C.L.I.F.F.
Computerized Logical Intelligent Fire Fighter

Proposal
by

Albert Teira

EEL5666 – Intelligent Machines Design Laboratory

9 August 2000

Table of Contents

Abstract

The Computerized Logical Intelligent Fire Fighter is an autonomous mobile agent, which detects fire and extinguishes it. By using a sensitive Ultra-violet sensor, Cliff can detect a flame as small as a candle up to 5 meters away. Cliff was designed to be able to navigate through a miniature house with four rooms. By using IR and a bump sensor, Cliff can navigate successfully through the house. Once UV is detected, Cliff will proceed to extinguish the flame with a sprinkler head attached to a water pump.

Executive Summary

Cliff main behaviors include navigating through a miniature house, detecting a flame, pinpointing the flame, and extinguishing it.

Cliff’s platform is comprised of a disc with wheels placed in a centerline of the disc, allowing him to be able to turn in place. To navigate through the house, Cliff is equipped with IR emitter / detector pairs and ten micro-switches. By having IR detectors mounted on either side of the platform parrallel to the wheels, Cliff is able to avoid contact with the walls of the miniature house as he navigates through it. In case of contact with the walls, a ring connected to the micro switches found all around the main body help him avoid further contact.

To detect and pinpoint the flame, Cliff uses the UV TRON made by Hamamatsu Photonics. The UV TRON sensor detects UV in a wide range and can detect form over 5 meters away. To help pinpoint the flame, Cliff uses software to help to narrow where the flame actually is.

To extinguish the flame, a 12V water pump activated by the microprocessor with a 5V relay. The pump is shut off when UV is no longer detected.

Once the flame is extinguished, Cliff continues to move randomly throughout the house.

Introduction

Cliff is an autonomous fire - fighting robot with three main behaviors. The first behavior if navigating through a miniature house using IR and bump detectors. The second behavior is finding a flame and pinpointing it. To do this a UV sensor is used in combination with some software algorithms. The third behavior is extinguishing the flame, which is done using a water pump and a separate 12V power supply.

This report is a detailed description of all cliff's systems, sensors, and behaviors. Included are circuit diagrams, experimental data, and C code for cliff's main operating program. Also, behavioral successes and failures are also noted and explained.

Integrated System

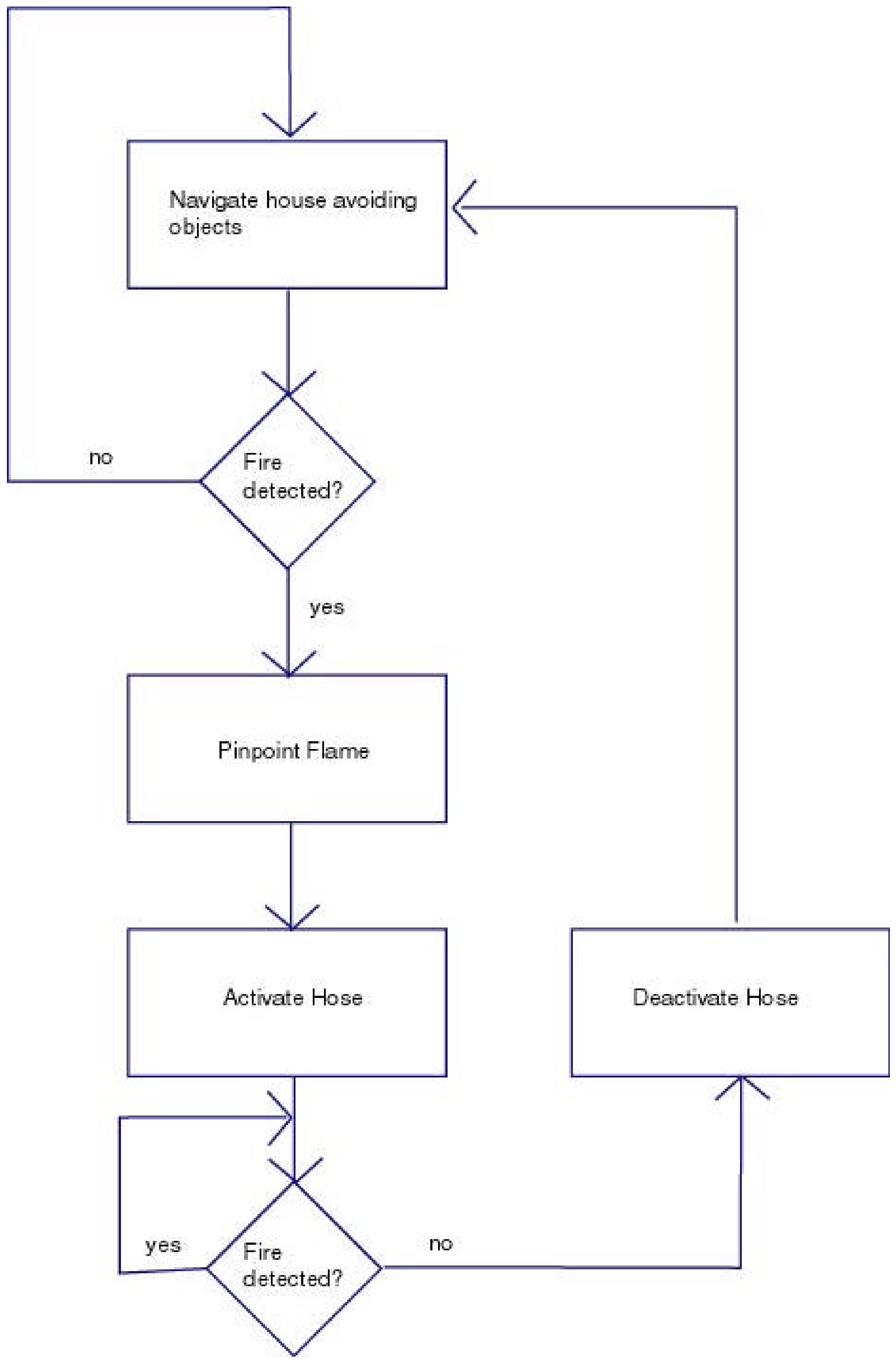
The core of cliff's processing system is the Motorola 68HC11 microcontroller running at 2 MHz. This particular processor includes eight analog input ports as well as a variety of output ports and has limited memory. When combined with the ME11 expansion board made by Mekatronix, the HC11's memory is increased to 32KB and a 40kHz pulse is also now added to the output characteristics. An additional feature of the ME11 board is a motor driver chip, which allows two DC motors to be controlled easily by the microprocessor.

Since the HC11 only provides eight analog inputs, an analog multiplexing circuit is added to the HC11 – ME11 system. This now gives cliff the capability to have 15 input ports, increasing his sensing capabilities. It works by latching the data bus and using three of those bits to select the proper channel in the multiplexing chip. Address $6000 is\accessed to activate the multiplexing circuitry. This configuration allows cliff to have 5 additional digital outputs, which are used for the hose activation and for the behavior LEDs found on the main control panel of cliffs cover. The complete circuit diagram is given in Appendix A.

On the main control panel located n the side of cliff's cover are two switches (a SPST and a SPDT) to control the on/off and download/run mode options. Also located on this panel is a power-on LED and a bank of 4 LEDs to allow feedback to help in debugging.

Cliff also has a resistor bank chip, which contains all the resistors for the different sensors such as the bump detector, the CDS cells, and the LEDs. Circuit diagrams are all provided in the descriptions of the respective sensors below.

The following flowchart describes the main operation of cliff's behaviors. It is a state – diagram describing each step in his behavior routine:



Navigate house avoiding objects
no
Fire detected?
yes
Pinpoint Flame
Activate Hose
Deactivate Hose
yes
Fire detected?
no

Figure 1

The given structure allows cliff many capabilities to accomplish his objectives. The additional memory allows the capability to write complicated software to better enhance his behavioral techniques. Also, the additional input ports give him a better idea of the environment he is in. With the possibility of having 15 sensors, cliff shouldn't have trouble determining the state of his surroundings.

The above systems (HC11 + ME11 + analog multiplexor) are powered by eight 1.2V NiCd batteries located underneath cliff's main platform. The water hose located on the side of his main cover is also powered by a separate battery pack to prevent the pump from discharging cliff's main power supply.

Mobile Platform

Cliff's platform is comprised of a wooden disc for the base and a wood cover placed on the base.

Scope

One of Cliff's major objectives is navigate through a miniature house comprised of four rooms. Most of the hallways are just a little over 13" wide, so turning was something that had to be done in place. This meant using a platform similar to an RC car would not be a good idea. A round base was chosen with the wheels aligned along a centerline of the base. This allowed Cliff to be able to turn in place as wheel as preventing him from getting stuck or having his body snag on one of the walls.

Specifications

Using the base of the Talrick robot, Cliff's main disc is pictured below:

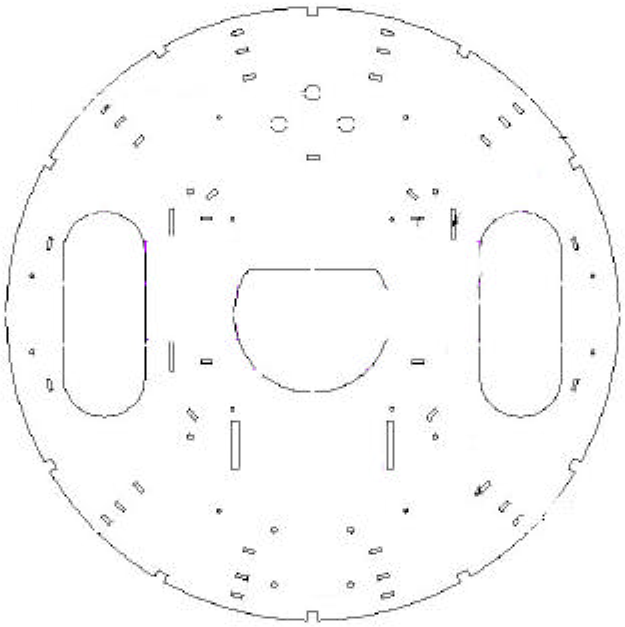


Figure 2

The body is large enough to support the microprocessor board and all the different sensors and devices needed to accomplish the behaviors. The platform also had to have enough power to carry the weight of the pump and all the electronics successfully. For this 45 oz-in servos were used as the motors.

Objectives

Cliff's main objective that related to the design of his mobile platform was to navigate through the miniature house successfully. Because the Talrick design has been perfected over time, the design is optimal for navigating through a small environment. Upon testing in the environment created (which comprised of a miniature house with white walls and four rooms; the diagram is included in Appendix , page) the platform chosen seems to be the best suited. If Cliff would get caught in a corner, the round body enables him to turn and slip free of the obstacle. Also, being able to turn in place helps him turn easily without the need of backing up or worrying if there is room to turn or not.

Actuation

Scope

The types of actuation on the cliff system are motor movement and water pump activation. The amount of actuation on the system was kept to a minimum to keep the robot as simple as possible

Specifications

The motor driving chip located on the ME11 expansion board handles the motor control. The motors are activated using a routine known as "init_motorme()" and are modified by calling the function "motorme(motor, speed in %)" . This makes it easy to interface and control the motors using software.

The water pump mounted on the side of cliff is activated using a 5V relay. By writing a "1" to one of the digital outputs, the relay closes and connects the pump to a separate 12V battery pack mounted on the main cover of the body. Activating the pump is accomplished with the following code: "#define HOSE_ON *(unsigned char *)(0x6000) = 0xff" and then "HOSE_ON;". The pump was connected to a water reservoir (a hamster bottle) on one end and a sprinkler head on the other. The water pump was chosen because of its low input voltage and small size, making it perfect for use on a small robot. The pump does pull a large amount of current thus it has its own battery pack to supply the current required for operation.

The motors gave me no problems, they were easy to install and operate using the ME11. The pump did give me some operating problems throughout the development of cliff. First, the water reservoir seemed to work best with a hole cut out of the top of the bottle to allow air to pass in while the water was being drained. Secondly, the first pump I used had two copper plates as the input voltage terminals. By soldering wire to these I was able to operate the pump with little success. The solder did not bond well to the copper plates and when the pump was activated, the battery would drain in a matter of seconds. The second pump had wires built in as opposed to the copper plates. These wires soldered to a female header worked remarkably better than the original pump.

Sensors

Scope

Cliff's objectives can only be realized by using the right sensors. Below is a table of all the sensors used in the cliff system and to what behavior they relate to.

Specifications

Sensor	Behavior
Ultra-violet UV TRON sensor	Flame detection, pinpoint flame
CDS cells (Pyro)	Flame detection, pinpoint flame
CDS cells (Floor)	Recognizing floor markings
Bump (micro switches)	Object avoidance, collision avoidance
Infrared emitter / detector pairs	Object avoidance

Table 1

Objectives

These sensors all suit cliff to perform all the behaviors necessary to accomplish his goal. To navigate through the house, the IR, bump, and CDS cells (Floor) are used. For the flame detection, the UV and CDS (Pyro) cells are utilized.

Individual Sensor Information

UV Sensor

The UV sensor used in the cliff system is the UV TRON made by the Hamamtsu Photonics Corporation, part # R2868.Along with the sensor from Hamamatsu, a driving circuit they manufacture is also used; part # C3704.

Scope

Cliff will need to be able to detect a flame successfully to be able to locate it. There are many different ways to sense fire, one of which is light. To distinguish the light that the fire emits from other ambient light, a narrow spectrum that only the flame emits must be isolated and detected separate from other sources. One of these spectrums that can be observed is the ultra-violet spectrum. By detecting the UV the flame emits, one could successfully locate the fire amidst an environment of other light. By using a UV detecting bulb sampled from Hamamtsu Photonics and the accompanying driving circuit, this UV released by the flame can be accurately detected and easily integrated into the HC11 microprocessor.

Specifications

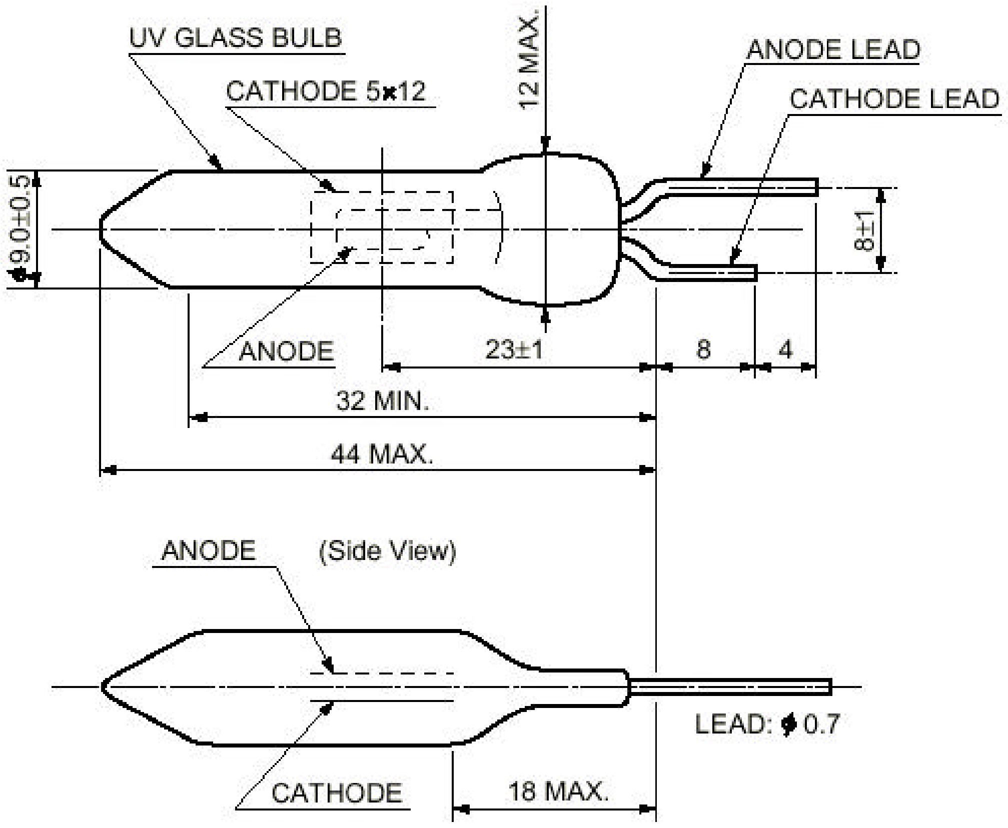


FIGURE 3

The main element in the UV sensor is a bulb with two terminals that operates at about 350V. By using the photoelectric effect of metal and the gas multiplication effect, the bulb is able to detect a spectrum of 185nm to 260nm while being insensitive to visible light. The UV TRON’s spectral response is illustrated in FIGURE 2.

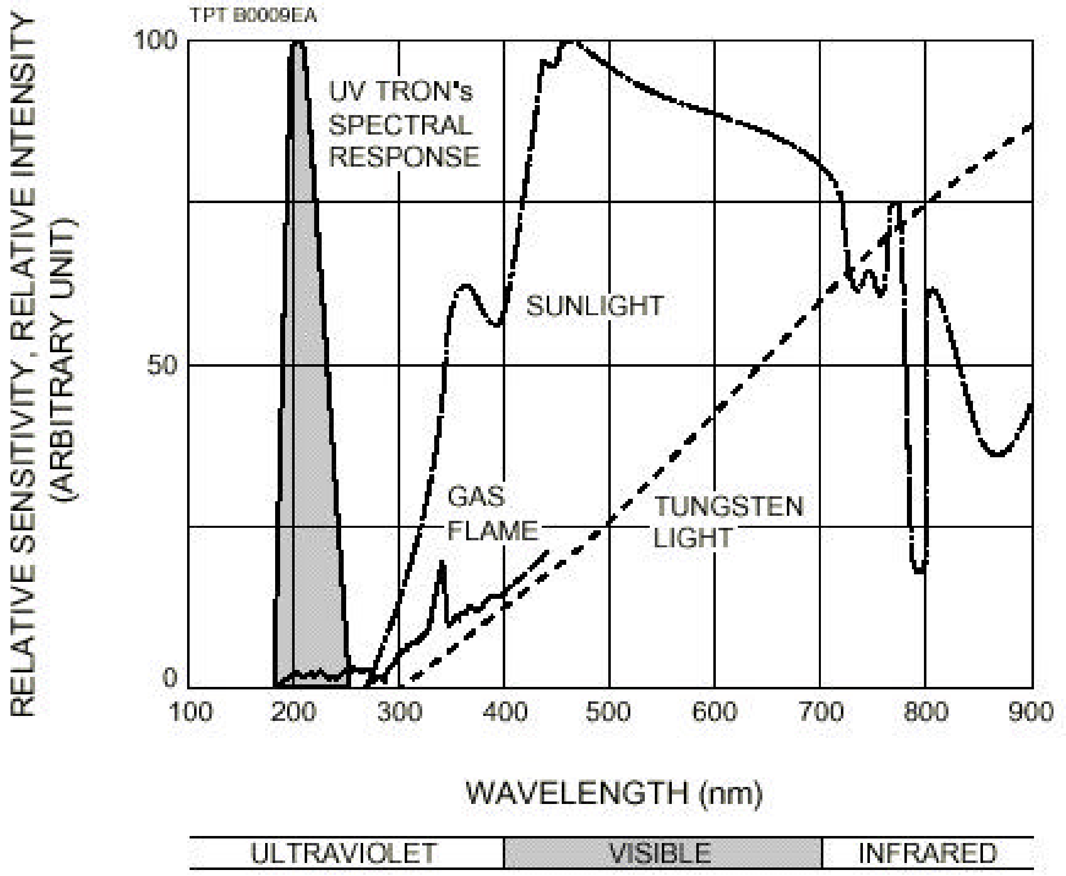


FIGURE 4

With such a narrow range of sensitivity, the UV TRON is well suited for detecting even the smallest flame in a room full of other sources of light. The bulb also has a wide range of sensitivity:

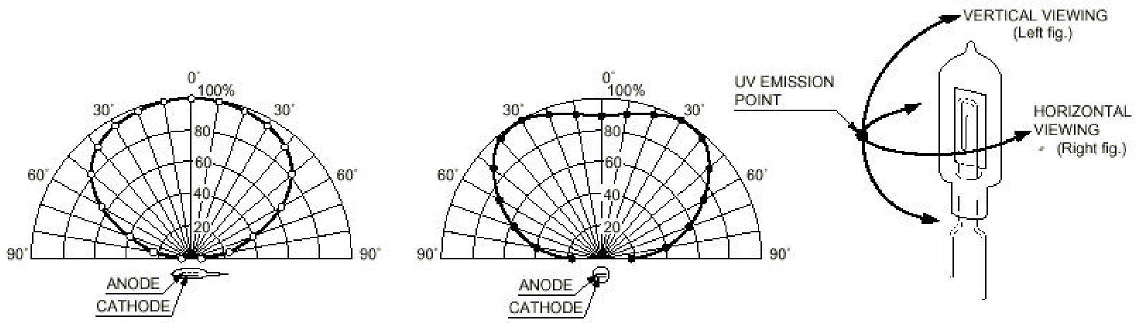


FIGURE 5

UV TRON Driving Circuit C3704

The second part of the sensor is the driving circuit for the UV TRON bulb. Due to the operating voltage of the bulb being approx. 350V, the driving circuit provides operation of the sensor with a low input voltage (10 to 30V) and provides a 10ms 5V pulse for an output. The driving circuit also provides a jumper to adjust the level to adjust the level of noise canceling the circuit performs. Another feature of the driving circuit is the option to increase the length of the pulse by adding a capacitor across a jumper provided on the board (1 μF for a 1s pulse, 10 μF for a 10s pulse, etc.). A functional schematic of the driving circuit is provided in FIGURE 4.

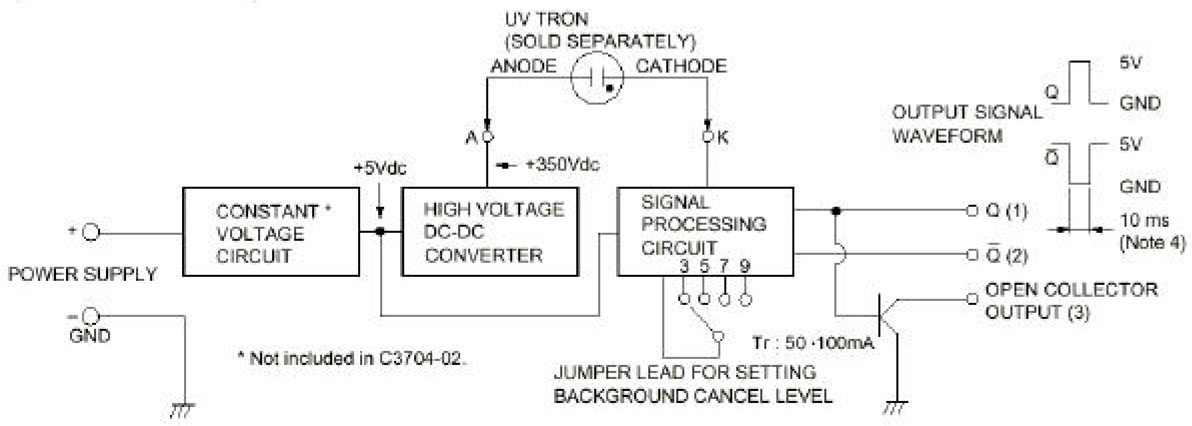

UV TRON
(SOLD SEPARATELY)
ANODE
CATHODE
A
K
+5Vdc
+350Vdc
CONSTANT * VOLTAGE CIRCUIT
HIGH VOLTAGE DC-DC CONVERTER
SIGNAL PROCESSING CIRCUIT
3 5 7 9
+
POWER SUPPLY
−
GND
* Not included in C3704-02.
JUMPER LEAD FOR SETTING BACKGROUND CANCEL LEVEL
Tr : 50 -100mA
OUTPUT SIGNAL WAVEFORM
Q
5V
GND
Q̄
5V
GND
10 ms
(Note 4)
Q (1)
Q̄ (2)
OPEN COLLECTOR OUTPUT (3)

FIGURE 6

By providing 3 different outputs, integration to the HC11 is made easy and convenient. The open collector output has high impedance; therefore the other two outputs are the reasonable ones to use for Cliff.

Experimental Data

The UV sensor was tested in the lab with a DC power supply and an oscilloscope and then later tested connected to the Cliff. In order to limit the range of the sensor, a cylinder with a slit cut vertically helped to accomplish the limiting of the range. Different slit width were tested and the range in degrees was observed. The graph of these results can be found in FIGURE 5. Attaching the sensor to a wooden board and attaching a wooden stick perpendicular to the front of the sensor measured the degrees. This I will refer to as the sensor board. A candle was then lit directly in front of the sensor approx. 21" away. The sensor board could be rotated to the left or to the right until the sensor did not detect the flame.

To measure the angle, a piece of paper was placed under the sensor board. By drawing a line where the perpendicular stick is, the angle at which the flame is no longer detected can be determined by measuring the angle drawn on the paper. Using trigonometry the angle is calculated. Even when limiting the slit to 1/16", then range was still wide, but is suitable for locating the flame.

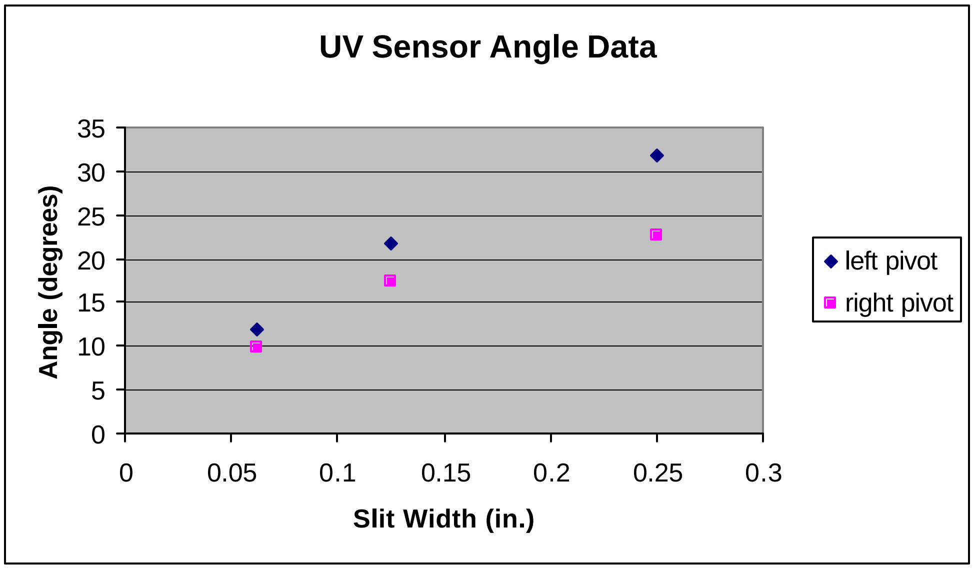


FIGURE 7

The second test consisted of actually hooking the output of the sensor to one of the analog ports of the HC11 and writing to the hyper terminal the value of the port. When there was no flame, the value was 1. When a flame was detected 138 was the value. It should have been around 255, but the difference is good enough (it might have to do with the fact that I connected it to 5V instead of 10V). The response was real-time and should give no timing trouble to the HC11 due to the 1 μF capacitor increasing the pulse width to 1s.

Integration to the HC11 Microprocessor

Integrating the UV sensor to the HC11 can be done in a few different ways. Since the output is a pulse, the IRQ or Pulse Accumulator systems can be used to detect an edge and can generate interrupts to tell Cliff whether UV was detected or not. Another way is to use the analog input of the HC11 and read either 0V or 5V. This introduces a timing issue, but can be compensated by extending the pulse width to 1s rather than 10ms to ensure the processor detects the pulse successfully. This makes using the sensor in software easy. In testing, when there is no flame, the analog port value is 1. When the flame is detected, the value is now 138. This gives a good range to distinguish between the two cases of there being a flame and of there not being a flame. By reading the port and outputting to the hyper terminal, the response seems almost instantaneous, therefore timing is not a huge issue when using the sensor this way.

Software Algorithm

Since the sensor works very much like a switch, it is very hard to determine just where in the sensors range the flame actually is. CDS cells were tried to achieve this short-range capability, but there was limited success. Thus the UV sensor must be treated in a special way in software to be able to pinpoint the flame. One way to do this is in the following fashion.

1. Once UV is detected, turn in place clockwise until there is no flame detected.
2. Now turn in place counterclockwise until there is no flame present.
3. By measuring the time it took to sweep the entire range of the sensor, you should have a pretty good idea of how far the flame is. If you compare it to some known value, then you know you are the proper distance away from it.
4. If not, turn clockwise for ½ the time measured, then move forward again for a small time and go back to step 1.

This algorithm was tested and implemented, however time constraints prevented further improvements on it and therefore was never used successfully. Instead a cheap version of the algorithm is used:

1. Turn clockwise for ½ seconds and record UV data.
2. Turn counterclockwise for 1 second and record the UV data.
3. Now turn counterclockwise for ½ second, you should now be in the same position you started in.
4. Make a decision. If you saw fire when you turned left and not right, then the fire is to you left and turn clockwise for ¼ second. If the opposite turn counterclockwise for ¼ second. Move forward after every cycle.

This algorithm in theory is inferior to the first one, but works with relative success. Because of time it was the best option for the algorithm. Timing is crucial when using the sensor in this way. Please see Appendix for actual code.

CDS cells (Pyro)

Scope

The UV sensor does not have the capability to detect how far away the flame is; it can only detect its presence. For this a short-range sensor must be implemented. Cliff uses CDS cells to detect the brightness of the flame. By using two CDS cells, the flame can be pinpointed.

Specifications

The CDS cell is a photosensitive resistor that changes resistance under different light conditions. By using a voltage divider circuit hooked up to one of the analog ports of the HC11, the CDS cell can be used as a sensor with ease. The following circuit was built for using the CDS cell:

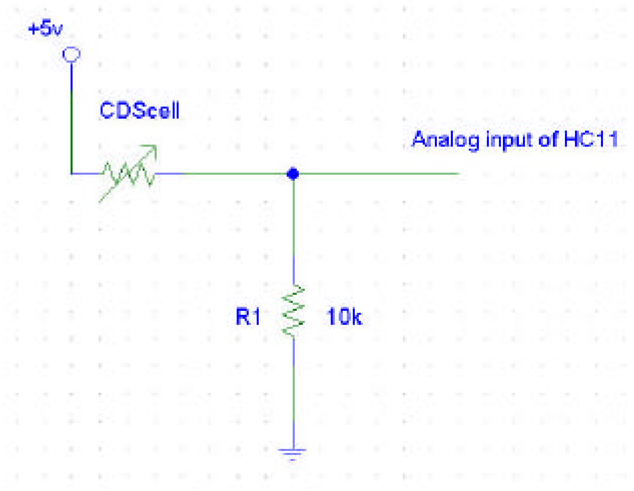


Figure 8

Flame Distance	Left CDS cell Reading	Right CDS cell Reading
No flame	13	15
6”	14	15
3”	21	22
1”	26	22

Table 2

Software Algorithm

The software algorithm was simple; the analog port was read and processed accordingly. Please see Appendix for actual code.

Problems Encountered

Unfortunately, the Pyro CDS cells had limited success in pinpointing the flame. The readings were too different in different rooms, and the flickering of the candle flame cause problems as well. Perhaps through some intense software these problems can be overcome, but they were not for the cliff system.

CDS Cells (Floor)

Scope

Since much of the design of cliff was centered around the Trinity College Fire-Fighting Robot Contest, floor CDS cells are implemented to distinguish between white and black markings on the floor.

Specifications

Two CDS cells and three green LEDs are mounted on the bottom of cliffs platform (see Appendix for details) . They are used in the same way the Pyro CDS (please see figure) cells are used, in a voltage divider circuit hooked up to the analog port of the HC11.

Experimental Data

The floor CDS cells worked in distinguishing the difference between a black surface and a white surface, but the difference is not large enough to be used effectively. The difference was only about three or four on the digital scale of the HC11. An option is to use white LEDs rather than green ones to enhance the CDS cell response.

IR Emitter / Detector Pairs

Scope

One of cliffs behaviors is to successfully navigate through a miniature house with four rooms. A strong way to avoid walls is a must, and the IR sensors for the most part provide this a good way to navigate through the house. Cliff's path in the house is random, thus having him avoid the wall constantly is very important. Using eight IR emitters and six IR detectors, cliff can successfully avoid most contact with the walls of the miniature house.

Specifications

The IR emitter used is an IR LED with a 40kHz signal driving it. This signal is provided by the ME11 board and allows up to eight LEDs to be connected directly to the board. The IR detectors used are known as "Sharp Cans" and can detect the 40kHz signal being sent out by the LEDs. By hacking the detectors (modifying the internal circuitry) the detectors give an analog value that is easily read by the analog input port of the HC11 processor. The range is usually between 80 and 130, 80 being less IR detected than at 130.

Experimental Data

Readings were taken by echoing to the hyper terminal the values seen by the sensors as a whit book was held at different distances from the robot. Below is a graph that illustrates the response:

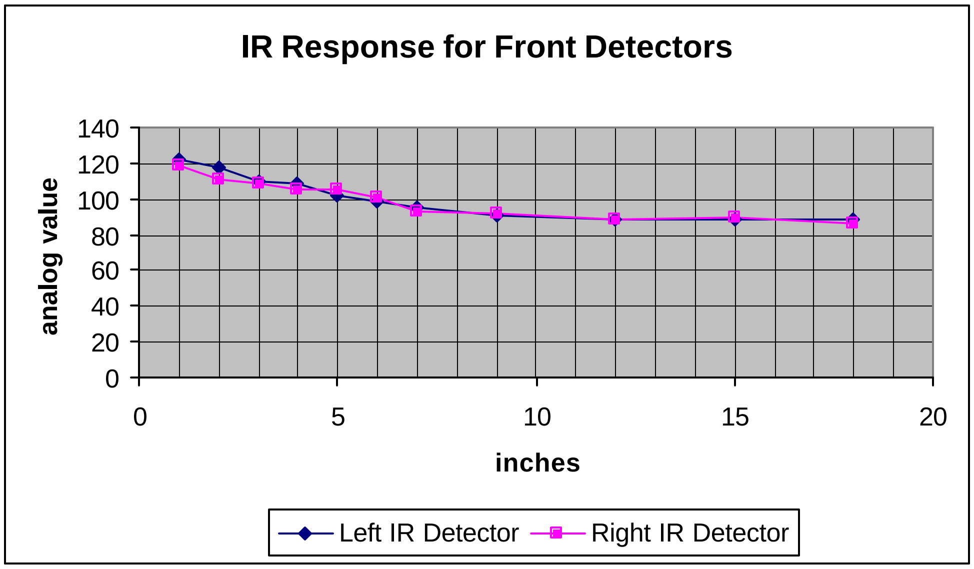


Figure 9

The side IR detectors had similar values to those of the front.

Software Algorithm

Handling the IR readings with software is done easily. Reading the analog values of the detectors and controlling the motors accordingly is the main task of the collision avoidance algorithm. Please see Appendix for code.

The IR detectors worked extremely well in the miniature house environment. The side IR detectors were very important in keeping a certain distance form the walls of the house.

Cliff uses a calibration routine to set the thresholds for any particular room, since different rooms can have different levels of IR.

Bump (micro switches)

Scope

To be able to successfully avoid obstacles, cliff cannot relay only on his IR sensors. The bump sensor serves as a backup incase the IR does not pick up an object and cliff makes contact with it.

Specifications

Cliff's bump sensor is comprised of a ring attached to 10 micro switched located all around the perimeter of cliff's main platform. By using a voltage divider circuit with different resistor values, the bump sensor can be connected to one of the analog inputs and still offer the control of being able to know what part of the robot made contact. The circuit is shown below:

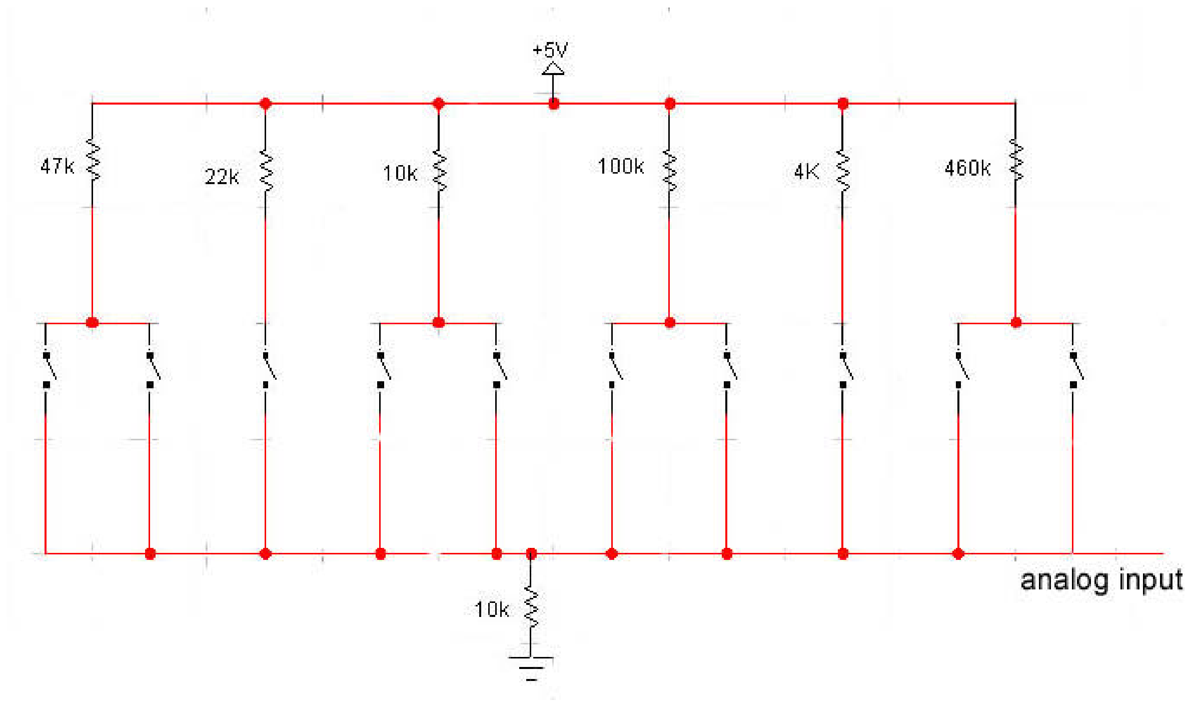


Figure 10

Experimental Data

The values read by the HC11 when the different areas of the bump sensor are contacted are illustrated below:

(directional descriptions are given with respect to the front of cliffs body facing north)

Region	**Analog Value**
N	175
NW	91
NE	82
S	192
SW	102
SE	129

Table 3

Software Algorithm

To respond to a bump detected, cliff turns for a fixed amount of time to attempt to free himself from the object he encountered. This can be done by controlling the motors for a fixed amount of time relative to the point of contact.

The bump sensor worked well in the miniature house. In some areas, such as the edge of a wall in a doorway, the IR sensors could not pick up the wall since it was extremely thin. In this case, the bump sensor proved to be an important part of cliffs system.

Behaviors

Cliff's main behaviors include object avoidance and flame detection and extinguishing.

Scope

The behaviors of cliff are modeled after the Trinity College Contest Rules for the Trinity College Fire-Fighting Robot Contest. In the contest, the robot must navigate through a miniature house made up of four rooms, one of which contains a candle. The robot must locate the candle, approach it, and then proceed to extinguish it. Once extinguished, cliff returns to wondering about the house.

Specifications

The first behavior is the most important - navigating through the house. Successful navigation through the house is essential in locating the room with the candle. After trying different house navigating techniques, it seems random works the best. Although not too much testing was done as far as wall following techniques were made, but random seemed to ensure all the rooms were eventually reached. A successful navigation algorithm may include both randomness and wall following to ensure all possible routes are taken. A small bit of randomness inside the algorithm is important as well to ensure the robot does not get stuck somewhere in the house. There were places where cliff became stuck almost every time he would try to navigate through the house. By adding a random element to the code, cliff can escape such traps. In the actual code the randomness is initiated if both IR detectors are over their respective thresholds. The robot turns in a random direction for a random amount of time.

The second important behavior is detecting and approaching the flame. This is done by scanning the area once the UV sensor detects fire. By taking measurements in a certain range, cliff is able to pinpoint the location of the flame to some degree. By turning clockwise and then counterclockwise for given time interval, he can sweep with the hose activated to ensure the area with the candle is sprayed with water.

The third behavior is to extinguish the fire, which is done by activating a 12V water pump using a 5V relay and a digital output on the HC11.

Experimental Layout and Results

Scope

To determine cliff's performance, a miniature house was constructed matching the house built in the Trinity College Fire-Fighting Robot Contest. A floor diagram can be found in Appendix C. The house was built and test runs were made using a white, eight-inch candle with the floor of the house being the floor of the lab.

Specifications

The house was built of cardboard and painted with flat white paint. The house can fold up, making it easy to store when not being used. Since it was made of cardboard, it was difficult for cliff's bump sensor to be tripped, so the house was taped to the ground to ensure its rigidity.

Navigation through the house was objective number one, and the most important one. If navigation is not successful, the fire can never be put out. First, a completely random routine was written, where whenever cliff's form IR detectors were over their thresholds, cliff would turn for a random time in a random direction. This did not work so well, so the only randomness in his navigating routine is turning in a random direction for a random amount of time whenever both front IRs are tripped.

To extinguish the flame, cliff was placed in the same room to see how well he detected the candle and how well he could pinpoint it. This is where the behavioral LEDs came in very handy, due to the ability to test to see if he detected UV without having to wet the arena every time he was tested. Once this was fine tuned, however several runs with the hose on were conducted.

Conclusion

The cliff system was successful in that he accomplished all his objectives to a certain degree. One of the only problems was being able to pinpoint the flame accurately. With more time, perhaps the algorithm described above could have been implemented with more success, as it could have proven to be most effective. A better short-range flame detection unit could be a good item to add to cliff's system. This could help him overcome the pinpointing problem. A better navigating algorithm is a good thing to try to implement in cliff as well. A wall following – random combination would probably be most effective in navigating through the house. In retrospect, if the project were to be built from the beginning again, a better actuator for the extinguishing mechanism would be implemented. Cliff just couldn't cover enough area to completely douse the candle with water. Perhaps the sprinkler head mounted on a servo would be a good idea, since past fire-fighting robots have had success with it. Better wheels would also be implemented since his current wheels tend to come off often. Although not a problem towards the ends of the development, there were instances where his wheels would just come off.

The key to cliff and all his systems is time. There was an incredible amount of time dedicated to the development of cliff and his success in largely due to the fact that most of his hardware was complete three weeks before the final demo. Allowing at least two weeks for software is a good way to plan the construction of a robot like cliff. Also taking extra consideration of the water in the reservoir not leaking to your electronics is very important. Use sponges, hot glue, or gum to secure the reservoir.

The Computerized Logical Intelligent Fire Fighter was a success and hopefully in the years to come he can be improved so he can be submitted to the Trinity Fire-Fighting Contest.

Documentation

Books, Manuals

Costales, Bryan C from A to Z, Prentice Hall, Inc., Englewood Cliffs, New Jersey 07632, 1985

Hamamatsu Photonics Manuals associated with the UV TRON (R2868) and the UV TRON DRIVING CIRCUIT (C3704). usa.hamamatsu.com

Mekatronix Assembly manuals for the ME11 expansion board and the Talrick platform, www.mekatronix.com

Motorola All manuals associated with the 68HC11 microprocessor, www.motorola.com

Past Reports

Denise, An Autonomous Fire Fighting Robot by Sean Justice, Summer 1999

Skippy, the Fire Fighting Robot by Warren L. Thorton Jr., Spring 2000

Appendix A – Cliff's Main Program in C

```
/*****************************************************************
 *                                                              *
 * Title: cliff.c                                               *
 * Programmer: Albert C. Teira                                    *
 * Date: 3 August 2000                                            *
 * Version: 1.0                                           *
 *                                                        *
 * Description:                                           *
 *        Main program for the autonomous mobile firefighting robot    *
 *     known as C.L.I.F.F. (Computerized Logical Intelligent Fire   *
 *     Fighter. Program has Cliff wonder randomly throughout a      *
 *     Miniature house and avoiding obstacles while searching for   *
 *     a candle, detectable by UV light. Once the UV is detected,   *
 *     Cliff will pinpoint the flame and proceed to extinguish it.  *
 *     Once extinguished, Cliff will continue random path to       *
 *     navigate around the house.                          *
 *                                                        *
 *********************************************************************/

// includes

#include <clocktjp.h>
#include <tjpbase.h>
#include <stdio.h>
#include <analog.h>
#include <mil.h>

/*********************************************************************/
/* Function Defines                                      */
/*********************************************************************/

void getData();
void turnLeft90();
void turnRight90();
void stop();
void checkIR();
void turn();
void checkSideIR();
void checkUV();
void checkPCDS();
void checkBumper();
void calibrate();
void putFireOut();
void pinPointFire();
void fightFire();
void followRight();

/*********************************************************************/
/* Variable Defines                                      */
```

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

#define IR_ON *(unsigned char *)(0x7000) = 0xff
#define SEL_0  *(unsigned char *)(0x6000) = 0x00
#define SEL_1  *(unsigned char *)(0x6000) = 0x01
#define SEL_2  *(unsigned char *)(0x6000) = 0x02
#define SEL_3  *(unsigned char *)(0x6000) = 0x03
#define SEL_4  *(unsigned char *)(0x6000) = 0x04
#define SEL_5  *(unsigned char *)(0x6000) = 0x05
#define SEL_6  *(unsigned char *)(0x6000) = 0x06
#define SEL_7  *(unsigned char *)(0x6000) = 0x07

#define PAT_0  *(unsigned char *)(0x6000) = 0x00
#define PAT_1  *(unsigned char *)(0x6000) = 0x80
#define PAT_2  *(unsigned char *)(0x6000) = 0x40
#define PAT_3  *(unsigned char *)(0x6000) = 0x20
#define PAT_4  *(unsigned char *)(0x6000) = 0x10

#define HOSE_ON *(unsigned char *)(0x6000) = 0xff
#define HOSE_OFF *(unsigned char *)(0x6000) = 0x00

int l_ir, r_ir, left_speed, right_speed, i, bump, l_fcds, r_fcds;
int lf_ir, lb_ir, rf_ir, rb_ir, uv, l_pcds, r_pcds;
int lir_th, rir_th, lsir_th, rsir_th, theresfire;
int pinpointed, flameright, flameleft;
int total_time, start_time, end_time, getcloser;
int i, x;
unsigned rand;



/*********************************************************************/
/* Main                                                   */
/*********************************************************************/

void main(void)
{

/* initializations ****************************************/

        init_motorme();
        init_servome();
        init_serial();
        init_analog();
        init_clocktjp();
```

```
          // turn on IR emitters

          IR_ON;

          // calibrate

          calibrate();

/* main while loop ***************************************/

   while(1)     {

          // get date from sensors

          getData();

          // check IR readings and handle accordingly

          checkIR();

          // check side IR readings and handle them accordingly

          checkSideIR();

          // check bump sensor and handle it accordingly

          checkBumper();

          // check UV reading

          checkUV();

          // if fire is detected, proceed to fight fire, else continue random
          // path

          if (theresfire == 1)     {

                    fightFire();

          } // end if

                    // activate motors for a set time given by te for loop below

                    motorme(RIGHT_MOTOR, right_speed);
                    motorme(LEFT_MOTOR, left_speed);
```

```
        for (i=0; i<30000; i++);

    } // end while


} //end main

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

/**********************************************************************
****
 *                                                                    *
 * calibrate(): the function counts down and then begins to           *
 *              set the thresholds of the IR redings                  *
 *                                                                    *
 *              Calls: stop(), getData(), turnLeft90(), turnRight90() *
 *                                                                    *
 *              Modifies: lir_th, rir_th, lsir_th, rsir_th        *
 *                                                                    *

***********************************************************************
***/

void calibrate()        {

        wait(1000);

        PAT_1;

        wait(1000);

        PAT_2;

        wait(1000);

        PAT_3;

        wait(1000);

        PAT_4;

        wait(2000);
```

```
PAT_1;

getData();

lir_th = l_ir;
rir_th = r_ir;

motorme(RIGHT_MOTOR, 25);
motorme(LEFT_MOTOR, 25);

wait(1000);

stop();

motorme(RIGHT_MOTOR, right_speed);
motorme(LEFT_MOTOR, left_speed);

turnLeft90();

motorme(RIGHT_MOTOR, right_speed);
motorme(LEFT_MOTOR, left_speed);

wait(1000);

getData();

rsir_th = rf_ir;

wait(1000);

turnRight90();
turnRight90();

motorme(RIGHT_MOTOR, right_speed);
motorme(LEFT_MOTOR, left_speed);

wait(1000);

getData();

lsir_th = lf_ir;

wait(1000);

turnRight90();
```

```
        motorme(RIGHT_MOTOR, right_speed);
        motorme(LEFT_MOTOR, left_speed);

} // end calibrate

/************************************************************************
****
 *                                                                      *
 * getData():   function reads the analog ports and sets variables to the  *
 *              readings gathered by the analog readings                *
 *                                                                      *
 *              Calls: none                                             *
 *                                                                      *

 *              Modifies: r_ir, l_ir, rb_ir, rf_ir, lb_ir, lf_ir,      *
 *                        l_pcds, r_pcds, uv, r_fcds, l_fcds, bump      *
 *                                                                      *


*************************************************************************
***/
void getData() {

        r_ir = analog(1);
        l_ir = analog(0);
        rb_ir = analog(2);
        rf_ir = analog(3);
        lb_ir = analog(5);
        lf_ir = analog(6);

        SEL_1;
        l_pcds = analog(7);

        SEL_4;
        r_pcds = analog(7);

        SEL_3;
        uv = analog(7);

        SEL_5;
        r_fcds = analog(7);

        SEL_6;
        l_fcds = analog(7);
        SEL_7;
        bump = analog(7);

} //end getData()

/***********************************************************************
****
 *                                                              *
 * checkIR():   function compares the IR readings to the thresholds      *
 *              set by the calibration routine. The function then takes   *
 *              care of the object avoidance my controlling the motors        *
 *              accordingly.                                    *
 *                                                              *
 *              Calls: none                                     *
 *                                                              *
 *              Modifies; none                                  *
 *                                                              *


***********************************************************************
***/

void checkIR()          {

        if (l_ir > lir_th)          {

                right_speed = -20;
                left_speed = 20;
        }

        else if (r_ir > rir_th)    {

                right_speed = 20;
                left_speed = -20;
        }

        else if ((r_ir > rir_th) && (l_ir > lir_th))       {

                turn();

        }


        else    {
                right_speed = 75;
                left_speed = 75;
```

```
		}
} // end checkIR()

/************************************************************************
****
 *                                                                    *
 * turn():		function makes cliff turn in a random direction for a          *
 *			random amount of time. This function was taken from code   *
 *			provided in class by dr. Antonio Arroyo                    *

 *                                                                    *
 *			Calls: none                                               *
 *                                                                    *
 *			Modifies; none                                            *
 *                                                                    *


*************************************************************************
***/

void turn()	{

rand = TCNT;

if (rand & 0x0001)	{

		right_speed = 20;
		left_speed = -20;

}
else {

		right_speed = -20;
		left_speed = 20;

}

		motorme(RIGHT_MOTOR, right_speed);
		motorme(LEFT_MOTOR, left_speed);

i = (rand % 1024);
if (i > 250) wait(i); else wait(250);

}
```

```
/**********************************************************************
****
 *                                                                    *
 * checkSideIR(): function reads side IR detectors and compares them to    *
 *                 the thresholds set by the calibratin routine. If the     *
 *                 values are over these thresholds, the function will      *
 *                 handke the motors accordingly to avoid object            *

 *                                                                    *
 *                 Calls: none                                        *
 *                                                                    *
 *                 Modifies; none                                     *
 *                                                                    *


**********************************************************************
***/

void checkSideIR()     {

        if (lf_ir > lsir_th)        {

                right_speed = 0;
                left_speed = 45;
        }
        else if (rf_ir > rsir_th) {

                right_speed = 45;
                left_speed = 0;
        }
        else    {
        }


} // end checkSideIR()

/**********************************************************************
****
 *                                                                    *
 * checkUV(): function checks the value of uv set by getData() and       *
 *                assigns the variable 'theresfire' a boolean value, a "0"   *
 *                if there is no fire or a "1" if there is fire.          *

 *                                                                    *
 *                Calls: none                                         *
 *                                                                    *
 *              Modifies; theresfire                                *
 *                                                                  *


*********************************************************************
***/

void checkUV()        {

        if (uv > 100)   {

                theresfire = 1;

        }
        else    {

                theresfire = 0;

        }

} // end checkUV

/********************************************************************
****
 *                                                                  *
 * fightFire(): function calls the different fire fighting routines and    *
 *              handles the actual step-by-step of extinguishing the      *
 *              flame.                                              *

 *                                                                  *
 *              Calls: pinPointFire(), putFireOut(), turnRight90()
                  *
 *                                                                  *
 *              Modifies; none                                      *
 *                                                                  *


*********************************************************************
***/

void fightFire()        {

        pinPointFire();

        putFireOut();
```

```
            turnRight90();
            turnRight90();

} // end fightFire();

/************************************************************************
****
 *                                                                      *
 * pinPointFire(): function takes diffrent reading from around the      *
 *                 environment to determine where the candle is to a    *
 *                 certain degree. The function will not exit until    *
 *                 the flame has been pinpointed.                       *

 *                                                                      *
 *                 Calls: checkUV(),                                    *
 *                                                                      *
 *                 Modifies; flameleft, flameright, pinpointed          *
 *                                                                      *


*************************************************************************
***/

void pinPointFire()     {

 pinpointed = 0;

 while(pinpointed == 0)         {

 // turn clockwise for 1 second , then aquire data

   motorme(RIGHT_MOTOR, -20);
   motorme(LEFT_MOTOR, 20);

            wait(1000);

            checkUV();

            if (theresfire == 1)      {

                        flameleft = 1;

            }
            else      {

                        flameleft = 0;
```

```
        }

// turn counter-clockwise for 2 seconds, the n acquire data

        motorme(RIGHT_MOTOR, 20);
  motorme(LEFT_MOTOR, -20);

        wait(2000);

        checkUV();

        if (theresfire == 1)      {

                flameright = 1;

        }
        else     {

                flameright = 0;

        }

// now turn clockwise for 1 second to return to original position

        motorme(RIGHT_MOTOR, -20);
  motorme(LEFT_MOTOR, 20);

        wait(1000);

// now turn motors off and wait for 1 second

        motorme(RIGHT_MOTOR, 0);
  motorme(LEFT_MOTOR, 0);

        wait(1000);

        if ((flameleft == 1) && (flameright == 1))    {

                pinpointed = 1;

        }
        else if  ((flameleft == 0) && (flameright == 0))       {

                motorme(RIGHT_MOTOR, 20);
                motorme(LEFT_MOTOR, 20);
                wait(500);

                pinpointed = 0;

        }

        else if ((flameleft == 0) && (flameright == 1))        {

                motorme(RIGHT_MOTOR, -20);
                motorme(LEFT_MOTOR, 20);

                wait(500);

                motorme(RIGHT_MOTOR, 0);
                motorme(LEFT_MOTOR, 0);

                pinpointed = 1;
        }

        else if ((flameleft == 1) && (flameright == 0))        {

                motorme(RIGHT_MOTOR, 20);
                motorme(LEFT_MOTOR, -20);

                wait(500);

                motorme(RIGHT_MOTOR, 0);
                motorme(LEFT_MOTOR, 0);

                pinpointed = 1;
        }

        else {

                pinpointed = 0;

        }

        } // end while loop

} // end pinPointFire()

/**********************************************************************
****
 *                                                                      *
 * putFireOut(): function handles the fire extinguishing process. Will     *
 *                activate hose and spread the water by moving in place   *
 *                and approcahing the flame. Will not exit until the      *
 *                flame is gone.                                          *

 *                                                                        *
 *                Calls: checkUV()                                        *
 *                                                                        *
 *                Modifies; none                                          *
 *                                                                        *


*************************************************************************
***/

void putFireOut()       {



        while(theresfire == 1) {

        HOSE_ON;

 // turn for 1/2 second

        motorme(RIGHT_MOTOR, 20);
   motorme(LEFT_MOTOR, -20);

   wait(500);

 // stop for 1/2 second

   motorme(RIGHT_MOTOR, 0);
   motorme(LEFT_MOTOR, 0);

   wait(500);

 // turn for 1 second

   motorme(RIGHT_MOTOR, -20);
   motorme(LEFT_MOTOR, 20);

   wait(1000);

 // stop for 1/2 second
```

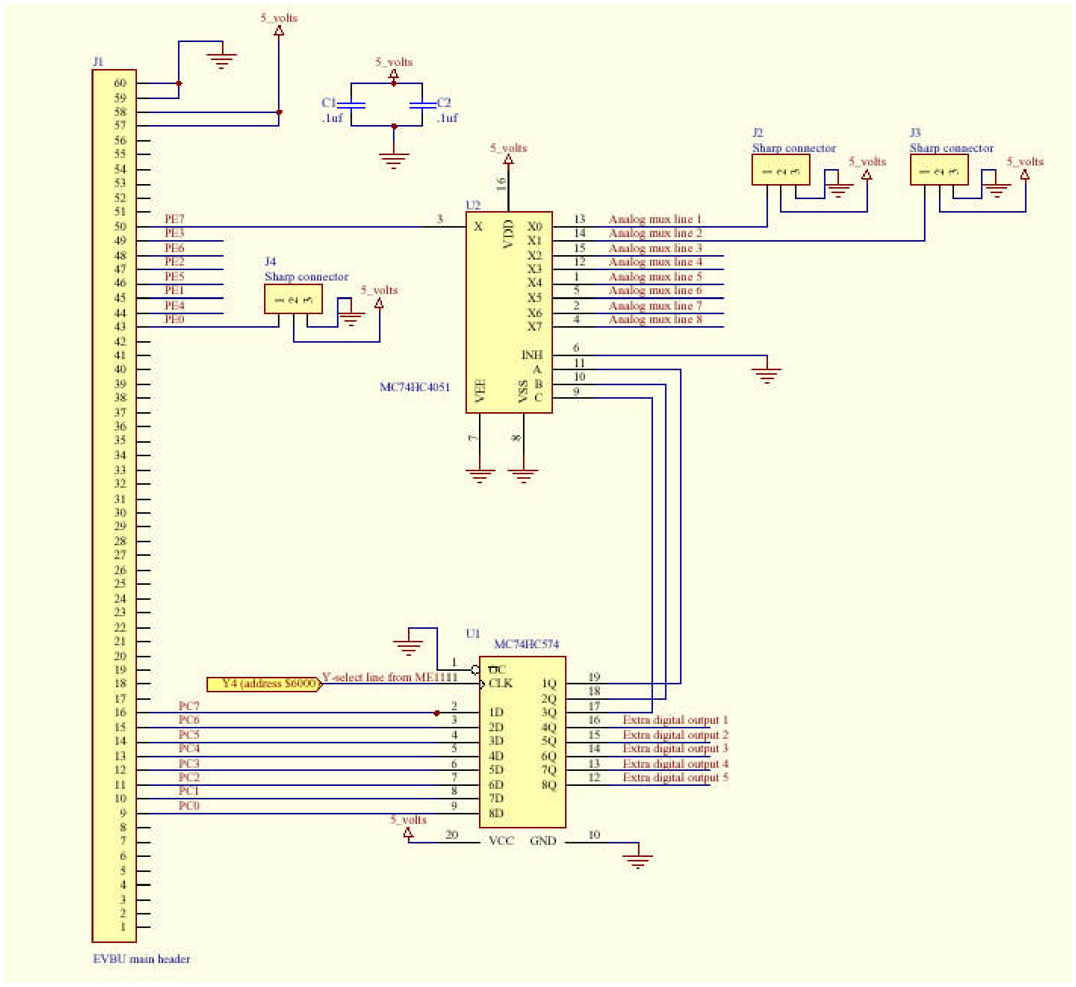
```
   motorme(RIGHT_MOTOR, 0);
   motorme(LEFT_MOTOR, 0);

   wait(500);

 // turn for 1/4 second

   motorme(RIGHT_MOTOR, 20);
   motorme(LEFT_MOTOR, -20);

   wait(500);

 // stop for 1/2 second

   motorme(RIGHT_MOTOR, 0);
   motorme(LEFT_MOTOR, 0);

   wait(500);

 // move forward for 1/2 second

          motorme(RIGHT_MOTOR, 20);
   motorme(LEFT_MOTOR, 20);

   wait(1000);

   checkUV();

          } // end while loop


} // end putFireOut()

/*********************************************************************
****
 *                                                                   *
 * checkPCDS(): function checks the Pyro CDS cell values and acts        *
 *                    accordingly                                  *

 *                                                                   *
 *                    Calls: pinPointFire(), putFireOut(), turnRight90()       *
 *                                                                   *
 *                    Modifies; none                                   *
 *                                                                   *
```

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

void checkPCDS()      {

          if (l_pcds > 10)          {

                    stop();
          }

} // end checkPCDS()

/**********************************************************************
****
 *                                                                   *
 * checkBumper(): function checks the bump semsor and controls the motors
                    accordingly.                                     *

 *                                                                   *
 *                  Calls:                                           *
 *                                                                   *
 *                  Modifies; none                                   *
 *                                                                   *


***********************************************************************
***/

void checkBumper()   {

          if (bump > 150)          {

                    motorme(RIGHT_MOTOR, -20);
                    motorme(LEFT_MOTOR, -20);

                    PAT_1;

                    wait(500);

                    motorme(RIGHT_MOTOR, 20);
                    motorme(LEFT_MOTOR, -20);

                    wait(1000);

          }
} // end checkBumper()

/**********************************************************************
****
 *                                                                    *
 * turnLeft90(): function controls motors to turn in place 90 degrees      *
 *                   counterclockwise.                                 *

 *                                                                    *
 *                   Calls: stop()                                    *
 *                                                                    *
 *                   Modifies; none                                   *
 *                                                                    *


**********************************************************************
***/

void turnLeft90()        {

          motorme(RIGHT_MOTOR, 30);
          motorme(LEFT_MOTOR, -30);

          wait(800);

          stop();

} // end turnLeft90()

/**********************************************************************
****
 *                                                                    *
 * turnRight90(): function controls motors to turn in place 90 degrees      *
 *                    clockwise.                               *

 *                                                                    *
 *                    Calls: stop()                                   *
 *                                                                    *
 *                    Modifies; none                                  *
 *                                                                    *


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

```
void turnRight90()      {

            motorme(RIGHT_MOTOR, -30);
            motorme(LEFT_MOTOR, 30);

            wait(800);

            stop();

} // end turnRight90()

/**********************************************************************
****
 *                                                                    *
 * stop(): function controls motors to stop                         *

 *                                                                    *
 *            Calls: stop()                                         *
 *                                                                    *
 *         Modifies; none                                           *
 *                                                                    *


***********************************************************************
***/

void stop()      {

            right_speed = 0;
            left_speed = 0;
}
```

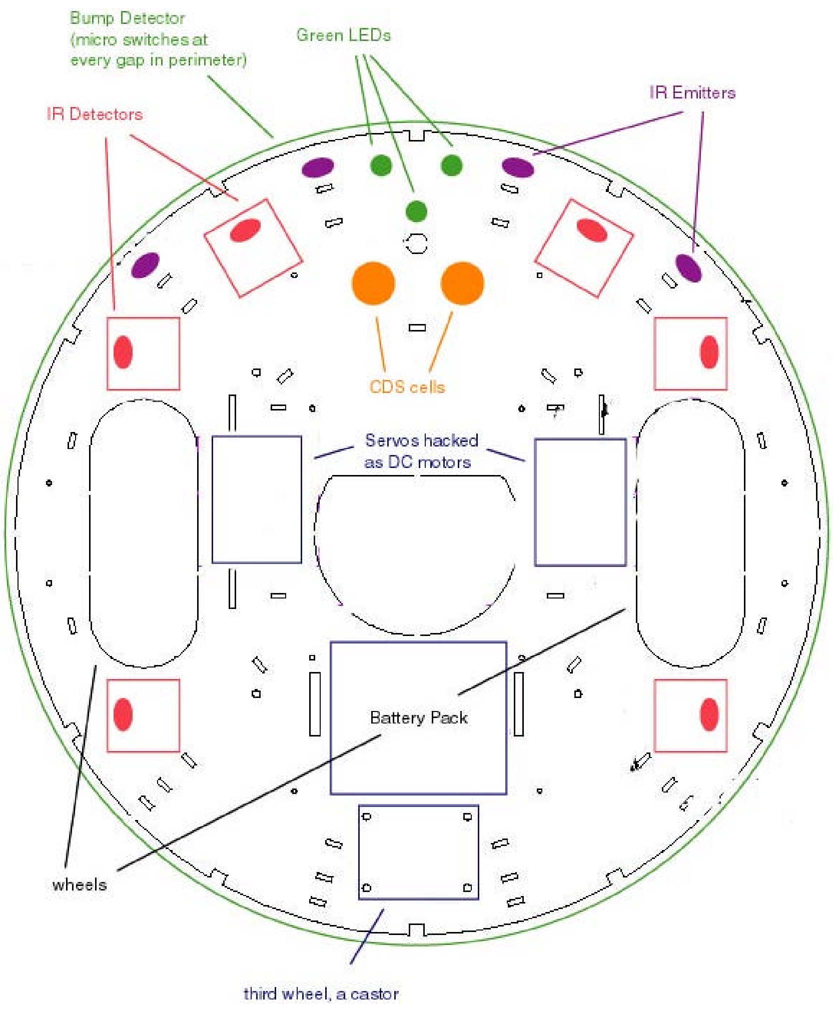
Appendix B – Circuit Diagrams

Analog Multiplexor



Appendix C – Additional Diagrams

Bottom View

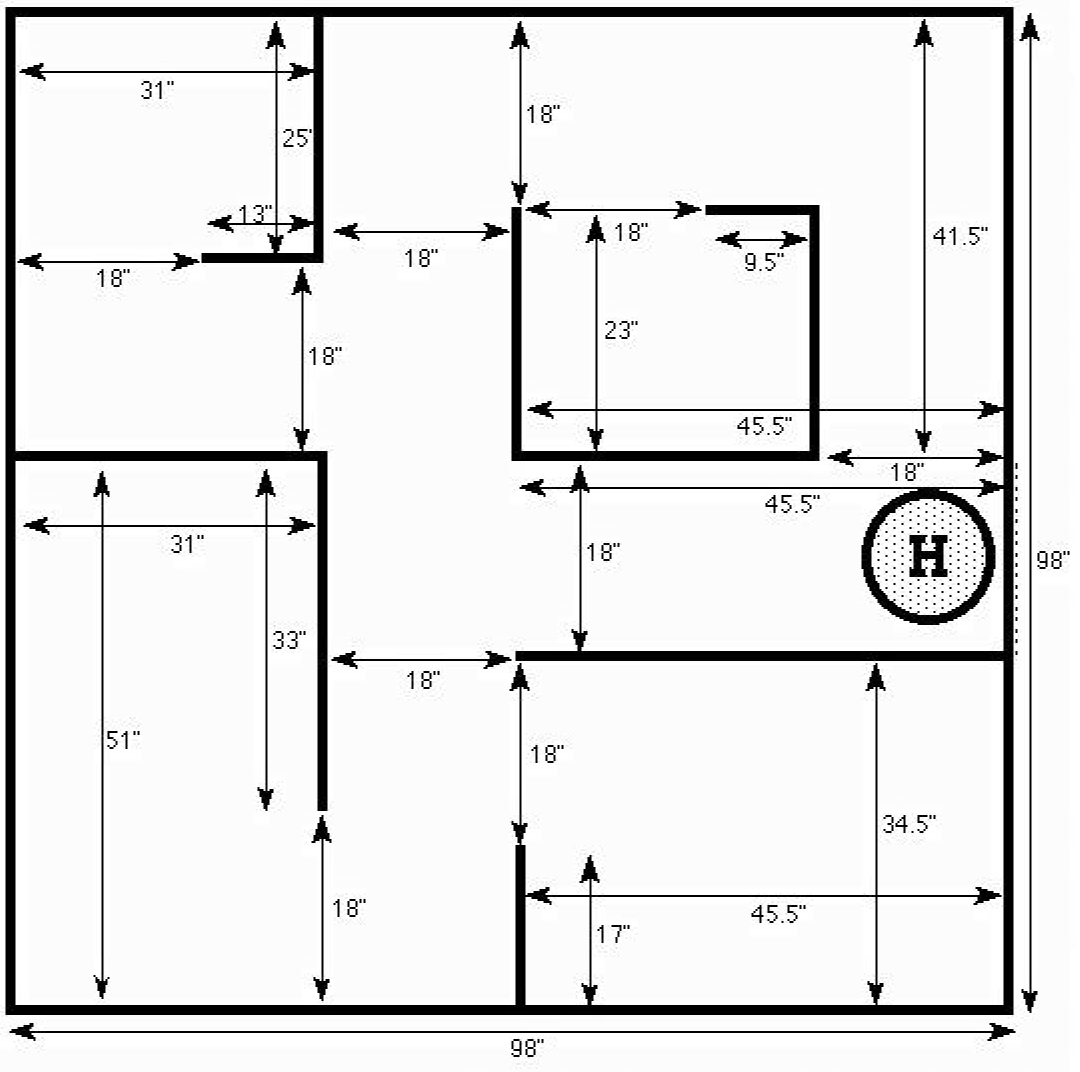


Miniature House Diagram

Trinity College FIRE FIGHTING Home Robot Contest

2000 Arena Floor Plan

Contest Rules, Attachment A

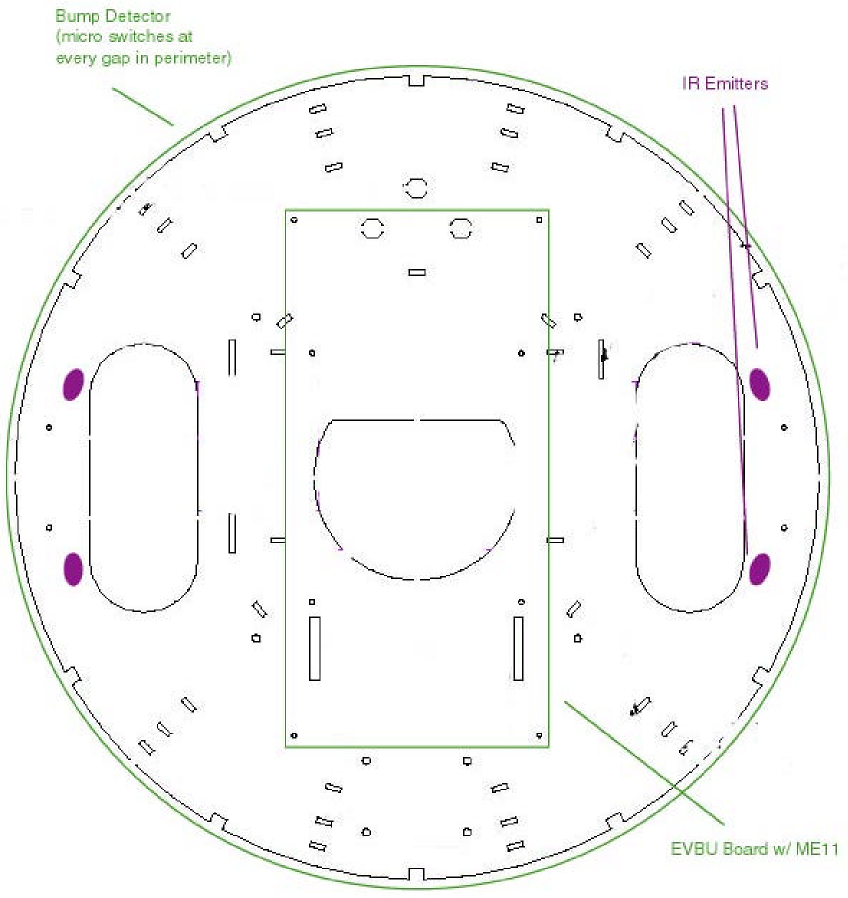


Maximum size of Robot is 12.25" x 12.25"

All walls are 13" high

These dimensions are approximations and are NOT 100% accurate and that's why the numbers don't add up exactly. Welcome to the real world!

Top View (no cover)


Bump Detector
(micro switches at
every gap in perimeter)
IR Emitters
EVBU Board w/ ME11