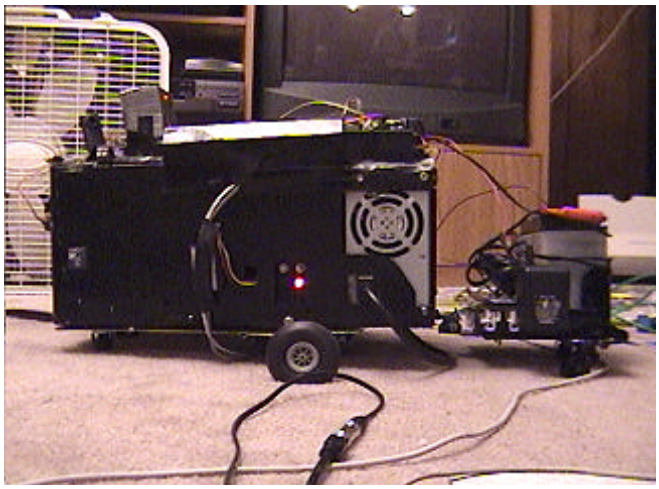
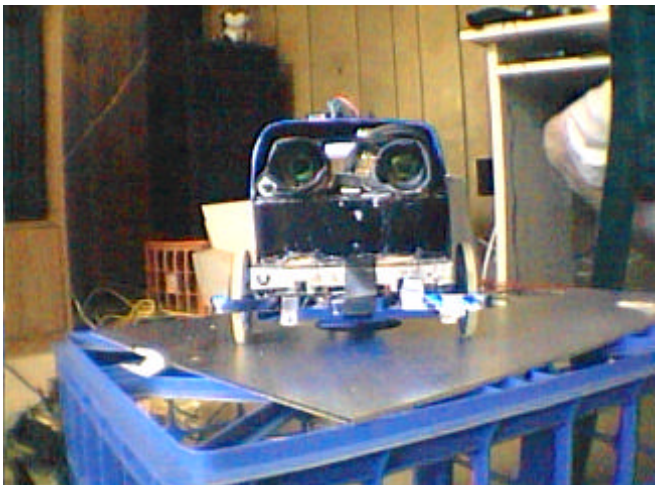
University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Lab

FLAME and SMURF
An Autonomous Fire Rescue System





12/04/2001
Patrick McGinley
Richard L. Phillips II

Table of Contents

Abstract

In this report, the detailed designs for two autonomous robots are presented. The robots were designed to perform a coordinated firefighting operation on the second-story of a model building. While the entire objective was not completed, the fire extinguishing robot is fully functional, and the other has a working machine vision system. Design goals and specifications for construction and operation are discussed. A summary of completed work is presented, as well as insight into possible future developments.

Executive Summary

The SMURF is a fully operational firefighting robot, which serves as a proof of concept of the original idea. It is tasked with locating multiple small fires within a confined area and extinguish them, and is quite reliable. The obstacle avoidance, fire locating, and fire extinguishing behaviors have all been implemented. Testing has shown that the SMURF can consistently extinguish 2 small candles within a closed 4'x8' arena in under 3 minutes. A possible application of this robot alone would be in environment containing hazardous materials. A large flat, enclosed area, such as a factory floor, would be most suitable.

The FLAME meets most of the original project goals. The level of quality achieved with the realtime machine vision system was a major accomplishment, and it is capable of detecting yellow or red objects from their surroundings, and even yellow objects in relatively low light. The vision system even includes the option to switch target colors in realtime via hardware switches. Due to time constraints, the lifting arm intended to place the SMURF into its arena was never actualized, but the majority of the FLAME was completed successfully.

Introduction

Firefighting and victim rescue usually involves serious danger to the rescuer. To help reduce this threat, this paper details a robotic system for autonomously extinguishing fires. The project entails design and development of a pair of cooperative autonomous vehicles to target a location where fires have been reported, and to find and extinguish any fires.

Integrated System

The heart of the FLAME robot is an AMD K6/2-550 processor running on an Epox MVP3-G5 motherboard. Windows Millennium Edition will be used as the operating system. The parallel and serial ports will serve as the interface for the majority of the sensors through the ADC subsystem. One or more 12V high current lead-acid batteries will power the FLAME, depending on the configuration selected. Due to the demands placed on the drive and arm motors on the FLAME, a secondary 12V power supply may run the motors to prevent draining the computer's supply during high-activity periods.

The SMURF is based on a Motorola 68HC11 microprocessor on a Mekatronix MRC11 board. The MRC11 will be paired with the Mekatronix MRSX01 sensor expansion board. SMURF will use a random wander until it finds a fire, assesses the find via a fire sensor comparison, then responds by pumping if necessary.

Mobile Platform

The current FLAME mechanical platform is made chiefly of a 1/16" steel frame partially constructed from an salvaged computer case and aluminum panels. However, the FLAME must be of adequate weight to handle lifting another robot without significantly changing its own position, and the batteries and structure will aid in balancing by acting as a

counterweight during locomotion.

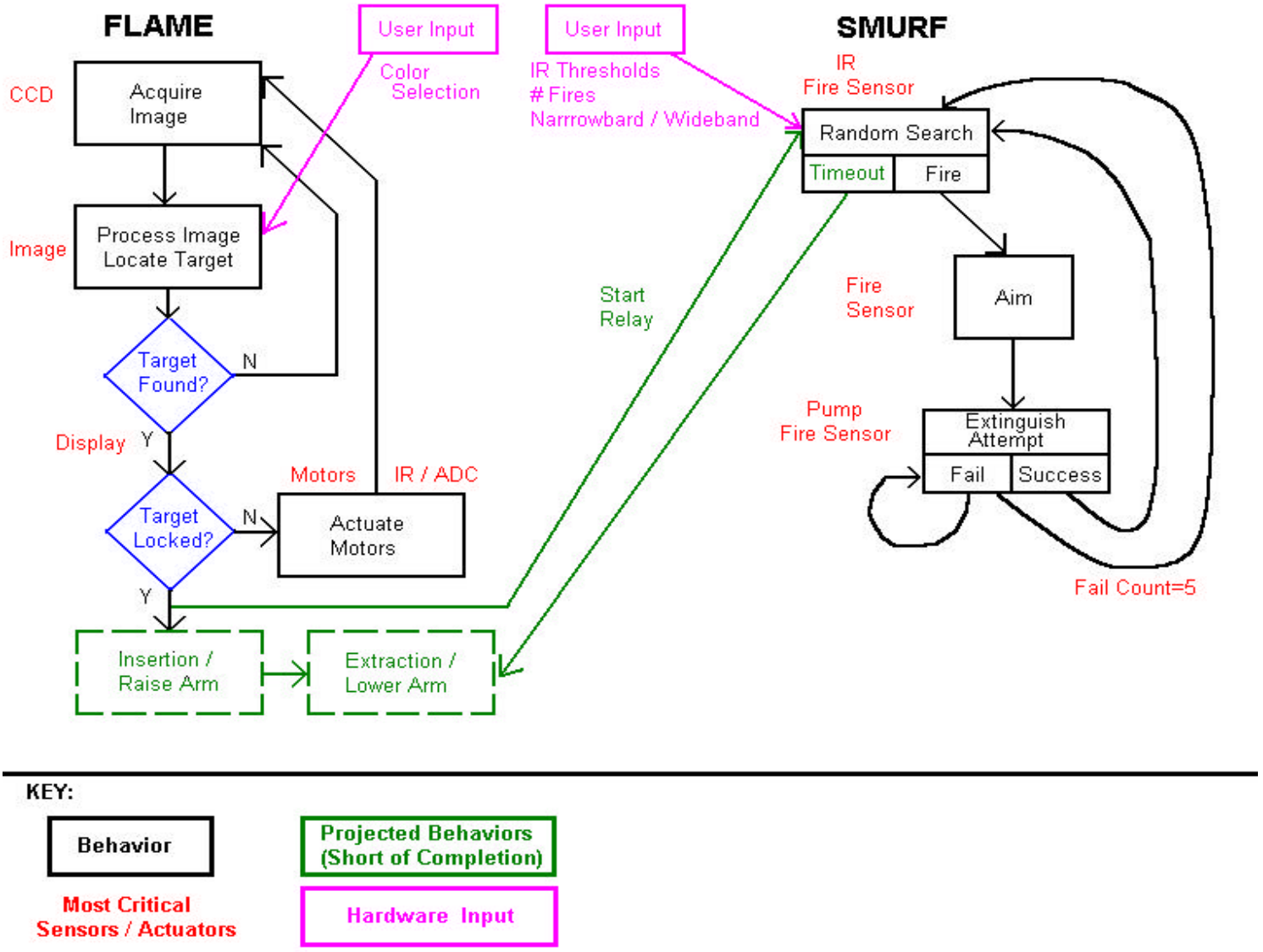


Figure 1. Robot Operation Flowchart.

The main power trailer (see Figure 2) includes the EverStart 9Ah lead-acid battery, the Solar 175W DC-AC Power Inverter, and the option to add the Pixera power supply as well. (The Pixera supply is extremely well regulated, and produces far less picture interference than tapping the ATX power supply to he camera). One or two additional Yuasa 12V 2Ah gel-cells can be attached to the top of the FLAME as a direct motor driver DC supply, or alternate leads can be run directly to the EverStart.

Figure 2. FLAME with Power Trailer.

The layout of the main components of the FLAME system are shown in Figure 3. As marked in the Figure, they are: (1) LED Object Tracking Display, (2) CCD Camera, (3) Motor Driver Box / Display, (4) Motor Driver Relays, (5) Parallel Port Output Buffer / Inverter, (6) DC-AC Power Inverter, (7) EverStart 9Ah Main Battery, (8) Pixera AC-DC supply, (9) ADC Package. Adjacent to the ADC Package but covered by the top shroud in the Figure is the additional 2Ah motor battery. All major circuit connections are displayed in the overall electrical schematic diagram, Figure 4.

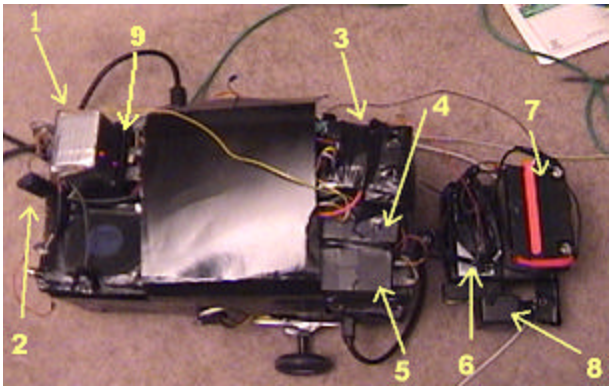


Figure 3. FLAME Component Locations.

In contrast, the SMURF, as seen in Figures 5-6, will be kept to minimum physical size possible. This will enable it to better navigate in tight areas inside the room, and reduce power consumption while active, as less energy is required to move less mass. (Such a constraint will also benefit the FLAME by reducing the the weight it would be

required to transport if the lifting arm had been completed). The main housing is a 4.75.x4.75x2.1" waterproof aluminum electrical box. An aluminum shell was constructed to protect the fire sensors, pump, and outboard electronics boxes (pump isolator circuit, sensor amplifiers), and a smooth plate was fashioned for the underside to protect the servos and wiring, as well as hold the battery.

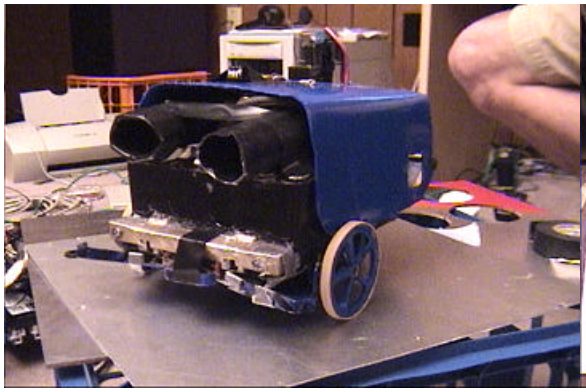


Figure 5. SMURF.

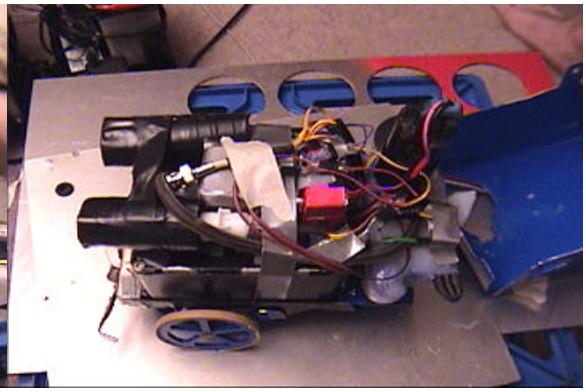


Figure 6. SMURF without outer shell.

The pump and water tank were attached to the rear of the unit, outside of the main housing but inside the outer shell.

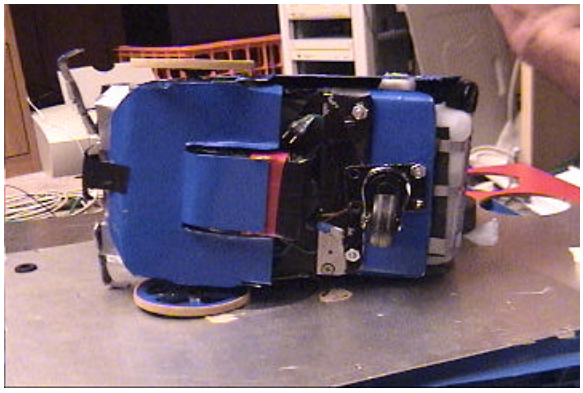


Figure 7. Underside of SMURF with skidplate.

The power supply for the microprocessor is a pack of 7 AAA NiCd cells, rated at 250 mAh, which can be mounted inside the waterproof computer housing. The servo/pump power supply consists of 6xAA NiCd, 1100mAh, mounted underneath the main housing and secured by the lower skid plate, as shown in Figure 7.

BLANK PAGE

Actuation

The FLAME will be propelled by two Mabuchi RS-540SH heavy duty 12V motors. These will be connected to the main drive wheels via double-reduction planetary gearboxes, extracted from HandiWorks HW072 cordless screwdrivers, providing the necessary output torque to drive the 20+ lb. robot, tow the power supply trailer, and several pounds of additional weight, which would more than encompass carrying the SMURF.

The SMURF possesses two main classes of actuators, a drivetrain and a pumping system. The wheels were designed in AutoCAD 14 and prototyped with the T-Tech milling machine. The wheels were cut from the 1/8" thick aluminum-backed particle board, enabling them to withstand the potentially wet environment they will be subjected to. The drivetrain will consist of two small, independently operated hacked Tower Hobbies T53 servos. These two servo motors will drive the main wheels, enabling both forward and reverse motion as well as turns in either direction. The torque rating for each servo is 42 in.-oz., with a maximum rotational speed of 272°/s (45 rpm). With 2.5" diameter wheels, this translates to a maximum linear speed of .494 ft/s (.337 mph, .15 m/s). The pumping system consists of three main components: the pump, tank, and nozzle. The 12V centrifugal pump was removed from a 1992 Pontiac Trans Am, the tank was a modified model airplane fuel tank, cut down to a capacity of approximately 8 fl.oz. (6 fl.oz. usable by the pump), and the nozzle was taken from a 1991 Buick Skylark. The pump was able to operate at much less than its rated voltage, down as far as 4.5 volts, so it was powered from the SMURF unregulated servo battery supply. It had more than adequate pumping volume, as it was able to empty the tank in under 5 seconds with no nozzle restricting the flow. The nozzle was not the optimal design, as it employed a flat,

triangular spray pattern roughly 30° wide. While the horizontal profile was excellent, the lack of vertical distribution meant that without external actuation, it had a limited linear range from the target in which it was effective. In the later designs, a choice was made to focus the spray at a distance of 3-6", in order to coordinate it with the range of the fire sensors when locating small candle flames.

Multiple modes of travel were possible for the SMURF. Rotation was fairly accurate, at approximately 3° per millisecond of turn at the midpoint of the pulse width range ("50% speed"). In addition to forward and backward motion, both gentle turning and spins in place were implemented in software.

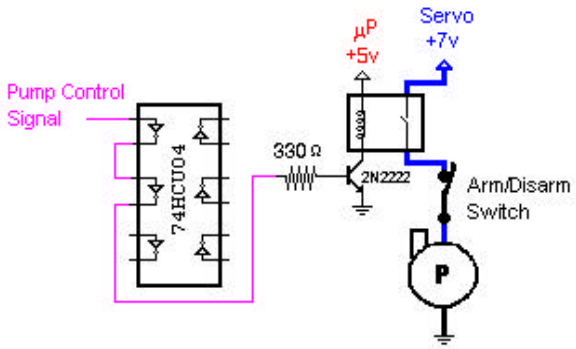


Figure 8. Pump isolation and control circuit.

The pumping function consisted of two main components. The first was a prime delay, and the second was a slight rotational twisting to provide better spray coverage than firing while still. Since the pump was powered through the Servo battery pack but triggered by a digital output pin of the processor, an isolation circuit, depicted in Figure 8, was used to separate the two power systems. It was also equipped with a manual arm/disarm switch, to force the pump off while downloading code or troubleshooting.

Sensors

Scope / Objectives - SMURF

The unique function within the SMURF sensor subsystem is fire detection. The sensor chosen for this purpose is the United Detector Technologies Precision Silicon Photodetector (P/N 14-00-003, PIN #10AP). The sensor, shown in Figure 9, claims to match the CIE response curve within ±2%, giving the robot an element of 'human' type perception. It reacts to both color and brightness changes, which can be exploited for a wide range of applications. However, the current focus is only on the detection of a flame and differentiation of the flame from other classes of objects.

Figure 9. UDT Silicon Photodetector.

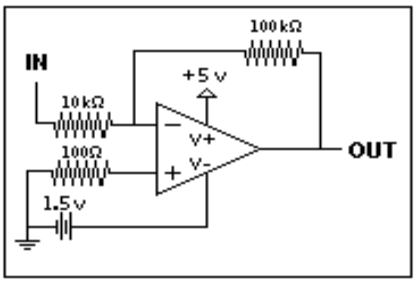
Electrically, the sensor is best used as a voltage source, functioning in a similar fashion as a photo-transistor. It's output voltage range is approximately 0.1-(-0.5) VDC, with light sources occupying the -0.15V to -0.5V range, and non light-emitting objects are generally between 0.1V and -0.15V. An inverting amplifier was designed to create an appropriate input voltage range to interface with the 68HC11 processor and MRSX-01 expansion board.

The photodetector has an active light reception area of 1 cm^2. The rise time of this sensor is typically 1µs, which is far superior to the lag time experienced with CdS cells. The operating temperature range is from 0°C to 70°C. The data sheets for this sensor are included in Appendix 6.

Experimental Layout and Results - SMURF

The initial investigation into the response of the silicon photodetector was done by measuring the output voltage when aiming the detector at a light source. It was most

responsive to fluorescent lights, followed by halogen and then incandescent bulbs. For small flames and poorly lit objects, very low output voltages (on the order of -.02 V) were observed. Since the ADC on the robot is only 8-bit 0-5V, yielding .02V resolution, it was insufficient to simply sample directly from the sensor. An inverting amplifier was designed to expand the useful range of the sensor. The circuit is shown in Figure 10. It employs a gain of K= -10, and by using the +5V regulated supply from the MRC-11, limits the maximum value to a +5V output.



Quantitative tests were performed using the dSPACE data acquisition system. The block diagram is shown as Figure 11. The test involved the UDT photodetector mounted on the SMURF and the amplifier circuit. Four sets of experiments were performed, using both high and low fluorescent background light levels in combination with both tall (1¾") and short (~¼") flames from a butane lighter. The flame was moved linearly away from the sensor starting at a distance of 1" and ending at a distance of approximately 18" over the course of 20s. Each set of trial conditions was repeated three or more times, and the results are plotted in Figures 12-15.

Figure 12 displays a test in which a large flame was placed in front of the flame sensor and moved from directly in front to 18 inches away from the front of the robot. The test was repeated several times and Figure 12 shows a representative selection of those tests. The trials show a very high degree of repeatability. This graph shows that a threshold of 1.75 volts can be used to eliminate a high amount ambient light and still detect a flame at 6 inches.

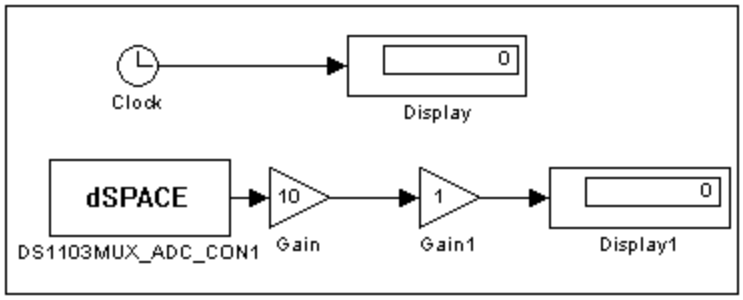


Figure 11. Simulink Block Diagram

In the next test, Figure 13, the fluorescent lights that were directly overhead were shut off, which reduced the amount of ambient light. This reduced the ambient reading off the photodetector to approximately 0.7 volts. This allows for the threshold to be lowered to 1 volt and still be able to detect flames at 6-8 inches. These trials once again show the repeatability of the sensor.

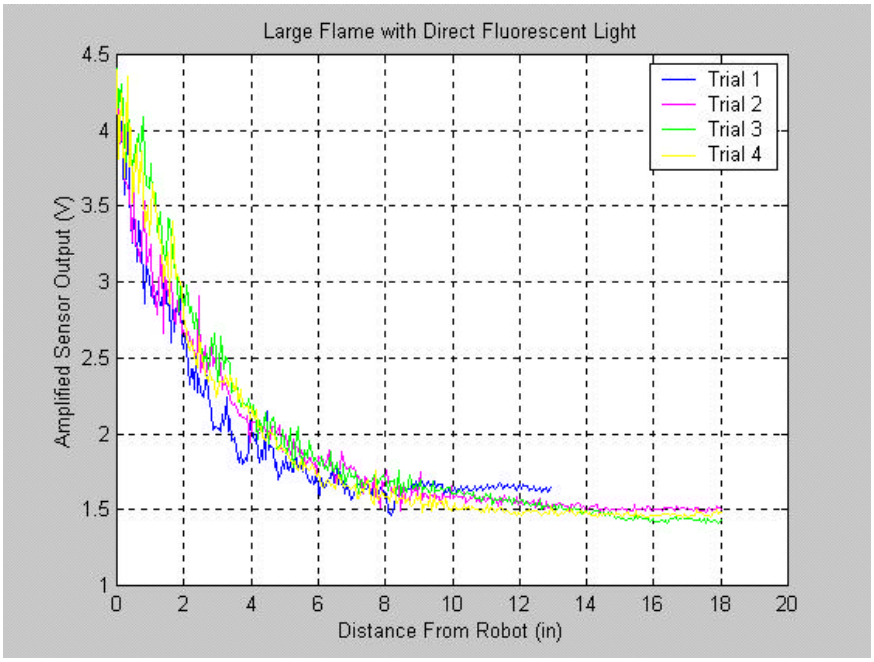


Figure 12. Experimental Data for Flame Sensor.

For the next set of experiments the flame was reduced to a small flame. Figure 14 shows the test run with overhead fluorescent lights on. The flame is still detectable at 6 inches with the threshold set at 1.5 volts. This shows that the detector should be able to

pick out a relatively small flame almost as easily as the larger flames.

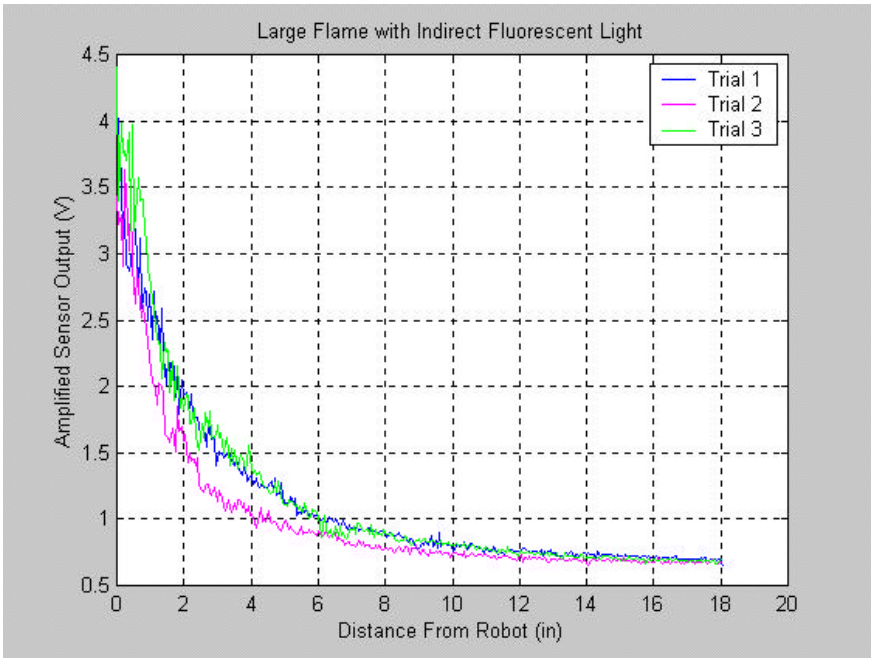


Figure 13. Experimental Data for Flame Sensor.

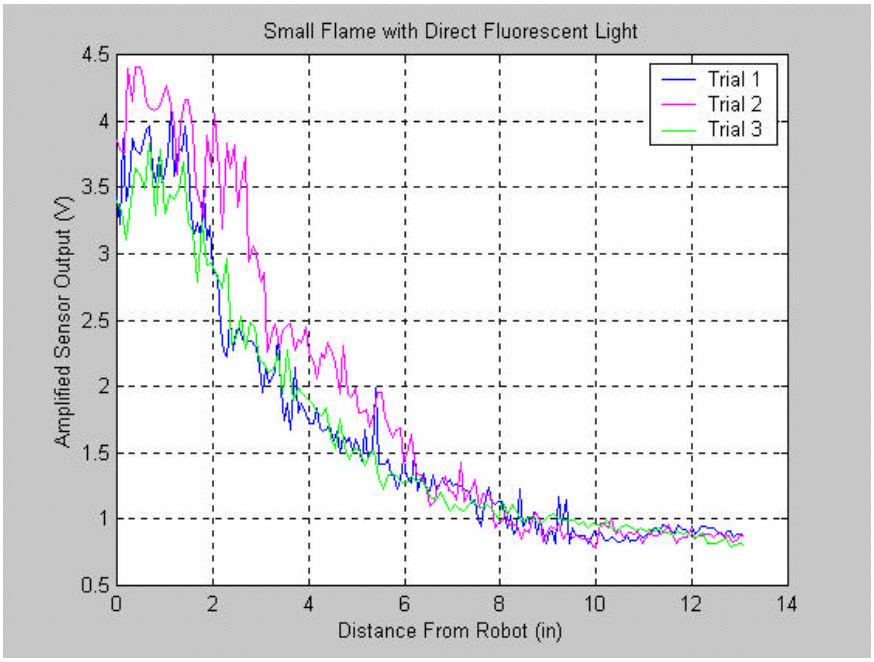


Figure 14. Experimental Data for Flame Sensor.

In the next experiment, a small flame was offset by 10° from the straight forward line, Figure 15. When the sensor is offset the response falls at a greater rate than the direct line trials. However the flame can still be picked up at 4 inches. Because of this data, two

photodetectors will be used together to increase the data gathering capabilities of the robot.

There was an exponential relationship between flame distance and output voltage, but it seems quite reasonable to apply a linear estimation. One potential linear model chosen to approximate the longer-distance section of the curve, which would be used in noticing the flame while searching. A second linear model that would be more appropriate for close-range targeting after the flame has been seen.

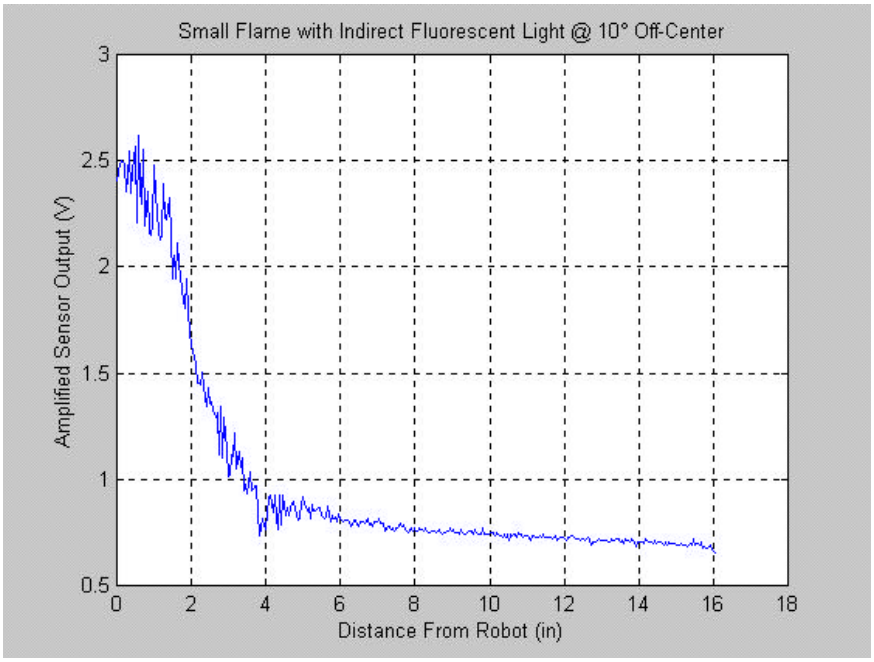


Figure 15. Experimental Data for Flame Sensor.

The use of the amplifier greatly increases perceived sensitivity, and allows high-intensity light sources such as fluorescent, halogen, and incandescent bulbs (that are much brighter than the target flame) to reach a saturation value. By not preserving the full range of the detector, more accurate differentiation can be done within the lower, more linear range consisting of non light-emitting objects and low-intensity light sources such as candles or LEDs.

Scope/Objectives - FLAME

The primary task of the FLAME robot is to locate the building and place the

SMURF robot inside. However, locating the building is a nontrivial task. The robot will undertake this mission with the aid of a machine vision system, capable of discerning objects of specific colors from the remainder of the room. The main components of the system are the Pixera PXG-150N-PH CCD NTSC video camera (Figure 16), and the Cybertainment CybermailAV PCI video capture board, based on the Brooktree BT878 capture chip.

Figure 16. Pixera Camera.

The Pixera CCD camera is a ¼" pinhole lens 512x492 element CCD sensor and has a stated horizontal resolution of >330 TV lines. The resolution claim seems to be accurate, but the effective resolution is somewhat hampered by visual noise in the signal. The color depth is decent compared to other mini CCD cameras. Since color saturation and brightness can be adjusted through the software drivers of the BT878, repeatability of color shades is more important than their absolute appearance. The camera requires no more than 200mA at approximately 5VDC, and its video output is a standard NTSC composite signal.

The CybermailAV is a very basic BT878 video capture card, consisting of little more than the capture chip, the PCI interface, and three analog video inputs (2x composite video, 1x S-Video). It interfaces to the motherboard via the PCI bus. Its drivers are essentially the Brooktree reference drivers, so the performance was solid and stable though unspectacular. Drivers were only available for the Microsoft operating systems based on the Win9x kernel.

The second uncommon sensor package element on the FLAME is the ADC subsystem. It is based on the Maxim MAX118 1Msps, 8-channel, 8-bit parallel ADC chip. The purpose of the device is to allow analog input to the FLAME's PC via the parallel port. The MAX118 is essentially a complete solution in a single chip (The only external part necessary is a .1μF capacitor across the power supply).. It uses the parallel port's STATUS register for input, and the CONTORL and STATUS registers for control. Since it requires only one initialize input, three address inputs, and one output to control sampling and return the acknowledge signal, $\sim C_2$ and $\sim S_7$ were used. The STATUS register only returns the 5 MSB's to the PC, so that only the 4 MSB's of the conversion are available to when using a single parallel port. The ADC's D_7-D_4 are connected to S_6-S_3. To initialize a conversion, the address was placed on the CONTROL register's $C_3C_1C_0$ lines. The $\sim C_2$ bit is asserted (low) to initialize a sample, and the ~ACK signal is received on $\sim S_{7..}$ By averaging the data over 100 or more samples, it was found to be quite stable and repeatable. Code for the ADC sampling is included in Appendix 4.

Experimental Layout and Results - FLAME

The testing of the FLAME vision system was somewhat subjective, due to its implementation, and the range of variables involved in such a complex problem. The output of the system is also more difficult to measure than just a simple voltage or resistance. However, since the purpose of the vision system on the robot is for navigation more than for object identification, locating the object is of primary importance. Thus, by placing various targets in different situations and comparing their location as determined by the algorithm to what a human would choose were they asked to find the object, the algorithm can be evaluated.

The first iterations of the vision algorithm were based on separating the red, blue, and green color planes, and analyzing them separately to find certain objects. But it was found through experimentation that the camera was most responsive to bright colors, particularly yellow and orange, and fairly responsive to blue, then green and red to a lesser extent. An example of the bright-color response is shown in Figure 17, a picture captured and then all colors below an average brightness threshold of 65% are changed to black.

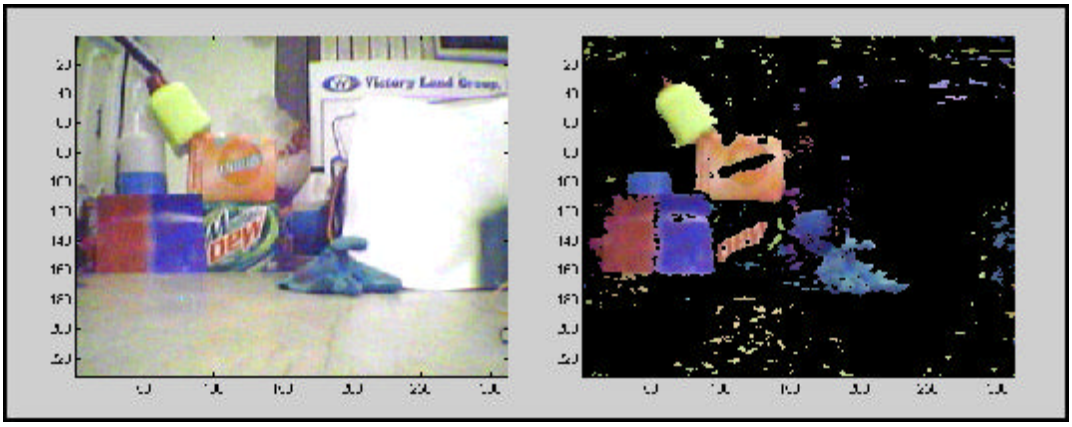


Figure 17. Bright Color Recognition.

The following image processing algorithm was developed

- Input the image file
- Remove pixels not meeting the criteria for 'yellowness'
- Create a 2-D array representing 'yellow'=1 or 'not yellow' =0 elements in the image
- Perform a weighted average of this array with its 4 nearest neighbors
- Find the weighted area-averaged center point of the 'yellow' pixels
- Return the center point's coordinates

Currently, there are two different criteria for 'yellowness' that were determined experimentally to yield useful results. They are based on the 24-bit RGB color representations, in which each color is realized as a combination of Red, Green, and Blue components with values ranging from 0-255.

Table 1. Color Criteria by RGB Components.

Color	Red	Green	Blue

Yellow	>200	>200	<170
	>120	>120	<90
Low Light Yellow	>120	>120	<90
	>100	>100	<60
Red	>135	<85	<100
	>180	<140	<150
Blue/Green	<140	>145	>180
	<170	>180	>150

Any pixels not meeting either criterion, Table 1, are ignored in finding the center point of the targets. For every pixel that does comply with either definition of yellow, a value of 1 is entered into the 2-D array. Each pixel in the interior of the array (excluding the edge pixels, where camera noise often makes the data unusable) is redefined as the weighted average of R times itself and each of its vertical and horizontal neighbors.

$$A_{i,j} = \frac{(R*A_{i,j} + A_{i-1,i} + A_{i+1,j} + A_{i,j-1} + A_{i,j+1})}{(R+4)}$$

Then, an weighted area-average is calculated. Each A_{ij} term is squared, so that the <0,1> range is preserved, but points close to other yellow points retain high coefficients, and the effect of isolated points is minimized. Then, the x- and y-coordinates are multiplied by these weighting coefficients, summed, and divided by the total number of nonzero points used. The resulting average (x,y) is then the center of concentration of yellowness.

The current algorithm, which is being developed in MATLAB^{TM}, is fairly slow. It can completely process a 176x144 (QCIF PAL size) image in approximately 8-12 seconds, and an 80x60 image in roughly 3-4 seconds. There is some improvement in accuracy of the concentration center coordinates using the larger picture, but it may not be justified by the processing time required. Additional accuracy was also gained by blurring the points in the original image before applying the yellowness criteria, which helped further reduce isolated point noise problems, but it increased overall processing time by almost 35%, so it

was not considered worthwhile.

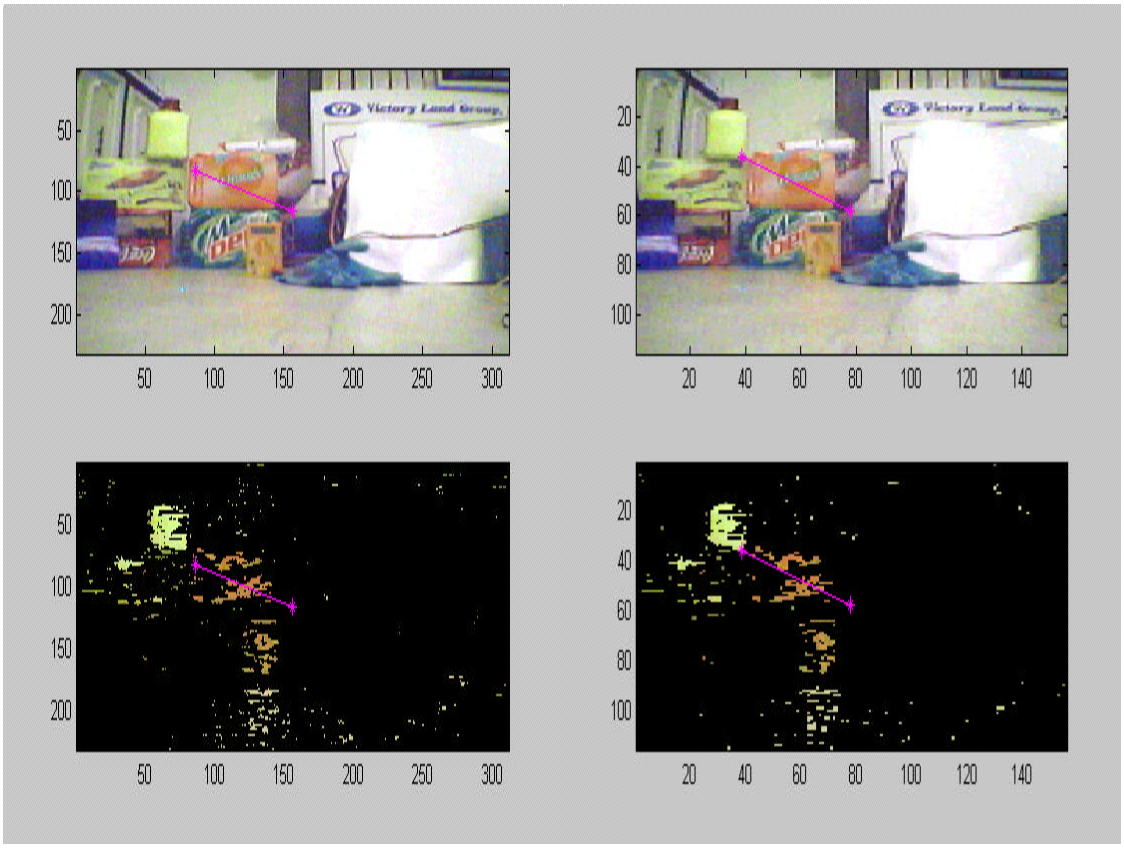


Figure 18. *320x240 Resolution, Multiple Targets.*

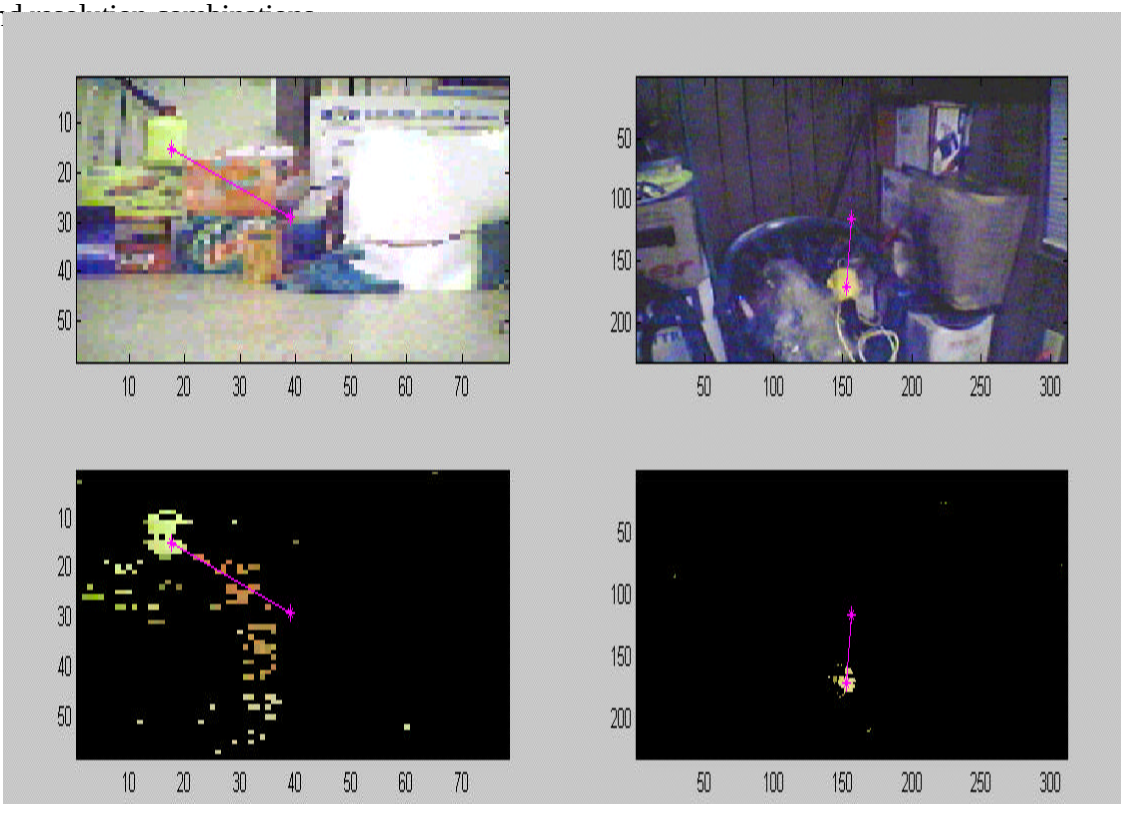


Figure 19. *174x144 Resolution.*

Figures 18 through 21 show some representative results from various color, light, and resolution combinations.

Figure 20. 80x60 Resolution *Figure 21. Single Target with Low Light.*

The previous figures showed results obtained from the code written in MATLAB™. This program provided the framework for the final algorithm code that was written in C++ (attached in Appendix 2). The actual video capture portion of the FLAME software was derived from a program written by Vadim Gorbatenko, and the original first order edge detector portion of his program was replaced by the FLAME image processing and control routines. This increased the capture and processing rate from approximately 35 seconds per frame in MATLAB™ to 1/10 of a second per frame in C++. Figure 22 through 25 display the results obtained from the C++ version of the program under normal lighting conditions.

Figure 22. Original Picture .

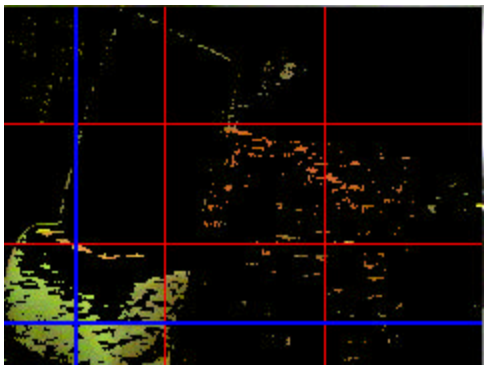


Figure 23. Yellow in Normal Mode

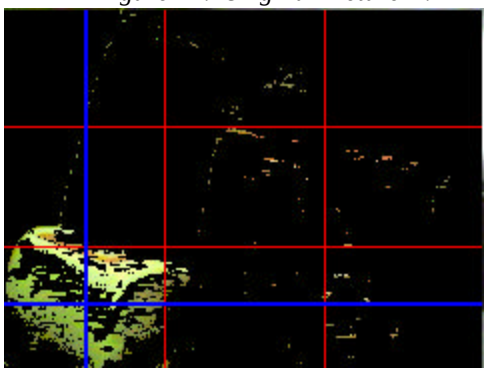


Figure 24. Yellow in Low Light Mode.

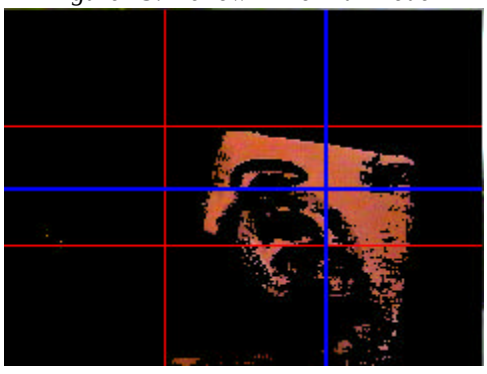


Figure 25. Red Target Mode.

Experiments were then conducted to test the algorithm in a very low light situation. Figures

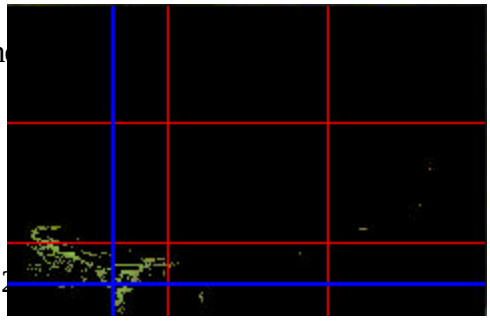


Figure 26. Original Picture.

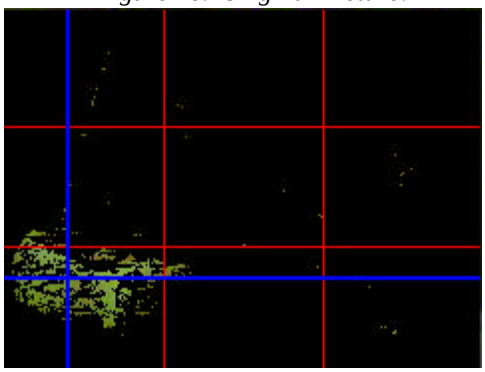


Figure 27. Yellow in Normal Mode

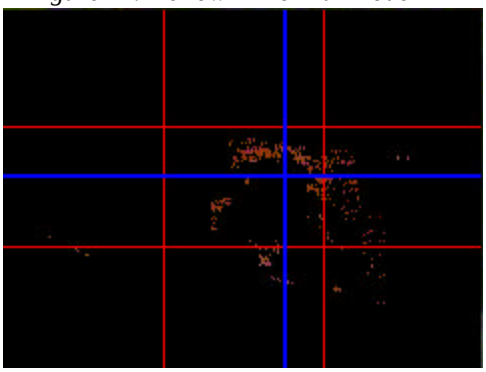


Figure 28. Yellow in Low Light Mode.

Figure 29. Red Target Mode.

Behaviors

The primary objective of the FLAME robot is to correctly place the SMURF robot correctly into the second story window and remove it after it has completed its mission. In order to accomplish this goal, it is imperative that the FLAME can accurately locate the window and position itself. The first step is to find the building. The robot will spin counter-clockwise until it detects a large amount of yellow via its onboard CCD camera.

Once FLAME's vision system detects the general direction of the building, it enters a more precise targeting mode, where it attempts to properly align itself in front of the window. The color contrast between the yellow facade and the background room colors provides the necessary amount of information to locate the window. The image is then processed onboard and the image is separated into RGB components. The color components of each pixel is analyzed individually.

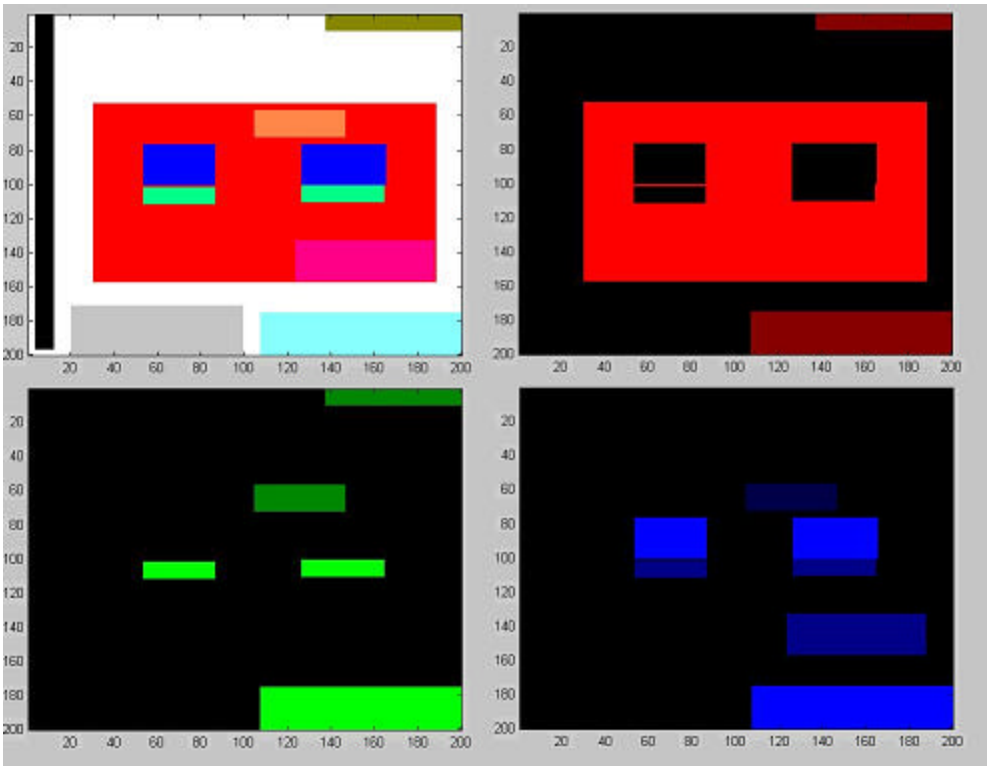


Figure 30. Image Processing of Theoretical Target into RGB components.

The vision program performs a comparison of the three color components of each pixel, and if the values are within a certain threshold, indicating a shade of black, white, or gray, all three components are set to zero so that neutral colors will not be mistaken for the target area. Examples using both an actual picture from the CCD camera and a 16-color theoretical representation of the building are shown in Figures 26 and 30. IR ranging is also be used in conjunction with the vision system to help position the robot in front of the target. Collision avoidance and bump sensing are included in the positioning and ranging behavior.

Using the image processing algorithm previously discussed in the sensors section, the field of view is broken up into a three by three grid. Where the area average is in the grid determines the movement that FLAME should execute. An LED box, with four LED's in a square, is also included with this system to visually display the average location. The two LED's in any one direction indicate the grid direction that the average is

located in. When the average falls into one of the upper three sections, top two LED's, FLAME backs away from the target and when the average is inside one of the lower three sections, bottom two LED's, FLAME proceeds forward toward the target. If the average is inside one of the left three grid sections, left two LED's, a left turn is commanded and a right turn is commanded for the three right sections, right two LED's. When the average is in the center section, all four LED's, this indicates that FLAME is lined up and commands a stop in motion. When two of the conditions are met at the same time both sets of LED's light and FLAME executes the corresponding two direction commands.

The initial goal of FLAME was to insert and extract SMURF from a building. Due to time constraints, the lifting arm part of the chassis was not completed. The following describes the proposed assembly and its characteristics. The lifting arm will be equipped with sensors enabling it to accurately determine if a safe insertion of the SMURF can be accomplished. The goal is to ensure the safety of the SMURF by preventing the arm from being extended into the facade of the building or by allowing the SMURF to deploy prematurely, potentially resulting in a catastrophic fall.

Triggered by the directional switch on the platform atop the arm, the FLAME will begin to extract the SMURF from the building, and return to search mode. Some direct communication between the robots may be necessary in addition to the switch.

The purpose of the SMURF robot is to seek and extinguish fires. Once a flame is targeted SMURF extinguishes the fire using an onboard water cannon. This cannon consists of a simple relay-operated windshield-washer pump, which will draw from the onboard water reservoir, and pump it through the fixed nozzle.

Instead of using the *wait* function included with Talrik library, the *observe* function was written. The *observe* command includes the *wait* command but also reads sensors and

can trigger collision avoidance and firefighting modes. Essentially the *wait* command is broken up into 25ms segments with sensor reading in between the segments.

Two direction changing commands are included in program code for SMURF, *spin* and *turn*. In the *spin* command one wheel rotates forward and the other wheel rotates backward. This allows SMURF to maneuver in tight areas and to rotate in place to avoid close obstacles. The *turn* command keeps both wheels rotating in the forward direction but slows one wheel down to allow SMURF to travel in an arc instead of straight lines. This helps to avoid obstacles that are at a greater distance and do not require an immediate and drastic change in direction.

Three switches, Table 2, are included as user interface to change certain program variables. The first switch a narrowband versus wideband (default) flame detection threshold setting. The narrowband setting moves the upper and lower bounds closer together to help eliminate extraneous fire detections. The second switch controlled the number of fires SMURF looked for. In one position SMURF looks for four fires (default) and in the second position 14 fires. The third switch controls the IR levels for collision avoidance. The default position is a high IR threshold which helps in collision avoidance. The second position is a reduced IR threshold which allows for navigation in smaller areas.

Table 2. Switch Position Chart.

Switch	Position 1	Position 2
1	Wideband Flame Detection	Narrowband Flame Detection
2	4 Fires	14 Fires
3	Far IR Threshold	Near IR Threshold

The following is the desired behaviors of SMURF for when the lifting arm assembly is completed. If a period of time passes were SMURF finds no fire and no victims, then SMURF will be programmed to return to FLAME and start the extraction process. The SMURF would have the capability to load itself onto the lifting arm of

FLAME and signal FLAME to initiate the retrieval process. The FLAME will then extract SMURF from building, thus completing their missions.

Experimental Layout and System Results

After installing the hardware on the SMURF platform a flat obstacle was placed at various distances from the IR emitters/detectors. From these readings, preliminary IR threshold values were obtained for collision avoidance routines. For testing purposes, a temporary 4'x8' arena was set up, similar to that shown in Figure 31. SMURF was placed in the arena, and those values were tested under real conditions and corrected as necessary. The same procedure was followed to setup the UDT photodetectors to detect various flames. The sensors respond more willingly to large, bright flames, but for safety purposes, small candles were used to simulate fires.

After sensor calibration was completed, a preliminary obstacle avoidance and firefighting program was written, and two candles were placed at random locations in the arena. After several test runs, it was determined that SMURF can regularly extinguish one candle in one minute and two in approximately three minutes. It was also noted that SMURF had difficulty navigating passages less than twice its major dimension, or approximately 18 inches.

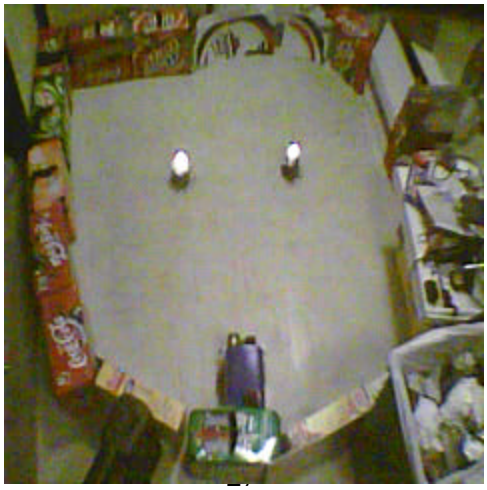


Figure 31. SMURF in Arena.

Using quick charged battery packs, (Servos: 6xAA NiCd, 1100 mAh, charged 12V @ 2A for 4 min, and µP: 7xAAA NiCd, 250 mAh, charged 12V @ 1A for 3 min) runtimes of over 30 minutes were routinely observed. This also included a minimum of 25 pumping cycles (though water tank refills were required after approximately 8-12).

As previously mentioned, the FLAME vision system was tested in several lighting conditions - incandescent, natural, and fluorescent, as well as bright and dark levels of each. The color differentiation criteria listed in Table 1 were also adjusted slightly from the original approximations based on feedback from these experiments.

The yellow targets were the most obvious to the camera, even in low light. The red was not as readily recognizable as yellow, but the image processor was still able to successfully detect it under most conditions. The poorest of the color choices used was blue/green, in part due to the color response of the particular camera used. In lower light, the automatic gamma and hue adjustment tended to make the entire picture bluish, and bright blue and green were rarely seen.

Conclusion

Overall, the SMURF / FLAME project has been fairly successful. A pair of robots that can be used for firefighting were designed and prototyped. Both met the majority of goals originally set forth in the original proposal, with the exception of the communication aspects. The SMURF can be started by closing a switch (or relay), which was part of the original concept, but the insertion and extraction from the building were not completed.

The SMURF is able to detect and extinguish small fires in a relatively short period of time using a random search. Once the fire is found, the targeting procedure to pinpoint the fire is generally accurate, the only exception being when a fire is detected just on the

outside edge of the field of view of one flame sensor, when it is possible that the SMURF will turn away from the fire rather than toward it. The successive approximation procedure that is used before the water is pumped, along with the wiggling action while pumping, usually results in an effective and efficient spray profile. The SMURF can navigate a small (4'x8') arena and extinguish multiple fires within it in just a few minutes, and was not observed to become trapped in any limit cycles.

The chief drawbacks of SMURF are the limited field of view of the fire sensors, (about 10°), the lack of a wall following behavior, and accuracy of the nozzle. Adding more fire sensors aimed in different directions would help remedy the field-of-view problems. Wall following could be implemented in software, pending the addition of side-facing IR detectors. The nozzle should be articulated, and possibly replaced by one with a circular spray pattern instead of a wide horizontal one. Vertical aiming would likely be enough, as the horizontal aiming could be achieved by turning the robot itself.

The FLAME was a mobile AMD K6/2-500 (a Pentium II class) computer, with onboard video capture and realtime processing capability at >10fps. It was able to track and follow yellow objects in greatly varying lighting conditions, and red objects in most normally lit situations, and could be switched between these modes in realtime as well. The vision system far exceeded expectations, especially in poor lighting, and its speed was also quite impressive.

The FLAME also has a few notable limitations. First, with its power consumption, it needs fairly large batteries. The current configuration requires that the 9Ah lead acid battery be trailered behind the robot due to weight concerns. A DC-DC converter would also benefit power conversion efficiency greatly. The current system uses a 175W DC-AC inverter serving the ATX power supply. (While not efficient, it was a cost effective

solution). A faster processor would allow for an increased frame rate for the vision algorithm, which would smooth out turns and benefit object following performance. Mounting of the planetary gears from the cordless screwdrivers was difficult, as there is only a pin holding them in their normal application. A capture card with Linux or Windows 2000 driver support would be a great benefit, in terms of overall system stability.

Documentation

Engdahl, Tomi. "Parallel Port Output." 1996-2000.
http://www.hut.fi/Misc/Electronics/circuits/parallel_output.html

ePanoram.net (ELH Communications, Ltd.) "PC hardware projects page." 2001.
http://www.epanorama.net/project_pc.html#pc_parallel

Gorbatenko, Vadim. "CFrameGrabberClass for C++ Source Code."
www.codeproject.com

Harries, Ian. "Interfacing to the IBM-PC Parallel Printer Port." 01-26-1998.
http://www.doc.ic.ac.uk/~ih/doc/par/

Hauppauge Inc. "Hauppauge WINTV PCI Frequently Asked Questions." 2001.
http://www.hauppauge.com/html/faq.htm#MOMLIST

Acknowledgements

The SMURF / FLAME project team would like to thank the following individuals for their contributions to the project-

Prof. A. Antonio Arroyo, Scott Nortman, and Aamir Qaiyumi:

> For help and guidance throughout the semester...

Jonathan Gamoneda:

> For help troubleshooting the OC pins on the defective processor, since mechanical engineers and assembly language do not work together very well...

Dr. Carl Crane III, Mechanical Engineering Dept:

> For lending us a more than adequate 486 DX50 laptop to use as a programming / debugging station for SMURF...

The Phillips and Bratton Families:

> For nearly endless supplies of scrap aluminum and old computer parts...

Anthony Hinson:

> Whose semi-portable briefcase computer inspired the basic FLAME entire-K6/2-in-a-small-box design...

Appendix 1

SMURF

· Program Code

```
#include <stdio.h>
#include <tkbase.h>

#define PUMP_ADDRESS 0x01
#define PRIME 220                                   // pump prime delay time to add to pumping duration  **estimated**
#define PLUGGED_IN 0x00                 //NOT USED
#define IR_LEFT IRDT[3]
#define IR_RIGHT IRDT[6]
#define FLAME_SENSOR_1 IRDT[0]
#define FLAME_SENSOR_2 IRDT[2]
#define FLAME_SENSOR_3 IRDT[10]
#define FIRE_TOL 8
#define RAND TCNT&0x01
#define DIG_OUT *(unsigned char *)(0xffb9)

//Globals for sensors / control

int empty=0;
int speed=100;                                      // set speed in main or in functions
const int turnconst=3;                  // degrees per milisecond of turning,   **estimated**
int iri=0;
int irl[10];
int irr[10];
int ir_left=80, ir_right=80;
int firefind=0;                                     // 'if fire found' =1, no fire =0
int firesout=0;                                     // number of fires extinguished
int count1=0;
int count2=0;
int RBUMPER=0;
int FBUMPER=0;
int FLAME_OLD=0;
int FAIL_COUNT=0;
int FIRE1=0;
int FIRE2=0;


int FIRE_NUMBER=7;                  // number of fires to find
int MED_THRESH=115;     // ir thresholds
int NEAR_THRESH=124;    // ir near
int FLAME_THRESH=57;    // fire sensor 'detect' threshold
int FLAME_THRESH_HI=255;// fire sensor 'no detect' threshold

//                                          _____________________________
//                                          _____________________________
//
//                                                  Prototypes
//                                          _____________________________
//                                          _____________________________

void observe(int);
int pump(int, int);
void straight(int);
void stop();
void turn(int , int , int);
void spin(int);
void blindspin();
void irsample();
int search();
int ir(int);
void fireavg();

//                                          _____________________________
//                                          _____________________________
//
//                                                  Functions
//                                          _____________________________
//                                          _____________________________

//  ____________________________________________________________
// |                                                            |
// |    OBSERVE - Wait with sampling command                    |
// |____________________________________________________________|

void observe(int duration){
                int n=1;
                while (duration>25){
                                duration=duration-25;
                                read_IR();
                                RBUMPER=rear_bumper();
                                FBUMPER=front_bumper();
                                wait(25);
//new entry
                                if (firesout<FIRE_NUMBER){n=1;}
                                if ( ((FLAME_SENSOR_1 > FLAME_THRESH)&&(FLAME_SENSOR_1 < FLAME_THRESH_HI))|| ((FLAME_SENSOR_3 >
FLAME_THRESH)&&(FLAME_SENSOR_3 < FLAME_THRESH_HI))|| ((FLAME_SENSOR_2 > FLAME_THRESH)&&(FLAME_SENSOR_2 < FLAME_THRESH_HI)) ){
                                while ((n == 1) && (FAIL_COUNT < 5)){
```

```
                                                printf("\t FIRE...FIRE...FIRE!!!!!\n");
                                                fireavg();
                                                n=fire();
                                                if (n == 0){
                                                        firesout++;
                                                }
                                                if (n == 1){
                                                FAIL_COUNT++;
                                                }
                                        }
                                }
//end new entry

                //simple avoid code here ----------------------
                }
                wait(duration);
                read_IR();
                RBUMPER=rear_bumper();
                FBUMPER=front_bumper();
                return;
}


//  ____________________________________________________________
//  |                                                           |
//  |    PUMP - Pumps water for specified time period           |
//  |___________________________________________________________|

int pump(int duration, int range){
//              if (!empty){ return empty;}
//              else{
                        DIG_OUT=0xf3;
                        duration=duration*10+PRIME;
                        // PRIME s to prime + duration in hundreths of seconds
                        //wait(duration);
                        while (duration>750){
                                wait(250);
                                blindspin(11);
                                wait(80);
                                blindspin(-20);
                                wait(80);
                                blindspin(15);
                                wait(80);
                                blindspin(-10);
                                wait(80);
                                blindspin(5);
                                wait(80);
                                duration=duration-750;
                        }
                        if (duration>0){
                                wait(duration);
                        }
                        DIG_OUT=0x03;
                //check if empty....
                        return 0;
//              }
}

//  ____________________________________________________________
//  |                                                           |
//  |    STRAIGHT - Drives robot 'straight' at given speed      |
//  |___________________________________________________________|

void straight(int newsp){
                speed=newsp;
                servo(0, 3000-20*speed);
                servo(1, 3000+20*speed);
                // * could add correction factor to speed of one in reverse if not straight.....
                return;
}

//  ____________________________________________________________
//  |                                                           |
//  |              STOP - Stops robot                           |
//  |___________________________________________________________|

void stop(){
                speed=0;
                servo(0, 0);
                servo(1, 0);
                return;
}
//  ____________________________________________________________
//  |                                                           |
//  | TURN - Turns robot while travelling, can change speed     |
//  |___________________________________________________________|

void turn(int dir, int turnmag, int newspeed){   // dir 0=left, 1=right,  amount =0 to 100
                int turnsp=0, regsp=0, x=0;
                speed=newspeed;
                if (dir==0){
                        turnsp=3000 - 20*speed*.01*turnmag;
                        regsp=3000 + 20*speed;
                        servo(0, turnsp);
                        servo(1, regsp);
                        printf("turn speed %4d \t\t normal speed %4d \t\n",turnsp, regsp);
                }
```

```
		if (dir==1){
			turnsp=3000+20*speed*.01*turnmag;
			regsp=3000-20*speed;
			servo(0, regsp);
			servo(1, turnsp);
			printf("turn speed %4d \t\t normal speed %4d \t\n",turnsp, regsp);
		}
			// * could add correction factor to speed for the one in reverse if not straight enough.....
		return;
}

//  ____________________________________________________________
// |                                                            |
// |    SPIN - Turns robot in place, then stops                 |
// |____________________________________________________________|

void spin(int amount){   // amount =-360 to 360
		int turntime=amount*turnconst;
		speed=50;       // set const turn speed, 50, 100, whatever
		if (turntime>0){
			servo(0, 3000+20*speed);
			servo(1, 3000+20*speed);
			printf("\t\t\t  turntime= %d ", turntime);
			if (turntime>100){
				wait(100);
				turntime=turntime-100;
				while (turntime>100){
					observe(100);
					turntime=turntime-100;
				}
				wait(turntime);
			}
			else{wait(turntime);}

		}
		if (turntime<0){
			servo(0, 3000-20*speed);
			servo(1, 3000-20*speed);
			turntime=-1*turntime;
			printf("\t\t\t  turntime= %d ", turntime);
			if (turntime>100){
				wait(100);
				turntime=turntime-100;
				while (turntime>100){
					observe(100);
					turntime=turntime-100;
				}
				wait(turntime);
			}
			else{wait(turntime);}
		}
		stop();
		return;
}

//  ____________________________________________________________
// |                                                            |
// |    BLINDSPIN - Turns robot in place without looking        |
// |____________________________________________________________|

void blindspin(int amount){   // amount =-360 to 360
		int turntime=amount*turnconst;
		speed=50;       // set const turn speed, 50, 100, whatever
		if (turntime>0){
			servo(0, 3000+20*speed);
			servo(1, 3000+20*speed);
			printf("\t\t\t  turntime= %d ", turntime);
			wait(turntime);
		}
		if (turntime<0){
			servo(0, 3000-20*speed);
			servo(1, 3000-20*speed);
			turntime=-1*turntime;
			printf("\t\t\t  turntime= %d ", turntime);
			wait(turntime);
		}
		stop();
		return;
}

void slowblindspin(int amount){   // amount =-360 to 360
		int turntime=amount*turnconst*2;
		speed=15;       // set const turn speed, 50, 100, whatever
		if (turntime>0){
			servo(0, 3000+20*speed);
			servo(1, 3000+20*speed);
			printf("\t\t\t  turntime= %d ", turntime);
			wait(turntime);
		}
		if (turntime<0){
			servo(0, 3000-20*speed);
			servo(1, 3000-20*speed);
			turntime=-1*turntime;
			printf("\t\t\t  turntime= %d ", turntime);
			wait(turntime);
		}
		stop();
```

```
		return;
}

//  ___________________________________________________________
//  |                                                          |
//  | IRSAMPLE - Samples IR detectors, results into arrays     |
//  |__________________________________________________________|

void irsample(){
		irl[iri]=IR_LEFT;
		irr[iri]=IR_RIGHT;
		iri=(iri+1)%10;
		return;
}

//  ___________________________________________________________
//  |                                                          |
//  |     FIREAVG - Averages last 10 fire sensor values        |
//  |__________________________________________________________|

void fireavg(){
		int fire1sum=0,fire2sum=0, q=0;
		for (q=0;q<10;q++){
			read_IR();
			fire1sum=fire1sum+FLAME_SENSOR_1;
			fire2sum=fire2sum+FLAME_SENSOR_2;
			}
			FIRE1=.1*fire1sum;
			FIRE2=.1*fire2sum;
		return;
}

//  ___________________________________________________
//  |                                                  |
//  |     IR - Averages last 10 ir sample values       |
//  |__________________________________________________|

int ir(int lr){
		int avg=80, sum=0, q=0;
		if (lr==0){
			for (q=0;q<10;q++){
				sum=sum+irl[q];
			}
			avg=.1*sum;
		}
		else{
			for (q=0;q<10;q++){
				sum=sum+irr[q];
			}
			avg=.1*sum;
		}
		return avg;
}

//  ___________________________________________________
//  |                                                  |
//  | SEARCH - Normal collision avoid & look for fires |
//  |__________________________________________________|

int search(void){
  int x=0;
  int n=1;
printf("into search\n");

		straight(100);
		while(1){
			read_IR();
//		while( (IR_LEFT<MED_THRESH) && (IR_RIGHT<MED_THRESH) && (FIRE<FLAME_THRESH) ) {
			RBUMPER=rear_bumper();
			FBUMPER=front_bumper();
			x++;
			if ((x%5)==1){				printf("read bumpers\nIRL= %3d \tIRR= %3d\n", IR_LEFT,IR_RIGHT);}

		//if FLAME_SENSOR
			if (firesout<7){n=1;}
			if ( ((FLAME_SENSOR_1 > FLAME_THRESH)&&(FLAME_SENSOR_1 < FLAME_THRESH_HI))||((FLAME_SENSOR_3 > FLAME_THRESH)
&&(FLAME_SENSOR_3 < FLAME_THRESH_HI))|| ((FLAME_SENSOR_2 > FLAME_THRESH)&&(FLAME_SENSOR_2 < FLAME_THRESH_HI))  ){
				while ((n == 1) && (FAIL_COUNT < 5)){
					printf("\t FIRE...FIRE...FIRE!!!!!\n");
					fireavg();
					n=fire();
					if (n == 0){
						firesout++;
					}
					if (n == 1){
					FAIL_COUNT++;
					}
				}
			}

		//if IR
			if ((IR_LEFT > NEAR_THRESH) && (IR_RIGHT > NEAR_THRESH)){
				printf("TOO CLOSE....TOO CLOSE...AHHH!!!!!!\n");
				printf("\t\t backing up\n");
				stop();
				wait(250);
				straight(-50);
```

```
                                        observe(300);
                                        stop();
                                        spin(60);
                                        if ((TCNT&0x01)==0){
                                                spin(23*(TCNT&0x03)+24);
                                        }
                                        if ((TCNT&0x01)==1){
                                                spin(-31*(TCNT&0x03)-21);
                                        }
                                        straight(20);
                                }
                                if ((IR_LEFT > NEAR_THRESH) && (IR_RIGHT < NEAR_THRESH)&& (IR_RIGHT > MED_THRESH)){
                                        printf("\t\t SPIN RIGHT\n");
                                        spin(20+(TCNT&0x03)*5);
                                        stop();
                                        straight(20);
                                        observe(250);
                                }
                                if ((IR_LEFT < NEAR_THRESH) && (IR_LEFT > MED_THRESH)&& (IR_RIGHT > NEAR_THRESH)){
                                        printf("\t\t SPIN LEFT\n");
                                        spin(-27*(TCNT&0x03)-20);
                                        stop();
                                        straight(20);
                                        observe(250);
                                }
                                if ((IR_LEFT > MED_THRESH)&& (IR_LEFT < NEAR_THRESH) && (IR_RIGHT > MED_THRESH)&& (IR_RIGHT < NEAR_THRESH)){
                                        printf("\t\t backing up\n");
                                        stop();
                                        wait(250);
                                        straight(-10);
                                        observe(500);
                                        if ((TCNT&0x01)==0){
                                                spin(23*(TCNT&0x03)+27);
                                        }
                                        if ((TCNT&0x01)==1){
                                                spin(-25*(TCNT&0x03)-23);
                                        }

                                }
                                if ((IR_LEFT > MED_THRESH) && (IR_LEFT < NEAR_THRESH) && (IR_RIGHT < MED_THRESH)){
                                        printf("/t/t turning RIGHT\n");
                                        turn(1, 35, 100);
                                        observe(700);
                                }
                                if ((IR_LEFT < MED_THRESH)  && (IR_RIGHT > MED_THRESH)&& (IR_RIGHT < NEAR_THRESH)){
                                        printf("\t\t turning LEFT\n");
                                        turn(0, 45, 80);
                                        observe(500);
                                }
                        //if BUMP
                                if (FBUMPER < 14){
                                        printf("\t\t going straight\n");
                                        straight(100);
                                        observe(500);
//                                      read_IR();
                                }
                                if ((FBUMPER>15)&&(FBUMPER<25)){
                                        straight(-50);
                                        observe(300);
                                        spin(-70);printf("      spin LEFT\n");
                                        stop();
                                        straight(100);
//                                      read_IR();
                                }
                                if ((FBUMPER>25)&&(FBUMPER<50)){
                                        straight(-50);
                                        observe(300);
                                        spin(70);printf("      spin RIGHT\n");
                                        stop();
                                        straight(100);
//                                      read_IR();
                                }

                        }
                        return 0;
}

//  ____________________________________________________
//  |                                                    |
//  |    FIRE - Firefighting behavior function           |
//  |____________________________________________________|

int fire(){
//turn, approach slowly, stop, check/turn, pump, check
                int i=1, dir=1, spinamount=0, dirold=1, FLAME_OLD_AVG=(FIRE1+FIRE2)/2;
                fireavg();
                //if ((FIRE1<FLAME_THRESH)&&(FIRE2<FLAME_THRESH)){return} //no bogus fires
                for(i=1;i<7;i++){
                        FLAME_OLD_AVG=(FIRE1+FIRE2)/2;
                        dirold=dir;
                        if ((i==1)&&(FIRE1>FIRE2) ){blindspin(10);dir=1;}
                        if ((i==1)&&(FIRE1<FIRE2) ){blindspin(-10);dir=-1;}

                        fireavg();

                        if ( ((FIRE1-FLAME_OLD_AVG)<FIRE_TOL) && ((FLAME_OLD_AVG-FIRE1)<FIRE_TOL) ){
                                spinamount=0;
```

```
                        }

                        if (FIRE1-FLAME_OLD_AVG>FIRE_TOL){
                                if (dir==1){spinamount=10/i;}
                                if (dir==-1){spinamount=20/i;}
                                dir=1;
                        }
                        if (FIRE2-FLAME_OLD_AVG>FIRE_TOL){
                                if (dir==-1){spinamount=-10/i;}
                                if (dir== 1){spinamount=-20/i;}
                                dir=-1;
                        }

                        blindspin(spinamount);
                }
                pump(130,0);
                fireavg();
//if fail
                if ((FLAME_SENSOR_1 > FLAME_THRESH) ||(FLAME_SENSOR_3 > 2*FLAME_THRESH) ||(FLAME_SENSOR_2 > FLAME_THRESH)){
                        return 1;
                }
//return 0 on success
//return 1 on fail

                //***** NOT CHANGED******
                return 0;
}


//  ____________________________________________
//  |                                           |
//  |                MAIN                       |
//  |___________________________________________|

int main(){
                int k=0, i=0, j=0, n=0, f=0, t=0, temp1=0, temp2=0, control=0;
                //ir on, sample 10x to start with correct avg before moving away
                DIG_OUT=0x03;
                init_analog();
                init_clocktk();
                init_serial();
                init_servos();

// do nothing until triggered
                while (!((FBUMPER<43)^(RBUMPER<126))) {
                        FBUMPER=front_bumper();
                        RBUMPER=rear_bumper();
                }
// then begin firefigthing
                printf("start\n");
                straight(100);
                wait(400);
                FBUMPER=front_bumper();
                RBUMPER=rear_bumper();

                if (RBUMPER<21){                FLAME_THRESH= 57; FLAME_THRESH_HI=255; FIRE_NUMBER=4;  MED_THRESH=113; NEAR_THRESH=118;}  //default
                if ((RBUMPER>22)&(RBUMPER<40))  { FLAME_THRESH= 57; FLAME_THRESH_HI=255; FIRE_NUMBER=4;  MED_THRESH=117;
NEAR_THRESH=124;}  // only IR hi
                if ((RBUMPER>40)&(RBUMPER<46))  { FLAME_THRESH= 57; FLAME_THRESH_HI=255; FIRE_NUMBER=14; MED_THRESH=113;
NEAR_THRESH=118;}  // #fires hi only
                if ((RBUMPER>47)&(RBUMPER<62))  { FLAME_THRESH= 57; FLAME_THRESH_HI=255; FIRE_NUMBER=14; MED_THRESH=117;
NEAR_THRESH=124;}  // fires hi, IR hi
                if ((RBUMPER>63)&(RBUMPER<81))  { FLAME_THRESH= 70; FLAME_THRESH_HI=160; FIRE_NUMBER=4;  MED_THRESH=113;
NEAR_THRESH=118;}  // flame hi only
                if ((RBUMPER>82)&(RBUMPER<92))  { FLAME_THRESH= 70; FLAME_THRESH_HI=160; FIRE_NUMBER=4;  MED_THRESH=117;
NEAR_THRESH=124;}  // flame hi, IR hi
                if ((RBUMPER>93)&(RBUMPER<103) ){ FLAME_THRESH= 70; FLAME_THRESH_HI=160; FIRE_NUMBER=14; MED_THRESH=113;
NEAR_THRESH=118;}  // flame hi, #fires hi
                if ((RBUMPER>104)&(RBUMPER<111)){ FLAME_THRESH= 70; FLAME_THRESH_HI=160; FIRE_NUMBER=14;  MED_THRESH=117;
NEAR_THRESH=124;}  // all high
                if ((RBUMPER>112)&(RBUMPER<126)){ FLAME_THRESH= 57; FLAME_THRESH_HI=255; FIRE_NUMBER=7;  MED_THRESH=117;
NEAR_THRESH=120;}  //old default
                search();
                control=search();

// if firesout=0.... then search again, because there's at least 1.
                if (control==0){
                        if (firesout==0) {search();}
                        return 0;
                }
                if (control==1){
                        straight(-40);
                        observe(200);
                        spin(-90);
                        return 0;
                }
                if (control==2){
                        fire();
                        return 0;
                }
                return 0;
}
```

Appendix 2

FLAME

・ Program Code

```
// FrameGrabberTestDoc.cpp : implementation of the CFrameGrabberTestDoc class
//
// (c) Vadim Gorbatenko, 1999
// gvv@mail.tomsknet.ru
// All rights reserved
// Original code by  Gorbatenko in normal font
// Modified code in Bold font

#include "stdafx.h"
#include "math.h"
#include "FrameGrabberTest.h"
#include "FrameGrabberTestDoc.h"
#include <stdio.h>
#include <io.h>
#include <dos.h>
#include <conio.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define     PROCESSOR_PAUSED                    0
#define     PROCESSOR_SIMPLE_VIEWER             1
#define     PROCESSOR_IMAGE_FILTER              2
#define     PROCESSOR_MOTION_DETECTOR   3
#define LPT1 0x0278
#define STATUS1 LPT1+1
#define CONTROL1 LPT1+2
//proto
//********ADC Command********
int adc(int);
//END PROTO
//forward definition for motion detector
LONG        summ_rect_arena32(LPBYTE data, int dx, int dy, int ptX, int ptY, int rx, int ry);
LONG        summ_rect_arena24(LPBYTE data, int dx, int dy, int ptX, int ptY, int rx, int ry);
////////////////////////////////////////////////////////////////////////////
int y=0,x=0,dir=0x0;
double coeff[720][720];
double horz=0,vert=0,Z=0,vertavg=20,horzavg=50;
double t0=0,t1=0;
double horz1=0,horz2=0,horz3=0,horz4=0,horz5=0,horzavg3=0;
double vert1=0,vert2=0,vert3=0,vert4=0,vert5=0,vertavg3=0;
int c=0, i=0, j=0, k=0, q=0, rb=0, N=400;
int sensavg[8],sum[8], sensor[8];
unsigned int a=_inp(STATUS1),data=0;
int acquired=0;
int RED1=200, RED2=120, GREEN1=200, GREEN2=120, BLUE1=170, BLUE2=90;
int color=0;
int pause1=0, pause2=0, pause3=0;
int sw11=0, sw12=0, sw13=0, sw21=0, sw22=0, sw23=0;
int bigT=1000000;
// CFrameGrabberTestDoc

IMPLEMENT_DYNCREATE(CFrameGrabberTestDoc, CDocument)

BEGIN_MESSAGE_MAP(CFrameGrabberTestDoc, CDocument)
        //{{AFX_MSG_MAP(CFrameGrabberTestDoc)
        ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
        ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
        ON_UPDATE_COMMAND_UI(ID_SIMPLE_FILTER, OnUpdateSimpleFilter)
        ON_UPDATE_COMMAND_UI(ID_SIMPLE_VIEWER, OnUpdateSimpleViewer)
        ON_UPDATE_COMMAND_UI(ID_DETECTOR, OnUpdateDetector)
        ON_COMMAND(ID_DETECTOR, OnDetector)
        ON_COMMAND(ID_SIMPLE_FILTER, OnSimpleFilter)
        ON_COMMAND(ID_SIMPLE_VIEWER, OnSimpleViewer)
        ON_COMMAND(ID_FILE_SAVE_AS, OnFileSaveAs)
        ON_UPDATE_COMMAND_UI(ID_FILE_SAVE_AS, OnUpdateFileSaveAs)
        ON_COMMAND(ID_PAUSE, OnPause)
        ON_UPDATE_COMMAND_UI(ID_PAUSE, OnUpdatePause)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
// CFrameGrabberTestDoc construction/destruction

CframeGrabberTestDoc::CFrameGrabberTestDoc(){}

CFrameGrabberTestDoc::~CFrameGrabberTestDoc(){}

BOOL CFrameGrabberTestDoc::OnNewDocument(){
        if (!CDocument::OnNewDocument())
                return FALSE;
        m_ProcessorMode = PROCESSOR_IMAGE_FILTER;
        return TRUE;
}
```

```
/////////////////////////////////////////////////////////////////////////////
// CFrameGrabberTestDoc diagnostics

#ifdef _DEBUG
void CFrameGrabberTestDoc::AssertValid() const{
	CDocument::AssertValid();
}
void CFrameGrabberTestDoc::Dump(CDumpContext& dc) const{
	CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CFrameGrabberTestDoc commands

void CFrameGrabberTestDoc::OnEditCopy() {
	if(m_ImageBitmap.GetSafeHandle() &&
		AfxGetMainWnd()->OpenClipboard()){
		EmptyClipboard();
		SetClipboardData(CF_DIB, m_ImageBitmap.DibFromBitmap());
		CloseClipboard();
		}
}

void CFrameGrabberTestDoc::OnUpdateEditCopy(CCmdUI* pCmdUI) {
		pCmdUI->Enable((BOOL)m_ImageBitmap.GetSafeHandle());
}

#define	IMAGEWIDTH(lpd) ((LPBITMAPINFOHEADER)lpd)->biWidth
#define	IMAGEHEIGHT(lpd) ((LPBITMAPINFOHEADER)lpd)->biHeight
#define	IMAGEBITS(lpd) ((LPBITMAPINFOHEADER)lpd)->biBitCount
#define	IMAGEDATA(lpd) (((LPBYTE)lpd) + (((LPBITMAPINFOHEADER)lpd)->biSize))
// This method called by CFrameGrabberTestView
void	CFrameGrabberTestDoc::ProcessImage(LPBITMAPINFO lpBi){
	static BOOL bRunNow = FALSE;
	if(!lpBi || bRunNow)		return;
	bRunNow = TRUE;
	switch(m_ProcessorMode){
	case PROCESSOR_SIMPLE_VIEWER:
			m_ImageBitmap.CreateFromDib(lpBi);
			UpdateAllViews(NULL);	break;
	case PROCESSOR_IMAGE_FILTER:
			if(IMAGEBITS(lpBi)!=32&&
				IMAGEBITS(lpBi)!=24)
			{
				AfxMessageBox("Can't run filter: unsupported color resolution!",MB_ICONWARNING);
				m_ProcessorMode =PROCESSOR_SIMPLE_VIEWER;
				break;
			}

			if(ApplyFilter(lpBi))
			{	m_ImageBitmap.CreateFromDib(lpBi);
				UpdateAllViews(NULL);}	break;
	case PROCESSOR_MOTION_DETECTOR:
			if(IMAGEBITS(lpBi)!=32&&
				IMAGEBITS(lpBi)!=24)
			{
				AfxMessageBox("Can't run motion detector: unsupported color resolution!",MB_ICONWARNING);
				m_ProcessorMode =PROCESSOR_SIMPLE_VIEWER;
				break;
			}
			if(RunDetector(lpBi))
			{	m_ImageBitmap.CreateFromDib(lpBi);
				UpdateAllViews(NULL);}	break;
	default:
				break;
	}
	bRunNow = FALSE;
}

void CFrameGrabberTestDoc::OnUpdateSimpleFilter(CCmdUI* pCmdUI) {
	pCmdUI->SetCheck(m_ProcessorMode==PROCESSOR_IMAGE_FILTER);}

void CFrameGrabberTestDoc::OnUpdateSimpleViewer(CCmdUI* pCmdUI) {
	pCmdUI->SetCheck(m_ProcessorMode==PROCESSOR_SIMPLE_VIEWER);}

void CFrameGrabberTestDoc::OnUpdateDetector(CCmdUI* pCmdUI) {
	pCmdUI->SetCheck(m_ProcessorMode==PROCESSOR_MOTION_DETECTOR);}

void CFrameGrabberTestDoc::OnDetector() {
	m_ProcessorMode=PROCESSOR_MOTION_DETECTOR;}

void CFrameGrabberTestDoc::OnSimpleFilter() {
	m_ProcessorMode=PROCESSOR_IMAGE_FILTER;}

void CFrameGrabberTestDoc::OnSimpleViewer() {
	m_ProcessorMode=PROCESSOR_SIMPLE_VIEWER;}

void CFrameGrabberTestDoc::OnFileSaveAs() {
	UINT oldp=m_ProcessorMode;
	m_ProcessorMode=PROCESSOR_PAUSED;
	CDC *pDC=m_ImageBitmap.BegingModify();
		pDC->SetTextColor(196);
		pDC->SetBkMode(TRANSPARENT);
		CSize sz=m_ImageBitmap.GetSize();
		char str[32];
```

```
                        char str2[32];
                        CString title;
                        title.Format("REC %s %s", _strtime(str),_strdate(str2));;
                        pDC->TextOut(16,sz.cy-16, title);
        m_ImageBitmap.EndModify();
        m_ImageBitmap.Save(NULL);
        m_ProcessorMode=oldp;}

void CFrameGrabberTestDoc::OnUpdateFileSaveAs(CCmdUI* pCmdUI) {
        pCmdUI->Enable((BOOL)m_ImageBitmap.GetSafeHandle());}

void CFrameGrabberTestDoc::OnPause() {
        m_ProcessorMode=PROCESSOR_PAUSED;}

void CFrameGrabberTestDoc::OnUpdatePause(CCmdUI* pCmdUI) {
        pCmdUI->SetCheck(m_ProcessorMode==PROCESSOR_PAUSED);}

//////////////////////////////////////////////////////////
// Processor stuff
BOOL CFrameGrabberTestDoc::ApplyFilter(LPBITMAPINFO lpBi){
                        ASSERT(lpBi);
                        int arena_dx = IMAGEWIDTH(lpBi);
                        int arena_dy = IMAGEHEIGHT(lpBi);
                        //
                        // This is the simplest Edge detector
                        // Pixel(x0,y0)= (abs(Pixel(x0,y0)-Pixel(x+1,y+1)) + abs(Pixel(x+1,y0)-Pixel(x0,y+1)))/2
                        //
                        switch(IMAGEBITS(lpBi))         {
                        case 32:
                                {
                                        LONG * ptr =(LONG *)IMAGEDATA(lpBi);
                                        LONG * tmpLine = new LONG[arena_dx];
                                        for(int y=0; y<arena_dy-1;y++)
                                        {
                                                for(int x=0; x<arena_dx-1;x++)
                                                {
                                                        int x1y1=x+arena_dx+1;
                                                        int x1y0 =x+1;
                                                        int x0y1=x+arena_dx;
                                                        LONG r_diag1 =          abs(ptr[x]&0xff -ptr[x1y1]&0xff);
                                                        LONG g_diag1 =          abs((ptr[x]>>8)&0xff - (ptr[x1y1]>>8)&0xff);
                                                        LONG b_diag1 =          abs((ptr[x]>>16)&0xff -(ptr[x1y1]>>16)&0xff);
                                                        LONG r_diag2 =          abs(ptr[x1y0]&0xff -ptr[x0y1]&0xff);
                                                        LONG g_diag2 =          abs((ptr[x1y0]>>8)&0xff -(ptr[x0y1]>>8)&0xff);
                                                        LONG b_diag2 =          abs((ptr[x1y0]>>16)&0xff -(ptr[x0y1]>>16)&0xff);
                                                        tmpLine[x] = ((r_diag1+r_diag2)>>1) |
                                                                                (((g_diag1+g_diag2)>>1)<<8) |
                                                                                (((b_diag1+b_diag2)>>1)<<16);
                                                }
                                                memcpy(ptr,tmpLine, (arena_dx-1)*sizeof(DWORD));
                                                ptr+=arena_dx;
                                        }

                                        delete tmpLine;
                                }
                                break;
                        case 24:
                                {
                                        BYTE * ptr =(BYTE *)IMAGEDATA(lpBi);
                                        BYTE * tmpLine = new BYTE[arena_dx*3];
                                        sensor[3]=adc(3);
                                        while(sensor[3]>5){
                                                k=0;
                                                while(k<1000){k++;}
                                                pause1=adc(3);
                                                k=0;
                                                while(k<1000){k++;}
                                                pause2=adc(3);
                                                k=0;
                                                while(k<1000){k++;}
                                                pause3=adc(3);
                                                sensor[3]=(pause1+pause2+pause3)/3;
                                        }
                                        k=0;
                                        for (q=0;q<50;q++){
                                                sensor[0]=adc(0);
                                                sensor[5]=adc(5);
                                                if (q==0){sum[0]=0;sum[5]=0;}
                                                sum[0]=sum[0]+sensor[0];
                                                sum[5]=sum[5]+sensor[5];
                                        }

                                        sensavg[0]=.015*sum[0];
                                        sensavg[5]=.015*sum[5];
                                        sensor[0]=sensavg[0];
                                        sensor[5]=sensavg[5];
                                        k=0;
                                        q=0;
                                        color=0;
                                        //Yellow (Default)
                                        if ((sensor[0]<10)&&(sensor[5]<10)){
                                                RED1=120;   GREEN1=120;                 BLUE1=90;
                                                RED2=200;   GREEN2=200;                 BLUE2=170;
                                                color=0;
                                        }
                                        //Yellow Low Light
                                        if ((sensor[0]>10)&&(sensor[5]>10)){
```

```
                RED1=120;   GREEN1=120;                 BLUE1=90;
                RED2=100;   GREEN2=100;                 BLUE2=60;
                color=0;
        }
        //Red
        if ((sensor[0]>10)&&(sensor[5]<10)){
                RED1=135;   GREEN1=85;          BLUE1=100;
                RED2=180;   GREEN2=140;                 BLUE2=150;
                color=1;
        }
        //Blue/Green
        if ((sensor[0 ]<10)&&(sensor[5]>10)){
                RED1=140;   GREEN1=145;                 BLUE1=180;
                RED2=170;   GREEN2=180;                 BLUE2=150;
                color=2;
        }

        for(y=0; y< arena_dy-1;y++)
        {
                for(x=0; x<arena_dx-1; x++)
                {
                        int x0y0 = x*3;
                        int x1y1 = x0y0+arena_dx*3+3;
                        int x1y0 = x0y0+3;
                        int x0y1 = x0y0+arena_dx*3;

                        if(color==0){

if(((ptr[x0y0+2]>RED1)&&(ptr[x0y0+1]>GREEN1)&&(ptr[x0y0+0]<BLUE1))||((ptr[x0y0+2]>RED2)&&(ptr[x0y0+1]>GREEN2)&&(ptr[x0y0+0]<BLUE2)
)){
                                        coeff[y][x]=1;
                        }
                        else{
                                        coeff[y][x]=0;
                        }
                }
                if(color==1){

if(((ptr[x0y0+2]>RED1)&&(ptr[x0y0+1]<GREEN1)&&(ptr[x0y0+0]<BLUE1))||((ptr[x0y0+2]>RED2)&&(ptr[x0y0+1]<GREEN2)&&(ptr[x0y0+0]<BLUE2)
)){
                                        coeff[y][x]=1;
                        }
                        else{
                                        coeff[y][x]=0;
                        }
                }
                if(color==2){

if(((ptr[x0y0+2]<RED1)&&(ptr[x0y0+1]>GREEN1)&&(ptr[x0y0+0]>BLUE1))||((ptr[x0y0+2]<RED2)&&(ptr[x0y0+1]>GREEN2)&&(ptr[x0y0+0]>BLUE2)
)){
                                        coeff[y][x]=1;
                        }
                        else{
                                        coeff[y][x]=0;
                        }
                }


                //draw overlay image
                tmpLine[x0y0]  = (BYTE)((coeff[y][x])*ptr[x0y0]);
                tmpLine[x0y0+1]= (BYTE)((coeff[y][x])*ptr[x0y0+1]);
                tmpLine[x0y0+2]= (BYTE)((coeff[y][x])*ptr[x0y0+2]);
                if(((arena_dx/3)>(x-1))&&((arena_dx/3)<(x+1))){
                                tmpLine[x0y0]  = (BYTE)(0);
                                tmpLine[x0y0+1]= (BYTE)(0);
                                tmpLine[x0y0+2]= (BYTE)(255);
                }
                if(((2*arena_dx/3)>(x-1))&&((2*arena_dx/3)<(x+1))){
                                tmpLine[x0y0]  = (BYTE)(0);
                                tmpLine[x0y0+1]= (BYTE)(0);
                                tmpLine[x0y0+2]= (BYTE)(255);
                }
                if(((arena_dy/3)>(y-1))&&((arena_dy/3)<(y+1))){
                                tmpLine[x0y0]  = (BYTE)(0);
                                tmpLine[x0y0+1]= (BYTE)(0);
                                tmpLine[x0y0+2]= (BYTE)(255);
                }
                if(((2*arena_dy/3)>(y-1))&&((2*arena_dy/3)<(y+1))){
                                tmpLine[x0y0]  = (BYTE)(0);
                                tmpLine[x0y0+1]= (BYTE)(0);
                                tmpLine[x0y0+2]= (BYTE)(255);
                }
                if((horzavg3>(x-1))&&(horzavg3<(x+1))){
                                tmpLine[x0y0]  = (BYTE)(255);
                                tmpLine[x0y0+1]= (BYTE)(0);
                                tmpLine[x0y0+2]= (BYTE)(0);
                }
                if((vertavg3>(y-1))&&(vertavg3<(y+1))){
                                tmpLine[x0y0]  = (BYTE)(255);
                                tmpLine[x0y0+1]= (BYTE)(0);
                                tmpLine[x0y0+2]= (BYTE)(0);
                }

        }
        memcpy(ptr,tmpLine, (arena_dx-1)*3);
        ptr+=arena_dx*3;
}
```

```
                                        delete tmpLine;

                                        // FIND COORDINATES of X, Y
                                        for(y=1; y< arena_dy-2;y++){
                                                for(x=1; x<arena_dx-2; x++){
                                                        coeff[y][x]=(5*coeff[y][x]+coeff[y-2][x]+coeff[y+2][x]+coeff[y][x-2]+coeff[y][x+2])/9;
                                                }
                                        }
                                        vert=0;
                                        horz=0;
                                        Z=0;
                                        for(y=2; y< arena_dy-3;y++){
                                                for(x=2; x<arena_dx-3; x++){
                                                        if(coeff[y][x]>.925){
                                                                vert=vert+(coeff[y][x])*(coeff[y][x])*x;
                                                                horz=horz+(coeff[y][x])*(coeff[y][x])*y;
                                                                Z=Z+1;
                                                        }
                                                }
                                        }
                                        if (Z>0){
                                                vertavg=vert/Z;
                                                horzavg=horz/Z;
                                        }
                                        if (Z==0){
                                                vertavg=2*arena_dy/5;
                                                horzavg=arena_dx/3;
                                        }
                                        t0=horzavg;
                                        t1=vertavg;
                                        vertavg=t0;
                                        horzavg=t1;
                                        vert5=vert4;
                                        vert4=vert3;
                                        vert3=vert2;
                                        vert2=vert1;
                                        vert1=vertavg;
                                        horz5=horz4;
                                        horz4=horz3;
                                        horz3=horz2;
                                        horz2=horz1;
                                        horz1=horzavg;
                                        vertavg3=(23*vert1+17*vert2+11*vert3+5*vert4+vert5)/56;
                                        horzavg3=(23*horz1+17*horz2+11*horz3+5*horz4+horz5)/56;
//_____________________________________________________________________
//
//_____________________________________________________________________

                                        //******Calculate direction******
                                        dir=0x0;
                                        acquired=0;
                                        //Top Left
                                        if ((horzavg3<(arena_dx/3))&&(vertavg3>(2*arena_dy/3))){
                                                dir=0xfd;
                                        }
                                        //Top Mid
                                        if ((horzavg3>(arena_dx/3))&&(horzavg3<(2*arena_dx/3))&&(vertavg3>(2*arena_dy/3))){
                                                dir=0x7c;
                                        }
                                        //Top Right
                                        if ((horzavg3>(2*arena_dx/3))&&(vertavg3>(2*arena_dy/3))){
                                                dir=0x5e;
                                        }
                                        //Mid Left
                                        if ((horzavg3<(arena_dx/3))&&(vertavg3>(arena_dy/3))&&(vertavg3<(2*arena_dy/3))){
                                                dir=0xf5;
                                        }
                                        //Mid Mid
                                if((horzavg3<(2*arena_dx/3))&&(horzavg3>(arena_dx/3))&&(vertavg3>(arena_dy/3))&&(vertavg3<(2*arena_dy/3))){
                                                dir=0x0f;
                                                acquired=1;
                                        }
                                        //Mid Right
                                        if ((horzavg3>(2*arena_dx/3))&&(vertavg3>(arena_dy/3))&&(vertavg3<(2*arena_dy/3))){
                                                dir=0x5a;
                                        }
                                        //Bottom Left
                                        if ((horzavg3<(arena_dx/3))&&(vertavg3<(arena_dy/3))){
                                                dir=0xf7;
                                        }
                                        //Bottom Mid
                                        if ((horzavg3<(2*arena_dx/3))&&(horzavg3>(arena_dx/3))&&(vertavg3<(arena_dy/3))){
                                                dir=0xd3;
                                        }
                                        //Bottom Right
                                        if ((horzavg3>(2*arena_dx/3))&&(vertavg3<(arena_dy/3))){
                                                dir=0x5b;
                                        }
                                        //_____________________
                                        //    end of direction section
                                        //_____________________
//_____________________________________________________________________
//outputs
//_____________________________________________________________________
                                        // sets avg to 0
```

```
                                        for (j=0;j<8;j++){sensavg[j]=0;sum[j]=0;}
                                        //avg
                                        for (j=0;j<N;j++){
                                                for (i=0;i<8;i++){
                                                        sensor[i]=adc(i);
                                                        if (j==0){
                                                        sum[0]=0;sum[1]=0;sum[2]=0;sum[3]=0;sum[4]=0;sum[5]=0;sum[6]=0;sum[7]=0;
                                                        }
                                                        sum[i]=sum[i]+sensor[i];
                                                }
                                        }
                                        for (j=0;j<8;j++){
                                                sensavg[j]=.0025*sum[j];
                                        }


                                        //****Sensor Override Direction

                                        //Rbmp
                                        if ((sensavg[1]<15) && (sensavg[2]>15)) {
                                                _outp(LPT1,0xf5);
                                                k=0;
                                                while(k<bigT){k++;}
                                                _outp(LPT1,0xf5);
                                        }
                                        //Lbmp
                                        if ((sensavg[1]>15) && (sensavg[2]<15)) {
                                                _outp(LPT1,0x5a);
                                                k=0;
                                                while(k<bigT){k++;}
                                                _outp(LPT1,0x5a);
                                        }
                                        //Close and NOT Lined Up  //(sensavg[1]>15)&&(sensavg[2]>15)&&
                                        if ((sensavg[4]>130)&&(sensavg[6]>130)&&(acquired==0)){
                                                _outp(LPT1,0x73);
                                                k=0;
                                                while(k<bigT){k++;}
                                                _outp(LPT1,0x73);
                                        }
// left hi, right lo  &&(acquired==0)
                                        if ((sensavg[4]<135)&&(sensavg[6]>135)){
                                                _outp(LPT1,0x5a);
                                                k=0;
                                                while(k<bigT){k++;}
                                                _outp(LPT1,0x5a);
                                        }

//left lo, right hi
                                        if ((sensavg[4]>135)&&(sensavg[6]<135)&&(acquired==0)){
                                                _outp(LPT1,0xf5);
                                                k=0;
                                                while(k<2*bigT){k++;}
                                                _outp(LPT1,0xf5);
                                        }


                                        //Close and Lined Up
                                        if ( (sensavg[1]>15)&&(sensavg[2]>15)&&(sensavg[4]>135)&&(sensavg[6]>135)&&(acquired==1)){
                                                k=0;
                                                _outp(LPT1,0x00);
                                                while(k<bigT){
                                                        k++;
                                                }
                                        }


                                        //******Move Direction********
                                        if ( (sensavg[1]>15)&&(sensavg[2]>15)&&(sensavg[4]<135)&&(sensavg[6]<135)){
                                                _outp(LPT1,dir);
                                        }

//then back up
                                        if (dir==0xfd){_outp(LPT1,0x7d);}
                                        if (dir==0x7c){_outp(LPT1,0x7c);}
                                        if (dir==0x5e){_outp(LPT1,0x7e);}
//then sit still
                                        if (dir==0x5a){_outp(LPT1,0x0a);}
                                        if (dir==0x0f){_outp(LPT1,0x0f);}
                                        if (dir==0xf5){_outp(LPT1,0x05);}
//then go fwd
                                        if (dir==0xf7){_outp(LPT1,0xd7);}
                                        if (dir==0xd3){_outp(LPT1,0xd3);}
                                        if (dir==0x5b){_outp(LPT1,0xdb);}
                                                }
                                default: break;
                                }
                return TRUE;
}
//Simplest Motion detector
BOOL CFrameGrabberTestDoc::RunDetector(LPBITMAPINFO lpBi){

                //constants definitions for motion detector
#define         ZONESX          4
#define         ZONESY          4
#define         ZONES           ZONESX*ZONESY
#define         DETECTION_LEVEL 0.05f
```

```
        static LONG lastIntensity[ZONES];
        static BOOL init = TRUE;
        LONG newIntensity[ZONES];
        int rx = IMAGEWIDTH(lpBi)/ZONESX;
        int ry = IMAGEHEIGHT(lpBi)/ZONESY;
        for(int y = 0; y<ZONESY; y++)
                for(int x = 0; x<ZONESX; x++)
                        if(IMAGEBITS(lpBi)==32)
                         newIntensity[y*ZONESX+x] = summ_rect_arena32(  IMAGEDATA(lpBi),
                                IMAGEWIDTH(lpBi),
                                IMAGEHEIGHT(lpBi),
                                x*rx, y*ry, rx, ry);
                        else
                        if(IMAGEBITS(lpBi)==24)
                         newIntensity[y*ZONESX+x] = summ_rect_arena24(IMAGEDATA(lpBi),
                                IMAGEWIDTH(lpBi),
                                IMAGEHEIGHT(lpBi),
                                x*rx, y*ry, rx, ry);
                        else return FALSE;
        BOOL ret=FALSE;
        if(!init){
                FLOAT lastRel[ZONES-1];
                FLOAT newRel[ZONES-1];
                for(int i=0; i<ZONES-1; i++)
                        lastRel[i] = (float)lastIntensity[i]/(float)(lastIntensity[i+1]+1);
                for(i=0; i< ZONES-1; i++)                       {
                        newRel[i]= (float)newIntensity[i]/(float)(newIntensity[i+1]+1);
                        float alarm = (float)fabs(lastRel[i]-newRel[i])/newRel[i];
                        if(alarm >DETECTION_LEVEL)
                                {ret=TRUE; break;}
                }
        }
        memcpy(lastIntensity,newIntensity, ZONES*sizeof(LONG));
        init = FALSE;
        return ret;
}
//some detectors stuff
LONG    summ_rect_arena32(LPBYTE data, int dx, int dy, int ptX, int ptY, int rx, int ry){
        LONG summ = 0;
        int lineBytes = dx*4;
        data+= (lineBytes*ptY + ptX*4);//offset
        for(int y = 0; y<ry; y++, data+=lineBytes)
                for(int x = 0; x< rx*4; x+=4){
                        summ+= data[x+1];
                        summ+= data[x+2];
                        summ+= data[x+3];
                }
        summ/=3;
        return summ;}

LONG    summ_rect_arena24(LPBYTE data, int dx, int dy, int ptX, int ptY, int rx, int ry)
{
        LONG summ = 0;
        int lineBytes = dx*3;

        data+= (lineBytes*ptY + ptX*3);//offset

        for(int y = 0; y<ry; y++, data+=lineBytes)
                for(int x = 0; x< rx*3; x+=3)
                {
                        summ+= data[x+0];
                        summ+= data[x+1];
                        summ+= data[x+2];}
        summ/=3;
        return summ;
}
int in1=0, in2=0, out1=0, a0=0x0278;
int adc(int addr){
        a0=addr,
        //read registers
        in1=_inp(STATUS1);
        in2=_inp(CONTROL1);
        //preserve old control data, resend ADDRESS
        out1= ((in2)&(0xf0))+addr+0x08;
        _outp(CONTROL1, (out1 ^ 0x0b) );
        //send SAMPLE COMMAND to ADC,
        out1=out1-8;
        _outp(CONTROL1, (out1 ^ 0x0b) );

        while(  (_inp(STATUS1)) < 0x81 ){}
        //wait for ~ACK
        in1=_inp(STATUS1);
        return ((in1 & 0x78)*2);
}
```

Appendix 3

FLAME

· Original Program MATLAB Code: vision10.m

```
function vision10(file)
%MACHINE VISION 1.0
t0=clock;
warning off
VISION=imread(file);
tol=50;
mid=100;
vision=double(VISION);
vision_orig=vision;
sizes=size(vision);
M=10;  % constant weight of middle pixel
RED=120;      % MIN
GREEN=120;    % MIN
BLUE=90;  % MAX

for a=1:sizes(1),
   for b=1:sizes(2),
      if (~(...
            (((vision_orig(a,b,1)>RED)&(vision_orig(a,b,2)>GREEN)&(vision_orig(a,b,3)<BLUE))...
            | ((vision_orig(a,b,1)>200)&(vision_orig(a,b,2)>200)&(vision_orig(a,b,3)<170)))...
            ));
         vision(a,b,:)=0;
      end
   end
end
%filter out white brown dark, etc....

VISb=zeros(sizes(1),sizes(2));
for a=1:sizes(1),
   for b=1:sizes(2),
      if (((vision(a,b,1)>RED)&(vision(a,b,2)>GREEN)&(vision(a,b,3)<BLUE))...
            | ((vision(a,b,1)>200)&(vision(a,b,2)>200)&(vision(a,b,3)<170)));
         VISb(a,b)=1;

      else vision(a,b,:)=[0 0 0]';
      end
   end
end

R=5;
VISbin=zeros(sizes(1),sizes(2));
%blur it
for a=3:(sizes(1)-2),
    for b=3:(sizes(2)-2),
      VISbin(a,b)=(VISb(a+1,b-1)+VISb(a+1,b+1)+VISb(a-1,b+1)+VISb(a-1,b-1)+R*VISb(a,b))/(R+4);
      VISbin(a,b)=(VISb(a+1,b-1)+VISb(a+1,b+1)+VISb(a-1,b+1)+VISb(a-1,b-1)+R*VISb(a,b))/(R+4);
      VISbin(a,b)=(VISb(a+1,b-1)+VISb(a+1,b+1)+VISb(a-1,b+1)+VISb(a-1,b-1)+R*VISb(a,b))/(R+4);
   end
end
%the end blur

%calculate avg point
vert=0;horz=0;Z=0;
for a=3:sizes(1)-2,
   for b=3:sizes(2)-2,
      if (VISbin(a,b)>.9)
         vert=vert+(VISbin(a,b)^2)*a;
         horz=horz+(VISbin(a,b)^2)*b;
         Z=Z+1;
     end
   end
end

if (Z>0)
   vertavg=vert/Z;
   horzavg=horz/Z;
end

disp('Target Acquired....')
disp('X-coordinate ')
disp(horzavg)
disp('Y-Coordinate ')
disp(vertavg)
disp('')

VIS2=uint8(vision);
figure(1);subplot(2,1,1);image(VISION);
hold on;plot([.5*sizes(2) horzavg],[.5*sizes(1) vertavg],'m*');plot([.5*sizes(2) horzavg],[.5*sizes(1) vertavg],'m');
hold off
figure(1);subplot(2,1,2);image(VIS2)
hold on;plot([.5*sizes(2) horzavg],[.5*sizes(1) vertavg],'m*');plot([.5*sizes(2) horzavg],[.5*sizes(1) vertavg],'m');
hold off
print -djpeg90 -r0 test
T=etime(clock,t0);
disp('Elapsed Time:');disp(T);
```

Appendix 4

ADC Test Code

```
#include <stdio.h>
#include <io.h>
#include <dos.h>
#include <conio.h>

#define LPT1 0x0278
#define STATUS1 LPT1+1
#define CONTROL1 LPT1+2


int adc(int addr){
		int a0=addr, in1=0, in2=0, out1=0;
		//read registers
		in1=_inp(STATUS1);
		in2=_inp(CONTROL1);
		//preserve old control data, resend ADDRESS
		out1= ((in2)&(0xf0))+addr+0x08;
		_outp(CONTROL1, (out1 ^ 0x0b) );
		//send SAMPLE COMMAND to ADC,
		out1=out1-8;
		_outp(CONTROL1, (out1 ^ 0x0b) );

		while(  (_inp(STATUS1)) < 0x81 ){}
		//wait for ~ACK
		in1=_inp(STATUS1);
		return ((in1 & 0x78)*2);
}


int main(){
		int c=0, i=0, j=0, b=0, N=500;
		int sensavg[8],sum[8], sensor[8];
		unsigned int a=_inp(STATUS1),data=0;

		for (j=0;j<8;j++){sensavg[j]=0;sum[j]=0;}

		while(b<300){

			for(j=0;j<N;j++){
				for (i=0;i<8;i++){
					sensor[i]=adc(i);
					if (j==0){sum[0]=0;sum[1]=0;sum[2]=0;sum[3]=0;sum[4]=0;sum[5]=0;sum[6]=0;sum[7]=0;}
					sum[i]=sum[i]+sensor[i];
				}
			}


			for (j=0;j<8;j++){
				sensavg[j]=.002*sum[j];
			}

			printf( "Sen0 %3d Rbmp %3d Lbmp %3d Sen3 %3d   Rir %3d   Sen5 %3d    Lir %3d
\n",sensavg[0],sensavg[1],sensavg[2],sensavg[3],sensavg[4],sensavg[5],sensavg[6]);
		//	printf( "Sen0 %3d Sen1 %3d Sen2 %3d Sen3 %3d Sen4 %3d Sen5 %3d Sen6 %3d
\n",sensavg[0],sensavg[1],sensavg[2],sensavg[3],sensavg[4],sensavg[5],sensavg[6]);

//1,4 green 05
			//2,8, red (0a


			if ((sensavg[1]<15) & (sensavg[2]>15)&(sensavg[4]<140) & (sensavg[6]<140)) {_outp(LPT1,0x01);}
			if ((sensavg[1]>15) & (sensavg[2]>15)&(sensavg[4]>140) & (sensavg[6]<140)) {_outp(LPT1,0x02);}
			if ((sensavg[1]<15) & (sensavg[2]>15)&(sensavg[4]>140) & (sensavg[6]<140)) {_outp(LPT1,0x03);}
			if ((sensavg[1]>15) & (sensavg[2]<15)&(sensavg[4]<140) & (sensavg[6]<140)) {_outp(LPT1,0x04);}
			if ((sensavg[1]<15) & (sensavg[2]<15)&(sensavg[4]<140) & (sensavg[6]<140)) {_outp(LPT1,0x05);}
			if ((sensavg[1]>15) & (sensavg[2]<15)&(sensavg[4]>140) & (sensavg[6]<140)) {_outp(LPT1,0x06);}
			if ((sensavg[1]<15) & (sensavg[2]<15)&(sensavg[4]>140) & (sensavg[6]<140)) {_outp(LPT1,0x07);}
			if ((sensavg[1]>15) & (sensavg[2]>15)&(sensavg[4]<140) & (sensavg[6]>140)) {_outp(LPT1,0x08);}
			if ((sensavg[1]<15) & (sensavg[2]>15)&(sensavg[4]<140) & (sensavg[6]>140)) {_outp(LPT1,0x09);}
			if ((sensavg[1]>15) & (sensavg[2]>15)&(sensavg[4]>140) & (sensavg[6]>140)) {_outp(LPT1,0x0a);}
			if ((sensavg[1]<15) & (sensavg[2]>15)&(sensavg[4]>140) & (sensavg[6]>140)) {_outp(LPT1,0x0b);}
			if ((sensavg[1]>15) & (sensavg[2]<15)&(sensavg[4]<140) & (sensavg[6]>140)) {_outp(LPT1,0x0c);}
			if ((sensavg[1]<15) & (sensavg[2]<15)&(sensavg[4]<140) & (sensavg[6]>140)) {_outp(LPT1,0x0d);}
			if ((sensavg[1]>15) & (sensavg[2]<15)&(sensavg[4]>140) & (sensavg[6]>140)) {_outp(LPT1,0x0e);}
			if ((sensavg[1]<15) & (sensavg[2]<15)&(sensavg[4]>140) & (sensavg[6]>140)) {_outp(LPT1,0x0f);}

			if ((sensavg[1]>15) & (sensavg[2]>15)&(sensavg[4]<140) & (sensavg[6]<140)) {_outp(LPT1,0x00);}

			//for (c=0;c<1000;c++){                                   //////                          }
			b++;
		}

		return 0;
}
```

Appendix 5

Table 3. SMURF Unique Components.

Part Name	Part #	Supplier	Cost ($)	Quantity
Silicon Photo Detector w/ Filter	PD-4	All Electronics	6.00	2
T-53 STANDARD SERVOS	LXUK84	Tower Hobbies	9.99	2
Dubro S12 12 oz. Square Fuel Tank	LXD719	Tower Hobbies	3.79	1
AC Delco Windshield Washer Pump	22057650	Sun-State Recycling	3.00	1
8.4V NICAD Pack, 7 AAA Cells	NCB-84	All Electronics	3.00	2
14.4V 1100 mA Pack, 12 AA Cells	NCB-10	All Electronics	8.75	2
Duratrax KWIK-PIT 500 Fuel Bottle	LMT033	Tower Hobbies	6.49	1
Light Duty Swivel Caster	CST-4	All Electronics	1.00	2

Table 4. FLAME Unique Components.

Part Name	Part #	Supplier	Cost ($)	Quantity
2 ¾ Dubro Wheels (2 Pack)	275TL	Hobbie Warehouse	7.09	1
Handi-works cordless screwdriver	HW072	WalMart	8.88	2
3A, 55V H-Bridge motor drivers	LMD18200	National Semiconductor	Sample	2
Pixera color NTSC CCD camera	PXG-150N-PH	Computer Geeks	21.95	1
Solar 175W DC to AC inverter	SBPI-175	Cummins Industrial Tools	20.00	1
8 bit, 8 channel, 10Msps ADC	MAX118	Maxim	Sample	1
S3 Virge DX 3375 2MB	N/A	Computer Geeks	13.75	1
Cybertainment CybermailAV PCI video capture card BT878 Chipset	N/A	Computer Geeks	14.50	1
12V Heavy Duty Mabuchi RS-540SH Motor	G7851	Electronics Goldmine	1.00	2
Light Duty Swivel Caster	CST-4	All Electronics	1.00	4
EverStart 12V 9Ah Battery	ES12N94B1	Walmart	20.00	1
12V 2.6Ah Battery	NP2.6-12	YUASA	Sample	1

Suppliers:

All Electronics	www.allelectronics.com	
Computer Geeks	www.compgeeks.com	
Electronics Goldmine	www.goldmine-elec.com	
Maxim	www.maximic.com	*FREE SAMPLES*
National Semiconductor	*www.national.com*	*FREE SAMPLES*
Tower Hobbies	www.towerhobbies.com	
	www.walmart.com	