HW4 Balancing EduMiP

James Strawson 2016

EduMiP Modeling

Referring to Example 17.10 in Numerical Renaissance, the final equations of motion for the MIP are as follows.

$$(m_b R_w L \cos\theta)\ddot{\phi} + (I_b + m_b L^2)\ddot{\theta} = m_b g L \sin\theta - \tau \tag{1}$$

$$(I_w + (m_b + m_w)R_w^2)\ddot{\phi} + (m_b R_w L \cos\theta)\ddot{\theta} = m_b R_w L \dot{\theta}^2 \sin\theta + \tau \tag{2}$$

You will notice these are functions of torque, whereas we wish to design a controller that outputs a PWM duty cycle to our motors. Therefore we must also include the dynamics of the motors themselves. We can reasonably model a geared DC motor's output torque as a function of it's speed. Since there are two motors we include a coefficient of two in front of the equation. The motor specs listed below are for the simple DC motor without a gearbox, so we must include the gearbox ratio G_r in the motor model too. You can solve for the torque constant k with the below provided stall torque $\bar{s}$ and free run speed ω_f.

$$\tau = 2G_r(\bar{s} * u - k * \omega_m) \tag{3}$$

$$\omega_w = \dot{\phi} - \dot{\theta} = \omega_m / G_r \tag{4}$$

Where:
$\bar{s}$ = Motor Stall Torque
k = Motor Constant
ω_w = wheel speed
ω_m = motor armature speed
G_r = gearbox ratio
u = normalized motor duty cycle (value between -1 and 1)

Your assembled EduMiP has roughly the following physical properties:

1. Encoder disks have 15 slots and therefore provide 60 counts per motor armature revolution
2. Nominal battery voltage $V_b = 7.4V$
3. Motors (without gearbox) have free run speed of $\omega_f = 1760 rad/s$ at V_b
4. Motors (without gearbox) have stall torque $\bar{s} = 0.003 Nm$ at V_b
5. Motors are connected to the wheel with a gearbox ratio $G_r = 35.57$
6. The motor armature has inertia $I_m = 3.6 * 10^{-8} Kg * m^2$
7. Wheels have a radius $R_w = 34mm$
8. Wheels have a mass $M_w = 27g$ each
9. Total assembled MiP body has a mass $M_b = 263g$
10. MiP center of mass to wheel axis $L = 36mm$
11. MiP body inertia $I_b = 0.0004 Kg * m^2$ about the wheel axis

Just as we had to compensate for the torque of both motors accounting for the gearbox, we must also take the gearbox into account when modeling the inertia of the wheels. Attaching a rotating inertia through a gear ratio instead of directly attaching it to a shaft increases the effective inertia by the square of the gearbox. By modeling the wheels as disks and adding their inertia to the effective inertia of the motor armatures, we arrive at the following estimate of the combined wheel inertia.

$$I_w = 2 * ((M_w R_w^2)/2 + G_r^2 * I_m) \tag{5}$$

1 Problem 1 - Stabilizing Body Angle

1.1 task

Using the physical properties given above and the equations of motion provided in example 17.6 of Numerical Renaissance, develop a model G1(s) as a transfer function from normalized duty cycle u of the motors to angle θ of the MiP body in Radians. Now design a stabilizing transfer function $D_11(s)$ to keep MiP upright. I suggest plotting the angle of the body as the MiP falls over from a small angle with no input to the motors as a sanity check to make sure your unstable transfer function models the speed at which you predict MiP will fall over due to gravity.

1.2 What to Submit

1. Transfer function $G_1(s)$ from motor duty cycle u to angle θ. Please evaluate all coefficients and provide numeric constants.
2. Plot of the MiP angle θ as it falls over from a small angle with time scale in seconds at the bottom and angle in radians on the left.
3. Your design for a stabilizing controller $D_1(s)$.
4. Bode plot of the open-loop system $G_1(s) * D_1(s)$ indicating phase and gain margin.
5. Discrete time equivalent controller $D_1(z)$ for a 200Hz sample rate. Also include the difference equation you will use in your C software implementation later.

2 Problem 2 - Stabilizing Wheel Position

You may notice that although your controller D_1 keeps your MiP upright, it still wants to drive away. Now design a second controller, D_2, to stabilize the wheel position ϕ. Since the dynamics of Theta and Phi are very different, you may use the successive loop closure method described in section 18.3.4 of Numerical Renaissance.

2.1 task

You may initially design the controller D_2 assuming the inner loop has a constant gain of 1. This is to say that we assume the feedback of $D_1(s) * G_1(s)$ is sufficiently fast that we treat it as 1 for the purpose of simplifying the design of D_2 at a much slower timescale. When you think you have a stabilizing controller, include your model for $G_1(s)$ and controller $D_1(s)$ to check performance. You may need to add a prescaler P to the output of your controller $D_2(s)$ if the feedback of $G_1(s) * D_1(s)$ has steady state gain not equal to 1. With the entire successive loop closure system modeled in Matlab, plot the step response for a unit step in wheel position. This should model the MiP wheels moving forward one radian. You should observe non-minimum phase behavior where the wheels must roll backwards briefly before rolling forward.

2.2 What to Submit

1. Your model for $G_2(s)$ and controller $D_2(s)$
2. Root locus demonstrating stability of outer loop with inner loop gain of 1
2. Continuous time step response assuming inner loop gain of 1
3. Continuous time step response with your $G_1(s)$ and $D_1(s)$ inner loop model included
4. Discrete time transfer function $D_2(z)$ for a sample rate of 20Hz.
5. Difference equation for $D_2(z)$ that you will implement in software.

3 Problem 3 - Software Implementation

3.1 Add to Your Complementary Filter Code

Now you will write software to make your two controllers work. Start by modifying your complementary filter code to also estimate the angle Phi by averaging the encoder readings for each wheel position. Then run a simple test that drives the motors proportionally to the angle estimate theta. This will not stabilize the system but will indicate to you that your function calls to drive the motors work as expected. A positive control input u should drive both motors forward. Note that one motor is mounted to the chassis backwards relative to the other! you will have to compensate for this in your software.

3.2 Implement D_1

Now try implementing your difference equation for D_1. It is advisable to start the controller when the MiP is picked upright and your theta angle estimate is within a small range of zero. It is also advisable to stop the controller if theta exceeds a certain range indicating it has fallen over. Remember to "zero-out" the controller by setting the past inputs and outputs to 0 before evaluating the difference equation for the first time. This should all be done inside the IMU interrupt function at 200Hz. It is also advisable to have a tunable proportional feedback gain K in your code for quick fine-tuning without needing to re-evaluate the transfer function coefficients. Right now the reference angle theta should be 0 and your complementary filter should include an offset to account for the mounting angle of the BeagleBone relative to the MiP body.

3.3 Implement D_2

When your EduMiP successfully stands upright, it will want to wander as there is no feedback on the position yet. Now create a separate thread which implements the outer loop difference equation for D2. This should use a ϕ angle reference value of 0 (standing still) and generate a new non-zero reference angle theta for the inner-loop controller D_1 to use. Remember to "zero-out" this controller as well by either setting the encoder positions to 0 when the MiP is picked up or saving the encoder positions at that time to use as the new reference angle *phi*.

3.4 Restrictions

1. You must use your own complementary filter for the angle estimate theta. You may not use the Euler Angles provided by the IMU API.
2. D_1 and D_2 must operate at different frequencies in separate threads.
3. You may use the provided balance example as a reference, but you must write the code that you submit yourself.

3.5 What to Submit

Include as the final part of your submission PDF your complete balance code. If possible, please color-code your software to make it easier to read. Most free software-oriented text editors such as Sublime-text and Notepad++ will do this automatically for you. Also include a video with your email submission demonstrating balancing performance.

4 Extra Credit - Steering Controller

4.1 Task

Since we average the encoder readings to generate our angle estimate Phi, the MiP can (and does) spin about its vertical axis while balancing. For extra credit, design and implement a proportional, PD, or Lead controller to keep the MiP pointing straight ahead if twisted by an outside disturbance. In your code you should estimate the turn angle in radians by taking the difference of the two encoder counters and applying the wheel diameter and track width of the MiP to estimate its turn angle. The output of your controller should be a normalized duty cycle (between -1 and 1) that is added to one motor and subtracted from the other such that a positive control input generates an increase in turn angle.

4.2 Further Questions

Lets say a large disturbance is detected and the steering controller outputs 1 which is then subtracted from D_1's output to one motor and added to the other. If the controller D_1 is also demanding torque from the motors, one motor will saturate (can't drive any more than +-1). What has now happened to the real control output of D1 to the motors? How can we avoid this problem? Should D_1 or the steering controller have higher priority in case of saturation? Why is integral feedback not particularly helpful for this steering controller?

5 Submission

Please email a single PDF to submissions@renaissance-press.com containing all plots and code requested in the above sections. Also attach to this email a short video demonstrating the performance of your EduMiP running **your own software**.