EEL 5666C - Intelligent Machine Design Lab

Sicherhietsuberprufung

Matt Lysaught

12.4.2001

Table of Contents

Abstract

Sicherheitsuberprufung is the German word for security check. This report describes the design of an autonomous machine originally intended to serve in a personal security capacity. This robot should respond to its environment based upon input from several sensors processed by its CPU.

Executive Summary

The original design goals for Sicherheitsuberprufing called for a security robot that would detect a breach by the sound of broken glass. The design also specified that the robot would be able to find its recharging station and recharge itself, thereby performing its duties indefinitely, unaided by humans.

The detection of broken glass was to be performed by using a DSP board to sample sound and detect a specified threshold of energy in the range of four to six kilohertz. However, the DSP board proved to be beyond the capacity of the University of Florida Department of Electrical and Computer Engineering to teach a student with no experience with DSP technology. Thus, the sound recognition behavior was dropped from the project, leaving a robot that can do nothing but run continuously.

The robot will be able to determine when it is low on batteries, and head directly to its recharging station. Once the robot has charged its batteries to the desired level, it will resume its aimless wandering.

Introduction

The need for a robot to be self sufficient is explicit from its definition as an autonomous machine. A robot that runs out of batteries thereby requires outside assistance to allow it to continue its mundane task. Finding this unacceptable, extensive work has been done giving robots the capability to recharge themselves.

To recharge itself, the robot must first find its recharging station. Thus, a way to differentiate its home base from the rest of the surroundings is required. For this experiment, a bright light was used because a security robot would be utilized mostly at night, in the absence of other interfering light sources.

Once the recharging station is identified by the robot, the robot must find a way to dock with it to receive power. This presents the option of DC power or AC power. Both have advantages and disadvantages. AC power does not require that polarity is observed, but requires that the robot have a rectifier onboard. DC power requires that the robot align itself properly with the recharging station. For this project, DC power was utilized.

Integrated System

The integrated system for Sicherheitsuberprufung consists mainly of the Mekatronix MRC11 and MRSX01. The MRC11 provides a Motorola MC68HC11 microprocessor with 64kB of RAM. The MRSX01 provides a multiplexed hardware expansion to increase the number of sensors available to the MRC11. The integrated system also includes two servos, two IR detectors, and three CdS cells.

Mobile Platform

The platform for Sicherheitsuberprufung was designed using AutoCAD 2000. The platform design is in the shape of a circle similar to the Mekatronix TJ to make obstacle avoidance as easy as possible. There is a horizontal level of the platform that was designed specifically for the placement of CdS cells. The platform was cut using the T-tech machine in the IMDL lab.

Actuation

The actuation for Sicherheitsuberprufung is provided by two servos. The servos are partially hacked to provide a motor with built in speed control. The partially hacked servo's speed is controlled by the pulse width modulating capabilities of the HC11. The servos are used to drive the two wheels on the base of the platform. Servo control is determined by obstacle avoidance code, and by CdS cell resistance levels during recharge mode.

Sensors

For obstacle avoidance, Sicherheitsuberprufung employs four bump switches and two infrared distance detectors. One bump switch is placed in the back, and three are placed in the front of the platform. The switches are surrounded by a wooden ring to ensure that they are pressed when the robot encounters an obstacle. The IR distance sensors are placed on the front of the platform to prevent the robot from hitting anything when moving forward. There were originally three IR sensors, but one was destroyed. This problem resulted because the data sheet for the Sharp GP2D12 was difficult to find. It was not on the Sharp web page, so instead, Professor Doty was consulted on the pinout. The information he provided was incorrect and cost me $10. The actual pinout for the IR is Vout – Gnd – Vdd.

To find its recharging station, Sicherheitsuberprufung employs three CdS cells. These photo resistors help the robot find the light emitted by the base. The CdS cells, the bump switches, and the IR detectors are connected to the analog port of the MRC11 through a multiplexed voltage divider network on the MRSX01.

Behaviors

Obstacle avoidance is performed by utilizing the bump sensors and IR sensors with some simple code. When a bump sensor is pressed, the robot will move a short distance in the opposite direction, and then turn a random amount before heading forward again. When an object is placed in the path of the IR detectors, the robot turns until there is nothing in front of it.

Finding the recharging station is performed using the CdS cells. The placement of the CdS cells can be seen in Figure 1 below. This figure depicts a horizontal cross-section of the robot. When a light is present in the light blue area, the left servo drives forward. When a light is present in the yellow area, the right servo drives forward. When light is present in the green areas, the robot moves forward. This allows the robot to move towards an intense light source. The

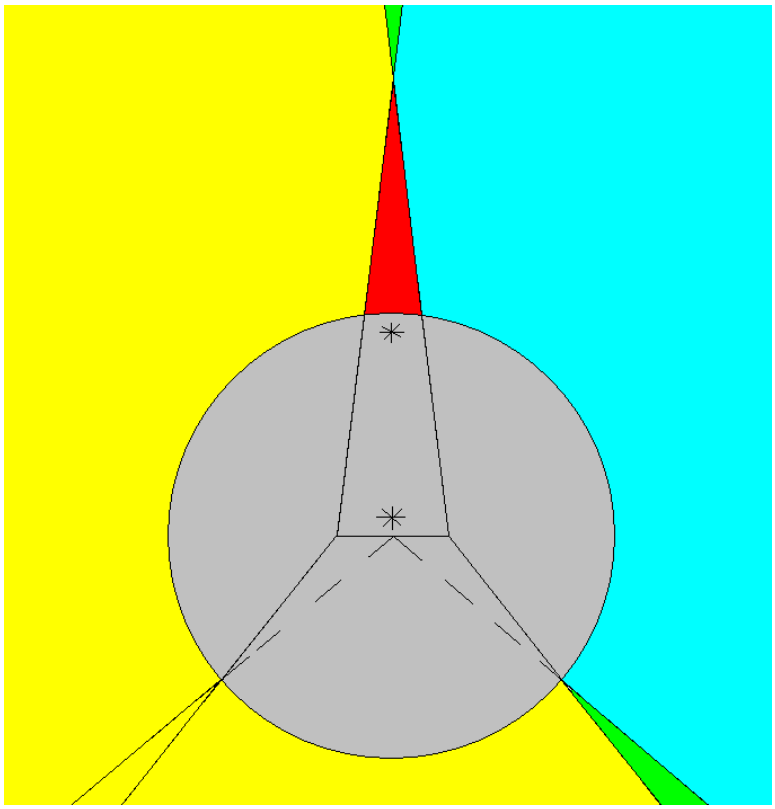

Figure 1

recharging station used by the robot consists of a light surrounded by two circular contacts. The inner ring is the positive supply, and the outer ring is ground. These contacts are made of aluminum foil. A top view of the recharging station is shown in Figure 2 on the right. The robot has two charging contacts that hang from the bottom of its platform. When these contacts touch the aluminum foil rings of the recharging station, the robot stops to recharge. Once the robot has completed recharging, it backs away from the station and resumes its previous activity.

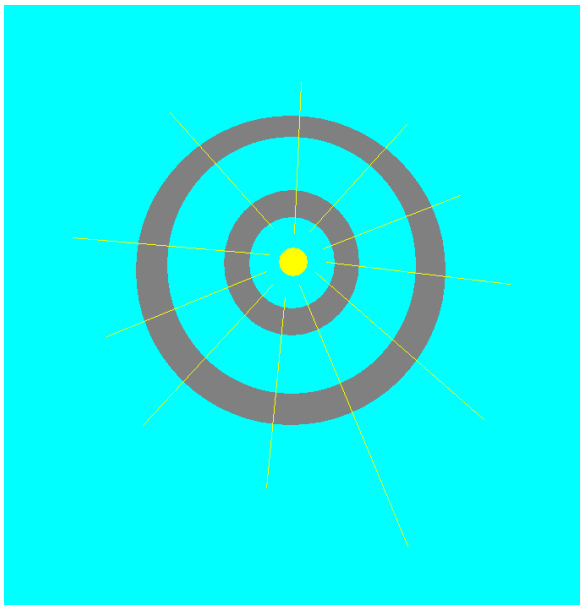

Figure 2

Conclusion

The robot avoids obstacles rather well. This is expected because obstacle avoidance code can be easily adapted to a circular platform.

It was a mistake to even consider using a DSP board to analyze sound. This takes a tremendous amount of work because no one at the school seems to know how to use one. I was not able to produce any notable results with the DSP.

The recharging program worked remarkably well. The robot functioned in a brighter area than expected. The robot is able to go directly to its recharging station nearly 100% of the time.

Appendices

Obstacle Avoidance Code

```
/*****************************************************************************
*                                                                            *
*                                                                            *
 * Title        avoid.c                                                      *
 * Programmer Matt Lysaught                                                  *
 * Date         Novemer 27, 2001                                             *
 * Version      3                                                            *
 *                                                                           *
 * Description                                                               *
 *    A very simple collision avoidance program.                             *
 *    Adapted from Talrik collision avoidance code.                          *
 *                                                                           *
*                                                                            *
*****************************************************************************
/

/****************************** Includes *********************************/
#include <tkbase.h>
#include <stdio.h>
/************************* End of includes ******************************/

/****************************** Constants ********************************/
#define IR_THRESHOLD 100
#define BUMPER_FUZZY_ZERO 10 /* Noise immunity for bumper readings */

/*Constant for stopping servo0*/
#define STOP_ZERO 2960

/*Constant for stopping servo1*/
#define STOP_ONE 3010

/*Constant for driving servo0 forward*/
#define FORWARD_ZERO (STOP_ZERO - 100)

/*Constant for driving servo0 forward*/
#define REVERSE_ZERO (STOP_ZERO + 100)

/*Constant for driving servo1 forward*/
#define FORWARD_ONE (STOP_ONE + 100)
```

```
/*Constant for driving servo0 forward*/
#define REVERSE_ONE (STOP_ONE - 100)

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

/***************************** Prototypes ******************************/
void turn(void);
/************************* End of Prototypes ****************************/

/******************************** Globals ******************************/

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


void main(void)
/******************************** Main *********************************/
{
unsigned int IR_delta[NIRDT], IR_Threshold[NIRDT];
int i, rb, rspeed, lspeed, delta_rspeed, delta_lspeed;

  init_analog();
  init_servos();
  init_clocktk();
  init_serial();

/*Start TALRIK moving forward when back bumper is pressed*/
  while(rear_bumper()<BUMPER_FUZZY_ZERO);

  servo(1,FORWARD_ONE);
  servo(0,FORWARD_ZERO);


  while(1)
  {

/* The following block will read the IR detectors and decide
   whether TALRIK needs to turn to avoid any obstacles
*/
    read_IR();

  if((IRDT[4] > IR_THRESHOLD ) || (IRDT[5] > IR_THRESHOLD ))
    {
      if(IRDT[4] > IRDT[5])
      /* Start turning when something in front and keep turning until nothing
        is in front */
```

```
        {
          servo(1,REVERSE_ONE);
          servo(0,FORWARD_ZERO);
            }
       else
            {
          servo(1,FORWARD_ONE);
          servo(0,REVERSE_ZERO);
               }

/*      while((IRDT[4] > IR_THRESHOLD ) || (IRDT[5] > IR_THRESHOLD ))
                    read_IR();                              */

    }
  else
    {
      servo(1,FORWARD_ONE);
      servo(0,FORWARD_ZERO);
    }

/* This "if" statement checks the front bumper. If the bumper is pressed,
   TALRIK will back up, and turn.
*/

    if(front_bumper()>BUMPER_FUZZY_ZERO)
    {
      servo(1,REVERSE_ONE);
      servo(0,REVERSE_ZERO);
      wait(450);
      turn();
    }

    if(rear_bumper()>BUMPER_FUZZY_ZERO)
    {
      servo(1,FORWARD_ONE);
      servo(0,FORWARD_ZERO);
      wait(450);
      turn();
    }

  }
}
/***************************** End of Main ******************************/

void turn()
```

```
/*****************************************************************************
*                                                                            *
 * Function:  Will turn in a random direction for a random amount of         *
 * time                                                                       *
 * Returns:  None                                                             *
 *                                                                            *
 * Inputs                                                                     *
 *   Parameters: None                                                         *
 *   Globals:    None                                                         *
 *   Registers:  TCNT                                                         *
 * Outputs                                                                    *
 *   Parameters: None                                                         *
 *   Globals:    None                                                         *
 *   Registers:  None                                                         *
 * Functions called: None                                                     *
 * Notes:                                                                     *
*****************************************************************************
/
{
 int i;
 unsigned rand;

 rand = TCNT;

 if (rand & 0x0001)
 {
   servo(1,FORWARD_ONE);
   servo(0,REVERSE_ZERO);
 }
 else
 {
   servo(1,REVERSE_ONE);
   servo(0,FORWARD_ZERO);
 }
 /*for (i = 0; i < rand; i++);*/
 i=(rand % 1024) + 35;
 wait(i);

}
```

Recharging Base Finding Code

```
/*****************************************************************************
*
 * Title        gohome.c
 * Programmer Matt Lysaught
 * Date         November 19, 2001
 * Version        1
 *
 * Description
 * Programs robot to drive to its home base
 * and stop there.
 *
 * Use Hyperterm with the following settings:
 *
 *      COM1, 9600 Baud, 8 bits, No Parity, 1 Stop bit, No Flow
 *      of Control, VT100 Terminal, WrapLines.
 *
 *Usage: Sends robot to recharge

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

/****************************** Includes *********************************/

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

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


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

/*Constant to check whether base is in view*/
#define BASE_IN_VIEW 70

/*Constant for driving servo0 forward*/
#define FORWARD_ZERO 2900

/*Constant for stopping servo0*/
#define STOP_ZERO 2960

/*Constant for driving servo1 forward*/
#define FORWARD_ONE 3070

/*Constant for stopping servo1*/
```

```
#define STOP_ONE 3010

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


void main(void)
/****************************** Main *********************************/
{
  unsigned int cv;

  init_analog();
  init_clocktk();
  init_servos();


  cv = charge_volts();

while(cv < 80)  /* Update the sensor fields indefinitely */
  {

  read_CDS();    /* Read all 6 CDS photoresistor voltage dividers */

  if ((CDS[3] > CDS[5]) || (CDS[4] > CDS[5]))
          {
          servo(1,FORWARD_ONE);
          servo(0,STOP_ZERO);
          }
  else
          {
          servo(1,STOP_ONE);
          servo(0,FORWARD_ZERO);
          }

  cv = charge_volts();

  } /*end while*/

  servo(0,STOP_ZERO);
  servo(1,STOP_ONE);

}

/***************************** End of Main ******************************/
```