Intelligent Machines Design Laboratory

EEL 5934 (Robotics)

Professor: Keith L. Doty

"Fetch"

Final Written Report

John M. Prida

12/4/95

Table of Contents:

Abstract

The robot “fetch” is a small, mobile robot equipped with a gripping claw and sensors that allow it to detect objects, pick them up, and deliver them to specific areas in its environment. The robot is based on a round wooden platform and is propelled on wheels driven by servo motors. The robot includes various infrared, contact, and touch sensors. The behaviors of the robot will allow it to avoid collisions, find small objects, manipulate the objects, and deliver the objects. The robot is controlled by a Motorola 68HC11E9 EVBU board with 32k RAM expansion. The controller is programmed with Integrated C.

Executive Summary

The Robotics course was taken simultaneously with the Electrical Engineering Design course. The minimum requirements of the Robotics course were accomplished, and an additional system was added to the robot fulfilling the requirements of the Design course. The requirements of the robotics course are to design an autonomous robot with at least three sensor groups, displaying three integrated behaviors. For the Electrical Engineering Design requirements a mechanical gripper and lift system was incorporated onto the robot. This complete system allows the robot wander about its environment and to interact with certain objects that it encounters.

The three sensor groups provide the robot with its only information about the environment that it inhabits. These sensor groups include contact and touch sensor on the lift and gripper and various IR sensors on the robot platform. The robot collects data from its sensors while the programmed behaviors translate the information into commands to its actuators. The robot’s behaviors include obstacle avoidance, object finding, and wall following. Based on certain environmental conditions the robot is able to logically

switch from one behavior to another. Each of the robot's individual components and behaviors will be discussed in detail.

Introduction

This report is a complete documentation of the robotics project including a description of the recently programmed behaviors. The report begins with an overview of the robot integrated system. It then continues with a description of the mechanical aspects of the robot; including the lift and gripper mechanisms their motor and servo actuators. The positioning, scope, and utility of the various sensors used on the robot are also described in detail. Finally, a description of the robot's behaviors is given, followed by the experimental procedures and conclusion.

Integrated System

The robot consists of two main subsystems: the platform and the gripper/lift assembly. The platform consists of the plywood base, the motors, and wheels. The "lift" and the "gripper" are essentially two separate units. The lift is mounted directly to the robot platform and achieves the vertical motion of the claw. The lift also holds a servo that performs the rotating motion. The grip assembly is then attached to this wrist servo and the lift assembly.

It is the gripper and lift that provide the basis for the robot's behaviors. With the gripper and its sensors the robot will be able to interact with certain objects in its environment. The gripper will allow the robot to pick up and manipulate the objects, and because the robot is mobile, it can also carry the objects from place to place. The collision avoidance behavior uses infrared emitters and detectors, and robot finds objects using infrared and contact sensors. The gripper assembly uses touch sensors and contact sensors to detect when the hand has grasped an object. IR sensors are also used to implement wall following and edge detecting behaviors.

Robot Mechanical Systems

Round, Wheeled Platform

The round platform was cut from model aircraft plywood using a coping saw. The platform measures 11'' in diameter and the plywood is ▯ '' thick. The wheels are taken from an RC car and are about 3▯'' diameter, giving the robot about 2'' ground clearance. Two front mounted castors (one is placed on either side of the lift) balance the robot. The servo motors are screwed on through the plywood to the bottom of the platform. The EVBU board is mounted on four, 2'' bolts set through the platform from the bottom. An additional circuit is mounted to the platform vertically behind the EVBU. The battery pack was eventually moved to the front of the robot and strapped to the left side of the lift. This placement helps concentrate the weight of the robot over the wheels and castors. The two figures below show the layout of the platform from top and side views.

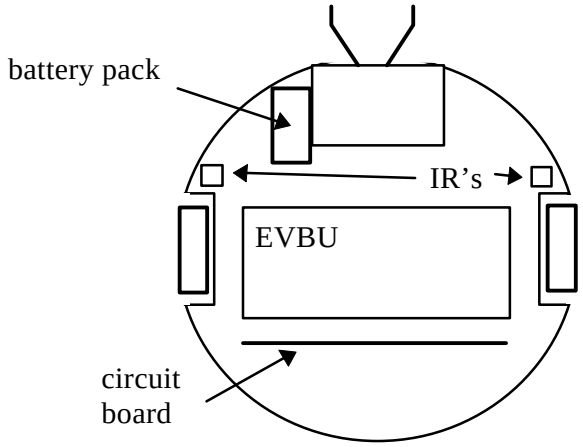

Figure 1. Platform (Top View)

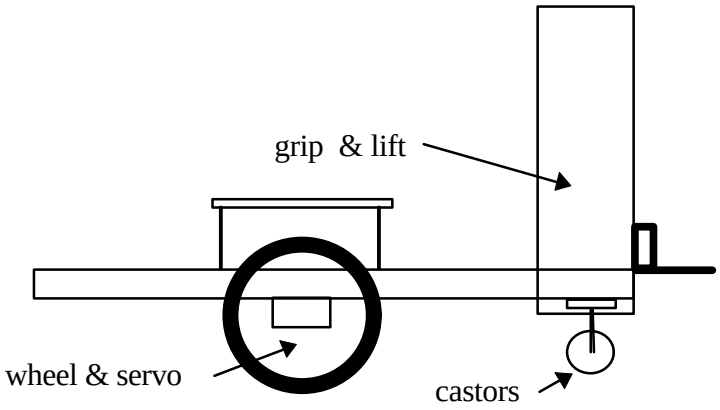

Figure 2. Platform (Side View)

Lift Structure

The lift assembly frame was built from ▯ in. model aircraft plywood and is held together with aluminum angle brackets and wood screws. The overall dimensions of the

lift are approximately 11 in. high (including motor and pulley), 3 in. wide, and 2 in. deep. The lift has two parallel rods that are mounted vertically on the lift frame and go through the lift platform. The larger of the two is a in. threaded rod that is use to raise and lower the lift. The threaded rod goes through the top of the lift frame where a pulley is attached to it. A second smaller rod is used as a guide and a means to stabilize the lift platform.

The lower limit of the lift is approximately 2 in. from the ground. The dimensions of the lift allow the gripper claw to be raised approximately 6 in. from this lower limit. These limits allow the gripper to be rotated at any position on the lift. The lift platform is a small piece of plywood onto which is mounted the wrist servo. This servo is used to rotate the gripper. A diagram of the lift is shown below.

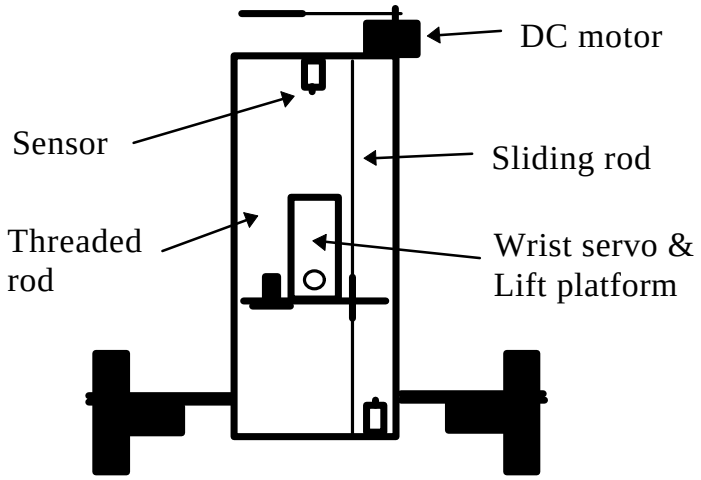

Figure 3. Lift Assembly

Gripper Assembly

The gripper fingers are linked together with two matching gears. The fingers are cut from plywood and are mounted to brackets that are attached to the gears. The brackets are mounted to the gripper platform and the gears mesh together so that the fingers open and close together. Attached to the left finger is a somewhat larger gear that meshes with a drive gear on the servo. The servo is also mounted to the gripper platform and is used to drive the grip open and closed. The angled ends of the gripper fingers overlap so that the grip can come fully closed on smaller objects. Various contact and touch sensors are mounted to the gripper fingers as well. A top view of the gripper assembly is shown below.

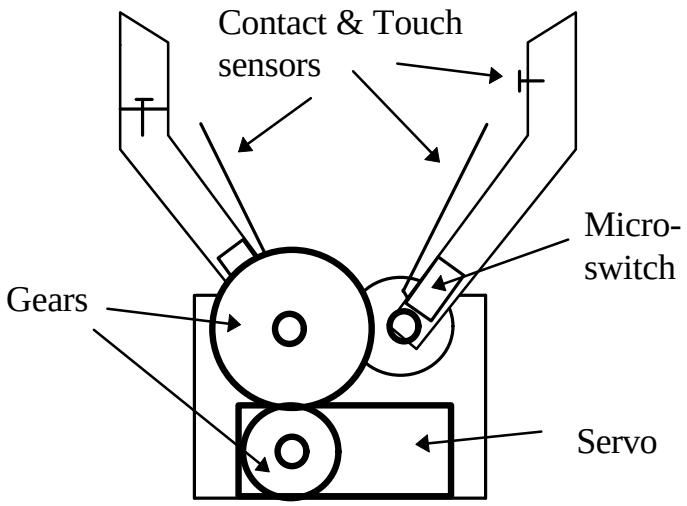

Figure 4. Gripper Assembly

Motor and Servo Actuation

Wheel Motors

The wheels of the robot are driven by two modified Futaba FP-148 servos. The servos are modified so that they rotate freely and act as gear head DC motors. The wheels are fitted to the six pronged heads that came with the servo and screwed into the servo drive gear. The servos spin at approximately 2 rev/sec at full speed, which translates into a maximum speed for the robot of about ft/sec.

Lift Motor

The lift is raised and lowered using a threaded rod linked to a small DC motor. A 1 in. pulley is attached to the threaded rod and a smaller in. pulley is attached to the motor. The two pulleys are linked with a simple rubber band. In order to control this lift motor an additional L293 motor driver was installed as well as a digital output port (74HC573 latch, address $6000). The lower four bits control the motor driver enable and direction pins.

Gripper Servos

Both the gripper and wrist utilize servos. The servos are unmodified Futaba FP-S28's, similar to the modified servos used for the wheel motors. Since IC provides control for only one servo, a simple AND gate circuit will be used to multiplex the servo signal line. This wrist servo requires the signal for only a short period of time to move

from one position to another. The select line will come from the new digital output port. A circuit diagram is shown below.

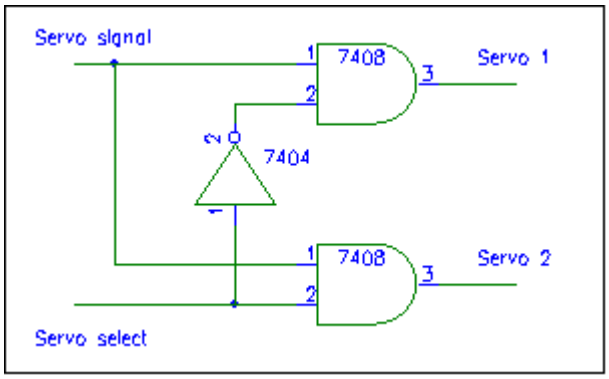

Figure 5. Servo select circuit diagram

A new PC board was mounted onto the robot behind the EVBU board in order to provide room for the additional circuitry. Because of the additional current demands and noise produced by the third motor and two servos, another 5V regulator was added to the new PC board. The lift motor will be supplied with the battery voltage and the servos with the +5V regulated power.

Several IC subroutines have been written to control the mechanical action of the grip and lift. Included in the appendix is a listing of these programs.

Sensor Suite

IR Sensor Layout

The majority of the robot's behavior is based upon data obtained from the its infrared sensors. Five Sharp infrared detectors and five infrared emitters are placed on the robot. Three of the five sensors are forward-looking. The two outside detectors face forward and slightly in, and their emitters are placed below them underneath the robot platform. A third forward-looking detector is placed just below the gripper on the bottom of the lift structure. The emitter for this sensor is slightly collimated in order to reduce the amount of reflection off the gripper, and also to concentrate the IR beam directly in front of the robot.

Another detector/emitter pair is placed on the top of the lift assembly and is faced forward and down at approximately a 45 degree angle. This sensor is used for obstacle avoidance, preventing the lift from striking taller obstacles that the lower sensors can not see. A fifth IR emitter and detector is placed on the right side of the robot facing to the side. This sensor is used for detecting the edge of a wall. The figure below shows the layout of the detectors and emitters.

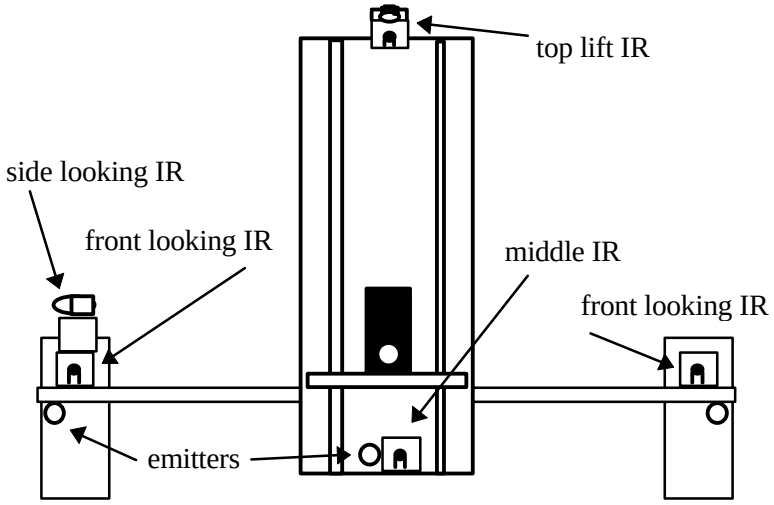

Figure 6. IR Sensor Layout

Contact and Touch Sensors

The lift has two limit switches made from micro-switches. One switch is mounted on the top of the lift frame and indicates when the lift is at its upper limit, the other is at the bottom of the frame indicating when the lift has reached its lower limit.

Each of the gripper fingers has a touch sensor made from smaller micro-switches. A feeler arm is attached to each micro-switch. These arms close the switches when the grip closes on an object. In addition to the touch sensors there are two small contacts on the ends of the fingers which sense when the grip has come fully closed without grabbing an object. A fourth contact sensor was added to the gripper in order to detect when an object is positioned in the gripper claw and ready to be grasped. The sensor was made from a micro-switch which was modified to be more sensitive (i.e. less force is required to

depress the switch). The switch is place on the front of the gripper just below the gripper fingers, and a flexible metal strip is placed in front of the switch.

Each of the contact sensors use pull up resistors to 5V, and are pulled down to ground when the switch is closed or a contact is made. These inputs are read through a new digital input port (74HC573 latch, address $7000). These sensors are indicated in Figures 3 and 4.

Programmed Behaviors

The robot has several sub-behaviors that when added produce the behavior of wall building. An overview of the robot's behaviors is as follows. The robot's initial behavior is that of obstacle avoidance and object finding. An obstacle is either a wall or some other object that is too large for the robot to pick up. The robot avoids collisions with obstacles. An object is defined as something that the robot is capable of picking up and wants to pick up. The robot uses the objects that it finds to build walls.

Once the robot thinks it has found an object it moves toward it to try and pick it up. If the robot successfully positions the object in the gripper it will grasp the object and lift it. Once the robot has an object it looks for a wall and begins to follow the wall. As the robot is following the wall it looks for an abrupt end of the wall or a sharp outside corner. At this point the robot stops, turns to the right, deposits the object at the corner, and resumes object finding behavior. The process is repeated each time the robot finds an object, and eventually a wall of objects is built at the edge.

A flow diagram of this process is shown below.

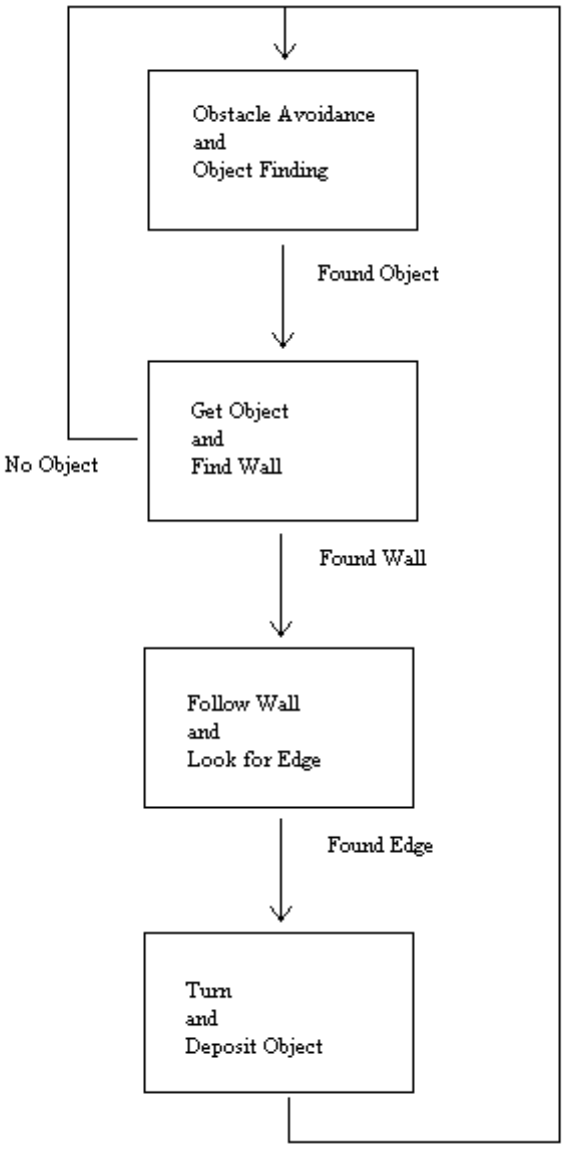

Figure 7. Behavior Flow Chart

Obstacle Avoidance

Obstacle avoidance is the robot's most basic behavior and is inherent in all of the robot's other behaviors. The robot performs obstacle avoidance with data primarily from the three forward looking IR sensors. When the robot senses an obstacle on its right or left side it veers either to the right or left accordingly. The closer the robot comes to an obstacle the quicker it will veer away. The robot also turns faster when obstacles are directly in front of it.

The sensor on the top of the lift is also used for obstacle avoidance. When this detector senses an object the robot knows it has encountered a tall object (undetected by the lower sensors) that is about to strike the lift assembly. In this case the robot backs up, turns 90 degrees, and continues.

Object Finding

While the robot is avoiding objects it is at the same time looking for objects. This behavior is also accomplished by interpreting data from the three front looking IR sensors. The robot thinks it has found an object when the middle sensor is above a certain threshold and the two outer sensors are below another threshold (the threshold values were determined experimentally). The figure below shows the region in front of the robot where an object will be detected.

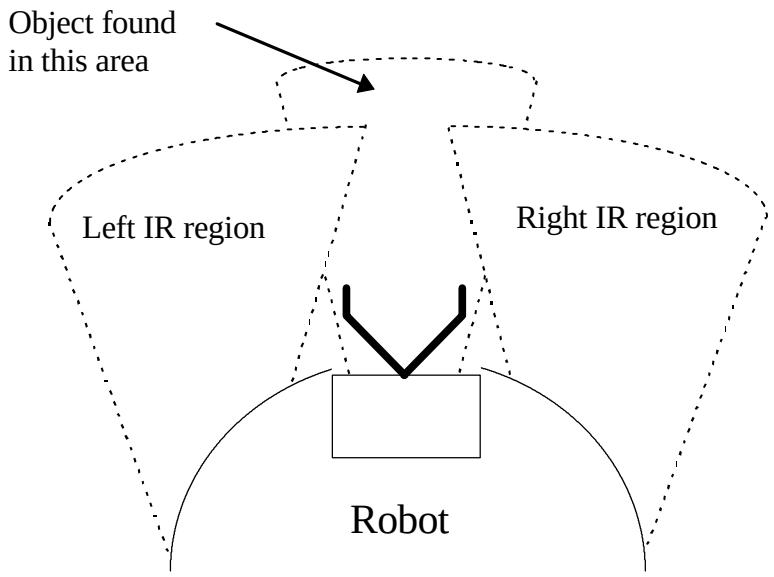

Figure 8. Object Detecting

When the robot thinks it has found an object it moves slowly forward trying to position the object inside its gripper (the robot makes slight adjustments based on the outer IR sensor readings). If the robot is not successful within a certain time-out period it backs up, turns, and continues looking for other objects. If the robot is successful it will grip and lift the object.

Wall Building

Once the object has been found, gripped, and lifted, the question of what to do with the object arises. The idea of a wall building behavior came from talking to Scott Jantz in the laboratory session. After the robot has found an object it begins to look for a wall to follow. The robot wanders until it finds a wall on its right side. When the right

and middle IR sensors are above a certain threshold and the left sensor is below another threshold the robot thinks it has found a wall. The robot proceeds to follow this wall until it finds an edge. The robot again uses it three front-looking IR sensors to implement wall following. If the right IR sensor is below a certain threshold the robot veers to the right, if it is above that threshold the robot veers left. If either the middle or left sensors detect an object the robot turns to the left sharply.

The robot detects an edge or corner with the right outward-looking sensor. If the sensor suddenly drops below a certain threshold the robot thinks it has found an edge and then turns and drops the object at this edge. Below is a picture of the robots environment:

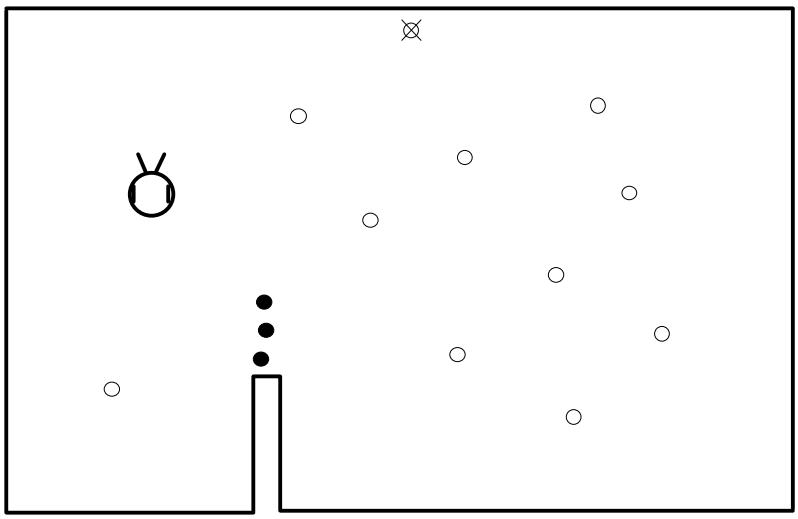

Figure 9. Robot Environment

The white dots are the objects the robot has not found, and the black dots are the objects that the robot has found and stacked along the detected edge. The X is an object that is too close to the boundary to be recognized by the robot as an object.

Experiments

Near the completion of the project, some experiments were performed to see how well the robot's behaviors worked in a controlled environment. An approximately 12' by 6' square, carpeted space was partitioned off as the robot's environmental boundary. A narrow wall was set protruding out from one of the boundaries. This wall was to serve as the wall that the robot will detect and 'build' on to (the experimental environment is similar to Figure 9). Six objects were placed randomly in the robot's environment. The 'objects' were cylindrical sections of 1½" PVC pipe, cut approximately four inches long. The sections were stood on end and a small piece of cardboard was glued to one end to help balance the objects.

The robot was set into the area and let run for approximately 20 -25 minutes . In that time period the robot managed to find all of the objects, but only successfully placed three of the object at the edge of the wall. The robot encountered several problems during the course of the experiments. If an object was too close to a wall the robot would either not detect the object, or when following the wall, mistake the object as an edge and set its object down next to the other object. The robot was also a bit clumsy, backing into objects because it has no rear-looking sensors. The robot also did not stack the objects strait, but tended to place them curving back toward the boundary. This prevented the robot from detecting the edge after only a few objects had been placed. In spite of these problems, I was pleased with the robot's obstacle avoidance, object finding, and wall following behaviors.

Other experiments performed throughout the building of the robot were the informal, trial and error variety. For instance most of the robots behaviors are based on

IR sensor input data, and this data is interpreted by comparing it to various thresholds. The exact value of these thresholds were determined by experimenting with different values as the robot interacted in its environment. Through trial and error the optimum values were determined.

Conclusion

Now, with the completion of the programmed behaviors, the robot has the ability to interact with its environment. Using only two types of sensors (IR and contact sensors) the robot is able to implement a variety of behaviors. These behaviors (collision avoidance, object finding, and wall following) combine to give an overall robot behavior of wall building. As of the writing of this report all of the behaviors were working well, but there are some minor adjustments that need to be corrected.

In its final state, several limitations of the robot system can be noted. The most notable of the limitations is the weakness of the gripper assembly. Solutions include devising a way to strengthen the grip so that it can hold heavier objects and developing more torque in the lift for raising heavier objects. The gripper has difficulty grasping certain shapes and cannot hold heavier objects. The link between the lift rotating servo and the gripper assembly is weak and tends to loosen as the robot performs its tasks.

The placement of the lift structure on the robot platform causes a problem in the robot's balance. The center of gravity of the robot is too far forward which causes the drive wheels to slip in some situations. Servo control is also a problem because only one

servo could be controlled at a time. This makes rotating the grip while holding an object difficult and also made the servos more susceptible to noise.

Fortunately, I will be able to continue work of the robot and gripper lift assembly in a follow up course. There are several enhancements that I would like to make in future that are particular to the gripper and lift assemblies. I would like to move the entire grip and lift assembly further back on the robot, thereby placing the weight over the drive wheels. I would also like to experiment with other gripper types, in particular a parallel arm gripper. These and other improvements will hopefully improve the overall performance of the robot.

APPENDIX

Program listing:

```
/*************************************************************/
/*                                                           */
/* Program:  Control program for autonomous robot            */
/* Behaviors: Collision avoidance, Object detection, Wall    */
/*            Following, and Wall Building                   */
/* Programmer: John M. Prida                                 */
/* Date:  12-4-95                                            */
/* Course: EEL 5934 Robotics - Intelligent Machines          */
/*         Design Laboratory                                 */
/* Instructor: Dr. Keith L. Doty                             */
/*                                                           */
/*************************************************************/
/*                                                           */
/* Global Variables                                          */
/*                                                           */
/*************************************************************/

/* sensor readings */

int ird_right,ird_left;
int ird_mid,ird_top;
int ird_wall,ird_wall_diff,ird_wall_last;

/* Thresholds */

int delt = -4;              /* delt = 6, good for lab */
int thold_right = 108-delt;
int thold_left = 104-delt;
int thold_mid = 102-delt-1;
int thold_top = 118;
int thold_wall = 100-delt;
int thold_edge = 100;

/* Motor Commands */

int turn_left = 10, turn_left_hard = 11, turn_left_slow = 12, vere_left = 13;
int turn_right = 20, turn_right_hard = 21, turn_right_slow = 22, vere_right = 23;
int forward = 30;
int backup = 40;
int halt = 50;
int inch_fwd = 60;
int go_where = forward;
int motor_min = -50;
int l_motor_max = 70;
int r_motor_max = 80;
int l_motor_speed = 0;
int r_motor_speed = 0;
int norm_dec = 3;
int hard_dec = 6;
int norm_inc = 18;
int norm_inc_var = norm_inc;
int delay1 = 0;
int delay2 = 1000;
int delay3 = 800;
int sound_flg = 0;
persistent int r_cnt;

int thold_obj_left = 101-delt+1;
int thold_obj_right = 104-delt+1;
int thold_obj_mid = 102-delt+0;

/* Behaviors */
```

```
int avoid = 0;
int get_obj = 10;
int pick_up = 20;
int deliver = 30;
int do_what = avoid;
int time_up = 0;
int inch_counts = 200;

/*************************************************************/
/* This module reads the IR sensor values                    */
/*************************************************************/


void sensor_input()
{
  while(1) {
              poke(0x7000,0b00000011);
       wait(100);                 /* IR stablization delay */
              ird_right = analog(0);
              ird_left = analog(3);
              poke(0x7000,0b00000100);
       wait(100);
              ird_mid = analog(2);
       poke(0x7000,0b00011000);
       wait(100);
              ird_top = analog(1);
       ird_wall = analog(4);
              }
}

/*************************************************************/
/* This module interprets the IR sensor data for collison */
/* avoidance and object detecting.                           */
/*************************************************************/

void sensor_interpret()
{
  while(1) {
              if ((ird_top>=thold_top)) {
         go_where = backup;    /* Object in front of lift */
                }
              else if((ird_mid >= thold_obj_mid) && (ird_right < thold_obj_right) && (ird_left < thold_obj_left)){
         do_what = get_obj;    /* Object found, go to get object */
                }
              else if ((ird_right>=thold_right) && (ird_mid < thold_mid)) {
                go_where = turn_left;
                }
              else if ((ird_left>=thold_left) && (ird_mid < thold_mid)) {
                go_where = turn_right;
                }
       else if (ird_mid>=thold_mid){
         if(ird_right > ird_left+4){
           go_where = turn_left_hard;
                }
         else{
           go_where = turn_right_hard;
                }
       }
              else {
         go_where = forward;   /* no obstacles ahead */
                }
  }
}

/*************************************************************/
/* This module sets the motor speeds and directions based */
/* the commands sent by sensor_interpret().                  */
/*************************************************************/

void motor_action()
```

```
{
  while(1) {

/* TURNS */

                if (go_where == turn_left) {
                  l_motor_speed -= norm_dec;
                  }
                else if (go_where == turn_left_hard) {
                  l_motor_speed -= hard_dec;
                  }
                else if (go_where == turn_right) {
                  r_motor_speed -= norm_dec;
                  }
                else if (go_where == turn_right_hard) {
                  r_motor_speed -= hard_dec;
                  }

/* FORWARD */

                else if (go_where == forward) {
                  if((l_motor_speed <= 40) && (r_motor_speed <= 40))
                    norm_inc_var = 1;
                  else
                    norm_inc_var = norm_inc;

                                if(l_motor_speed<l_motor_max)
                                  l_motor_speed += norm_inc_var;
                                else
                                  l_motor_speed = l_motor_max;
                                if(r_motor_speed<r_motor_max)
                                  r_motor_speed += norm_inc_var;
                                else
                                  r_motor_speed = r_motor_max;
                }

/* BACK UP */

                else if (go_where == backup) {
                  sound_flg = backup;
                  motor(1,-30);
                  motor(0,-30);
                  wait(delay2);
                  motor(0,50);
                  motor(1,-50);
                  wait(delay3);
                  sound_flg = 0;
                  }

/* STOP! */

                else if(go_where == halt){
                  l_motor_speed = 0;
                  r_motor_speed = 0;
                  }

/* INCH FORWARD */

                else if(go_where == inch_fwd){
        if((ird_top >= thold_top) || (ird_right > (thold_right+5)) || (ird_left > (thold_left+5)))
                                time_up = inch_counts;
                  time_up ++;
                  if(ird_right > (ird_left+3)){
          l_motor_speed = 30;
          r_motor_speed = 10;
                  }
                  else if((ird_left+3) > ird_right){
          l_motor_speed = 10;
          r_motor_speed = 30;
                  }
        else{
```

```
          l_motor_speed = 30;
          r_motor_speed = 30;
        }
      }

/* SET MOTORS */

   if(l_motor_speed < motor_min){
     l_motor_speed = motor_min;
     }
   if(r_motor_speed < motor_min){
     r_motor_speed = motor_min;
     }
   motor(0,l_motor_speed);
   motor(1,r_motor_speed);
   wait(delay1);
  }
}

/*************************************************************/
/* This module sets the motor speeds and directions for   */
/* wall following behavior.                                  */
/*************************************************************/

void motor_action_wall()
{
  while(1){
/* FORWARD */

      if (go_where == forward) {
               if((l_motor_speed <= 40) && (r_motor_speed <= 40))
                 norm_inc_var = 1;
               else
                 norm_inc_var = norm_inc;

        if(l_motor_speed<l_motor_max-30)
          l_motor_speed += norm_inc_var;
        else
          l_motor_speed = l_motor_max-30;
        if(r_motor_speed<r_motor_max-30)   /* lower max speeds */
          r_motor_speed += norm_inc_var;
        else
          r_motor_speed = r_motor_max-30;
              }

/* FOR WALL FOLLOW */

      else if(go_where == turn_left_slow){
        l_motor_speed = -30;
        r_motor_speed = 30;
      }
      else if (go_where == vere_left) {
        l_motor_speed -= (norm_dec-2);
        r_motor_speed = r_motor_max-30;
        if(l_motor_speed <= 20) l_motor_speed = 30;
      }
      else if (go_where == vere_right) {
        r_motor_speed -= (norm_dec-2);
        l_motor_speed = l_motor_max-30;
        if(r_motor_speed <= 20) r_motor_speed = 20;
      }
      else if(go_where == turn_right_slow){
        l_motor_speed = 30;
        r_motor_speed = -30;
      }

/* STOP! */

               else if(go_where == halt){
                 l_motor_speed = 0;
                 r_motor_speed = 0;
```

```
                }

/* BACK UP */

                else if (go_where == backup) {
                  sound_flg = backup;
                  motor(1,-30);
                  motor(0,-30);
                  wait(delay2);
                  motor(0,50);
                  motor(1,-50);
                  wait(delay3);
                  sound_flg = 0;
                }

/* SET MOTORS */

    if(l_motor_speed < motor_min){
      l_motor_speed = motor_min;
      }
    if(r_motor_speed < motor_min){
      r_motor_speed = motor_min;
      }
    motor(0,l_motor_speed);
    motor(1,r_motor_speed);
    wait(delay1);
  }
}

/*************************************************************/
/* This module produces sounds for diffent robot actions. */
/*************************************************************/

void produce_sounds()
{
  while(1){
                while(sound_flg == backup){
                  tone(600.0,.4);
                  tone(300.0,.4);
                }
                while(go_where == forward){
                  tone(1000.0,.05);
                  tone(950.0,.05);
                  tone(1050.0,.05);
                  wait(500);
                }
                while((go_where == inch_fwd) || (sound_flg = inch_fwd)){
                  tone(100.0,.9);
                  tone(115.0,.2);
                  wait(1000);
                }
  }
}

/*************************************************************/
/* This subroutine positions a found object into the      */
/* gripper.  If cant get object back to collision         */
/* avoidance, else pick up the object.                    */
/*************************************************************/

void object_get()
{

int object_in_grip;

  go_where = halt;
  wait(1000);
  time_up = 0;     /* try for time out period (inch counts) */
  object_in_grip = digport(5);

    while((object_in_grip == 1) && (time_up < inch_counts)){
```

```
                object_in_grip = digport(5);
                go_where = inch_fwd;
    }

  go_where = halt;
  wait(1000);

    if(object_in_grip == 0)     /* Object in grip = yes */
                do_what = pick_up;
    else{                       /* Missed object */
                go_where = backup;
                wait(1800);
                go_where = halt;
                do_what = avoid;
    }
}

void object_manip()
{
  sound_flg = inch_fwd;
  grip_close();
    if(got_object == 1){      /*pick up object & switch to delivery mode*/
       grip_close();
       wait(500);
                lift(1);
                sound_flg = 0;
                do_what = deliver;
    }
    else{                   /*no object so back to avoid*/
                go_where = backup;
                wait(1800);
                go_where = halt;
                sound_flg = 0;
                do_what = avoid;
    }
}

/*************************************************************/
/* This subroutine looks for a wall on the right side of  */
/* the robot, once the robot has got an object.          */
/*************************************************************/

void find_wall()
{
  while(!((ird_mid>=thold_mid) && (ird_right>=thold_right+2) && (ird_left<thold_left+5))){ /*loop untill there is a wall on th right ot
the robot*/
    if (ird_right>=thold_right) {
                go_where = turn_left;
    }
    else if (ird_left>=thold_left) {
                go_where = turn_right;
    }
    else if (ird_mid >= thold_mid){
                go_where = backup;
                wait(1800);
    }
    else{
                go_where = forward;
    }
  }
  go_where = halt;
  wait(1000);
}

/*************************************************************/
/* This subroutine follows a wall on the right side of    */
/* the robot and looks for the edge of the wall. When an  */
/* edge is found the robot turns and deposits the object  */
/* at the edge.                                          */
/*************************************************************/
```

```
void follow_wall()
{

  while(ird_wall > thold_edge){
       if((ird_mid>=(thold_mid+4)) || (ird_left>=(thold_left+3)) || (ird_right>=thold_right+8)){
         go_where = turn_left_slow;
       }
       else if(ird_right < (thold_right-2)){
         go_where = vere_right;
       }
       else if(ird_right > (thold_right+2)){
         go_where = vere_left;
       }
       else{
         go_where = forward;
       }
  }
  go_where = halt;
  wait(1000);
  while(ird_right <= (thold_right+5)){
       go_where = turn_right_slow;
  }
  go_where = halt;
  wait(1000);
  grip_rotate(0.0);
  lift(-1);
  grip_rotate(168.0);
  servo_deg(70.0);
  wait(800);
  go_where = backup;
  wait(1800);
  go_where = halt;
  grip_home();
  wait(1000);
}

/*****************************************************************/
/*                                                               */
/* Main Program:                                                 */
/* Starts Processes and switches behaviors.                      */
/*                                                               */
/*****************************************************************/

void main()
{

int pid1,pid2,pid3;

  wait(3000);
  beep();
  grip_home();
  r_cnt++;
  r_cnt = 1;  /* always run main on reset */
    if(r_cnt & 0b00000001){

      start_process(sensor_input());
      pid2 = start_process(motor_action());
      start_process(produce_sounds());

      while(1){
                if(do_what == avoid){       /*avoid obstacles & look fo objects*/
                          beep();
                          pid1 = start_process(sensor_interpret());

                          do_what = 5;
                }

                if(do_what == get_obj){     /*get the found object*/
                          kill_process(pid1);

                          object_get();
```

```
        }

        if(do_what == pick_up){     /*pick up the found object*/
                object_manip();
        }

        if(do_what == deliver){     /*deliver the object*/
      find_wall();
      kill_process(pid2);
      pid3 = start_process(motor_action_wall());
      follow_wall();
      kill_process(pid3);
      pid2 = start_process(motor_action());
      do_what = avoid;
        }


    }
   }
}


/*******************************************************/
/* John Prida's extended library functions        */
/*******************************************************/
/* GLOBALS */

float last_deg;
int got_object;
int buffer=0b00000000;
float over_grab = 25.0;


/*******************************************************/
/* Controls gripper, grips and holds object       */
/* Returns a 1 if it has an object, 0 if no object */
/*******************************************************/

void grip_close(){

int finger_l, finger_r, no_obj;

float i=0.0;
float open= 40.0;
float deg=open;

got_object=0;
bset(0b00010000);
servo_deg(open);
servo_on();
wait(400);
no_obj = digport(2);
finger_l = digport(3);
finger_r = digport(4);

while(no_obj == 1 && got_object == 0){

        deg = deg + 1.0;
        servo_deg(deg);
        no_obj = digport(2);
        finger_l = digport(3);
        finger_r = digport(4);

        if(finger_l == 0 && finger_r == 0){
          for(i=1.0; i<over_grab && no_obj==1; i=i+1.0)
            servo_deg(deg+i);
          if(no_obj == 1)
            got_object = 1;
          }
        }
```

```
if(got_object == 1){
  last_deg=deg+i;
  }
else{
  last_deg=open;
  servo_deg(open);
  wait(500);
  servo_off();
  }
}

/*************************************************************************/
/*  Brings grip to 'home position. grip open @ 0 degrees, lift bottom */
/*************************************************************************/

void grip_home()
{
servo_off();
bclear(0b00010000);
servo_deg(0.0);
servo_on();
wait(1000);
servo_off();
bset(0b00010000);
servo_deg(40.0);
servo_on();
wait(500);
servo_off();
lift(-1);
}

/*************************************************************************/
/*  Rotates grip while holding object.                                   */
/*************************************************************************/

void grip_rotate(float j)
{

  while(j<172.0){
       j=j+2.0;
                servo_off();
                bclear(0b00010000);
                servo_deg(j);
                servo_on();
       wait(80);
                servo_off();
                bset(0b00010000);
                servo_deg(last_deg);
                servo_on();
                wait(60);
  }
}

/*************************************************************/
/* Wait(n) subroutine, waits n=int milliseconds.          */
/* Resolution is 1ms.                                     */
/*************************************************************/

void wait(int milli_seconds)
{
  long timer_a;

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

/*************************************************************/
/* Raise or lower lift to limit, +n raise, -n lower       */
/*************************************************************/
```

```
void lift(int lift_dir){

int flag;

        if(lift_dir > 0){
          flag = digport(0);
                while(flag == 1){
                  motor_ex(2,1);
                  flag = digport(0);
                }
          off_ex(2);
        }
        if(lift_dir < 0){
          flag = digport(1);
                while(flag == 1){
                  motor_ex(2,-1);
                  flag = digport(1);
                }
          off_ex(2);
        }
}

/*************************************************************/
/* Reads value of digital input port, n=bit number        */
/* Returns a 1 if bit high (+5), 0 if bit low (gnd)       */
/*************************************************************/

int digport(int bit_num){

int mask, dp_out ;
int mask0 = 0b00000001;
int mask1 = 0b00000010;
int mask2 = 0b00000100;
int mask3 = 0b00001000;
int mask4 = 0b00010000;
int mask5 = 0b00100000;
int mask6 = 0b01000000;
int mask7 = 0b10000000;

        if(bit_num == 0)
          mask=mask0;
        if(bit_num == 1)
          mask=mask1;
        if(bit_num == 2)
          mask=mask2;
        if(bit_num == 3)
          mask=mask3;
        if(bit_num == 4)
          mask=mask4;
        if(bit_num == 5)
          mask=mask5;
        if(bit_num == 6)
          mask=mask6;
        if(bit_num == 7)
          mask=mask7;

        dp_out = peek(0x7000);
        dp_out = (dp_out & mask);
        if(dp_out > 0)
          dp_out = 1;
        return(dp_out);
}


/*************************************************************/
/* External Motor (motors 3 & 4) contol routines          */
/* On-Off & Fwd-Bkwd control only (+ fwd, - bkwd)         */
/*************************************************************/

void motor_ex(int mot_num,int mot_dir){
```

```
        if(mot_num == 2){
                if(mot_dir == 0)
                  bclear(0b00000010);
                else if(mot_dir > 0)
                  bset(0b00000011);
                else {
                  bclear(0b00000001);
                  bset(0b00000010);
                }
        }

        if(mot_num == 3){
                if(mot_dir == 0)
                  bclear(0b00001000);
                else if(mot_dir > 0)
                  bset(0b00001100);
                else {
                  bclear(0b00000100);
                  bset(0b00001000);
                }
        }
}


/*************************************************************/
/* Turns off external motors, same as motor_ex(n,0);     */
/*************************************************************/

void off_ex(int mot_num){

        if(mot_num == 2)
          bclear(0b00000010);
        if(mot_num == 3)
          bclear(0b00001000);
}


/*********************************************************************/
/*  sets or clears bit w/o affecting other bits, 6000 only.     */
/*********************************************************************/


void bset(int s_mask){

s_mask = s_mask | buffer;
poke(0x6000,s_mask);
buffer=s_mask;
}

void bclear(int c_mask){

c_mask = (c_mask ^ 0b11111111) & buffer;
poke(0x6000,c_mask);
buffer=c_mask;
}
```