Come Together, Right Now

ENSC 384

Dan Hendry (30113878)
Brian Hook (200046894)
John Jamel (301080425)

Simon Fraser University
School of Engineering Science

August 12, 2010

Course Instructor: Professor Siamak Arzanpour

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

This report details controller design and system integration for the Mechatronics Design II project competition. It describes the creation and tuning of a 'switching' proportional, integral, derivative (PID) controller using Simulink. Four major areas are covered. First, basic support functionality required by the controller is discussed including the use of filters, software pulse width modulation (PWM) and voltage control (sections 2.1 through 2.4). Second, a discussion of the optimal controller and the process of parameter tuning is presented in addition to the limitations of simulation for this particular controller (sections 3.1 through 3.6). Next, other key portions of the Simulink diagram are discussed in addition to the methods used for measuring system performance (sections 4.1 through 4.2). Finally, the final system performance is presented along with a discussion of potential improvements (sections 5.1 through 5.3). The complete Simulink block diagram may be found in appendix A.

1.2 Competition and General System Configuration

The competition involves picking up a metal puck using a mechanical arm and electromagnet, moving it 90°, then dropping it in a hole. The mechanical arm starts at 0°, the puck at 45° relative to the arm and the hole is positioned at −45°. Position graphs throughout this

report show the arm moving from 0°, to 45°, then to −45° and finally returning to 0°. The final step, although not required by the competition, was added for convenience. The y-axis for position graphs shows position in radians, not degrees.

A Sennsoray Model 626 data acquisition card was used to read encoder positions, read voltages using analog-to-digital converters, and control the circuit using digital-to-analog converters. The Simulink controller time-step was set to $200\mu s$, the maximum sampling and control rate supported by the hardware. The motor was controlled using an H-bridge motor controller circuit and powered with 9.5 Volts by a power-supply with a rated current capacity of 2.5 Amps.

Chapter 2

Support Components

2.1 Filters

The motor encoder and PWM amplifier are both discreet, digital devices. Given the small controller time-step, speed values (derivative of encoder position) were seen to be very unstable between samples and contain inaccurate high frequency components. Additionally, an ideal PWM amplifier with resistive load is characterized by producing either zero volts or its supply signal for any instance sample. Due mainly to these two effects, there are certain signals within the controller which require filtering and averaging via a low pass filter to be used effectively. Low pass filters are, in and of themselves, dynamic systems containing poles. By adding them to the controller, the total number of system poles increases complicating analytical analysis. The impact of these new poles may be minimized by ensuring they are positioned as far away from the system poles as possible.

When designing the controller heuristically, it was important to minimize the phase shift introduced by low pass filters. This was done by using low order filters with the highest pass band possible. Butterworth filters of order three and with a corner frequency of 80-120 radians per second were found to be the most effective. See figure 2.1 for an example of the the system response when the passband of a filter is too low or its order is too high causing excessive phase shift The characteristic which most strongly indicates a filter is influencing the system in an undesirable way is the presence of oscillations with a center that is not the reference signal.

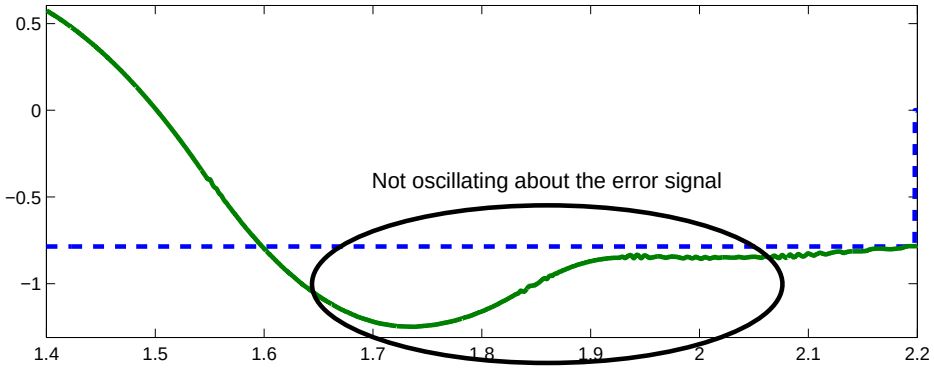

Figure 2.1: Symptom of Excessive Phase Shift

2.2 Signals, Conditioning and Output Constants

Throughout the controller, various input signals require conditioning and certain output values must be chosen. The MOSFET transistors used in the H-bridge motor controller and magnet control circuit are driven directly by the DAC. A 0-9 Volt range was used in an effort to ensure crisp switching. As described in section 2.4, the actual motor terminal voltage is measured and used in a feedback loop for error correction. A differential ADC was connected across the terminals and passed through a low pass filter to account for digital nature of PWM as described in section 2.1.

The encoder output also required simple processing to determine motor speed and position (in radians), both before and after the gearbox, and the back EMF generated. The Simulink block diagram responsible for this processing may be seen in Figure 2.2. Encoder gain is given by equation 2.1 and K_b is provided by the motor datasheet. After various unit conversions, it was was found to be $K_b = 0.02424 \frac{V}{rad/s}$.

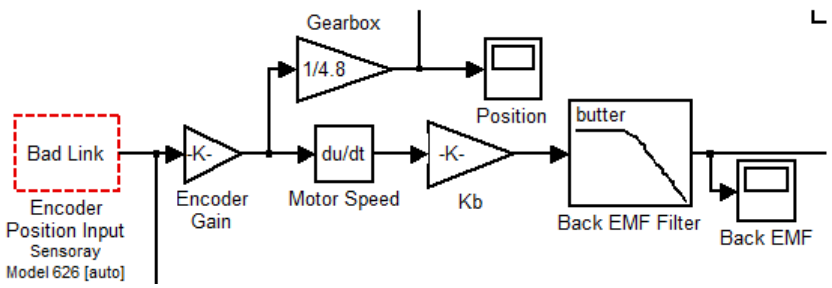

Figure 2.2: Encoder Conditioning

$$K_{encoder} = \frac{2\,\pi\,\frac{rads}{revolution}}{500\,\frac{slots}{disk} \times 2\,\frac{pulses}{slot} \times 2\,disks} \tag{2.1}$$

2.3 Software PWM

The motor used to position the arm in this project is driven by a MOSFET H-bridge. For greatest efficiency, MOSFETS should always be switched to their on or off state. The on-off switching performed by the MOSFETs, can used to generate a power signal with a controllable average value using pulse width modulation (PWM). The PWM for this project was designed to operate in the range of 9.5V to -9.5V and was implemented in Matlabs Simulink environment after it was determined that accurate control couldnt be achieved using the external op-amp oscillator, peak detector and additional circuitry. A Simulink block diagram showing the PWM implementation may be seen in Figure 2.3.

The key component in this diagram is the 'PWM Source', a repeating ramp from zero to one which resets every 10 ms. As such, the nominal PWM frequency is 100 Hz. Given the controller sampling rate of $200\mu s$ and output range of 0-9.5 Volts for each side of the PWM generator, a control resolution of approximately 190 mV is achieved. When compared to the very fine resolution and approximate 300 Hz frequency of the op-amp based oscillator and peak detector, the software PWM appears to be a poor choice. It is however vastly more deterministic and far less prone to external distrubances. An important consideration, due to the bound on sampling rate, when choosing the PWM frequency is the tradeoff between resolution and frequency. By increasing the switching frequency, the PWM resolution will be directly reduced.

Ideally, the PWM should have a switch frequency much higher than the dynamics of the system, motor and controller, such that the motor appears to be powered by a true DC voltage source. The PWM frequency should be much greater than the natural frequency of the system to prevent interference and phase delay. Equivalently stated, the input signal to the PWM generator should be stable within the period of each pulse. This desire for separation between the PWM frequency and system natural frequency is antithetical to the desire for increased natural frequency to improve system response and reduce settling time.

Conversely, there is also a desire for greater PWM resolution such that the motor can be positioned accurately. Increasing resolution also helps to prevent undesirable limit-cycling; a source of inaccuracy and mechanical vibration.

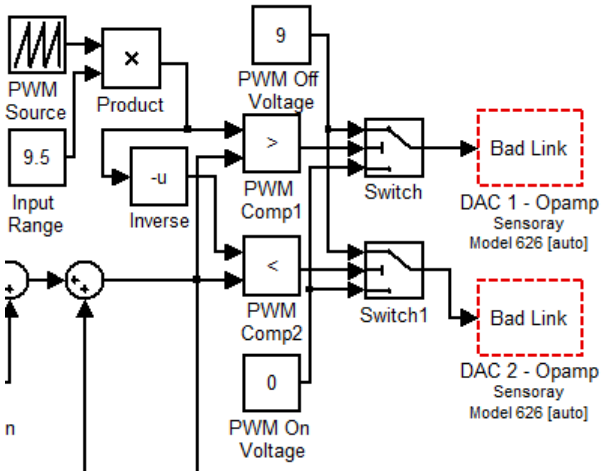

Figure 2.3: Software PWM Simulink Schematic

2.4 Voltage Control

There are two common approaches for controlling a simple DC motor, current control and voltage control [1]. Since motor torque is directly related to current, the output of a current controller corresponds to the desired motor torque [2]. Similarly, since motor speed is directly related to its armature voltage, the output of a voltage controller corresponds to the desired motor speed [3]. The PID controller used for this project is designed to be a voltage controller. As such, its output is the desired voltage across the armature or alternatively the desired motor speed.

It is important to note that the PWM amplifier used in this project (either software PWM or the original hardware PWM) is NOT in inherently a voltage amplifier but an open loop power amplifier. Its input signal controls the on-off power-supply duty cycle seen by the load each period. For a purely resistive load, a PWM amplifier may be used as a voltage amplifier. The motor however, produces back EMF that is directly proportional to the speed of the armature. In other words, the load characteristics change based on motor speed. As such, there is no direct correlation between the PWM amplifier input voltage and its output

voltage when connected to a motor. Without compensating for back EMF, the lookup table technique used by most other groups is valid only for the motor speed and load for which it was designed. The general approach is naive, highly inaccurate, and unsuitable for a position control system.

The Simulink block diagram used to condition the controller output signal and compensate for limitations of the PWM amplification technique may be seen in Figure 2.4. In addition to back EMF compensation, simple error compensation is also used. The actual armature voltage is measured, compared to the desired voltage and the resulting error signal added. A final form of compensation, simple feed forward, is also introduced. It was added to correct for a consistent offset observed in the measured motor voltage signal. A demonstration of voltage control performance may be seen in Figure 2.5. The measured voltage tracks the input voltage very well, showing the excellent degree of control which can be obtained using a software based solution. It should be noted that over the course of the test, there was significant load variation, manually applied to the motor shaft, with very little impact on its speed.

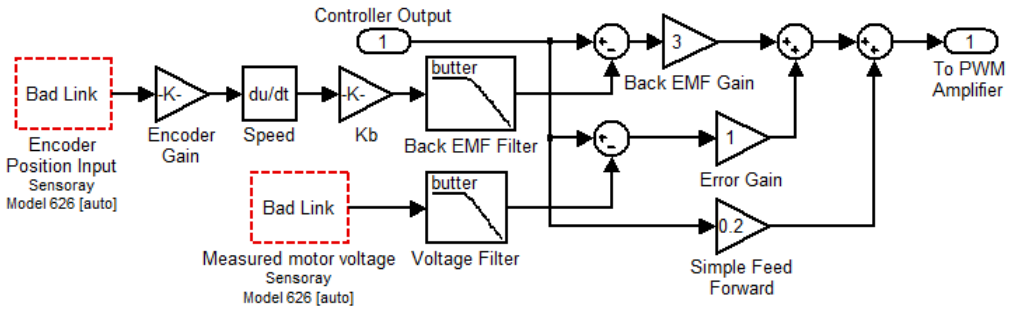

Figure 2.4: Voltage Control

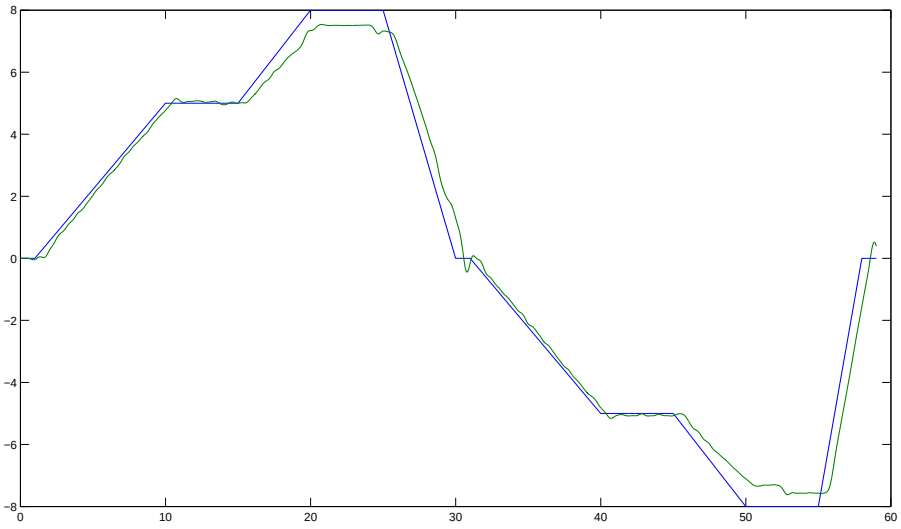

Figure 2.5: Voltage Control With Back EMF Compensation

Chapter 3

Controller Design

3.1 Optimal Controller

Consider a theoretically optimal controller for positioning the arm. It would drive the motor with as much power as possible to accelerate the arm when a new position is desired. At a critical point, it would reverse the polarity of the motors power-supply and apply as much power as possible to decelerate it. This deceleration would continue and end only once the arms velocity reaches zero. The critical point should be such that when the arms velocity reaches zero and the power supply switches off (no further acceleration), the arm is positioned exactly where it should be. As described, the motor controllers output would always be saturated (or zero), it would switch polarity only once each time a new position is requested, and there would be no overshoot of the arms position. Practically, implementing this behavior exactly is nearly impossible as it allows for no error and variability.

Based on the above mentioned criteria for optimality, the controller created for this project comes amazingly close realizing perfect performance. The output of the controller after voltage conditioning, captured during an experiment and shown saturated at the limits where the H-bridge always drives the motor one direction, can be seen in Figure 3.1. Over the entire course of the experiment, with the exception of returning to zero, only the areas indicated by the A and B labels show non-optimal controller behavior. These pick-up and drop-off locations, are very narrow compared to the rest of the signal, which implies that the system control is close to optimal.

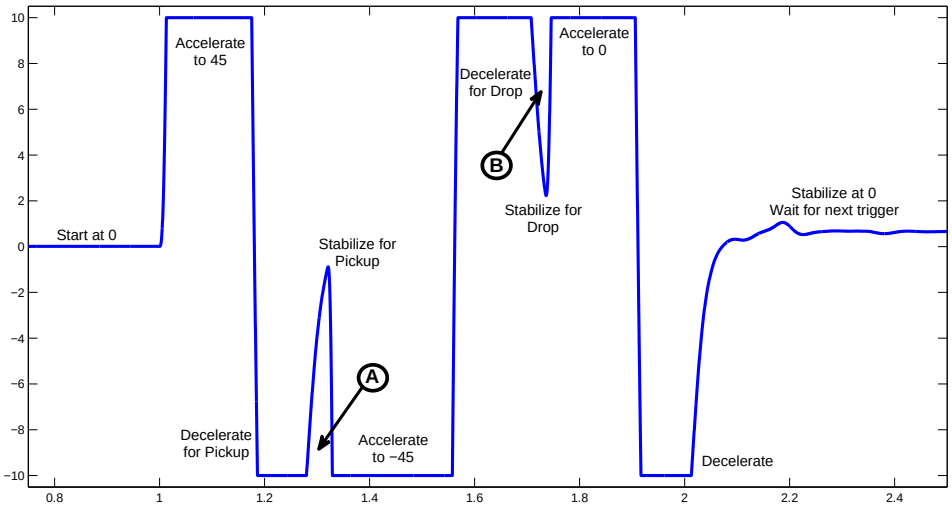

Figure 3.1: Controller Output (Voltage)

Two approaches may be used to achieve the optimal controller within the limits of a PID. First, a simple PID controller can be used with very large gains such that the controller output saturates the PWM input. The disadvantage of this technique is that the standard analytical techniques used to design a PID are no longer valid. Additionally, very large gains increase the natural frequency of the system, increasing the risk of undesirable interaction with the relatively low frequency software PWM (see section 2.3). The second approach, used in this project is to switch controllers based on the error signal. The general logic is as follows.

1. **P** - When the error signal is large, or the truss is positioned far from where it should be, use a purely proportional controller. This decreases response time and saturates the motor controller immediately.

2. **PD** - When the error signal reaches a critical threshold, switch to a slightly over-damped PD controller to slow the system down and perform final positioning. This threshold should be less than the optimal threshold discussed above such that error feedback may be used to correct for inaccuracy and system variation.

3. **PID** - Finally, when the error signal is relatively small, add integral gain to correct for

any steady state error.

3.2 Controller Switching Thresholds and Overall Tuning

In and of itself, the switching PID controller described in section 3.1 has five parameters; three controller gains plus the two switching thresholds. Although all could likely be determined analytically or via simulation, the non-standard nature of the controller and reliance on external physical effects (see section 3.6) makes the process non-trivial. As such, finding parameters and tuning was accomplished by applying a systematic experimental method.

Of critical importance is the threshold where the proportional controller is replaced by a PD controller. Figure 3.2 below shows two plots of position vs. time as the truss moves from the pick-up position (0.79 rad = +45°) to the drop position (-0.79 rads = -45°) and back to initial position (0 rads). The blue plot represents a correctly calibrated controller that begins changing the truss direction at the drop position. As the truss decelerates, the red plot shows a measure of overshoot at 1.76s. This overshoot can be attributed to the switch from a P to PD controller occurring too 'late'. The condition used to determine when switching should occur is the error magnitude. The overshoot can be corrected by increasing error magnitude at which a switch will happen. It should be noted that different thresholds are used when the arm moves 45° and when it moves 90°.

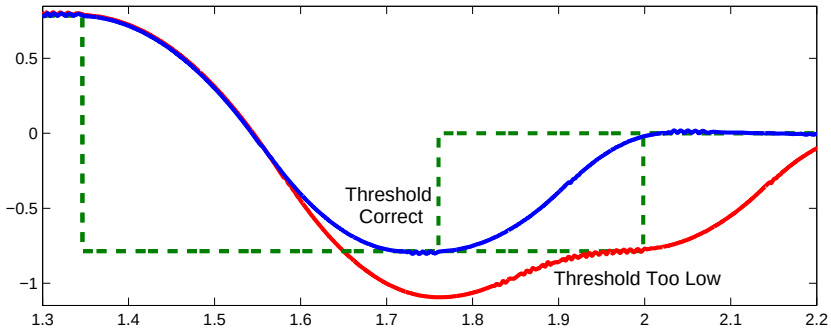

Figure 3.2: Threshold Selection

3.3 Determining Proportional Gain

Due to the nature of the saturating, switching PID as described in section 3.1, the exact proportional, derivative and integral gains are relatively unimportant compared to the switching thresholds. A PID controller is used, essentially, to compensate for error in the switching threshold.

Two system characteristics act to bound the range of values proportional gain is able to take. First, it must be large enough saturate the PWM amplifier in order to realize a near optimal controller. Second, it cannot be so large that the discreet nature of encoder position measurements becomes apparent in the control signal causing instability or excessive vibration. Within this range, it is also limited by a need to keep the natural frequency of the system below that of the software PWM amplifier. Finally, it needs to be set large enough to overcome the limited resolution of the software PWM such that fast, accurate positioning is possible. It was the last condition that was the primary driver, after extensive experimentation, for setting proportional gain at 25.

3.4 Determining Derivative Gain

Derivative gain is used to decelerate the arm and is activated when the threshold, described in section 3.2, is met. In order for immediate deceleration to occur, derivative gain is bounded to meet the requirement the PD controller be over-damped. As with proportional gain, once this requirement is met the exact value for derivative gain is relatively unimportant and should only every have a significant impact on the system if the switching threshold (section 3.2) is incorrect. For robustness, an incorrect switching threshold was assumed when tuning derivative gain. Figure 3.3 shows two system responses where derivative gain is incorrect and increases the time needed to drop the puck. Once again, the overshoot is not due to the gains of the PD controller but due to an incorrect controller switching threshold.

The Simulink block diagram used to implement derivative control may be seen in Figure 3.4. Of particular interest is the 'desensitize' block. This dead-zone is used to reject small, unstable derivative signals caused by the discreet nature of the encoder. Without this block,

the system was seen to vibrate excessively at its steady state. A dead-zone range of ±0.07 was found to effectively eliminate most vibrations.

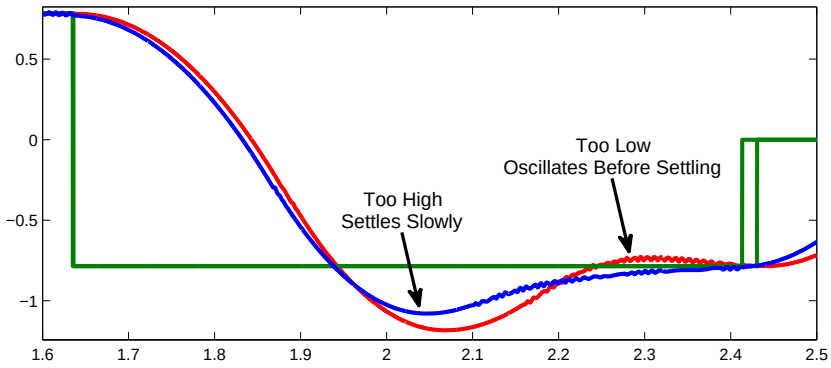

Figure 3.3: Response For Various Derivative Gains - Incorrect Switching Threshold

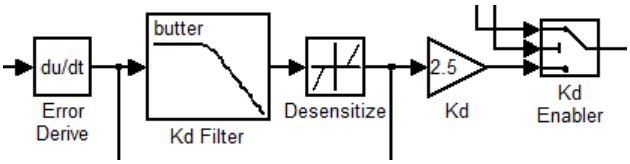

Figure 3.4: Simulink Derivative Gain Path

3.5 Determining Integral Gain

The primary reason to add integral gain to any PID controller is to reduce steady state error. With the exception of this critical benefit, most other effects of integral gain are undesired, such as increasing settling time and adding an additional pole to the system. Given the lack of any systematic disturbance which could cause steady state error, integral gain was only added to the controller as a precaution. Additionally, it is only enabled when the magnitude of the error signal is small (under 0.1 radians) to correct for poor PWM resolution. Figure 3.5 shows a typical case where the enabling threshold and gain for integration feedback is too high.

The block diagram implementing integral gain may be found in Figure 3.6. To prevent actions taken by the PD controller influencing the PID controller, the integrator is reset each time it is enabled.

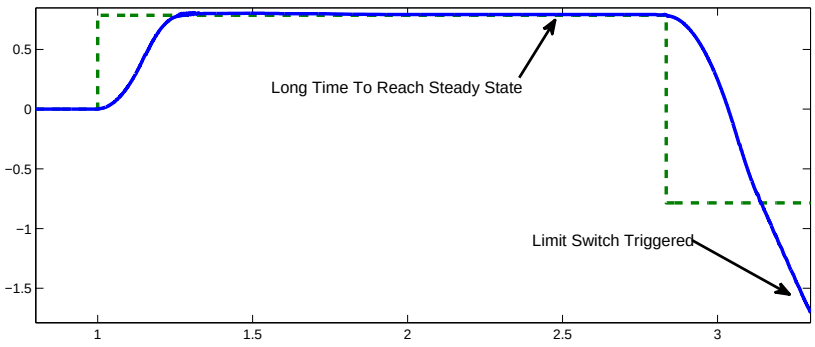

Figure 3.5: Too Much Integral Gain

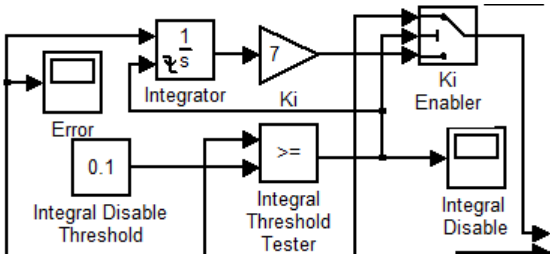

Figure 3.6: Simulink Integral Gain Path

3.6 Limitations of Simulation

The controller presented in this report was designed almost exclusively by experimental and heuristic methods. Virtually no result produced via analytical means or simulation was incorporated into the controller as they were found to be generally inaccurate. Although it is likely controller gains, switching thresholds, filter pass-bands, stability conditions and other tunable parameters could be found analytically and/or modeled accurately, such a process was found to be impractical and unnecessary, especially given the project time constraints. The following list details various reasons analytical analysis and simulation was impractical; many follow from the fact that the controller operates in the near optimal, nonlinear region of the hardware.

- Motor acceleration was seen to be limited by current saturation within the power-supply, not the motor. The saturating characteristics of the power-supply are complex, difficult to measure and model, and have a large impact on system performance.

- Software PWM is highly dependent on sampling rate and hardware speed characteris-

tics. Mixing sampling rates within a simulation introduces secondary complexities.

- Position is a discreet variable due to the encoder being used.
- Mechanical vibration was seen to be a significant issue and dictated certain controller design features. These vibrations are difficult to simulate.
- External forces with non-linear characteristics, such as torsion in the magnet wire are unknown.

Chapter 4

Other Simulink and System Components

4.1 Stability Detector

A steady, stable system state is an obvious prerequisite for activating the electromagnet to pickup the puck and deactivating it to release the puck for reliable system performance. A Simulink block diagram showing the stability detector may be seen in Figure 4.1. Both speed and position are checked against experimentally established thresholds. As previously discussed, the speed signal must be passed through a low pass filter. The pre-filtered signal is also checked in an effort to eliminate phase shift error caused by the filter.

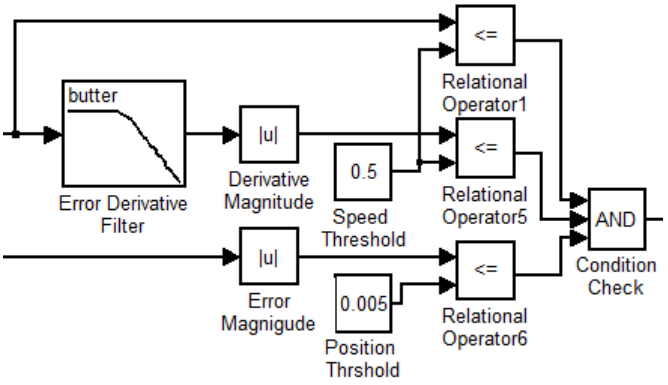

Figure 4.1: Stability Detection

4.2 Timing

Two methods were used to measure system performance or drop time. First, the time between sending a manual trigger and the release of the puck by the electromagnet over the hole. Second, the time from which the arm starts moving until the limit switch is hit by the metal puck. Given correct truss positioning, where the puck was able to fall directly to the limit switch, the second method was seen to increase the measured performance time by approximately 50 msec. When an explicit offset was introduced, causing the puck to hit the edge of the hole, the discrepancy between the two measurements increased to 100-250 ms.

Chapter 5

Conclusion

5.1 Final Performance

A graph showing overall system positioning performance may be seen in Figure 5.1. The total time between triggering the system and releasing the puck over the hole was seen to be 0.7432 seconds (the first method from section 4.2). The total time between the arm starting to move and the puck hitting the limit switch was seen to be 0.7948 seconds (the second method from section 4.2).

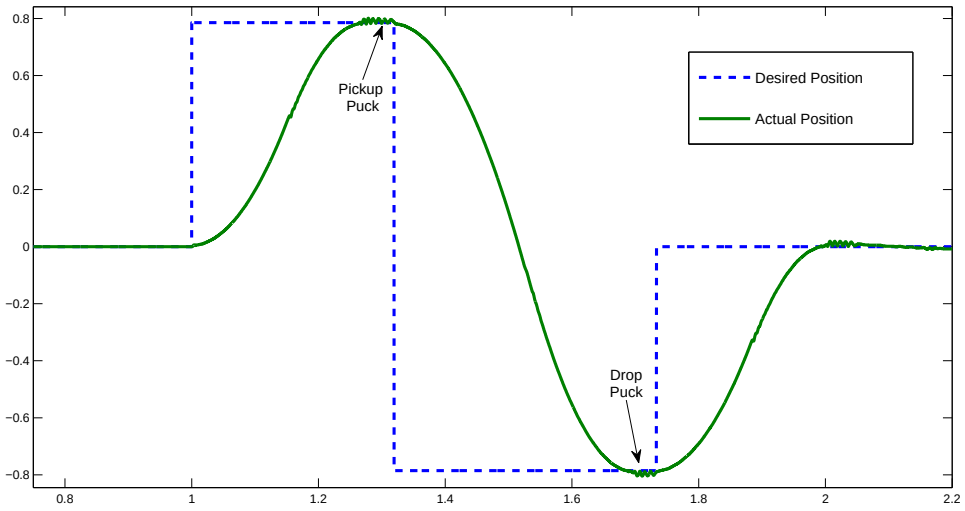

Figure 5.1: System Performance

5.2 Limitations and Improvements

Despite the controller itself having near optimal performance there are many ways to improve overall performance and increase its robustness and reliability. These changes include:

- Use a power supply able to output more current such that it is the motor, not power supply which saturates.
- The limited controller sampling rate was not fast enough for robust and effective software PWM. A dedicated microcontroller or digital PWM generation circuit should be used.
- 'Slip' was observed, potentially caused by limited encoder or software PWM resolution. This could be corrected by using a more accurate encoder or positioning mechanism, such as laser sensors.
- The controller switching threshold is important and sensitive, having a large impact on performance. In this project, thresholds were found manually and selected based on one of two movement modes (45° and 90°). This was sufficient; however, a more robust design would compute the threshold based on speed and position.
- Adding a physical damping could minimize physical vibrations.

5.3 Conclusion

This report has discussed the basic features of the controller design such as filters, signal conditioning, software PWM, and voltage control. Furthermore, the report explained the approach taken to design the controller to maximize performance. Limitations to simulating this specific controller were discussed in addition to general system and controller limitations including proposed solutions to improve the response and increase the execution time. Finally the final system performance was presented, an overall time of 0.7432 seconds based on the common method used for measuring.

Bibliography

[1] G. R. Slemon and A. Straughn, *Electric Machines*. Addison-Wesley, 1982.

[2] C. W. de Silva, *MECHATRONICS An Integrated Approach*. CRC Press LLC, 2005, p. 789.

[3] N. Phuc, "Speed and Torque Control," July 9, 2009,, http://cnx.org/content/m28695/1.1/.

Appendix A

Full Simulink Block Diagram

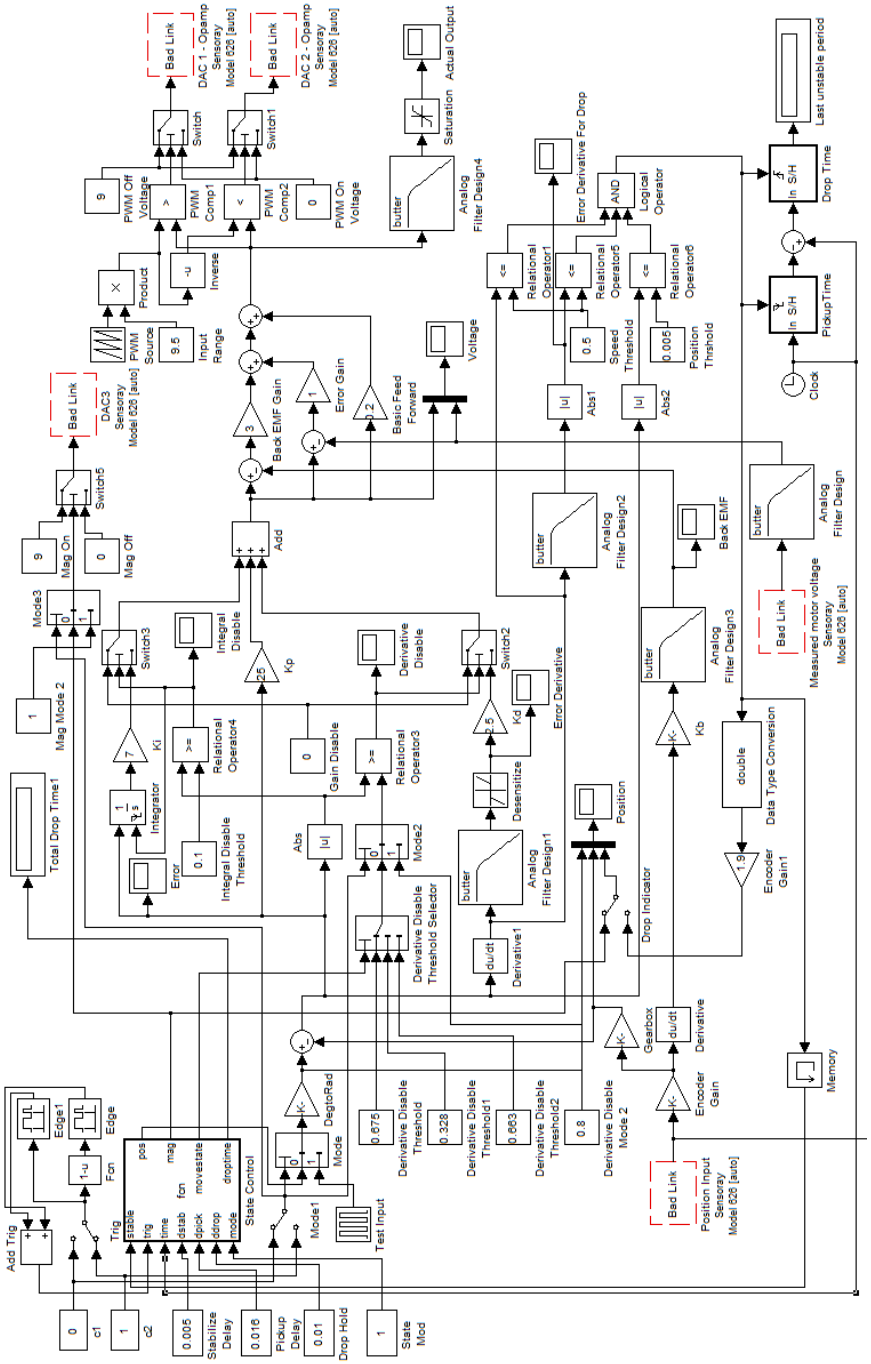

Figure A.1: Complete Simulink Block Diagram

Appendix B

State Controller Code

```
function [pos,mag,movestate,droptime] = fcn(stable, trig, time,dstab,dpick,ddrop,mode)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.

persistent state;
persistent delayuntill;
persistent skipnext;

persistent startdrop;
persistent lastdroptime;

persistent lastpos;
persistent lastmag;
persistent lastmovestate;

if isempty(state)
   state = 0;
   delayuntill = 0;
   lastpos = 0;
   lastmag = 0;
   lastmovestate = 0;
   skipnext = false;
   startdrop = 0;
   lastdroptime = 0;
end

pos = lastpos;
mag = lastmag;
movestate = lastmovestate;
droptime = lastdroptime;

if (time < delayuntill) || skipnext
    skipnext = false;
    %statedbg = state;
```

```
    return
end

laststate = state;

% Standard
if mode == 0
switch state
    case 0
        % Wait for user to trigger start
        pos = 0;
        mag = 0;
        movestate = 0;
        if trig
            state = state + 1;
            pos = 90;
            %skipnext = true;
        end
    case 1
        % Move to 90 then wait
        if trig && stable
            state = state + 2;
            pos = 45;
            movestate = 1;
            %skipnext = true;
        end
    case 3
        % Move to 45 then wait for manual trigger
        pos = 45;
        movestate = 1;
        if stable && trig
            state = state + 1 ;
            pos = 90;
            %skipnext = true;
            startdrop = time;
        end
    case 4
        % Move to 90 then wait
        if stable
            state = state + 1 ;
            delayuntill = time + dstab;
        end
    case 5
        % Pickup mag then wait
        if stable
            mag = 1;
            movestate = 0;
            state = state + 1 ;
            delayuntill = time + dpick;
```

```
        end
    case 6
        % Start move to 0
        pos = 0;
        state = state + 1 ;
        %skipnext = true;
    case 7
        % Move to 0, drop and wait
        if stable
            state = state + 1 ;
            mag = 0;
            delayuntill = time + ddrop;
            lastdroptime = time - startdrop;
        end
    case 8
        % Repeat
        pos = 45;
        mag = 0;
        movestate = 1;
        state = 3;
        %skipnext = true;
    otherwise
        state = 0;
end

% Competition - 45 -> 90 -> 0
elseif mode == 1
switch state
    case 0
        % Start up
        pos = 0;
        mag = 0;
        movestate = 1;
        state = state + 1;
        delayuntill = time + 1;
    case 1
        % Go to pickup
        pos = 45;
        state = state + 1;
        %skipnext = true;
    case 2
        % Wait for additional stable time
        if stable
            state = state + 1 ;
            delayuntill = time + dstab;
        end
    case 3
        %Pickup
        if stable
```

```
            state = state + 1 ;
            mag = 1;
            movestate = 0;
            %skipnext = true;
            delayuntill = time + dpick;
        end
    case 4
        % Move to drop
        movestate = 0;
        pos = -45;
        state = state + 1;
        %skipnext = true;
    case 5
        % drop
        if stable
            mag = 0;
            state = state + 1;
            delayuntill = time + ddrop;
            lastdroptime = time - 1;
        end
    case 6
        mag = 0;
        pos = 0;
        movestate = 1;
    otherwise
        state = 0;
end

% Competition - 90 -> 0
elseif mode == 2
switch state
    case 0
        % Start up - wait for 1 second
        pos = 0;
        mag = 0;
        movestate = 2;
        state = state + 1;
        delayuntill = time + 1;
    case 1
        % Start up - get mag
        mag = 1;
        state = state + 1;
        delayuntill = time + dpick;
    case 2
        % Go to drop
        pos = -90;
        state = state + 1;
    case 3
        % Wait for additional stable time
```

```
        if stable
            state = state + 1 ;
            mag = 0;
            delayuntill = time + ddrop;
            lastdroptime = time - 1;
        end
    case 4
        %End
        pos = 0;
        mag = 0;
        movestate = 0;
    otherwise
        state = 0;
end
end

lastpos = pos;
lastmag = mag;
lastmovestate = movestate;
%statedbg = state;

if laststate ~= state
    skipnext = true;
end

droptime = lastdroptime;
```