An Autonomous Agent

EEL 5934 Robotics Project

Spring 1995

Mark Skowronski

Intelligent Machines Design Laboratory

Department of Electrical Engineering

University of Florida

TABLE OF CONTENTS

ABSTRACT

The goal of the project An Autonomous Agent is to build a 'small, microprocessor-controlled, electronically sensualized, autonomous agent that exhibits various tasking behaviors'. The agent incorporates collision-avoidance, straight-line, and berserking behaviors to achieve its specific application--to model an industrial-strength safety robot that safely removes strewn nails from a construction site. Infrared (IR) LEDs along with Sharp IR detectors enable the agent to avoid collisions. A near-IR emitter-detector pair from Siemens is used to count the number of revolutions of a gear on the power train for each wheel. This feedback is interpreted by a Motorola MC68HC11 EVBU board to achieve straight-line motion. The berserking behavior activates periodically to point the robot in a random direction. A product of collision avoidance is the random motion required to effectively cover an area strewn with nails or other metal debris.

EXECUTIVE SUMMARY

An Autonomous Agent is a joint project for EEL 4914--Senior Design and EEL 5934--Robotics. Its goal is to develop a 'small, microprocessor-controlled, electronically sensualized, autonomous agent that exhibits various tasking behaviors'. In addition, this agent models an industrial-strength robot that removes nails strewn around a construction area in an effort to decrease the possibility of injury at the work site. To achieve this safety application, the agent incorporates several subsystems: propulsion, guidance, interaction with the environment, and control. To keep costs reasonable, the size and shape of the robot are scaled down to the size of a small pet. Two differentially-driven servo motors provide locomotion for the Lego-based agent, and the Motorola MC68HC11 EVBU microprocessor board provides 'intelligence' for controlling the motors and making decisions. The agent uses feedback from infrared (IR) emitter/detector sensors to determine object proximity as well as wheel speed. By measuring through the A/D ports the reflected IR intensity emitted by the IR LEDs, the agent senses the presence of objects in its vicinity. The robot counts the revolutions of an encoded gear in the servo gear train by facing a near-IR emitter/detector sensor toward the gear and measuring the amount of reflected light with a digital input port. The random motion of the robot in the environment as it avoids collisions provides the 'sweep patterns' necessary to effectively cover the work space. To increase the motion randomness, a berserking behavior activates periodically. This feature causes the robot to spin for a brief time then stop when the front of the robot is clear of objects. The agent picks up nails off the ground and stores them onboard using a permanent magnet and a turntable. The magnet attracts the nails and holds them against the spinning turntable. The turntable pulls the nails away from the magnet's grip, where they eventually fall into a collection pan located between the turntable and the ground. Sources of information and ideas include The 6.270 Robot Builder's Guide, Fred Martin; Mobile Robots: Inspiration to

Implementation, Joseph Jones and Anita Flynn; and handouts and advice from the Robotics class.

INTRODUCTION

In order to build an autonomous agent, the following features are necessary: power to move around, the ability to sense surroundings, a mechanism to interact with the environment, and intelligence to control and make decisions. Two differentially-driven modified Futaba servos provide the power to move the robot--the modification was developed in a previous semester of EEL 5934. Infrared Sharp detectors and near-infrared Siemens emitter/detector pairs provide information about collision avoidance and straight-line motion, respectively. The standard for on-board intelligence is the Motorola MC68HC11 EVBU board studied in EEL 3701C and EEL 4744C--a familiar microprocessor capable of the necessary level of computation required for this project. Instead of programming in assembly language, a version of C called Interactive C (IC) offers the ease of coding the necessary behaviors in a high-level language as well as a structure to run multiple processes 'simultaneously'. IC was developed by Randy Sargent and Fred Martin of the MIT Media Laboratory for an MIT undergraduate design course and made available through EEL 5934.

The mechanism to remove nails from the ground and store them on the robot is the permanent magnet/turntable design. Originally a treadmill would provide the power necessary to pull the nail away from the magnet, but Scott Jantz--a TA for Robotics--recommended that a turntable design would be easier to build and probably more reliable. In an effort to conserve power, an ungeared 5 V dc motor--powered by 5% duty cycle pulses--spins the turntable at about 20 rpm. Although ungeared, the motor still has enough torque to pull nails away from the magnet field. The pulses are produced by a

TLC555 Timer chip at about 10 pulses per second--pulse width modulation is cheaper and easier to implement and install than a gear train.

BODY

Integrated Hardware Systems

The robot is made of the following subsystems: Platform, 6811 EVBU board, Battery Pack, Servos, Shaft Encoders, Sharp boxes/IR, Motor/Turntable, and Magnet.

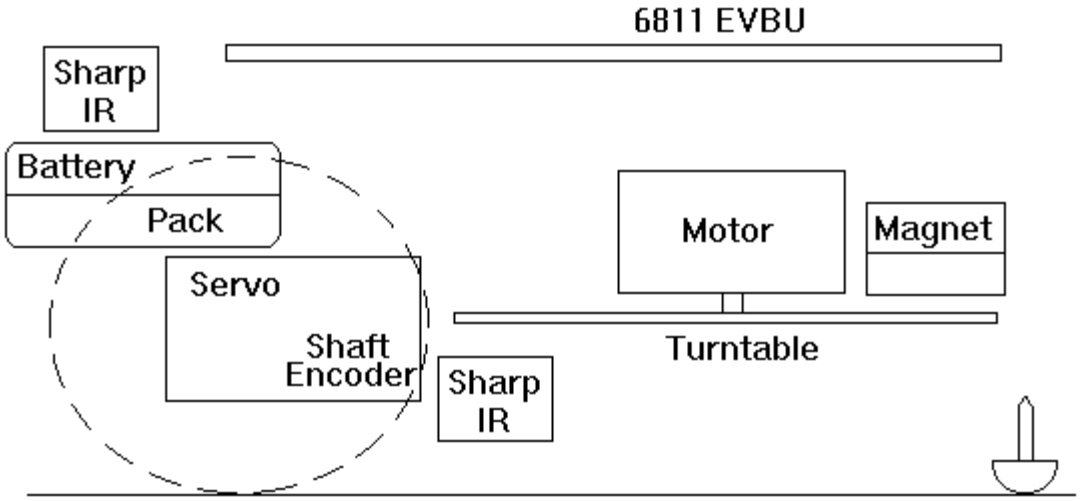

Figure 1: Robot Subsystems--Side View

Figure 1 shows the arrangement of the subsystems. The shaft encoders are mounted inside the servo housings, and the turntable connects to the motor shaft, which rests on the platform along with the magnet. The IR Sharp detector boxes mount above the battery pack in front of the wheel and also directly behind the wheels. All power is directed from the 6811--the 6811 gets power from the 2x4 AA battery pack. The 6811 board also supports all logic circuits and auxiliary resistors and capacitors.

Mobile Platform

The platform is made of Legos and provides an interconnection for the other system

components. Non-Lego parts are connected to Lego parts by hot glue, silicone glue, or nuts and bolts. Legos offer a variety of advantages: available, light weight, and versatile in design and construction. During several phases of the project, I enhanced the existing platform to incorporate a new idea or a new feature. The ease of working with Legos shifts emphasis to the structure design and eliminates the need for special tools or interconnections. However, adjoining pieces only hold together well when connected by several rows of Lego nodules--extra strength is added to critical joints with glue. Most of the Legos used are the thin plates--they provide more cross-bracing as well as the thinnest Lego building unit available for greater precision in thickness.

Locomotion

Two modified Futaba servos provide motion for the robot. The servos offer several advantages: inexpensive, reliable, recommended, geared, easily mountable, capable of moving the intended load. EEL 5934 provided a demonstration on how to modify the servos, which included the removal of the internal potentiometer, logic chips, and gear stop as well as the connection of leads directly to the motor terminals. Figure 2 shows a diagram of the servo before and after the modification.

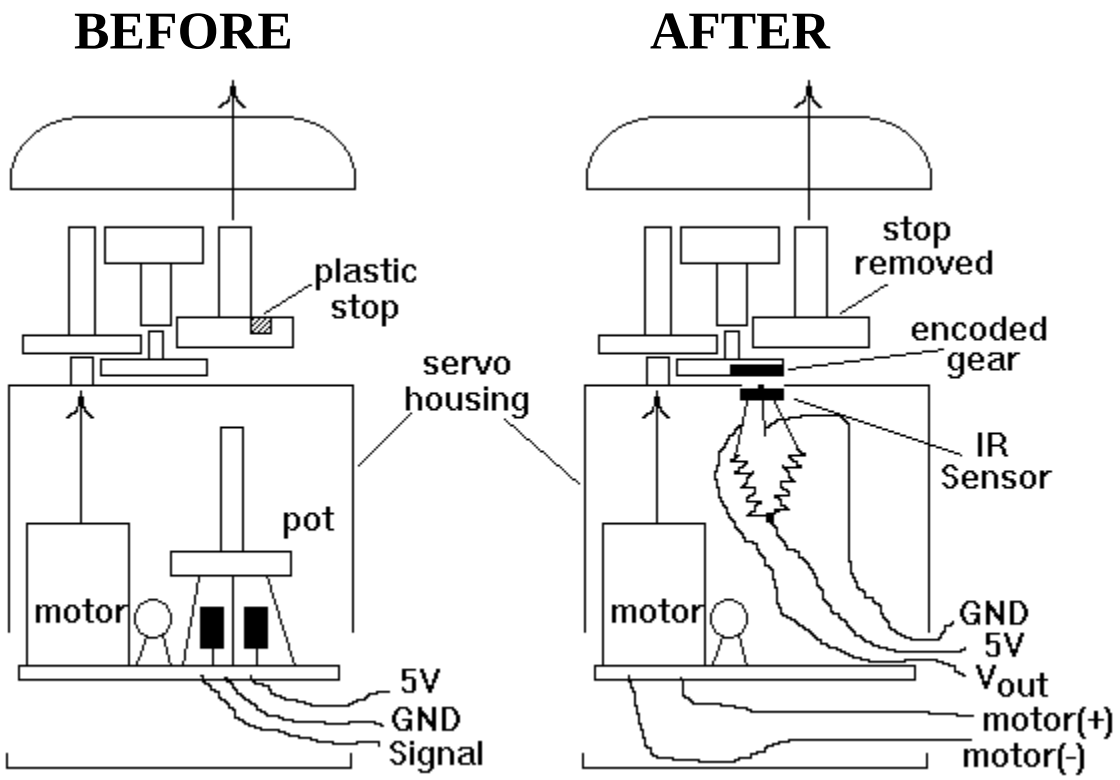

Figure 2: Modification of Futaba Servo

Each servo powers one wheel--this differential configuration allows the robot to rotate about itself. Turning is easy, but moving in a straight line becomes difficult. Assuming that both wheels are identical in size, weight, and coefficients of friction, a straight-line path can be realized if both wheels are spinning at the same rate. Shaft encoders are used to measure the speed of each wheel.

Shaft Encoders

Shaft encoders work on the following principle: IR reflects off white surfaces and does not reflect off black surfaces. A disk made of black and white segments attached to the spinning wheel will alternate between black and white as seen by a stationary sensor positioned in front of the segments. This periodic change is detected by an IR emitter/detector sensor pair and correlates to a wheel speed. The Siemens SFH900 IR emitter/detector is an ideal sensor for this application because of its small range and small

packaging--provided to EEL 5934 by Erik de La Inglasia. Chad Huff--a student in EEL 5934--suggested mounting the sensor inside the servo housing and using the first gear off the motor shaft as the encoder wheel. This design cuts out ambient light and provides a potentially large encoder rate because of the high-speed gear.

The useful range of the sensors is 1 mm, which means the encoder wheel can not wobble more than 1 mm in the axial direction. The high-speed gear is already fixed with respect to the servo housing, so the servo gear will not wobble significantly with respect to this tolerance. Figure 3 shows a diagram of the position of the sensor inside the servo housing.

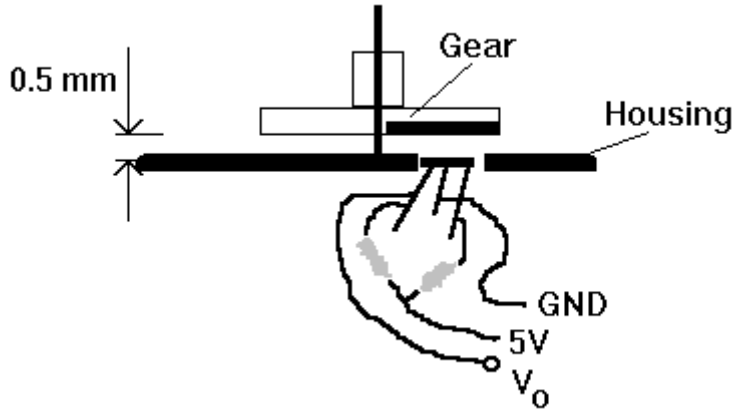

Figure 3: Position of Siemens Sensor Inside Servo

Notice in Figure 3 that the body of the sensor is pushed inside the hole made in the housing. The housing wall is thicker than the 1 mm range of the sensor. Figure 4 shows the biasing circuit for the Siemens sensor.

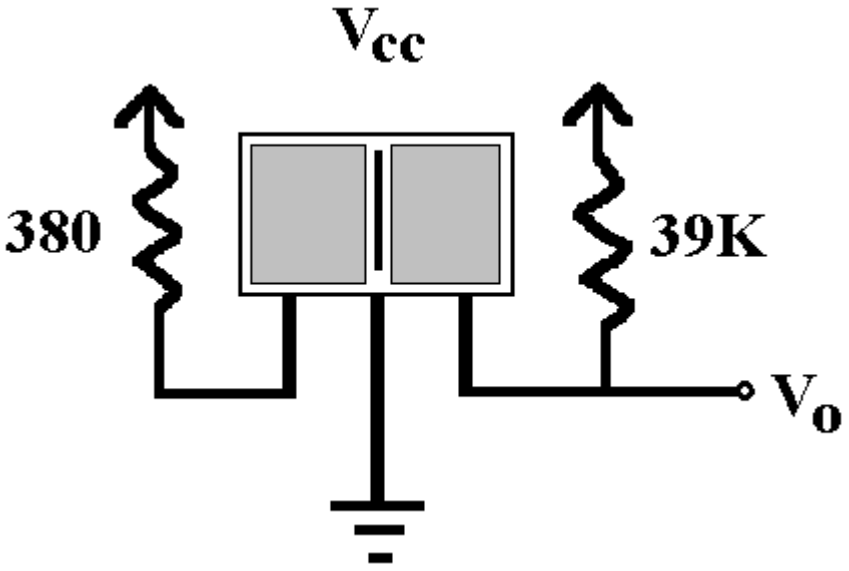

Figure 4: Shaft Encoder Circuit--Front View

The analog output wave swings between 1.0 V and 3.8 V--satisfactory for digital input straight into the pulse accumulators of the 6811. The straight-line behavior (code in appendix) uses the shaft encoders to get some feedback on the speed of the wheels. This version sets the left wheel at a preset speed and adjusts the right wheel in 2% increments in order to equalize the shaft encoder ticks per time period. The left wheel is the standard speed because it is about 10% faster than the right wheel for the same motor command. This is most likely due to friction inside the gear train.

IR Sharp Boxes

The Sharp GP1U58X modulated IR detectors work on the following premise: each input a 40 KHz modulated IR signal and output a 0-5V analog voltage proportional to the IR intensity. The robot emits IR light in front of a Sharp detector through IR LEDs. That light is reflected off an object and detected by the Sharp sensor--the closer the object, the more intense the reflected light and the greater the analog output voltage of the Sharp box. That analog signal is sent to an A/D port on the 6811 microprocessor, and software reads that port to use in various behaviors. Originally, the Sharp boxes produced digital output, but the following modifications allow for an analog output signal.

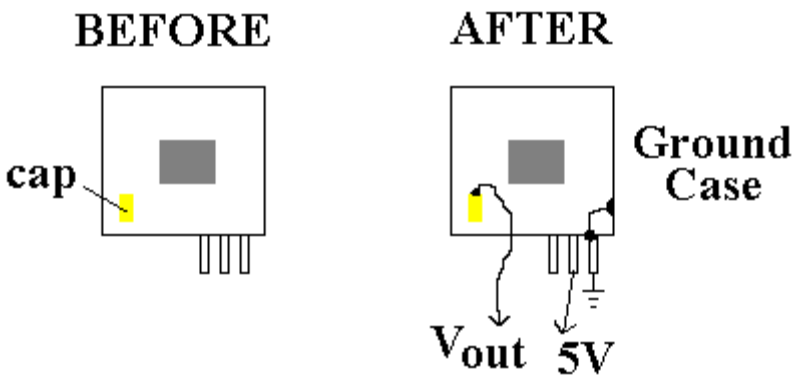

Figure 5: Modifications to Sharp GP1U58X IR Detector

To modify the Sharp sensor, open up the metal case and solder a wire to the capacitor shown in Figure 5. Then ground the metal case to the ground pin. The added wire is the analog output and is connected to an A/D port. The IR light is provided using the circuit in Figure 6.

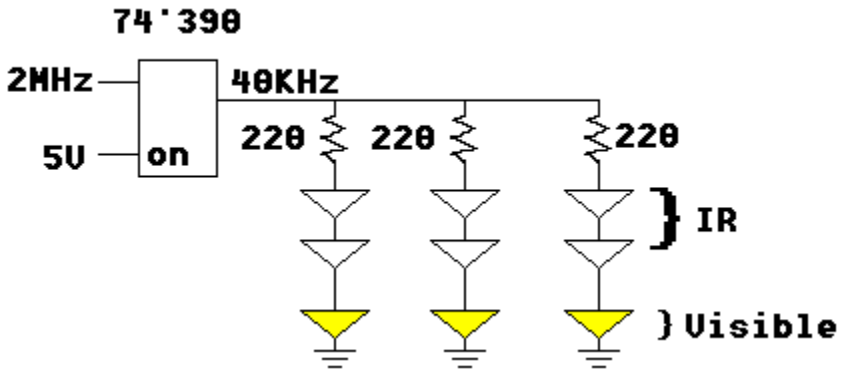

Figure 6: IR LED Biasing and Modulation Scheme

The 74'390 chip divides the 2 MHz E clock by 50 to produce a 40 KHz signal. This signal is always on since the chip select line is tied high. The resistors limit the current through the LEDs, and the visible LEDs verify that the IR LEDs are on. Figure 7 shows how an IR LED is connected to a Sharp box.

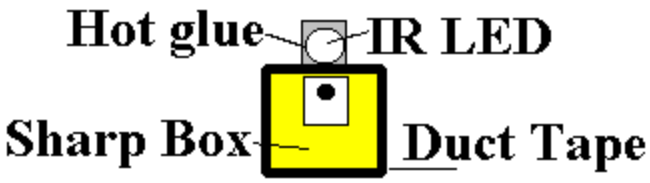

Figure 7: Sharp Box and IR LED Interconnections

Duct tape surrounds the Sharp box and provides insulation between the grounded box and the LED. The LED is mounted through hot glue onto the duct tape. The glue provides a stable, insulated, and simple mount for the LED. Figure 8 shows the ranges for the six emitter / detector pairs, and Figure 9 shows the sensor configuration on the robot.

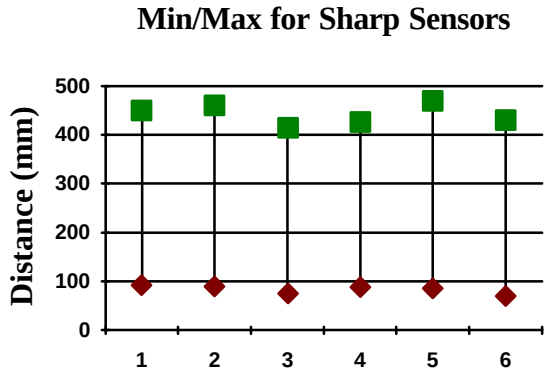

Figure 8: Range of IR Emitter/Detectors

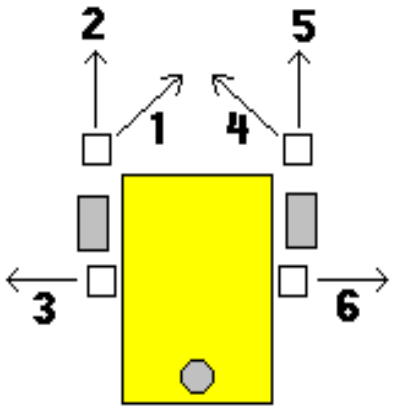

Figure 9: Sensors--Top View

The sensor arrangement shown in Figure 9 uses the 'cross-eyed' configuration first developed by Tay Choy in a previous semester of EEL 5934. Sensors 1 and 4 criss-cross in front of the robot and provide collision avoidance protection from objects not detected by sensors 2 and 5.

The range of voltage for the six sensors is 84-131 on a scale of 0-255 for the 5 V reference on the A/D converters. This corresponds to about a 0.9 V swing on the analog output of the Sharp sensors. The software used to read and interpret the incoming Sharp data is based on the class handout 'A (Bad) Programming Example For Obstacle Avoidance' by Bruce Norton. The algorithm compares the six Sharp sensor inputs to a pre-determined constant threshold. When a certain combination of sensors exceed the threshold, a recommended set of motor commands are stored in global variables. For example, if sensors 2 and 5 (the front-facing sensors) go high, then the behavior recommends to turn both wheels backwards. These recommendations are then used by an arbitrating behavior which decides whether to avoid collisions, travel in a straight line, or perform some other behavior--the arbitrator decides which motor commands to send to the servos.

Magnet and Turntable

The permanent magnet and turntable lift metal nails from the floor and store them on the robot away from the magnet. Figure 10 shows the magnet/turntable schematic.

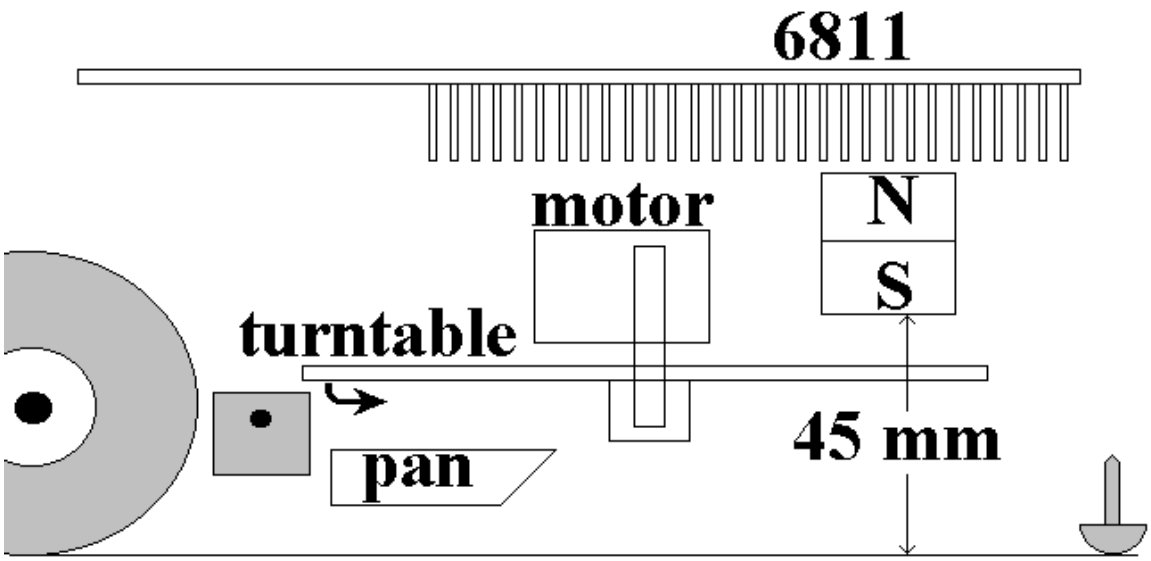

Figure 10: Magnet/Turntable Schematic--Side View

The permanent magnet attracts a nail from the ground and holds it against the spinning turntable. The Lego turntable pulls the nail away from the magnet and toward the pan on the robot. At a critical distance, gravity attracts the nail into the pan--the distance varies depending on the weight and magnetic properties of the nail. A permanent magnet is used instead of an electromagnet for the following reasons: an electromagnet consumes more power than a permanent magnet; permanent magnets are cheaper and more available than electromagnets; for the same size and weight, a permanent magnet produces a stronger magnet field than an electromagnet.

The turntable is powered by a 5 V dc motor which operates on 5% duty cycle pulses produced by a TLC555 Timer chip. Figure 11 shows the complete turntable motor circuit.

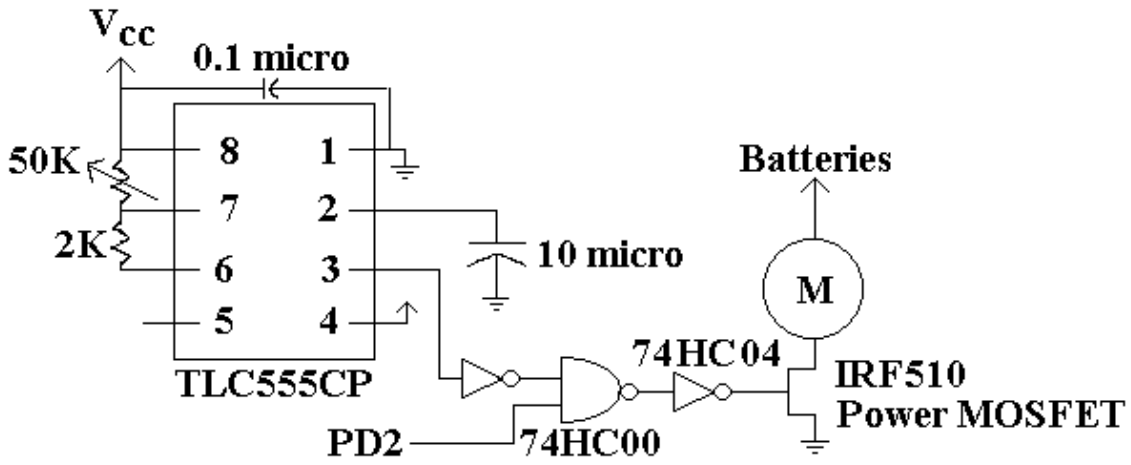

Figure 11: Turntable Motor Circuit

The two resistors and 10 micro F capacitor bias the 555 chip to produce a 95% duty cycle square wave at about 5 Hz. Since the 555 can only produce a duty cycle between 50% and 99.9%, a 95% cycle is produced and inverted before ANDing with the digital control line PD2 on the 6811. The AND signal controls the power MOSFET gate voltage. When PD2 is low, the gate voltage is low and the MOSFET does not switch on. When PD2 is high, the 555 pulses turn the MOSFET on and off so that the turntable spins at about 20 rpm.

Behaviors

A behavior is a routine performed by the robot based on internal or external data--it is designed to achieve a single task. An Autonomous Agent incorporates several behaviors to achieve its overall goal: to pick up nails and store them on board the robot. Collision avoidance is the most important behavior--it attempts to keep the robot from colliding with objects in its environment (code in Appendix). Another behavior is the straight line algorithm which directs the agent to move in a straight trajectory. These two behaviors work together to achieve optimum coverage of the robot's surroundings. However, some areas may not be traversed by the robot--this may occur if the agent 'gets stuck' in an oscillating pattern (in a corner, perhaps) or if the objects in the environment steer the robot in the same direction repetitively. Figure 12 shows two pathological scenarios for robot motion.

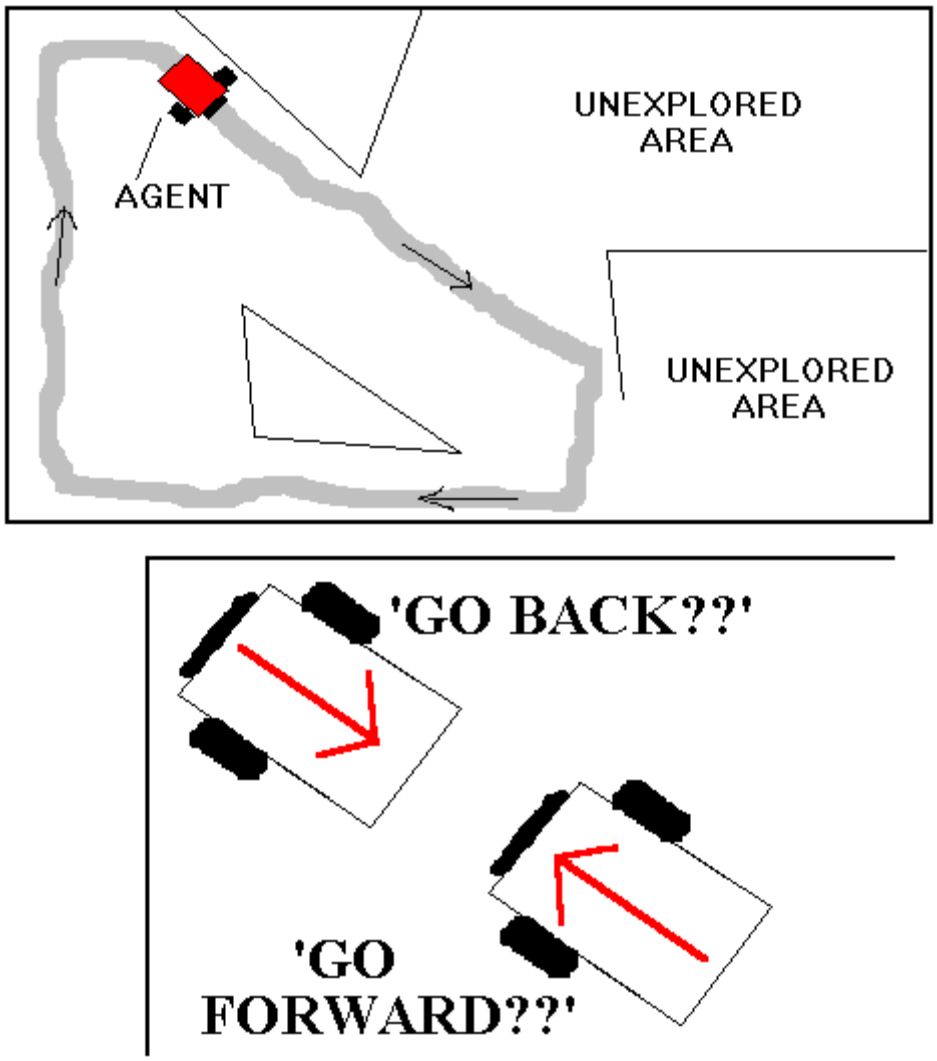

Figure 12: Pathologic Scenarios for Collision Avoidance--Top Views

The first diagram in Figure 12 shows the agent traversing a circular path. The agent is 'steered' by the objects in the room to only traverse this path--this leads to some areas of the environment going unexplored. The second diagram shows how a robot can get stuck in a corner. When the agent approaches the corner, it senses an object in front of it and moves backwards. At a certain distance, it no longer detects an object in front of it and proceeds forward again. This pattern can repeat until the robot works itself out of the corner--which may take several minutes! To solve these problems, An Autonomous Agent incorporates a behavior called go_berserk(). This behavior causes the robot to spin

until the front of the robot is clear of objects. The behavior is triggered by two sources: 1) an internal clock to trigger the behavior every 30 seconds, 2) internal global counters that track how often the different motor recommendations are implemented and trigger the behavior when the counters exceed a preset limit. Another problem is the nature of changing speeds for each servo--abrupt changes from forward to reverse cause jerky motions that are detrimental to the robot's hardware connections. To solve this problem, a behavior is used that steps the motor commands sent to the servos in 3% increments up or down. Finally, the arbitrating behavior decides which commands to send to the stepper function--it bases its decisions on internal global counters that track how often the different motor recommendations are implemented. Figure 13 shows a complete diagram of the control structure of the code for An Autonomous Agent.

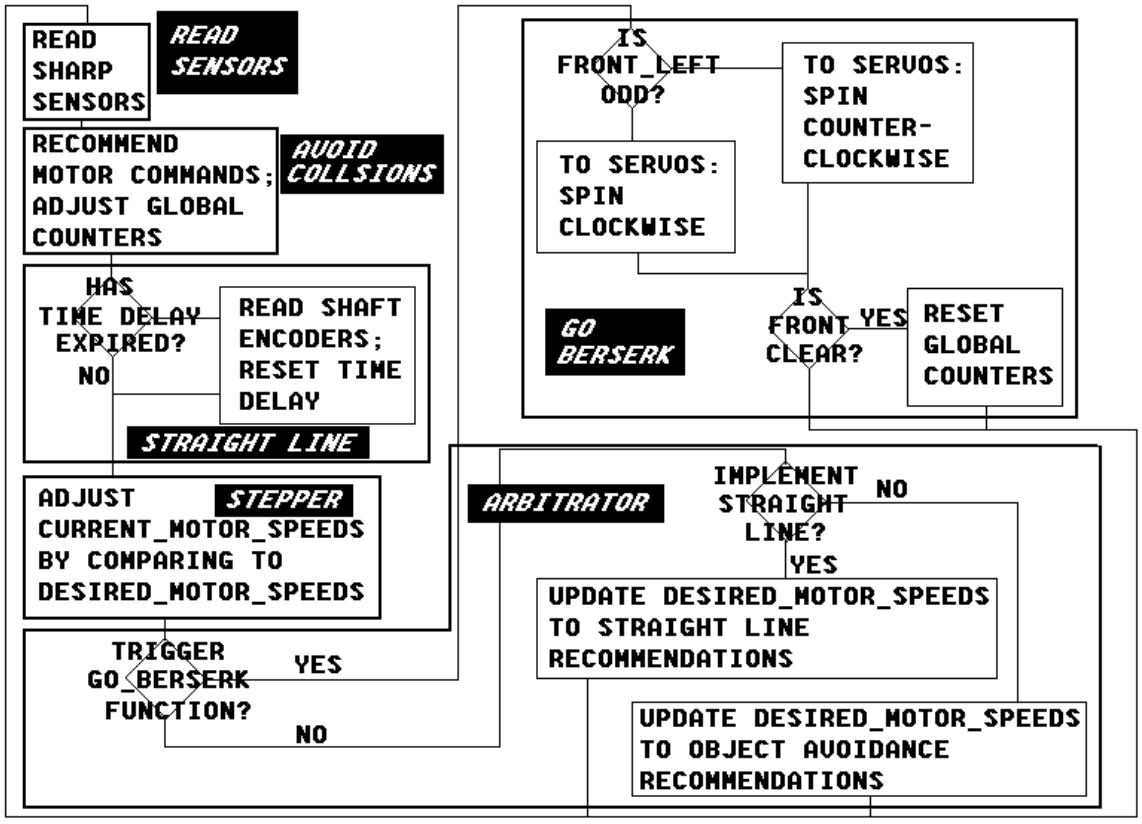

Figure 13: Control Flow Diagram for An Autonomous Agent

The thick lines enclose separate behaviors with each behavior named by a black box with white lettering. After the arbitrator is finished, control flow returns to the top of the diagram at the *READ SENSORS* behavior. Note: missing from this diagram is the control block that switches the turntable motor on and off--the motor is turned on for a preset time delay, then turned off for the same delay. This process repeats and allows for a more sophisticated design for determining when to switch on the turntable motor (i.e.: detect when a nail is on the turntable). The main advantage of using an arbitrator is that it eliminates conflict between several behaviors trying to drive the servos. The behaviors only produce motor command recommendations--stored in global variables--which are transferred to desired_motor_speeds variables. The desired speeds are the final values for which the current_motor_speeds adjust to in the stepper function.

CONCLUSIONS

The goal to build an autonomous agent is met with varying degrees of success. The collision avoidance algorithm based on Bruce Norton's example has produced reasonable results. The E-Fair provided a unique testing environment where several discoveries were made concerning collision avoidance and other subsystems of the agent. The circular patterns depicted in Figure 12 became evident after running the robot for at least 30 minutes, and the fact that the robot preferred to turn to the right prompted a redesign of the IR LED circuit from parallel to serial connections depicted in Figure 6--one pair of LEDs would hog current from other LEDs connected in parallel, thus distorting the areas of illumination around the agent. Without the stepper function to smooth out motor changes, the jerky motion of turning eventually takes its toll. An Autonomous Agent broke several solder joints after 80 minutes of constant collision avoidance. The E-Fair also provided a comparison between other robot designs from EEL 5934 and An Autonomous Agent. The superior performance of other robots in collision avoidance

prompted the redesign of this agent's algorithm as well as the addition of the berserking function. The need for collision avoidance and sensors is based on the realistic and practical application developed for this robot--to model an industrial-strength robot that removes nails from a construction site for safety purposes. The agent does not 'sense' the metal nail that it seeks to pick up--the permanent magnet always attracts whatever is under it. By its random motion alone, the agent covers virtually all areas in the test environment, thus removing all the nails. However, the intention of the turntable and pan is to collect the nails away from the magnet for easy disposal. The location of the turntable--under the belly of the agent--is to protect the mechanism from unavoidable collisions. The strength of the permanent magnet limits the clearance of the necessary hardware, and the arrangement of the wheels limits the size of the turntable. With the current design, the magnet extends over the edge of the Lego turntable enough to allow some nails to find their way around the turntable and against the side of the magnet. If enough nails find their way against the side of the magnet, they will jam the rotation of the turntable and have to be removed by hand. Better designs are achievable with more time and effort. This project has introduced me to several engineering realities: wire wraps and solder joints do not last forever; reliable diagnostic programs provide a benchmark for hardware performance as well as a hardware debugging tool; good ideas can fail because of factors like cost of implementation, availability of parts or board space or tools, the advent of a better idea; unforeseen changes in the environment can cause deviations in desired performance; a longer wire is easier to work with than a wire not long enough; ideas come from books, lectures, TAs, classmates, friends, family, total strangers, experiments, failure, conversations, and day dreaming.

APPENDICES

The following is the code used to operate An Autonomous Agent:

```
/*-------------------------------------------------------------
-----------*/
/*
*/
/*   OA14_MDS.C
*/
/*   Object Avoidance Code for An Autonomous Agent, Version 14.
*/
/*   Mark Skowronski, April 25, 1995
*/
/*
*/
/*   Based on oa13_mds.c
*/
/*   This version is an attempt to fix some of the subtle
problems         */
/*   associated with Version 13.  For one, the code has too many
processes */
/*   which cause a lag in performance.  This code tries to
tighten up      */
/*   some redudant algorithms.  The motor commands in
go_berserk() also    */
/*   occasionally conflict with other processes, resulting in
undesirable  */
/*   berserking behaviors.  The speed limiters in
straight_line() are      */
/*   necessary because the changable speed--the right motor
speed--can     */
/*   increment past 100% on some switchings, causing the agent
to lock up. */
/*
*/
/*   Required code:
*/
/*      lib_rw10.c
*/
/*      encoders.c
*/
/*      encoders.icb
*/
/*      oa14_mds.c
*/
/*
*/
/*-------------------------------------------------------------
-----------*/

/* Global variables */
/* Six analog Sharp sensor variables: */
int left_side, left_front, left_back;
```

```
int right_side, right_front, right_back;

/* For speed_stepper function */
int norm_speed = 30; /* 30% of maximum speed. */
int current_left_motor_speed=norm_speed,
desired_left_motor_speed=norm_speed;
int current_right_motor_speed=norm_speed,
desired_right_motor_speed=norm_speed;
int speed_stepper_delay=15; /* in ms */

/* To initiate go_berserk function */
int its_time_to_go_berserk = 0; /* 0 = false, 1 = true. */
int berserking_left_speed, berserking_right_speed; /* Locking
motor speeds */
                                                    /* used in
go_berserk().*/
/* For collision avoidance */
int threshold=120;
int left_motor = 0;
int right_motor = 1;
int left_speed_oa=norm_speed, right_speed_oa=norm_speed;

/* For straight line */
int go_forward_counter=0, go_forward_counter_threshold=200;
int left_speed_sl=norm_speed, right_speed_sl=norm_speed;
int straight_line_delay=200; /* in ms, delay time before reading
shaft */
                             /* encoders. */

/* For turntable */
int turntable_driver_delay = 5000; /* in ms */
int turntable_loop_counter=1;

void init_function(){
  enable_encoder(left_motor);
  enable_encoder(right_motor);
}

void wait(int milli_seconds){
  long timer_a;
  timer_a = mseconds() + (long) milli_seconds;
  while(timer_a > mseconds()) {
    defer();
  }
}

/* Motor Command Functions */

void go_right(){
  left_speed_oa = norm_speed;
  right_speed_oa = -1*norm_speed;
  go_forward_counter = 0;
}
```

```
void go_left(){
  left_speed_oa = -1*norm_speed;
  right_speed_oa = norm_speed;
  go_forward_counter = 0;
}

void go_forward(){
  left_speed_oa = norm_speed;
  right_speed_oa = norm_speed;
  go_forward_counter++;
}

void go_back(){
  left_speed_oa = -norm_speed;
  right_speed_oa = norm_speed/2;
  go_forward_counter = 0;
}

void read_analog_sensors(){
  while(1){
    left_side = analog(4);
    left_front = analog(2);
    left_back = analog(3);
    right_side = analog(0);
    right_front = analog(5);
    right_back = analog(6);
    defer();
  }
}

/* Behaviors */

void avoid_obstacles(){
  while(1){
    if (left_front>threshold && left_side>threshold &&
       left_back>threshold && right_front>threshold &&
       right_side>threshold && right_back>threshold){
       /* If all the Sharp sensors are high... */
         its_time_to_go_berserk = 1;}
    else if (left_side>threshold && right_side>threshold &&
       left_front>threshold && right_front>threshold) go_back();
    else if (left_front>threshold && left_side>threshold ||
       left_front>threshold && left_back>threshold)  go_right();
    else if (right_front>threshold && right_side>threshold ||
       right_front>threshold && right_back>threshold)
go_left();
    else  go_forward();
    defer();
  }
}

void straight_line(){
  while(1){
```

```
  /* This algorithm sets the left wheel at norm_speed and
adjusts the speed*/
  /* of the right wheel in 2% increments.
*/
    reset_encoder(0);
    reset_encoder(1);
    wait(straight_line_delay);

    /* If left wheel is faster, recommend increase speed of
right wheel */
    if (read_encoder(0) > read_encoder(1))
right_speed_sl=right_speed_sl+2;
    else   right_speed_sl = right_speed_sl - 2;

    defer();
  }
}

void speed_stepper(){
/* This function changes the current_motor_speeds in 3%
increments if   */
/* they are not exactly on the desired_motor_speeds.  If desired
equals */
/* current, then nothing changes.
*/

  while(1){
    if (desired_left_motor_speed > current_left_motor_speed)
      current_left_motor_speed += 3;
    else if (desired_left_motor_speed <
current_left_motor_speed)
      current_left_motor_speed -= 3;
    if (desired_right_motor_speed > current_right_motor_speed)
      current_right_motor_speed += 3;
    else if (desired_right_motor_speed <
current_right_motor_speed)
      current_right_motor_speed -= 3;
    wait(speed_stepper_delay); /* in ms, built-in defer()
function. */
    defer();
  }
}

void bc_and_td(){
/* Berserking Clock and Turntable Driver Function:
*/
/* This function performs two functions:  drive the turntable on
and off*/
/* and set the berserking flag.  The berserking delay is 6 *
t_d_delay. */
  while(1) {
    for (turntable_loop_counter=1;turntable_loop_counter<=3;
         turntable_loop_counter++){
      wait (turntable_driver_delay);
```

```
      bit_set(0x1008, 0x04); /* Set PD2. */
      wait (turntable_driver_delay); /* Built-in defer()
function. */
      bit_clear(0x1008, 0x04); /* Clear PD2. */
    }
    its_time_to_go_berserk = 1;
    defer();
  }
}

void go_berserk() {
/* This behavior does three tasks:
/*      1) Get the robot pointed in potentially different
directions    */
/*         periodically
*/
/*      2) Get the robot out of a sticky situation
*/
/*      3) Spin rapidly if all the Sharp sensors go high.
*/
  while(1) {
    if(its_time_to_go_berserk == 1) {
      if ((left_front & 0x01) == 1) {   /* If left_front is
odd... */
        berserking_left_speed = -norm_speed; /* adds some
randomness to */
        berserking_right_speed = norm_speed;}/* this behavior.
*/
      else {
        berserking_left_speed = norm_speed;
        berserking_right_speed = -norm_speed;}
      /* Send motor commands to the servos immediately. */
      motor(left_motor,berserking_left_speed);
      wait(100); /* Delay to allow transients to die down. */
      motor(right_motor,berserking_right_speed);

      wait(1100); /* Wait at least 1.1 seconds. */
      while(left_front>threshold && right_front>threshold){
        defer(); /* Wait until the front of the robot is clear
*/
      }
      current_left_motor_speed = berserking_left_speed;
      current_right_motor_speed = berserking_right_speed;
      its_time_to_go_berserk = 0; /* Reset berserking flag. */
      go_forward_counter = 0;  /* Reset SL flag--OA after
berserking. */
    }
    defer();
  }
}

void arbitrate(){
  while(1) {
    if (its_time_to_go_berserk == 1) {
```

```
      its_time_to_go_berserk = 1;} /* Do not avoid objects or go
in    */
                                   /* a straight line until
berserking */
                                   /* function is finished */
    else if (go_forward_counter > go_forward_counter_threshold)
{
      /* Update desired motor speeds from straight line
recommendations */
      desired_left_motor_speed = left_speed_sl;
      desired_right_motor_speed = right_speed_sl;
      /* Send motor commands from speed_stepper() to the servos.
*/
      motor(left_motor,current_left_motor_speed);
      wait(100);
      motor(right_motor,current_right_motor_speed);}

    else {
      /* Update desired motor speeds from obj. avoidance
recommendations */
      desired_left_motor_speed = left_speed_oa;
      desired_right_motor_speed = right_speed_oa;
      /* Send motor commands from speed_stepper() to the servos.
*/
      motor(left_motor,current_left_motor_speed);
      wait(100);
      motor(right_motor,current_right_motor_speed);}

    defer();
  }
}

void main(){
  init_function();
  start_process(read_analog_sensors());
  start_process(avoid_obstacles());
  start_process(straight_line());
  start_process(speed_stepper());
  start_process(bc_and_td());
  start_process(arbitrate());
  start_process(go_berserk());
}
```