SCOUT

James Houben

University of Florida

Intelligent Machines Design Laboratory

EEL 5666

Spring 1998

Table of Contents

Abstract

There are many problems involved in the mapping of a room with an autonomous mobile robot. They include getting the robot to move in a straight line, avoid obstacles, keep from getting stuck in a corner, determining walls from furnishings, remembering the path it has taken, and covering the entire area. Theses are the issues that I took on when creating Scout. I did accomplish many of these tasks, but some were never realized or did not work as well as hoped. This report walks through the design and the behaviors of my robot as well as how these above mentioned tasks were undertaken.

Executive Summary

Scout is an autonomous mapping robot consisting of a circular wooden platform, four servo motors, 16 AA batteries, a microcomputer, and several sensor arrays. It was designed, constructed, wired, and tested in three and a half months for less than $300.

I chose the task for this robot because I had many ideas that I believed would solve some of the more common problems faced by others trying to complete the task of mapping a room before me. I succeed in many areas and failed in others, but I felt I had to make my attempt to continue the evolution of robots.

The main problem I consistently ran into during development was that of not being prepared for the task at hand. I had to relearn C to interact with IC. I also had little idea where to start or where to go for help starting. No other college courses I had partaken in prepared me for what was ahead.

Once the jitters were gone the only other problem to be faced was the of building a structurally sound and well programmed machine in a short amount of time.

Introduction

Scout is designed to autonomously map a room using only its on board sensors to interact with its environment. These sensors relay data to the brain of the robot which intern decides the next course of action. The action taken is based on the task at hand and the problems detected. All actions possible are programmed into Scout and chosen based on the findings. This report explains the integrated systems, platform, sensors, and behaviors chosen to accomplish the task at hand. Mapping a room.

Integrated System

Scout is controlled by a MC68HC11 micro-controller with 32k of SRAM and a MSCC11 board. These two boards joined together consist of the brain work for Scout. For the ease of programming and debugging I chose to use IC as the interface language between myself and the robot. I also chose to use to use a simple frame and only two types of sensors. This was meant to keep the cost low and the focus on it's programming.

Platform

I designed Scout using a Talrik© frame. I chose this design because I wanted a platform capable of making 90° turns. By using a circular frame and two opposing motors, I could easily achieve this. I also chose this frame because I could mount a pan-tilt sensor array on the bridge and it's open wheel design had many options for wheel encoders to be placed.

Actuation

Scout uses four servos, two hacked to act as motors and twounhacked used in the pan-tilt sensor array. The two hacked servos act as the drive system of the robot and are mounted opposite of each other. This means that they can be run in opposite directions to perform

turns. The two servos are mounted on the bridge of the robot. One is mounted vertically and used to turn the sensor array 180° on the horizontal plane. The other is mounted horizontally onto the top of the first and is used to pivot the sensor 180° in the vertical plane.

Sensors

There are four different sensor systems on Scout. They consist of an obstacle avoidance sensor suite, two wheel encoder pairs, bump sensors, and a pan-tilt sensor array. Each sensor system is described below.

The first sensor system constructed was for obstacle avoidance. This sensor consists of two analog IR emitter/detector pairs attached to the front of the robot and directed forward at slight angles to each side. The idea of these sensors is to detect if an object is present by bouncing an infrared beam off of the object. This beam is then seen by the detector and given a value by the A/D converter on board the MC68HC11. From this value the proximity of the object can be determined.

The second sensor system is the bump sensors. They consist of two switches mounted beside the obstacle avoidance pairs. They are only switches that detect when they have been depressed. These are used to find objects the obstacle avoidance pairs failed to locate and to prevent the robot from doing harm to itself.

The third system used is the wheel encoder sensors. They consist of a digital IR emitter/detector pair each. These act the same way as the analog pair except that they work several times faster and can only tell if the object is present, not the objects

proximity. The purpose of these pairs is to count the number of white areas that pass before them on the wheels and thusly determine the distance the wheel travels.

The remaining sensor system is the pan-tilt sensor array. This device uses two servos mounted on top of the bridge to scan an area for an object. The array uses an IR cannon consisting of several infrared emitters working together to create an intensified beam and a single detector. The purpose of the emitter/detector pair is the same as the other systems, but the use of more emitters increases the range of the array.

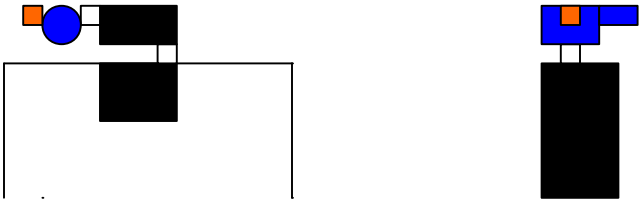

Figure 1: Front and side view of sensor array.

Behaviors

Scout has one main behavior, but must use several other behaviors to accomplish its main objective of mapping. First is must locate an object using its obstacle avoidance IR pairs. Once this object has been detected it must decided if it is a wall or a minor obstacle. This is accomplished using the pan-tilt array. The array is designed to scan up a object until it no longer senses the object. Thusly finding the top and knowing the height.

The next thing it must be able to do is to determine the distance traveled. This is accomplished using the wheel encoders. The wheel encoders also provide another valuable task. That of keeping the robot running straight. This task is accomplished by

monitoring the distance traveled by each wheel and altering the speed the motors until they read the same distance.

The robot must also be able to store a reasonable replica of the room in memory. This task is taken care of by using an array stored in memory with different numbers entered into each location based on the height of the objects detected. This map also serves another purpose, that of knowing where the robot has and has not been. By placing yet another different number into the memory array, the robot can determine where it has and has not been.

Conclusion

During the construction of this robot, I learned many valuable things. I learned that engineering is not just creating new things, but also learning how to adapt what has already been designed to fit the needs of the task you are trying to complete. I also learned patience is a vital part of engineering. Most days things will go wrong and one must have the desire to work through them and concentrate on the task at hand. I also learned that there are many engineers out there that need help and I am one of them and that working on a project alone does not mean that you can not get help.

I learned more physical things as well. I learned the programming language IC, which intern made me relearn C. I also learn how to solder and became quite good at it as well. Lastly, I learned that it is best to take small steps and to trust that nothing still works just because it has not been messed with since it did.

Appendices

Code

```
/*************************************************************/
/***   Scout Main Program                                  ***/
/***   James Houben                                        ***/
/***   Intelligent Machines Design Labs                    ***/
/***   University of Florida                               ***/
/***   Spring 1998                                         ***/
/*************************************************************/

/* Global Variables & Constants */
int right, left, ticks1, ticks2, low, saved, height,  found_wall, found_corner;

   right=analog(0);
   left=analog(4);
   top=analog(3);

/* Servo motor positions in degrees */
float   servo_right = 180.0, servo_left = 0.0, servo_front = 90.0,
       low = 30.0;

/*  Start of Main Program   */
void main()
{
found_wall =0;
found_corner =0;
while (found_wall ==0)
{
  find_object();
 }
while (found_corner ==0)
{
  find_corner();
}


/*   Locate starting Point   */
void find_object()
{
  if (right <109 && left <109)
  {
    motor(0,100.0);
    motor(1,100.0);
   }
```

```
  else
  {
    if (right >=109 && left >=109)
    {
       motor(0,0.0);
       motor(1,0.0);
       scan();
       find_height();
       turn_left();
       found =1;
     }
    else
    {
      if (right >=109 && left <109)
      {
        motor(0,20.0);
        motor(1,50.0);
       }
      else
      {
        if (right <109 && left >=109)
        {
          motor(0,50.0);
          motor(1,20.0);
         }
       }
     }
   }
}


void find_corner()
{
pan(180);
if (top >=109)
{
  if (right >=109 && left >=109)
  {
    motor(0,0.0);
    motor(1,0.0);
    found_corner =1;
   }
  else
  {
       if (right >=109 && left <109)
       {
```

```
          motor(0,20.0);
          motor(1,50.0);
         }
        else
        {
          if (right <109 && left >=109)
          {
            motor(0,50.0);
            motor(1,20.0);
           }
         }
      }
 }
}


void pan(int degree1)
{
if (degree1 > present_degree1)
{
while (degree1 > present_degree1)
   {
   servo_deg1((float)present_degree1);
   present_degree1 = present_degree1++;
   msleep((long)2);
 }
}
else
{
   if (degree1 <= present_degree1)
   {
      while (degree1 < present_degree1)
     { servo_deg1((float)present_degree1);
       present_degree1 = present_degree1--;
       msleep((long)2);
      }
    }
 }
}


void scan()
{
degree2 =30;
tilt();
if (analog(3) >= 109)
```

```
{
  degree2 = degree2 +5;
  msleep((long)10);
  tilt();
 }
}


void tilt()
{
if (degree2 > present_degree2)
{
  while (degree2 > present_degree2)
  {
  servo_deg2((float)present_degree2);
  present_degree2 = present_degree2++;
  msleep((long)2);
   }
 }
else
{
  if (degree2 <= present_degree2)
  {
    while (degree2 < present_degree2)
    { servo_deg2((float)present_degree2);
      present_degree2 = present_degree2--;
      msleep((long)2);
     }
   }
 }
}


void counter()
{
ticks1 =0;
ticks2 =0;
while (1)
{
  if (analog(3) <= 109 && analog(6) <= 109)
  {
    ticks1 = ticks1++;
    ticks2 = ticks2++;
    while (analog(3) <= 109 && analog(6) <=109)
    {
     }
```

```
    }
   else
   {
     if (analog(3) <= 109)
     {
       ticks1 = ticks1++;
       while (analog(3) <= 109)
       {
        }
      }
     else
     {
       if (analog(6) <= 109)
       {
         ticks2 = ticks2++;
         while (analog(6) <= 109)
         {
          }
        }
      }
    }
 }
}


void turn_right()
{
  motor(0,-50.0);
  motor(1,50.0);
}


void turn_left()
{
  motor(0,50.0);
  motor(1,-50.0);
}
```