Justin Chang

MAE 144 UCSD

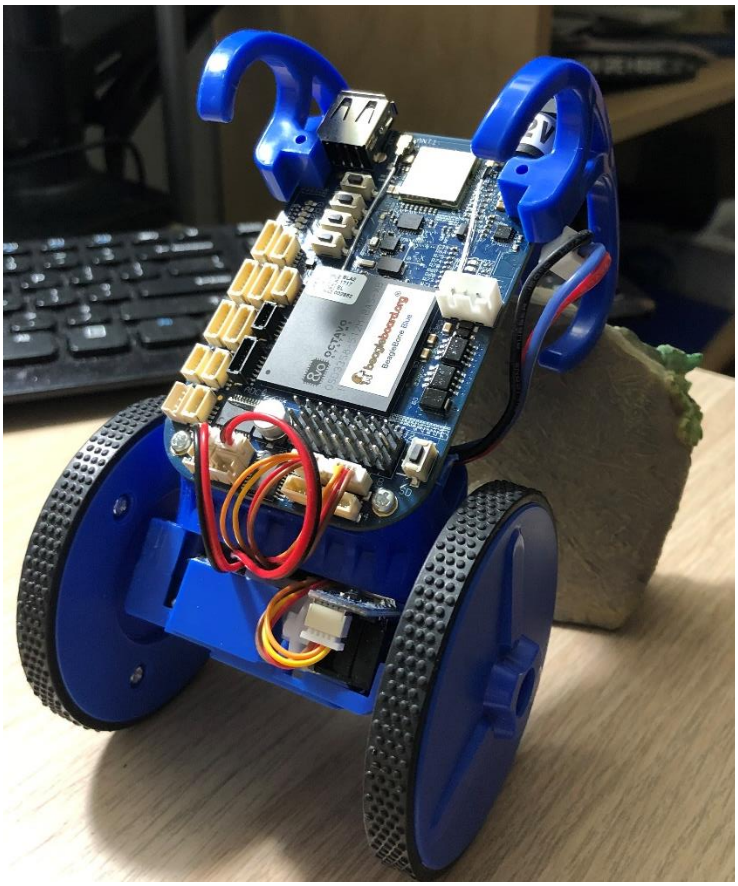
12/13/19

Group 8: Basil Hu, Khang Nguyen

MiP Project Part 1

MiP Balance Project

Figure 1: MiP Demonstration



In order to balance the EduMiP, the dynamics of the system must be studied first. The equations of motion in the following are similar to that of an inverted pendulum on a cart:

$$(I_w + (m_w + m_b)r^2)\ddot{\phi} + (m_b rl cos\theta)\ddot{\theta} - (m_b rl sin\theta)\dot{\theta}^2 = \tau \ldots (1)$$

$$(m_b rl cos\theta)\ddot{\phi} + (I_b + m_b l^2)\ddot{\theta} - (m_b gl sin\theta) = -\tau \ldots (2)$$

With θ as the body's angle and ϕ as the wheel angle.

Physical properties are as follows:

m_w	wheel mass
m_b	Total assembled MiP body mass
r	Wheel radius
l	MiP center of mass to wheel axis
I_b	MiP body inertia
gravity	gravity

The given equations for connecting motor torque and speed are the following:

$$\tau = 2G(\bar{s}u - kw_m) \ldots (3)$$

$$w_w = \dot{\phi} - \dot{\theta} = \frac{w_m}{G} \ldots (4)$$

To find the torque constant, I divided the stall torque $\bar{s}$ by the free run speed w_f (given).

$$k = \frac{\bar{s}}{w_f} \ldots (5)$$

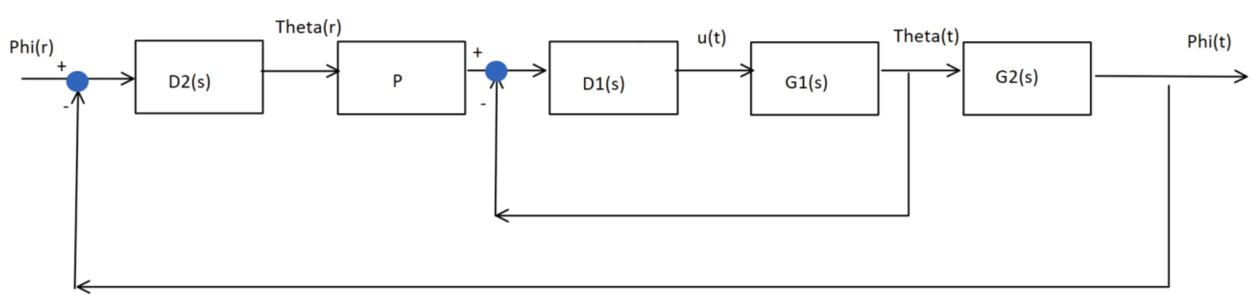
In order to find the combined wheel inertia, the following estimate of the combined wheel inertia was used:

$$I_w = 2\left(\frac{m_w r^2}{2} + G^2 I_m\right) \ldots (6)$$

Now, all constants are known except for θ, ϕ, and u (motor's normalized duty cycle)

The design of the system and the closed-loop system is depicted in the following figure:

Figure 2: System's Block Diagram



The fast controller D1(s) takes the relationship between the microcontroller's PWM duty cycle and the MiP body angle. This loop has to be fast due to the volatility of the body angle becoming unstable relatively quickly from gravity. The frequency of the inner loop was manually set at 100Hz to compensate for the required “fast” response to the system. Once the inner loop is stabilized, the outer loop stabilizes the whole MiP’s location at a lower frequency. Since the physical location of the MiP is related to the wheel angle, I used ϕ as my reference for tracking. The outer loop D2(s) is at a lower frequency because it isn’t as important to stabilize the wheels compared to the MiP’s body angle. Therefore, the frequency was set at 20 Hz which is significantly less than the fast loop’s 100Hz.

Note: The loop prefactor P was added into the system because there isn’t realistic asymptotic tracking for my inner loop as my analysis later shows.

Fast Controller Design:

Next, we want to solve the equations of motion with the “fast” controller - namely, the transfer function between θ and u.

Subbing in Equation 4 into Equation 3yields the following equation:

$$\tau = 2G\left(\bar{s}u - \frac{\bar{s}}{w_f}G(\dot{\phi} - \dot{\theta})\right) \dots (7)$$

Plugging in Equation 7 to Equation 1 & 2:

$$(I_w + (m_w + m_b)r^2)\ddot{\phi} + (m_b r l cos\theta)\ddot{\theta} - (m_b r l sin\theta)\dot{\theta}^2 = 2G\left(\bar{s}u - \frac{\bar{s}}{w_f}G(\dot{\phi} - \dot{\theta})\right) \dots (8)$$

$$(m_b r l cos\theta)\ddot{\phi} + (I_b + m_b l^2)\ddot{\theta} - (m_b g l sin\theta) = -2G\left(\bar{s}u - \frac{\bar{s}}{w_f}G(\dot{\phi} - \dot{\theta})\right) \dots (9)$$

Rearranging Equation 8 and 9 yields the following:

$$(I_w + (m_w + m_b)r^2)\ddot{\phi} + \frac{2G^2\bar{s}}{w_f}\dot{\phi} + (m_b r l cos\theta)\ddot{\theta} - \frac{2G^2\bar{s}}{w_f}\dot{\theta} - (m_b r l sin\theta)\dot{\theta}^2 = 2G\bar{s}u$$

$$(m_b r l cos\theta)\ddot{\phi} - \frac{2G^2\bar{s}}{w_f}\dot{\phi} + (I_b + m_b l^2)\ddot{\theta} + \frac{2G^2\bar{s}}{w_f}\dot{\theta} - (m_b g l sin\theta) = -2G\bar{s}u$$

Assuming small angles where $sin\theta = \theta$, $cos\theta = 1$, and $\dot{\theta}^2 = 0$ results in the following two equations:

$$(I_w + (m_w + m_b)r^2)\ddot{\phi} + \frac{2G^2\bar{s}}{w_f}\dot{\phi} + (m_b r l)\ddot{\theta} - \frac{2G^2\bar{s}}{w_f}\dot{\theta} = 2G\bar{s}u \dots (10)$$

$$(m_b r l)\ddot{\phi} - \frac{2G^2\bar{s}}{w_f}\dot{\phi} + (I_b + m_b l^2)\ddot{\theta} + \frac{2G^2\bar{s}}{w_f}\dot{\theta} - (m_b g l)\theta = -2G\bar{s}u \dots (11)$$

For simplicity's sake, I will use the following variables:

a = $I_w + (m_w + m_b)r^2$

b = $\frac{2G^2\bar{s}}{w_f}$

c = $m_b rl$

d = $2G\bar{s}$

e = $m_b rl$

f = $I_b + m_b l^2$

g = $m_b gl$

After Laplace transforming Equation 10 and 11,

$$(as^2 + bs)\Phi + (cs^2 - bs)\Theta = dU \ldots (12)$$

$$(es^2 - bs)\Phi + (fs^2 + bs - g)\Theta = -dU \ldots (13)$$

then multiplying 12 by $(es^2 - bs)$ and 13 by $(as^2 + bs)$ and having equation 12 minus 13 gives the following:

$$[(es^2 - bs)(cs^2 - bs) - (as^2 + bs)(fs^2 + bs - g)]\Theta = d(es^2 - bs + as^2 + bs)U \ldots (14)$$

Finally, the Transfer function between Θ and U is expressed.

$$\frac{\Theta}{U} = \frac{d(es^2 - bs + as^2 + bs)}{(es^2 - bs)(cs^2 - bs) - (as^2 + bs)(fs^2 + bs - g)} \ldots (15)$$

After some cleaning up, Equation 15 becomes:

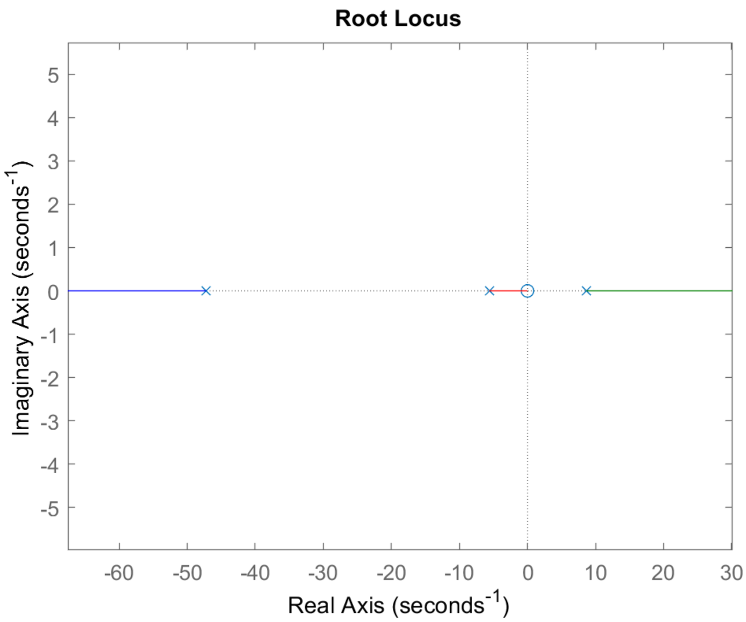
$$\frac{\Theta}{U} = \frac{d(e + a)s^2}{(ec - af)s^4 + (-eb - bc - ab - bf)s^3 + (ag)s^2 + (bg)s}$$

Plugging in the physical properties into MATLAB:

$$G1 = \frac{\Theta}{U} = \frac{-882.7s}{s^3 + 44.15s^2 - 192.8s - 2299} \ldots (16)$$

With the root locus plot looking like this:

Figure 3: Root locus of G1(s)



The root locus plot demonstrates that with just a proportional controller K the G(s) cannot stabilize. Therefore, I have decided to use a controller with a pole at the origin and a zero. The pole at the origin is intended to have asymptotic tracking and a pole-zero cancellation of the system's zero. The zero is located at the left of the system's pole so that the pole at around -48 can go to that new zero and maintain stability. The gain is left at negative so that the root locus may reach stability as the right-hand side pole moves to the left and eventually reaching stability. To achieve the qualities above, my controller of choice was the proportional integral (PI) controller:

$$D1 = -\frac{(s+8)}{s} \dots (17)$$

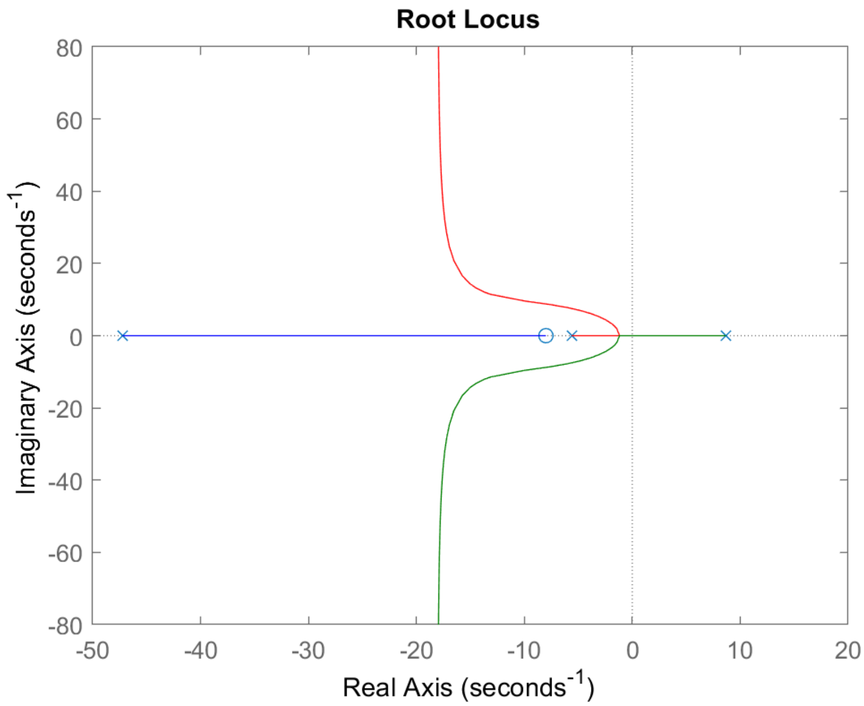
Gain: -1

Pole: 0

Zero: -8

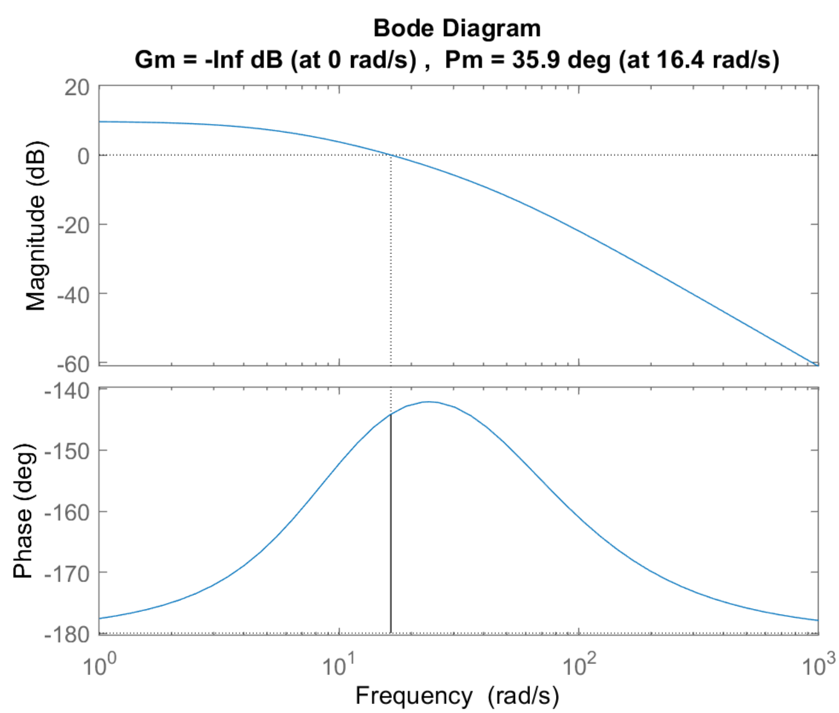
The root locus after implementing my controller looks like the following:

Figure 4: Root Locus of Open Loop System



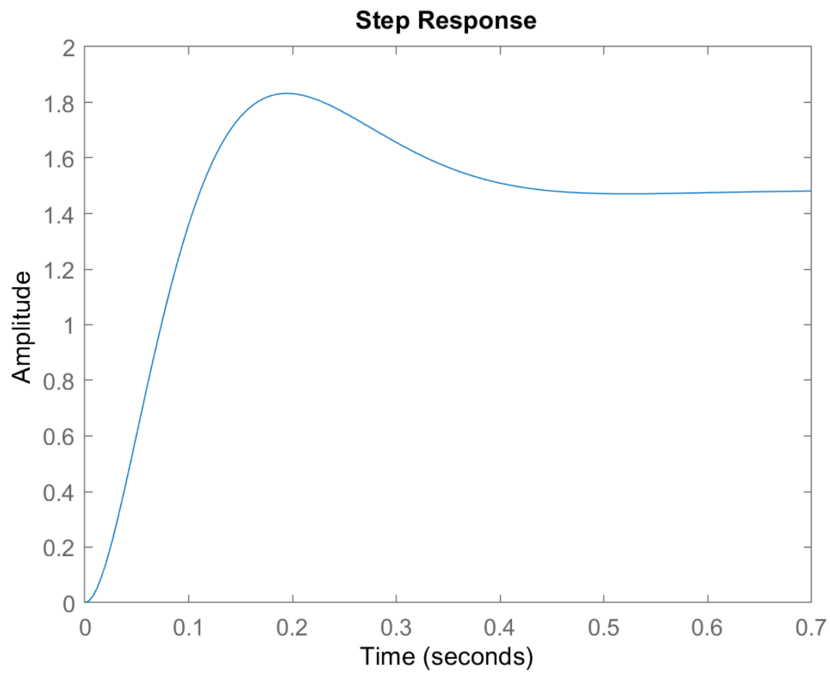
The bode plot of the open-loop system G1(s)D1(s) has the following features:

Figure 5: Bode Plot of Open Loop System G1(s)D1(s)



The plot shows a phase margin at around 35. 9 degrees for 16.4 rad/s. There is no gain margin. The bode plot maintains tracking at low frequencies but also rejection at high frequencies, hence making the system more robust.

Figure 6: Step Response of Closed-Loop System



The step response shows that there's no asymptotic tracking as the value leads to 1.48. Therefore, a loop prefactor is needed in the outer loop to ensure asymptotic tracking.

Since this is a fast controller, I ensured that the rise time is less than 0.2 seconds. To verify, I estimate my rise time to be $t_r = \frac{1.8}{w_g} = \frac{1.8}{16.4} = 0.1098s$ which is exactly what I needed.

Lastly, to convert my controller into discrete time with a 100Hz sample rate, I used the MATLAB command c2d with Tustin's approximation to get

$$D1(z) = \frac{-1.04z + 0.96}{z - 1} \ldots (18)$$

An interesting note to make when identifying the corrected phase margin is to take into account the phase loss due to my method of finding D(s). I used a discrete equivalent design that has a cascade of effect from an ADC – D(z) – DAC. The h/2 (h as sample interval) delay results from the zero-order hold of the DAC. With this method, a significant amount of extra phase lead at the gain crossover frequency should be built into the CT control design D(s) to compensate. I accounted for the effect of the h/2 delay by including a Pade approximation of the delay in series with D(s). Since the Pade approximation only affects the phase of the Bode Plot with a phase gain of around 15 degrees, I added 15 degrees to my phase margin bumping it up to 50.9 degrees.

Slow Controller Design:

After implementing the first controller, the MiP will want to wander around as there is no feedback on the physical position yet. To compensate that, an outer loop controller is required.

To derive the transfer function for G2(s) – relationship between ϕ and θ – we continued from equations 12 and 13.

$$(as^2 + bs)\Phi + (cs^2 - bs)\Theta = dU \ldots (12)$$

$$(es^2 - bs)\Phi + (fs^2 + bs - g)\Theta = -dU \ldots (13)$$

Adding equation 12 and 13 gives the following expression:

$$(a + e)s^2\Phi = -\Theta((\mathrm{c} + \mathrm{f}) - \mathrm{g}) \ldots (19)$$

Then the transfer function between ϕ and θ becomes:

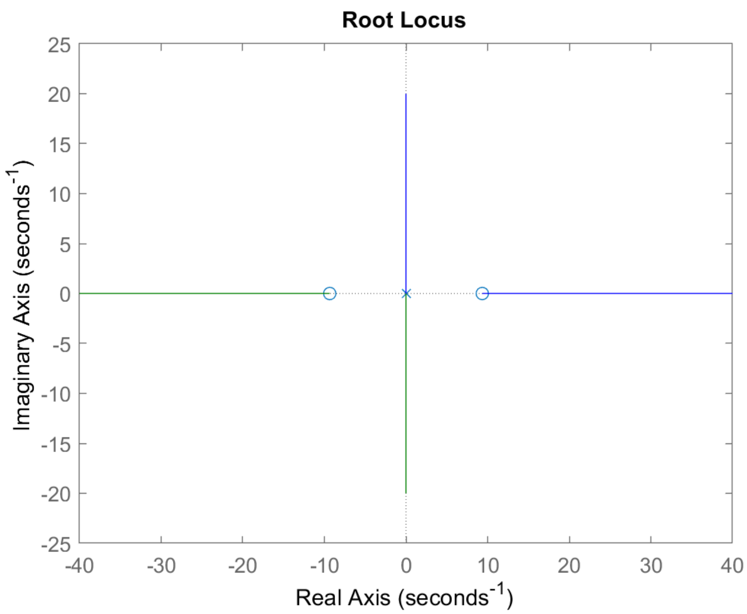
$$\frac{\Phi}{\Theta} = \frac{-(c + f)s^2 + g}{(a + e)s^2} \ldots (20)$$

Plugging in the values in MATLAB gives the numerical result:

$$G2 = \frac{-1.476s^2 + 128.9}{s^2}$$

The root locus looks like the following:

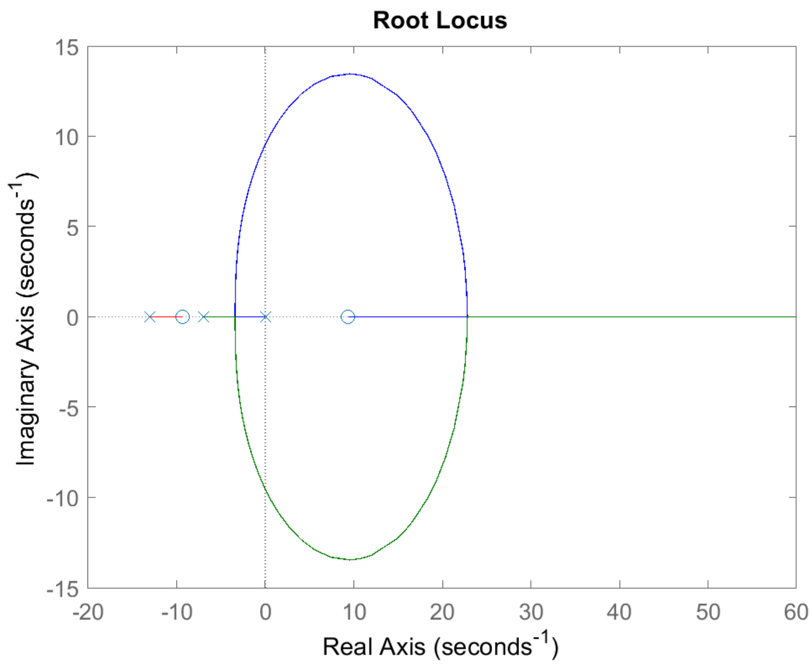
Figure 7: Root Locus Of G2(s)



In order to make this system stable, I designed proportional derivative (PD) controller with an extra pole. The PD controller was so that the poles don't branch off at zero but instead at the right-hand side. This is beneficial because at small gains I can make the system stable. Next, the pole for the PD I designed it to the left of the zero at -10. This is so that with any gain that closed loop pole will always be stable. At high gains, the system will branch out and reach the right-hand side zero and another one to infinity. With that in mind, I kept my gain low at 1. All in all, my controller looks like the following:

$$D2 = \frac{s}{(s+13)(s+7)}$$

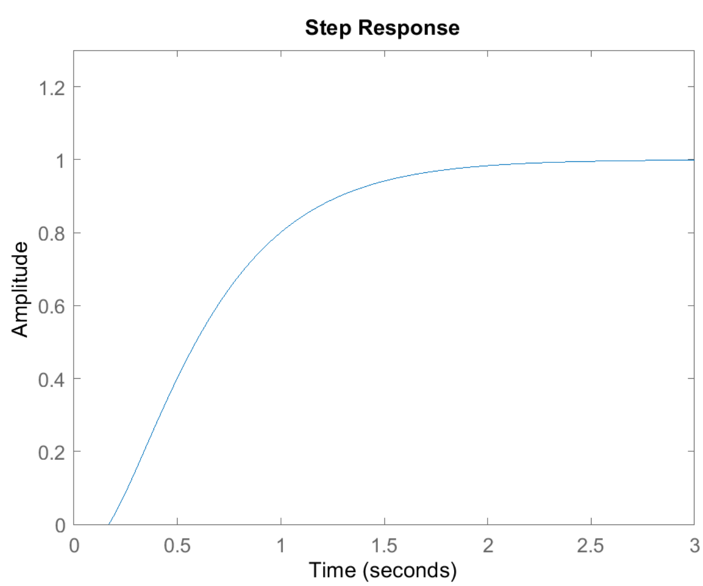
Figure 8: Root Locus Of Open Loop System



The root locus demonstrates the stability of the outer loop with the assumption that the inner loop has a gain of 1.

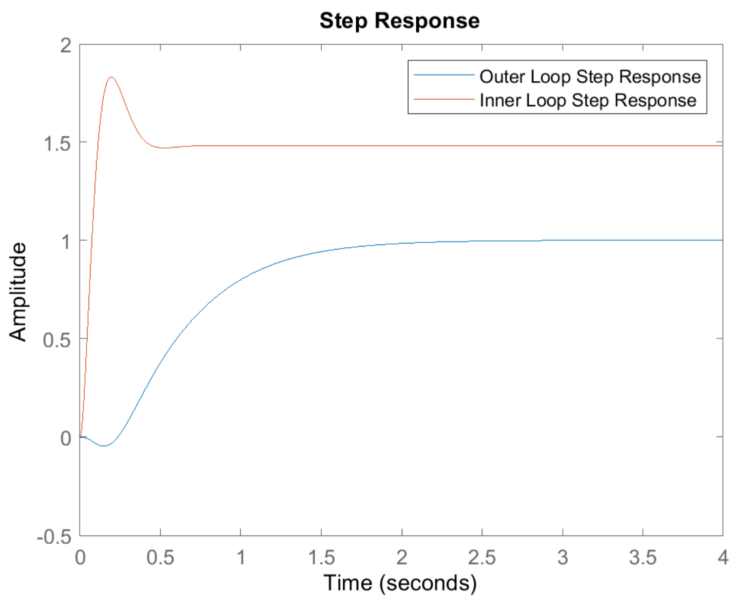
To verify, we plotted the ideal step response (inner loop gain of 1).

Figure 9: Step Response of Ideal Closed Loop System



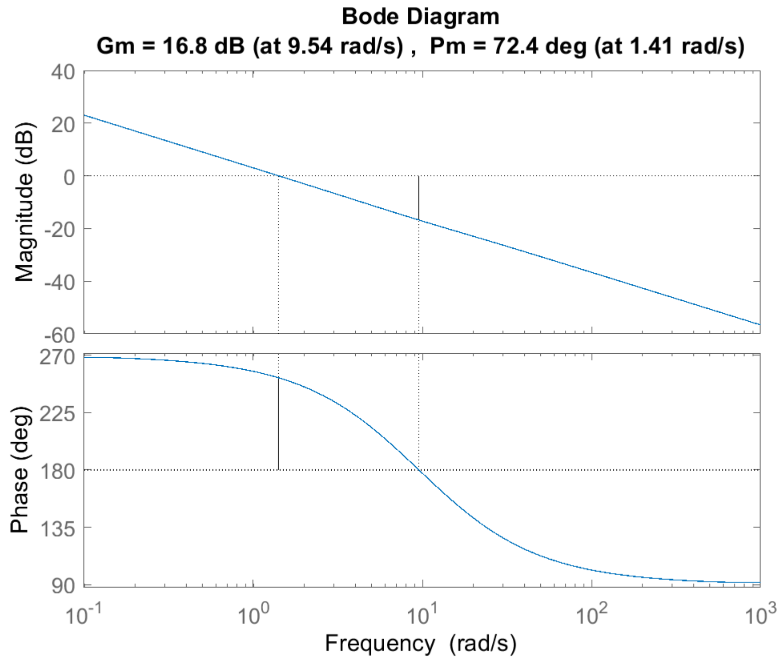
The step response indeed reaches asymptotic tracking and is stable. Now, my realistic model of G1(s) and D1(s) after multiplying the loop prefactor [1/(steady state value from the inner loop's step response)=1/1.48] is needed to see how my loops together. I then plotted the step response for the inner loop and the realistic outer loop (with inner loop included).

Figure 10: Inner & Outer Loop Step Response



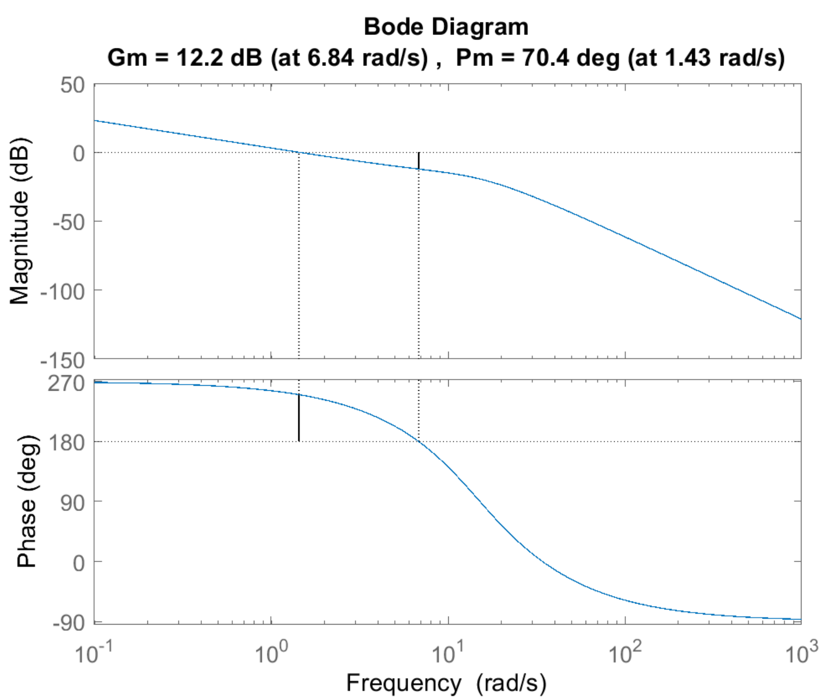
The open loop bode shows that there is a feasible gain margin of 16.8dB at 9.54 rad/s and phase margin of 72.4 degrees at 1.41 rad/s. Having 72.4 degrees as my phase margin is really good as there's a lot of leeway before instability. In reality, similar to how D1(s) needed a 15-degree phase bump due to Pade' approximation, my phase margin increases to around 87.4 degrees.

Figure 11: Bode Plot of Open Loop G2(s)D2(s)



Finally, to see how the full outer loop with the actual closed inner loop T1(s) of the successive loop closure problem, the Bode Plot looks like the following:

Figure 12: Bode Plot of Full Outer Loop



For the full outer loop, there is a 12.2 dB gain margin at 6.84 rad/s and a 70.4 degree phase margin for 1.43 rad/s. This means with the inner loop included, the whole system should theoretically be stable with a lot of leeway before instability.

Finally, using Tustin's approximation, the D2 controller with 20Hz at discrete time is the following:

$$D2(z) = \frac{0.01606z^2 - 0.01606}{z^2 - 1.212z + 0.3577}$$

Resources Used:

For this report, I used MATLAB to design my controller and see my system's root locus and Bode Plot behaviors. I then wrote my script in C using Microsoft's Visual Studio Code. Lastly, I implemented my code using Putty to have access with SSH and transferred files using WinSCP.

Appendix A: Balance Code

```c
/**
 * @file MiP_final
 *
 * This program calculates the angle that the MIP is currently at
 *  with a starting point at the vertical axis.
 *
 * Once the angle is found, then it implements the controller connecting
 * the PWM duty cycle to the MiP orientation.
 * This calculation is inside the fast loop as corrections for the MiP body
 * angle is volatile. Frequency for the loop is set at 100Hz.
 *
 * Once the body is stabilized, the next controller needed is to stabilize
 * the position of the angle. This means that the wheels need to be stabilized.
 * For this controller, since it's not as important as the body angle volatility,
 * is a slower controller at 20Hz.
 *
 *
 *
 */
#include <stdio.h>
#include <robotcontrol.h> // includes ALL Robot Control subsystems


// The time constant is set to slightly favor the accelerometer
// as 1/1.7>0.5 and favoring the low pass filter
#define timeconstant 1.7  //time constant set as 1.7;
#define dt 0.01 //dt is 0.01s since it's 100Hz


// function declarations
void on_pause_press();
void on_pause_release();

//interrupt declaration
void theta_calculations(void);
//pthread declaration
void* slow_loop(void* reference);

// function declarations for sensor readings
// gyroscope filter
float highpass(float gain,float input);
//accelerometer filter
float lowpass(float gain,float input);
```

```c
/**
 * This template contains these critical components
 * - start the signal handler
 * - initialize subsystems
 * - while loop that checks for EXITING condition
 * - cleanup subsystems at the end
 * - This module incorporates pthreads & interrupts
 * - Interrupt to do the calculations
 * - pthread with lower priority for slower loop
 * @return    0 during normal operation, -1 on error
 */


//initialie the variables

// for the low-pass filter difference equation
float low_oldoutput=0;
// for the high pass filter difference equation
float high_oldoutput = 0;
float high_oldinput = 0;


// Variables for the calculation of theta
float theta_a_raw;
float theta_a; //theta_a after filter

float theta_g_raw;
float theta_g_raw_old = 0; //needed because it's an integration formula
float theta_g; //theta_g after filter

//theta_f = theta_a + theta_g
float theta_f;
float theta_f_old;


//output for D1
float duty_old = 0;
float duty;
```

```c
//output for D1
float duty_old = 0;
float duty;

// old variables for D2 & D1 controller
float theta_diff_old;
float inputd2_old2;
float outputd2_old2;
float inputd2_old1;
float outputd2_old1;

//left motor in radians
float L_rad=0;
//right motor in radians
float R_rad =0;
// average left and right motor
float phi = 0;
//output of the outer loop controller
float theta_ref = 0;

// initialize the conversion factor c
// ppr = 35.577 * 15 * 4 = 2134.62
// c = 2*pi/ppr = 0.0029434678
const float c = (2*3.141592)/(35.577 * 15 * 4);
static float k1 = 1;
static float k2 = 0.6;

//function declarations for slow & fast controller
float d1(float theta_diff);
float d2(float phi_diff);

float gain = dt/(timeconstant+dt); //define the gain
float wc = 1/timeconstant;


// Preload a file for csv
char filename[] = "Justin";
FILE *filepointer;
```

```c
int main()
{
    //open the file needed to be exported
    filepointer=fopen(strcat(filename,".csv"),"a"); //opens a file for reading and appending

    // make sure another instance isn't running
    // if return value is -3 then a background process is running with
    // higher privaledges and we couldn't kill it, in which case we should
    // not continue or there may be hardware conflicts. If it returned -4
    // then there was an invalid argument that needs to be fixed.
    if(rc_kill_existing_process(2.0)<-2) return -1;

    // start signal handler so we can exit cleanly
    if(rc_enable_signal_handler()==-1){
            fprintf(stderr,"ERROR: failed to start signal handler\n");
            return -1;
    }

    // initialize pause button
    if(rc_button_init(RC_BTN_PIN_PAUSE, RC_BTN_POLARITY_NORM_HIGH,
                                            RC_BTN_DEBOUNCE_DEFAULT_US)){
            fprintf(stderr,"ERROR: failed to initialize pause button\n");
            return -1;
    }

    // initialize hardware
    rc_mpu_config_t conf = rc_mpu_default_config();

    //initialize mpu with dmp
    if(rc_mpu_initialize_dmp(&data, conf)){
        printf("rc_mpu_initialize_failed\n");
        return -1;
    }



    // Assign functions to be called when button events occur
    rc_button_set_callbacks(RC_BTN_PIN_PAUSE,on_pause_press,on_pause_release);

    //Initialize motors
    rc_motor_init();
    //initialize encoders
    rc_encoder_eqep_init();
```

```c
    //interrupt functions
    rc_mpu_set_dmp_callback (&theta_calculations);

    //create thread
    pthread_t slow_thread = 0;

    //initialize the pthreads
    if(rc_pthread_create(&slow_thread, slow_loop, (void*) NULL, SCHED_OTHER, 0)){
        fprintf(stderr, "failed to start thread\n");
        return -1;
    }

    // make PID file to indicate your project is running
    // due to the check made on the call to rc_kill_existing_process() above
    // we can be fairly confident there is no PID file already and we can
    // make our own safely.
    rc_make_pid_file();

    printf("\nDone with initializing\n");

    printf("Hold the pause button for 2 seconds to exit\n");
    //set initial state to running
    rc_set_state(RUNNING);

    // run the !EXITING state loop inside main
    while(rc_get_state()!=EXITING){
                //run green LED if RUNNING
                if(rc_get_state()==RUNNING){
                        rc_led_set(RC_LED_GREEN, 1);
                        rc_led_set(RC_LED_RED, 0);
                }
                else{
                // run red LED if PAUSED
                        rc_led_set(RC_LED_GREEN, 0);
                        rc_led_set(RC_LED_RED, 1);
                }
                rc_usleep(100000);
                // always sleep at some point
    rc_usleep(10000); //100 Hz
    }
```

```
    //join threads
    rc_pthread_timed_join(slow_loop,NULL,1.5);
        //turn off LEDs
        rc_led_set(RC_LED_GREEN, 0);
        rc_led_set(RC_LED_RED, 0);
    //close file
    fclose(filepointer);
    // close file descriptors
        //clean up LED
        rc_led_cleanup();
    rc_cleanup();
    rc_mpu_power_off();
    rc_button_cleanup();  // stop button handlers
    rc_remove_pid_file(); //remove pid file LAST
    return 0;
}


/**
 * Make the Pause button toggle between paused and running states.
 */
void on_pause_release()
{       //toggle from running to pause
        if(rc_get_state()==RUNNING){
            rc_set_state(PAUSED);
        }
        //toggle from pause to running
        else if(rc_get_state()==PAUSED){
            rc_set_state(RUNNING);
        }
        return;
}
```

```
/**
* If the user holds the pause button for 2 seconds, set state to EXITING which
* triggers the rest of the program to exit cleanly.
**/
void on_pause_press()
{
        int i;
        const int samples = 100; // check for release 100 times in this period
        const int us_wait = 2000000; // 2 seconds
        // now keep checking to see if the button is still held down
        for(i=0;i<samples;i++){
                rc_usleep(us_wait/samples);
                if(rc_button_get_state(RC_BTN_PIN_PAUSE)==RC_BTN_STATE_RELEASED) return;
        }
        printf("long press detected, shutting down\n");
        rc_set_state(EXITING);
        return;
}


//Low pass filter difference equation
float lowpass(float gain,float input){
        //difference equation for a low pass filter
        float output = gain*input + (1-gain)*low_oldoutput;
        low_oldoutput = output;
        return output;
}

//High pass filter difference equation
float highpass(float gain,float input){
        //difference equation for a high pass filter
        float output = gain*(high_oldoutput + input - high_oldinput);
        high_oldoutput = output;
        high_oldinput = input;
        return output;
}
```

```c
/**
 * This function calculations inside the fast loop (interruption)
 * averages out the gyroscope and accelerameter reading
 * After it finds a reading, then apply the controller
 * to find what the duty cycle is.
 * This function also applies the duty cycle to the motors
 * to balance the MiP body angle
 *
 * If the angle is over 1 radian, then give up the program
 *
 */
void theta_calculations(void)
{
    // read sensor data with accelerometer
    if(rc_mpu_read_accel(&data)<0)
    {
        printf("read accel data failed\n");
    }
    // read sensor data with gyroscope
    if(rc_mpu_read_gyro(&data)<0)
    {
        printf("read gyro data failed\n");
    }

    //collect data for theta_a_raw
    theta_a_raw= atan2(-data.accel[2],data.accel[1]);
    //collect data for theta_g_raw from euler approximation
    //0.01 because that's the delta T for 100Hz
    theta_g_raw = theta_g_raw + (data.gyro[0]*DEG_TO_RAD*0.01);
    //update the old as the new
    theta_g_raw_old = theta_g_raw;
    //low pass filter for theta_a
    theta_a = lowpass(gain,theta_a_raw);
    //high pass filter for theta_g
    theta_g = highpass(1-gain,theta_g_raw);
    //theta_f is the sum plus the offset for the way MiP naturally balances
        // At this offset angle is the MiP should naturally stand still
    theta_f = theta_a + theta_g+0.25;
        //Apply the fast controller with loop prefactor 1/1.48
        duty = d1(theta_ref/1.48 - theta_f);

```

```c
        //set the position of the motors
        // one is CW the other is CCW
        rc_motor_set(2,-duty);
        rc_motor_set(3,duty);

        // if the MiP body angle is over +/- (1 radian = 57 degrees)
        // then just give up
        if(theta_f>1)
        {
            rc_set_state(EXITING);
            return;
        }
        if(theta_f<-1)
        {
            rc_set_state(EXITING);
            return;
        }


}

/**
 * This function calculations inside the slow loop
 * It gets the encoder position (feed back values of x(t))
 * Then it applies the controller to find the theta reference for
 * the fast controller and to stabilize G1(s)
 *
 */
void* slow_loop(__attribute__ ((unused)) void* reference)
{
 while(rc_get_state()!=EXITING)
    {
            //set the encoder positions to zero initially
            rc_set_encoder_pos(2,0);
            rc_set_encoder_pos(3,0);
            //convert the encoder to radians
            L_rad = rc_get_encoder_pos(2)*c;
            R_rad = rc_get_encoder_pos(3)*c;
            //averges the left and right encoder readings
            //and make it absolute as the readings are
            //dependent on the MiP body angle
            phi = (L_rad+R_rad)/2 - theta_f;
```

```c
                phi = (L_rad+R_rad)/2 - theta_f;
                theta_ref = d2(phi);
        usleep(200000); //set frequency to 20Hz
    }
}


/**
 * This function is the fast loop controller
 * input: difference in theta_reading & theta_ref
 * output: duty cycle for the motors
 * D1(s) = -(s+8)/s
 * Using Tustin's Approximation
 * D1(z) = (-1.04z + 0.96)/(z-1)
 */
float d1(float theta_diff)
{
    float output = duty_old +k1*(-1.04*theta_diff + 0.96*theta_diff_old);
    duty_old = output;
    theta_diff_old = theta_diff;
    return output;
}

/**
 * This function is the slow loop controller
 * input: difference in phi and phi ref (which is zero)
 * output: theta reference for the fast loop controller
 *
 *  D2(s) = s/(s+13)(s+7)
 * Using Tustin's Approximation
 * D2(z) = (0.01606z^2-0.01606)/(z^2-1.212z+0.3577)
 */


float d2(float phi_diff)
{
    float output = 1.212*outputd2_old1 -0.3577*outputd2_old2 + k2*(0.01606*phi_diff - 0.01606*inputd2_old2);
    outputd2_old2 = outputd2_old1;
    outputd2_old1 = output;
    inputd2_old2 = inputd2_old1;
    inputd2_old1 = phi;
    return output;
}
```

Appendix B: MATLAB Code

```
% MAE 144 EduMIP Project Part 1
% Group 8 (Justin Chang, Khang, Basil)

%% Given Parameters
clear all; clc; close all;

mw      = 0.027;
mb      = 0.180;
wf      = 1760;
sbar    = 0.003;
G       = 320/9;
Im      = 3.6*10^-8;
R       = 0.034;
L       = 0.0477;
Ib      = 2.63*10^-4;
Vn      = 7.4;

DT1     = 0.01; %100Hz sample rate
DT2     = 0.05; %20Hz sample rate
gravity = 9.81;

% wheel inertia
Iw = 2*(mw*R^2/2+G^2*Im);

%% Plant & Controller TFs
a = Iw + (mw+mb)*R^2;
b = 2*(G^2)*sbar/wf;
c = mb *R*L;
d = 2*G*sbar;
e = mb*R*L;
f = Ib + mb*(L^2);
g = mb*gravity*L;

%% Inner Loop
numG1 = [d*(e+a) 0 0];
denG1 = [(e*c-a*f) (-e*b-b*c-a*b-b*f) (a*g) (b*g) 0];
G1 = tf(numG1,denG1);
G1 = minreal(G1);
```

```
%System's Root Locus
figure(1);
rlocus(G1);

%Controller design D1
k = -1;
D1 = tf([1 8],[1 0]);
D1 = D1*k;

% Discretization
 [numD1, denD1] = tfdata(D1,'v');
 [numD1z ,denD1z] = c2d(D1,DT1,'Tustin');
 D1z = tf(numD1z,denD1z,DT1);

%Root locus after controller
figure(2);
rlocus(G1*D1)

%Bode Plot G1*D1
figure(3);
margin(G1*D1)

%Step response
 figure(4);
 T = G1*D1/(1+G1*D1);
 step(T)
```

```
%% Outer Loop
numG2 = [(-c-f) 0 (g)];
denG2 = [(a+e) 0 0];
G2 = tf(numG2,denG2);
%G2 = minreal(G2)

% Ideal system without controller
figure(5)
rlocus(G2)
```

```
%D2 Controller
 K2       = 1;
extrapole= tf([1],[1 7]);
PD        = tf([1 0],[1 13]);
D2        = K2*extrapole*PD;


% Ideal system with outer loop controller
figure(6)
rlocus(G2*D2)

% Ideal Step response
figure(7)
step(G2*D2/(1+G2*D2));
ylim([0,1.3]);

%Realistic step response
figure(8)
step(G2*D2*G1*D1*(1/1.48)/((1+G1*D1)*(1+G2*D2)));
hold on;
step(T);
hold off;
legend('Outer Loop Step Response','Inner Loop Step Response');
xlim([0,4]);

% Open-Loop Bode Plot
figure(9)
margin(G2*D2);

%Open-Loop Bode Plot with Inner Loop Model Included
figure(10)
margin(G2*D2*T/1.48);

%Discretization
[numD2, denD2] = tfdata(D2,'v');
[numD2z, denD2z] = c2d(D2,DT2,'Tustin');
D2z = tf(numD2z,denD2z,DT2);
```