**MEMORANDUM** 16 Feb 2011

From: MIDN 3/C McVay and 4/C Tomaszewski

To: Prof. G. E. Piper

Subj: Lab Report

Encl: (1) Documented Matlab scripts and “C” programs

(2) Pseudo Code for Algorithmic Control

**1. OBJECTIVES:**

i) To practice PWM commands on the ES308 board.

ii) To use A/D conversion concepts in a project

iii) To become familiar with sensor feedback and sensor calibration

iv) To explore the differences between algorithmic control and classical control

v) To become familiar with proportional and integral feedback control

.

**2. NARRATIVE AND OBSERVATIONS/RESULTS**: The first part of the lab was calibrating the sensor. Here we determined the relationship between the Vad readings and the actual distance that the readings relate to. This was done by measuring the height of the elevator and taking a reading at that point, repeating at every ½ inch. We did this from ½ inch away from the sensor through 9.5 inches away, and graphed a plot of the A/D converter reading against the distance from the sensor. We found the linear portion of the graph to range from about 4 inches to about 7 inches from the sensor. Because this seemed like a short range, we decided to set our floors at the outer-most range of the linear portion, being 4 and 7 inches from the sensor. These distances corresponded to A/D voltages of 2475 and 2332 counts, respectively. The corresponding equation to obtain a distance from the A/D reading was:

Distance = -0.020979\*VAD + 55.923

The next part of the lab was the power calibration. In this step, we placed one PMW motor in the elevator to act as a weight, and then we adjusted the power settings on the elevator to properly raise, lower, and hold position. The original values for PMW motor speed used to properly operate the elevator with 1 motor as a weight were 30% going up, -20% going down, and 5% holding. We used a “for loop” with a 10ms delay to command the elevator to hover while continuously transmitting position through the serial port to MATLAB.

The next step was to move the elevator from floor to floor. Here we added a “while (1)” loop so that the elevator would continuously loop from floor to floor. Then we plotted the elevator’s position vs. time showing three cycles. The figures shown on the right have been inverted to show the top floor toward the top of the graph, and the bottom floor at the bottom of the graph.

Figure : Original commands

The next step was to test the elevator system when disturbances are added. In this case the disturbances were weight. We added 2 AA batteries before it would no longer hover, and a second motor slowed ascent significantly. Then we graphed this data. Next, we adjusted the power so that the motor would effectively move and hover with the added weight. We re-adjusted our PMW motor speed values to 45% going up, 18% holding, and -5% going down. We once again graphed the data for three cycles. The last part of this step was to take off the weight and keep the new speeds, then graph once more.

Figure 2: Original commands Plus Weight

Figure 3: New commands plus weight

As the graphs depict, adding weight prevented the elevator from hovering, decreased ascension rate, and increased rate of descent. Increasing the power and removing the extra weight had the opposite effect. Instead of sagging while trying to hover, the elevator would rise; instead of decreasing ascension rate, it increased it; and instead of causing the elevator to descend faster, it caused the elevator to descend more slowly.

Figure 5: Original k value

Figure 4: New commands without weight

In step five we used proportional control. The general idea around this was the closer the elevator got to the desired floor, the slower it went. This has real life application because you do not want your elevator jerking to a stop. Our original value of proportional control constant (k) used was 55.

Figure 6: Original k value with weight

The next step we repeated the disturbance experiment. This time we added 6 AA’s plus a second motor, and that prevented the elevator from hovering. However, we could not add a significant amount of weight to keep it from ascending. The new proportional control value that we used to adjust for a second motor was 85.

Figure 7: New k value with weight

Adding weight with proportional control did not disturb the system as much as it had using classical control; however, it did still have a slowing effect on the elevator in the upward motion in addition to increasing rate of descent. Upon increasing the value of k to account for the added weight and removing it, the elevator still worked fine, it just moved more quickly.

Figure 8: New k Value without Weight

The last step we used integral power control. We used a proportional control constant (k) value of 65 and an integral control constant (Ki) value of 20 with no weight. Next, we added a second motor, graphed, adjusted k to 85 and Ki to 30, graphed, removed the second motor, and graphed once more.

Figure 9: Original values of k and Ki



3. **CONCLUSIONS:** While without weight the elevator worked great with every type of control the addition of weight severely hindered the performance of the first type. This is important because in real world applications the elevator will usually have weight on in. However the Ki values under “better control” seemed to increase performance at the top floor, but decreased performance at the bottom floor, taking more time to get to the bottom and often going below it before re-adjusting and coming back up. Using proportional control increased performance across the board, even with added weight, over the original motorspeed commands. Overall the proportional controls were proven most effective with and without weight.

Figure 11: New values of k and Ki plus weight

Figure 10: Original values of k and Ki plus weight

**4. COMMENTS:** This lab steps were broken down into manageable steps that allowed us to easily follow the objectives.

Figure 12: New values of k and Ki without weight

M. McVay, L. Tomaszewski

MIDN USN

Encl(1) Commented Code and MATLAB scripts

Final C Code

**NOTE: The code that has been “commented out” between “while (1)” and “if(T<4)” is the**

**original code for classical or “algorithmic” control. The code for proportional control is**

**embedded in the “integral control” code.**

#use "ES308\_SBC.LIB"

void tx(int);

const int Vad\_floors[2]={2332,2475};

void MS\_wait(unsigned long N)

{

while(MS\_TIMER < N);

return;

}

void main(void)

{

float mospeed, Vin\_est, desired\_voltage, error, k, T, integral\_value;

char s[32]; // character string read from keyboard

int value, Vad, i, Vad\_desired, Ki;

unsigned long T\_ms; //declaring all variables

mospeed=0.0;

ES308\_Init(); // Inits all ES308 Rabbit I/O to default values

MsDelay(10);

PWM\_Init(1, 2); // port 1 mode 2

// Set up PWM channel port and mode to drive a DC motor

MsDelay(10);

serBopen(9600); // Baudrate

serBflowcontrolOff();

serBdatabits(8);

serBparity(PARAM\_NOPARITY);

T=0;

T\_ms=MS\_TIMER;

k=85;

Ki=30;

integral\_value=0;

while(1) //loop continuously

{

//Vad=Norm\_Get\_AD\_Val(0); //get Vad and send it to matlab

//tx(Vad);

//while(Vad<2475) //loop to lift elevator to

//{ //top floor

//mospeed=45; //values used are for added weight

//PWM\_MotorSpeed(1,fabs(mospeed));

//MsDelay(10);

//ServoControl(1, 'F');

//MsDelay(10);

//Vad=Norm\_Get\_AD\_Val(0);

//tx(Vad);

//}

//for(i=0;i<100;i++) //loop to hold elevator at top

//{ //for one second

//mospeed=18;

//PWM\_MotorSpeed(1,fabs(mospeed));

//MsDelay(10);

//ServoControl(1, 'F');

//Vad=Norm\_Get\_AD\_Val(0);

//tx(Vad);

//}

//while(Vad>2332)

//{

//mospeed=-5; //loop to descend elevator

//PWM\_MotorSpeed(1,fabs(mospeed)); //to bottom floor

//MsDelay(10);

//ServoControl(1, 'R');

//MsDelay(10);

//Vad=Norm\_Get\_AD\_Val(0);

//tx(Vad);

//}

//for(i=0;i<100;i++) //loop to hold elevator at bottom

//{ //floor

//mospeed=20;

//PWM\_MotorSpeed(1,fabs(mospeed));

//MsDelay(10);

//ServoControl(1, 'F');

//Vad=Norm\_Get\_AD\_Val(0);

//tx(Vad);

//}

if(T<4) //for first 4 seconds, work toward

{ //bottom floor

Vad\_desired=2332;

printf("Vad\_desired = %d\t",Vad\_desired);

}

else if(T<8) //for last 4 seconds, work toward top

{ //floor

Vad\_desired=2475;

printf("Vad\_desired = %d\t",Vad\_desired);

}

else

{

T=0;

}

Vad=Norm\_Get\_AD\_Val(0); //read sensor

tx(Vad); //send data to matlab

Vin\_est=(-10)+Vad\*(20/(pow(2,12)-1));

desired\_voltage=(-10)+Vad\_desired\*(20/(pow(2,12)-1));

error=desired\_voltage-Vin\_est; //calculate difference between

integral\_value += error\*0.03; //sensor voltage and desired

printf("error = %f\t",error); //voltage

mospeed=k\*error+18+Ki\*integral\_value; //adjust motor speed accordingly

PWM\_MotorSpeed(1,fabs(mospeed));

if(mospeed<=0) //set proper direction of motor

{

ServoControl(1, 'F');

}

else

{

ServoControl(1, 'R');

}

T\_ms=T\_ms+10; //continue timer

MS\_wait(T\_ms); //and reset value of time so that the counter

T=T+0.03; //increases in real time

printf("T = %f \r",T);

}

serBclose(); //close the port

}

void tx(int num)

{

int k;

k = num & 0xFF; // turns num into 8 bit number k

serBputc(k); //send k over port

Matlab code

Matlab Code:

t=0;

tstart=tic; %start timer

e=plot(0,0,'r-'); %start plot

hold on %plot over previous values

while(s.BytesAvailable > 0)

fread(s,1); %Clear Buffer

end

while(1)

while(s.BytesAvailable<2)

end

dat=fread(s,1,'int16'); %get data from C

t=toc(tstart); %keep track of time

dat=(-.020979)\*dat+55.923; %convert data to

plot(t,dat,'o'); %distance from sensor

pause(0.01); %and plot

while(s.BytesAvailable > 0)

fread(s,1); %clear buffer

end

end

Enclosure (2): Pseudo Code for Algorithmic Control

Loop Continuously

Check elevator position

Send position to MATLAB

While (elevator is below top floor)

Move upward

Check elevator position

Transmit elevator position to MATLAB

End

For one hundred repetitions:

Hold position

Check position

Transmit position to MATLAB

Delay 10 ms

End

While (elevator is above bottom floor)

Lower elevator

Check position

Send position to MATLAB

End

For one hundred repetitions:

Hold position

Check position

Transmit position to MATLAB

Delay 10 ms

End