_emf*velocity2)/supply_voltage;

    motorRev2 = duty_factor2< 0;
    motorFwd2 = duty_factor2>=0;
    float absDuty2 = abs(duty_factor2);
    if (absDuty2 > duty_max) {
        duty_factor2 *= duty_max / absDuty2;
        absDuty2= duty_max;
    }
    pwmOut2.write(absDuty2);
    prev_current_des2 = current_des2;
}

int main (void) {

  encoder1.reset();
  encoder2.reset();

  // Link the terminal with our server and start it up
  server.attachTerminal(pc);
  server.init();

  // Continually get input from MATLAB and run experiments
  float input_params[NUM_INPUTS];
  while(1) {
    if (server.getParams(input_params,NUM_INPUTS)) {
      pwm_period_us               = input_params[0]; // PWM_Period in mirco seconds
      current_control_period_us   = input_params[1]; // Current control period in micro seconds
      impedance_control_period_us = input_params[2]; // Impedance control period in microseconds seconds
      exp_period                  = input_params[3]; // Experiment time in seconds

      R                       = input_params[4]; // Terminal resistance (Ohms)
      k_emf                   = input_params[5]; // Back EMF Constant (V / (rad/s))
      nu1                     = input_params[6]; // Friction coefficienct 1 (Nm / (rad/s))
      nu2                     = input_params[7]; // Friction coefficienct 1 (Nm / (rad/s))
      supply_voltage          = input_params[8]; // Power Supply Voltage (V)

      angle1_init             = input_params[9]; // Initial angle for q1 (rad)
      angle2_init             = input_params[10];// Initial angle for q2 (rad)

      current_gain            = input_params[11]; // Proportional current gain (V/A)
      K_xx                    = input_params[12]; // Foot stiffness N/m
      K_yy                    = input_params[13]; // Foot stiffness N/m
      K_xy                    = input_params[14]; // Foot stiffness N/m

      D_xx                    = input_params[15]; // Foot damping N/(m/s)
      D_yy                    = input_params[16]; // Foot damping N/(m/s)
      D_xy                    = input_params[17]; // Foot damping N/(m/s)

      xDesFoot                = input_params[18]; // Desired foot position x (m)
      yDesFoot                = input_params[19]; // Desired foot position y (m)

      duty_max                = input_params[20]; // Maximum duty factor

      pwmOut1.period_us(pwm_period_us);
      pwmOut2.period_us(pwm_period_us);

      // Attach current loop!
```

```
    currentLoop.attach_us(CurrentLoop,current_control_period_us);

    // Setup experiment
    t.reset();
    t.start();



    motorFwd1 = 1;
    motorRev1 = 0;
    pwmOut1.write(0);

    motorFwd2 = 1;
    motorRev2 = 0;
    pwmOut2.write(0);

    // Run experiment
    while( t.read() < exp_period ) {
      // Perform control loop logic

      angle1 = encoder1.getPulses() *PULSE_TO_RAD + angle1_init;
      velocity1 =encoder1.getVelocity() * PULSE_TO_RAD;

      angle2 = encoder2.getPulses() * PULSE_TO_RAD + angle2_init;
      velocity2 = encoder2.getVelocity() * PULSE_TO_RAD;

      // Forward Kinematics
      float xLeg = l_AC*sin(angle1 + angle2) + l_DE*sin(angle1) + l_OB*sin(angle1);
      float yLeg = -l_AC*cos(angle1 + angle2) - l_DE*cos(angle1) - l_OB*cos(angle1);

      // Jacobian Computation
      float J_x_q1 = l_AC*cos(angle1 + angle2) + l_DE*cos(angle1) + l_OB*cos(angle1);
      float J_y_q1 = l_AC*sin(angle1 + angle2) + l_DE*sin(angle1) + l_OB*sin(angle1);
      float J_x_q2 = l_AC*cos(angle1 + angle2);
      float J_y_q2 = l_AC*sin(angle1 + angle2);

      // Compute foot velocities
      float dxLeg = velocity1*(l_AC*cos(angle1+angle2) + l_DE*cos(angle1) + l_OB*cos(angle1)) + velocity2*l_AC*cos(angle1 + angle2);
      float dyLeg = velocity1*(l_AC*sin(angle1+angle2) + l_DE*sin(angle1) + l_OB*sin(angle1)) + velocity2*l_AC*sin(angle1 + angle2);

      // Set desired virtual force
      float fx = K_xx*(xDesFoot - xLeg) - D_xx*dxLeg + K_xy*(yDesFoot - yLeg) - D_xy*dyLeg;
      float fy = K_yy*(yDesFoot - yLeg) - D_yy*dyLeg + K_xy*(xDesFoot - xLeg) - D_xy*dxLeg;

      // Use jacobian to transform virtual force to torques
      float tau_des1 = J_x_q1*fx + J_y_q1*fy;
      float tau_des2 = (J_x_q2*fx + J_y_q2*fy);

      // Set desired currents
      current_des1 = tau_des1/k_emf;//(-K_xx*angle1-D_xx*velocity1+nu1*velocity1)/k_emf;
      current_des2 = tau_des2/k_emf;;//(-K_yy*angle2-D_yy*velocity2+nu2*velocity2)/k_emf;

      // Form output to send to MATLAB
      float output_data[NUM_OUTPUTS];
      output_data[0] = t.read();
      output_data[1] = angle1;
      output_data[2] = velocity1;
      output_data[3] = current1;
      output_data[4] = current_des1;
      output_data[5] = duty_factor1;

      output_data[6] = angle2;
      output_data[7] = velocity2;
      output_data[8] = current2;
      output_data[9] = current_des2;
      output_data[10]= duty_factor2;

      output_data[11] = xLeg;
      output_data[12] = yLeg;
      output_data[13] = dxLeg;
      output_data[14] = dyLeg;
      output_data[15] = fx;
      output_data[16] = fy;

      // Send data to MATLAB
      server.sendData(output_data,NUM_OUTPUTS);
      wait_us(impedance_control_period_us);
    }
    // Cleanup after experiment
    server.setExperimentComplete();
    // control.detach();
    currentLoop.detach();
    pwmOut1.write(0);
    pwmOut2.write(0);

   } // end if
  } // end while
} // end main
```