

Rockefeller



The Autonomous Coin Collecting Robot

**By Kelly C. Krempin
EEL 5666
7/31/98**

TABLE OF CONTENTS

Acknowledgments	3
Abstract	3
Executive Summary	4
Introduction	5
Integrated System	5
Mobile Platform	5
Actuation	6
Sensors	7
Behaviors	12
Conclusion	13
Experimental Layout and Results	14
Appendix	16

ACKNOWLEDGMENTS

Charles Fedderwitz, EE and CE student and fellow classmate.

Scott Ettinger, EE graduate student and friend.

Jenny Laine and Scott Jantz, Teaching Assistants for EEL 5666 Lab.

ABSTRACT

The intent of this project was to successfully engineer and construct an autonomous mobile robot that would exhibit sophisticated machine intelligence behaviors. I chose to build an autonomous coin collecting robot based on a small microcomputer. The microcomputer I used enabled me to implement electronic sensors, tasking behaviors, and manipulation of motors and servos. This project required the integration of various sub-disciplines in electrical and computer engineering, such as, microcomputer interfacing and programming, analog to digital electronics, computer aided engineering, and communications.

EXECUTIVE SUMMARY

During the summer months of 1998, I engineered and built an autonomous mobile robot for EEL 5666, Intelligent Machine Design Laboratory (IMDL). This robot was to exhibit characteristics that would involve both the fields of electrical and computer engineering. Various disciplines included microcomputer interfacing and programming, analog to digital electronics, computer aided engineering, control, and communications. The robot I designed, named Rockefeller, successfully accomplished its object of being an autonomous coin collecting robot. The early stages of the project were dedicated to accomplishing obstacle avoidance behaviors. Obstacle avoidance is an intricate part of becoming autonomous because it allows the robot to maneuver freely in its environment. Initially, obstacle avoidance was accomplished through two types of sensors. IR emitter were used to bounce IR signals off objects in the robots path which were in turn received by IR detectors. By processing the signals sent from these detectors, code would determine the appropriate action for the robot to take to perform obstacle avoidance. Hacked servos were used as motors to control the robot's wheels which allowed it to maneuver. These motors were also controlled by obstacle avoidance code. Bumper sensors were also implemented to aid in collision detection in the event IR detection failed. The next important stage in development, concerned the coin detection sensor circuit. A metal detector was hacked to achieve an appropriate signal that could be used to detect when the metal detector was over a coin. This signal was wired directly to an analog to digital pin on the microcomputer. Associated code was also integrated to sample this signal and perform appropriate actions to acknowledge coin detection. Calibration code was also added to help in debugging purposes and to inform me of what values the robot had ascertained for the coin and the ground. The next stage was designing the platform structurally to enable the robot to perform its tasks and behaviors. I designed the platform in AutoCAD and milled it out of plywood on a T-Tech machine. This platform also included the complex design of the arm and bucket apparatus. At this point the robot was completely mechanically integrated and could perform basic obstacle avoidance and coin calibration and detection. The last few weeks concentrated on intensive coding to perform coin collection and corner detection and resolution behaviors until a fully functional autonomous coin collecting robot was realized.

INTRODUCTION

The financial gain achieved from coin forging has long been the reward of men and women armed with metal detectors wandering across wide regions of land. Now, through the advancement of micro-controller technology, the laborous activity of wandering can be eliminated while still obtaining a financial reward. I designed Rockefeller, the autonomous coin collecting robot, to search, detect and collect coins that maybe scattered across a given area. Rockefeller stores the collected coins in its bucket for later retrieval by its owner. Whether for financial gain or historic appreciation, Rockefeller will collect coins of any denomination while its owner attends to other interests.

INTEGRATED SYSTEM

The processing power of the robot and all memory and system control will be achieved through the use of Motorola's 68HC11 EVBU complemented with Mekatronix's ME11 expansion board. I used the ME11's 32K of SRAM to store all the software needed to control the robots movements, calibrations, behaviors and calculations. The Motorola EVBU board contains the MC68HC11E9 processor. I used the ports of this processor to control actuators, emitters, detectors and LED's. The software, which I wrote in IC (see attached IC code), will intergrate all these signals and systems to produce an autonomous robot that can avoid obstacles, detect and collect coins and then store them for later retrieval.

MOBILE PLATFORM

I designed the platform in AutoCAD (see attached AutoCAD drawings). I then milled it out of plywood on a T-tech machine. I designed the body with two independent front wheels controlled by hacked servos, two free spinning castors in the rear, a chasis to enclose the 68HC11 EVBU, ME11 and metal detector circuit boards, a compartment to retain collected coins, another compartment for the spool and its associated motor, surface mounted controls for easy access, metal detector and bumper attachment slots and a rotating arm and bucket for coin collecting. The platform design allows the robot to traverse freely powered by its independent front wheels which also provides turning cababilities. Bumpers act as collision detection and IR emitters and detectors aid in obstacle avoidance. The metal detector is attached to the underside or "belly" of the

platform, this allows for adequate coin detection. The rotating arm is attached to the sides of the robot and can rotate 180°. The arm rotates from a resting position on the coin retaining compartment to the ground's surface to capture a detected coin.

AutoCAD allowed me to make exact measurements and be creative in my design, however, next time I would be wise to account for the offset attributed to the milling bit in order to avoid hours of wood sanding and adjustments.

ACTUATION

In order to successfully detect and collect coins, I needed to devise an intricate combination of motors and servos to cover a specific area and then manipulate a detected coin into the coin retainer. I used two hacked servos (see <http://www.mekatronix.com/index.html> for the hack instructions) to act as the motors to control the speed and direction of the two front wheels. A SN754410 motor driver chip along with a 74HC04D inverter chip are used to deliver the appropriate current and signal to achieve the desired speed and direction (see <http://www.mekatronix.com/index.html> for the circuit schematic). Port A pins 6 and 5 on the MC68HC11E9 microprocessor provides a pulse width modulation (PWM) through the output compare (OC) function that the motor driver chip uses to determine the amount of current to deliver to achieve the desired speed. Port D pins 5 and 4 provide the signals to the motor driver chip to determine the intended direction. Another hacked servo along with motor driver circuitry is used to control the speed and direction of the spool. I use the spool to open and close the “door” to slide a detected coin into the bucket on the end of the rotating arm. Port A pin 3 is used with PWM to control the speed and port D pin 3 for direction. I used the predefined subroutines *motor(int x, float y);* and *stop();* (see attached IC code) to control the motors which I labeled 0 for left, 1 for right and 2 for the spool. Direction and speed are determined by the floating point values in the range 100.0 to -100.0.

I used an “unhacked” servo for the arm rotation. I used port A pin 7 to provide the PWM to achieve the desired rotation angle from 0° to 180° and tapped off +9V from my power supply to deliver enough current to rotate the arm at a satisfactory rate. I used the predefined subroutines *servo(int x);* and *servo_stop();* to manipulate the servo's angle of rotation. I used the arm servo and hacked servos in combination to manipulate a detected

coin to a “known” position up against the arm’s bucket which then enabled me to capture the coin in the bucket with a push and slide maneuver (see attached IC code). This maneuver worked surprisingly well after some initial obstacles were overcome.

One problem I encountered had to do with being unable to manipulate motor speed after turning on and using the servo. Apparently the IC source code for the servo initializes port A pin 7 to initialize OC1 for all the pins on port A. This causes all the port A motor pins to be inaccessible to their corresponding IC code subroutines. To fix the problem I had to clear the TOC1 counter after each servo maneuver. I wrote the following IC subroutine to accomplish this task:

```
void servo_stop()  
{  
servo_off();  
poke(0x1016,0x00);  
poke(0x1017,0x00);  
}
```

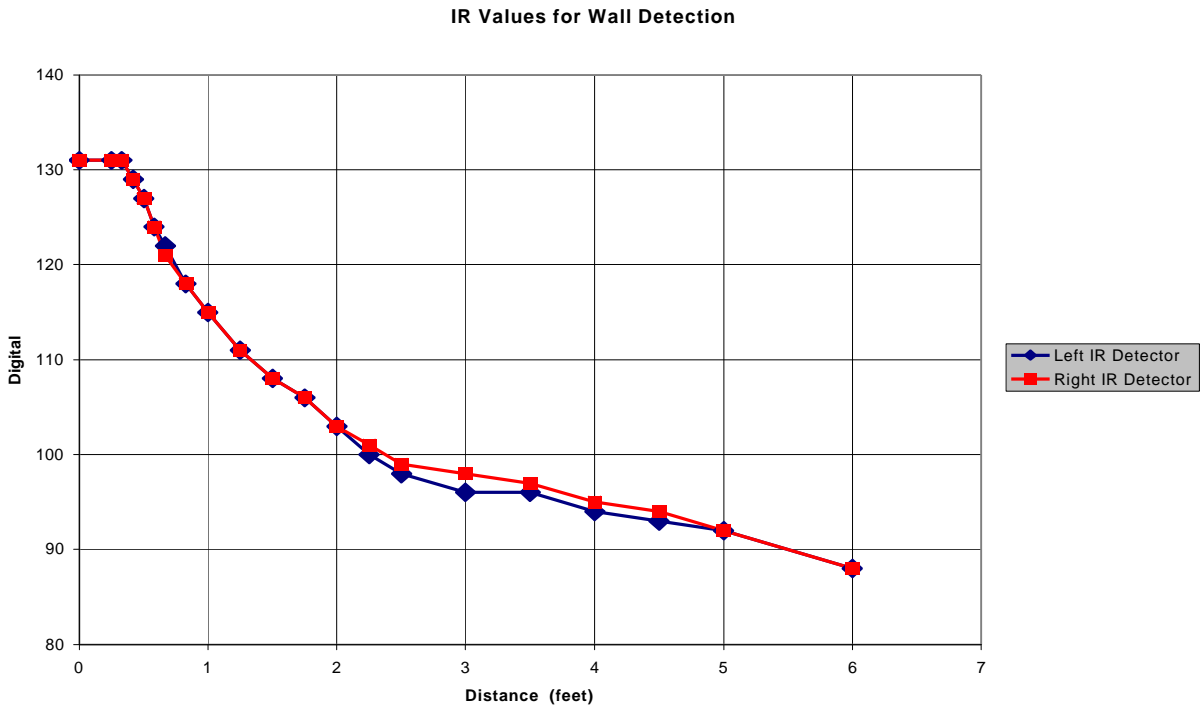
Another problem I encountered that was hindering my coin collecting maneuver was that when I tried to use my motors my servo would drop to the 0° angle until I stopped the motor in use, at which time the servo would return to its intended angle. The problem lied in the fact that I had created a “ground loop” instead of a “ground star” which was causing voltage fluctuation on the servo’s power circuitry. By running the ground from the 68HC11 EVBU board instead, I was able to obtain a constant voltage when running the motors. This fix could also be attributed to the voltage regulator contained on the board as well. Although these fixes were very complicated and time consuming to figure out, the results of everything working in combination correctly made the coin collecting maneuver quit simple and accurate.

SENSORS

I integrated 3 different and separate sensors (IR, bumper, and metal detection) for my robot. Each sensor plays an important part in achieving the goal of coin detection, collection and obstacle avoidance.

IR sensors allow for object detection, which in turn can be used to implement obstacle avoidance behavior. I mounted two infrared emitters on the top frontal portion of the

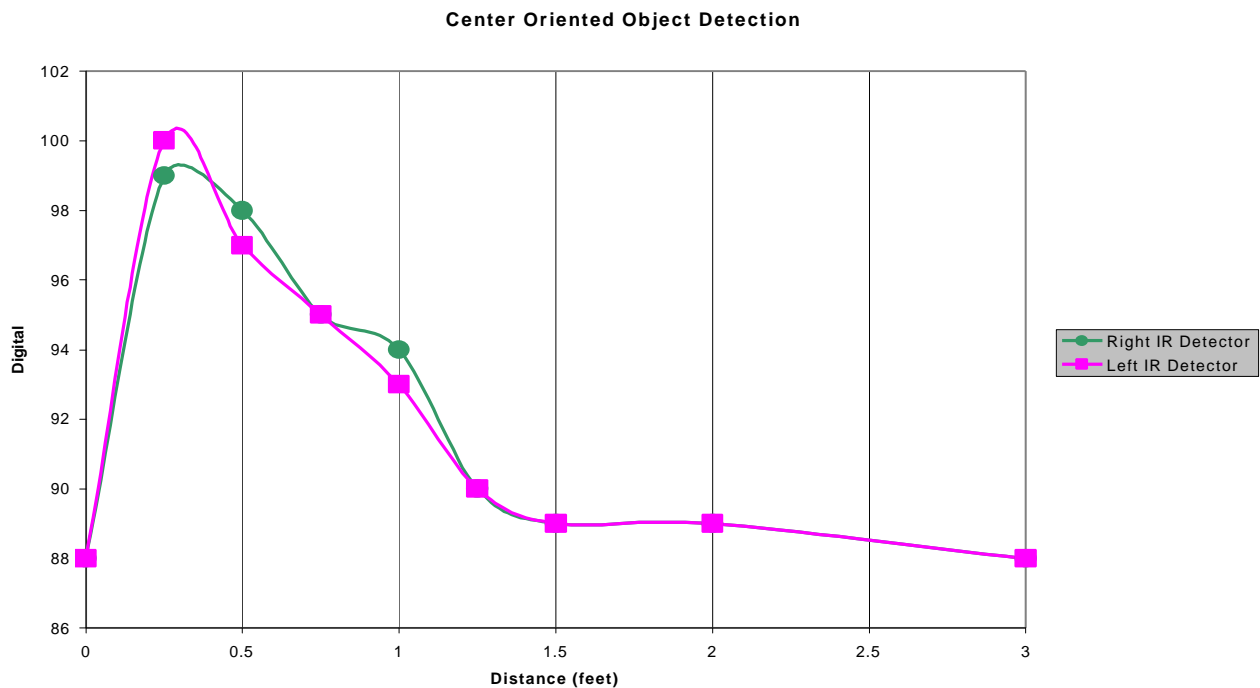
body. I mounted the two hacked infrared detectors (see <http://www.mekatronix.com/index.html> for the hack) on the bottom frontal portion of the body. I then ran experiments to see what type of response I could obtain from this IR configuration. The following graph depicts the IR detection values, when converted from analog to digital (0 to 255 = 0 to 5 V), as the robot approaches a wall.



As the two sensors approach the wall their output voltage, converted to digital, increases gradually. From an infinite distance to approximately two feet from the wall, the increase is almost linear, however, the voltage output rises dramatically, almost exponentially, as the robot approaches the wall from two feet. There is a slight difference in the two resulting voltages that the left and right IR sensors output within the range of 5 to 2 feet. The difference is acceptable as output can be expected to vary slightly. This experiment was conducted against a white wall. The color and texture of an object or wall can have a direct impact on the values obtained from the IR sensors. Since darker colors, especially black, absorb light at a higher rate, IR sensor will receive less IR reflection when presented with a dark object or wall. The sensor will not be as sensitive and the distance

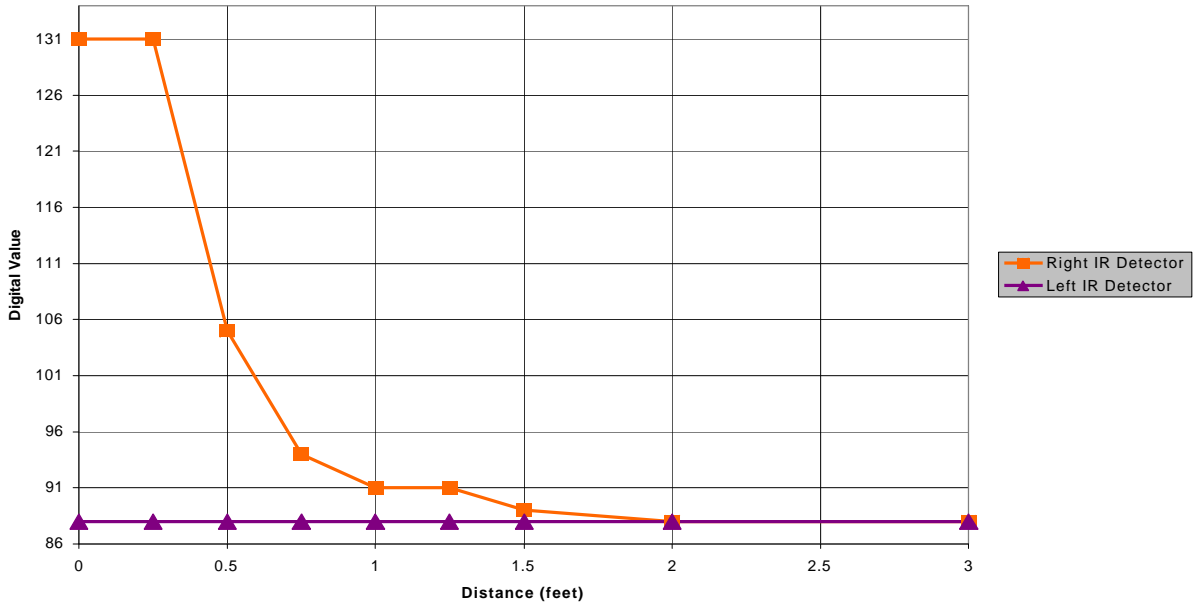
before detection values are obtained will be correspondingly less. If the robots' expected environment is to contain walls of a darker color, the IR sensor's decreased sensitivity should be accounted for.

The IR sensors are also able to detect free standing objects as well as surrounding walls. The following graphs depict the outcome of object detection experiments. I oriented a 2 x 6 inch object in three different positions (right, center and left) with respect to the front of the robot. A diagram of the layout can be seen in Figure 1 in Experimental Layout and Results. The graphs for the different object orientation positions, center, right and left follow respectively.



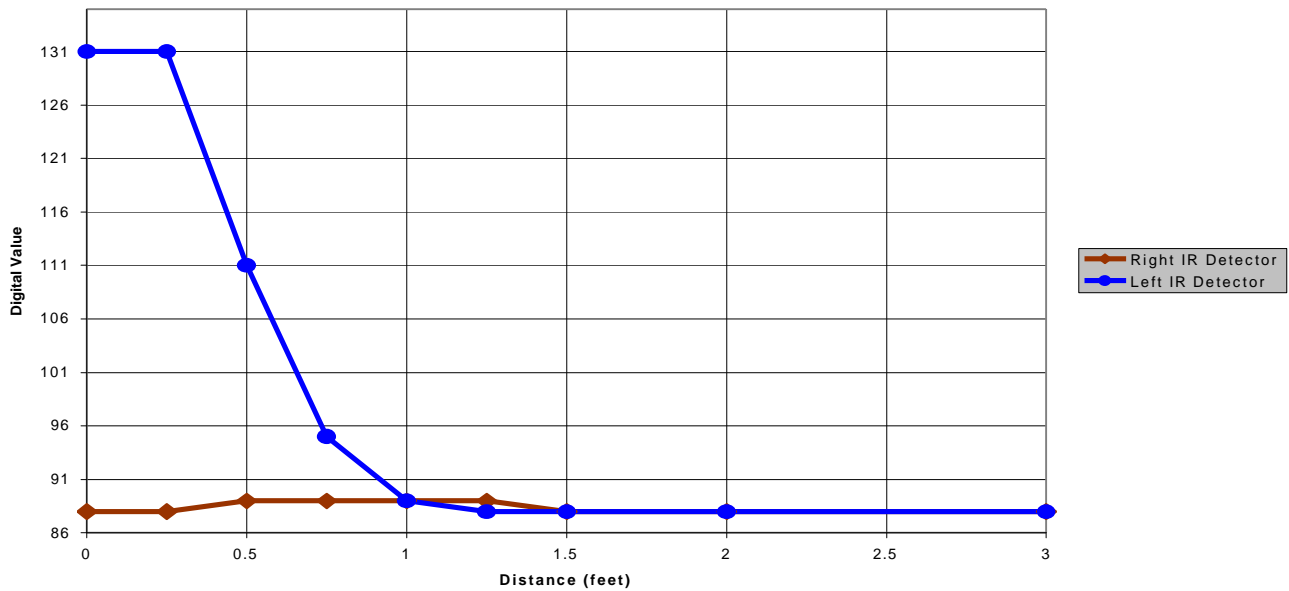
Notice that there is a dramatic increase in voltage output as the object is placed within 1.5 feet of the front of the robot. Then as the object rests right up against the front of the robot, there is no detection. When the object was positioned on the right side, the IR sensors provided very different readings.

Right Oriented Object Detection



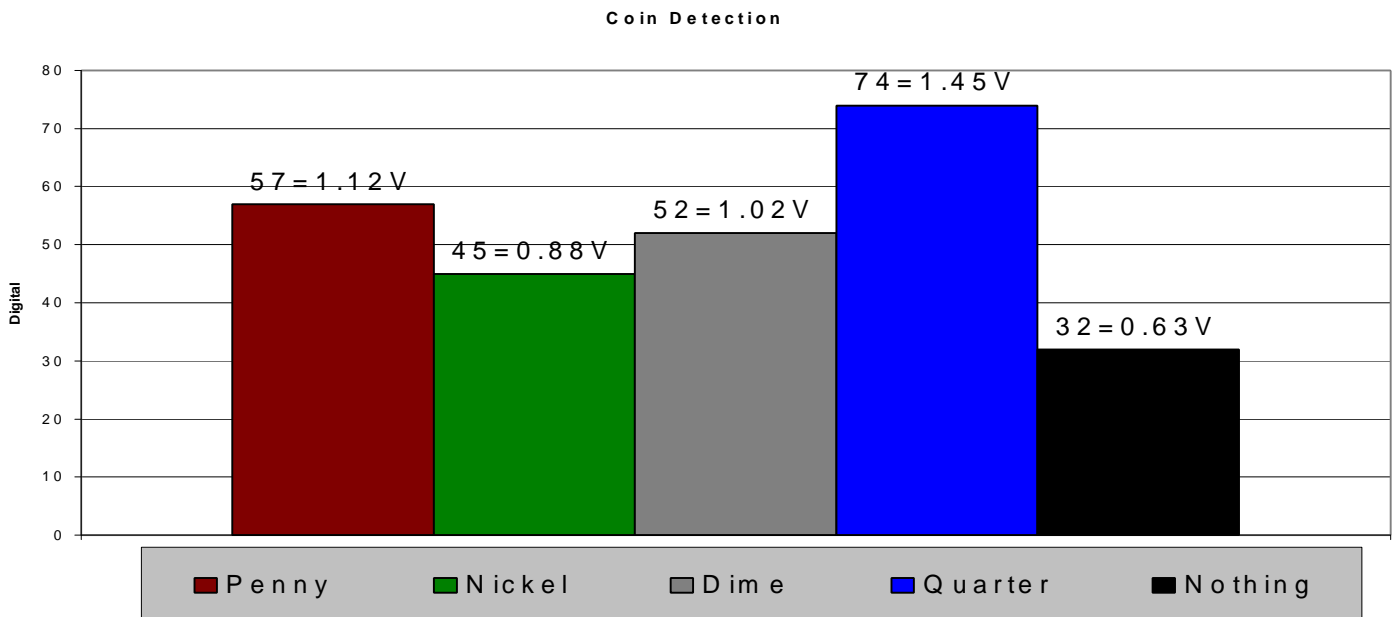
The left IR sensor never really detects any object in its path, however, at a distance of approximately 1 foot, the right IR sensor begins to dramatically output a reading to a maximum of 131 at approximately 0.25 feet. A correspondingly opposite result is found when the object is positioned on the left side. The opposite side IR, in this case, the right IR, does not detect anything, while the left IR begins dramatic detection at approximately the 1 foot mark.

Left Oriented Object Detection



Along with IR sensors, bump sensors provide for object detection and obstacle avoidance behavior. I implemented 3 bumpers in the front (right, center and left) of the robot. Figure 2 in Experimental Results and Layout depict the circuit design and corresponding voltage levels that allow the robot to determine which one or combination of bumpers have been depressed. This design has the benefit of only having one signal, as opposed to separate signal indicators for each bumper, that can be connected to an analog to digital conversion pin.

I also used a metal detector coil to detect coins that maybe lying underneath the robot. The metal detector was a Radio Shack Ferrous/Non-Ferrous metal detector. I found an op-amp on the metal detector circuit board that would deliver a satisfactory range of



voltage levels. I wired the pin from this op-amp to the analog to digital conversion pin on the EVBU board. This provided me with a strong enough signal to reliably detect underlying coins. The preceding bar graph was the results of detecting different coin denominations underneath the metal detector coil. Different levels of output voltage are detected for each respective coin denomination. These levels can be converted to digital values and are shown atop each corresponding bar. It would be nice to determine which denomination of coin was detected by its “identifying” voltage level, however, these levels were obtained as the coin was placed directly under the center of the coil. The resulting voltage output signal will vary as the coin is moved to different locations from

the center. Further experimentation must be done if each coin is to receive its own identifying voltage level.

BEHAVIORS

In order to behave consistently and efficiently as an autonomous coin collecting robot, I had to program Rockefeller so it would exhibit certain characteristic behaviors.

Rockefeller is programmed to exhibit six distinctly different behaviors, all of which are included in the attached IC code in the appendix.

Rockefeller is autonomous, which means it can function and accomplish its task without any outside help, except of course, for battery replacement. In order to be autonomous, obstacle avoidance behaviors had to be programmed into Rockefeller. Without obstacle avoidance, Rockefeller would get stuck or destroy itself by hitting obstacles all the time. Another behavior needed in order to be autonomous, was corner detection and resolution. Since I programmed Rockefeller to turn away from detected obstacles, an equidistant spacing of obstacles on each side, such as a corner, would cause Rockefeller to turn from side to side indefinitely trying to avoid both obstacles. In order to rectify this situation, I programmed an algorithm to detect the characteristic pattern of left-right obstacle detection within a specified amount of time. If obstacles are detected on alternating sides in a short amount of time, Rockefeller will turn 180° and move in the opposite direction of the assumed corner.

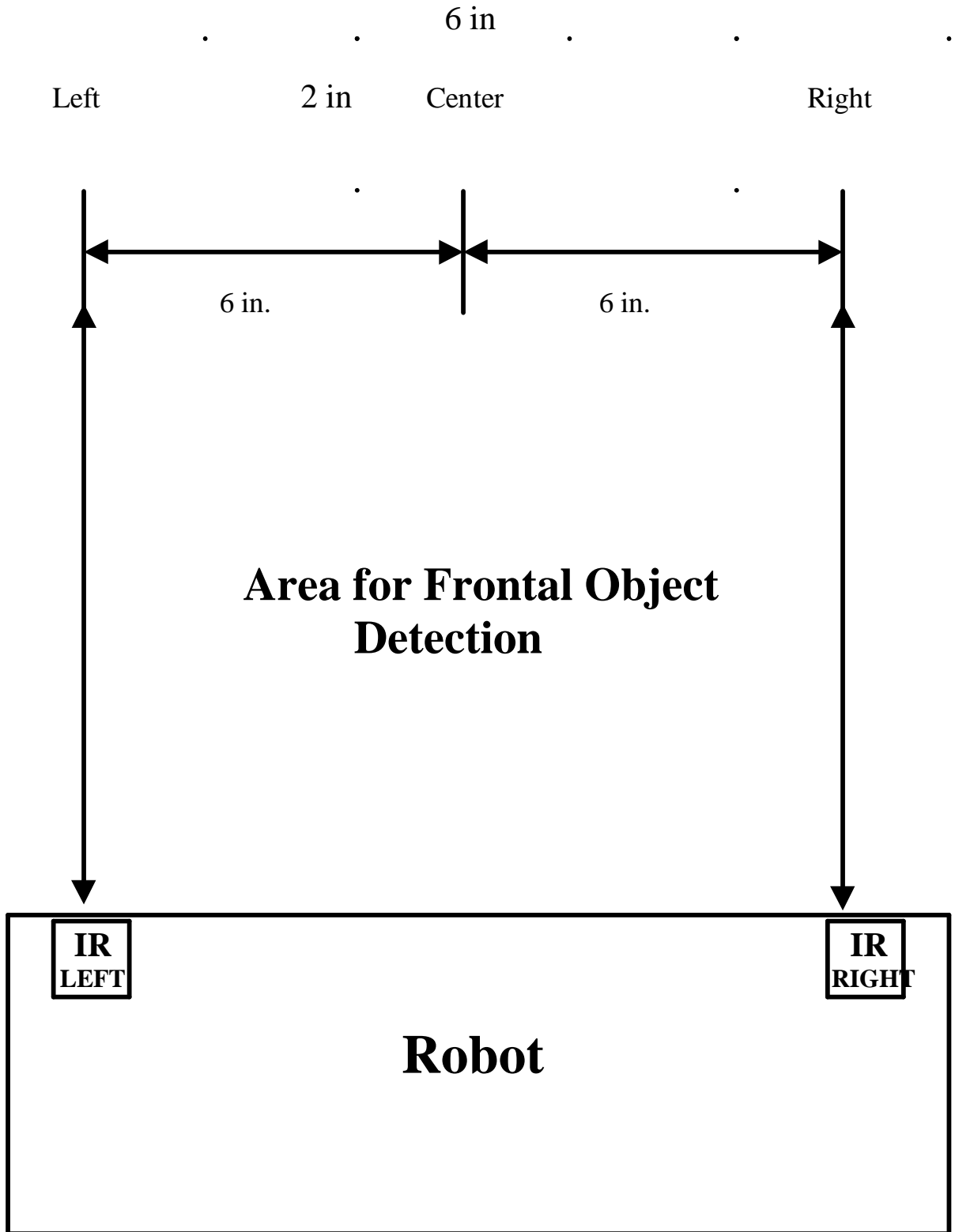
The other three behaviors have to do with coin detection and collection. In order to cover as much surface area as possible, I use a random algorithm to search for possible coins. I also calibrate the metal detector to detect a signal in a specific range when the robot is first turned on. This allows the user to specify a specific coin denomination that they want collected. I also programmed the robot to feedback to the user what values it has obtained for the base ground value with no coin and then what value it has calibrated for each specific coin to be detected and collected. This allowed me to debug any associated problems and informed me of what values Rockefeller would interpret as being the coin or coins in question.

All these behaviors, in combination, allowed Rockefeller to achieve its goal of being a successful autonomous coin collection robot.

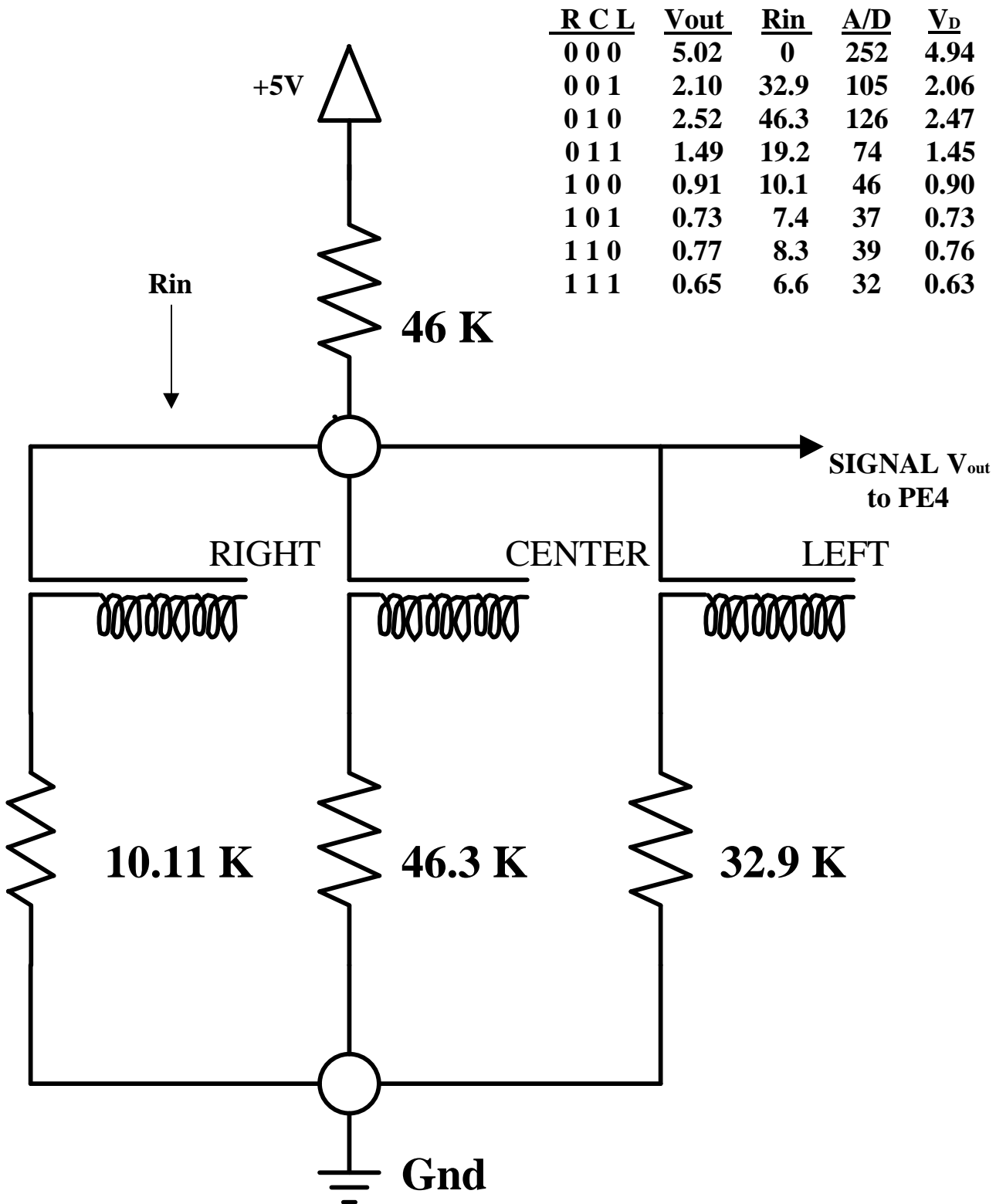
CONCLUSION

In conclusion, through an integrated system of sensors, behaviors and actuators a reliable and successful autonomous coin collecting robot named Rockefeller was made to alleviate the time consuming task of finding lost coins. The basic concept of specifying a particular object for an autonomous robot to seek out, detect and collect has a wide range of applications in the real world. I believe the ideas and concepts I have outlined in this report can be broadly applied to general forms of object manipulation by autonomous robots.

EXPERIMENTAL LAYOUT AND RESULTS
OBJECT ORIENTATION FOR MEASUREMENTS (FIGURE 1)



FRONT BUMPER SENSOR DETECTION CIRCUIT (FIGURE 2)



<u>R</u>	<u>C</u>	<u>L</u>	<u>Vout</u>	<u>Rin</u>	<u>A/D</u>	<u>V_D</u>
0	0	0	5.02	0	252	4.94
0	0	1	2.10	32.9	105	2.06
0	1	0	2.52	46.3	126	2.47
0	1	1	1.49	19.2	74	1.45
1	0	0	0.91	10.1	46	0.90
1	0	1	0.73	7.4	37	0.73
1	1	0	0.77	8.3	39	0.76
1	1	1	0.65	6.6	32	0.63

APPENDIX

IC CODE FOR ROCKEFELLER

```
/***** ROCK.LIS *****/
a:lib_rwl.a
a:servo.icb
a:servo.c
a:rock5.c

/***** ROCK5.C *****/

/*  ROCK
   Version : 5.0
   Date   : 7/26/98
   Programmer: Kelly C. Krempin
*/

int LEFT_IR, RIGHT_IR; /* values used for control */
int init_lir, init_rir; /* initial IR values used for calibration */
int lir, rir; /* realtime IR values used in calibration */
int front_bump, back_bump; /* analog to digital conversions of bumper */
/* circuits located in front and back */

int detect;
int left_corner_cnt, right_corner_cnt, dcnt, fdcnt, mod_i;
float b_time, e_time;
int i;

int LEFT_MOTOR=1;
int RIGHT_MOTOR=0;
int TURN = 450; /* for a "normal" turn */
int THRESHOLD = 3; /* the higher the # the closer it will get */
int NOTHING, QUARTER;

int rotation;
int ROTATION_PAUSE = 100;
int UNRAVEL_WAIT = 800;
int RAVEL = 500;
int SCOOT_BACK = 300;

/****** SubRoutines *****/
/******

/***** Turning IR emitters on and off *****/

void IR_on()
{
poke(0x7000, 0xc0);
}

void IR_off()
{
poke(0x7000, 0x00);
}
```

```

}

/***** Motor manipulations for turning *****/

void turn_left()
{
  motor(LEFT_MOTOR, -10.0);
  motor(RIGHT_MOTOR, 30.0);
}

void turn_right()
{
  motor(LEFT_MOTOR, 30.0);
  motor(RIGHT_MOTOR, -10.0);
}

void straight_ahead()
{
  motor(LEFT_MOTOR, 30.0);
  motor(RIGHT_MOTOR, 30.0);
}

void straight_ahead_easy()
{
  motor(LEFT_MOTOR, 30.0);
  motor(RIGHT_MOTOR, 30.0);
}

void go_backwards()
{
  motor(LEFT_MOTOR, -30.0);
  motor(RIGHT_MOTOR, -30.0);
}

void stopp()
{
  motor(LEFT_MOTOR, 0.0);
  motor(RIGHT_MOTOR, 0.0);
}

void servo_stop()
{
  servo_off();
  poke(0x1016,0x00);
  poke(0x1017,0x00);
}

/***** LED's *****/
void long_green()
{
  poke(0x7000,0x02);
  wait(1000);
  poke(0x7000,0x00);
  wait(500);
}

```

```

void long_red()
{
poke(0x7000,0x01);
wait(1000);
poke(0x7000,0x00);
wait(500);
}

void short_green()
{
poke(0x7000,0x02);
wait(300);
poke(0x7000,0x00);
wait(500);
}

void short_red()
{
poke(0x7000,0x01);
wait(300);
poke(0x7000,0x00);
wait(500);
}

void rg_blink()
{
for(i=0;i<5;i++)
{
poke(0x7000,0x03);
wait(200);
poke(0x7000,0x00);
wait(200);
}
}

/*****

/***** Delay *****/

void wait(int milli_seconds)
{
long timer_a;

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

void reset_corner_cnt()
{
while(1)
{

```

```

    b_time = seconds();
    e_time = seconds();
    while( e_time < ( b_time + 30.0 ) )
    {
        e_time = seconds();
    }
    left_corner_cnt = 0;
    right_corner_cnt = 0;
}
}

/***** IR calibration *****/

void init_read_IR()
{
    IR_on();
    wait(100);
    init_lir = analog(0);
    init_rir = analog(1);
    IR_off();
}

void init_read_detector()
{
    long_red();
    long_red();
    long_red();
    read_detector();
    wait(100);
    NOTHING = detect;
    short_green();
    short_green();
    for(i=0;i<5;i++)
    {
        read_detector();
        wait(500);
        if(detect > NOTHING)
        {
            NOTHING = detect;
            short_green();
            short_green();
        }
    }
    long_green();
    for( mod_i=(NOTHING/10); mod_i > 0; mod_i--)
    {
        short_red();
    }
    short_green();
    for( mod_i=(NOTHING%10); mod_i > 0; mod_i--)
    {
        short_red();
    }
    long_green();
    read_detector();
    wait(100);
}

```

```

while( detect <= (NOTHING+20) )
{
  read_detector();
  wait(500);
}
QUARTER = detect;
short_green();
short_green();
for( dcnt=1 ; dcnt<15 ; dcnt++)
{
  read_detector();
  wait(100);
  if( detect >= QUARTER )
  {
    QUARTER = detect;
    short_green();
    short_green();
  }
}
long_green();
for( mod_i=(QUARTER/10); mod_i > 0; mod_i--)
{
  short_red();
}
short_green();
for( mod_i=(QUARTER%10); mod_i > 0; mod_i--)
{
  short_red();
}
long_green();
}

```

/****** Reading sensors *****/

```

void read_IR()
{
  IR_on();
  wait(100);
  lir = analog(0);
  rir = analog(1);
  IR_off();
  LEFT_IR = lir - init_lir;
  RIGHT_IR = rir - init_rir;
}

```

```

void read_front_bmp()
{
  front_bump = analog(4);
}

```

```

void read_back_bmp()
{
  back_bump = analog(5);
}

```

```

void read_detector()
{
  detect = analog(3);
}

/***** Avoidance *****/

void ir_avoid()
{
  read_IR();
  if ( LEFT_IR >= THRESHOLD || RIGHT_IR >= THRESHOLD )
  {
    if( LEFT_IR > RIGHT_IR )
    {
      left_corner_cnt = left_corner_cnt + 1;
      turn_right();
      wait(TURN);
    }
    else if ( RIGHT_IR > LEFT_IR )
    {
      right_corner_cnt = right_corner_cnt + 1;
      turn_left();
      wait(TURN);
    }
    else
    {
      go_backwards();
      wait(1000);
    }
  }
  else straight_ahead();
}

void front_avoid()
{
  read_front_bmp();
  if( front_bump <= 136 )
  {
    stopp();
    wait(300);
    go_backwards();
    wait(1000);
    if( front_bump >= 95 && front_bump <= 115 )
    {
      turn_left();
      wait(TURN);
    }
    else if( front_bump >= 42 && front_bump <= 55 )
    {
      turn_right();
      wait(TURN);
    }
  }
  else
  {

```

```

        turn_left();
        wait(1300);
    }
}
else straight_ahead();
}

void rock()
{
long_red();
wait(4000);
servo(4500);
servo_on();
servo(4500);
wait(400);
servo_stop();
init_read_detector();
wait(300);
init_read_IR();
left_corner_cnt = 0;
right_corner_cnt = 0;
while(1)
{
if( left_corner_cnt >= 3 && right_corner_cnt >= 3 )
{
poke(0x7000,0x01);
stopp();
wait(300);
go_backwards();
wait(1500);
turn_left();
wait(3000);
poke(0x7000,0x00);
left_corner_cnt = 0;
right_corner_cnt =0;
}
else
{
read_detector();
if( (detect <= (QUARTER+10)) && (detect >= (QUARTER-10)) )
{
stopp();
rg_blink();
coin_collect();
}
ir_avoid();
front_avoid();
}
}
}

void coin_collect()
{
servo(4500);
servo_on();

```

```

rotation=4500;
servo(rotation);
wait(100);
motor(2,-10.0);
while(rotation>1400)
{
rotation=rotation-50;
servo(rotation);
wait(40);
}
wait(RAVEL);
stop();
servo(1100);
wait(1000);
servo_stop();

motor(0,27.0);
motor(1,27.0);
wait(1100);
motor(0,17.0);
motor(1,17.0);
wait(500);
stop();
wait(800);
motor(0,-10.0);
motor(1,-10.0);
wait(SCOOT_BACK);
stop();

wait(500);
servo(1100);
servo_on();
servo(1310);
wait(500);
motor(2,-10.0);
wait(1400);
stop();
servo(1310);
wait(1000);
servo(1100);
wait(500);
servo_stop();
wait(500);
motor(2,35.0);
wait(700);
servo(1100);
servo_on();
rotation=1100;
while(rotation<2200)
{
rotation=rotation+100;
servo(rotation);
wait(ROTATION_PAUSE);
}
wait(1650);
servo(3500);

```

```

wait(UNRAVEL_WAIT);
stop();
wait(800);
rotation=3500;
while(rotation<4500)
{
rotation=rotation+100;
servo(rotation);
wait(80);
}
wait(1000);
servo_stop();
}
/*****/

/***** MAIN *****/

void main()
{
start_process(rock());
start_process(reset_corner_cnt());
}
/*****/

```