SnackBot

Chris Shepherd

EEL 5666

5/26/05

Table of Contents

Abstract

SnackBot is a mobile autonomous robot. His main purpose in life is to fetch you snacks and drinks so that you don't have to sacrifice either your viewing enjoyment or your snack craving. The secondary purpose of SnackBot is for entertainment. One command will cause SnackBot to enter a victory dance to celebrate a Gator's touchdown. He will do this through the use of remote control detection, line following, and color detection. This paper will cover all aspects of SnackBot's body and control.

Executive Summary

This is the final report for the robot SnackBot. This report will cover all aspects of the project. It will start with an overview of the idea generation for SnackBot and the problem that is solved by the robot. This will also include a discussion of SnackBot's main purpose and goals. This report will then proceed to characterize SnackBot through discussions of SnackBot's main features. This will include discussions of SnackBot's mobile platform, actuation, sensors, and behaviors. Each topic will give a thorough analysis of SnackBot's capabilities and how these capabilities help in achieving the desired goal. An overview of all testing procedures and results will then be presented. These will be used as evidence of the capabilities discussed. A conclusion as to the success of the project will then be offered and all necessary reference material will be attached.

Introduction

SnackBot solves a problem that all people are faced with at some point in their life. That time when one is immersed in one's favorite television program but has an urge for a favorite snack or beverage. Usually one can wait until a commercial comes on. But what if you're watching a movie, or, as many masters of the remote control will do, you are watching two or even three programs at the same time? In such situations there may not be an opportune time to run to the kitchen. In such a situation maybe you could rely on your roommate to run to the kitchen for you. But chances are he or she is just as involved in the program as you are. And even if they are willing to leave the room, now you owe them one. Now you have to get up the next time he or she has a snack craving. Now this would mean you only have to get up about half the time you have a carving, but the entire purpose of asking in the first place was so that you didn't have to miss any of your program.

This is where SnackBot comes in. SnackBot is built to be able to take remote control snack and drink orders, go to the kitchen, and return the item requested to you at the couch. This will allow you to enjoy your program without having to make the decision between missing some of the show or suffering through an unsatisfied craving.

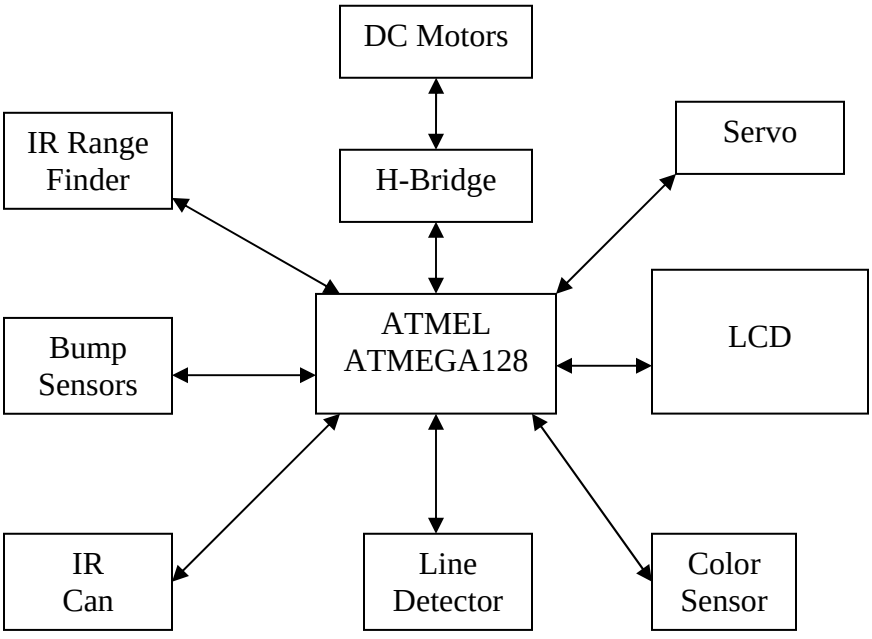
For the purpose of this project certain limitations must be established. Since every home is laid out differently, SnackBot will require a high contrast line to be placed on the floor in order for him to find his way to and from the kitchen and couch. Also, due to the limited time for this project SnackBot will not possess the ability to open and close refrigerator and pantry doors. Instead the available items will need to be placed at the end of each path of the line that SnackBot follows. SnackBot will distinguish one item

from another by the coloring of its package. In each the refrigerator and pantry there will be one red, one blue, and one green item. For our demonstrations these will be a Coca Cola, a Pepsi, and a Mountain Dew at the refrigerator, and an Original Pringles, a Sour Cream n' Onion Pringles, and a Chips Ahoy at the pantry.

Integrated System

The operating system for SnackBot is built upon the ATMEL ATMEGA128 processor board, which was ordered from bdmicro.com. Its complete datasheet is contained on the reference CD. This board will control all aspects of SnackBot's behavior. The board is connected to two DC motors through an H-Bridge, one servo, an LCD screen, a color sensor, a line tracking sensor, bump sensors, an IR range finder, and an IR can. The complete functional block diagram is shown below in Figure 1.

Figure 1



The DC motors are used for the movement of the robot while the servo is used to open and close a claw that grabs the requested items. The IR can is used to receive remote control commands. The IR range finder and bump sensors are used for obstacle avoidance as SnackBot makes its way along its path. The line tracking sensor is used to determine SnackBot's position with reference to the marked line so it can adjust accordingly. The color sensor is used to determine the color of the packaging of an item. This will distinguish one item from another. The LCD was used for debugging purposes while creating the software for SnackBot and will display messages as to owner as to its progress in fetching the item.

Mobile Platform

The platform of SnackBot was designed to help meet the objective of the robot. Each component of the platform was designed in AutoCad and then cut out on the shop's T-Tech machine. The platform consists of a 5.5" x 5.5" square bottom, and a 5" radius circular top. These pieces are connected by four 5" long square legs. The motors and wheels are placed under the bottom of the platform, causing the bottom to rest 2" above the ground. This ground clearance allows for the line follower and color sensor to be placed under the robot. The body of the robot thus blocks much of the ambient light, allowing more accurate readings from both sensors. This height also allows the claw to be attached at a height near the center of mass of a can of soda. This makes movement of the can easier and more stable as it is less likely to tip over.

The long legs place the top high enough off the ground to allow a can to fit under the canopy of the circular top. This top then holds a bumper connected to the bump

sensors. This places the bump switches far enough out as to still be useful in obstacle avoidance while a can is in tow in front of the robot.

Actuation

SnackBot has several moving components. The first category includes all components needed for the motion of SnackBot. SnackBot moves through the use of two DC motor-driven wheels. The motors chosen were the gear head motors GHM-02 from lynxmotion.com. They have a stall torque of 123.20 oz-in and run at 120 RPMs. Motors of this torque and speed were chosen after being advised by the TA that I would need motors with a stall torque of at least 100 oz-in. in order to move a full can of soda. The motors are connected to the processor board through a dual H-Bridge motor driver also found at lynxmotion.com. This H-Bridge has three inputs per motor. The first two determine whether the motor is being driven clockwise, counterclockwise, or is applying brakes. These inputs are connected to port A of the processor. I then defined constants FORWARD, BACKWARD, LEFT and RIGHT in terms of what must be applied to port A to turn the motors in the proper direction.

The third input then determines the speed of the motor. If an input of 5 volts is held constant the motor will run at full speed. Conversely, if an input of zero volts is applied the motor is turned off. If a waveform with an average voltage somewhere between 0 and 5 volts is applied the motor will run at a speed determined by this average voltage. Therefore, to control the speed of each motor a PWM was set up on timer 3 of the processor. Timer 3 was set up in mode 8 (see page 133 of the processor's datasheet). This created a PWM of frequency 30 Hertz that could be used to control the motors and

the servo for the hand. By adjusting OCR3n I could adjust the duty cycle of the PWM on that pin, and thus control the speed of each motor. For examples see the code attached in the appendices.

The second category of actuation includes all components needed in the grabbing of an item. This system was made mostly from the Joinmax Digital gripper kit JM-GRP-01 from pololu.com. However the gripping fingers on this kit are only about an inch long. These were obviously not long enough to wrap around and grab a can. Thus I cut custom fingers out of wood that would fasten to the gripper fingers. These custom fingers were designed to be the radius of a Pringles can and wrap around and behind the can, securing it tightly.

This kit uses a servo to rotate gears that open and close the hand. The control of this servo is much simpler than that of the motors. The servo requires a waveform of frequency between 20 and 40 Hertz with a high pulse between 1 and 2 milliseconds. This is why I chose to use a 30 Hertz wave for the motors, so that both the motors and this servo could be controlled by the same internal timer. If the high pulse is 1 millisecond long the hand will close completely, and if the high pulse is 2 milliseconds long, the hand will open completely. Due to the front legs and custom fingers the hand could not be opened or closed completely. So I varied the waveform and through trial and error was able to determine a suitable waveform for opening and closing the hand. This code can also be found in the appendices.

Sensors

In order to accomplish the set objective SnackBot is equipped with five sensors. The first of these is a Sharp IR can used in receiving remote control commands. The part simply has three pins, a power, ground, and output. The output is self-modulated at 40 kHz to block out all frequencies except that of the remote. The output was connected port D7. This pin was then simply polled for all data. The data is sent serially as a set of pulses. The data is determined by measuring the time between pulses. A start bit is defined as a space of 2 to 4 milliseconds between pulses. Once this start bit has been received four data bits were read in. These bits were also determined by the time between pulses. If a gap of less than one millisecond was found, the bit is a zero. If the gap is over one millisecond the bit is a one. Once these four bits were received they made up a number between 0 and 15. I chose to use numbers 1-3 as the three options for drinks, numbers 4-6 as the options for snacks, and 7 was added for entertainment and used to make SnackBot dance in celebration of a gator's touchdown.

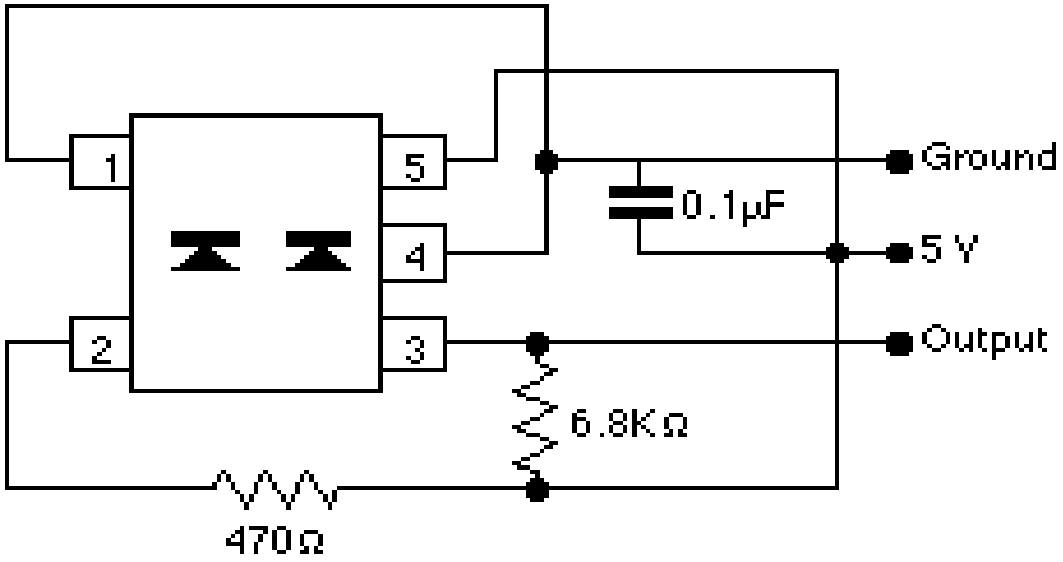
The next two sensors were used in avoiding obstacles. These are the bump sensors and the IR range finder. The bump sensors are simply push button switches commonly used as reset switches. These switches provide no conductivity between their two pins if the button is not pressed, and shorts the pins together if the button is pressed. The buttons were connected to a bumper wrapping 180 degrees around SnackBot. One pin was then connected to ground, while the other pin was connected to port C, pins 4 through 7 were used for the four bump switches. In software I then enabled the pull up resistors on port C. In this manner each pin on port C will read high due to the pull up resistors if the bumper is not compressed, and will read low if the bumper has been

pressed. These values are read in continuously while the robot is moving and are used to signal the processor if the robot runs into something.

To try and prevent the robot from ever hitting an object in the first place, a Sharp IR range finder (GP2D12) from acroname.com was used. This device only has three pins: power, ground, and an analog output. The sensor emits infrared light and then measures the intensity of light reflected back to it. This intensity is represented by the analog output. The closer an object is to the sensor, the higher the intensity of reflected light, and thus the higher the output voltage. This analog value was then connected to pin 0 of port F. Port F can then be set as an A2D converter and convert this analog value to a digital number. SnackBot will continuously read in this value while moving, and if this number ever exceeds a threshold determined in a start up calibration routine the robot will stop to avoid hitting the object. SnackBot will then warn the owner that something is along its designated path and ask it to be removed.

Now that SnackBot can receive a command and avoid any obstacle along its way, it must be able to follow a line to and from the kitchen. This is accomplished through the use of the fourth sensor, a custom made line detector. The main component to the line detector is the Hamamatsu Photoreflector found at acroname.com. These reflectors, like the range finder, also emit infrared light and measure the amount reflected back. However, unlike the range finder, the output voltage is digital. If the reflector is placed over a black surface an output of 0 volts is generated. If placed over a white surface the output is 5 volts. Four such reflectors were used, each wired in the manner seen in Figure 2. Two are positioned in the center of the robot, and two towards the outside. The two in

Figure 2



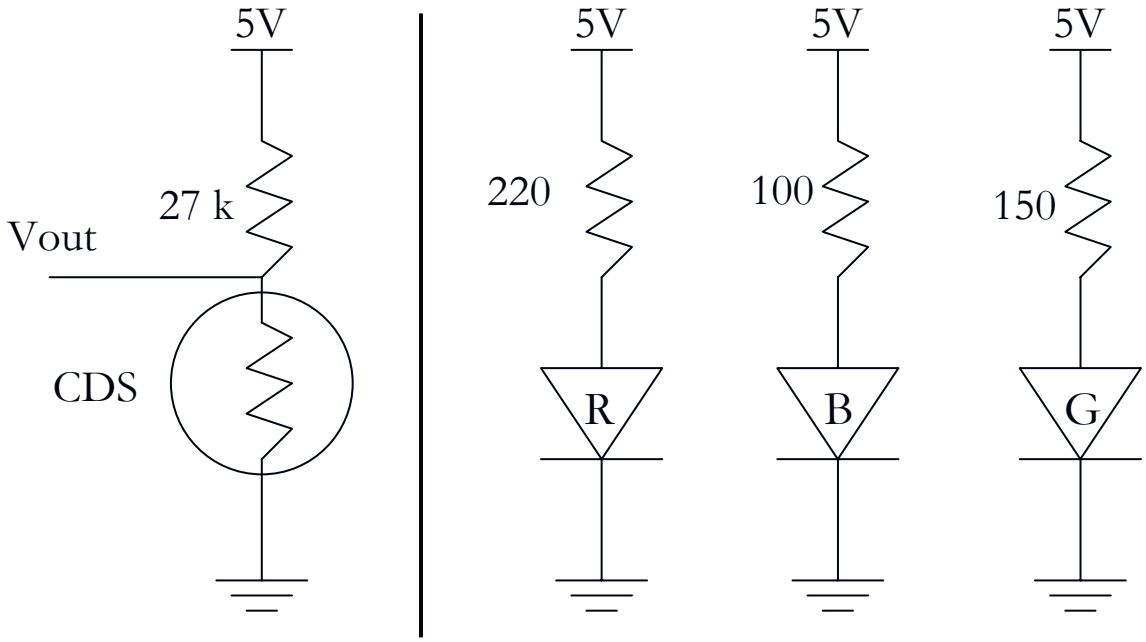
the center were used to determine if SnackBot is directly over the line. If only one of the two is over the line then SnackBot is slightly off center and adjusts by making a small turn in the direction of the line. If SnackBot gets off the line far enough as to have an outside sensor detect the line, he will adjust by making a much sharper turn inwards. The outside sensors are also used to detect intersections. If both of the outside reflectors sense a line at the same time then SnackBot must be at an intersection. These intersections are used to inform SnackBot as to his whereabouts on his path. The four reflectors are attached to port C pins 0-3. While line following these pins are read continuously and used to inform SnackBot has to the direction of the line.

The last sensor is the color sensor. This sensor is used to determine the color of the packaging of each object, which is how one item will be separated from the others. The theory behind this sensor is a bit more complicated then the previous sensors. The basic theory is that objects of different colors look different colors because of the wavelength of the light they reflect. For example, a red object looks red because it

reflects “red wavelengths” while absorbing all other wavelengths. Conversely a blue object appears blue because it absorbs this “red wavelength” light as well as all other wavelengths that are not “blue wavelength” light. Going a step further, in the presence of only red light, a red object will reflect all light while a blue or green object will absorb all light and thus appear black. This theory is taken advantage of through the use of a CDS cell and three LEDs

. A CDS cell is a variable resistor whose resistance changes dependant upon the amount of light that hits the cell. The color sensor turns on one LED at a time to shine only red, blue, or green light. A red object will reflect all red light, causing the CDS cell to have a much lower resistance than if a blue or green object, which will absorb red light, were present. Each time an LED is lit up the analog value is read from the voltage divider circuit shown in Figure 3 and converted to a digital value. In theory the lower of the three readings will correspond to the object’s color.

Figure 3



Behaviors

SnackBot is capable of several behaviors. He is able to take commands from a remote control, avoid obstacles, follow a line, and determine color. As discussed above the detection of remote control commands is done through the use of a Sharp IR can. To accomplish this task the two subroutines wait_for_low_to_high() and wait_for_low() were written (These can be found in the final program in the appendix). Wait_for_low_to_high() is used to detect the rising edge that signifies the end of a pulse. This function simply waits in an infinite loop until the signal goes low. It then has a short delay to account for the fall time of the signal before entering another infinite loop that checks for a rising edge. Once this rising edge is detected TCNT0 and ms_count are reset to zero. As TCNT0 runs it will increment ms_count every one millisecond. The value of ms_count is then read when the function wait_for_low() detects a falling edge. This function is similar to wait_for_low_to_high() in that it contains an infinite loop that is terminated by a change in voltage level. Ms_count is then used to determine the data that was transmitted.

If the value of ms_count is 2, 3, or 4 then the data transmitted was a start bit. The software waits for a valid start bit and then reads in the next four bits of information. Again ms_count is used to determine the length of the time between pulses. If the gap is less than one millisecond then the bit is a zero. If the gap is greater than one millisecond then the bit is a one. Each bit is read and shifted into the variable value as the most significant bit. After all bits are collected value is incremented by one. This makes value a number between 1 and 16 and causes value to match the number pressed on the remote (i.e. buttons 1-7 are now mapped to values 1-7 rather than 0-6). While this was not

necessary, it makes the code a little easier to understand. Finally, the process is repeated until the same value is read in twice in a row. This helps in reducing errors caused by ambient light fluctuations.

The second behavior of SnackBot is obstacle avoidance. While SnackBot is following a predetermined path, and thus should not have to worry about walls or large pieces of furniture, there is no guarantee that the path will be free of debris. Possible objects in the path of SnackBot could include shoes, backpacks, or even the owner's dog. Obviously SnackBot wished to avoid upsetting the dog by running into it, so SnackBot must be able to detect objects in its path in order to stop and avoid the collision. SnackBot does so through the use of the IR range finder. Referring to the final program in the appendices, you will notice a short calibration routine at the beginning of the program. This routine allows the user to place an object in front of SnackBot any distance they wish to call the threshold. They then press the left bumper to signal SnackBot to take the threshold reading. This threshold is used to determine if SnackBot is too close to an object. If SnackBot gets closer to an object than the desired threshold he will stop immediately and give the user a warning. Once the user has removed the object SnackBot will continue with its job.

When SnackBot is returning from the kitchen he will be towing a snack or drink. This item will be held in front of the robot and thus cause SnackBot to always sense an object directly in front of it. Therefore the IR range finder must be ignored while and item is in tow. In order to avoid obstacles in this situation SnackBot is also fitted with the bump sensor discussed earlier. While this will not prevent SnackBot from hitting an

object, it will prevent SnackBot from trying to run over the object, and will alert the user that the path needs to be cleared.

The third behavior is SnackBot's ability to follow lines. This is done using the line detector sensor discussed earlier. The line detector continuously updates SnackBot as to its position in relation to the line. If both of the middle sensors are positioned over the line SnackBot will continue moving straight. This is done in the code in the appendices by setting port A equal to FORWARD with the motors having equal speed. If one of the two center sensors does not detect the line SnackBot will know he is slightly off the line. In this event SnackBot continues moving forward, but the speed of the motors is adjusted. The motor nearer to the line is slowed down causing SnackBot to turn in that direction.

If SnackBot were to get off of the line so far as to have one of the outside sensors detect a line he will make a much more drastic turn. In this situation SnackBot will all but completely turn off the motor nearer to the line while speeding up the outside motor to almost double its normal speed. This allows SnackBot to reacquire the line and continue on its path. The values of the motor speeds were determined through trial and error until a combination that provided a smooth and reliable line following was found.

While line following, SnackBot is also able to detect intersections. These intersections are used to relay SnackBot's position along the path to himself. This allows SnackBot to know where to turn towards the refrigerator or pantry, when to stop and release the item for its owner, and when it is back at it's home position. These intersections are detected when the line detector detects lines under both of the outside sensors.

Once SnackBot reaches the refrigerator or pantry he must be able to pick out the item you requested. This is done through his last behavior, color detection. There will be one red, one blue, and one green item in both the snack and drink categories. SnackBot will pick out the item requested based on the color of its packaging. To accomplish this SnackBot will be equipped with the color sensor discussed earlier. This color detection is accomplished in the subroutine detect_color(). The powers for the three LED's are connected to the most significant three bits of port A. Each LED is lit up one at a time while a reading of the CDS cell voltage divider is taken. The algorithm for determining the color is as follows: if the three readings are to close to each other to distinguish a color then the object is green; if the blue reading is less than the red reading the object is blue; if this fails and the red reading is less than the green reading the object is red; if this also fails the object must be blue. A more in depth analysis of how this algorithm was determined is available in the Experimental Layout and Results Section.

Experimental Layout and Results

Each of the experiments discussed in this section were performed multiple times. They were performed once while the sensor was built on a breadboard as a proof of concept. They were then repeated once the sensor boards had been cut out and soldered to prove that the boards were functioning properly. They were then repeated a final time once mounted on SnackBot. These tests were then used in the writing of SnackBot's software. This report will concentrate on these final tests as they are the most prevalent to the finished product. If you wish to study the results of the earlier tests, they are included in the accompanying notebook.

Experiment 1: Bump Sensors

The test of the bump sensors was very simple. The values of all bump switches were read in through port C. Since the switches were wired as active low the value was then inverted. This value was then printed to the LCD screen. This process was repeated continuously so all combinations of the switches could be measured. Each switch worked as expected in that it returned a value of 1 is the bumper was compressed, and a value of zero if the bumper was not pressed. Since the sensor is purely mechanical there was no need to test the sensor under different lighting circumstances.

Experiment 2: IR Range Finder

The range finder is a very simply interface as well, with only power, ground, and an output. The output was connected to pin 0 of port F. This port was then configured to make analog to digital conversions. The results of these conversions were then written to the LCD screen. Since the range finder is dependant on the amount of light reflected, the first test was to place an object an equal distance directly in front of the sensor under different lighting conditions. It quickly became apparent that that the sensor had some sensitivity to the ambient light. In a well lit room the reading would always be higher than a reading taken in a darker room. For this reason there was no need to manually determine a threshold value, as the threshold would relate to completely different distances under different lighting conditions. Therefore the calibration routine discussed earlier was written. This routine was then tested in an obstacle avoidance test program. The speed of the motors was varied and SnackBot successfully recognized the object and

stopped each time. The only result of note is that the stopping distance of SnackBot noticeably increased with increased speed.

Experiment 3: Remote Control

The remote control is another simple interface that took little hardware testing. The algorithm discussed earlier was written to read in and decipher the remote control signal. The output of this algorithm was then printed on the LCD screen. Once the program was working correctly several tests were conducted to determine the range and reliability of the algorithm. Again I expected ambient lighting conditions to affect the sensors performance. SnackBot was taken into my living room and placed next to a pair of sliding glass doors with the shades drawn. Direct sunlight did affect the performance slightly. While indirect sunlight approximately one out of every five readings had at least one bit error. To reduce this error I modified the program so that it has to receive the same command three times in a row before it will accept it. Statistically this will yield a correct result 99.2% of the time. In all further tests conducted I have yet to encounter an incorrect reading.

A test was also conducted to determine the range of the remote. Again SnackBot was placed in the worst possible conditions, in the living room with every light on and the sliding glass door shades drawn. The farthest I could get from SnackBot and still have a direct line of view was approximately 30 feet. At this distance SnackBot correctly determined the command given every time. Only once I left a direct line of sight to SnackBot was I able to get out of the range of the remote. This test proves that there will

be no problems with SnackBot's ability to receive commands, as the user will always be within sight of the robot.

Experiment 4: Line Detector

The line detector had only one test. Once it was mounted it was simply tested to ensure that it could indeed detect a line. This test program was very similar to the test program for the bump switches. The program read in the values of the four line detectors and output these values to the LCD screen. Like the bump switches, the line detector is also active low, returning a 0 if over the black line, and a 1 if over the white background. Therefore the values of each reflector were inverted to represent logic true and false. The LCD correctly outputted a one corresponding to each reflector that was over the line, and a zero for each reflector that was not over a line. Once the line detector was proved to be functioning properly the line following algorithm was created through simple trial and error.

Experiment 5: Color Sensor

To test the color sensor a test program was written to take the three color readings and display the results to the screen. This test was conducted under several lighting conditions with very little variation in each test. The average values for the readings are given in Figure 4.

Figure 4

Object	Red Light	Blue Light	Green Light
Coke	399	451	481
Pepsi	500	447	560
Mt. Dew	441	427	449
Original Pringles	444	577	545
Sour Cream n' Onion Pringles	563	602	546
Chips Ahoy	589	530	589

Earlier tests had led to the following algorithm: if the blue reading is less than the red reading the object is blue; if this fails and the red reading is less than the green reading the object is red; if this also fails the object must be blue. This algorithm holds true for all but the Mt. Dew. After much thought I can only attribute this to the shininess and glossiness of a can combined with the many graphics on a can of Mountain Dew. As you can see these factors lead to reading that are separated by a maximum of 27. This is in sharp contrast to the other five items, whose lowest maximum difference is more than twice as much at 56. After many unsuccessful attempts to adjust the algorithm so that the Mountain Dew would stand out more I decided to take a much simpler approach. I realized that the next closest maximum difference was of the other green object, the Sour Cream n' Onion Pringles. Thus it is apparent that green simply is a mixture of the three colors in close proportions. Therefore I added a prerequisite to the previous algorithm. If the maximum difference between the three readings is less than 45, it must be green. This weeds out the Mountain Dew, and possibly the Sour Cream n' Onion Pringles under

the right lighting conditions, before continuing with the previously stated algorithm. Once this prerequisite was added the color sensor has worked perfectly. I also added a loop so that it has to determine that an object is the same color twice in a row before it will accept that as a valid reading. This adds another level of error checking to increase reliability.

Conclusion

In summary SnackBot is an autonomous robot that is able to receive commands through a remote control. These commands correspond to three snacks and three drinks as well as an entertaining dance to celebrate a gator touchdown. SnackBot is then able to follow a predetermined path into the kitchen while avoiding obstacles. Once in the kitchen SnackBot goes to the pantry or refrigerator and inspects three items, one red, one blue, and one green. Based upon the coloring of the item SnackBot is able to determine if an item is the item the user requested. Once the requested item is found SnackBot will return the item to the user at the couch before returning to its home by the TV.

Over all I am very please with the results of SnackBot. The custom made line follower and color detector worked much better than I expected. Surprisingly, the hardest part was trying to figure out how to make 90 degree turns. I failed to properly place the line detector so as to detect an intersection at the correct distance to be able to simply turn one wheel off and the other on during the turn. Instead I had to turn as if attempting a 360 but varying the speeds of the two motors. If the relationship between the two speeds were not just right the robot would make its 90 degree turn but rather be off by a centimeter or two. The robot would then keep turning until it reacquired the line, making it turn approximately 10 degrees too far or too little.

If given further time to upgrade SnackBot I would do two main things. First, I would like to rebuild his platform out of a more aesthetically pleasing material, such as plastic. Secondly I would want to add the ability to lift the items off the ground. The items could be placed in a basket on top of SnackBot. That way he would be able to grab multiple items on a single trip.

Appendices

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <stdlib.h>
#include <inttypes.h>
#include <stdio.h>
#include "adc.h"

#define CPU_FREQ 16000000L  /* set to clock frequency in Hz */
#define LCD_DATA PORTB
#define LCD_CTRL PORTD
#define FORWARD 0x05
#define BACKWARD 0x0A
#define LEFT 0x09
#define RIGHT 0x06
#define STOP 0x00
#define START_MIN 100 /* 16us X 120 = 2.0ms min start bit */
#define START_MAX 260 /* 16us X 260 = 4.2ms max start bit */
#define BITTIME 60    /* 16us X 63  = 1ms   700us = 0  1200us = 1  */

#if CPU_FREQ == 16000000L
#define OCR_1MS 125
#elif CPU_FREQ == 14745600L
#define OCR_1MS 115
#endif
volatile uint16_t ms_count;

/*
 * ms_sleep() - delay for specified number of milliseconds
 */
void ms_sleep(uint16_t ms)                                          //the setup of
the interupts and timers needed for the ms_sleep
{
        //function were taken from the hw (hello world) program from bdmicro.com
  TCNT0  = 0;
  ms_count = 0;
  while (ms_count != ms)
    ;
}
/*
 * millisecond counter interrupt vector
 */
SIGNAL(SIG_OUTPUT_COMPARE0)
```

```
{
  ms_count++;
}
/*
 * Initialize timer0 to use the main crystal clock and the output
 * compare interrupt feature to generate an interrupt approximately
 * once per millisecond to use as a general purpose time base.
 */
void init_timer0(void)
{
  TCCR0 = 0;
  TIFR  |= _BV(OCIE0)|_BV(TOIE0);
  TIMSK |= _BV(TOIE0)|_BV(OCIE0);         /* enable output compare interrupt */
  TCCR0  = _BV(WGM01)|_BV(CS02)|_BV(CS00); /* CTC, prescale = 128 */
  TCNT0  = 0;
  OCR0   = OCR_1MS;                    /* match in aprox 1 ms */
}
// Initializes the display
void lcd_initialize(void)
{
        LCD_DATA = 0x38; LCD_CTRL = 0x04; ms_sleep(1); LCD_CTRL = 0x00;
ms_sleep(1);
        LCD_DATA = 0x0C; LCD_CTRL = 0x04; ms_sleep(1); LCD_CTRL = 0x00;
ms_sleep(1);
        LCD_DATA = 0x06; LCD_CTRL = 0x04; ms_sleep(1); LCD_CTRL = 0x00;
ms_sleep(1);
}
// Clears all text from the screen
void lcd_clear_display(void)
        {LCD_CTRL = 0x04; LCD_DATA = 0x01; ms_sleep(1); LCD_CTRL = 0x00;
ms_sleep(1);}

// Outputs a single character to the screen
void lcd_put_char(char c)
        {LCD_CTRL = 0x05; LCD_DATA = c; ms_sleep(1); LCD_CTRL = 1;
ms_sleep(1);}

// Outputs a string message (character array) to the screen
void lcd_put_string(char message[])
{
        int i;
        for(i=0;i<21;i++)
        {
                if(message[i]=='\0') break;
                else lcd_put_char(message[i]);
        }
```

```
}
// Turns the cursor on - cursor("on") and off -  cursor ("off")
void lcd_cursor(char command[])
{
	if(command[1]==116 || command[1]==110)
		{LCD_DATA = 0x0E; LCD_CTRL = 0x04; ms_sleep(1); LCD_CTRL =
0x00; ms_sleep(1);}
	else
		{LCD_DATA = 0x0C; LCD_CTRL = 0x04; ms_sleep(1); LCD_CTRL =
0x00; ms_sleep(1);}
}
// Goes to the beginning of the line specified by its argument (valid ranges = 1 to 4)
void lcd_goto_line(int line)
{
	switch(line)
	{
		case(1): {LCD_DATA = 0x80; LCD_CTRL = 0x04; ms_sleep(1);
LCD_CTRL = 0x00; ms_sleep(1); break;}
		case(2): {LCD_DATA = 0xC0; LCD_CTRL = 0x04; ms_sleep(1);
LCD_CTRL = 0x00; ms_sleep(1); break;}
		case(3): {LCD_DATA = 0x94; LCD_CTRL = 0x04; ms_sleep(1);
LCD_CTRL = 0x00; ms_sleep(1); break;}
		case(4): {LCD_DATA = 0xD4; LCD_CTRL = 0x04; ms_sleep(1);
LCD_CTRL = 0x00; ms_sleep(1); break;}
	}
}
//writes an eight bit value to the LCD in hex format
void lcd_write_int(uint16_t num)
{
	uint16_t i;

	i = num;
	i = i >> 4;
	i = i & 0x0f;			//isolates the first nibble
	LCD_CTRL = 0x05;
	if (i>9)
	{
		i = i - 9;
		i = i + 0x40;		//converts to ASCII
	}
	else
		i = i + 0x30;
	LCD_DATA = i;
	ms_sleep(1);
	LCD_CTRL = 1;
	ms_sleep(1);
```

```
	LCD_CTRL = 0x05;
	num = num & 0x0f;		//isolates the second nibble
	if (num>9)
	{
		num = num - 9;
		num = num + 0x40;		//converts to ASCII
	}
	else
		num = num + 0x30;
	LCD_DATA = num;
	ms_sleep(1);
	LCD_CTRL = 1;
	ms_sleep(1);
}
//this function waits for a rising edge on the most significant bit of port D
void wait_for_low_to_high(void)
{
	while(!(PIND & 0x80))
		;					/* if it's high, wait for
a low */
	for(int i =0;i<3;i++);			/* account for fall time */
	while((PIND & 0x80))
		;					/* wait for signal to
go high */
}
//this function waits for a falling edge on the most significant bit of port D
void wait_for_low(void)
{
	while(!(PIND & 0x80))
		;					/* wait for signal to
go high */
	for(int i =0;i<3;i++);   /* account for fall time */
}
//This function writes a 10 bit value to the LCD screen in hex format
//was used to output the 10-bit A2D conversion results for the color sensor and IR range
finder
void lcd_write_int_10_bit(uint16_t num)
{
	uint16_t i;

	i = num;
	i = i >> 8;
	i = i & 0x03;				//isolates top two bits
	LCD_CTRL = 0x05;
	if (i>9)
```

```
	{
		i = i - 9;
		i = i + 0x40;			//converts to ASCII
	}
	else
		i = i + 0x30;
	LCD_DATA = i;
	ms_sleep(1);
	LCD_CTRL = 1;
	ms_sleep(1);

	i = num;
	i = i >> 4;
	i = i & 0x0f;				//isolates next four bits
	LCD_CTRL = 0x05;
	if (i>9)
	{
		i = i - 9;
		i = i + 0x40;			//converts to ASCII
	}
	else
		i = i + 0x30;
	LCD_DATA = i;
	ms_sleep(1);
	LCD_CTRL = 1;
	ms_sleep(1);

	LCD_CTRL = 0x05;
	num = num & 0x0f;			//isolates lowest four bits
	if (num>9)
	{
		num = num - 9;
		num = num + 0x40;		//converts to ASCII
	}
	else
		num = num + 0x30;
	LCD_DATA = num;
	ms_sleep(1);
	LCD_CTRL = 1;
	ms_sleep(1);
}
//this function follows a line while backing up until it hits an intersection
void backward_to_intersection(void)
{
	uint16_t left, front_left, front_right, right;
	unsigned char line;
```

```
	int intersections;

	intersections = 0;

	do
	{
		line = PINC;					//reads in line detector values
and determines which sensors are over a line

		if ((line & 0x08) == 0)
			right = 1;
		else
			right = 0;

		if ((line & 0x04)==0)
			front_right = 1;
		else
			front_right = 0;

		if ((line & 0x02)==0)
			front_left = 1;
		else
			front_left = 0;

		if ((line & 0x01)==0)
			left = 1;
		else
			left = 0;

		if ((left == 0) && (front_left == 1) && (front_right == 1) && (right ==
0))  //robot centered above line -> keep going straight
		{
			PORTA = BACKWARD;
			OCR3A = 25000;
			OCR3B = 25000;
		}
		else if ((left == 0) && (((front_left == 0) && (front_right == 1)) ||
((front_left == 1) && (right == 1))))
		{
			PORTA = BACKWARD;
								//robot slightly to the
left -> turn right slowly
			OCR3A = 25000;
			OCR3B = 20000;
		}
```

```
		else if ((right == 0) && (((front_right == 1) && (left == 1)) || ((front_right
== 0) && (front_left == 1))))  //left
		{
			PORTA = BACKWARD;
                                                                //robot slightly to the
right -> turn left slowly
			OCR3A = 20000;
			OCR3B = 25000;
		}
		else if ((left == 0) && (front_left == 0) && (front_right == 0) && (right
== 1))  //sharp right
		{
			PORTA = BACKWARD;
                                                                //robot to the left ->
turn right sharply
			OCR3A = 40000;
			OCR3B = 30000;
		}
		else if ((left == 1) && (front_left == 0) && (front_right == 0) && (right
== 0))  //sharp left
		{
			PORTA = BACKWARD;
                                                                //robot to the right ->
turn left sharply
			OCR3A = 30000;
			OCR3B = 40000;
		}
		else if ((left == 1) && (right == 1))                   //if intersection is
detected
		{
			intersections++;
		}
	} while (intersections != 1);              //continue until intersection is detected
}
//this function follows a line while going forwards for a set amount of time
//used to touchdown celebration since don't want to go all the way to an intersection
void follow_line_for_time(int time, uint16_t threshold)
{
	uint16_t ir, left, front_left, front_right, right, left_bump, front_left_bump,
front_right_bump, right_bump;
	unsigned char bump, line;

	for (int i = 0; i < time; i++)
	{
		line = PINC;                                    //reads in line detector values
and determines which sensors are over a line
```

```
		if ((line & 0x08) == 0)
			right = 1;
		else
			right = 0;

		if ((line & 0x04)==0)
			front_right = 1;
		else
			front_right = 0;

		if ((line & 0x02)==0)
			front_left = 1;
		else
			front_left = 0;

		if ((line & 0x01)==0)
			left = 1;
		else
			left = 0;

		ir = adc_readn(0,5);  /* sample channel 0 5 times, take average */
//takes reading from IR range finder

		bump = PINC;				//reads in bump switche values and
determines which switches are compressed

		if ((bump & 0x80)==0)
			right_bump = 1;
		else
			right_bump = 0;

		if ((bump & 0x40)==0)
			front_right_bump = 1;
		else
			front_right_bump = 0;

		if ((bump & 0x20)==0)
			front_left_bump = 1;
		else
			front_left_bump = 0;

		if ((bump & 0x10)==0)
			left_bump = 1;
		else
			left_bump = 0;
```

```
                if ((ir >= threshold) || (front_left_bump == 1) || (left_bump == 1) ||
(front_right_bump == 1) || (right_bump == 1))
                {
                        i--;                                                    //if stopped
due to bumper or IR stopp counter
                        PORTA = STOP;
                        ms_sleep(100);
                        lcd_put_string("Object in Path");
                        lcd_goto_line(4);
                        lcd_put_string("Please remove object");
                        lcd_goto_line(3);
                }
                else if ((left == 0) && (front_left == 1) && (front_right == 1) && (right
== 0))  //robot centered above line -> keep going straight
                {
                        PORTA = FORWARD;
                        OCR3A = 25000;
                        OCR3B = 25000;
                        lcd_put_string("                ");
                        lcd_goto_line(4);
                        lcd_put_string("                ");
                        lcd_goto_line(3);
                }
                else if ((left == 0) && (((front_left == 0) && (front_right == 1)) ||
((front_left == 1) && (right == 1))))  //right
                {
                        PORTA = FORWARD;
        //robot slightly to the left -> turn right slowly
                        OCR3A = 24000;
                        OCR3B = 4000;
                        lcd_put_string("                ");
                        lcd_goto_line(4);
                        lcd_put_string("                ");
                        lcd_goto_line(3);
                }
                else if ((right == 0) && (((front_right == 1) && (left == 1)) || ((front_right
== 0) && (front_left == 1))))  //left
                {
                        PORTA = FORWARD;
        //robot slightly to the right -> turn left slowly
                        OCR3A = 4000;
                        OCR3B = 24000;
                        lcd_put_string("                ");
                        lcd_goto_line(4);
                        lcd_put_string("                ");
```

```
                lcd_goto_line(3);
        }
        else if ((left == 0) && (front_left == 0) && (front_right == 0) && (right
== 1))  //sharp right
        {
                PORTA = FORWARD;
    //robot to the left -> turn right sharply
                OCR3A = 50000;
                OCR3B = 1000;
                lcd_put_string("                ");
                lcd_goto_line(4);
                lcd_put_string("                ");
                lcd_goto_line(3);
        }
        else if ((left == 1) && (front_left == 0) && (front_right == 0) && (right
== 0))  //sharp left
        {
                PORTA = FORWARD;
    //robot to the right -> turn left sharply
                OCR3A = 1000;
                OCR3B = 50000;
                lcd_put_string("                ");
                lcd_goto_line(4);
                lcd_put_string("                ");
                lcd_goto_line(3);
        }
    }
}
//this function follows a line while going forwards until an intersection is reached
void follow_line_to_intersection(uint16_t threshold)
{
    uint16_t ir, left, front_left, front_right, right, left_bump, front_left_bump,
front_right_bump, right_bump;
    unsigned char bump, line;
    int intersections;

    intersections = 0;

    do
    {
        line = PINC;                                            //reads in line
detector values and determines which sensors are over a line

        if ((line & 0x08) == 0)
            right = 1;
        else
```

```
                right = 0;

        if ((line & 0x04)==0)
                front_right = 1;
        else
                front_right = 0;

        if ((line & 0x02)==0)
                front_left = 1;
        else
                front_left = 0;

        if ((line & 0x01)==0)
                left = 1;
        else
                left = 0;

        ir = adc_readn(0,5);  /* sample channel 0 5 times, take average */
//takes reading from IR range finder

        bump = PINC;                                            //reads in
bump switch values and determines which switches are compressed

        if ((bump & 0x80)==0)
                right_bump = 1;
        else
                right_bump = 0;

        if ((bump & 0x40)==0)
                front_right_bump = 1;
        else
                front_right_bump = 0;

        if ((bump & 0x20)==0)
                front_left_bump = 1;
        else
                front_left_bump = 0;

        if ((bump & 0x10)==0)
                left_bump = 1;
        else
                left_bump = 0;

        if ((ir >= threshold) || (front_left_bump == 1) || (left_bump == 1) ||
(front_right_bump == 1) || (right_bump == 1))
        {
```

```
                PORTA = STOP;
        //if object in path stop, give warning, and wait for it to be removed
                ms_sleep(100);
                lcd_put_string("Object in Path");
                lcd_goto_line(4);
                lcd_put_string("Please remove object");
                lcd_goto_line(3);
        }
        else if ((left == 0) && (front_left == 1) && (front_right == 1) && (right
== 0))  //robot centered above line -> keep going straight
        {
                PORTA = FORWARD;
                OCR3A = 25000;
                OCR3B = 25000;
                lcd_put_string("                ");
                lcd_goto_line(4);
                lcd_put_string("                ");
                lcd_goto_line(3);
        }
        else if ((left == 0) && (((front_left == 0) && (front_right == 1)) ||
((front_left == 1) && (right == 1))))  //right
        {
                PORTA = FORWARD;
        //robot slightly to the left -> turn right slowly
                OCR3A = 24000;
                OCR3B = 4000;
                lcd_put_string("                ");
                lcd_goto_line(4);
                lcd_put_string("                ");
                lcd_goto_line(3);
        }
        else if ((right == 0) && (((front_right == 1) && (left == 1)) || ((front_right
== 0) && (front_left == 1))))  //left
        {
                PORTA = FORWARD;
        //robot slightly to the right -> turn left slowly
                OCR3A = 4000;
                OCR3B = 24000;
                lcd_put_string("                ");
                lcd_goto_line(4);
                lcd_put_string("                ");
                lcd_goto_line(3);
        }
        else if ((left == 0) && (front_left == 0) && (front_right == 0) && (right
== 1))  //sharp right
        {
```

```
                        PORTA = FORWARD;
        //robot to the left -> turn right sharply
                        OCR3A = 50000;
                        OCR3B = 1000;
                        lcd_put_string("                ");
                        lcd_goto_line(4);
                        lcd_put_string("                ");
                        lcd_goto_line(3);
                }
                else if ((left == 1) && (front_left == 0) && (front_right == 0) && (right
== 0))  //sharp left
                {
                        PORTA = FORWARD;
        //robot to the right -> turn left sharply
                        OCR3A = 1000;
                        OCR3B = 50000;
                        lcd_put_string("                ");
                        lcd_goto_line(4);
                        lcd_put_string("                ");
                        lcd_goto_line(3);
                }
                else if ((left == 1) && (right == 1))
                {
                        intersections++;                                //if intersection has
been reached
                }
        } while (intersections != 1);           //loop contiuously until an intersection is
detected
}
//this function determines the color of an object
//it returns this color to the main program as a single character
char color_detect(void)
{
        char color = 'E', old_color;
        uint16_t red, green, blue;
        int rb, rg, bg, max_diff;

        do
        {
                old_color = color;

                PORTA = 0x80;
                ms_sleep(100);
                red = adc_readn(7,5);  /* sample channel 0 5 times, take average */
                ms_sleep(100);        /* sleep for 0.1 second */
```

```
		PORTA = 0x20;
		ms_sleep(100);
		green = adc_readn(7,5);  /* sample channel 0 5 times, take average */

		ms_sleep(100);		/* sleep for 0.1 second */

		PORTA = 0x40;
		ms_sleep(100);
		blue = adc_readn(7,5);  /* sample channel 0 5 times, take average */

		ms_sleep(100);		/* sleep for 0.1 second */

		PORTA = 0x00;


		if(red>blue)				//determines absolute difference
between red and blue
			rb = red - blue;
		else
			rb = blue - red;

		if(red>green)				//determines absolute difference
between red and green
			rg = red - green;
		else
			rg = green - red;

		if(blue>green)				//determines absolute difference
between green and blue
			bg = blue - green;
		else
			bg = green - blue;

		if((rb>=rg) && (rb>=bg))
			max_diff = rb;
		else if ((rg>=rb) && (rg>=bg))		//determines largest absolute
difference
			max_diff = rg;
		else
			max_diff = bg;

		if (max_diff < 45)			//if maximumdifference is less than
45 then object is green
			color = 'G';
		else if (blue < red)
			color = 'B';
```

```
                else if (red < green)
                        color = 'R';
                else
                        color = 'G';

        } while (old_color != color);           //will loop unti  l two straight readings return
the same result

        return color;
}
int main(void)
{
        uint16_t threshold;
        unsigned char bump, line;
        int time, n, prev_value, prev_value2, value, exit;
        char color_requested = 'R', location = 'R', color_detected;

        init_timer0();

        /* enable interrupts */
        sei();

        //Enable i/o ports
        DDRD = 0x07;
        DDRB = 0xFF;
        DDRC = 0x00;
        DDRA = 0xFF;

        //Enable pull-ups on port c
        PORTC = 0xF0;

        /* initialize A/D Converter */
        adc_init();

        ms_sleep(2048);

        lcd_initialize();
        lcd_clear_display();

        lcd_goto_line(2);
        lcd_put_string("   Calibrating IR");                    //calibration routine to
determine threshold
        do
        {
                threshold = adc_readn(0,5);
                bump = PINC;
```

```
        } while ((bump & 0x10) != 0);                                   //continuously
take readings until bumper is pressed

        lcd_clear_display();
        lcd_goto_line(2);
        lcd_put_string("Calibration Complete");
        ms_sleep(750);
        lcd_clear_display();

        //sets up PWM for motor and claw control
        TCCR3A = 0xA8;
        TCCR3B = 0x12;
        ICR3 = 65000;
        OCR3A = 25000;
        OCR3B = 25000;
        OCR3C = 1450;
        DDRE = 0x38;
        PORTE = 0x00;

        while(1)
        {
                prev_value = -1;
                prev_value2 = -2;
                exit = 0;

                do
                {
                        value = 0;
                        //takes in remote commands
                        do
                        {
                // wait for start bit
                                wait_for_low_to_high();
                                TCNT0 = 0;
                                ms_count = 0;
                                wait_for_low();
                                time = ms_count;
                        } while ((time < 2) || (time > 4));             // must be 2-4ms

                        for (n = 0 ; n < 4 ; n++)
                        {
                //get data bits
                                wait_for_low_to_high();
                                TCNT0 = 0;
                                ms_count = 0;
                                wait_for_low();
```

```
                                time = ms_count;
                                value = (value >> 1);
                                if(time > 0)                                    //if
time > 1ms bit was a 1 -> shift one into bit 3
                                        value += 8;
                        }
                        value++;

                        if
(((prev_value2==prev_value)&&(prev_value==value))&&((value==1)||(value==2)||(valu
e==3)||(value==4)||(value==5)||(value==6)||(value ==7)))
                                exit = 1;
        //causes continuous loop until three straight readings of a valid command are
recorded

                        prev_value2 = prev_value;
                        prev_value = value;

                } while (!exit);

                lcd_goto_line(1);

                //displays requested item to LCD and sets up variable to establish
location(refrigerator or pantry) and color
                if (value == 1)
                {
                        lcd_put_string("Coke            ");
                        lcd_goto_line(2);
                        lcd_put_string("                ");
                        color_requested = 'R';
                        location = 'R';
                }
                else if (value == 2)
                {
                        lcd_put_string("Pepsi           ");
                        lcd_goto_line(2);
                        lcd_put_string("                ");
                        color_requested = 'B';
                        location = 'R';
                }
                else if (value == 3)
                {
                        lcd_put_string("Mt. Dew         ");
                        lcd_goto_line(2);
                        lcd_put_string("                ");
                        color_requested = 'G';
```

```
                location = 'R';
        }
        else if (value == 4)
        {
                lcd_put_string("Original        ");
                lcd_goto_line(2);
                lcd_put_string("Pringles        ");
                color_requested = 'R';
                location = 'P';
        }
        else if (value == 5)
        {
                lcd_put_string("Chips Ahoy      ");
                lcd_goto_line(2);
                lcd_put_string("                ");
                color_requested = 'B';
                location = 'P';
        }
        else if (value == 6)
        {
                lcd_put_string("Sour Cream n' Onion ");
                lcd_goto_line(2);
                lcd_put_string("Pringles        ");
                color_requested = 'G';
                location = 'P';
        }
        else if (value == 7)                                    //touchdown
dance (three 360's)
        {
                lcd_put_string("Touchdown Gators!!!!");
                lcd_goto_line(2);
                lcd_put_string("                ");
                follow_line_for_time(900, threshold);


                for (int i = 0; i < 5; i++)
                {
                        OCR3A = 50000;
                        OCR3B = 50000;
                        PORTA = RIGHT;

                        do
                        {
                                line = (PINC & 0x06);
                        } while(line != 0x06);
```

```
                        ms_sleep(300);

                        do
                        {
                                line = (PINC & 0x06);
                        } while(line != 0x00);
                }

                goto end;
        }

        lcd_goto_line(3);

        follow_line_to_intersection(threshold);         //follows line to first
intersection (by couch)

        OCR3A = 25000;
        OCR3B = 25000;
        PORTA = FORWARD;

        do
        {
                line = PINC & 0x09;
        } while (line == 0);

        follow_line_to_intersection(threshold);         //follows line to
second intersection (middle of kitchen)

        PORTA = STOP;
        ms_sleep(500);

        //dependant upon snack or drink request turn towards refrigerator or pantry
        if (location == 'R')
        {
                OCR3A = 6000;
                OCR3B = 20000;
                PORTA = LEFT;
        }
        else if (location == 'P')
        {
                OCR3A = 20000;
                OCR3B = 6000;
                PORTA = RIGHT;
        }

        do
```

```
		{
			line = (PINC & 0x06);		//90 degree turn in specified
direction
		} while(line != 0x06);

		ms_sleep(300);

		do
		{
			line = (PINC & 0x06);
		} while(line != 0x00);


		PORTA = STOP;
		ms_sleep(200);

		follow_line_to_intersection(threshold);			//next
intersection makes beginning of three-way fork to each of the items

		PORTA = STOP;
		ms_sleep(100);

		OCR3C = 1825;						//open claw

		ms_sleep(500);

		OCR3A = 6000;
		OCR3B = 20000;
		PORTA = LEFT;					//inspect item to left
first

		do
		{
			line = (PINC & 0x06);
		} while(line != 0x06);

		ms_sleep(300);

		do
		{
			line = (PINC & 0x06);
		} while(line != 0x00);			//turn 90 degrees left

		PORTA = STOP;
		ms_sleep(100);
```

```
		OCR3A = 25000;
		OCR3B = 25000;
		PORTA = FORWARD;

		follow_line_to_intersection(2000);		//move to first object;

		PORTA = STOP;
		ms_sleep(100);

		OCR3C = 1350;							//close claw

		color_detected = color_detect();	//detect first color

		if (color_detected != color_requested)
		{
			OCR3C = 1825;
			ms_sleep(500);							//open
claw if incorrect color
		}

		OCR3A = 25000;
		OCR3B = 25000;
		PORTA = BACKWARD;

		ms_sleep(600);

		backward_to_intersection();		//back up back to the
ntersection

		PORTA = STOP;
		ms_sleep(100);

		if (color_detected == color_requested)		//first object was
correct.  turn back towards home
		{
			OCR3A = 1000;
			OCR3B = 40000;
			PORTA = FORWARD;

			do
			{
				line = (PINC & 0x06);
			} while(line != 0x06);

			ms_sleep(500);
```

```
                do
                {
                        line = (PINC & 0x06);
                } while(line != 0x00);                  //turn 90 degrees left
        }
        else
//inspect next object
        {
                OCR3A = 40000;
                OCR3B = 1000;
                PORTA = FORWARD;

                do
                {
                        line = (PINC & 0x06);
                } while(line != 0x06);

                ms_sleep(500);

                do
                {
                        line = (PINC & 0x06);
                } while(line != 0x00);                  //turn 90 degrees right

                PORTA = STOP;
                ms_sleep(100);

                OCR3A = 25000;
                OCR3B = 25000;
                PORTA = FORWARD;

                follow_line_to_intersection(2000);      //move to object

                PORTA =STOP;
                ms_sleep(100);

                OCR3C = 1350;
//close claw

                color_detected = color_detect();        //detect second color

                if (color_detected != color_requested)  //if incorrect color
                {
                        OCR3C = 1825;
                        ms_sleep(500);
//open claw
```

```
                    OCR3A = 25000;
                    OCR3B = 25000;
                    PORTA = BACKWARD;

                    ms_sleep(600);

                    backward_to_intersection();          //back up back to
intersection

                    PORTA = STOP;
                    ms_sleep(100);

                    OCR3A = 40000;
                    OCR3B = 1000;
                    PORTA = FORWARD;

                    do
                    {
                        line = (PINC & 0x06);
                    } while(line != 0x06);

                    ms_sleep(500);

                    do
                    {
                        line = (PINC & 0x06);
                    } while(line != 0x00);                    //turn 90
degrees right towards final object
                }
                else                                          //if second object was
correct color
                {
                    OCR3A = 20000;
                    OCR3B = 20000;
                    PORTA = RIGHT;

                    do
                    {
                        line = (PINC & 0x06);
                    } while(line != 0x06);

                    ms_sleep(1000);

                    do
                    {
```

```
                                line = (PINC & 0x06);
                        } while(line != 0x00);                  //180 degree
turn

                        follow_line_to_intersection(2000);      //follow line
back to intersection
                }

                if (color_detected != color_requested)                  //if
incorrect color
                {
                        PORTA = STOP;
                        ms_sleep(100);

                        OCR3A = 25000;
                        OCR3B = 25000;
                        PORTA = FORWARD;

                        follow_line_to_intersection(2000);              //move
to final object

                        PORTA = STOP;
                        ms_sleep(100);

                        OCR3C = 1350;
        //close claw
                        ms_sleep(500);
        //if incorrect color then desired item is not available

        //but don't want to return emptyhanded since user is still hungry/thirsty
                        OCR3A = 25000;
                        OCR3B = 25000;
                        PORTA = BACKWARD;
        // back up with object in tow

                        ms_sleep(600);

                        backward_to_intersection();

                        PORTA = STOP;
                        ms_sleep(100);

                        OCR3A = 40000;
                        OCR3B = 1000;
                        PORTA = FORWARD;
```

```
                        do
                        {
                                line = (PINC & 0x06);
                        } while(line != 0x06);

                        ms_sleep(500);

                        do
                        {
                                line = (PINC & 0x06);
                        } while(line != 0x00);                          //turn 90
degrees right   bck towards home
                }
        }

        PORTA = STOP;
        ms_sleep(100);

        OCR3A = 25000;
        OCR3B = 25000;
        PORTA = FORWARD;
        ms_sleep(500);

        follow_line_to_intersection(2000);              //forward to next
intersection (back to middle of kitchen)

        PORTA = STOP;
        ms_sleep(100);

        //turn right or left depending which side of the kitchen you are returning
from
        if (location == 'R')
        {
                OCR3A = 20000;
                OCR3B = 6000;
                PORTA = RIGHT;
        }
        else if (location == 'P')
        {
                OCR3A = 6000;
                OCR3B = 20000;
                PORTA = LEFT;
        }

        do
        {
```

```
            line = (PINC & 0x06);
        } while(line != 0x06);

        ms_sleep(500);

        do
        {
            line = (PINC & 0x06);
        } while(line != 0x00);                  //90 degree turn

        follow_line_to_intersection(2000);      //follow line to intersection
by couch

        PORTA = STOP;
        ms_sleep(100);

        OCR3C = 1825;                                   //open claw to release
item

        do
        {
            bump = PINC;
        } while ((bump & 0x10) != 0);

        PORTA = FORWARD;
        ms_sleep(500);

end:    follow_line_to_intersection(threshold);     //follow line back to home by
TV

        PORTA = STOP;
        ms_sleep(100);

        OCR3A = 20000;
        OCR3B = 20000;
        PORTA = RIGHT;

        do
        {
            line = (PINC & 0x06);
        } while(line != 0x06);

        ms_sleep(1000);

        do
        {
```

```
                line = (PINC & 0x06);
            } while(line != 0x00);                  //180 degree turn to point
back towards couch

            PORTA = STOP;
            ms_sleep(100);
        }

        return 1;

}
```