TA : Mike Pridgen
Adam Barnett
Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

Intelligent Machines Design Lab EEL-5666

SkyRiser

Final Written Report

Jeffery Lettman
4/22/2008

Table of Contents

Abstract

The goal of this project is to bring the time and cost saving advantages of mobile platforms to the construction site. The design called for a mobile platform capable of locating the studs on a construction site and securing the sheetrock using anchor screws. To accomplish this task, the platform will need to be capable of: tracking along walls, raising itself to a number of vertical heights, and drilling into the wall. For ease of control, the motion will be achieved using two motors mounted along a central axis. The wheels will be housed inside the platform to make wall following behaviors easier. The vertical lift will be accomplished using rack and pinion travel to accomplish forklift type motion. Linear motion required apply pressure to the drill and drive the drill forward will be accomplished through the use of a screw driven linear actuator. The robot will use a capacitance stud sensor to detect density differentials in the wall. Sonar rangefinders will be employed towards the end of achieving wall following behaviors. The value of a robot capable of meeting the goals laid out in this report would be substantial, allowing onsite construction projects, like the forty-eight currently in progress on campus, to be completed at considerable savings to cost and time.

Introduction

Computer numerically controlled (CNC) machining and stationary robotic arms have rapidly become staples of the construction and manufacturing industry. However autonomous tools have yet to become prevalent in this field limiting the ability of robotic agents to assist in on site assembly. The idea I have for my robotic platform is an autonomous construction stage. The specific goal of the design is to support building construction crews by anchoring wall paneling in place. The robot would be able to locate a stud and raise itself to sequentially drill a number of support screws into the wall.

Integrated System

The core processing power of my robot is the Atmel AVR MAVRIC-IIB, assembled and tested, Pin Headers, at a speed of 14.7456 MHz. The microcontroller will control all of the behaviors that my robot will emulate. The robot also has an LCD screen to display important status updates, and operational warnings.

Mobile Platform

The mechanical design needs to allow for easy navigation throughout the environment and provide the capability to follow walls along the perimeter of rooms. For this reason I believe that a two wheel platform would be the best choice. The wheels could be tucked inside the robot's frame in order to provide better wall following characteristics. The issue with this design will be the support and pressure needed to get the screw into the wall. To resolve this problem the wheels will be placed perpendicular to the motion of the screw and brackets will be angled in place to prevent tipping. Most importantly, the design will need a mechanical means of lifting itself to provide a vertical line of support screws in the wall. My original plan as illustrated in Figure 2.1 was to use a scissor lift to accomplish this vertical motion. Unfortunately in testing a build of the lift itself I discovered two design flaws. The scissor lift assembly, when made out of either metal or plastic, added significant weight to the lift mechanism. This of course resulted in a much higher torque requirement from the motor. In addition, using a horizontal drive system to generate the lift meant that for the first inch of vertical travel the horizontal motor assembly would be pushing directly into the lift, as illustrated below in Figure 2.2. With one side of the scissor lift necessarily fixed, almost all of the initial applied force went in to compressing the pin and reaction forces in the fixed point.

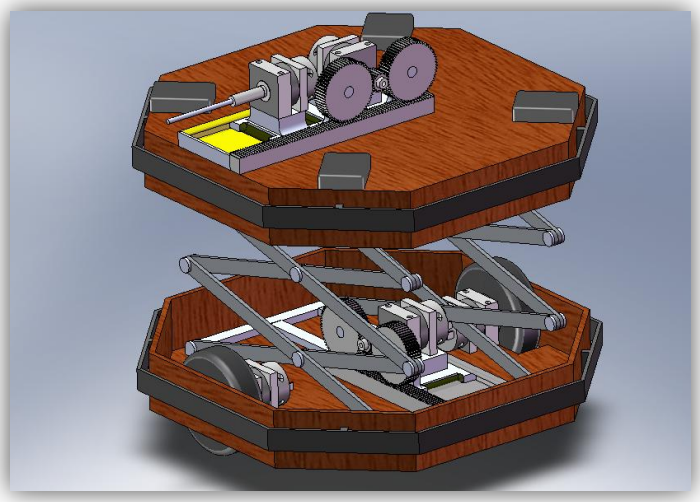

Figure 2.1 : SolidWorks Design Concept

While neither of these problems were necessarily deal breakers in and of themselves, in combination they created serious obstacles. In further researching scissor lifts in an attempt to overcome these hurdles, I learned that in addition to a horizontal pushing force, many scissor lift assemblies use pneumatic pistons to overcome this initial stalemate of forces. This solution is impractical for this small scale robot, so I decided to scrap this lift mechanism and design another.

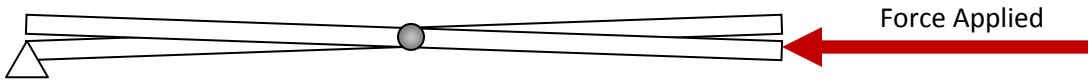

Figure 2.2 : Force Diagram

The lift mechanism I ended up designing in small scale tests worked like a fork lift. The rack gears were vertically aligned and the pinions were fixed perpendicular and powered be the worm gear assembly so that the lift would not slip under heavy loading, nor would the motor be required to stay on in order to provide breaking power to the assembly. This setup is illustrated in Figure 2.3.

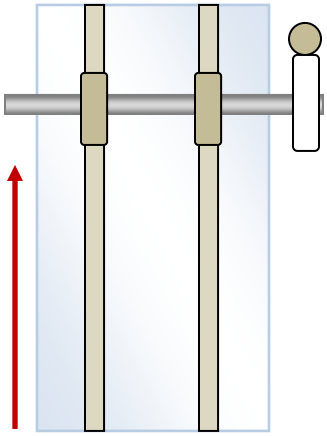

Figure 2.3: Forklift Diagram

Success of these small scale tests in generating high torque and resisting slipping under loading were encouraging enough to proceed forward with this as the main lift mechanism. I was forced to scrap the previously t-teched parts of my design at this point to proceed forward with new materials. This new design was optimal in that it utilized the same gears I previously ordered for use in the scissor lift. Rather than balsa wood, this design was built out of Plexiglas. The robot has 3 levels to provide space for all of the necessary components. The third level is 10 inches above the base and serves as a support platform for the worm gear and pinion assembly as well as a support for the upper end of the forklift mechanisms travel.

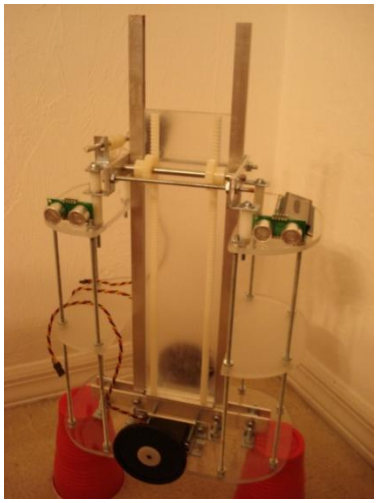

Figure 2.4: Front View

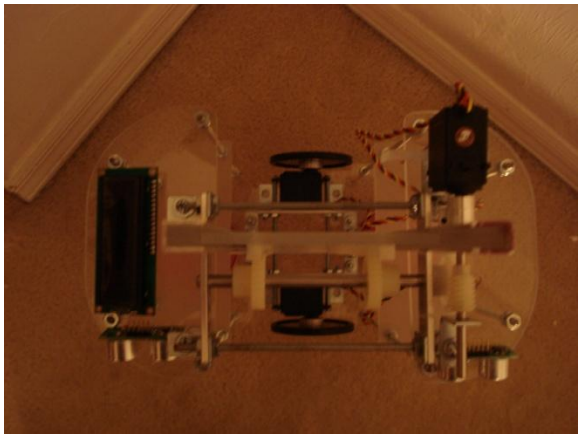

Figure 2.5: Top View

Actuation

In addition to the vertical motion of the mobile platform, the design also requires linear and rotational motion to drive the anchor screws into the sheetrock. Initially I planned to accomplish this with two motors for the two types of requisite motion. However, in the redesign of my platform, space, weight and wire management became more noticeable concerns. For these reasons I designed a mechanism which, when using self drilling drywall screws, will only need one motor to provide both these motions. The design as shown in Figure 3.1, uses a moving lead screw and fixed bolts to combine the motions. The motor is attached to a slide with guide wheels allowing it to only move in the x-direction along with the progress of the lead screw. The other end of the screw is fitted with a Philips head bit to drive the anchor screws. This whole assembly raises and lowers with the forklift. To cut down on the weight that needs to be lifted, the screws themselves are held in place at the pre-established levels. This feed mechanism is therefore supported by the frame of the robot and counterbalanced by the lead screw motor. This is not a particularly fast method, but it is only required that it move through an inch and a half of travel.

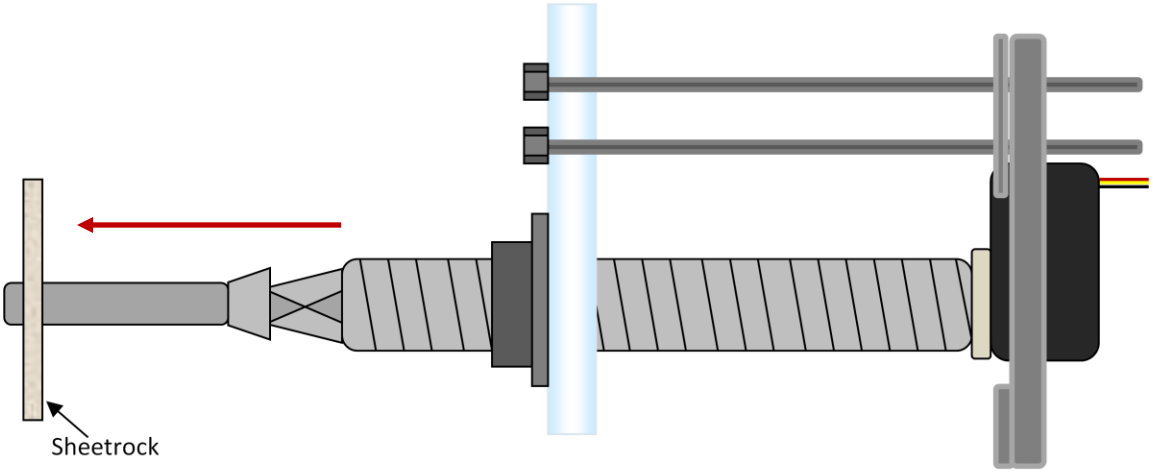

Figure 3.1 : Screw Mechanism

Sensors

One of the first tasks for this robot upon being introduced into an environment will be to find a wall so it can begin tracking and looking for studs.

The placement of the sonar units and the stud sensor on the robot can be seen in the image of the robot, Figure 4.1. I am using the SRF 05 sonar units which came highly recommended in class for ease of use. The sonar units require only four of the pins for its operation, two for power one to provide the trigger signal and one to send the echo data to the Mavric IIB Board.

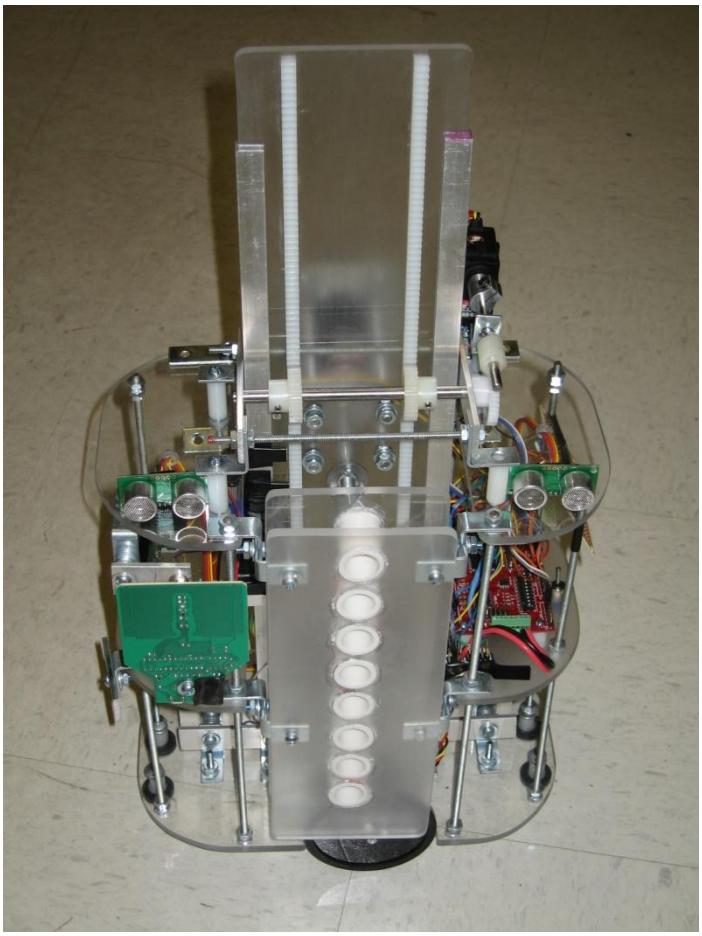

Figure 4.1 : Sonar Placement

These SRF 05 sonar units are used to provide the mobile platform with a wall following behavior. Because of the unique design of my robot, it will only need to follow the wall along one side. The sonar units along this side are arranged as indicated in Figure 4.2. Data returned from the sensors in this array will allow for the determination of the robots orientation relative to the wall. If the distance is greater on the front end the far wheel should be activated. The opposite case is also true.

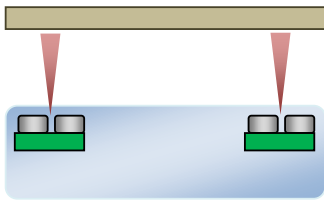

Figure 4.2 : Sonar Layout

Bump sensors are used to set the top of the vertical lifts path of motion as a place to pause and calibrate the readings on the CDS cells. They are also be used to determine the horizontal progress of the lead screw. Bump sensors are wired as a normally closed loop so if either one is depressed it is know that the screw driver assembly has reached its end of travel. A bump sensor is also used to extend the stud sensors power switch from the side of the sensor to the base of the forklifts travel. The bump sensor is soldered onto the stud sensor bypassing its own power button. When the lift is returned to its bottom most position the stud sensor is turned on.

Over the course of this project I went through three different stud finders in attempting to determine which would be optimum for use in this project. The first stud finder I purchased from Home Depot was a Zircon OneStep StudSensor i65. This stud finder uses capacitance plates to determine the true center of a stud. The device has Center Vision and is capable of determining the true sensor of a stud while differentiating it from electrical lines in the wall. However this sensor was far too sensitive and lost its calibration easily, even when being operated by hand. Therefore the final stud sensor used in the project was a more straightforward edge detector from Zircon, the StudSensor™ Pro SL-AC. The specifications of the Zircon StudSensor are as follows, as provided by the Zircon website:

Figure 4.3 : Stud Sensor

Dimensions: 6.07 in. H x 2.70 in. W x 1.18 in. D (154mm x 68mm x 30mm)

Weight: 5.7 oz. (1163g) with battery

Battery type: 9-V alkaline (included)

Position accuracy

Wood studs: Stud Scan mode: Typically within 1/8inch (3mm) using the dual scan and mark procedure; Deep Scan mode: typically within 3/16 inch (5mm) using the dual scan and mark procedure

Metal studs: Typically within 1/2 inch (13mm) using the dual scan and mark procedure.

Depth: Up to 3⁄4 inch (19mm) in Stud Scan mode; Up to 1-1/2 inch (38mm) in Deep Scan mode

AC position accuracy: Typically 90-250 V at 50-60 Hz within 6 inches (150 mm) of a hot unshielded wire in drywall.

AC depth: In typical drywall with Romex™ wiring, wires can be detected up to 2 inches (50 mm) deep

NOTE: Sensing depth and position accuracy can vary due to moisture content of materials, wall texture, and paint.

Operating Temperature: 20° to 120°F

Storage Temperature: -20° to 150°F

Humidity 80% RH (noncondensing)

Water Resistance: Splash and water resistant, not waterproof

When the stud is located the sensor uses an LED light to inform the user of its presence and location. Bypassing the actual inter calculations of the stud finder, I use a CDS cell to determine when the LED flashes the presence of the stud.

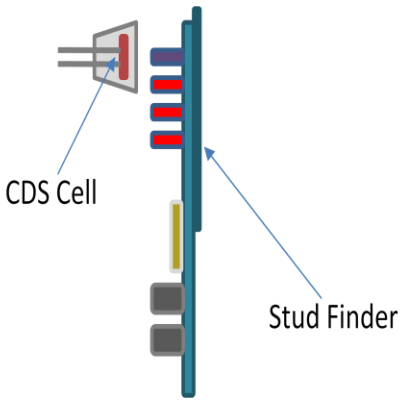

Figure 4.4 : CDS Stud Finder Arrangement

I also use CDS cells to determine the vertical progress of the lift. The CDS cell is affixed to the lift and blackout bars are affixed to the frame of the robot. By counting the number of bars that the CDS cell passes under the vertical progress of the lift can be tracked. The bottom of the lifts progress, and where all lift progress starts is the base of the robot. A bright blue LED marks this bottom position and differentiates this point from the blackout bars above.

To calibrate the robot to different lighting environments I ran an analog CDS test, whose results are listed in the table below. I was unable to draw any direct correlations between the numbers in this table so I decided to have the robot calibrate the CDS values to the room each time it starts up. This process is covered more thoroughly in the behavior section of the report.

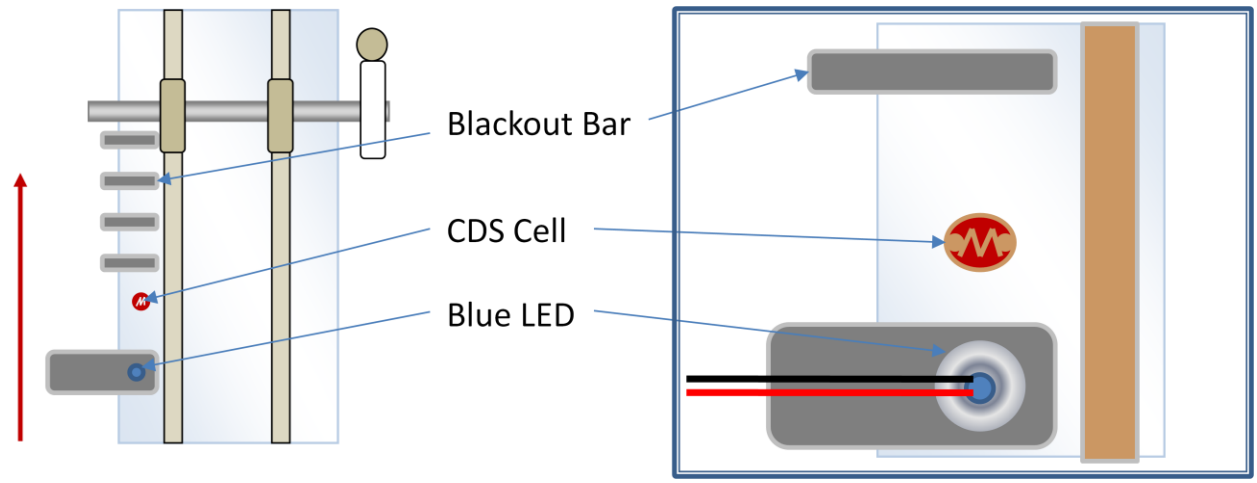

Figure 4.5 : CDS Vertical Lift Arrangement

Ambient Light Conditions Analog Cadmium Sulfide Photo-Resistor Cell Values			
Vertical Encoder Status	Low Light	Medium Light	High Light
LED On	8	9	13
Clear	13	15	29
Black Bar	36	49	116
Clear	10	12	26
Black Bar	31	42	112
Clear	10	13	26
Black Bar	36	48	110
Clear	10	13	25
Black Bar	36	48	118

Table 4.1 : CDS Cell Values

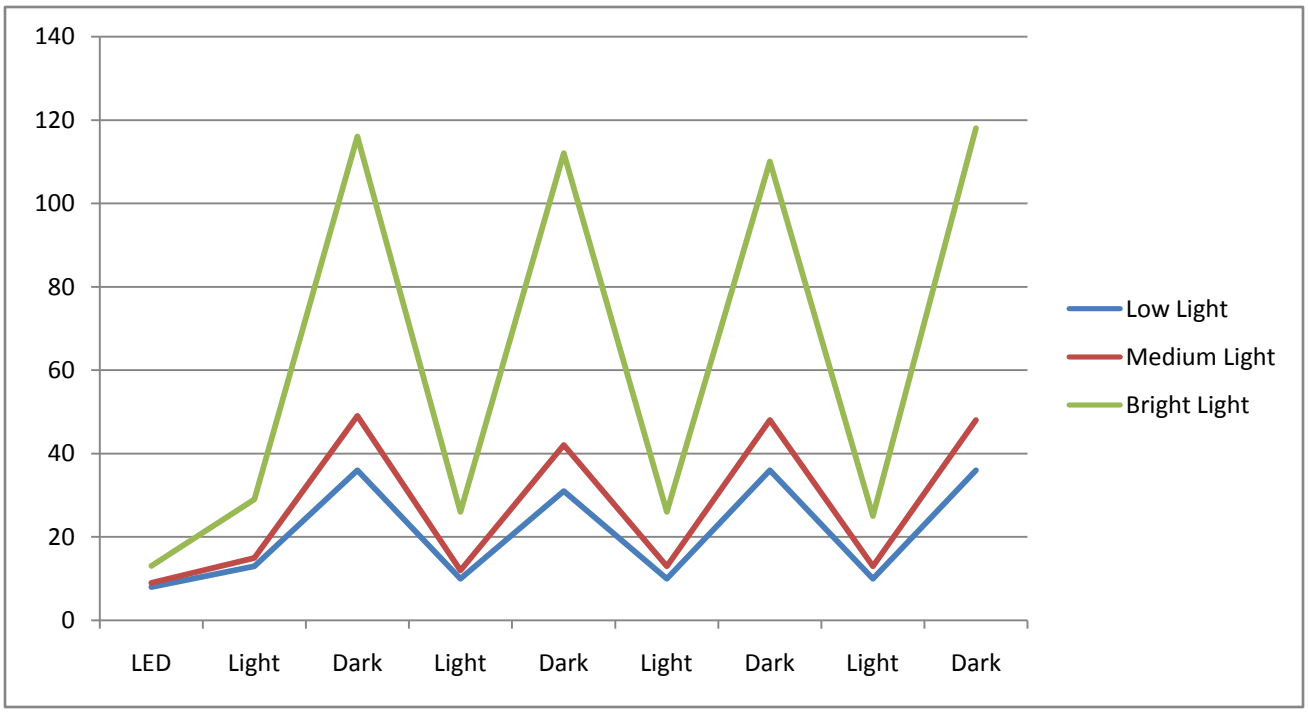

Chart 4.1 : CDS Cell Values

Behaviors

In turning the SkyRiser on it begins its initial calibration phase. As mentioned in the sensor section, changing the light in the room severely affects the accuracy of the lift CDS cell, making this calibration phase crucial. The lift is raised until the top bump sensor is tripped which marks a point at which the CDS cell is not covered. An average of the analog values at this height is determined and stored as the low value. The lift is then lowered until the value is twice as high as the low value and stops when it again returns to 35% of the low value. This ensures that it passes under a blackout bar. The highest value along this path of motion is stored as the high value. These values are then used as benchmarks when the lift goes through its drill pattern.

The SkyRiser then moves into its wall following behavior, attempting to align itself parallel to the wall. The robot moves towards the wall at an angle proportional to its distance from the wall. It does this by driving the outside wheel faster than the wheel close to the wall by a factor of the forward sonar value divided by the desired distance sonar value. When the robot is parallel to the wall within a pre selected range of sonar values, the platform stops moving forward and lowers the lift to turn on the stud sensor, it also stores the values of the forward and rear sonar units, it performs this only once. These values are then set as the benchmark values which the robot uses to align itself from the wall. This keeps the unit at the same distance from the wall as it was when the stud finder was calibrated. This is crucial because if the stud finder gets any closer to the wall it will register false positive stud findings.

When the stud sensor registers a stud, it bright LED light flashes which the CDS cell registers as a low value and relays to the Mavric IIB board. The platform then continues to wall follow until the light again turns off, so that the screw holding palette is aligned with the stud. SkyRiser then stops and begins its drill pattern.

The lift has two lift patterns which it alternates between. The first pattern drills in the first and third screws, while the second pattern affixes the second and forth. The lift reaches these heights by counting the number of blackout bars that the CDS cell passes beneath. When the lift is at the right height the drill runs until the bump switch is depressed, then backs up until the rear bump switch is depressed. When the drill pattern is complete, the lift is lowered until the CDS cell detects the blue LED, the stud sensor is once again turned on, and the values of the forward and rear sonar pings are once more stored as a reference for wall following. The SkyRiser can now resume its wall following behavior.

Conclusion

I believe that the real value of this robotic platform could be truly substantial. The platform could be set up to run at a site with little or no supervision so that workers could be set to other tasks, cutting down construction time and eliminating a repetitive and unskilled job. Onsite construction projects, like the forty-eight currently in progress on campus, could be completed at considerable savings to cost and time.

Improvements

While I believe that this robotic platform was a success, given more time on this project I would make the following improvements.

- A ratchet fixed at the end of the drill so that even if the screw is not countersunk, the screw will not be pulled back out of the wall.
- A self feeding screw system allowing many more screws to be used and allowing the platform to run longer.
- The incorporation of an IR module to detect the corner of a wall so that the wall following pattern could be extended all around the perimeter of a room.
- The inclusion of a motion detector to determine when a person is nearby to improve the safety of the machine at a construction sight.
- Replacing the gears of the ¼ scale servo motor with metal gears for better strength.
- Smoother wall following behavior through the addition of a better wall following algorithm.

References

[1] R. Berendsohn, 7 New Stud Finders Let You See Through Walls, Popular Mechanics, July 2004.

[2] How Stud Finders Work, www.howstuffworks.com, 2006.

[3] May, Andrew, Final Report: Ant. Intelligent Machine and Design Lab, April 25, 2006.

[4] Ammons, Wiley, Final Report: Zircon. Intelligent Machine and Design Lab, December 4th, 2001.

In assembling my code, included in the appendix below, I used the sonar and LCD code as developed by Professor Arroyo and I owe a huge thanks to Adam Barnett for his help with my servo code.

Appendix

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>
#include <avr/pgmspace.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

#define MOTOR_Lift OCR1A                    //pin5 port B
#define MOTOR_Drill OCR1B                   //pin6 port B
#define MOTOR_L OCR3A                       //pin3 port E
#define MOTOR_R OCR3B                       //pin4 port E

void init_timer();
void motor_move(uint16_t, uint16_t);
void ad_init(void);

void config_adc(void);
int  analog(int);

// Bump

#define BUMP PORTG // Bump switches will bo on PORTG
#define BUMP_DDR DDRG
#define BUMP_PIN PING

#define DIR_L PORTE2
#define DIR_R PORTE5
#define MOTOR_L OCR3A               //pin3 port E
#define MOTOR_R OCR3B               //pin4 port E

/*A-to-D*/
#define DDRAD DDRF
#define PORTAD PORTF

/*Global variables*/
/*Timer*/
volatile uint16_t us_48_count;
volatile uint16_t ms_count;

/*Sonar rangers*/
#define DDRSRF DDRD
#define PORTSRF PORTA
#define PINSRF PINA
#define L_TRIGGER 0x02
#define L_ECHO 0x01
#define R_TRIGGER 0x10
#define R_ECHO 0x08


/*LCD Panel*/
void lcd_delay();       // short delay (50000 clocks)
void lcd_init();        // sets lcd in 4 bit mode, 2-line mode, with cursor on and set to blink
void lcd_cmd();         // use to send commands to lcd
void lcd_disp();        // use to display text on lcd
void lcd_clear();       // use to clear LCD and return cursor to home position
void lcd_row(int row);  // use to put the LCD at the desired row
```

```
/*Delay functions*/
void us_48_sleep(uint16_t us_48)
{
TCNT0 = 0;
us_48_count = 0;
while (us_48_count != us_48) // Each loop takes 48 us NOT 1 us,
                                                        //based on the 128 divisor I set in prescalers in function declared
                                                        //timer_init()
;
}
void ms_sleep(uint16_t ms)
{
TCNT0 = 0;
ms_count = 0;
while (ms_count != ms*21) // Each while loop takes 48 us, but
;
}


/*A-to-D functions*/

void config_adc(void)
{
  DDRF = 0b00000000; // set port F to all input
              // Note: when JTAGEN fuse is set, F4 - F7 don't work
  PORTF = 0x00;      // make sure pull up resistor is not enabled

  ADMUX = 0b01000000; // 5V reference, select channel0 (pin F0)
  ADCSRA |= 0b10000111; // turn on ADC, don't start conversions
                // free funning
                // divide clock by 128

}



int analog(int analogch)
{
    int anval;
   ADMUX = 0b01000000|analogch;
   // Start AD conversion.
   ADCSRA |= (1 << ADSC);
   // Wait for ADC conversion to complete.
   while ( ADCSRA & (1 << ADSC) );
     anval = ADCL | (ADCH << 8); //place ACD value into one variable
   return anval;
}


uint16_t ad_readn(uint8_t channel, uint8_t n)
{
uint16_t t;
uint8_t i;
t = analog(channel); // Dummy read
/*Sample selected channel n times, take the average*/
t = 0;
for (i=0; i<n; i++)
{
lcd_delay();
t += analog(channel);
```

```
}
/*Return the average of n samples*/
return t / n;
}


// initialize PWM timer
void init_timer()
{
	//initializes motors, enable OC3A, B, and C

	DDRE = 0xFF	;		//PORT E out
	//DDR_OC3A = 0b1;
	//DDR_OC3B = 0b1;



	TCCR3A = 0xA8;  //10101000 clear OCA,B,C on campare match- non-inv PWM
	TCCR3B = 0x12;  //00010010 mode  8 - PWM phase/freq correct, clk(io)/8
	ICR3 = 18432;	//PWM = 50hz.  1/(14.7456 MHz/8)*2*18432 = 1/50
	TCNT3 = 0x00;	//

	MOTOR_L = 0;	//not moving
	MOTOR_R = 0;
	MOTOR_Lift=0;

	//PORTE = 0x0;

	DDRB = 0xFF	;		//PORT B out
	//DDR_OC3A = 0b1;
	//DDR_OC3B = 0b1;



	TCCR1A = 0xA8;  //10101000 clear OCA,B,C on campare match- non-inv PWM
	TCCR1B = 0x12;  //00010010 mode  8 - PWM phase/freq correct, clk(io)/8
	ICR1 = 18432;	//PWM = 50hz.  1/(14.7456 MHz/8)*2*18432 = 1/50
	TCNT1 = 0x00;	//

	MOTOR_Lift = 0;  //not moving
	MOTOR_Drill = 0;

}


void motor_move(uint16_t speed_R, uint16_t speed_L)
{
	int increment_L = 0;
	int increment_R = 0;

	if(MOTOR_L > speed_L)
	{
		increment_L = -1;
	}
	else if (MOTOR_L < speed_L)
	{
		increment_L = 1;
	}
	else
	{
		increment_L = 0;
	}
```

```
	if(MOTOR_R > speed_R)
	{
		increment_R = -1;
	}
	else if (MOTOR_R < speed_R)
	{
		increment_R = 1;
	}
	else
	{
		increment_R = 0;
	}

	while (MOTOR_L != speed_L || MOTOR_R != speed_R) // if either motor does not equal speed, run this loop
	{
		if (MOTOR_L != speed_L)
		{
			MOTOR_L = MOTOR_L + increment_L;
		}
		if (MOTOR_R != speed_R)
		{
			MOTOR_R =MOTOR_R + increment_R;
		}



		//lcd_cmd(0x01);		// clrscn rtn home
		//lcd_int(MOTOR_L);
	}
}

void special_motor_move(uint16_t speed_Lift, uint16_t speed_Drill)
{
	int increment_Lift = 0;
	int increment_Drill = 0;

	if(MOTOR_Lift > speed_Lift)
	{
		increment_Lift = -1;
	}
	else if (MOTOR_Lift < speed_Lift)
	{
		increment_Lift = 1;
	}
	else
	{
		increment_Lift = 0;
	}


	if(MOTOR_Drill > speed_Drill)
	{
		increment_Drill = -1;
	}
	else if (MOTOR_Drill < speed_Drill)
	{
		increment_Drill = 1;
	}
	else
	{
```

```
		increment_Drill = 0;
	}

	while (MOTOR_Lift != speed_Lift || MOTOR_Drill != speed_Drill) // if either motor does not equal speed, run this
loop
	{
		if (MOTOR_Lift != speed_Lift)
		{
			MOTOR_Lift = MOTOR_Lift + increment_Lift;
		}
		if (MOTOR_Drill != speed_Drill)
		{
			MOTOR_Drill =MOTOR_Drill + increment_Drill;
		}



		//lcd_cmd(0x01);		// clrscn rtn home
		//lcd_int(MOTOR_L);
	}
}


void stop(void)
{
motor_move(1398,1398);
}

void forward(int speed)
{
motor_move(1400+speed, 1400-(speed*1.23)); // Servos have to correct drift
                                                                                                    //(Hence, the "...*0.91")
}

void R_turn(int speed)
{
motor_move(1400-speed, 1400-(speed*1.23));
}

void L_turn(int speed)
{
motor_move(1400+speed, 1400+(speed*1.23));
}

void reverse(int speed)
{
motor_move(1400-speed, 1400+(speed*1.23));
}


void softR_turn(int speed)
{
motor_move(1400+(speed*1.7), 1400-(speed*1.23));
}

void softL_turn(int speed)
{
motor_move(1400+speed, 1400-(speed*1.5*1.23));
}

void slightR_turn(int speed)
{
```

```
motor_move(1400+(speed*1.5), 1400-(speed*1.23));
}

void slightL_turn(int speed)
{
motor_move(1400+(speed*1.2), 1400-(speed*1.23));
}

void variableR_turn(int speed, double ratio)
{
motor_move(1400+(speed*ratio), 1400-(1.1*speed));
}


void lift_up(int speed)
{
special_motor_move(1435+speed,1400);
}

void lift_down(int speed)
{
special_motor_move(1435-speed,1400);
}

void drill_in(int speed)
{
special_motor_move(1435, 1400+speed);
}

void drill_out(int speed)
{
special_motor_move(1435, 1400-speed);
}

void stop_special()
{
special_motor_move(1435, 1400);
}


void lcd_delay()      // delay for 10000 clock cycles
{
  long int ms_count = 0;
  while (ms_count < 500)
  {
    ms_count = ms_count + 1;
  }
}

void lcd_cmd( unsigned int myData )
{



  unsigned int temp_data = 0;

  temp_data = ( myData | 0b00000100 );
  temp_data = ( temp_data & 0b11110100 );
  PORTC = temp_data;
  lcd_delay();
  PORTC = (temp_data & 0b11110000);
```

```
  temp_data = ( myData << 4 );


  temp_data = (temp_data & 0b11110100);   // temp_data now contains the original
  temp_data = (temp_data | 0b00000100);   // lower nibble plus high clock signal

  PORTC = temp_data;                  // write the data to PortC
  lcd_delay();
  PORTC = (temp_data & 0b11110000);      // re-write the data to PortC with the clock signal low (thus creating the falling edge)
  lcd_delay();

}

void lcd_disp(unsigned int disp)
{



  unsigned int temp_data = 0;

  temp_data = ( disp & 0b11110000 );
  temp_data = ( temp_data | 0b00000101 );
  PORTC = temp_data;
  lcd_delay();
  PORTC = (temp_data & 0b11110001);
  lcd_delay();                // upper nibble

  temp_data = (disp << 4 );
  temp_data = ( temp_data & 0b11110000 );
  temp_data = ( temp_data | 0b00000101 );
  PORTC = temp_data;
  lcd_delay();
  PORTC = (temp_data & 0b11110001);
  lcd_delay();                // lower nibble

}


void lcd_init()
{
  lcd_cmd(0x33);      // writing 0x33 followed by
  lcd_cmd(0x32);      // 0x32 puts the LCD in 4-bit mode

  lcd_cmd(0x28);      // writing 0x28 puts the LCD in 2-line mode

  lcd_cmd(0x0F);      // writing 0x0F turns the display on, curson on, and puts the cursor in blink mode

  lcd_cmd(0x01);      // writing 0x01 clears the LCD and sets the cursor to the home (top left) position

  //LCD is on... ready to write

}

void lcd_string(char *a)
{



  while (*a != 0)
  {
    lcd_disp((unsigned int) *a);   // display the character that our pointer (a) is pointing to
    a++;                  // increment a
```

```
  }
  return;

}


void lcd_int(int value)
{



  int temp_val;
  int x = 10000;        // since integers only go up to 32768, we only need to worry about
                        // numbers containing at most a ten-thousands place

  while (value / x == 0)   // the purpose of this loop is to find out the largest position (in decimal)
  {                                 // that our integer contains.  As soon as we get a non-zero value, we know
    x/=10;                  // how many positions there are int the int and x will be properly initialized to the largest
  }                                 // power of 10 that will return a non-zero value when our integer is divided by x.

  if (value==0) lcd_disp(0x30);

  else while (x >= 1)           // this loop is where the printing to the LCD takes place.  First, we divide
  {                         // our integer by x (properly initialized by the last loop) and store it in
    temp_val = value / x;     // a temporary variable so our original value is preserved.Next we subtract the
    value -= temp_val * x;    // temp. variable times x from our original value.  This will "pull" off the most
    lcd_disp(temp_val+ 0x30); // significant digit from our original integer but leave all the remaining digits alone.
                              // After this, we add a hex 30 to our temp. variable because ASCII values for integers
    x /= 10;                  // 0 through 9 correspond to hex numbers 30 through 39.  We then send this value to the
  }                         // LCD (which understands ASCII).  Finally, we divide x by 10 and repeat the process
                                    // until we get a zero value (note: since our value is an integer, any decimal value
  return;                           // less than 1 will be truncated to a 0)

}

void lcd_clear()      // this function clears the LCD and sets the cursor to the home (upper left) position
{
  lcd_cmd(0x01);

  return;
}

void lcd_row(int row)   // this function moves the cursor to the beginning of the specified row without changing
{                                   // any of the current text on the LCD.

  switch(row)
  {

    case 0: lcd_cmd(0x02);
    case 1:   lcd_cmd(0xC0);

  }

  return;
}

int get_lSonar()
{

        int n = 0;

        PORTA = (PINA & 0x7F);    // these two lines create a rising edge
```

```
	PORTA = (PINA | 0x80);	// on PortA pin 7

	//lcd_clear();
	//lcd_string("rising edge done");

	while (n < 1)
	{
		//waste enough clock cycles for at least 10us to pass
		n += 1;
		n++;
		lcd_clear();
		lcd_int(n);
	}

	PORTA = (PINA & 0x7F);	// force PortA pin 7 low to create a falling edge
							// this sends out the trigger

	while (!(PINA & 0x01))
	{
		// do nothing as long as echo line is low
	}

	n = 0;	//re-use our dummy variable for counting

	while (PINA & 0x01)
	{
		n += 1;		// add 1 to n as long as PortA pin 0 is high
	}

	//when we get here, the falling edge has occured

	return n;

}

int get_rSonar()
{

	int n = 0;

	PORTD = (PIND & 0x7F);	// these two lines create a rising edge
	PORTD = (PIND | 0x80);	// on PortA pin 7

	//lcd_clear();
	//lcd_string("rising edge done");

	while (n < 1)
	{
		//waste enough clock cycles for at least 10us to pass
		n += 1;
		n++;
		lcd_clear();
		lcd_int(n);
	}

	PORTD = (PIND & 0x7F);	// force PortA pin 7 low to create a falling edge
							// this sends out the trigger

	while (!(PIND & 0x01))
	{
		// do nothing as long as echo line is low
	}
```

```
        n = 0;      //re-use our dummy variable for counting

        while (PIND & 0x01)
        {
                n += 1;                 // add 1 to n as long as PortA pin 0 is high
        }

        //when we get here, the falling edge has occured

        return n;

}


int main (void)
{

  int anval=177, anch=0;
  int analogLow = 0;
  int analogHigh = 1023;
  int anRval=1023;
  int counter= 0;
  int zircon_cds = 0;
  int lift_cds = 0;
        int lift_counter = 0;
        int pattern = 1;
        int lift_light=0;
        int lift_dark=0;
        int temp;

        BUMP_DDR= 0x00; // Set bump switches to inputs
        BUMP = 0xFF; // Enables internal pull up resistors of port

  config_adc(); // setup ADC converter

    anval=analog(anch);
          analogLow = ADCL; // read ACD low register
          analogHigh = ADCH; // read ACD high register
    anRval = analogLow | (analogHigh << 8); //place



  DDRC = 0xFF;                          // set portC to output  (could also use DDRC = 0b11111111)



  lcd_init();     // set lcd in 4 bit mode, 2-line mode, with cursor on and set to blink

  temp = PINB;                  // read portB, store value to temp
   PORTB = !(temp);

        long l_SonarVal = 0; long r_SonarVal = 0; long i = 0; double ratio=0;

        lcd_string("SkyRiser");   // if your LCD is wired up correctly, you will see this text
                                        // on it when you power up your Micro-controller board.
                for (i = 0; i < 1000; i++)
                        {
                                lcd_delay();  //delay to read LCD (humans reading)
                        }
        lcd_clear();
        lcd_string("Mobile Construction");
```

```
		for (i = 0; i < 1000; i++)
			{
				lcd_delay();  //delay to read LCD (humans reading)
			}

	init_timer();



	anval=analog(anch);
	  analogLow = ADCL; // read ACD low register
	  analogHigh = ADCH; // read ACD high register
anRval = analogLow | (analogHigh << 8); //place ACD value into one variable



		while((BUMP_PIN & 0x01) == 1)
		{
			lcd_clear();
			lcd_string("Drilling");
			drill_in(300);
		}
		while((BUMP_PIN & 0x01) == 0)
		{
			drill_out(100);
		}
		drill_out(300);
		for (i = 0; i < 500; i++)
			{
				lcd_delay();  //delay to read LCD (humans reading)
			}
		stop_special();

		 temp = PINB;			// read portB, store value to temp
	PORTB = !(temp);
		int timer=0;

		while(timer <1)
		{
			if((BUMP_PIN & 0x01) == 0 || (BUMP_PIN & 0x02) == 0 || (BUMP_PIN & 0x04) == 0)
			{
			timer=1;
			}
			lift_up(400);
		}
		stop_special();
		lcd_clear();
		lcd_string("Calibrating Sequence"); // Initial Calibration Sequence
		lift_light = ad_readn(1,50);
		for (i = 0; i < 200; i++)
			{
				lcd_delay();  //delay to read LCD (humans reading)
			}
		lcd_clear();
		lcd_string("Open Light= ");
		lcd_int(lift_light);
		for (i = 0; i < 1000; i++)
			{
				lcd_delay();  //delay to read LCD (humans reading)
			}
		lift_cds=ad_readn(1,15);
```

```
int darkest = 0;
while(lift_cds<(lift_light*2))
{
	lift_down(400);
	if(darkest<lift_cds)
	{
		darkest = lift_cds;
	}
	lift_cds=ad_readn(1,15);
}
while(lift_cds>(lift_light*1.35))
{
	lift_down(400);
	if(darkest<lift_cds)
	{
		darkest = lift_cds;
	}
	lift_cds=ad_readn(1,15);
}
lift_dark=darkest;

stop_special();
lcd_clear();
lcd_string("Calibrating Dark");
for (i = 0; i < 1000; i++)
	{
		lcd_delay();  //delay to read LCD (humans reading)
	}
lcd_clear();
lcd_string("Closed = ");
lcd_int(lift_dark);
for (i = 0; i < 1000; i++)
	{
		lcd_delay();  //delay to read LCD (humans reading)
	}


lift_cds = ad_readn(1,10);
while (lift_cds < 15)
{
lift_up(400);
lift_cds = ad_readn(1,15);
}
```

```
int r_sonarset=185;
int l_sonarset=190;

while(1)


{

  anval=analog(0);
	analogLow = ADCL; // read ACD low register
	analogHigh = ADCH; // read ACD high register
  anRval = analogLow | (analogHigh << 8); //place ACD value into one variable
```

```
zircon_cds = ad_readn(0,15);
        lift_cds = ad_readn(1,15);



                l_SonarVal = get_lSonar();
                r_SonarVal = get_rSonar();
                ratio = l_SonarVal/200;



                if(zircon_cds < 20)  // Stud is located
                {
                stop_special();
                stop();
                lcd_clear();
                lcd_string("Stopping");

                        while (zircon_cds < 20)  // Moves past the stud
                        {
                                l_SonarVal = get_lSonar();
                                forward(20);
                                for (i = 0; i < 1; i++)
                                {
                                        lcd_delay();
                                }
                                if(l_SonarVal < (l_sonarset+5))
                                {
                                        softL_turn(20);
                                        for (i = 0; i < 1; i++)
                                        {
                                                lcd_delay();
                                        }
                                }

                                zircon_cds = ad_readn(0,25);
                        }
                        forward(30);
                        for (i = 0; i < 750; i++)
                                {
                                        lcd_delay();  //delay to read LCD (humans reading)
                                }

                        stop();
                        lift_cds = 0;

                if(pattern ==2)                         //Begins drill pattern cycle
                {
                lift_counter = 0;
                while(lift_counter < 2)
                        {

                        while(lift_cds < (lift_dark/1.41))
                        {
                                lift_up(400);
                                lift_cds = ad_readn(1,15);
                        }
                        while(lift_cds > (lift_light*1.41))
                        {
                                lift_up(400);
                                lift_cds = ad_readn(1,15);
                        }
```

```
		while(lift_cds < (lift_dark/1.41))
		{
			lift_up(400);
			lift_cds = ad_readn(1,15);
		}
		while((BUMP_PIN & 0x01) == 1)
		{
			lcd_clear();
			lcd_string("Drilling");
			drill_out(300);
		}
		while((BUMP_PIN & 0x01) == 0)
		{
			lcd_clear();
			lcd_string("Drilling");
			drill_in(300);
		}
		drill_in(300);
		for (i = 0; i < 500; i++)
		{
			lcd_delay();  //delay to read LCD (humans reading)
		}
		stop_special();
		while((BUMP_PIN & 0x01) == 1)
		{
			lcd_clear();
			lcd_string("Drilling");
			drill_in(300);

		}
		while((BUMP_PIN & 0x01) == 0)
		{
			lcd_clear();
			lcd_string("Drilling");
			drill_out(300);
		}
		drill_out(300);
		for (i = 0; i < 500; i++)
		{
			lcd_delay();  //delay to read LCD (humans reading)
		}
		if (lift_counter == 0)
		{
		while(lift_cds > (lift_light*1.41))
		{
			lift_up(400);
			lift_cds = ad_readn(1,15);
		}
		}
		lift_counter++;
}
pattern = 1;
}

else if(pattern ==1)
{
lift_counter = 0;
while(lift_counter < 2)
		{

		while(lift_cds < (lift_dark/1.41))
```

```
		{
			lift_up(400);
			lift_cds = ad_readn(1,15);
		}
		while((BUMP_PIN & 0x01) == 1)
		{
			lcd_clear();
			lcd_string("Drilling");
			drill_out(300);
		}
		while((BUMP_PIN & 0x01) == 0)
		{
			lcd_clear();
			lcd_string("Drilling");
			drill_in(300);
		}
		drill_in(300);
		for (i = 0; i < 500; i++)
		{
			lcd_delay();  //delay to read LCD (humans reading)
		}
		stop_special();
		while((BUMP_PIN & 0x01) == 1)
		{
			lcd_clear();
			lcd_string("Drilling");
			drill_in(300);

		}
		while((BUMP_PIN & 0x01) == 0)
		{
			lcd_clear();
			lcd_string("Drilling");
			drill_out(300);

		}
		drill_out(300);
		for (i = 0; i < 500; i++)
		{
			lcd_delay();  //delay to read LCD (humans reading)
		}
		if (lift_counter == 0)
		{
		while(lift_cds > (lift_light*1.41))
		{
			lift_up(400);
			lift_cds = ad_readn(1,15);
		}
		while(lift_cds < (lift_dark/1.41))
		{
			lift_up(400);
			lift_cds = ad_readn(1,15);
		}
		while(lift_cds > (lift_light*1.2))
		{
			lift_up(400);
			lift_cds = ad_readn(1,15);
		}
		}
		lift_counter++;
}
pattern = 2;
```

```
}


        lcd_clear();
        lcd_string("Resetting");
while(lift_cds>11)
                {
                        lift_down(400);
                        lift_cds = ad_readn(1,15);
                }
                stop_special();
                lcd_clear();
                lcd_string("Calibrating...");
                r_sonarset= get_rSonar();
                l_sonarset=get_lSonar();
                for (i = 0; i < 2000; i++)
                        {
                                lcd_delay();  //delay to read LCD (humans reading)
                        }

}

else                    //wall following behavior
{
stop_special();
ratio = l_SonarVal/(l_sonarset);

if(counter == 0)
{

if(l_SonarVal >= 185 && r_SonarVal >=185 && l_SonarVal < 210 && r_SonarVal <210) //one time
calibration
{
                stop();

                lift_cds = ad_readn(1,9);
                lcd_clear();
                lcd_string("Lowering");
                lift_cds = ad_readn(1,15);
                while (lift_cds > 11)
                {
                lift_down(400);
                lift_cds = ad_readn(1,15);
                        }

                stop_special();
                lcd_clear();
                r_sonarset=r_SonarVal;
                l_sonarset=l_SonarVal;
                lcd_string("Calibrating...");
                for (i = 0; i < 2000; i++)
                        {
                                lcd_delay();  //delay to read LCD (humans reading)
                        }

                counter++;
        }
}
if (l_SonarVal == (l_sonarset) && r_SonarVal ==         (r_sonarset))
        {
                forward(35);
```

```
                                        lcd_clear();
                                        lcd_string("perfect");

                                        for (i = 0; i < 1; i++)
                                        {
                                                lcd_delay();  //delay to read LCD (humans reading)
                                        }
                                }


                        else if(l_SonarVal <= 700 && r_SonarVal <= 700 && l_SonarVal >= (l_sonarset+2) && r_SonarVal >=
                (r_sonarset+2) )
                        {

                                variableR_turn(15,ratio/2);
                                lcd_clear();
                                lcd_string("Clear");

                                for (i = 0; i < 1; i++)
                                {
                                        lcd_delay();  //delay to read LCD (humans reading)
                                }
                        }

                        else if(l_SonarVal < (l_sonarset))
                        {
                                softL_turn(10);
                                lcd_clear();
                                lcd_string("Wall here");

                                for (i = 0; i < 1; i++)
                                {
                                        lcd_delay();  //delay to read LCD (humans reading)
                                }
                        }

                        else if ( l_SonarVal >=l_sonarset && r_SonarVal <=r_sonarset)
                        {
                                softR_turn(15);
                                for (i = 0; i < 1; i++)
                                {
                                        lcd_delay();  //delay to read LCD (humans reading)
                                }

                        }

                        else
                        {
                                forward(10);
                                for (i = 0; i < 1; i++)
                                {
                                        lcd_delay();  //delay to read LCD (humans reading)
                                }
                        }

                }

                }

}
```