

MICROCONTROLLER SYSTEM CONTROL DESIGN

JON PRITCHARD | SCOTT VON THUN

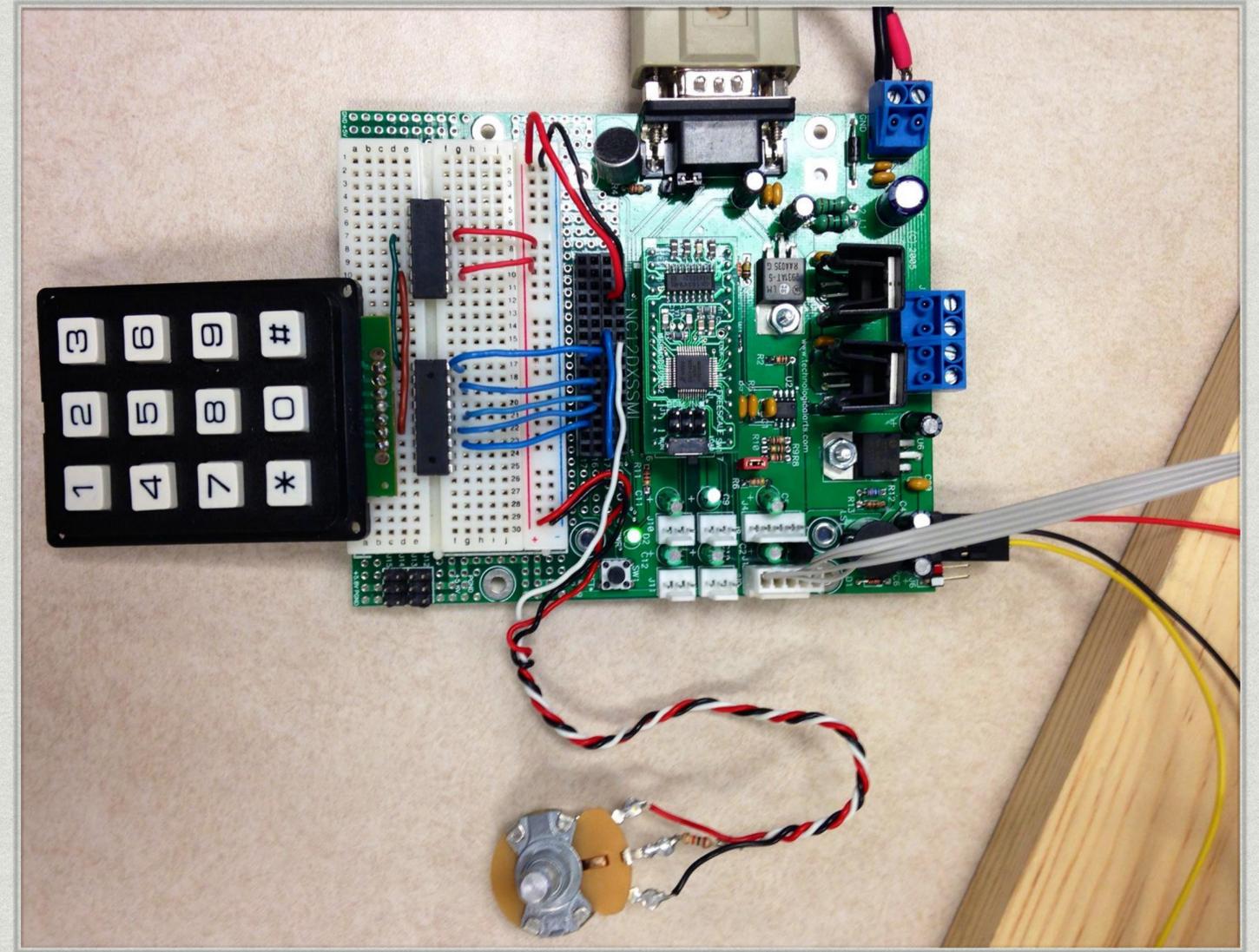
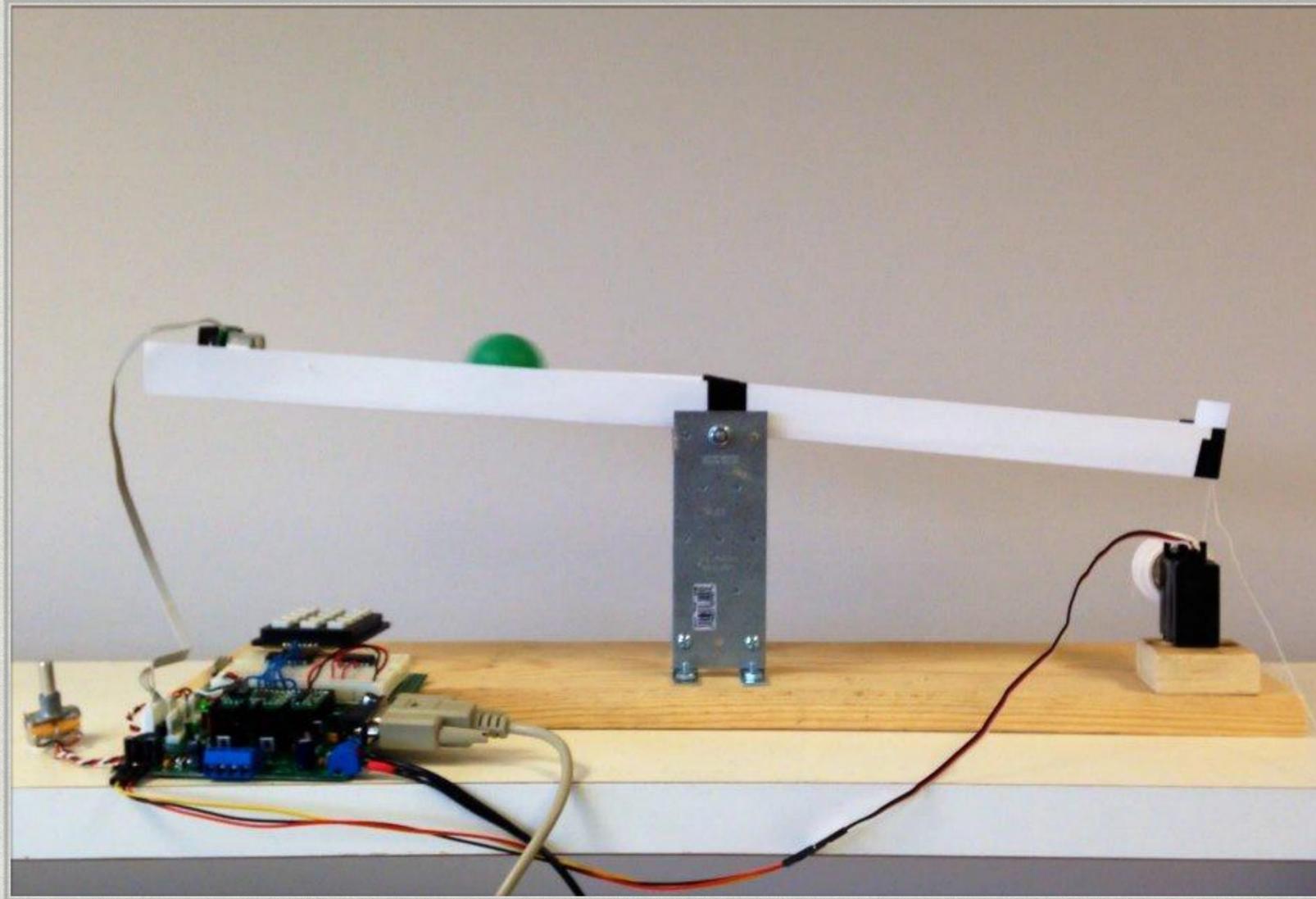
Problem Statement

- * Create a simple feedback environment that can be used to demonstrate various feedback control systems using a microcontroller

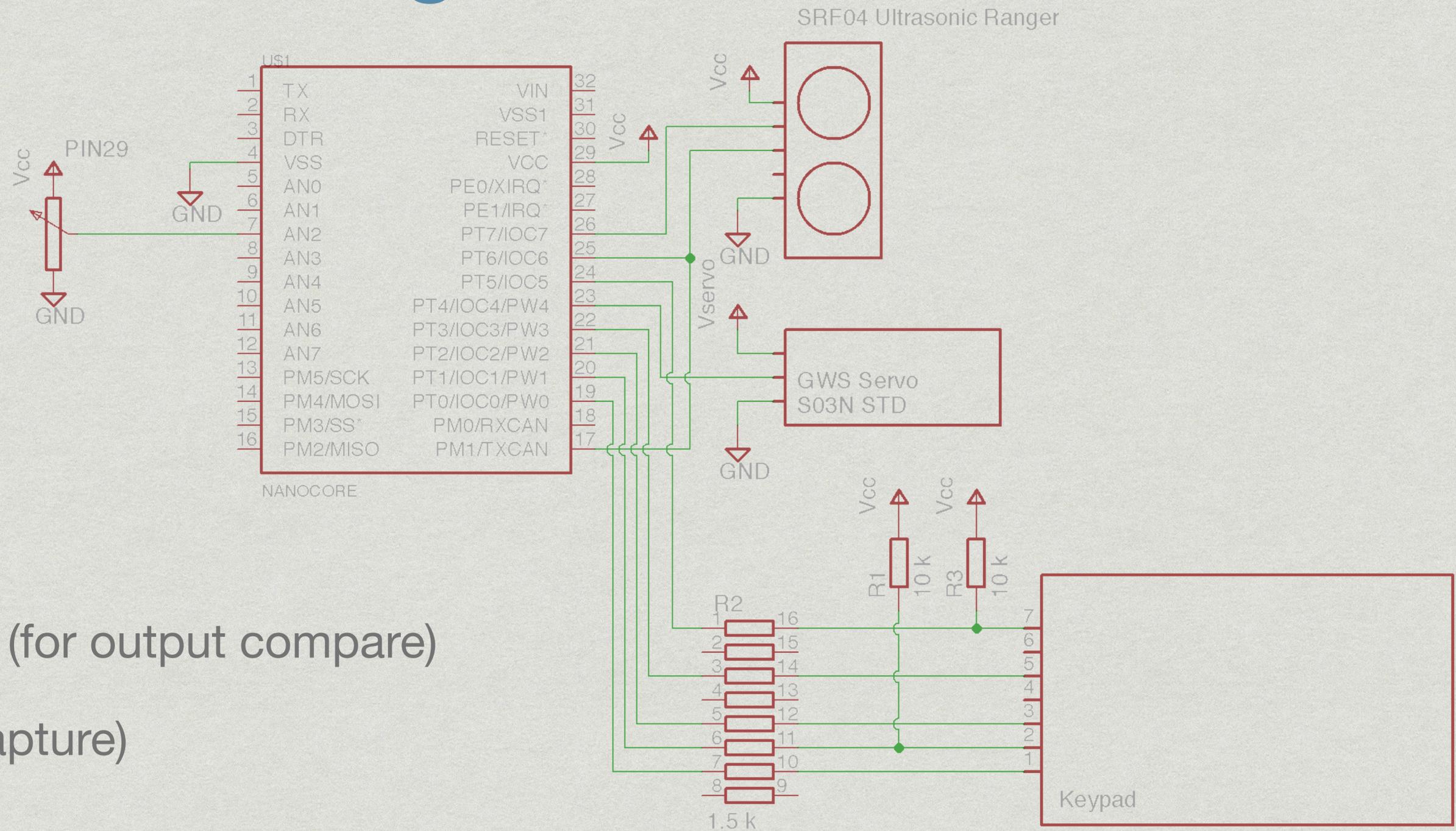
Modes of Operation

- * Manual control
- * Proportional positional control
- * Proportional-derivative control

Hardware Design



Hardware Design



- * PT6-PM1/J1 (for output compare)
- * PT7 (input capture)

Software Design

LOW LEVEL

- * Sonar (IC, OC)
- * Servo control (PWM)
- * Potentiometer (AD)
- * Keypad (time muxing)

HIGH LEVEL

- * Mode switch from keypad (main loop, sets global variable)
- * Control (RTI, state machine based on global variable)

Control Structure

MANUAL CONTROL

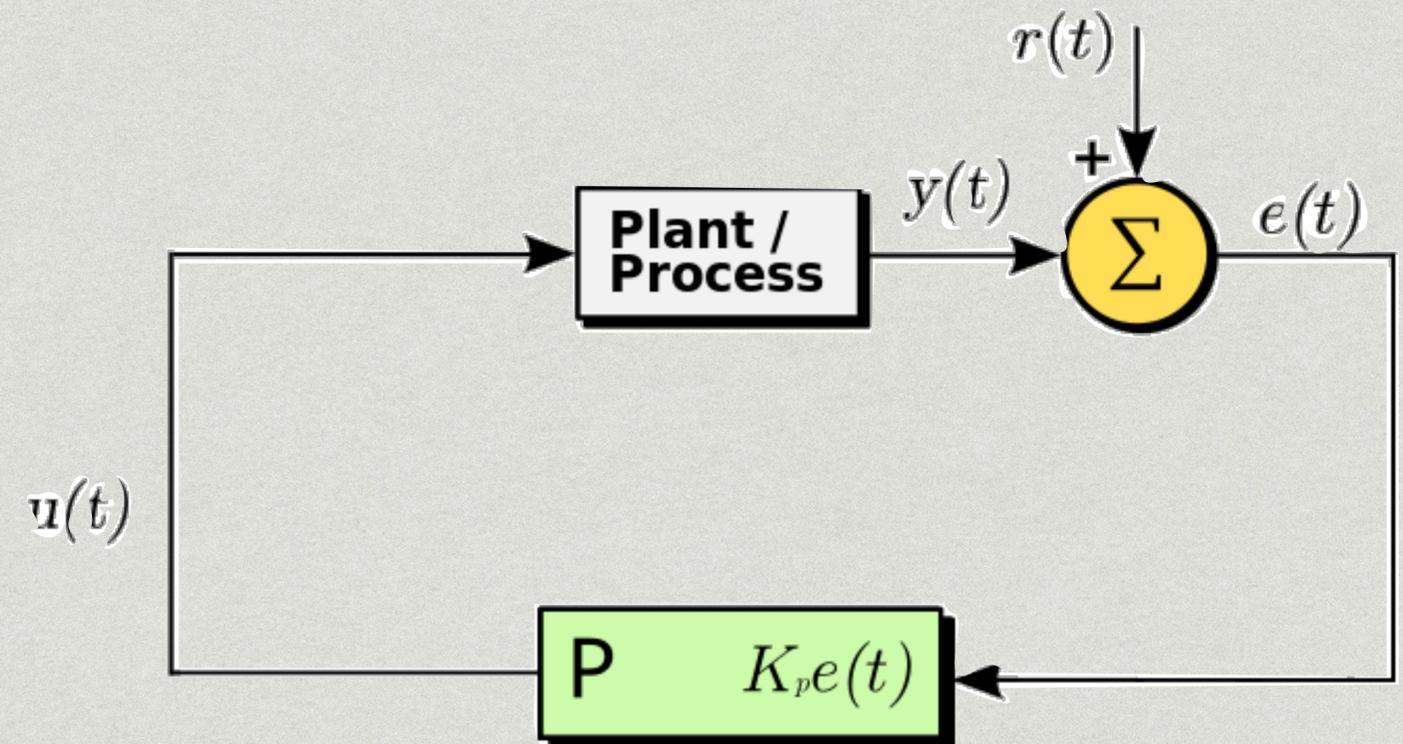
- * Humans are very good at many kinds of control (ex. adjusting shower temperature without burning oneself)
- * Human control can break down when control inputs need to be small.
- * Implementation
 - * Read analog value on AN02 from trimpot
 - * Convert to digital value, store on ATDDDR0L
 - * Shift and scale ATDDR0L and store it into PWM Register



Control Structure

PROPORTIONAL CONTROL

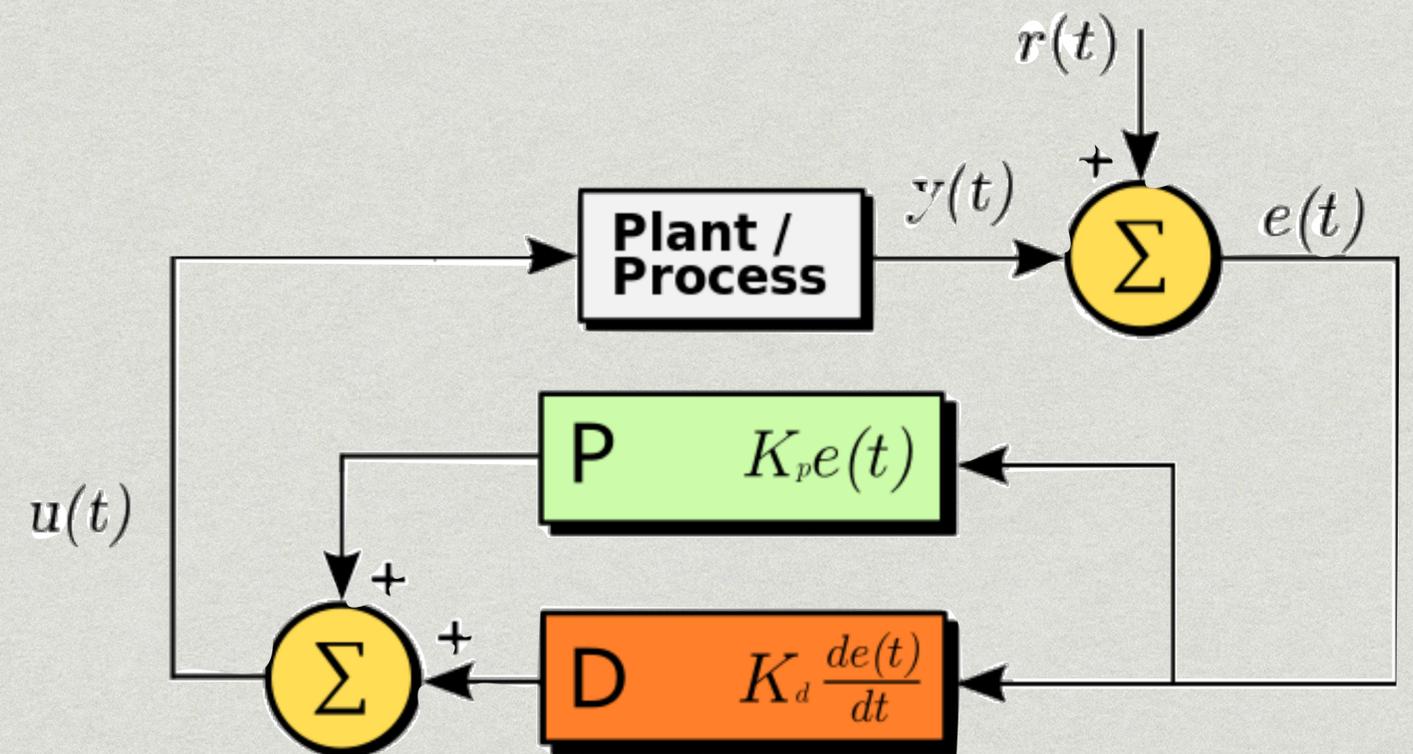
- * Uses the sonar to detect the position of the ball
- * Assigns a value to the PWM register based on the position of the ball
- * Gain (aggressiveness or conservativeness) of the response can be controlled using the potentiometer
- * Not stable for large disturbances



Control Structure

PROPORTIONAL-DERIVATIVE CONTROL

- * Uses Sonar to detect the position of the ball like P-only control
- * Calculates the ball's approximate velocity based on the previous position of the ball
- * Both the ball's position and velocity are used to calculate the controller response
- * Derivative Gain can be controlled with the trimpot
- * Creates stability for large disturbances, but overly aggressive derivative gain can make small disturbances less stable.



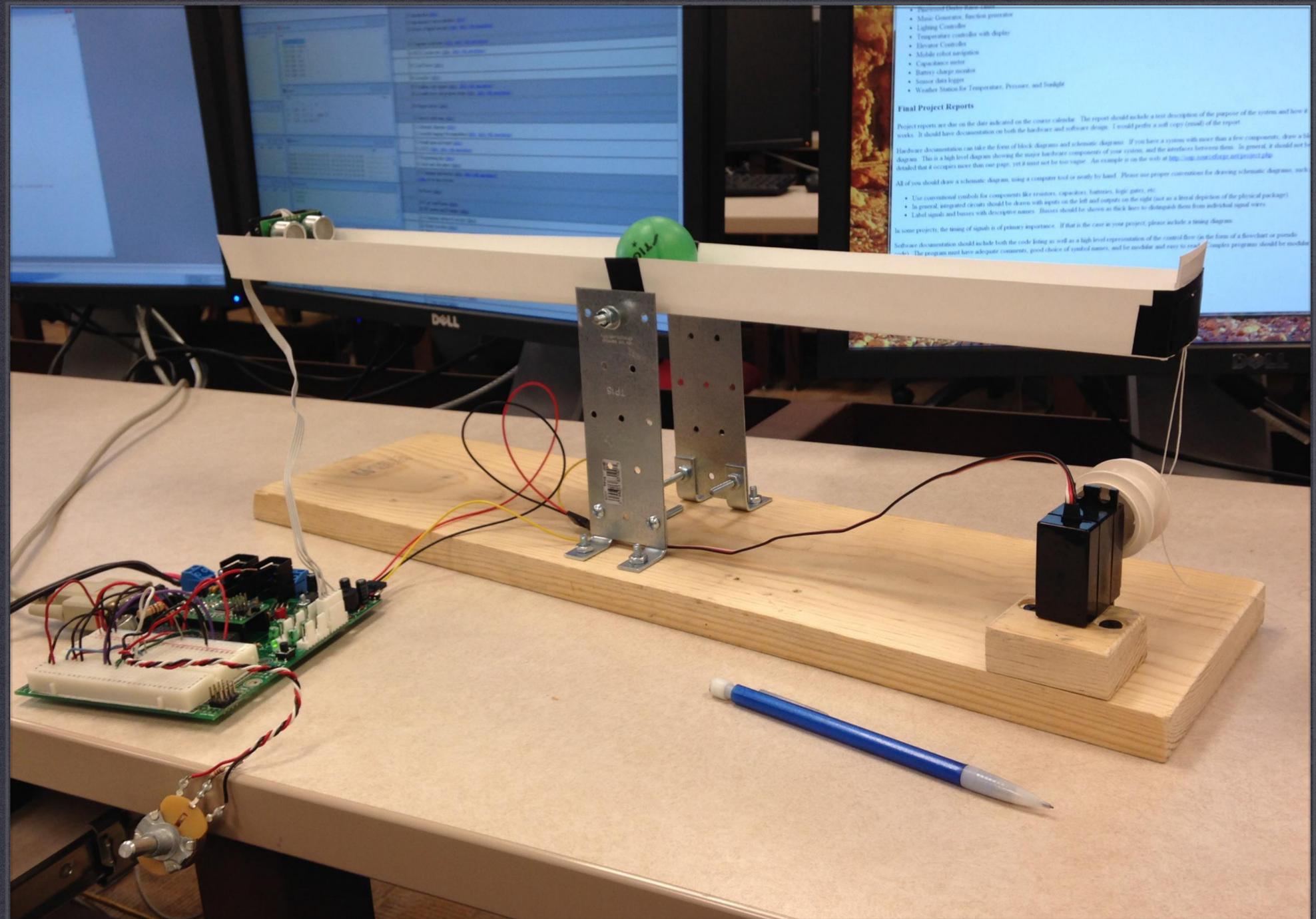
Results



Conclusion

- * Hardware/software interfacing
 - * Sonar
 - * Servo
 - * PWM, IC, OC, A/D, RTI
- * Control systems and design
 - * Manual
 - * Proportional
 - * P/D
- * Demonstration of control
- * Limiting factors
 - * Speed/resolution of servo
 - * Noise in sonar reading
- * Improvements
 - * Different physical control
 - * Multiple sensors
 - * More sophisticated control

QUESTIONS



- Pipeworked linear motion
- Music Generation: function generator
- Lighting Controller
- Temperature controller with display
- Elevator Controller
- Mobile robot navigation
- Capacitance sensor
- Battery charge monitor
- Sensor data logger
- Weather Station for Temperature, Pressure, and Humidity

Final Project Reports

Project reports are due on the date indicated on the course calendar. The report should include a brief description of the purpose of the system and how it works. It should have documentation on both the hardware and software design. I would prefer a soft copy (email) of the report.

Hardware documentation can take the form of block diagrams and schematic diagrams. If you have a system with more than a few components, draw a block diagram. This is a high-level diagram showing the major hardware components of your system, and the interfaces between them. In general, it should not be detailed that it occupies more than one page, yet it must not be too vague. An example is on the web at <http://comp.utoronto.ca/~dga/pjreport.pdf>.

All of you should draw a schematic diagram, using a computer tool or neatly by hand. Please use proper conventions for drawing schematic diagrams, such as:

- Use conventional symbols for components like resistors, capacitors, batteries, logic gates, etc.
- In general, integrated circuits should be drawn with inputs on the left and outputs on the right (just as a literal depiction of the physical package).
- Label signals and buses with descriptive names. Buses should be shown as thick lines to distinguish them from individual signal wires.

In some projects, the timing of signals is of primary importance. If that is the case in your project, please include a timing diagram.

Software documentation should include both the code listing as well as a high-level representation of the control flow (in the form of a flowchart or pseudo-code). The program must have adequate comments, good choice of symbol names, and be modular and easy to read. Complex programs should be modular.

```
#include <hides.h> /* common defines and macros */
#include "derivative.h" /* derivative-specific definitions */
//#include <stdio.h> /* contains sprintf() */
```

```
/******
```

Get a keypress (if any) from the keypad. If a key is pressed, it returns its code, which is an integer from 1 (top left) to 12 (lower right). If no key is pressed, it returns 0. If the same key is still pressed from last time, return 0.

```
*****/
```

```
int getkey(void);
void DelayuSec(int t);
void interrupt VectorNumber_Vrti rti_isr();
int processPWM(int input);
unsigned int T1;
long PWIDTH;
long PWIDTHLAST = 0;
long DELTAPWIDTH = 0;
unsigned int USEC;
```

```
unsigned int DISTCM;
int Kd = 0;
int i = 1;
int state = 3;
int checkkey = 0;
char cbit;

// Define the Port T bits corresponding to columns
char columnBits[] = {
0x04, // column 0 (left) is pin PT2
0x01, // column 1 (center) is pin PT0
0x08 // column 2 (right) is pin PT3
};
// Define the Port T bits corresponding to rows
char rowBits[] = {
0x02, // row 0 (top) is pin PT1
0x20, // row 1 (2nd) is pin PT5
//0x20, // row 2 (3rd) is pin PT5
//0x08 // row 3 (bottom) is pin PT3
};
```

Backup - Code

```

void main(void) {
    DDRT = 0x5D; //Set PT6, PT4, PT3, PT2, PT0 as outputs, rest of pins for input.

    ATDCTL2 = 0xC0; //enable ATD and fast flag clear
    ATDCTL3 = 0x08; //set ATD for 1 channel conversion
    ATDCTL4 = 0x85; //set ATD for 2 MHz,2 sample clks,8 bits
    ATDCTL5 = 0xA2; //right justif, continuous conv, use AN02

    RTICTL = 0x3F; //Set RTI clock to lognest period
    CRGINT = 0x80; //RTIE=1 to enable rti interrupts

    PWMPRCLK = 4; // Set clock prescaler (M=16)
    PWMCLK = 0xFF; // Select clock SA
    PWMSCLA = 75; // Divider for clock SA (N=12)
    PWMPOL = 0xFF; // Positive going pulse
    PWMDTY4 = 7; // Pulse width 15*100usec
    PWMPER4 = 200; // Period is 200*100usec
    PWME = 0xFF; // Enable all channels
    MODRR = 0x10; // Route PWM channel to PIN 4 on PTT.

    //Setup
    DDRM = 0x00;
    TSCR1 = 0x90;
    TSCR2 = 0x06;

    //Input Capture
    TIOS &= ~0x80;
    TCTL3 = 0x40;
    TFLG1 = 0x80;
    TIE |= 0x80;

    //Output Compare
    TIOS |= 0x40;
    TCTL1 = 0x30;
    TFLG1 = 0x40;
    TIE |= 0x40;

    EnableInterrupts;

    for(;;) {

        checkkey=getkey();
        if (checkkey != 0) {
            state = checkkey;

        }

        _FEED_COP(); /* feeds the dog */
    } /* loop forever */
}

```

Backup - Code

Backup - Code

```
// IC7 ISR

void interrupt VectorNumber_Vtimch7 ic7_isr(void) {

    // Test bit EDG3A to see what IC7 was configured to do

    if (TCTL3 & 0x40) { // EDG7A was a 1, meaning IC7 was configured to look
for rising

        T1 = TC7;

        TCTL3 = 0x80; // EDG7B:EDG7A = 1:0 to look for falling edge next

    } else {

        // EDG7A was a 0, meaning IC7 was configured to look for falling

        PWIDTHLAST = PWIDTH;

        PWIDTH = TC7 - T1;

        TCTL3 = 0x40; // EDG7B:EDG7A = 0:1 to look for rising edge next

    }

}
```

```
// OC6 ISR

void interrupt VectorNumber_Vtimch6 oc6_isr(void) { // Test bit OL6 to see
what OC2 was configured to do

    if (TCTL1 & 0x10) { // OL6 was a 1, meaning OC6 went high upon
compare

        TC6 = TC6 + 4; // This is about 10 us

        TCTL1 = 0x20; // OM6:OL6 = 1:0 to go low next

    } else { // OL6 was a 0, meaning OC6 went low upon compare

        TC6 = TC6 + 37500; // This is about 100 ms

        TCTL1 = 0x30; // OM6:OL6 = 1:1 to go high next

    }

}
```

```

// Keypad

int getkey(void) {

    int col, row, rbit, key;

    static lastKey = 0;

    int keycodes[2][3]= {{1,2,3}, {4,5,6}};

    // Scan across columns 0,1,2

    for (col=0; col<3; col++) {

        // Write a 0 to the bit corresponding to the column;

        // write 1 to all other bits in Port T

        cbit = columnBits[col];

        PTT = ~cbit;

        // Scan across rows 0,1

        for (row=0; row<2; row++) {

            // Get the mask corresponding to this row

            rbit = rowBits[row];

            // Read Port T and check only that bit; a key is

```

```

// pressed if the bit is 0.

        if (!(PTT & rbit)) { // A key is pressed

            //DelayuSec(10000);

            if (!(PTT & rbit)) { // check same key again

                key = keycodes[row][col]; // get keycode

                // If this is still the same key, return 0

                if (key==lastKey) return(0);

                // It is a new key being pressed

                lastKey = key;

                return(key);

            }

        }

    }

    // We got to here, so no key is being pressed

    lastKey = 0;

    return(0);

}

```

Backup - Code

Backup - Code

```
void interrupt VectorNumber_Vrti_rti_isr() {

    CRGFLG = 0x80;

    if (state == 1) {

        PWMDTY4 = (ATDDR0L/15) + 7;

    } else if (state == 2) {

        if ((PWIDTH > 60) && (PWIDTH <1010)) {

            PWMDTY4 = processPWM(24-(PWIDTH/(55)));

        } else {

        }

    } else if (state == 3) {

        /*****

        This is the code for the PD controller.

        From experimentation, the normal range for DELTAPWIDTH is 0 to about 120

        Kd ranges from 10 to about 52 and adjusts the gain of the derivative term

        A smaller Kd produces a more aggressive response (better for stabilizing large disturbances).

        A larger Kd produces a more conservative response (better for stabilizing small disturbances).

        *****/
```

```
        i

        f ((PWIDTH > 60) && (PWIDTH < 1010)) {

            Kd = (ATDDR0L/6) + 10;

            DELTAPWIDTH = (PWIDTH - PWIDTHLAST);

            if (DELTAPWIDTH > 150) { //this if statement attempts to get rid of some of large spikes in

                // DELTAPWIDTH, Only correcting the derivative seems to work best

                PWIDTHLAST = PWIDTH;

                DELTAPWIDTH = 0;

            }

            PWMDTY4 = processPWM(24 - (PWIDTH/55) - (DELTAPWIDTH/Kd));

        }

    }

}
```

Backup - Code

```
int processPWM(int input) {  
  
    int output = 0;  
  
    if (input < 7) {  
  
        output = 7;  
  
    } else if (input > 24) {  
  
        output = 24;  
  
    } else {  
  
        output = input;  
  
    }  
  
    return output;  
  
}
```