Robot Security Guard

EEL5934 Final Report

Joseph Larkin
University of Florida
Department of Electrical Engineering
5/1/95

Table of Contents

Abstract

This paper describes the purpose of my robot project. Next it discusses the overall system at a high level. The details of the mobile platform and actuation are given. It then provides details of the sensors I have developed and installed. Next it presents the behaviors programmed into my robot. Lastly, experimental layouts and results, conclusions and possible future work are described.

Executive Summary

The purpose of this project is to build a robot security guard. It must be capable of patrolling and area, detecting intruders, and approaching and following intruders. In order to do this it has several sensors including Shaft Encoders, IR Proximity Sensors, a Pyroelectric Sensor, a microphone phone noise detection and a light sensor. It also has a speech synthesizer that allows the robot to communicate with humans in a natural manner. A Motorola EVBU board is used for the "brains" of my project. The robot's behaviors are programmed in C. All the previously mentioned sensors have been developed and installed and behaviors have been implemented to achieve my purpose with this robot. The robot has been tested and performs well.

Introduction

The goal of building a robot security guard requires several sensors and behaviors. A security robot must roam around an area without colliding into obstacles. It must not get stuck in a part of the area. It must continuously monitor the area for intrusion. Once an intruder is detected, the robot should approach it and follow it. For the purpose of my project and intrusion can be a person, animal or any other source of heat detectable with a pyroelectric sensor. My robot also can detect loud noises and changes in light intensity. A speech synthesizer chip allows my robot to communicate in a natural manner with people. My robot would be suitable (with some enhancement) to guarding an area like a warehouse. Since it will pick up noise, light changes and heat sources, my robot would not be suitable for an automated factory or other "busy" place.

My robot has two driven wheels and a castor. It is fairly small (9" wide, 9" long and 4" high). It is constructed of thin plywood and some metal mending plates. The wheels are driven by geared DC motors. The robot is controlled by an Motorola EVBU board using a 68HC11 microcontroller with 32k of added RAM. I used Interactive C to program behaviors.

My robot has several sensors to allow it to perform its duties. The first of these sensors are the shaft encoders. These are useful for straight line motion. Five Infrared (IR) proximity detectors are used for use in collision prevention. The robot also has a Pyroelectric sensor to detect people and other sources of heat. It also has two other simple sensors to my robot, a microphone and a Cadmium Sulfide (CdS). My robot also includes a General Instruments SPO256-AL2 "Narrator" chip to perform speech synthesis.

All of these sensors and actuators work well. My robot has several programmed behaviors so it acts like a security guard. My robot continuously senses the environment for intrusion by detecting changes in light or loud sounds, and for the presence of heat sources. If intrusion is detected my robot tries to find the source and approach and intruder. My robot avoids obstacles while it is performing its other behaviors. I have programmed my robot to report status using a piezoelectric speaker and a speech synthesizer.

Robotic System

The robotic System is composed of several functions. A Motorola HC11 EVBU with added memory provides control over the rest of the system. Power and motor control is provided by a second board that is mounted underneath the robot. This board has a linear voltage regulator, a motor driver chip, a Quad NAND gate chip, and protection diodes. Several sensors provide the robot with information about its environement. Two motors are used for propulsion. A piezoelectric speaker and speech synthesis module are used to report status information.

Several sensors are necessary for the robot to perform its tasks. Infrared (IR) proximity sensors are used for collision avoidance and wall following. A pyroelectric sensor is used to detect heat sources including people. A Cadmium Sulfide (CdS) cell is used to determine changes in light levels and an amplified microphone can detect loud noises.

Mobile Platform

The mobile platform is a key element in an autonomous agent. Mine is constructed using plywood and some metal mending plate. Two 4-AA battery packs are mounted on the bottom of the platform. The main electronics are packaged on two boards. The first board contains a 5v voltage regulator, a filter capacitor, a 74LS00 chip (used as an inverter), and the '293 motor driver chip and diodes. The second board is a Motorola EVBU board with running in expanded mode. I added an octal latch (74HC573), a triple three input NAND gate (74HC10) and a 32Kx8 SRAM to provide extra RAM for Interactive C (IC). The design of these circuits is from Flynn and Jones. I added a PAL device
(PALCE22V10) for address decoding for two 8 bit digital output ports (2 74HC374s). One of the ports is used to control the IR LEDs used for proximity detection and the other is used to control the speech synthesizer (SPO256-AL2 Narrator chip).

I also added several headers as connectors on the circuit boards. The platform is somewhat heavy, which reduces battery life. A benefit of this heavy duty construction is that it is sturdy.

Actuation and Output Devices

Two Tsusaka 10V DC gearhead motors (part TG-32L-CG-1) propel the robot. These are surplus motors obtained for about five dollars a piece. They have a built in mounting bracket that was very convenient. These motors run at about one revolution per second at 12VDC. With the 3.5" diameter wheels, the robot moves about 8" per second with 8 NICAD batteries. The motors provide adequate torque. Interactive C provides Pulse Width Modulation (PWM) control for the motors. These motors will run with PWM values of 50% and up.

The robot moves on two rubber wheels from a hobby shop. They are bolted to a threaded coupler. Lock washers prevent the wheels from loosening. Heat shrink tubing was placed on the motor shafts and then threaded and glued the couplers. This has proved adequate, though not perfect.

Other features of the robot include a piezoelectric speaker and a speech synthesis module. The voice synthesis module is made from a SPO256-AL2 Narrator chip, a small audio amplifier (LM386), a small speaker, and several discrete components. This module is mounted on a small piece of perf board. A multiwire connector brings power, ground, 7 digital outputs and one input from the HC11 board. The schematics for this module is shown in figures 1 through 3. This design is taken from G. McCombs book figure 23-3 with minor modifications. The output port provides selection of phonemes and a strobe signal to tell the chip data is available. The connections to the phoneme generating chip are shown in figure 2 and filter and the audio amplifier circuit are shown in figure 3.

The voice module works by generating phonemes. Phonemes are the basic parts of a spoken language. English has about 40 basic phonemes. By sequential outputting phonemes any English Language word can be generated. Code for speech is shown in the appendix. The appendices also contain a data sheet on the SPO256-AL2 chip. This chip is no longer being manufactured but they can still be obtained from the surplus house B. G. Micro.

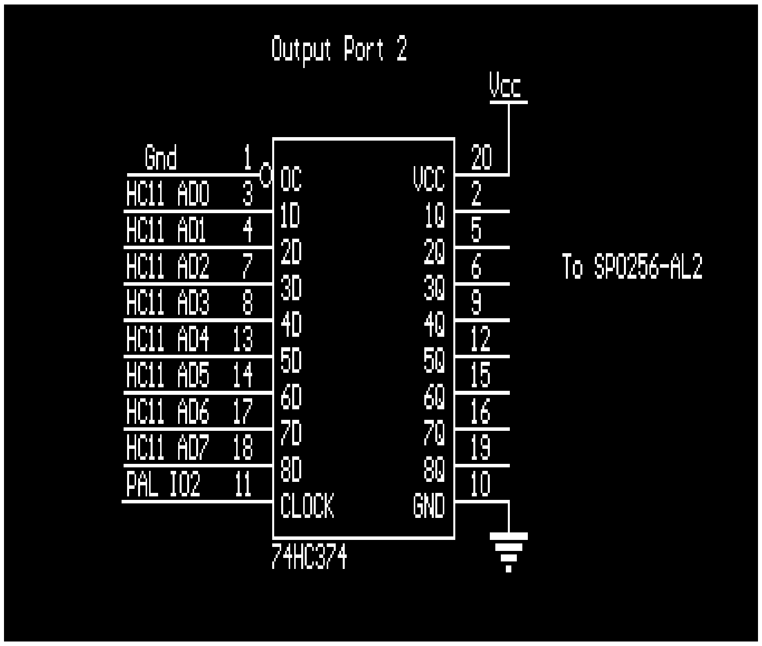

Figure 1 Output Port for Speach Synthesizer

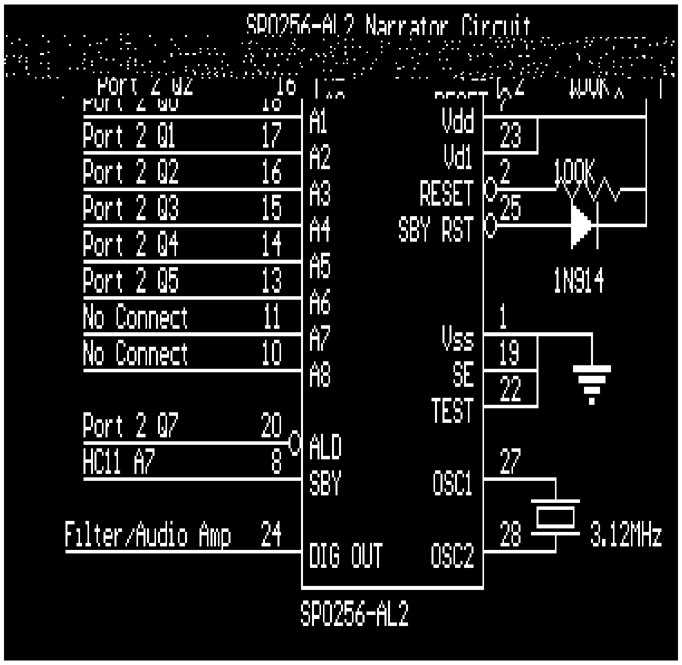

Figure 2 Connections to Phoneme Generator

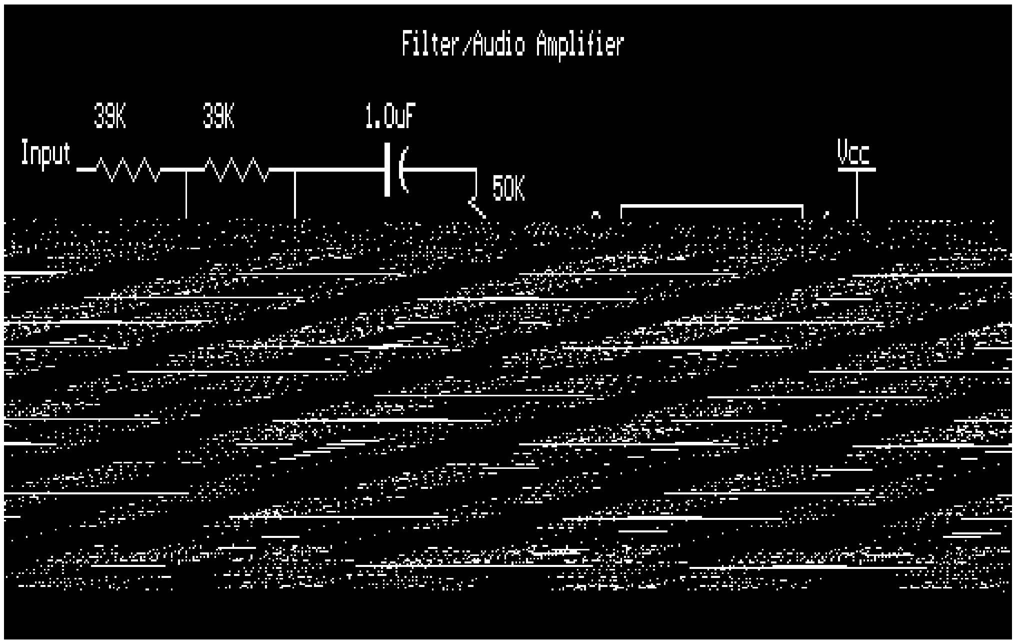

Figure 3 Audio Filter and Amplifier

Robotic Sensors

Shaft Encoders

Shaft encoders are installed on both wheels. The encoders use a Siemens G3355 IR transmitter receiver pair. These were hooked up with a current limiting resistor of 200 Ohms and a pullup resister of 30K Ohms. The Quad NAND gate on the power board was used to shape the output pulses before the were input to the HC11. A disk with alternating black and white stripes is used to make and break reflections. The output of the inverters is input into A0 and A1 of the HC11. The files ENCODERS.C and ENCODERS.ICB provide interrupt servicing when a pulse is detected. The program counts pulses to determine how far the wheels have turned.

IR Proximity Sensors

Five IR Proximity detectors are used for obstacle avoidance. There are three in the front. There is one that points straight ahead in the center and two that point outwards from the front corners. Two more are installed pointing sideways near the back of the robot. All of these use IR LEDs that are modulated at 40KHz. Modified Sharp GP1U52X detect reflections of the IR output of the LEDs. These sensors were modified to output an analog voltage that varies depending on the amount of 40KHz modulated IR they receive. The modulation is used so noise from ambient lighting lighting is rejected. The 40KHz signal is provided from the HC11 E clock divided down by a 74HC390 decade counter chip. A PAL (CE22V10) provides the latch signal for an output port (74HC374) This is shown in figures 4 and 5. The port drives three groups of 2 LEDs (Figure 6). Each group can be turned on or off without affecting the other groups. The LEDs are installed so one group has two LEDs pointed forward in the center of the front. Another group is used for the two front outward pointed sensors and the last group point out sideways in the back of the robot. Code for collision avoidance is shown in the appendix that contains my robot code.

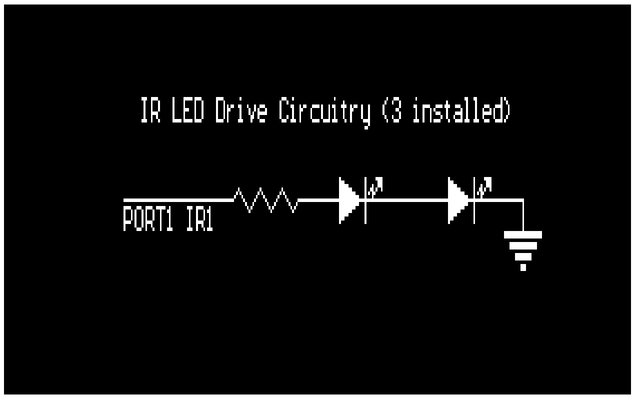

Figure 6 IR LED Circuitry

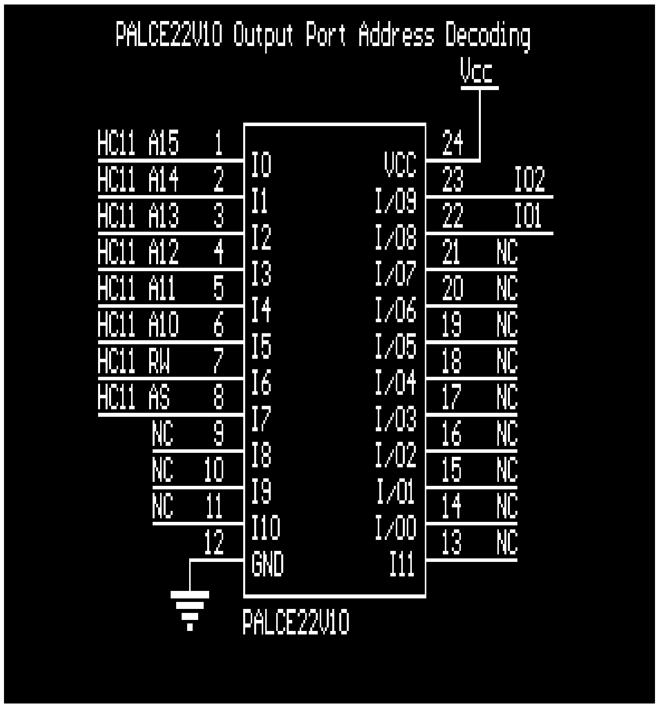

Figure 4 PAL Output Port Address Decoder Circuitry

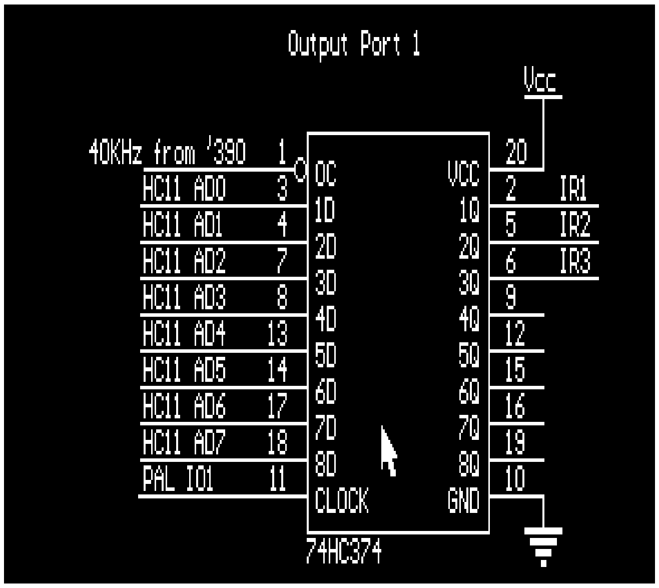

Figure 5 IR Driver Output Port

Pyroelectric Sensor

A pyroelectric sensor is installed to detect people, animals and fire. It is a sensitive IR detector that can detect heat sources. It outputs an analog voltage that varies as a warm object passes in front of its field of view. This sensor can also be scanned back and forth to detect non-moving heat sources. I procured an Eltec sensor (Model 442) with a fresnel lens from Electronics Plus. The sensor requires only 5VDC power, ground and signal connections. The output is onnected to E7 on the HC11.

This sensor can provide a voltage swing of a couple of tenths of a volt from a person across a room. The voltage swing increases to a volt or more when the person is closer. The voltage change as a warm object moves across the Pyro Sensors Field of View is similar to one cycle of a sin wave. If the object moves the other direction, the voltage shift is reversed. The direction of the voltage shift can be used to keep the robot pointed at the object. One of my robot behaviors uses this property to approach the intruder or heat source.

Microphone

A simple amplified microphone detects loud sounds. The circuit to interface with the microphone is shown in figure 8. The resistors after the capacitor are used to bias the output voltage to 2.5 volts. If they are not included the output "floats" no reading of the voltage can be performed. This microphone is not very sensitive so only loud noises can be detected.

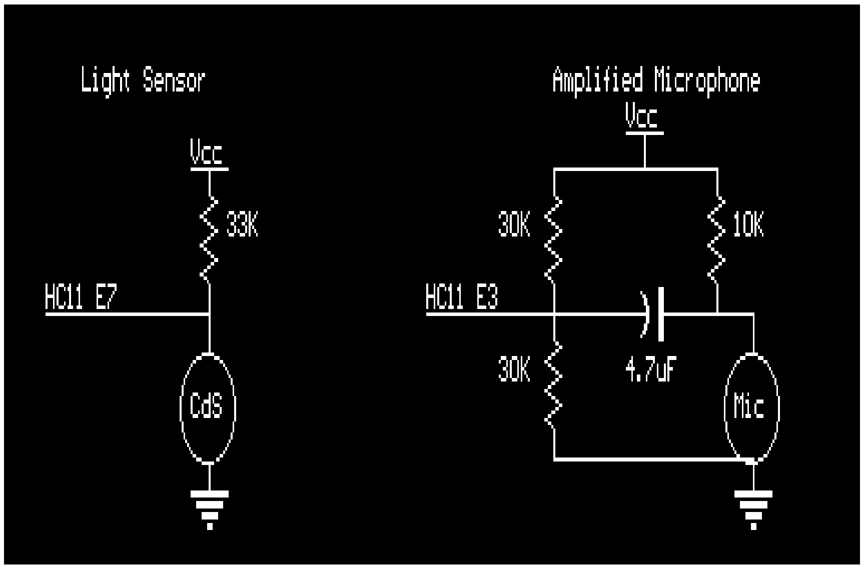

Figure 8 Light and Noise Sensors

Light Sensor

A CdS cell on the robot detects changes in light level. The circuit for this is shown in figure 8. The voltage change from completely dark to very bright is about 3 volts. Small changes are readily detected as well. An abrupt change in light level can be used to detect an intruder. An intruder may turn lights on or off, or use a flashlight. These conditions can all be detected.

Robotic Security Guard Behaviors

The robot has several behaviors. These behaviors are all running at the same time as separate processes. The subsumption architecture described in Jones and Flynn arbitrates between the behaviors. The following are the programmed behaviors:

Obstacle Avoidance
Detecting and Approaching heat sources
Scanning for heat sources
Hunting after loosing heat source
Monitoring Sound
Monitoring changes in light level
Sleep
Wander
Report status using speech synthesizer

Obstacle avoidance is the highest priority behavior. The IR proximity detectors are read twice a second and obstacles are avoided if detected. The routines used for this are read_IR(), and avoid_obstacle().

The next set of behaviors handle information for the pyroelectric sensor. A high priority routine (read_pyro()) monitors the pyro sensor quite often and tries to move the robot in the direction of the heat source. If this behavior has had a detection, but has lost contact with the heat source, the hunting behavior (fishtail()) takes over. Another separate behavior, scan_pyro(), is a low priority task that spins the robot around in a circle periodically if no pyro detect has occurred in a while. These three behaviors work well in finding heat sources, approaching them, and finding them again if the robot is thrown off track for some reason.

Two low priority behaviors monitor for loud noises (noise(), and listen()) and changes in light levels (light() and look()). The only effect these behaviors have is to "wake up" the robot if it is "sleeping". Sleeping (go_to_sleep()) is a very low priority behavior that tries to stop the motors periodically. Since nearly every other behavior is of higher priority, the robot will only sleep if there are no obstacles detect, no heat sources detected, no loud sounds and no changes in light levels.

The very lowest priority task is wander(). This behavior simply tries to turn both motors on to moderate forward speed. A flag is used to activate this behavior. The flag is set in most other behaviors when they have finished and a default behavior is needed. The only behavior that does not set this flag is the sleeping behavior.

The behavior to report status using the voice synthesizer is controlled only by the subsumption arbitration routine (arbitrate()). Status is reported every time a new behavior is activated (and other behaviors subsumed). This prevents the robot from "babbling" an reporting the same status over and over.

The subsumption architecture made it fairly easy to get multiple behaviors to work together properly. Before this code was written, several behaviors were implemented and had to work cooperatively. Before adding subsumption, interactions between processes caused subtle bugs that were very difficult to fix. Currently, the code is written so up to 25 behaviors can be handled. Even this high limit could be easily increased by changing array definitions. This type of arbitration worked very well.

Experimental layout and results

An incremental and mostly empirical approach was used in developing my robot. First the mobile platform was built and tested without the use of the microcontroller board. The motor were tested for speed and torque to verify they would be adequate for this application. Next the power regulator and motor control board was constructed. This was tested without use of the HC11 board by hooking motors to it and verifying they could be controlled. Next the unmodified HC11 EVBU board was hooked to the platform and motor control was tested using simple assembly language programs loaded using the BUFFALO monitor.

After this was completed, the HC11 board was upgraded. Memory was expanded to 32K of SRAM. This was tested under BUFFALO. The original HC11 chip (E7 revision) was then replaced with a A2. This was required to run Interactive C. Once this was done simple programs were written to verify IC performed correctly.

Several subsystems were added and tested independently. Shaft Encoders were installed and tested. Unfortunately, the code to perform straight line motion with this did not perform adequately. Even though the encoders reliably reported 32 pulses per revolution, the software from Jones and Flynn did not keep the robot moving in a straight line. Part of the problem may have been due to the use of the version 1.0 Rug Warrior Library. This software only outputs a very limited number of motor speeds. The 1.1 Library handles motor speeds in 1% increments and might fix the straight line problems.

Infrared (IR) proximity detector were added and tested to verify that the HC11 board could detect reflections from obstacles. The obstacle avoidance code was tested by running the robot in a very cluttered environment. Changes were made to the code based on observing the robots behavior.

The Pyroelectric sensor was first tested independent of the robot. Its output was hooked to an oscilloscope. The voltage swings were examined for various heat sources and distances to determine the sensitivity. The sensor was added to the robot and code written to use it. Thresholds and other code parameters were determined empirically.

Other sensors and actuators including the noise and light sensors, and the voice modules were developed in a similar way. An empirical method was used for much of the development because the environment the robot works in is very complicated. Many more individual tests could have been made to further quantize sensor behavior and to try to characterize the environment. However, even after this was done many code parameter such as thresholds and timings would still have had to be tweaked to get good performance.

Robotic Limitations, Possible Future Work, and Conclusion

Some simple changes to the robot would enhance its functionality. Adding more words to the robots vocabulary would be a simple change that would allow more complex interaction with people. This would involve simply enlarging one array and translating more English words to phonemes. Other changes that would make human interaction better would be to add some sort of input device and to further amplify the robots voice. Even something as simple as a keypad would be valuable. The robots voice is low volume. A simple change like driving the audio amplifier with the unregulated 10 volts would help this. Further amplification could also help, but would require more extensive modifications.

Adding a further stage of amplification and some sort of envelope detection to the amplified microphone would also be a valuable change. The robot can only detect loud sound sounds and even these are not always reliably detected. Additional amplification and peak detection hardware would help this by providing a larger signal to the microcontroller board and also off-load some sensor processing from the HC11 to external hardware.

Adding a pan head to the robot would provide for much better tracking of people. This could be accomplished relatively simply with a single servo motor. This change would allow the robot to perform much better tracking on moving heat sources. This would make the robot much better for acting as a security guard.

The robots behaviors could also be modified to provide more functions as well. Since the subsumption architecture is used in the robot, adding additional behaviors and modifying behaviors is not very difficult. Obstacle avoidance, heat source tracking and the wander behavior could all be enhanced relatively easily.

The robot never managed good straight line motion using shaft encoders. Testing with the 1.1 Rug Warrior Library could be done to see if this fixes the problem. If this was the problem a very simple change could fix this. Besides straight line motion, the shaft encoders could also be used for collision detection and recovery. A high priority behavior that detects when the robot should be moving but isn't could be used to provide motor commands to get the robot out of a "stuck: state. This redundancy would make the system more robust.

Even without the above changes, my robot functions well and has enough sensory data to perform a wide range of behaviors. Much has been accomplished in a fairly short time on this project. Some of the credit for this surely belongs to the Teaching Assistants in the class. Their experiences with building robots and troubleshooting facilitated my work. Credit also certainly is due Dr. Doty for developing a very informative and interesting class. His insistence on a fairly aggressive schedule with reasonable goals led to much being accomplished in a semester.

Documentation

[1] A M. Flynn and J. L. Jones, Mobile Robots - Inspiration to Implementation. Wellesley, Mass: A. K. Peters, 1993.

[2] F. Martin, The 6.270 Robot Builder's Guide. 1992.

[3] M68HC11EVBU - Universal Evaluation Board User's Manual. Phoenix: Motorola Literature Distribution, 1992.

[4] HC11- M68HC11 Reference Manual. Phoenix: Motorola Literature Distribution, 1991.

[5] HC11 - M68HC11 E Series Programming Reference Guide. Phoenix: Motorola Literature Distribution, 1993.

[6] G. McComb, The Robot Builder's Bonanza - 99 Inexpensive Robotics Projects, Blue Ridge Summit, PA, TAB Books, 1987.

Appendices

C Code

```
/* main.c - main routine with behaviors for robot.
   written in IC using 1.1 Rug Warrior Library.

    This file also requires util.c and speech.c.

    Joe Larkin
    EEL5934




*/





float NORMAL = 90.0;
float FAST = 100.0;
float TURN = 80.0;
float SLOW = 70.0;
float STOP = 0.0;

void ao() {
    motor(0,0.0);
    motor(1,0.0);
    }


/*array to hold status of routines. individual routine status
is in this array per priority
status 0 means innactive;  set by behavior routine
status 1 means active       "              "
status 2 means serviced;   set by arbitration routine
*/
int active[25];

/*go routine stores motor commands in mtr table for arbitrator use*/
float mtr0[25];
float mtr1[25];
void go(int m, float sp, int priority) {
    if (!m) mtr0[priority] = sp;
    else mtr1[priority] = sp;
    }


/* read ir analogs twice a second ; part of obstacle avoidance
behavior*/
int a0=0, a1=0, a2=0, a4=0, a5=0;
void read_ir() {
    while(1) {
       poke(0x2000,0xff);
       wait(50);
       hog_processor();
       a0 = analog(0);
       a1 = analog(1);
       a2 = analog(2);
       a4 = analog(4);
       a5 = analog(5);
       defer();
       poke(0x2000,0x00);
       wait(500);
       }
     }

/* IR obstacle avoidance behavior - highest priority task -
   priority 0*/
int ir_threshold = 110;
void ir_avoid() {
    int priority =0;
    int turn = 0;
    int what;
    while(1) {
        what = 0; active[priority]=0;
        wait(500);
        while (a1>ir_threshold || a0 > ir_threshold && a2 > ir_threshold) {
            go(0,-TURN, priority);
            go(1,-TURN, priority);
            active[priority] = 1;
            what = 1;
            }
         if (what) {
           go(turn,TURN, priority);
           active[priority] = 1;
           wait(333);
           }

        while(a0>ir_threshold) {
           go(0,-TURN, priority);
           go(1,TURN, priority);
           active[priority] = 1;
           what = 2;
           turn = 1;
           }

        while(a2>ir_threshold) {
           go(1,-TURN, priority);
           go(0,TURN, priority);
           active[priority] = 1;
           what = 3;
           turn = 0;
           }

       if (what || active[priority]) {
          hog_processor();
          active[priority]=0;
          wanderf=1;
          defer();
        }
      }
   }


/* scan_pyro spins robot around every 20 seconds if no pyro detects  low
   lowest prioity of three pyro related behaviors
priority = 20*/
void scan_pyro() {
   int priority = 20;
   float time;

   while(1) {
      go(0,SLOW, priority);
      go(1,-SLOW, priority);
      active[priority] = 1;
      time = seconds();
      while (!pyro_detect && ((seconds() - time) < 6.0)) defer();
      active[priority]=0;
      if (!pyro_detect) {
           wait(100);
           wanderf=1;
           }

      time = seconds();
      while (seconds() - time < 25.0) {
           active[priority]=0;
           if (pyro_detect || old_pyro) time = seconds();
           defer();
        }

      }
   }


/* pyro variables and read pyro routine , high priority = 1; main pyro
behavior also handles reading pyro
*/
int pyro_delta = 25;
int pyro_detect = 0;
int old_pyro = 0;
void read_pyro() {
    int priority = 1;
    int delta,count=0;
    float temp;
    delta = 0;

    while(1) {
        count++;
        hog_processor();
        delta = (analog(7) - 128);
        defer();
        if (-delta>pyro_delta) {
            pyro_detect = 1;
            old_pyro = 0;
            go(1,FAST, priority);
            go(0,SLOW, priority);
            active[priority] = 1;
            count = 0;
            }
        if (delta>pyro_delta) {
            pyro_detect = 1;
            old_pyro = 0;
            go(0,FAST, priority);
            go(1,SLOW, priority);
            active[priority] = 1;
            count = 0;
            }
        if (count>=50 && pyro_detect == 1) {
            temp = mtr0[priority];
            go(0,mtr1[priority], priority);
            go(1,temp, priority);
            active[priority] = 1;
            }

        if (count>=100 && active[priority]) {
            pyro_detect = 0;
            old_pyro = 1;
            active[priority]=0;
            fishtailf=1;
            }

        if (count>200) {
            pyro_detect=0;
            old_pyro=0;
            active[priority]=0;
            }
        wait(20);
        }
    }




/* light sensing routines and variables very low priority, can only
wake-up robot*/
int light_min;
int light_max;
int light_delta = 3;
int wait_t = 1000;

/* monitors light sensor*/
void look() {
   int temp;
   light_max = -1, light_min=256;

   while(1) {
      hog_processor();
      temp = analog(6);
      defer();
      if (temp > light_max) light_max = temp;
      if (temp < light_min) light_min = temp;
      wait(100);
      }
}

/* light change behavior, low priority can only wake up robot*/
void light() {
    int priority = 21;
      light_max = -1, light_min=256;
      while(1) {
        if (light_max-light_min>light_delta) {
            go(0,SLOW,priority);
            go(1,SLOW,priority);
            active[priority] = 1;
            wait(100);
            active[priority]=0;
            wanderf=1;
            }
        light_max = -1, light_min=256;
        active[priority]=0;
        wait(wait_t);
        }
   }



/* sound sensing routines and variables, very low priority, can only
wake-up robot*/
int noise_min;
int noise_max;
int noise_delta = 1;

/* continously reads sound sensor*/
void listen() {
   int temp;
   noise_max = -1, noise_min=256;

   while(1) {
      hog_processor();
      temp = analog(3);
      defer();
      if (temp > noise_max) noise_max = temp;
      if (temp < noise_min) noise_min = temp;
      }
}

/* wake robot up if sleeping*/
void noise() {
   int priority=22;
   noise_max = -1, noise_min=256;
   while(1) {
     if (noise_max-noise_min>noise_delta) {
            go(0,SLOW,priority);
            go(1,SLOW,priority);
            active[priority] = 1;
            wait(100);
            active[priority]=0;
            wanderf=1;
           }
     noise_max = -1, noise_min=256;
    active[priority]=0;
     wait(wait_t);
     }
   }


/* sleep is very low priority, is only activated if nothing else going on
    and also on startup. Sleeps periodically
   Priority 23.
*/
void go_to_sleep() {
    int priority =23;

    while(1) {
       active[priority]=1;
       go(0,0.0,priority);
       go(1,0.0,priority);
       wait(2000);
       active[priority]=0;
       wait(15000);
       wanderf=1;
       wait(3000);
       }

}


/*  wander is lowest priority, it gets set by most other behaviors when they
    are finished. This lets robot have a kind of non-intrusive default
    task.

    Priority 24.
*/
int wanderf;
void wander() {
    int priority =24;
    long mtime = mseconds();
    while(1) {
    while (!wanderf) defer();
       go(0,SLOW,priority);
       go(1,SLOW,priority);
       active[priority]=1;
       while(active[priority] == 1 && mtime - mseconds() < 500L) defer();
       active[priority]=0;
       wanderf = 0;
```

```
      wait(75);
    }
}


/*  fishtail is low priority, it gets set by read_pyro when pyro
    detect is stale. This routine wobbles the robot more, trying to
    recover detection. Middle priority behavior dealing with pyro

    Priority 19.
*/
int fishtailf;
void fishtail() {
    int priority =19;
    int i=0;
    while(1) {
      while(!fishtailf) defer();
      i=0;
      go(1,-NORMAL,priority);
      go(0, STOP,priority);
      active[priority]=1;
      wait(500);
      while (i<5 && !pyro_detect) {
        go(0, STOP,priority);
        go(1, NORMAL,priority);
        wait(1000);
        go(1,STOP,priority);
        go(0,NORMAL,priority);
        wait(1000);
        i++;
        }
      go(1,NORMAL,priority);
      go(0,NORMAL,priority);
      active[priority]=0;
      fishtailf = 0;
      wanderf=1;
      }

   }


/* these routines along with routines in speech.c handle talking
behavior. It is set up so that status will not be repeated unluss change
occurs in active behavior*/
float stat_wait;
int old_stat=-1;
int talking = 0;
void fresh_talk_status() {
    int i;
    stat_wait = seconds()+10.0;
    while (1) {
    while(stat_wait-seconds() <10.0) wait(500);
    old_stat=-1;
    stat_wait = seconds();
```

```
        }
    }

void talk_status(int process) {

   if (old_stat != process) {
      old_stat = process;
      while(talking == 1);
      talking=1;
      talk(process);
      stat_wait = seconds();
      talking = 0;
      }

}



/* subsumption arbitrator. Only let highest priority active behavior
control robots behavior. Uses very simple method to do this. Has array
for processes with status. Arbitrate goes through array and only
performs highest priority motor command*/
void arbitrate() {
   int done, i;

   while(1) {
      done = 0; i = 0;
      status=-1;
      while(i<25 && !done) {
         hog_processor();
         if (active[i]) {
            motor(0, mtr0[i]);
            motor(1, mtr1[i]);
            done = 1;
            active[i] = 2;
            status = i;
            }
         defer();
         i++;
         }
      if (status!=-1) start_process(talk_status(status));
      wait(10);
      }
   }


/* main routine is very simple. Checks if center IR gets stron return
if so exit main. Else say O.K. and start arbitrator and other behaviors.
All behaviors (except reporting status) run as seperate tasks. Therefore
main()  realling only acts as initializer for robot*/

void main() {
  /*turn off voice if one and init talking behavior*/
  quiet();
  init_words();

/* test if should turn off robot if large return from center IR */
  poke(0x2000,0xff);
  wait(100);
  if (analog(1) >120) {
    beep();
    wait(100);
    beep();
    return;
    }
  poke(0x2000,0x0);

  /*say ok*/
  talk(100);



  start_process(fresh_talk_status());

  /*start wander behavior*/
  start_process(wander());


  /* start subsumption arbitrator*/
  start_process(arbitrate());

  /*start sleep behavior*/
  start_process(go_to_sleep());

  /* start IR obstacle avoidance*/
  start_process(read_ir());
  start_process(ir_avoid());


  /*start light behavior*/
  start_process(look());
  start_process(light());

 /*start noise behavior*/
  start_process(listen());
  start_process(noise());


  /*let robot "sleep" for 7 seconds*/
  wait(7000);
  /* start pyro behaviors */
  start_process(fishtail());
  start_process(read_pyro());
  start_process(scan_pyro());
  }
```

```
/* speech.c - speech routines for robot using GI SPO256-AL2 Narrator
   chip. Written in IC.

    Joe Larkin
    EEL5934
    */


/*arrays to hold vocabulary string (phonem data) and pointers to words in that
string*/
char words[100];
int vocab[20];

/*initialize vocabulary - to add more words increase size of words[] and
  add new phonems strings. End each word with a pause to shut the chip
  up and seperate words with 255. Have vocab contain offset into array
  words[] to point to beginning of words */

void init_words() {
/*noise   */
words[0] = 56;
words[1] = 5;
words[2] = 55;
words[3] = 4 ;
words[4] = 255;
vocab[0] = 0;

/*light */
words[5] = 1;
words[6] = 45;
words[7] = 6;
words[8] = 17;
words[9] = 4;
words[10] = 255;
vocab[1] = 5;

/* IR */
words[11] = 24;
words[12] = 6 ;
words[13] = 3;
words[14] = 59;
words[15] = 4;
words[16] = 255;
vocab[2] = 11;

/*heat*/
words[17] = 27;
words[18] = 19;
words[19] = 2;
words[20] = 17;
words[21] = 4;
words[22] = 255;
vocab[3] = 17;
```

```
/* sleeping */
words[23] = 55;
words[24] = 45;
words[25] = 19;
words[26] = 9;
words[27] = 12;
words[28] = 44;
words[29] = 4;
words[30] = 255;
vocab[4] = 23;

/* scanning */
words[31] = 55;
words[32] = 55;
words[33] = 2;
words[34] = 42;
words[35] = 26;
words[36] = 11;
words[37] = 12;
words[38] = 44;
words[39] = 4;
words[40] = 255;
vocab[5] = 31;

/* o.k. */
words[41] = 53;
words[42] = 3;
words[43] = 42;
words[44] = 7;
words[45] = 20;
words[46] = 4;
words[47] = 255;
vocab[6] = 41;

/* zero */
words[48] = 43;
words[49] = 60;
words[50] = 53;
words[51] = 4;
words[52] = 255;
vocab[7] = 48;

/* one */
words[53] = 46;
words[54] = 15;
words[55] = 15;
words[56] = 11;
words[57] = 4;
words[58] = 255;

vocab[8] = 53;

/* two */
```

```
words[59] = 13;
words[60] = 31;
words[61] = 4;
words[62] = 255;
vocab[9] = 59;

/* three */
words[63] = 29;
words[64] = 14;
words[65] = 19;
words[66] = 4;
words[67] = 255;
vocab[10] = 63;

/* four  */
words[68] = 40;
words[69] = 40;
words[70] = 58;
words[71] = 4;
words[72] = 255;
vocab[11] = 68;

/* five */
words[73] = 40;
words[74] = 40;
words[75] = 6;
words[76] = 35;
words[77] = 4;
words[78] = 255;
vocab[12] = 73;

/* hunting */
words[79] = 27;
words[80] = 15;
words[81] = 11;
words[82] = 13;
words[83] = 12;
words[84] = 44;
words[85] = 4;
words[86] = 255;
vocab[13] = 79;

/* wander */
words[87] = 46;
words[88] = 23;
words[89] = 11;
words[90] = 21;
words[91] = 51;
words[92] = 4;
words[93] = 255;
vocab[14] = 87;


/*  Some additional words from G. McComb book
```

```
    I           24  6
    am            7  7 16
    a           24
    talking     13 23 23  2 42 12 44
    computer    42 15 16  9 49 22 13 51
    wordsing     13  7 55  2 13 12 44
    one         46 15 15 11
    two         13 21
    three       29 14 19
    four        40 40 58
*/
}

/* talk translates process priorities into words and then outputs
to SPO256-AL2 chip */
void talk(int i) {
    if (i==0) i = vocab[2];  /* IR */
    if (i==1) i = vocab[3];  /*heat*/
    if (i==19) i = vocab[13]; /*hunting*/
    if (i==20) i = vocab[5]; /*scanning*/
    if (i==21) i = vocab[1]; /*light*/
    if (i==22) i = vocab[0]; /*noise*/
    if (i==23) i = vocab[4]; /*sleeping*/
    if (i==24) i = vocab[14]; /*wander*/
    if (i==100) i = vocab[6];

    /*say the word - voice output port is at $2400*/
    while(words[i] != 255) {
         poke(0x2400,0x00);
         poke(0x2400,words[i]);
         while(peek(0x1000)<128) {};
         i++;
      }

   }


/*just say word number i in words[]*/
void say(int i) {
    i = vocab[i];
    while(words[i] != 255) {
         poke(0x2400,0x00);
         poke(0x2400,words[i]);
         while(peek(0x1000)<128) {};
         i++;
      }

   }


/*output a pause - good if routines crash or if need to shutup quick*/

void quiet() {
```

```
poke(0x2400,0x00);
poke(0x2400,0x04);
while(peek(0x1000)<128) {};
}
```

```
/* util.c - some utility routines for robot prgrammed in IC

Joe Larkin
EEL5934
*/



int status;
float m0, m1;

/*returns sign of oerand*/
int sgn(int i) {
    if (i<0) return -1;
    return 1;
    }


/*wait using defer better than sleep()*/
void wait(int milli_seconds) {
   long timer_a;
   timer_a = mseconds() + (long) milli_seconds;
   while (timer_a > mseconds()) {
      defer();
      }
}


/*report active status using piezo speaker*/
void sound_status() {
    int i;

   while(1) {
    wait(800);
    for (i=0; i<status; i++) {
        wait(40);
        tone(800.0, 0.04);
        }
    }
   }

/*return abs value of operand*/
int abs(int i) {
   if (i>0) return(i);
   else return(-i);
   }
```

PALCE22V10 PDS Design File

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    Robot 1 pal for address decode for IR latch and speech port
PATTERN
REVISION
AUTHOR   J Larkin
COMPANY  Dept. of Electrical Engineering U of Florida
DATE     04/05/95

CHIP  _rob1  PAL22V10

;---------------------------------- PIN Declarations ---------------
PIN  1          A15                                  ;
PIN  2          A14                                  ;
PIN  3          A13                                  ;
PIN  4          A12                                  ;
PIN  5          A11                                  ;
PIN  6          A10                                  ;
PIN  7          RW                                   ;
PIN  8          E                                  ;
PIN  9          AS                                  ;
PIN  10         NC                                  ;
PIN  11         NC                                  ;
PIN  12         GND                                  ;
PIN  13         NC                                  ;
PIN  14         NC                                  ;
PIN  15         NC                                  ;
PIN  16         NC                                  ;
PIN  17         NC                                  ;
PIN  18         NC                                  ;
PIN  19         NC                                  ;
PIN  20         NC                                  ;
PIN  21         NC                                  ;
PIN  22         /LATCH2                                  ;
PIN  23         /LATCH1                                ;
PIN  24         VCC                                  ;

;----------------------------------- Boolean Equation Segment ------
EQUATIONS
LATCH1 = /A15 * /A14 * A13 * /A12 * /A11 * /A10 * /RW * E
LATCH2 = /A15 * /A14 * A13 * /A12 * /A11 * A10 * /RW * E
;---------------------------------- Simulation Segment ------------
SIMULATION

;-------------------------------------------------------------------
```

SPO256-AL2 Data Sheets

(not included in HTML document)