ELINEM

An Autonomous Agent That Teaches Kids Colors

University of Florida
Department of Electrical and Computer Engineering

EEL 5666

Intelligent Machines Design Laboratory

Name: Lynette Miller
Date: 08/08/03
TAs: Uriel Rodriguez
Louis Brandy
Vinh Trinh
Instructor: Dr. A. A. Arroyo

Table of Contents

Abstract

The goal of this project was to design an autonomous agent to entertain young children while being educational. ELINEM is an autonomous color teaching robot that randomly moves around in a designated area asking a child to find specific colors. If the child picks up a block with a color that does not correspond with the color requested, the robot will eject the block. If the color of the block matches the color requested, ELINEM will hold onto the block until all the blocks have been collected.

Executive Summary

ELINEM has a playing mat that is black with white edges. It roams freely demonstrating obstacle avoidance while staying within its playing area with IR Emitter/Detectors and photoreflectors. ELINEM interacts with children by speaking with the use of a voice synthesizer. The robot will ask for the child to find a color and will wander and continue asking for the color until the child has chosen a color block and has placed it into the robot. Once the robot has detected that a block has been chosen, it will stop and determine the color of the selected block with the use of LED's and a CdS cell. ELINEM tells the child what the color of the block is and whether or not the choice is correct. If the choice is incorrect the robot ejects the block with the use of a solenoid and asks the child to try again. If the child is correct the robot drops the block into the storage area with an un-hacked servo. This continues until all the colors have been found.

Introduction

Parents enjoy toys in which their child can play with while learning. This robot moves and interacts with children while teaching them colors. In order for the child to learn from their mistakes the robot will tell the child what the correct color of the block is before ejecting the block.
This paper will discuss each of the robot's interlinked systems including the platform and drive systems, computing hardware and electronics, and sensors.

Integrated System

The robot uses a square platform for optimum block storage. This allows the robot to hold more blocks than a rounded platform. Mounted on this are two 12 V gear head motors to drive this system. The brain of ELINEM is an Atmel Mega 128 microcontroller on a letATwork development board. Two Infrared (IR) sensors (Sharp GP2D12) are used to detect the presence of an obstacle in the robot's vicinity. Two photoreflectors (Hamamatsu P5587) are also mounted at the front of the robot to sense if the robot is crossing the boundary it is to remain within. One CdS cell is used for color detection, which uses LEDs that help in this task. A voice synthesizer module (V8699A) creates the robot's voice that will instruct the child to find a color and whether or not they are correct.

Mobile Platform

The robot uses a square platform in order to hold more blocks within a smaller area. The platform will be built in order to support the weight placed on it and small enough to maneuver within a suitable playing area. A two driven wheel and caster design was chosen for its simplicity. The battery is located at the rear of the platform and is supported by a ball caster with the front end supported by the two wheels. The platform was designed in AutoCad and cut from 1/8" balsa wood on the T-Tech machine in the lab.

Actuation

A 7.2Ah 12 V sealed lead acid battery was used as the power source. The robot's drive system has to be powerful enough to handle the weight placed on it by the battery, platform, and any pressure placed on it by the child. These needs could not be met by hacked servos. Instead the robot uses 12 V gear head motors.

Sensors

The three types of sensors used in this robot are: IR detectors, photoreflectors, and CdS cell. The CdS cell is used to detect the color of the blocks put through the robot. Values are read from the CdS cell when each LED is turned on and from this the color can be determined.

Two infrared detectors are used for this robot. They are mounted on the top in the front of the robot. They are mounted facing inwards at 20 degrees in order to detect a larger range. These sensors are the Sharp GP2D12 IR detectors which output an analog voltage relating to the amount of IR light bouncing back from an object. The closer the object the greater amount of IR the detector receives.

Two photoreflectors are mounted on the bottom at the front of the robot. The photoreflectors are mounted of the robot in the front and are used to remain within the playing area. The sensors are the Hamamatsu P5587 photoreflectors. They are surface mounted on boards designed in Protel and milled out on the T-Tech machine in the lab.

Behaviors

ELINEM demonstrates obstacle avoidance by using its infrared sensors. When an object is detected, it will turn either left or right until an object is no longer in its way. After the child puts an incorrect block into the robot, the robot will tell the child what color the block actually is and then eject it by use of a solenoid. If the block is matched correctly then the robot uses a servo to put the block into the storage area located at the front of the robot.

Conclusion

ELINEM moves around within its playing mat avoiding obstacles. While doing this ELINEM asks for the player to find a color. The robot is able to detect the color of the blocks and communicate this to the child. The goal of this project was to build a robot that would teach kids colors. At this end of this project I discovered that children that know their colors can also find this robot fun and entertaining.

Documentation

Atmel Corp., "Atmel AtMega128 Datasheet",

http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

Bergmann-Electronics, "letATwork Manual",

http://www.bergmann-electronics.com/datenblaetter/letatwork/letatwork_man_en.pdf

Appendices

lcd.c

```
//Author: Max Billingsley
/*  PORTB0 - DB4
 * PORTB1 - DB5
 * PORTB2 - DB6
 * PORTB3 - DB7
 * PORTB4 - RS
 * PORTB5 - EN
 * RS: Register Select:
 * 0 - Command Register
 * 1 - Data Register
 */

#include <inttypes.h>
#include <avr/io.h>
#include "lcd2.h"

void lcd_init(void)
{
      lcd_send_command(0x33);
      lcd_send_command(0x32);
      lcd_send_command(0x2c);
      lcd_send_command(0x0f);
      lcd_send_command(0x01);
}
```

```
void lcd_delay(void)
{
	uint16_t time1;

	for (time1 = 0; time1 < 65000; time1++);
		for (time1 = 0; time1 < 65000; time1++);
}

void lcd_send_str(char *s)
{
	while (*s) lcd_send_byte(*s++);
}

void lcd_send_byte(uint8_t val)
{
	uint8_t temp = val;

	val >>= 4;
	val |= 0x10;		/* set data mode */
	PORTA = val;

	lcd_delay();

	PORTA |= ENABLE;
	PORTA &= ~ENABLE;

	temp &= 0x0f;
	temp |= 0x10;		/* set data mode */
	PORTA = temp;

	lcd_delay();

	PORTA |= ENABLE;
	PORTA &= ~ENABLE;

	lcd_delay();
}

void lcd_send_command(uint8_t val)
{
	uint8_t temp = val;

	val >>= 4;
	PORTA = val;
```

```
        lcd_delay();

        PORTA |= ENABLE;
        PORTA &= ~ENABLE;

        temp &= 0x0f;
        PORTA = temp;

        lcd_delay();

        PORTA |= ENABLE;
        PORTA &= ~ENABLE;

        lcd_delay();
 }

#define TO_ASCII(x) ((x) + 0x30)

char * uint8_to_str(uint8_t val, char *str)
 {
        int i;

        for (i = 2; i >= 0; i--) {
                str[i] = TO_ASCII(val % 10);
                val /= 10;
        }

        str[3] = '\0';

        return str;

} /* end of intstr() */
```

Lcd2.h

```
//Author: Max Billingsley

 #define ENABLE 0x20

 void lcd_init(void);
 void lcd_delay(void);
 void lcd_send_str(char *s);
 void lcd_send_byte(uint8_t val);
 void lcd_send_command(uint8_t val);
```

Demo.c

```
#include <inttypes.h>
#include <avr/io.h>
#include "lcd2.h"
#define TO_ASCII(x) ((x) + 0x30)

char * uint8_to_str(uint8_t val, char *str);
void simple_adc_init(void);
void io_init(void);
void pwm_init(void);
void long_wait(void);
void mywait(uint16_t waittime );
uint8_t getad(int channel);
void move_this(void);
void whiteLED(void);
void turn_left(void);
void turn_right(void);
void reverse(void);
void forward(void);
void stop(void);
void cds(void);
int detect(int white, int blue, int green, int orange, int red);
void declare(int color);
void read_color(void);
void wrong(void);
void correct(void);
void askforcolor(void);
void voice_init(void);
void voice_delay(void);
void voice_send_str(char *s);
void voice_send_byte(uint8_t val);

uint8_t analog;
uint8_t analog1;
uint8_t analog2;
uint8_t analog3;
uint8_t analog4;
char output[10];

int channel;
int white, blue, green, orange, red;
int color;
int block = 0;
int request = 1;
int count = 0;
```

```
int main(void)
{
        io_init();
    simple_adc_init();
    lcd_init();
        pwm_init();
        voice_init();
        voice_send_byte(' ');
        voice_send_byte(' ');
        voice_send_byte(' ');
        voice_send_byte(' ');
        long_wait();

        while (1) {
                lcd_send_command(1);
                analog1 = getad(1);
                analog2 = getad(2);
                analog3 = getad(3);
                analog4 = getad(4);
                move_this();

                if(count == 8){
                        askforcolor();
                        count = 0;
                }
                count++;
                if (request == 9){
                        voice_send_str("you have found all the colors,");
                        voice_send_byte(0);
                }
                whiteLED();
                if (block){
                        lcd_send_command(1);
                        stop();
                        read_color();
                        color = detect(white, blue, green, orange, red);
                        declare(color);
                        block = 0;
                }

                mywait(300);
        }//end of while
        return 0;
}//end of main
```

```
void askforcolor(void)
{
	if (request != 9){
		voice_send_str("Find the color,");
		if (request == 1){
			voice_send_str("blue,");
		}
		if (request == 2){
			voice_send_str("orange,");
		}
		if (request == 3){
			voice_send_str("yellow,");
		}
		if (request == 4){
			voice_send_str("purple,");
		}
		if (request == 5){
			voice_send_str("green,");
		}
		if (request == 6){
			voice_send_str("black,");
		}
		if (request == 7){
			voice_send_str("white,");
		}
		if (request == 8){
			voice_send_str("red,");
		}
		voice_send_byte(0);
	}
}
void whiteLED(void)
{
	OCR1C=0x0280; //test
	cbi(PORTC, 6);
	sbi(PORTC, 2);
	mywait(500);
	cds();
	white = analog;
	lcd_send_str("White: ");
	lcd_send_str(output);
	if (white < 80){
		block = 1;
	}
}
void turn_left(void)
```

```
{
	PORTC |= (1 << 0);//DIRECTION - left
	PORTC &= ~(1 << 1);
	PORTB |= (1 << 5);//ENABLE - go
	PORTB |= (1 << 6);

}

void turn_right(void)
{
	PORTC |= (1 << 1);//DIRECTION - right
	PORTC &= ~(1 << 0);
	PORTB |= (1 << 5);//ENABLE - go
	PORTB |= (1 << 6);
}

void reverse(void)
{
	PORTC |= (1 << 0);//DIRECTION - reverse
	PORTC |= (1 << 1);
	PORTB |= (1 << 5);//ENABLE - go
	PORTB |= (1 << 6);
}

void forward(void)
{
	PORTC &= ~(1 << 0);
	PORTC &= ~(1 << 1);
	PORTB |= (1 << 5);//ENABLE - go
	PORTB |= (1 << 6);
}

void stop(void)
{
	PORTB &= ~(1 << 5);//stop
	PORTB &= ~(1 << 6);
}

void move_this(void)
{
		if (((analog1 > 50) && (analog2 > 50))||((analog3 > 50) && (analog4 >
50))) {//CHANGE 3 AND 4 TO WHITE = 255: <<<<<
			lcd_send_str("back up");
			reverse();
			long_wait();
			turn_left();
```

```
                long_wait();
                long_wait();
        }
        else if (analog2 > 50) {//CHANGE 3 TO <  <<<<<<<<<<
                lcd_send_str("left");
                turn_left();
        }
        else if (analog3 > 50) {
                lcd_send_str("back up");
                reverse();
                long_wait();
                lcd_send_str("left");
                turn_left();
        }
        else if (analog1 > 50) {//CHANGE 4 TO <  <<<<<<<<<<
                lcd_send_str("right");
                turn_right();
        }
        else if (analog4 > 50) {
                lcd_send_str("back up");
                reverse();
                long_wait();
                lcd_send_str("right");
                turn_right();
        }
        else if (((analog1 < 51) && (analog2 < 51))||((analog3 < 51) && (analog4
< 51))) {//CHANGE 3 AND 4 TO <  <<<<<<<<<
                lcd_send_str("go");
                forward();
        }
}

uint8_t getad(int channel)
{
        ADMUX &= 0xE0;
        ADMUX |= channel;
        ADCSRA |= (1 << ADSC);
        while (!(ADCSRA & (1 << ADIF)));                    // wait until conversion
complete
    analog = ADCH;
    ADCSRA |= (1 << ADIF);

        uint8_to_str(analog, output);
    //lcd_send_command(1);
        lcd_send_str(output);
        lcd_send_str(" ");
```

```
        return analog;
}
 void voice_send_str(char *s)
 {
    while (*s) voice_send_byte(*s++);
 }

 void voice_send_byte(uint8_t num)
 {
    DDRD = 0x00;
        sbi(PORTE, 7);
        cbi(PORTE, 6);

        voice_delay();
        voice_delay();
        voice_delay();

        sbi(PORTE, 6);
        cbi(PORTE, 7);
        DDRD = 0xff;
   ////////////////////////////////
    PORTD = num;
    sbi(PORTE, 7);

         ////////////////////////
    voice_delay();

 }
void pwm_init(void)
 {
       TCCR1A = 0x0A;
       TCCR1B = 0x12;            // divider = 8;

       ICR1 = 20000;
 }

void simple_adc_init()
 {
    ADMUX |= (1 << ADLAR);                    // left adjust result
    ADCSRA |= (1 << ADEN);                    // enable
    //ADCSRA |= (1 << ADFR);                  // free running
    ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);          // prescaler =
128
 }

void io_init(void)
```

```
{
      DDRA = 0x3f;
                  DDRB = 0xff;
                  DDRC = 0xff;
                  DDRD = 0xFF;
                  DDRE = 0xC0;
                  DDRF = 0x00;
}
void voice_init(void)
{
   //DDRE = 0xff;
         DDRD = 0x00;

         sbi(PORTE, 7);
         cbi(PORTE, 6);

         while (!(PIND & 0x10))
         {

         }

         sbi(PORTE, 6);
         cbi(PORTE, 7);

         DDRD = 0xff;
}


void voice_delay(void)
{
   uint16_t time1;

   for (time1 = 0; time1 < 65000; time1++);
         for (time1 = 0; time1 < 65000; time1++);
}
void mywait(uint16_t waittime )
{
         uint16_t time1, time2, time3;
         for (time1 = 0; time1 < waittime; time1++) {
         for (time2 = 0; time2 < 500; time2++){
         for (time3 = 0; time3 < 50; time3++);
         }}
}
void long_wait(void)
{
      uint16_t time1, time2;
```

```
        for (time1 = 0; time1 < 1000; time1++)
            for (time2 = 0; time2 < 65000; time2++);
}

void wrong(void){
            voice_send_str("Try again,");
            voice_send_byte(0);
            OCR1C=0x01f0; //close
            mywait(2000);
            //OCR1C=0x0280; //test
            //mywait(2000);
            //OCR1C=0x01f0; //close
            //mywait(2000);
            sbi(PORTC, 7);//solenoid on
            mywait(500);
            cbi(PORTC, 7);//solenoid off
}
void correct(void){
            voice_send_str("You are correct,");
            voice_send_byte(0);
            OCR1C=0x0370;//drop block
            long_wait();
            OCR1C=0x01f0; //close
            request++;
}
void declare(int color){
            if (color != 9){
                        voice_send_str("This is the color,");
                        if (color == 1){
                                    lcd_send_str("BLUE");
                                    voice_send_str("blue,");
                        }
                        else if (color == 2){
                                    lcd_send_str("ORANGE");
                                    voice_send_str("orange,");
                        }
                        else if (color == 3){
                                    lcd_send_str("YELLOW");
                                    voice_send_str("yellow,");
                        }
                        else if (color == 4){
                                    lcd_send_str("PURPLE");
                                    voice_send_str("purple,");
                        }
                        else if (color == 5){
```

```
                        lcd_send_str("GREEN");
                        voice_send_str("green,");
                }
                else if (color == 6){
                        lcd_send_str("BLACK");
                        voice_send_str("black,");
                }
                else if (color == 7){
                        lcd_send_str("WHITE");
                        voice_send_str("white,");
                }
                else if (color == 8){
                        lcd_send_str("RED");
                        voice_send_str("red,");
                }
        }
        else if (color == 9){
                lcd_send_str("NOTHING");
        }

        voice_send_byte(0);

        if (request == color){
                correct();
        }
        else {
           if (color != 9){
                        wrong();
                }
        }
}
 void read_color(void) {
        //OCR1C=0x01f0; //shut
        OCR1C=0x0280; //test
        long_wait();
        cbi(PORTC, 6);
        sbi(PORTC, 2);
        mywait(500);
        cds();
        white = analog;
        lcd_send_str("White: ");
        lcd_send_str(output);

        cbi(PORTC, 2);
        sbi(PORTC, 3);
        mywait(500);
```

```
	cds();
	blue = analog;
	lcd_send_str("Blue: ");
	lcd_send_str(output);

	cbi(PORTC, 3);
	sbi(PORTC, 4);
	mywait(500);
	cds();
	green = analog;
	lcd_send_str("Green: ");
	lcd_send_str(output);

	cbi(PORTC, 4);
	sbi(PORTC, 5);
	mywait(500);
	cds();
	orange = analog;
	lcd_send_str("Orange: ");
	lcd_send_str(output);

	cbi(PORTC, 5);
	sbi(PORTC, 6);
	mywait(500);
	cds();
	red = analog;
	lcd_send_str("Red: ");
	lcd_send_str(output);
}
int detect(int white, int blue, int green, int orange, int red)
{

	if ((white + blue + green + orange + red) >= 746){
		return 9;
		}
	if ((white + blue + green + orange + red) >= 525){   //581
		if((green + orange) >= 190){
			return 6;
		}
	}
	if ((white + blue + green + orange + red) >= 365){		//503//420
		if ((white + green + orange + red - blue) >= 160){   //200
			if (green + green + green > 279)
				return 4;
		}
		else if((white + green + orange + red - blue) >= 110){
```

```
                return 8;
            }
        }
        if ((white + blue + green + orange + red) >= 300){          //384//275
            if ((blue + green) >= 220){             //236//163
                if ((green + orange) >= 160){
                    if ((orange + red) >= 80)
                        return 8;
                }
            }
            if ((blue + red - green) >= 110){
      //130
                return 5;
            }
            else if ((orange + red) >= 111) {
                return 1;
            }
        }
            if ((white + blue + green + orange + red) >= 200){          //273

                return 2;
            }
            if ((blue + blue + green) >= 125){
      //blue + green 102          L65
                return 3;
            }
            else {
                return 7;
            }

 }
void cds(void)
{
        ADMUX &= 0xE0;
        ADCSRA |= (1 << ADSC);
        while (!(ADCSRA & (1 << ADIF)));                                    // wait
until conversion complete
        analog = ADCH;
        ADCSRA |= (1 << ADIF);
        uint8_to_str(analog, output);
}
```