JACO
Trash Retrieval Robot

University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machine Design Laboratory

Aaron Grassian
April 29, 1997
Instructor: Keith L. Doty

TABLE OF CONTENTS

ABSTRACT

The goal for this project is to create a robot that will generate the first step leading to the replacement of manual labor. In this specific case, the task intended to be eliminated is that of cleaning a floor space. Two basic things can accomplish this goal with an autonomous robot built within the limitations of this class: vacuuming and trash disposal.

ROBOT OPERATION and BEHAVIOR

Jaco's trash retrieval system consists of two components, a vacuum and a conveyer belt. These two modules work in conjunction with each other to perform the task of cleaning a set area. Jaco begins its operation in cleaning mode where it utilizes the vacuum and performs an increasing spiral pattern across the cleaning area. This ensures that the vacuum does not miss any areas.

Ideally, the surface to be cleaned would be lacking in obstacles and have a boundary of a perfect circle. Since this will never be the case, the robot was programmed to perform in a more realistic environment. During the spiral behavior, the robot constantly uses infra-red emitter-detector pairs and bumper switches to 'see' and 'feel' its environment. Depending on which of these sensors detects an obstacle governs what action the robot will take to move out of the way. After the robot has completely navigated around the obstacle, it continues in its spiral behavior.

An extra IR sensor determines whether the object blocking the robot's path is a piece of garbage or an obstacle. Whenever a piece of trash has been located, the robot needs to pick it up. The robot shuts down the vacuum, and turns on its conveyer belt system. The robot then spins on its axis for two seconds. During this spin routine, the piece of trash, which is made partially of Velcro, is picked up by the Velcro mate conveyer belt. The trash moves up the conveyer belt and is ripped off by a metal stopper. The garbage then falls into a trash can. The trash can is affixed to a lever which is connected through a rotor to a potentiometer. The micro-controller

monitors the position of this potentiometer whenever it has discovered a piece of trash and can determine when trash has entered the receptacle.

Once a piece of trash has fallen into the trash bucket, the convey er belt is turned off and the robot enters a patrol mode. This mode consists of the robot following the walls of the cleaning area. During this time, both the vacuum and conveyer belt are off. This mode is used to free the cleaning area of robots and allow human traffic. The patrol mode is a timed mode utilizing a counter within software. Whenever this counter has elapsed, the robot returns to cleaning mode. Figure 1 shows the behavior integration in flow chart form.

ADDITIONAL MOTORS

Two additional motors were added to the traditional Talrik configuration. This was accomplished by using the output port address $7000 as a control for relay s. The first step that had to be taken was to chang e the jumper that modulated the 40 kHz of the latch to ground, enabling the latch at all times. This returned the address $7000 to a mode which could be turned either to Vdd or ground with the poke statement in software. All of the IR LEDs (four) that were tied to this port and were being latched on at 40kHz were now sunk throug h the 40 kHz pin originally jumped to the latch. There are now four ex tra pins that could be used to control other modules.

This robot uses two of the remaining four pins to control the switching on and off of the vacuum and conveyer belt. A common way of controlling switching is using relays, but, the current coming from the HC11 was not strong enough to energize the coil. Therefore, transistors had to be used to amplify the current, see Figure 2.

TRASH RECEPTACLE SENSOR MECHANICS

The trash receptacle consists of a plastic cup cut so that objects coming off of the conveyer belt will fall inside. Attached to the cup is a long bar. A rotor assembly is mounted onto the base of the robot so that the rotor is horizontal to the platform of the robot. The middle of the bar is attached to this rotor allowing it to spin perpendicular to the base of the robot, see Figure 3.

On the opposite end of the rotor is a potentiometer. Any spinning of the bar translates into actuation of the potentiometer. In order for the bar to remain fixed with respect to the robot when the receptacle is empty, a counter force must be applied to the side opposite of where the basket is attached. This force is realized with a rubber band attached to the base of the robot. This downward directed force is equal and opposite to the force exhibited by the weight of the trash receptacle. Figure 1. Trash Receptacle Sensor Mechanical Schematic

The sensor is calibrated so that when the trash can is at its maximum capacity, the potentiometer is actuated to its fullest resistance (where the stop tab prevents it from being turned any further). The potentiometer is attached to an analog port of the micro-controller which, therefore, allows the nerve center of the robot to detect any change in the weight of the receptacle as well as when it is at full capacity.

TRASH RECEPTACLE CIRCUIT DESCRIPTION

The potentiometer has a full operating range of 144.47 Ω to 5.00 kΩ. The actuation of the bar from the highest to lowest trash receptacle point is 4.5 kΩ to 5.00 kΩ. The potentiometer resistance is linearly dependent on turning of the knob. This, in turn, makes it a linear function of the turning of the trash can bar.

The circuit for this sensor, see Figure 4, involves one resistor equal to the average of the high and low resistance of the operating range of the potentiometer. One lead of this resistor is wired directly to five volts. The other is connected to a potentiometer lead and the micro-controller analog port. The second lead of the potentiometer is connected to ground. This simple voltage divider circuit gives an output to the micro-controller of: $5*(R_{POT})/(R+ R_{POT})$. When the receptacle is full, the analog port will read 2.56 volts. Any other time, the reading will be somewhere in between 2.43 and 2.55 volts. Although this is a small voltage range, the robot is only looking for when a change occurs in this value and not specific values.

SOFTWARE CONSIDERATIONS

The software for Jaco, located in the appendix, controls and integrates several behaviors at one time. This is done through an arbitrater. All behaviors are set each time the program completes one tasking cycle. The arbitrater, however, is the module which sets how the robot will behave. This is accomplished through the setting and clearing of priority flags due to what the robot's environment is or what the software is processing. A clear example of the arbitrater at work is the differentiater between spiral mode and collision avoidance. Both of these behaviors work simultaneously. As the robot passes through each iteration of code, motor values are set for spiralling and collision avoidance to dummy variables. In an ordinary situation, the spiralling priority is set and the motors are set to the spiralling motor dummy variables. If the sensors detect an object within a certain threshhold, though, the collision avoidance flag is set and that behavior takes over within the arbitrater until the object is clear of the sensors. The spiral routine then takes over where it left off.

CONCLUSION

The knowledge and experience that I have gained in this class in the construction of this robot as well as the implementation of a controlling nerve center for an autonomous machine has been tremendous. The final demonstration of the robot showed that the orig inal goals of the project were met, but with some difficulties. Re lays worked correctly for a ctuation of both motors, but a wiring problem prevented the vacuum from being powered. The trash made for the conveyer belt was too lig ht and ended up being smacked away by the robot instead of being collected.

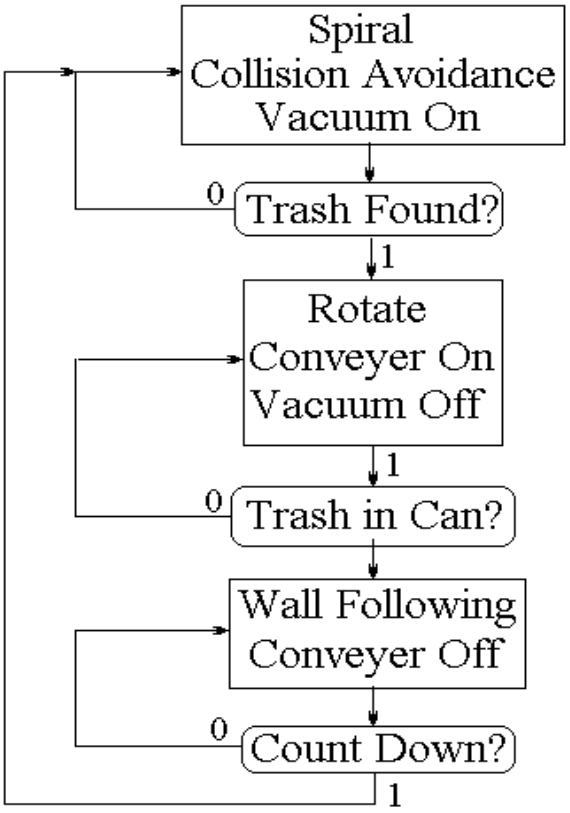

Figure 1. Behavior Integration Flow Chart

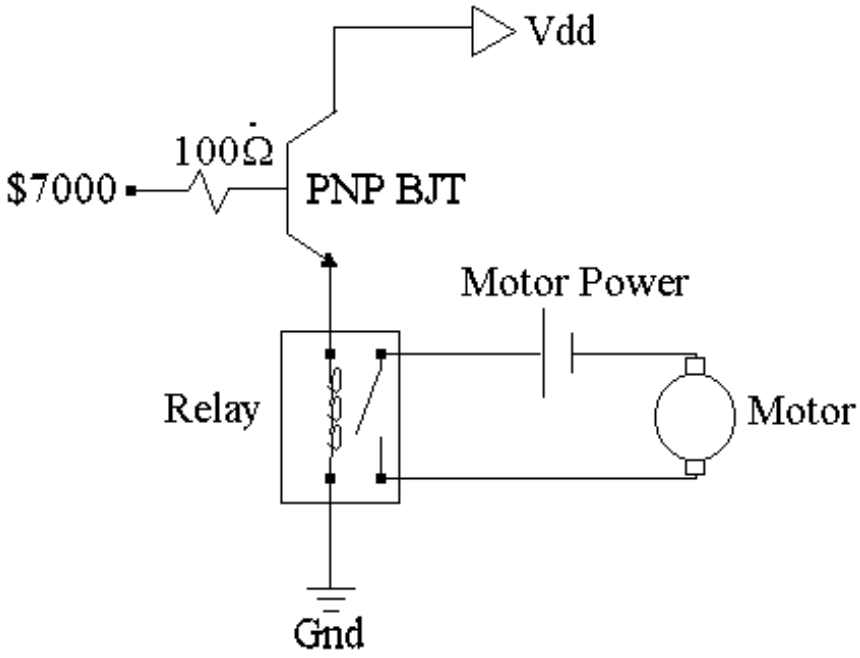

Figure 2. Additional Motor Controller Circuit

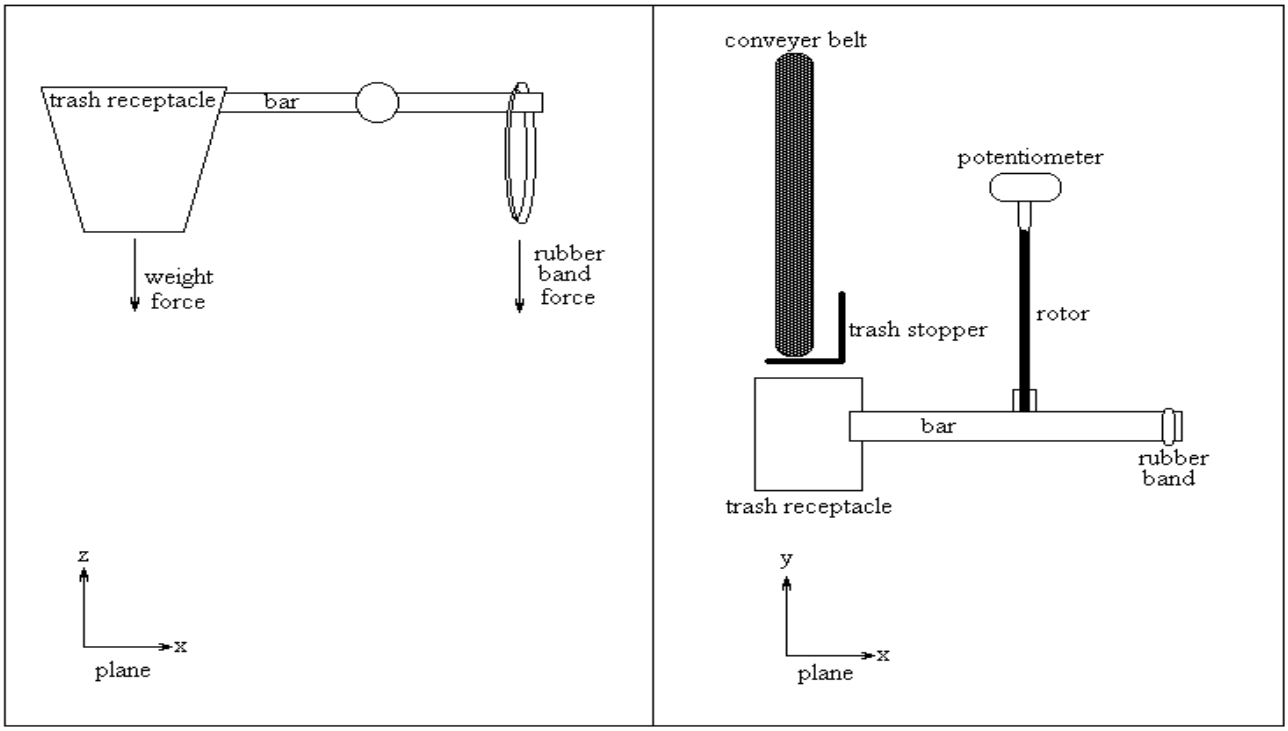

Figure 3. Trash Receptacle Sensor

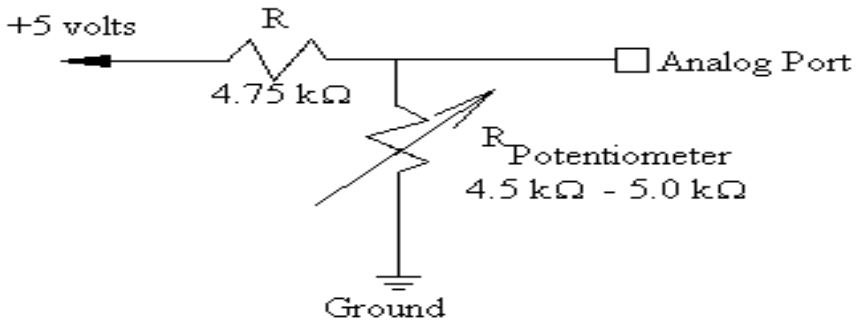

Figure 4. Trash Receptacle Sensor Circuit

APPENDIX

```
/*Aaron Grassian*/
/*IMDL 5666 Spring 1997*/
/*JACO Code*/



/*---------------------------------------------------------------*/
/*global initializations*/
/*collision avoidance sensor variables*/
int left_eye;
int right_eye;
int center_eye;
int bump;

/*trash pickup state variables*/
int trash_can;
int trash_found;
int trash_in_can;
int height_eye;
int ptrash_can;
int got_it;

/*test motor flags*/
int vacuum=1;
int belt=0;
int ir=1;

/*arbitraters*/
int bump_delay=0;
int ca_pri=0;
int spiral_pri=1;
int wall_follow=0;
float counter=0.0;
float actual_left;
float actual_right;
float ca_left;
float ca_right;
float spiral_right;
float spiral_left;
float wall_follow_right;
float wall_follow_left;
/*---------------------------------------------------------------*/
/*time delay function*/

void wait(int milli_seconds)        /*wait function definition*/
{
       long timer_a;
       timer_a=mseconds() +(long) milli_seconds;
       while(timer_a > mseconds())
      {
       defer();
      }
}
/*---------------------------------------------------------------*/

/*sensor module*/

void sensor_module()
```

```
{
while(1)
      {
      left_eye=analog(2);     /*read sensors into globals*/
      center_eye=analog(1);
      right_eye=analog(0);
      bump=analog(4);
      height_eye=analog(3);
      trash_can=analog(5);
      }
}

/*-----------------------------------------------------------------*/

void collision_avoid()           /*collision avoidance routine*/
{
while(1)
      {
      bump_delay=0;
      ca_pri=0;
      if (center_eye>=100 && height_eye>=95)
             {      /*if center detects a close object*/
             ca_pri=1;
             ca_left=-75.0;        /*back up and turn to the right*/
             ca_right=40.0;        /*right slow back, left fast back*/
             }
      else if (right_eye>95)
            {    /*if the right eye detects a close object*/
            ca_pri=1;
            ca_right=0.0;        /*swiftly turn left away from object*/
            ca_left=-50.0;         /*right=speed up, left = slow down*/
            }
      /*   else if (left_eye>115){ */   /*if the left eye detects a close
object*/
      /*  ca_pri=1;      */
      /*  ca_left=100.0; */       /*swiftly turn right away from object*/
      /*  ca_right=20.0; */        /*left=speed up, right=slow down*/
      /*  }  */
      else if (bump>70 && bump<90)
            {  /*if center bump turn back*/
            ca_pri=1;
            ca_left=-80.0;
            ca_right=-10.0;
            bump_delay=1;  /*set delay in arbitrater*/
            }
      else if (bump>115 && bump<140)
            {
            ca_pri=1;                       /*if right bump turn back*/
            ca_left=-100.0;
            ca_right=-20.0;
            bump_delay=1;  /*set delay in arbitrater*/
            }
      else if (bump>30 && bump<50)
            {  /*if left bump turn back*/
            ca_pri=1;
            ca_right=-100.0;
            ca_left=-20.0;
            bump_delay=1;          /*set delay in arbitrater*/
            }
      defer();
      }
```

```
}

/*-------------------------------------------------------------*/
void cleaning_mode ()
{
while(1)
      {
      if(spiral_pri==0) /*if just entering cleaning mode, reset spiral*/
            {          /*vacuum should be on*/
            spiral_right=90.0;
            spiral_left=0.0;
            }
      else if(spiral_left<70.0)    /*until right=90 and left=70*/
            {
            spiral_right=90.0;
            spiral_left=spiral_left + 0.01;
            }
      if(left_eye<=95 && right_eye <= 95 && center_eye>=95
         && center_eye <=100 && height_eye<95 && spiral_pri==1)
            /*if large trash object detected, set trash found flag*/
            {
            trash_found=1;                  /*vacuum should be off*/
            spiral_pri=0;
            }
      defer();
      }
}
/*-------------------------------------------------------------*/
void trash_can_sense ()
{
while(1)
      {
      if(trash_in_can==1)
            {
            trash_can=analog(5);
            got_it= ptrash_can-trash_can;
            if(got_it>=10)
                  {
                  spiral_pri=0;
                  wall_follow=1;
                  trash_in_can=0;
                  }
            }
      defer();
      }
}

/*-------------------------------------------------------------*/
void arbitrate ()
{
while(1)
      {
      if(ca_pri==1 && spiral_pri==1)
            {
            poke(0x7000,0b11111110);
            vacuum=1;
            belt=0;
            ir=1;
            motor(1,ca_right);
            motor(0,ca_left);
```

```
            if(bump_delay==1)
                  {
                  wait(1000);
                  bump_delay=0;
                  }
            }
      else if(spiral_pri==1 && ca_pri==0)
            {
            poke(0x7000,0b11111110);
            vacuum=1;
            belt=0;
            ir=1;
            motor(1,spiral_right);
            motor(0,spiral_left);
            }
      else if(wall_follow==1)
            {
            poke(0x7000,0b11111111);
            vacuum=0;
            ir=1;
            belt=0;
            motor(1,wall_follow_right);
            motor(0,wall_follow_left);
            if(bump_delay==1)
                  {
                  wait(1000);
                  }
            }
      else if(trash_found==1)
            {
            poke(0x7000,0b11111101);
            belt=1;
            vacuum=0;
            ir=0;
            motor(1,-50.0);
            motor(0,50.0);
            wait(2000);
            spiral_pri=0;
            ca_pri=0;
            trash_found=0;
            trash_in_can=1;
            trash_can=analog(5);
            ptrash_can=trash_can;
            }
      else if(trash_in_can==1)
            {
            poke(0x7000,0b11111101);
            belt=1;
            vacuum=0;
            ir=0;
            motor(1,0.0);
            motor(0,0.0);
            }
      defer();
      }
}
/*-------------------------------------------------------------*/
void wall_following ()
{
while(1)
      {
```

```
	bump_delay=0;
	if(right_eye>=95 && right_eye<=100)
		{
		wall_follow_right=75.0;
		wall_follow_left=75.0;
		}
	else if(right_eye>100 && right_eye<110)
		{
		wall_follow_right=100.0;
		wall_follow_left=0.0;
		}
	else if(center_eye>=105)
		{
		wall_follow_right=-20.0;
		wall_follow_left=-80.0;
		}
	else if(center_eye>=100 && right_eye>=95)
		{
		wall_follow_right=50.0;
		wall_follow_left=-50.0;
		}
	else if (left_eye>=100)
		{
		wall_follow_right=40.0;
		wall_follow_left=85.0;
		}
	else if (bump>70 && bump<90)
		{  /*if center bump turn back*/
		wall_follow_left=-80.0;
		wall_follow_right=-10.0;
		bump_delay=1;  /*set delay in arbitrater*/
		}
	else if (bump>115 && bump<140){
		wall_follow_left=-100.0;
		wall_follow_right=-20.0;
		bump_delay=1;  /*set delay in arbitrater*/
		}
	else if (bump>30 && bump<50)
		{  /*if left bump turn back*/
		wall_follow_right=-100.0;
		wall_follow_left=-20.0;
		bump_delay=1;          /*set delay in arbitrater*/
		}
	defer();
	}
}
/*-------------------------------------------------------------*/
void mode_setter ()
{
while(1)
	{
	if(spiral_pri==1)
		{
		counter=0.0;
		}
	else if(wall_follow==1)
		{
		counter=counter + 0.1;
		}
	if(counter>=500.0)
		{
```

```
            spiral_pri=1;
            wall_follow=0;
            }
      defer();
       }
}

/*-------------------------------------------------------------*/
void main()
{
start_process(sensor_module());      /*load in values every 100 msec*/
start_process(mode_setter());
start_process(collision_avoid());    /*actuate motors*/
start_process(cleaning_mode());
start_process(wall_following());
start_process(trash_can_sense());
start_process(arbitrate());
}
/*FIN*/
```