University of Florida

Dept. of Electrical and Computer Engineering

EEL5666C

Intelligent Machine Design Lab

RIDLAR
(Reliable Intelligent Domino Laying Autonomous Robot)

First Written Report

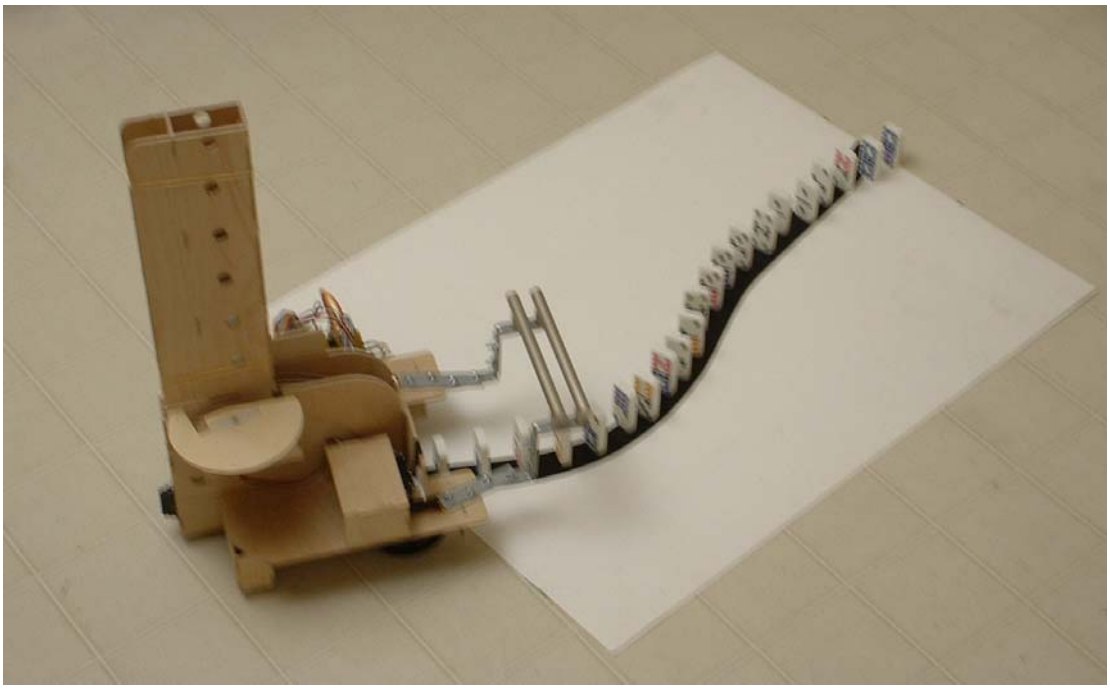
Matthew Yoder

April 19, 2004

Table of Contents

Abstract

RIDLAR is a domino laying, line-following, autonomous robot. It does just what its name says it does. It follows a line on the ground and lays dominos along it with the purpose of knocking them over.



Executive Summary

RIDLAR uses the four pair photorefelctor line-tracking unit to follow a high-contrast line. Along the way it uses two servos in coordination to lay dominos on the line. RIDLAR uses a CdS cell to determine whether or not a domino has entered the slot. When a domino is not detected it outputs an error message to the LCD display. A domino not entering the slot can happen because the domino doesn't get pushed all the way out of the tower, because it gets stuck partway down the slide, or if it is just out of dominoes. Finally, RIDLAR has an infrared range finder mounted on its nose to recognize the first domino it laid in the loop of dominoes. When

the IR reaches its threshold it attempts to go knock it over and then outputs a success message on the LCD display. During all of this process, RIDLAR outputs its state on the LCD display.

Introduction

As a child I loved to play with my Domino Rally toy set. The makers of Domino rally offered a mechanical domino layer that was very unreliable and would only go in a straight line. Also, it could only lay 20 dominos before it had to be refilled. The goal with RIDLAR is to have the most reliability and to be able to follow different shaped lines. Kids will love RIDLAR. All they have to do is load dominoes into the tower, insert the tower and flip a switch. It will stop itself when it encounters problems and wait for help. It runs on 15-minute rechargeable batteries so there are no cords to plug into the wall. After only 15 minutes of recharging it is ready for hours of domino laying fun.

Integrated System

Figure 1 shows the overall electronics system that RIDLAR utilizes. The batteries run to the power/header board where the voltage is regulated down to 5V and distributed to all of the sensors, switches, and servos. All sensors, servos, and switches connect directly to the power/header board (PCB of the board shown in Figure 2). The board supplies power and ground to the peripherals and routes the signal pins to another header for a wire that goes to the MAVRIC board. Perhaps, in another version of RIDLAR, I will design a board that doesn't require single wires for signals and just plugs directly into the headers on the MAVRIC.

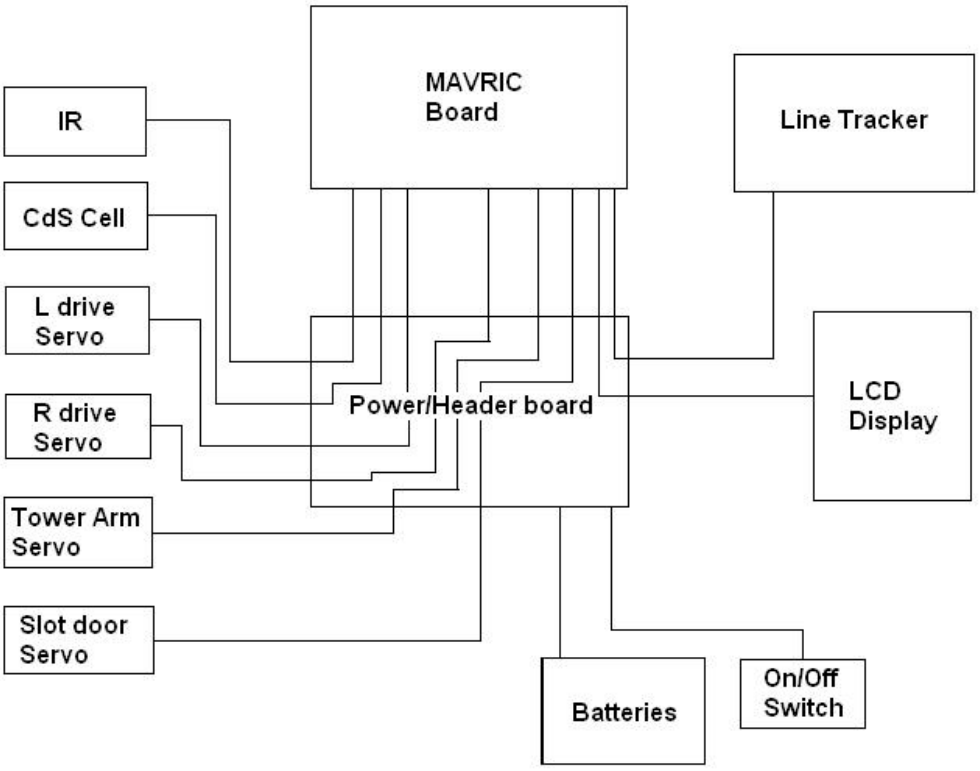
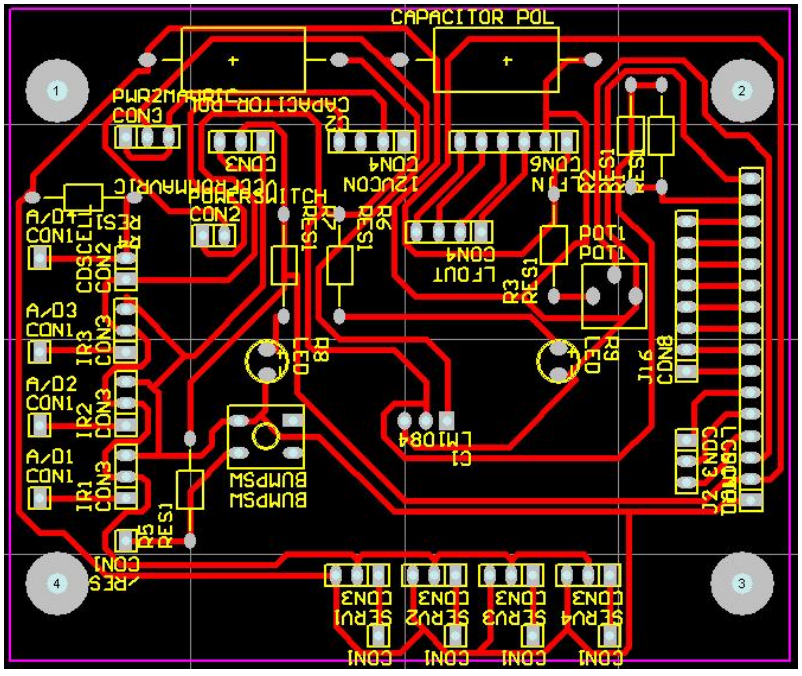


Figure 1. Overall electronics system diagram.

Figure 2 – Header/Power board PCB →



The MAVRIC board was a little overkill for this project. I could have saved a few dollars and had a smaller board from the same place. The MAVRIC II has a smaller footprint and has

everything but the extra 64K of EEPROM that I never used. However I probably wouldn't have been able to fit all of my stuff on the header board and used the same standoff-hole dimensions if I had the MAVRIC II. The scale in Figure 2 is one ince/square.

Mobile Platform

The platform is the critical component of RIDLAR. Without a precisely engineered platform, RIDLAR wouldn't lay dominos. Each component is strategically placed so that the entire sequence of actuation places a domino on the ground.

RIDLAR's dimensions are roughly 15" x 10" x 19". The base has a large footprint and basically holds all the pieces together. The pieces that house the drive servos fit right into the base. The case for the slot door servo sits on top of the base. The slide fits right into the base as well. A more in depth report on the domino laying aspect of the platform is in my special system report. I attached some cabinet hardware to the back end to counter the wait of the stack. I was having problems with weight distribution. The hardware not only looks cool and adds a counter weight but makes a nice handle for picking RIDLAR up and carrying him.

Actuation

Servos provide all of the actuation on RIDLAR. Two hacked servos provide the drive and steering. I reversed the motor leads on one of them so that I didn't have to worry about different output compare register values for the same direction. One servo (the tower arm) pushes the domino from the tower of dominos into the slot. Another servo (the slot door) will open up the slot when the domino is stable. The LCD display provides feedback about its state. I don't

recommend the LCD display that I bought for this application. See the appendix for documentation. It has four lines and I only ever used one.

Sensors

RIDLAR makes use of three different sensors. The most important is the line tracker. I copied Williams four-pair design, but instead I used four hamamatsu P5587 photo reflectors. See the appendix for the data sheet and price. Figure 3 shows the PCB of the line follower. The LED's were removed from the final circuit since they created a voltage divider with the digital output. I didn't realize when I designed it that the output wasn't driven by a transistor or buffer of some sort. If you want to have LED's on your line follower circuit, make them active low to the Vout. To do this basically copy the circuit as shown and just flip the LED's around and connect the cathode (positive) to 5V. The scale in Figure 3 is one inch per square just like Figure 2.

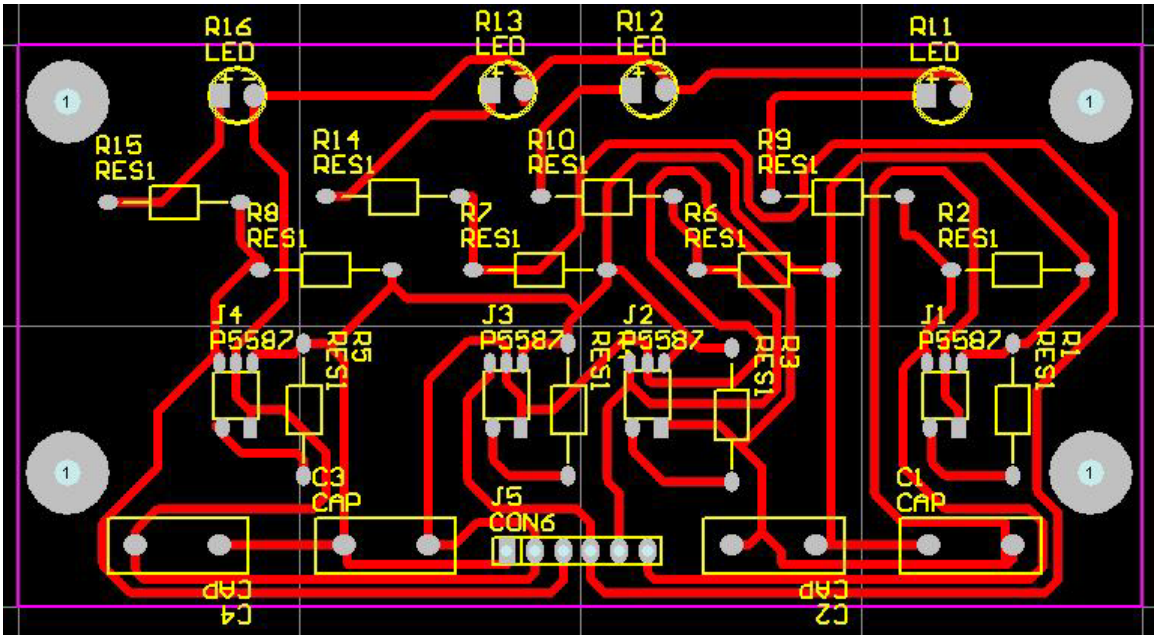


Figure 3 - Line tracker PCB

The CdS cell recognizes the presence of a domino in the slot. The slot door has a hole in it that lines up with the CdS cell, so if there is no domino there will be light on it. I chose a 200kΩ

resistor for the voltage divider. I tested it in several lighting conditions and found that I could get roughly 3-4V when a domino is in the slot and less than 1V when there is nothing in the slot. Resistance in the CdS cell varied from roughly 50kΩ to 500kΩ. During one of the really low-lighting condition test it was 700kΩ. The CdS cell was one of the ones from the lab so I'm not sure if there is any specific part number for it.

Finally, the IR senses when there is an object in close proximity. In this case the only object that it should see is another domino. When there is a domino directly in its path it should attempt to go knock it over. I used the typical Sharp GP2D12 with analog output. See the appendix for a link to the data sheet.

Behaviors

RIDLAR utilizes an array of very simple behaviors to comprise what appears as one complex behavior. RIDLAR cannot behave the same on sharp turns as he can on more gradual turns. Dominos need to be close together around turns in order for them to knock each other over.

Table 1 is a list of different conditions and the behaviors associated with them.

Sensor Conditions	**Behaviors**
Relatively straight line	Normal Line following and domino laying
Gradual turning line	Gradually turn with the line and lay domino
Sharply turning line	Continue line following and lay dominos closer together. No sudden rotations or RIDLAR will accidentally knock over dominos behind it.
No line detected	Continue domino laying in a straight line
IR detects domino	Stop domino laying and go knock it over
CdS cell detects that there is no domino after the tower arm moved.	Stop everything, output an error message and wait for reset.

Table 1 – Sensor conditions and the associated behaviors

RIDLAR automatically calibrates the CdS cell upon startup. It does this by measuring the voltage through on of the ADC pins. That should be about what the voltage should be when there is not a domino in the slot. I then make the threshold that value minus 100 to determine when there are errors. This allows for roughly 10 percent of error in each direction from the original measured value during calibration.

Programming RIDLAR wasn't very complicated. See the appendix for the full code. I used the GNU C compiler with Programmer's Notepad.

Experimental Layout and Results

I didn't do very much structured experimentation other then making small code changes and seeing if it did what I wanted. I experimented with different lighting conditions on the CdS cell to determine the necessary biasing resistor for the voltage divider. I did this by varying both the position of the slot door, and the presence of a domino in different lighting conditions and recording the resistance in the CdS cell. As a result I decided that a 200kΩ resistor would be the best for the situation.

Conclusion

RIDLAR ended up doing everything that I wanted it to do. There are a few things that I would like to have done differently if I had it to do over. First of which, I would put the stack closer to the middle of the robot centered between the wheels. I had many problems with weight distribution since the wood is sometimes slightly warped and one wheels will have less pressure to the ground than the other. Secondly, I would design a more friendly power board. The power board's pins didn't line up with the pins on the MAVRIC board creating a large mess of wires. This is due partly to bad planning and lack of board space. I would also add some extra power

jacks so that I could test things on the breadboard without an extra power supply. Also, I would add a voice chip so that it would be more kid friendly. It would tell them what to do when it encountered an error. Finally, I would add more photo reflectors to my line follower and space them all out evenly. Since RIDLAR can only move in discrete lengths it needs to know more precisely how far off the line he is so that he know how sharply to turn. The four-pair design is great for continuous line following where it can sample several times per second, but when you can only sample at discrete lengths a more accurate reading is necessary.

Documentation

Atmel Atmega128 - http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf
Atmel ISP - http://www.atmel.com/dyn/resources/prod_documents/doc2492.pdf
Sharp GP2D12 IR module - http://www.junun.org/MarkIII/datasheets/GP2D12_15.pdf
Hamamatsu P5587 - http://www.junun.org/MarkIII/datasheets/P5587.pdf
BDMicro MAVRIC - http://www.bdmicro.com/images/mavric.pdf
LCD display - http://www.junun.org/MarkIII/datasheets/20400LED.pdf (display)
http://www.junun.org/MarkIII/datasheets/HD44780u.pdf (controller)
Dominoes - http://www.gamepreserve.com/store/catalog-list.cfm?Cat_Display=14#11172
GWS Servos - http://www.junun.org/MarkIII/Info.jsp?item=18 (not hacked)
Futaba Servos - http://www.servocity.com/html/s148_standard_precision.html (not hacked)
Ball casters - http://www.pololu.com/products/misc/0174/
National Semiconductor - http://www.national.com/pf/LM/LM1084.html#Datasheet
Voltage Regulator

Appendices

Appendix A: Price list

Manufacturer	Part number	Part description (Qty)	Price
BDMicro	MAVRIC	Atmega128 development board (1)	$115.00
Atmel	AVR-ISP	In System Programmer (1)	$ 29.00
Sharp	GP2D12	IR range finder (1)	$ 8.25
Hitachi	HD44780	20x4 LCD display w/backlight (1)	$ 16.00
Junun	None	3pin cable for IR sensor (1)	$ 1.10
Junun	None	Injection molded wheels (2)	$ 6.00
GWS	S03N 2BB	Standard BB servo (2)	$ 10.50
The Game Preserve	11172	Double 9 Dominos set of 55 (2)	$ 19.90
Futaba	S148	Standard Servo (2)	$ 29.90
Pololu	0174	Set of 3 plastic ball casters (1)	$ 1.95

Hamamatsu	P5587	IR photo reflector (4)	$ 8.00
National Semiconductor	LM1085	Voltage Regulator (1)	$ 2.00
Various	Various	Resistors, Capacitors, POT, wires, wood, copper clad board, headers, machine screws, nuts, switches, etc.	Free (provided by the lab)

Total of $247.60 (This does not include parts that were ordered and either were broken or didn't make it to the final design.)

Appendix B: Source Code

```
//RIDLAR's source code
//I'm not the best programmer in the world so my code may not be easy to follow
//I'll try to add comments in where I can.
#include <avr/io.h>
#include <avr/delay.h>
#include <adc.c>//provided by BDMicro  handles A/D conversions
#include <avr/signal.h>
#include <avr/interrupt.h>
#include <inttypes.h>

volatile uint16_t ms_count;

/*
 * ms_sleep() - delay for specified number of milliseconds
 */
void ms_sleep(uint16_t ms)
{
  TCNT0  = 0;
  ms_count = 0;
  while (ms_count != ms)
    ;
}


/*
 * millisecond counter interrupt vector
 */
SIGNAL(SIG_OUTPUT_COMPARE0)
{
  ms_count++;
}


/*
 * initialize timer 0 to use the real time clock crystal connected to
```

```
 * TOSC1 and TOSC2 to generate a near 1 ms interrupt source
 */
void init_timer(void)
{
 /*
  * Initialize timer0 to use the 32.768 kHz real-time clock crystal
  * attached to TOSC1 & 2.  Enable output compare interrupt and set
  * the output compare register to 32 which will cause an interrupt
  * to be generated every 0.9765625 milliseconds - close enough to a
  * millisecond.
  */
 TIFR  |= BV(OCIE0)|BV(TOIE0);
 TIMSK |= BV(OCIE0);    /* enable output compare interrupt */
 TIMSK &= ~BV(TOIE0);   /* disable overflow interrupt */
 ASSR  |= BV(AS0);      /* use asynchronous clock source */
 TCNT0  = 0;
 OCR0   = 32;           /* match in 0.9765625 ms */
 TCCR0  = BV(WGM01) | BV(CS00); /* CTC, no prescale */
 while (ASSR & 0x07)
   ;
 TIFR  |= BV(OCIE0)|BV(TOIE0);
}

void lcd_init(void)
{
        //initialize lcd display
        //step 1------
        PORTD = 0x38;
        PORTC = 4;
        ms_sleep(1);
        PORTC = 0;
        ms_sleep(1);
        //step 2------
        PORTD = 0x0E;
        PORTC = 4;
        ms_sleep(1);
        PORTC = 0;
        ms_sleep(1);
        //step 3------
        PORTD = 0x06;
        PORTC = 4;
        ms_sleep(1);
        PORTC = 0;
        ms_sleep(1);
}
```

```
void lcd_char_out(char c)
{
	PORTC = 5;
	PORTD = c;
	ms_sleep(1);
	PORTC = 1;
	ms_sleep(1);
}

void lcd_backspace(void)
{
	PORTC = 4;
	PORTD = 0x02;
	ms_sleep(1);
	PORTC = 0;
	ms_sleep(1);
}

void stopMotors()
{
	OCR1B = 1420;
	OCR1C = 1410;
}

void successMessage()
{
	lcd_char_out('A');
	lcd_char_out('l');
	lcd_char_out('l');
	lcd_char_out(' ');
	lcd_char_out('D');
	lcd_char_out('o');
	lcd_char_out('n');
	lcd_char_out('e');
	for(unsigned char i = 0; i<6; i++)
	{
		lcd_backspace();
	}
}

void errorMessage()
{
	lcd_char_out('E');
	lcd_char_out('r');
	lcd_char_out('r');
	lcd_char_out('o');
```

```
	lcd_char_out('r');
	for(unsigned char i = 0; i<3; i++)
	{
		lcd_backspace();
	}
}

unsigned char lineTrack(void)
{
	static unsigned char prevDir = 0;

	unsigned char LT_PORT;
	LT_PORT = PINA & 0x0F;

	if (LT_PORT != prevDir)
	{
		switch (LT_PORT)
		{
			case 0x0F:	// 1 11 1
			{
				//left Mforward, right Mforward
				OCR1B = 1500;
				OCR1C = 1500;
				ms_sleep(250);
				break;
			}
			case 0x0E:	// 1 11 0
			{
				//left MFforward, right sreverse
				OCR1B = 1900;
				OCR1C = 1390;
				ms_sleep(460);
				break;
			}
			case 0x0D:	// 1 10 1
			{
				//left MFforward, right Sforward
				OCR1B = 1800;
				OCR1C = 1440;
				ms_sleep(330);
				break;
			}
			case 0x0C:	// 1 10 0
			{
				//left MFforward, right Sreverse
				OCR1B = 1900;
```

```
            OCR1C = 1390;
            ms_sleep(460);
            break;
    }
    case 0x0B:      // 1 01 1
    {
            //left Sforward, right MFforward
            OCR1B = 1450;
            OCR1C = 1800;
            ms_sleep(330);
            break;
    }
    case 0x0A:      // 1 01 0
    {
            //left Sforward, right MFforward
            OCR1B = 1450;
            OCR1C = 1800;
            ms_sleep(330);
            break;
    }
    case 0x09:      // 1 00 1
    {
            //left MFforward, right MFforward
            OCR1B = 1800;
            OCR1C = 1800;
            ms_sleep(220);
            break;
    }
    case 0x08:      // 1 00 0
    {
            //left MFforward, right Sreverse
            OCR1B = 1900;
            OCR1C = 1390;
            ms_sleep(460);
            break;
    }
    case 0x07:      // 0 11 1
    {
            //left Sreverse, right MFforward
            OCR1B = 1395;
            OCR1C = 1800;
            ms_sleep(420);
            break;
    }
    case 0x06:      // 0 11 0
    {
```

```
        //left MFforward, right Sreverse
        OCR1B = 1900;
        OCR1C = 1390;
        ms_sleep(460);
        break;
}
case 0x05:      // 0 10 1
{
        //left MFforward, right sforward
        OCR1B = 1800;
        OCR1C = 1440;
        ms_sleep(330);
        break;
}
case 0x04:      // 0 10 0
{
        //left MFforward, right Sforward
        OCR1B = 1800;
        OCR1C = 1440;
        ms_sleep(330);
        break;
}
case 0x03:      // 0 01 1
{
        //left Sforward, right MFforward
        OCR1B = 1450;
        OCR1C = 1800;
        ms_sleep(330);
        break;
}
case 0x02:      // 0 01 0
{
        //left stop, right MFforward
        OCR1B = 1420;
        OCR1C = 1800;
        ms_sleep(380);
        break;
}
case 0x01:      // 0 00 1
{
        //left Sforward, right MFforward
        OCR1B = 1450;
        OCR1C = 1800;
        ms_sleep(330);
        break;
}
```

```
                        case 0x00:      // 0 00 0
                        {
                                //left MFforward, right Sreverse
                                OCR1B = 1900;
                                OCR1C = 1390;
                                ms_sleep(460);
                                break;
                        }
                }
        }
        //if (LT_PORT == 0xF && prevDir != 0xF)
                //prevDir = LT_PORT;

        return LT_PORT;
}

int main (void)
{
        init_timer();

        /* enable interrupts */
        sei();
        //variables for A/D conversion
        uint16_t ir;
        uint16_t CdS;
        uint16_t CdSCalibration;

        uint8_t random;

        /* initialize A/D Converter */
        adc_init();

        unsigned char counter;//not used
        //set PORTB for output
        DDRB = 0XFF;
        //set PORTC for output
        DDRC = 0xFF;
        //set PORTD for output
        DDRD = 0xFF;
        //set PORTE for output
        DDRE = 0XFF;

        //lcd_init();
        //lcd_backspace();
        //lcd_char_out('n');
```

```
//TIMER 1 INITIALIZATION
TCCR1A = 0xA8;
TCCR1B = 0x12;
TCNT1H = 0x00;
TCNT1L = 0x00;
ICR1H = 0x4A;
ICR1L = 0xAA;
OCR1AH = 0x00;
OCR1AL = 0x00;
OCR1BH = 0x00;
OCR1BL = 0x00;
OCR1CH = 0x00;
OCR1CL = 0x00;

//TIMER 3 INITIALIZATION
TCCR3A = 0xA8;
TCCR3B = 0x12;
TCNT3H = 0x00;
TCNT3L = 0x00;
ICR3H = 0x4A;
ICR3L = 0xAA;
OCR3AH = 0x00;
OCR3AL = 0x00;
OCR3BH = 0x00;
OCR3BL = 0x00;
OCR3CH = 0x00;
OCR3CL = 0x00;

ms_sleep(40);
lcd_init();

lcd_backspace();
lcd_char_out(' ');
//servo initialize to default position
//pauses between motions decrease current spikes
OCR3A = 1700;//center tower arm
ms_sleep(250);
stopMotors();//drive servos are stopped
ms_sleep(250);
OCR1A = 725;//close slot door
ms_sleep(1536);
CdSCalibration = adc_readn(0,5);

//Wait until the CdS cell indicates a domino in the slot.
//Wait half a second for the domino to stabilize
```

```
	//Move the two drive servos for set period of time.
	//Make any necessary maneuvers that the pattern dictates (turn slightly/sharply left/right for very short time)

	//Start over


	while (1)
	{
		ms_sleep(512);    /* wait 1/2 second */
		lcd_backspace();
		lcd_char_out('1');
		OCR3A = 750;//Move tower arm servo (moves the domino into the slot)
		ms_sleep(1536);    /* wait 1.5 seconds */
		//check for domino
		CdS = adc_readn(0,5);
		if (CdS > CdSCalibration - 100)//if no domino, send error message and wait for reset
		{
			lcd_backspace();
			errorMessage();
			OCR3A = 1800;//move tower arm out of the way of the tower
			while (1)
			{
				//wait for reset
			}
		}
		else
		{
			//continue
		}
		OCR1A = 1500;//opens the slot
		OCR3A = 1700;//returns the tower arm back to the middle
		ms_sleep(768);//wait ¾ of a second
		lcd_backspace();
		lcd_char_out('2');
		ir = adc_readn(1,5);
		if (ir > 200)
		{
			OCR1B = 1800;//all ahead full
			OCR1C = 1800;//all ahead full
			ms_sleep(1600);
			stopMotors();
			lcd_backspace();
			successMessage();
```

```
                while(1)
                {
                        //wait for reset
                }
        }
        else
        {//continue as usual//
        }
        //OCR1B = 1420;//LEFT MOTOR STOP (LOOKING FROM REAR)
        //OCR1C = 1410;//RIGHT MOTOR STOP (LOOKING FROM REAR)
        //OCR3A = 1650;//Move the slot servo (releases the domino from the slot)
        OCR1B = 1450;//move short distance so domino isn't so close to the slot
        OCR1C = 1440;
        ms_sleep(150);
        stopMotors();
        ms_sleep(50);
        lcd_backspace();
        lcd_char_out(lineTrack() + 48);//maneuver along line
        //lcd_char_out(lineTrack() + 48);
        //ms_sleep(250);
        stopMotors();
        ms_sleep(512);
        OCR1A = 725;
        lcd_backspace();
        lcd_char_out('4');
        ms_sleep(512);

    }
    return 1;
}
```