Massachusetts Institute of Technology
Department of Mechanical Engineering

2.004 *Dynamics and Control II*

Laboratory Session 6:
Active Gimbal Control – Dual-Axis Gimbal

Laboratory Objective:

In this lab session you will program an Arduino UNO microcontroller board in order to control and stabilize a dual-axis gimbal by rejecting any unwanted disturbances. Specifically we wish to actively mitigate both pitch and roll motions of the 2.004 gimbal system (see Fig. 1) so the phone that is being held by the gimbal can stay stationary[1].

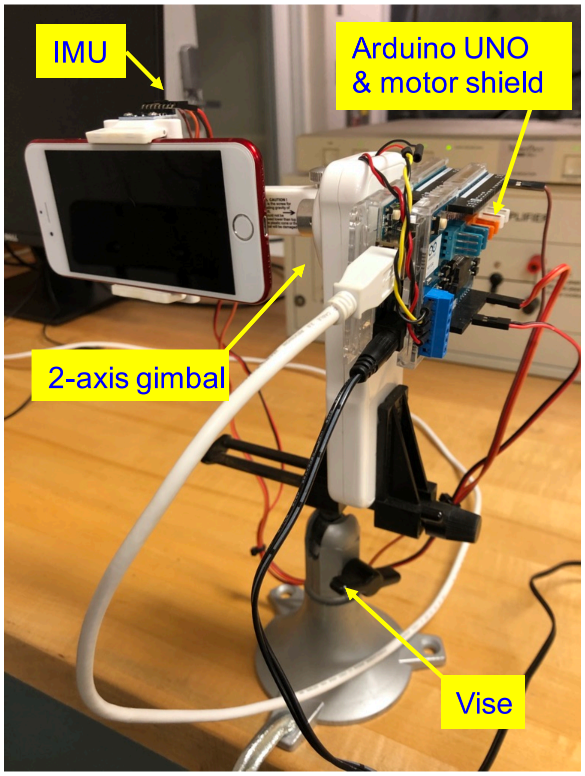

Figure 1: 2.004 dual-axis active gimbal system.

Experiment #1: Arduino Template File and IMU Sensor Offsets

1. Copy the entire contents in **Z:\Course Lockers\2.004\Labs\Lab 5** to Windows desktop or to a local folder. Alternatively, you can download all Lab 5 files from Canvas: `https://canvas.mit.edu/courses/10470/assignments/129631`.

[1]A commercial gimbal controller typically uses a proprietary control algorithm such as adaptive control, as well as an advanced tracking algorithm to command the gimbal.

2. Open the Arduino template file: "gimbal_control_template.ino" with the Arduino software. Read and understand the operation of this program. Connect the Arduino Uno board to the computer using the USB cable. On the top menu bar, go to **Tools** → **Board** and select the appropriate board, then **Tools** → **Serial Port** to select the appropriate COM port (usually the last one on the list).

3. All the needed variables have been defined for you in the Arduino template code "gimbal_control_template.ino". Take a look at the file and make sure you understand the operation of the code. You can comment out and/or uncomment the `#define` statements near the top of the template code to control things like serial output type or setpoint type.

4. Match your gimbal # to the Unit # in the Excel file "Dual-Axis-MPU-6050-offsets.xlsx" and type in the calibrated offset values into the 6 lines (that is, replace the zeros on Lines 154 – 159) indicated below in the Arduino template file inside the `setup()` function.

```
  /* Refer to your gimbal # and calibration spreadsheet */
  mpu.setXAccelOffset(0);
  mpu.setYAccelOffset(0);
  mpu.setZAccelOffset(0);
  mpu.setXGyroOffset(0);
  mpu.setYGyroOffset(0);
  mpu.setZGyroOffset(0);
```

   Note: Contact a lab staff if you wish to re-calibrate the IMU sensor since performing IMU calibration requires some extra steps.

5. In the template file we use the convention *variable_1* and *variable_2* to represent variables associated with pitch axis and roll axis respectively. Feel free to change any variable names and/or the code structure in the template file if you prefer.

   In the main loop we use the following function calls to command pitch and roll motors (Lines 289 and 298).

```
motorControl(DIR_A, PWM_A, error_1, d_error_1, sum_error_1, Kp_1, Kd_1,
    Ki_1); // pitch motor
motorControl(DIR_B, PWM_B, error_2, d_error_2, sum_error_2, Kp_2, Kd_2,
    Ki_2); // roll motor
```

   where `motorControl()` is a common function that is used to send a PWM command signal to a motor.

```
void motorControl(int DIR_x, int PWM_x, float error, float d_error, float
    sum_error, float kp_x, float kd_x, float ki_x)
{
  float pwm_command;
```

```
    Pcontrol = error * kp_x;
    Icontrol = sum_error * ki_x;
    Dcontrol = d_error * kd_x;

    Icontrol = constrain(Icontrol, -200, 200); // I control saturation limits
        for anti-windup

    pwm_command = Pcontrol + Icontrol + Dcontrol;

    if (pwm_command > 0)
    { digitalWrite(DIR_x, HIGH);
      analogWrite(PWM_x, (int) constrain(pwm_command, 0, 255));
    }
    else
    {
      digitalWrite(DIR_x, LOW);
      analogWrite(PWM_x, (int) constrain(abs(pwm_command), 0, 255));
    }
  }
```

Experiment #2: Controller Design

In the prelab we asked you to find the PD controller gains based on desired transient performance requirements and parameters from a known P control impulse response. That is, the PD controller gains K_p' and K_d' can be expressed as functions of the desired damping ratio, ζ', the desired natural frequency, ω_n', the initial P controller gain K_p and its closed-loop response parameters ζ and ω_n.

The closed-loop characteristic equation for a rotational plant, $\frac{K}{s(Js+B)}$, with a P controller is

$$s^2 + \frac{B}{J}s + \frac{KK_p}{J} = 0,$$

where J is the inertia, B the damping coefficient, K the system gain, and K_p the P controller gain. We can then equate terms of this equation to the standard second order system's response, $s^2 + 2\zeta\omega_n s + \omega_n^2$, and solve for $\frac{B}{J}$ and $\frac{K}{J}$ to get

$$\frac{B}{J} = 2\zeta\omega_n$$
$$\frac{K}{J} = \frac{\omega_n^2}{K_p}.$$

Now, we consider the characteristic equation for the closed-loop system with the PD controller

$$s^2 + \left(\frac{B + KK_d'}{J}\right)s + K_p'\left(\frac{K}{J}\right) = 0.$$

Then, equating terms with standard second order system's response, $s^2 + 2\zeta'\omega_n' s + \omega_n'^2$, we have

$$2\zeta'\omega_n' = \frac{B}{J} + K_d'\left(\frac{K}{J}\right)$$
$$(\omega_n')^2 = K_p'\left(\frac{K}{J}\right).$$

Substituting for $\frac{B}{J}$ and $\frac{K}{J}$ found earlier and solving for K_p' and K_d', we have

$$K_p' = K_p\left(\frac{\omega_n'}{\omega_n}\right)^2$$
$$K_d' = \frac{2K_p}{\omega_n^2}\left(\zeta'\omega_n' - \zeta\omega_n\right).$$

The supplied Matlab function "PD_design_from_P_impulse.m" will let you determine K_p' and K_d' using the above equations. Take a look at this function to understand its usage. You can type `help PD_design_from_P_impulse` at Matlab command prompt to get the syntax of this function. For this method to work we need to experimentally generate an impulse response with a pure P controller.

Steps:

1. Setup a pure proportional control with a "reasonable" gain: try $K_p = 10$ for the roll axis (Line 32) and set the rest of the controller gains to 0.0. Make sure to enable `MATLAB_SERIAL_READ` and disable `PRINT_DATA` so we can send data to Matlab for plotting and taking measurements.

2. Upload the code to Arduino UNO. Plug in the 9 VDC wall wart power adapter to the UNO board.

3. Use the provided Matlab script "SerialRead.m" to capture and plot the data in real-time. **Make sure you change the COM port number in the Matlab script to match the Arduino COM port**. Run the script and a Matlab figure window should appear with serial data being plotted in real-time.

4. While holding the pitch axis upright with one hand, use a finger to disturb the roll axis to generate an impulse response. Try several times if needed. Make sure the response has at least two oscillations, if not you may need to increase your K_p gain. Stop the data streaming by pressing any key on the keyboard.

5. Measure the offset from 0 degree as well as the first and second peak values and their respective timestamps from the response plot.

6. Provide all the required values to the Matlab function "PD_design_from_P_impulse.m" and run it (make sure you read and understand the script). It will then calculate ζ and ω_n from the impulse response, and design a PD controller for the roll axis. The desired

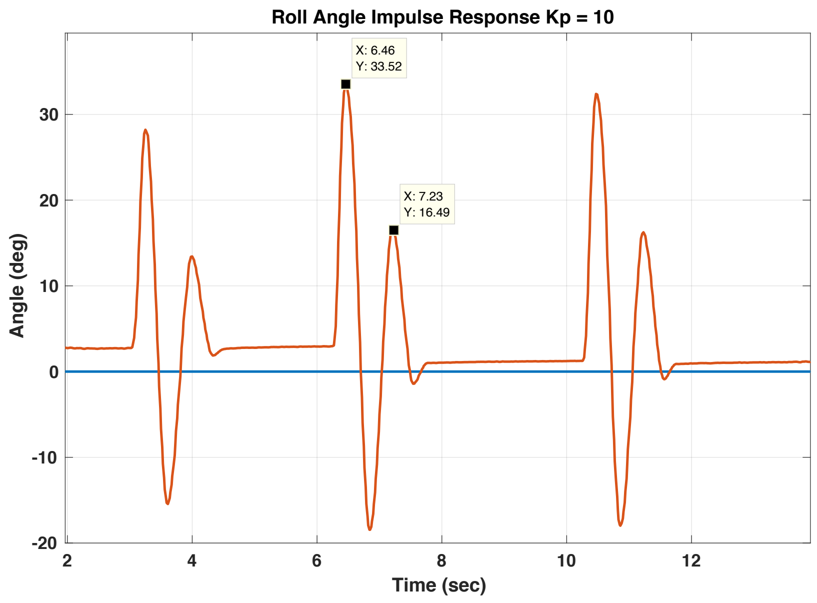

Figure 2: Captured impulse responses.

PD controller should produce a critically damped response with a bandwidth of 3 Hz ($\omega_{BW} \approx \omega_n = 2\pi f$), which means the closed loop system has two repeated poles at -18.85 rad/s. (You may reduce the bandwidth to 2 Hz if you notice any instability.)

7. Implement the PD controller on Arduino to stabilize the roll axis.

8. The PD control action should stabilize the roll axis so you don't need to hold it by hand. Use $K_p = 5$ or some other reasonable value for the pitch axis, and repeat the above steps to design a PD controller for the pitch axis based on the same closed loop specifications.

9. Implement the PD controller for the pitch axis.

10. With both PD controllers running, use a finger to test both axes and see if they are functioning properly. Capture a response plot when the system is driven by a square wave (enable `#define SQUARE_WAVE` near the top of the Arduino code) and verify the performance.

Experiment #3: Performance Evaluation

1. A simple first-order filter is provided in the code to reduce the effect of high frequency noise in the derivative signal:

```
filt_d_error_i = alpha * d_error_i + (1 - alpha) * filt_d_error_i;
```

which is the weighted sum of the current error derivative and the previous filtered error derivative. The weighting factor `alpha` is defined as

$$\alpha = \frac{\Delta T}{(\Delta T + \tau)}$$

where ΔT is the sample period and τ is the desired low-pass filter time constant. (This is the same low-pass filter we implemented in Lab 2 for filtering out high frequency noise in the derived wheel velocity signal.)

Use `filt_d_error_i` variables (`i=1,2`) to compute D control action for the PD controllers (Lines 289 and 298). Experiment with different weighting factor `alpha` if necessary, between 0 and 1, to see the effect of smoothing. Significant time delay may occur with aggressive filtering (smaller `alpha`) and may result in system instability.

Use the square wave on the roll axis while setting the pitch axis stationary (*i.e.*, change Line 49 to `#define set_point_1 0`). Change `Serial.print(variable_name)` in `#ifdef MATLAB_SERIAL_READ` (Lines 311 – 319) to output `d_error_2, filt_d_error_2`, and `roll_rate` (velocity signal from IMU). Capture a plot showing the three signals and compare them.

2. The IMU can also measure angular velocities directly with its built-in gyros, which are in general much better than taking the derivative approximation in the control loop. Now use `pitch_rate` and `roll_rate` instead of `d_error_1, d_error_2` or `filt_d_error_1, filt_d_error_2` to compute the D control action (Lines 289 and 298). Use the square wave setpoint to command both pitch and roll axes and capture the responses. Make sure to change back the lines in `#ifdef MATLAB_SERIAL_READ` (Lines 311 – 319) so we can output `set_point, pitch, roll`.

3. (Optional) Observe the behavior of the gimbal, and fine-tune your controllers if necessary.

4. The closed-loop bandwidth should be close to the natural frequency which was set to 3 Hz based on our design. Here we will use a sine wave set point with an amplitude of 15 degrees and a frequency of 1 Hz to command the roll axis of the gimbal. Capture and print out the waveforms and see if the response is able to track the reference sine wave.

5. Note that the default PWM frequency of Arduino UNO in this default configuration is 490.2 Hz, which makes the motors produce an audible noise. To eliminate this noise you can set the PWM frequency register to 31 kHz by enabling Line 198, and then upload the code.

```
// uncomment the line below to enable the highest PWM frequency to avoid
   audible noise
// may need to lower controller gains to avoid oscillation
TCCR2B = TCCR2B & B11111000 | B00000001; // for PWM frequency of 31372.55 Hz
```

Some motors may not be able to handle this high frequency PWM signal and may produce vibrations. In such case you may need to go back to the default frequency or use lower controller gains.

6. Now use the zero degree setpoint for both pitch and roll axes (commenting out all setpoints in the Arduino code), as we are ready to use it for its intended purpose. You may also want to disable both `MATLAB_SERIAL_READ` and `PRINT_DATA` to improve loop time. Disconnect the USB cable after the code has been uploaded.

   Carefully remove the gimbal from the vise. Hold the gimbal carefully with your hand since the wire connections are fragile. Move your hand to test the gimbal. If the controllers work properly the phone should stay stationary when you rotate and/or tilt the gimbal.

   Replace the dummy phone with your working phone if you want to check image stability. Note that the gimbal has weight and size limit so do not try to mount your phone if it is much larger or heavier than the dummy phone. You may also need to remove the phone from its case so it can fit properly. Use your phone's camera function to look at live video while moving the gimbal. If needed, unplug the Arduino UNO from the wall wart power supply and plug in the battery clip and turn on its switch.

★Additional Extra Credit Task★:

In today's extra credit task you will synchronize the motion of your gimbal system to track an oscillating mass suspended by a spring. Ask a staff member for a spring and mass. Using the spring constant and mass, calculate the frequency of oscillation and implement this frequency (in Hz) in the sine wave setpoint of the controller. Then, try to track the mass in the center of the camera while it is in motion. You may have to adjust the distance between the gimbal and the mass to ensure that the amplitude of the trajectory is within the field of view.

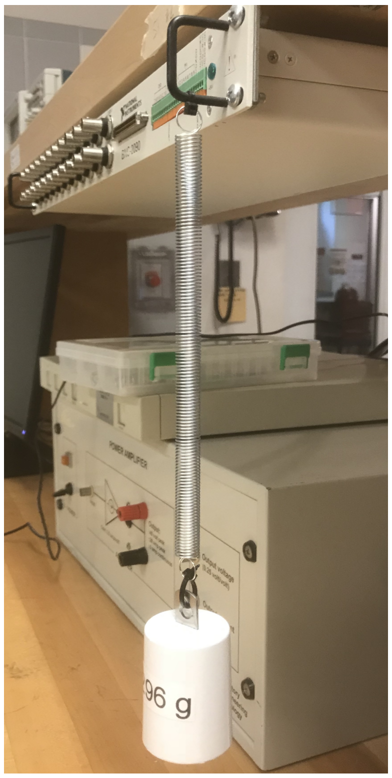

Figure 3: Mass-Spring system.

1. Open the camera of your phone and position the gimbal properly so that the camera can capture the whole trajectory when it pitches.
2. Tune the sine wave frequency so that the phone moves in phase with the oscillation of the mass. For your information:
   - Mass: $m = 0.296$ kg.
   - Nominal spring constant from OEM: $k = 17.52$ N/m. The actual spring constant may vary quite a bit so you may want to estimate the spring constant experimentally by observing the oscillation period or by measuring the static displacement.
3. Demonstrate your tracking system to a staff member.

Appendix: Moment of Inertia

If needed we can also estimate the inertia for pitch and roll respectively. The inertia J of the roll axis can be calculated based on the assumption that a modern smart phone can be approximated as a thin rectangular plate of height h, width w and mass m with axis of rotation at the center (see Fig. 4), and we get

$$J = \frac{m_1}{12}(h^2 + w^2)$$

For the pitch axis we assume the phone rotates about the pitch motor axis as a point mass of radius $r = \frac{h}{2}$ and mass $(m_1 + m_2)$.

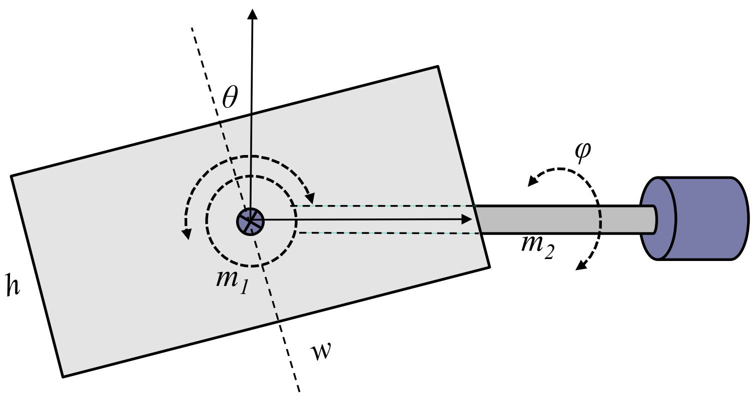

Figure 4: Moment of inertia of a rotating rectangular plate.

You will need to measure the size and weight of your phone to calculate the inertia. For your information the combined mass of the holder and sensor is 20 grams, and m_2 is about 110 grams. If you wish to use the provided iPhone 7 the dimension is 13.8×6.7 cm and the mass is 142 grams.