Skippy

The Fire Fighting
Robot

Final Report
by
Warren L. Thornton, Jr
EEL 5666
April 25, 2000

Table of Contents

Abstract

Skippy is an autonomous fire fighting agent, with the ability to detect small fires through IR sensors. Once a fire is detected, Skippy will avoid obstacles to reach the fire. Once Skippy has located the fire he will proceed to use a fan attached to his back to eliminate the threat.

Executive Summary

Skippy is an autonomous agent designed to recognize and extinguish flames in his environment. Skippy uses Infrared emitter, Sharp Cams, Infrared phototransistors, micro switches, and a fan to accomplish his goal.

In order to successfully fight fires, Skippy must be able to navigate a room without being hindered by obstacles. To accomplish this, Skippy uses the IR emitters to send out a 40 kHz IR signal that bounces off any objects in his path. The Sharp Cams then detect the signal and produce a proportional voltage that is sent to the microprocessor. This allows Skippy to determine the distance of the object and determine whether to turn or not. In case the obstacle avoidance system fails, the back up obstacle detection system comes into play. This involves ten micro switches (bump switches) that send a voltage to the processor if Skippy run into an object. Skippy then turns in another direction and moves away from the obstacle.

To detect and approach a flame, Skippy uses three IR phototransistors. One is place in the front center and one on either side approximately 60 degrees from the center. Skippy sets up threshold values for each on the sensors at initialization and uses these values along with updated sensor reading to locate the flame. Once the fire is in range Skippy uses his fan to blow out the fire.

This sensor suite along with a program I wrote in ICC11 allowed Skippy to autonomously roam a room, recognize flames, and eliminate the threat.

Introduction

Background

Home fires continues to be a major problem in our modern world. Yearly they cause millions of dollars in damage, emotional trauma, and take far too many lives. Often these fires start small and are quite extinguishable. However, the lack of a proper early warning system due to slackness of the homeowner often allows the preventable to happen. Skippy promises to be the solution to this problem.

Scope

Skippy is intended to be a household agent. He will be stationed in a residence, perhaps making his rounds throughout the night. Once a fire is detected, he will proceed to its location and attempt to extinguish it. In the final version, Skippy will begin to alert the household as he searches for the fire. In the unlikely event that our autonomous agent cannot contain the fire the residents will have ample time to evacuate the home.

To perform the searching function will require infrared sensors and micro switches. Fire detect will require a second type of infrared detector. Alerting the household residents will require the use of an alarm system. The Motorola 6811 micro-controller directs the entire system.

Integrated Systems

Description of System

The key to Skippy is the Motorola 68HC11 micro-controller in conjunction with the ME11 expansion board. These serve as the brains of the system. The ME11 is a Mekatronics product which allow access to 32K of RAM and the ability to easily use output ports. The board is fairly small, but with the 32k RAM it should be quite able to the programming the complex behaviors. The micro-controller also has 8 analog input ports and the ability to expand this number with another circuit. I chose to simply use the eight pins, however, future versions of Skippy may need the additional sensory expansion.

Mobile Platform

Scope

I chose to begin the project with the platform design used by Talrik. I chose this design because I would need a platform large enough to carry all the sensors and also able to the fire extinguishing system.

Specifications

The Talrik platform seemed to be a logical choice because of it tried and test physical form. Also its size allows for upgrades and the ability to mount all the required sensors and fire extinguisher system. I decided to mount the processor on top of the platform rather than the bottom. This allows better access to the I/O ports and helped to balance the platform.

Actuation

Scope

Skippy will be required to travel through a household environment. He will need to have a high degree of freedom. The motors must have enough torque to travel on both carpet and hard surfaces. Skippy will use two servos that are hacked the way that is taught in IMDL to serve as dc motors. This involves removing the internal circuitry that limits the amount of current to the servo. The process also involves removing a stopper on the gear head that constricts the movement from a full 360 degrees.

The two motors and processor are powered by an eight-pack of batteries. I mounted this pack on the bottom rear of the Talrik platform. This put more weigh on the back half of the robot where the caster wheel is located. To power the fan that extinguishes fires, I used a battery four-pack that I also mounted on the back.

Sensors

Scope

Skippy must perform several functions to be an effective weapon against fires. Thus he uses a wide array of sensors to locate and fight fires. To avoid obstacles he will use infrared emitters, IR sensors, and micro-switches. To detect fire he was origanlly to use two types of sensors. First he will us a UV Tron from Hamamatsu to detect flames at a distance. Next he will use photo transistors to pinpoint the flame in the room and target it for extinguishing. In the final project, Skippy only used to the photo-transistors for flame location.

Infrared Sensors and Emitters

In order to fight a fire the robot must be able to get to it with allowing minor obstacles to get in its way. The infrared emitters are actually LED's that project light in the direction the robot is traveling. If an obstacle is in its path, the reflected light will be detect and the robot will turn to avoid the obstacle.

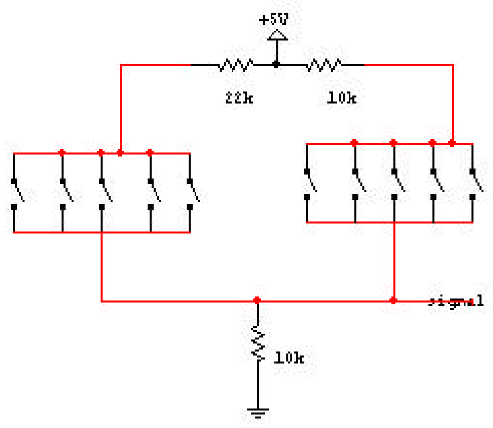
Four IR LED's are used in the sensor suite. Two in the front left and two in the front right. Each group of two are wired in a series combination. Three hacked Sharp Cams are used to detect the IR light reflected off surfaces in the robot’s path. I chose to use a threshold of 117, so that the IR would not be overly sensitive and cause Skippy to move away from the candle base.

Micro-Switches

As a backup system for the infrared, the micro-switches will act a bump detectors. They will be mounted around the perimeter of the robot. If the robot runs into any obstacles the switches are pressed. The 6811 will read the change in voltage and adjust position.

There are actually 10 micro-switches, five in the front and five in the back. The five front switches are wired in parallel, making a continuous front bumper. If any one switch is depressed then its as if all are depressed. The back bumper is constructed in a similar manner. The front and back bumpers are actually in parallel with each other through different resistance's, but into the same analog port of the microprocessor. The front through a 10k resistor and the back through a 22k resistor. Thus the analog port will read one of four values as seen in the table.

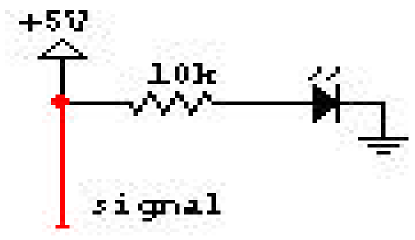
Position of Switches	Analog Value
Open Switch	0
Front Depressed	78-79
Back Bumper Depressed	128-129
Both Bumpers Depressed	156



Flame Detection

In order to locate the fire, Skippy will use some type of buddy system. Heat detectors alone will not accomplish the job. The robot may mistake the heat of a human as that of the fire. Past projects have shown that a photo transistor (an infrared detector of 980 nm) would detect the unique IR signature of flame. This system and a long range UV sensor Skippy should quickly locate the flames. Unfortunately, I was not able to incorporate the UV sensor into the final version of Skippy.

The circuit design I will implement for the photo transistors was created by Scott Jantz. The anode is tied to 5V through a resistor and the cathode is connected to ground. The signal is taken from the anode. If no flame is present, the signal read 255. As the flame get closer, the signal drops. The use of three of these detectors (one in the middle and one 60 degrees off to each side) will allow Skippy to pinpoint a flame in the room. The illustration below shows the circuit and the graph illustrates the distance from flame versus signal.



The long range UV Tron from Hamamatsu sends a 10ms pulse went it senses a flame. The frequency of the pulses increase as the flames nears. I was not able to add this sensor to the final version of Skippy, however, he worked well without it. I do believe that the UV would enhance Skippy's abilities and allow him to locate fires from further away.

Behaviors

Bumper Sensing/Obstacle Detection

In case of a failure in the infrared sight, Skippy will need to know if he has run into an object. Therefore, if he runs into a wall, he will back up and take another sensor reading. He will then turn in the direction that does not have an obstacle in the path and proceed forward.

Obstacle Avoidance

To avoid obstacles is a basic requirement for an autonomous machine. Skippy will randomly move about a room checking his sensors. Skippy is programmed with a two special routines called look left and look right. If he sees an object with his left IR he will jump to his turn right routine until he no longer sees that object, then he proceed forward and again start looking for a flame. If he sees an object with his right IR he will jump to his turn left routine and continue to turn left until he no longer sees an object..

Flame Detection

When Skippy is reset he initializes his IR photo-transistors by taking a reading of the ambient light in a room and making this value his Flame Threshold. As Skippy roams around he continually takes reading from the IR photo transistors. If this reading deviate from the threshold by a significant about he will assume their is a flame and start to move in that general direction.

Flame Search

Once Skippy detects a flame he jumps to a follow the flame routine. This routine compares the reading from the two side sensors with that of the center. If the reading is smaller than the threshold and smaller than the center, then Skippy turn into the direction of the flame and proceeds towards the flame.

Extinguishing Flame

Once the center flame sensor reach a minimum value Skippy knows that he is in extinguishing range. He then turns 180 degrees, so that his fan faces the fire. Then using pin 3 of the D port, Skippy sends a signal that turn the fan on for 5 seconds. Then he sends a signal that turns the fan off and waits for 2 seconds. Finally, Skippy does his victory dance.

Celebration

Skippy's victory dance is actually a series of turns and wiggles. Skippy start's his dance by wiggling which alternates turning left and right 5 times. He then turns 90 degrees and wiggles again. Next, he turns 180 degrees and wiggles again. Finally, he turns 90 degrees so that his fan faces the fire again, ready to find his next fire.

Experimental Layout and Results

After completing the construction of Skippy, I performed several test runs to determine his capabilities. I discovered two major problems in his performance. First, the sensitivity of the flame sensors varied with the type of resistor used. Lower values (less than 1k) picked up the smallest changes in brightness, yet they were extremely flaky. Higher resistor values (47k and up) gave nominal reading of 255, but it was hard to changes in the values when Skippy approached a flame. I went with 22k resistors in each of the circuits, this allowed for adequate sensitivity. I also put black tubing around the sensors to block out as much ambient light as possible.

The second problem involved bright spots in the room due to sunlight through open windows. The sunlight would fool Skippy into thinking that the bright spot in the room was the location of the candle. Fortunately, the sunlight was not bright enough to trigger the extinguisher system, but it cause Skippy to move in the wrong direction. To correct this problem, I would initialize Skippy's sensors in the brightest spot in the room. He would therefore setup thresholds that would not be easily affect by sunlight. Once I corrected those problems Skippy was able to extinguish multiple fires successfully.

Conclusion

Although it was a challenge in designing and building Skippy, I'm glad that I've had the experience. I would like to have incorporated the UV Tron sensor, so that Skippy could detect a flame from across a room. I would have also liked to have used a CO2 extinguishing system, so that Skippy could have eliminated larger fires. Perhaps I'll be able to add those missing pieces in the near future. However, working on Skippy has taught me how to start a project with simply a goal and to see it to completion. I've also been able to put together the knowledge I've gleaned from four years at the University of Florida into and actual working autonomous agent. I would encourage other students to accept the challenge and take IMDL, it's not an easy course, but it will bring you one step closer to being a true engineer.

Documentation

Megatronic. www.megatronic.com Infrared emitters and detectors; servos, ME11 board, and Talrik platform.

IMDL. www.mil.ufl.edu/IMDL. Information on servo and Sharp Cam hack.

Radio Shack. IR Photo-Transistors.

Appendix

Code for Skippy

```
/*************************************************************************
*************
*
*Title:        Skippy
*Programmer: Warren L. Thornton, Jr
*Date:         April 7, 2000
*Version:      1
*
*Description
*
*
*
*Use hyperterm with the following settings:
*
*COM1, 9600 Baud, 8bits, No Parity, 1 Stop Bit, No Flow of control,
*VT100, Wraplines.
*
*
*
```

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

/************************************INCLUDES***********************
*************************/
#include <tjpbase.h>
#include <stdio.h>
#include <motorme.h>
#include <hc11.h>
#include <mil.h>
/*******************************End of
Includes**************************************/

/*********************************Constants***************************
********************/

/*IR emitters */
#define IRE_OUT       *(unsigned char*)(0xffb9)

/*Constants for driving all the 40kHz modulated IR emitters on when load ino
IRE_OUT*/

#define IRE_ALL_ON 0xff
```

```
/*Constants for turning all the 40kHz modulated IR Emitters off when loaded into
IRE_OUT*/

#define IRE_ALL_OFF 0x00

#define F_UPPERLIMIT 80

#define F_LOWERLIMIT 70

#define B_UPPERLIMIT 132

#define B_LOWERLIMIT 125

#define IRThreshold 115

#define Bump 0

#define Right_Eye 2

#define Left_Eye 1

#define Right_Flame 7
```

```
#define Center_Flame 6

#define Left_Flame 5

#define Maxspeed 100

#define Halfspeed 50

#define QuarterSpeed 25


/********************************End of
Constants*************************************/

int rspeed = 0;
int lspeed = 0;
int flame = 0;
int obstacle = 0;
unsigned int left = 0;
unsigned int right = 0;
int flame_on = 0;
int time = 750;
```

```
/****************************************Functions****************************
*************/

void sensor_read(void);
void right_turn(void);
void left_turn(void);
void check_bumper(void);
void long_check_fire(void);
void follow_flame(void);
void short_check_fire(void);
void look_right(void);
void look_left(void);
void forward(void);
void reverse(void);
void stop(void);
void init_flame(void);
void turn_180(void);
void fan(void);
void turn_90(void);
void search(void);
void setdigport(void);
void leftt(void);
void rightt(void);

void shake(void);

void victory_dance(void);


/***********************************End of

Functions*********************************/


int Flame_ThresholdC = 255;

int Flame_ThresholdR = 255;

int Flame_ThresholdL = 255;

int Sensor_Data[8];

int Fire_DataL[10];

int Fire_DataC[10];

int Fire_DataR[10];


void main()

/**************************************MAIN*************************

********/

{

	init_analog();

	printf("init_analog\n");

	init_motorme();

	printf("init_motorme\n");
```

```
        init_clocktjp();

        printf("init_clock\n");

        init_flame();

        setdigport();


        IRE_OUT = IRE_ALL_ON;  /* Turns on all the 40kHz modulated IR emitters */

        wait(300);                          /* Allow IR detectors to reach final values */


/*Start SKIPPY moving forward when back bumper is pressed*/



        sensor_read();


        while(Sensor_Data[Bump] < B_LOWERLIMIT){


                printf("Waiting to start\n");

                write("Bumper Value ");

                write_int(Sensor_Data[Bump]);

                write("Left_Flame: ");

                write_int(Sensor_Data[Left_Flame]);

                write("Center Flame: ");

                write_int(Sensor_Data[Center_Flame]);

                write("Right Flame: ");
                write_int(Sensor_Data[Right_Flame]);

                sensor_read();


  }




        forward();


        while(1){


                check_bumper();

                printf("Checking Bumper\n");

                look_right();

                printf("Looking Right\n");

                look_left();

                printf("LookingLeft\n");

                follow_flame();

                forward();


        }


/*************************************End of

Main*************************************/
```

```
}

void sensor_read()
{       int i;
        for (i=0; i < 8; i++) {
                Sensor_Data[i] = analog(i);

        }

}

void check_bumper()
{
        sensor_read();

        if ((Sensor_Data[Bump] > F_LOWERLIMIT) && (Sensor_Data[Bump] <
F_UPPERLIMIT)) {
                stop();
                wait(500);
                reverse();
                wait(2000);
                look_right();
                look_left();

        }

        if ((Sensor_Data[Bump] > B_LOWERLIMIT) && (Sensor_Data[Bump] <
B_UPPERLIMIT)) {
                stop();
                wait(500);
                forward();
                wait(2000);
                look_right();
                look_left();

        }

}

void look_right()
{
        sensor_read();
        write("Right Eye Sees: ");
        write_int(Sensor_Data[Right_Eye]);
```

```
	if (Sensor_Data[Right_Eye] > IRThreshold) {
		obstacle = 1;
		left=1;
		left_turn();
		printf("Turning Left\n");
	}

	else left = 0;

}

void look_left()
{
	sensor_read();
	write("Left Eye Sees: ");
	write_int(Sensor_Data[Left_Eye]);
	if (Sensor_Data[Left_Eye] > IRThreshold) {
		obstacle = 1;
		right=1;
		right_turn();
		printf("Turning Right\n");
	}
```

```
        else right=0;
}


void right_turn() {
        if(right == 1) {
                rspeed = -Halfspeed;
                lspeed = Halfspeed;
                motorme(RIGHT_MOTOR, rspeed);
                motorme(LEFT_MOTOR, lspeed);
                if (obstacle == 1){
                        sensor_read();
                        while(Sensor_Data[Left_Eye] > IRThreshold){
                                sensor_read();
                                        if(Sensor_Data[Left_Eye] < IRThreshold){
                                                forward();
                                                right = 0;
                                                obstacle = 0;
                                        }
                        }
                }
        }


         if(flame == 1){
            sensor_read();
            while(Sensor_Data[Center_Flame] > Sensor_Data[Right_Flame]){
                sensor_read();
                    if( Sensor_Data[Center_Flame] <
Sensor_Data[Right_Flame]){
                        forward();
                        right = 0;
                        flame = 0;
                    }
            }
        }

}

void left_turn() {
    if (left == 1) {
        rspeed = Halfspeed;
        lspeed = -Halfspeed;
        motorme(RIGHT_MOTOR, rspeed);
        motorme(LEFT_MOTOR, lspeed);
        if(obstacle == 1){
            sensor_read();
            while(Sensor_Data[Right_Eye] > IRThreshold){
                sensor_read();
                    if(Sensor_Data[Right_Eye] < IRThreshold){
                        forward();
                        left = 0;
                        obstacle = 0;
                    }
            }
        }

        if(flame == 1){
            sensor_read();
            while((Sensor_Data[Center_Flame] > Sensor_Data[Left_Flame])
&& (Sensor_Data[Right_Flame] > Sensor_Data[Left_Flame])){
                sensor_read();
                    if((Sensor_Data[Center_Flame] <
Sensor_Data[Left_Flame]) || (Sensor_Data[Right_Flame] < Sensor_Data[Left_Flame])){
                        forward();
                        left = 0;
                        flame = 0;
                    }
                }
        }
    }
```

```
}

void forward() {

        rspeed = Halfspeed;
        lspeed = Halfspeed;
        motorme(RIGHT_MOTOR, rspeed);
        motorme(LEFT_MOTOR, lspeed);

}

void reverse() {
        rspeed = -Halfspeed;
        lspeed = -Halfspeed;
        motorme(RIGHT_MOTOR, rspeed);
        motorme(LEFT_MOTOR, lspeed);

}

void stop() {
        rspeed = 0;
        lspeed = 0;
        motorme(RIGHT_MOTOR, rspeed);
        motorme(LEFT_MOTOR, lspeed);
}

void follow_flame(){

        sensor_read();
        write("Right Flame: ");
        write_int(Sensor_Data[Right_Flame]);
        write("Center Flame: ");
        write_int(Sensor_Data[Center_Flame]);
        write("Left Flame: ");
        write_int(Sensor_Data[Left_Flame]);

        if((Sensor_Data[Right_Flame] < Sensor_Data[Center_Flame]) &&
(Sensor_Data[Right_Flame] < Flame_ThresholdR)){

                right = 1;
                flame = 1;
                /*printf("Following flame right\n");
                wait(1000);*/
                right_turn();
        }
```

```
            if((Sensor_Data[Left_Flame] < Sensor_Data[Center_Flame]) &&
(Sensor_Data[Left_Flame] < Flame_ThresholdL)){

                        left = 1;
                        flame = 1;
                        /*printf("Following Flame left\n");
                        wait(1000);*/
                        left_turn();
            }

             if((Sensor_Data[Center_Flame] < Sensor_Data[Left_Flame]) &&
(Sensor_Data[Center_Flame] < Sensor_Data[Right_Flame]) &&
(Sensor_Data[Center_Flame] < Flame_ThresholdC)){
                        forward();
                        /*printf("Going forward\n");
                        wait(1000);*/

                        if(Sensor_Data[Center_Flame] <= 85){
                                    stop();
                                    wait(1000);
                                    reverse();
                                    wait(300);
                                    stop();
                                    wait(500);

                                    turn_180();

                                    stop();

                                    write_int(Sensor_Data[Center_Flame]);

                                    write("Fan On");

                                    fan();

                                    victory_dance();


                                    flame_on = 0;

                                    stop();


                        }

            }

}


/*void short_check_fire() {


            int j, k, sumL=0, sumC=0, sumR=0, averageL, averageC, averageR;

            for(j=0; j < 11; j++){

                        Fire_DataL[j]=analog(Left_Flame);

                        Fire_DataC[j]=analog(Center_Flame);

                        Fire_DataR[j]=analog(Right_Flame);

            }
        for(k=0; k < 11; k++){
                sumL += Fire_DataL[k];
                sumC += Fire_DataC[k];
                sumR += Fire_DataR[k];
        }

         Sensor_Data[Left_Flame] = (sumL/10);
         Sensor_Data[Center_Flame] = (sumC/10);
         Sensor_Data[Right_Flame] = (sumR/10);

}*/

void init_flame(){
        sensor_read();
        Flame_ThresholdR = (analog(Right_Flame)-8);
        Flame_ThresholdC = (analog(Center_Flame)-8);
        Flame_ThresholdL = (analog(Left_Flame)-8);
        write("Left_Flame: ");
        write_int(Flame_ThresholdL);
        write("Center Flame: ");
        write_int(Flame_ThresholdC);
```

```
	write("Right Flame: ");
	write_int(Flame_ThresholdR);
}
void turn_180(){
	rspeed = -Maxspeed;
	lspeed = Maxspeed;
	motorme(RIGHT_MOTOR, rspeed);
	motorme(LEFT_MOTOR, lspeed);
	wait(900);
	stop();

}

void fan(){
	SET_BIT(PORTD, 0x08);			/* turns fan on (set bit 3 to 5 volts) */
	wait(5000);
	write("Turning off Fan");
	CLEAR_BIT(PORTD, 0x08);				/* turns fan off (set bit 3 back to 0
volt) */
	wait(1500);
}

void search(){
        forward();

        wait(time);

        stop();

        sensor_read();


        if((Sensor_Data[Center_Flame] >= Flame_ThresholdC) &&
(Sensor_Data[Right_Flame] >= Flame_ThresholdR) && (Sensor_Data[Left_Flame] >=
Flame_ThresholdL)){

                turn_90;

                time += 750;


        }


        else {

            flame_on=1;


            while(flame_on == 1){

                check_bumper();

                look_left();

                look_right();

                follow_flame();

            }

        }
```

```
}


void turn_90(){
        rspeed = -Maxspeed;
        lspeed = Maxspeed;
        motorme(RIGHT_MOTOR, rspeed);
        motorme(LEFT_MOTOR, lspeed);
        wait(450);
        stop();
}


void setdigport(){
         SET_BIT(DDRD, 0x08);                /* set PortD bit 3 as dig output */


         CLEAR_BIT(PORTD, 0x08);             /* set PortD bit 3 to 0 (0 volts) */


}


void righttt(){
        rspeed = -Maxspeed;
        lspeed = Maxspeed;
        motorme(RIGHT_MOTOR, rspeed);
        motorme(LEFT_MOTOR, lspeed);
        wait(50);

        rspeed = Maxspeed;

        lspeed = -Maxspeed;

        motorme(RIGHT_MOTOR, rspeed);

        motorme(LEFT_MOTOR, lspeed);

        wait(50);

        stop();

}


void leftt(){

        rspeed = Maxspeed;

        lspeed = -Maxspeed;

        motorme(RIGHT_MOTOR, rspeed);

        motorme(LEFT_MOTOR, lspeed);

        wait(50);

        rspeed = -Maxspeed;

        lspeed = Maxspeed;

        motorme(RIGHT_MOTOR, rspeed);

        motorme(LEFT_MOTOR, lspeed);

        wait(50);

        stop();

}
```

```
void shake(){

        int a;

        for(a = 0; a < 5; a++){

                rightt();

                leftt();

        }

}


void victory_dance(){

        shake();

        turn_90();

        shake();

        turn_180();

        shake();

        turn_90();

        shake();

        stop();

        wait(3000);

}
```