Student:	Andrew May
Teacher's Assistants:	Adam Barnett and Sara Keen
Teacher:	A. A. Arroyo
Associate Teacher:	E. M. Schwartz

ANT

FINAL REPORT

DATE: April 25, 2006

EEL5666: Intelligent Machine Design Lab

Department of Electrical and Computer Engineering

University of Florida

Introduction

What if you could not get at something? For example, your home was hit by a hurricane. As a result, all of your belongings are soaked. You want to retrieve your valuables, but you don't want mold spores in your lungs. Now, wouldn't it be great if you could tell an oversized ant to enter that hazardous area and retrieve those valuables for you?

Now you can - with Ant, your object retrieving robot.

Other potential situations you could rely on Ant for include passing through live weapon fire, accessing buildings that have been biologically contaminated, and reaching things at the other end of the table.

Although I see Ant going anywhere, strictly speaking, the goals of this project are contained in a lab environment. In a lab, the robot will pick up a pre-determined object and drop it off at a desired point. Issues include obstructions blocking the object and edges of the surface that Ant is on.

Ant is built around a microcontroller. The microcontroller is sandwiched between layers of sensors and actuators. On the lower layer, servos move Ant. Below the servos and at the top of the robot, sensors tell Ant what happened in the environment. To transfer signals between Ant's electronics, circuit boards were implemented. As a result of this hardware, software was written that created the behaviors of Ant.

See the appendix for instructions on getting started with programming and example software.

Illustrations are posted at <plaza.ufl.edu/amay>.

Integrated System

As shown in the following graphic, Fig.1, a microcontroller drives Ant's four sensors, LCD feedback, and two actuators.

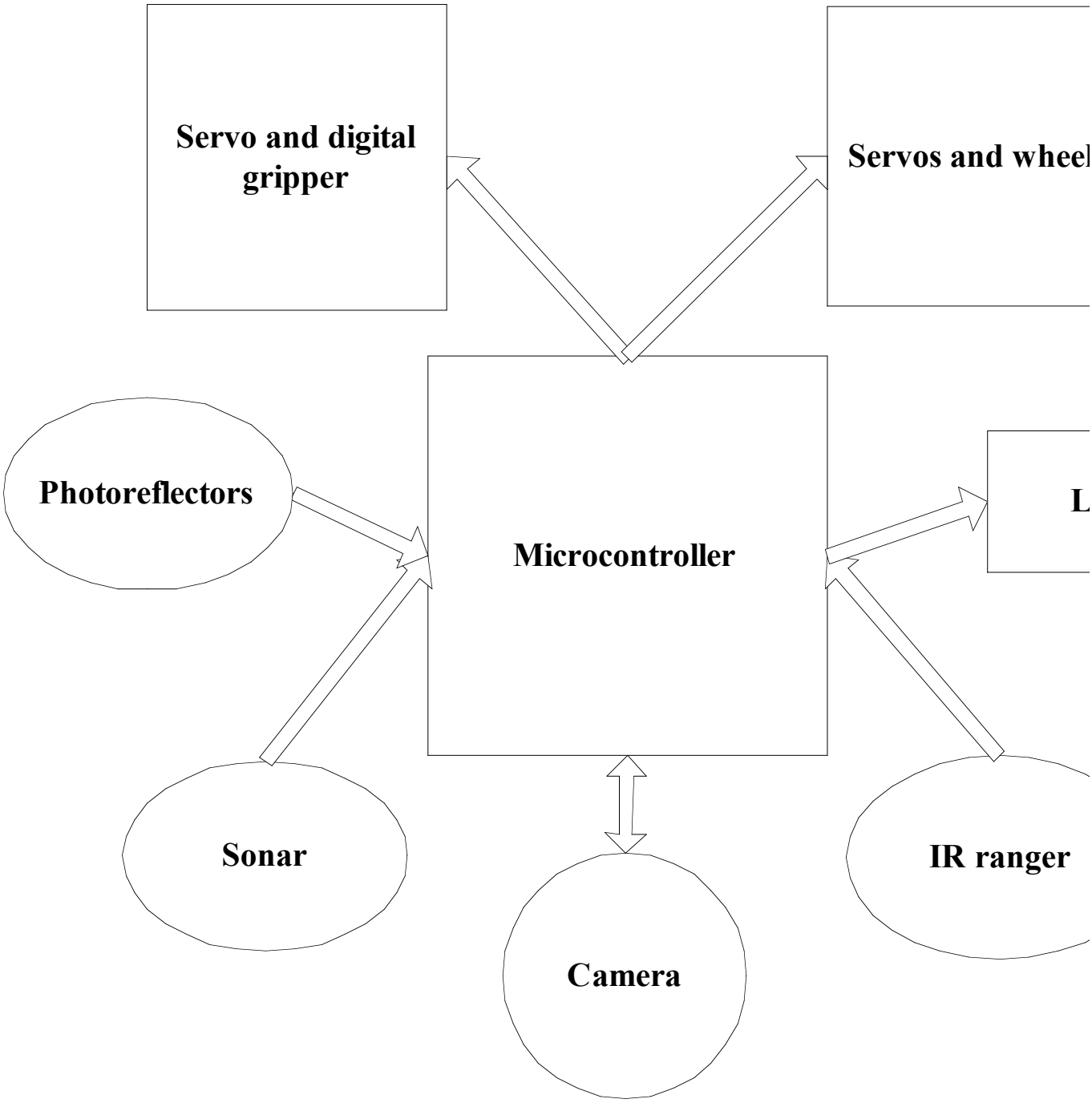


Fig. 1

Mobile Platform

My entire mobile platform was made out of wood and printed circuit board (PCB). Wood was more adaptable than plastic or metal. I added holes and cut into my mobile platform after the original design. PCB increased reliability by excluding, for example, breadboards and wire-wrap. I separated my mobile platform into an upper and lower shelf with PCB sandwiched in between to minimize its size.

The upper shelf raised my camera and infrared detector up to a height that provided enough light to the camera and enough distance to the infrared detector. On this same upper shelf, I put an LCD to see what Ant thought. To protect this level, I mounted two sonar rangers angled out at fifteen degrees. These sonar rangers allowed Ant to perform obstacle avoidance.

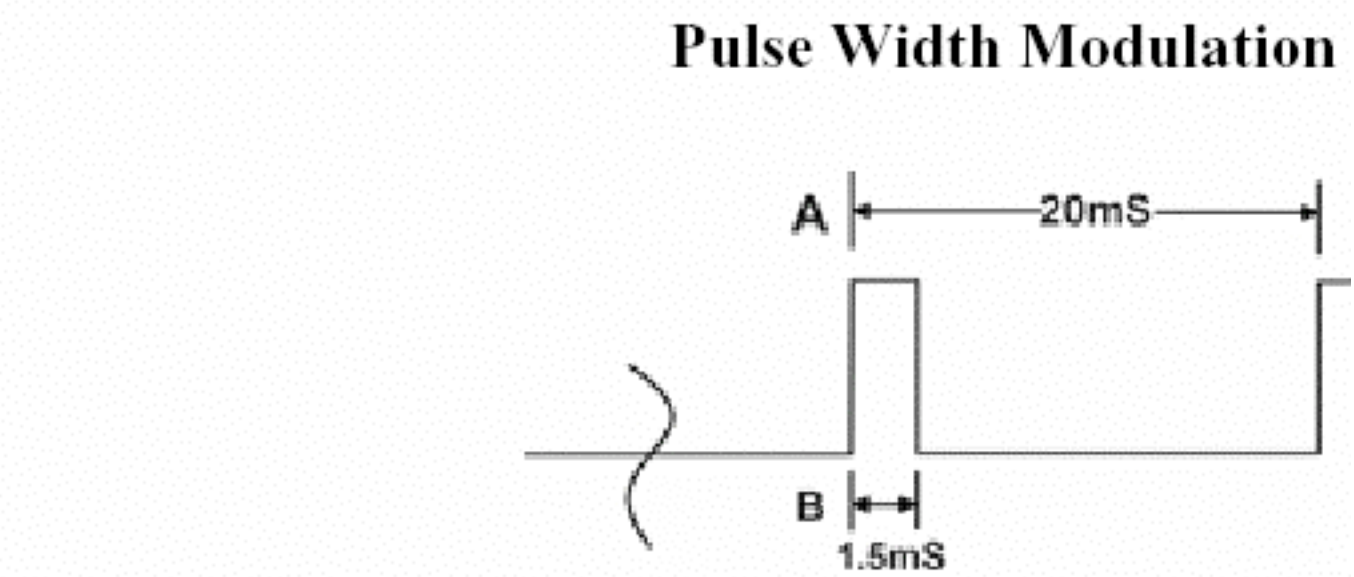
On the lower shelf, I interfaced locomotion, a gripper, a line-tracker module, and mounts for circuit boards. I screwed wheels to the very back to reduce the chance of Ant falling off an edge and improve how Ant responds to line-detection. I mounted a gripper low and in front to grab an object successfully. The line-tracker module was shifted with varying length screws and standoffs until the proper distance from the ground was tested and verified. The PCBs were stacked above the lower shelf to avoid debris that may roll

underneath the lower shelf.

I chose the length and width of the PCB I designed to match the PCB I already purchased to use fewer screws and other mounting parts. I added standoffs on top of the PCBs to allow me to move wires in between the PCBs and mobile platform shelves.

Actuation

I implemented two hacked servos to rotate the wheels and an un-hacked servo to operate the gripper. My servos behaved as Paul Dutka (Summer '05) stated: "Un-hacked servos provide movement to a position between 0º to a little over 180º. In hacking a servo, you simply remove the tab that drives the pot, and then adjust it so that it does not turn at the “zero” position (1.5 ms). This allows the servo to rotate a full 360º and control speed instead of position. The servo circuitry uses a PWM signal to control the servo motor. The signal has a 20ms period with a pulse width from 1ms to 2ms to control the servos’ position / speed (See Fig. 2).



	1.0 ms	1.5 ms
Un-hacked	Left – 90	Center – 0
Hacked	Backwards – Full Speed	Still

Fig. 2

This signal is used to provide the desired speed / position. The farther the position, the faster the servo turns to achieve it." I coded a minimal shift in speed algorithm to preserve the mechanics in these hacked servos resulting in smoother acceleration. One problem with these servos is that one servo pulled when the batteries lost power. To correct this pulling, I wrote software to move one servo slower than the other.

The un-hacked servo closed to a pre-determined angle. I experimented to determine the exact pulse length to close the gripper properly. I used what many including Dr. Schwartz call a divide-and-conquer approach. I started with a value below and above the desired but unknown value. If the desired behavior was closer to the lower

value, then I tested in the lower-half of the previous range and vice-versa. I repeated this process until I narrowed the values down to a single one - the desired value.

Sensors

I chose sensors that interacted with Ant's environment enough to achieve Ant's mission.

1) Infrared (IR) Detector

I chose the Sharp DP2Y0A02YK. This model of IR detector begins reading at eight inches (Instead of four inches in most other Sharp IR detectors) but goes beyond the distance of a basic GP2D12. An analog signal comes from this IR detector that increases when distance decreases.

I used an IR detector to detect an approaching edge or obstacle in front of the robot. To account for the eight inch minimum distance from the IR detector, I mounted the IR detector six inches high and at an angle. An object was detected when the analog output from the IR detector increased. An approaching edge was detected when the analog output from the IR detector decreased significantly.

The choice of which IR detector suits you is complex. I recommend that you consult <http://www.acroname.com/robotics/info/articles/sharp/sharp.html>.

2) Photoreflector

I chose the Hamamatsu P5587 Photoreflector. This model of photoreflector or digital infrared (DIR) detector sets the output low when over a black line. A circuit to drive this sensor was pictured at the supplier's website.

I used DIR detectors to differentiate between white poster board and black electrical tape. I screwed the PCB with two DIR detectors about 0.25 inches from the table pointing straight down. I used only two DIR detectors for line-acquiring and line-tracking on straight ways and around curves.

3) Sonar Ranger

I chose the Devantech SRF05 sonar ranger. The SRF05 has an LED to indicate when it sends a pulse. To operate my sonar ranger, I sent a pulse to a trigger line for 48 microseconds. Next, I waited until the echo line went high. After that, I counted the number of 48 microseconds until the echo line went low. Finally, I added two to the total number of 48 microseconds counted and divided by 2.8 to get the approximate distance between the sonar ranger and the object in inches. I used two sonar rangers. I pulsed one 65ms out of phase of the other to preserve the echo received. I set the software to avoid distances less than one and a half body lengths. One key advantage in sonar rangers over IR detectors is close range operation. IR detectors can not continue reading under four or eight inches.

One sensor that I never tested was the one from Maxbotix. If you can, I would recommend experimenting with this sensor because to see if it gives analog values of the distance as its suppliers claim.

4) Camera (Special Sensor)

I chose the Carnegie Mellon University Camera One (CMUCam1 or just CMUCam). The CMUCam2 was too much for the function that I desired.

I instructed the camera to recognize the color of Nutri-Grain bar (Red or Not-Red). I placed the camera above the jaws so that it would be as close to the bar as possible. This close proximity reduced the chance of reading colors that appeared to be the same as the object that Ant was seeking but were not. The camera was mounted with screws that were on a couple right angle connectors. The camera was pointed downward at about a 45 degree angle. The software was written in large part because of Fernando Hernandez (Summer '05). However, I modified Mr. Hernandez's code to exploit the Get

Mean (GM) function instead of the Track Color (TC) function. GM was more accurate than TC because I was stopping Ant at a definite point. The lighting conditions greatly affected this sensor. I tested my robot in the environment of demonstration and media days to prevent accidents (This testing can be extrapolated to be called alpha and beta user testing - one of the benefits of this class). I turned the camera on when I was in position to see a color only.

Circuit Boards

I made two circuit boards to drive Ant. One circuit board supplies power, the other board line-tracks.

1) Power portal

I poured most of Ant's energy through a central PCB. The power board has multiple ports to match the differing components that the power board runs. One 5V regulator (7805) spins Ant's servos only. Servos tend to jitter more than any other part and can put electronics running from the same supply into a brownout. Another 7805 drives up to five IR detectors. The third, a 7805 as well, joined a potentiometer to power and adjust the contrast on an LCD. Uniquely, a 6V regulator powers the CMUCam because the CMUCam has a regulator already. In addition, I included a terminal to connect the CMUCam to the UART from the microcontroller. As a result, I could replace the LCD on the robot for a laptop that generated more information from the CMUCam if I wanted to. The microcontroller is powered off the battery because it also has its own regulator. From the microcontroller, the sonar rangers and DIR detectors are supplied 5V. A power switch allowed the robot to turn off when I needed the robot to stop. On power up, the program in flash memory ran from the beginning automatically. For this board, I used traces of 20 mils to prevent the traces from breaking during soldering or heat during operation. The screw holes were of 125 mils.

2) Line-tracker

Custom was at the heart of this design. I followed the sensor data sheet exactly. I

made footprints in Protel '99 (An older version of Altium DXP). I laid out the sensors 0.5 inches apart in the center and one inch between the inner and outer sensors to capture the electrical tape correctly.

Behaviors

1) Edge of the world detection

By edge of the world, I mean where the ground that Ant started on ended. As I stated previously, when the light hitting the IR detector decreased beyond an experimentally determined value then there was an edge approaching.

2) Line tracking

Ant line-tracked similar to how ants follow pheromone trails. Ant reached the object because of the line-tracking behavior that I incorporated. When my robot sensed the black line for the first time, it turned until it acquired the line. While on the line, the robot turned to the right if the left sensor read the line and vice-versa. I added more speed when turning depending on which direction Ant went on a curve.

3) Obstacle avoidance

Detection of obstacles was from IR detection and sonar ranging. Ant detects an increase in infrared, double checks that increase, and turns if an increase in infrared was sustained to eliminate turning from camera flashes, for example. Next, interleaved with checking for a line, sonar ranges are checked for a minimum distance. When that minimum distance is breeched Ant goes away from that obstacle.

4) Color detection

Generated by the CMUCam, I took mean red, green, and blue values. From these color values, Ant grabbed a red object, avoided a blue object, and wondered what

happened if there was no object apparently. Thinking about the color required a delay of a second, because the camera showed false color values while in motion. Variance in ambient light dictated a range of color values around 60. Unlike I specified in my special sensor report, the track color (TC) function was neglected because I was using the confidence value only. The TC function is better for following a color that is moving. TC can move a robot to center on a color too, but not as well. I cut costs by using GM instead of TC because I didn't need white LEDs consequently.

Experimental Layout and Results

1) IR detector

I hooked up an LCD to see what the analog value from the IR detector was. I intended to record the values in the environment that the final user would be in. Based on the LCD displays, I wrote software to act on the analog values received. During this research, I repeated scans of the same place. I saw variance in the data of 10%. A large issue was that the IR detector thought an edge was ahead when there was a black line ahead actually. I fixed this slowdown by recording values while looking perpendicular to, in parallel with, and at a 45 degree angle to the black line numbering six. Then, I wrote an if statement to ignore all IR values around the range of these six but still see an edge.

2) Sonar ranger

With an LCD and a JTAG in circuit emulator (ICE) programmer, I looked at the calculated distances. Then, I turned Ant to see how much distance Ant needed to clear an obstacle. Last, I chose a large enough distance and added avoidance into the software.

3) DIR detector

I checked if any of the four original ones worked. The right most one didn't because I melted the copper off the board when I soldered it. Consequently, I based the line-following behavior on the two center DIR detectors primarily. I modified the software to wait in a while loop until the sensors showed the robot was centered on the

line. These experiments taught me that I should try to be as random as possible so that my robot can adjust to any scenario that it enters.

4) CMUCam

It was crucial that I conduct all experiments with this sensor in the environment of the final user, because this color reader changes from ambient light drastically. Another source of noise for this sensor was if the sensor was in motion. I had 100% error in values that were displayed while - and less than 50 ms after - Ant was moving. I read mean color values of ambient light to assure the color values of the objects desired differed. I read color values arbitrarily, but the only ones that mattered in my case were those colors shown at the moment the camera almost touched the object. This certain moment is an artifice that is impossible in real life. I reran sections of the final software to expand the range of color values that I based a "red," "blue," or "no" object on. I checked the colors to see if ambient light looked the same as red or blue, but it did not. See my special sensor report for more details.

Conclusion

Ant is not ready to go into destroyed buildings and retrieve whatever you tell it to yet. However, Ant has performed obstacle avoidance, edge detection, line-following, and color recognition. I modified circuit boards and software to drive Ant too. Ant is limited to flat surfaces, seeing the desired object before retrieving it, and avoiding obstacles that are taller than it. My photoreflector and power boards kept running long after I expected them to. The one technical caveat that I pass on is that sometimes a problem in your robot is not as bad as you think it is. For example, I spent hours running the robot to try to fix the servos, but I did not realize my problem was that my batteries were low.

In the future, if I started Ant over, I would not short out LCDs, buy as fast as I did, or neglect the assistance of software. To enhance Ant, I would make the gripper omni-directional, use the track color function with white LEDs to find objects randomly, replace servos with motors, place sonar rangers close to the ground, enclose the wires and battery pack with mechanical fasteners, and add a voice chip (Perhaps you can use a voice chip as a special sensor). I would also specify that Ant have four edge of the world IR detectors at the corners of the robot's mobile platform to allow Ant to get closer to the edges safely.

I considered Ant an investment. In my opinion, little time or money that I spent

on this robot was wasted.

Documentation

CMUCam User's Manual v2.00

Anthony Rowe and Carnegie Mellon University, edited by Charles Rosenberg and Illah Nourbakhsh

< http://www-2.cs.cmu.edu/~cmucam/Downloads/CMUcamManual.pdf >

SRF04 Ultrasonic Ranger Technical Specification

< http://info.hobbyengineering.com/specs/devantech-srf04-tech.pdf >

Hamamatsu Photoreflector Data sheet

< http://www.acroname.com/robotics/parts/R64-P5587.pdf >

MAVRIC-IIB Manual

< http://www.bdmicro.com/images/mavric-iib.pdf >

Atmega128 Complete Datasheet

< http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf >

Thanks to:

Fernando Hernandez (Su' of '05) for his CMUCam documentation.

Matthew Yoder (Sp' of '04) for his circuit board diagrams.

Matt Moore for his PCB milling.

Ken (peer) for his approach to SRF05 programming.

Foley (peer) for his logistical help.

Drew (peer) for his website assistance.

Appendix

To remedy the lack of information on how to program a robot, I will discuss where to download programming tools and methods to program a robot based on an Atmega128 microcontroller that uses a USB-to-serial driven JTAG ICE II programmer.

1) Hardware

Choose programmer: I used a USB-to-serial JTAG ICE programmer because it eases debugging, however ISPs are available too. Navigate the maze of download links to find the software drivers for the hardware that you choose.

2) Downloads

To write and compile C code, download the latest version (v0.6.1) of Programmer's Notepad from <http://www.pnotepad.org/download/>.

To program the flash memory of the Atmega128 microcontroller, download the latest version (4.12 (build 460)) of AVR Studio by clicking on the tiny CD icon. To do less adaptable but cheaper and more popular serial or parallel programming, download the latest version of PonyProg2000 from the latest link (v2.06f BETA) at <http://www.lancos.com/ppwin95.html> instead of AVR Studio.

To have a makefile and starting examples of code (In addition to the examples of

former IMDLers), download and unzip at least one sample, hello world, from <http://www.bdmicro.com/code/hw/>.

3) Configuration

In Programmer's Notepad, open the hw.c file. From the Tools tab, click Make All. As a result, you should see file called hw.hex (In this case the hex file was provided already). This is the file that the programmer needs.

In AVR Studio of PonyProg, open the hw.hex file.

If you have AVR Studio, your fuse bits and other settings are set automatically. If you have PonyProg, you must set refer to William Dubel's tutorial on the website to set the programmer manually. Turn off verification to save yourself time.

In the programmer, run the hw.hex file (This should be from flash). When you turn the power off and back on, the program will run from its beginning.

Resave hw.c as main.c.

Change the makefile. Open the Makefile file in Programmer's Notepad. Change all instances of "hw" to "main". Delete "#CPU = at90can128". From now on, open a new folder for a new program, copy and paste a former main.c file and Makefile file into that folder. Redo Make All from the tools tab to produce the main.hex file for that program. Open this different main.hex file in the programming software.

In this way, you will always have the software you need to run your program. However, there will be no way to add libraries into your code. Libraries are supplemental files with ".h" and ".c" extensions that can clean and organize your code better. In general, approach a new program with a minimalist approach. Go online and look at robots what did what are trying to do and copy the code exactly. After that, learn what the

code does and modify it to suit your needs.

4) Help?

If you have any questions, corrections, or statements that hide a question, feel free to contact me by email at <amay@ufl.edu>.

Feel free to visit <plaza.ufl.edu/amay> for illustrations, other documentation, and weekly reports.

Software:

```
/*
** Title:            Autonomous Machine Software
** Purpose:          Decide how a robot will react to its environment
** Author:           Andrew May
** Date of Creation: 04-22-06
** Date of Revision: 04-25-06
*/

/*Standard C libraries*/
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>
#include <avr/pgmspace.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

/*Constant definitions*/

/*Servos*/
#define SERVO3 OCR3C
#define SERVO5 OCR1B
#define SERVO6 OCR1C

#define GRIPPER SERVO3
#define L_WHEEL SERVO6
#define R_WHEEL SERVO5

#define GRIPPER_MID 2850
#define WHEEL_MID 2850

/*A-to-D*/
#define DDRAD DDRF
#define PORTAD PORTF
```

```
/*Sonar rangers*/
#define DDRSRF DDRD
#define PORTSRF PORTD
#define PINSRF PIND

#define L_TRIGGER 0x02
#define L_ECHO 0x01
#define R_TRIGGER 0x10
#define R_ECHO 0x08
#define C_TRIGGER 0x80
#define C_ECHO 0x40

/*Photoreflectors*/
#define PINDIR PINA //PORTA holds Digital IR
#define DDRDIR DDRA
#define PORTDIR PORTA

/*
** LCD definitions:
** PORTC BIT
**  7   6   5   4   3   2   1   0
** DB7 DB6 DB5 DB4  E   RS  NC  NC
*/

#define DDRLCD DDRC
#define PORTLCD PORTC

#define ENABLE 0x08
#define RS 0x04

/*Global variables*/
/*Timer*/
volatile uint16_t us_48_count;
volatile uint16_t ms_count;

/*CMUCam & UART*/
volatile int MAX_MSG_SIZE = 30;
volatile unsigned char CMUResponseBuffer[15];

/*Delay functions*/
void us_48_sleep(uint16_t us_48)
{
      TCNT0 = 0;
      us_48_count = 0;
      while (us_48_count != us_48) // Each loop takes 48 us NOT 1 us,
based on the 128 divisor I set in prescalers in function declared
timer_init()
            ;
}

void ms_sleep(uint16_t ms)
{
      TCNT0 = 0;
      ms_count = 0;
      while (ms_count != ms*21) // Each while loop takes 48 us, but
total function takes 21 times that = 21*48*ms
```

```
            ;
}

SIGNAL(SIG_OUTPUT_COMPARE0)
{
      us_48_count++;
      ms_count++;
}

void timer_init(void)
{
      /*
      ** Initialize timer0 to generate an output compare interrupt, and
      ** set the output compare register so that we get that interrupt
      ** every 48 microseconds.
      */
      TIFR  |= _BV(OCIE0);
      TCCR0  = _BV(WGM01)|_BV(CS02)|_BV(CS00); // CTC, prescale = 128
      TCNT0  = 0;
      TIMSK |= _BV(OCIE0);    // Enable output compare interrupt
      OCR0   = 6;          // Match in 48 us
}

/*
** SIG_OVERFLOW1 - the timer 1 output compare function automatically
ends
** the pulse precisely as specified by the OCR1x register which
** represents the servo position
*/
SIGNAL(SIG_OVERFLOW1)
{
      TCNT1 = 0;

      /*
      ** Configure to set outputs on compare match so we can turn on
the
      ** pulse in the next statement
      */
      TCCR1A |=
_BV(COM1A1)|_BV(COM1A0)|_BV(COM1B1)|_BV(COM1B0)|_BV(COM1C1)|_BV(COM1C0)
;

      /* force compare match to set outputs */
      TCCR1C |= _BV(FOC1A)|_BV(FOC1B)|_BV(FOC1C);

      /* configure to clear outputs on compare match so that the output
      * compare function ends the pulse */
      TCCR1A &= ~(_BV(COM1A0)|_BV(COM1B0)|_BV(COM1C0));
}

/*For gripper*/
/* SIG_OVERFLOW3 - this interrupt handler starts the pulse for servos
** 1, 2, & 3; the timer 3 output compare function automatically ends
** the pulse precisely as specified by the OCR3x register which
** represents the servo position
*/
SIGNAL(SIG_OVERFLOW3)
```

```
{
	TCNT3 = 0;

	/*
	** Configure to set outputs on compare match so we can turn on
the
	** pulse in the next statement
	*/
	TCCR3A |=
_BV(COM3A1)|_BV(COM3A0)|_BV(COM3B1)|_BV(COM3B0)|_BV(COM3C1)|_BV(COM3C0)
;

	/*Force compare match to set outputs*/
	TCCR3C |= _BV(FOC3A)|_BV(FOC3B)|_BV(FOC3C);

	/*
	** Configure to clear outputs on compare match so that the output
	** compare function ends the pulse
	*/
	TCCR3A &= ~(_BV(COM3A0)|_BV(COM3B0)|_BV(COM3C0));
}

void servo_init(void)
{
	/*
	** Use Timers 1 and 3 to generate the pulses for 6 R/C servos;
each
	** timer can do up to 3 servos.
	*/

	/*
	** Configure OC3A for mode 0: normal, top=0xffff prescale=8
(f~=30):
	**
	** WGM33=0, WGM23=0, WGM13=0, WGM03=0, CS32=0, CS31=1, CS30=0
	*/
	DDRE |= _BV(PORTE3) | _BV(PORTE4) | _BV(PORTE5);
	TCCR3A &= ~(_BV(WGM31) | _BV(WGM30) | _BV(COM3A1) | _BV(COM3B1) |
_BV(COM3C1));
	TCCR3A |= _BV(COM3A0) | _BV(COM3B0) | _BV(COM3C0);
	TCCR3B &= ~(_BV(WGM33) | _BV(WGM32) | _BV(CS32) | _BV(CS30));
	TCCR3B |= _BV(CS31);
	TCNT3 = 0;
	TCCR3C |= _BV(FOC3A) | _BV(FOC3B) | _BV(FOC3C);
	ETIMSK |= _BV(TOIE3);

	/*
	** Configure OC1A for mode 0: normal, top=0xffff prescale=8
(f~=30):
	**
	** WGM33=0, WGM23=0, WGM13=0, WGM03=0, CS32=0, CS31=1, CS30=0
	*/
	DDRB |= _BV(PORTB5) | _BV(PORTB6) | _BV(PORTB7);
	TCCR1A &= ~(_BV(WGM11) | _BV(WGM10) | _BV(COM1A1) | _BV(COM1B1) |
_BV(COM1C1));
	TCCR1A |= _BV(COM1A0) | _BV(COM1B0) | _BV(COM1C0);
	TCCR1B &= ~(_BV(WGM13) | _BV(WGM12) | _BV(CS12) | _BV(CS10));
```

```
	TCCR1B |= _BV(CS11);
	TCNT1 = 0;
	TCCR1C |= _BV(FOC1A) | _BV(FOC1B) | _BV(FOC1C);
	TIMSK |= _BV(TOIE1);

	GRIPPER = 3800; // Open gripper
	L_WHEEL = WHEEL_MID; // Neutralize wheels
	R_WHEEL = WHEEL_MID;
}

/*Gripper functions*/
void open(void)
{
	GRIPPER = 3800;
}

void close(void)
{
	GRIPPER = 1700;
}

void grab(void)
{
	GRIPPER = 2550;
}

/*A-to-D functions*/
void ad_init(void)
{
	DDRAD = 0; // Set AD data direction register (DDRF) for input
	PORTAD = 0;

	ADMUX = _BV(REFS0); // Select AVCC (internal) for AD voltage
source, Only 8 bit A to D accuracy, and channel 0 (PORTF0) only

	ms_sleep(16); // Wait for power up

	ADCSRA = ~(_BV(ADIE)); // Set AD: Enable, Start a dummy
conversion, Set AD to run freely, No AD interrupt, Clear Flag, Divide
clock by 128 (16 MHz / 128 = 125 kHz)
}

uint16_t ad_read(void)
{
	return ADC;
}

void ad_chsel(uint8_t channel)
{
	/*Select channel*/
	ADMUX = (ADMUX & 0xe0) | (channel & 0x07);
}

/*
** ad_readn() - A/D Converter, read multiple times and average
**
** Read the specified A/D channel 'n' times and return the average of
```

```
** the samples
*/
uint16_t ad_readn(uint8_t channel, uint8_t n)
{
	uint16_t t;
	uint8_t i;

	ad_chsel(channel);
	t = ad_read(); // Dummy read

	/*Sample selected channel n times, take the average*/
	t = 0;
	for (i=0; i<n; i++)
	{
		ms_sleep(1);
		t += ad_read();
	}

	/*Return the average of n samples*/
	return t / n;
}

/*Sonar ranger functions*/
void srf_init(void)
{
	DDRSRF = 0x92; // PORTD bit 1, 4, and 7 are triggers & bit 0, 3,
and 6 are echos
	PORTSRF = 0;
}

uint8_t l_sonar(void)
{
	uint16_t counter = 0; // Count

	PORTSRF = L_TRIGGER; // Trigger SRF
	us_48_sleep(1); // Hold trigger high for 48 microseconds
	PORTSRF &= ~L_TRIGGER;

	while (!(PINSRF & L_ECHO))
	{
		; // Wait until echo line rises
	}

	while (PINSRF & L_ECHO)
	{
		counter++;
		us_48_sleep(1); // Count instances of 48 microseconds until
echo line falls
	}

	return ((counter+2)/2.8); // Distance in inches
}

uint8_t r_sonar(void)
{
	uint8_t counter = 0; // Count
```

```
	PORTSRF = R_TRIGGER; // Trigger SRF
	us_48_sleep(1); // Hold trigger high for 48 microseconds
	PORTSRF &= ~R_TRIGGER;

	while (!(PINSRF & R_ECHO))
	{
		;
	}

	while (PINSRF & R_ECHO)
	{
		counter++;
		us_48_sleep(1);
	}

	return ((counter+2)/2.8); // Distance in inches
}
uint8_t c_sonar(void)
{
	uint8_t counter = 0; // Count

	PORTSRF = C_TRIGGER; // Trigger SRF
	us_48_sleep(1); // Hold trigger high for 48 microseconds
	PORTSRF &= ~C_TRIGGER;

	while (!(PINSRF & C_ECHO))
	{
		;
	}

	while (PINSRF & C_ECHO)
	{
		counter++;
		us_48_sleep(1);
	}

	return ((counter+2)/2.8); // Distance in inches
}

/*This function was not tested*/
/*uint8_t avg_c_sonar(void)
{
	uint16_t total_counter = 0;
	int i;

	for (i=0;i<4;i++)
	{
	ms_sleep(50);
	uint16_t counter = 0; // Count

	PORTSRF = C_TRIGGER; // Trigger SRF
	us_48_sleep(1); // Hold trigger high for 48 microseconds
	PORTSRF &= ~C_TRIGGER;

	while (!(PINSRF & C_ECHO))
	{
		;
```

```
	}

	while (PINSRF & C_ECHO)
	{
		counter++;
		us_48_sleep(1);
	}

	total_counter += counter;
	}

	total_counter = total_counter/4;

	return ((total_counter+2)/2.8); // distance in inches
}
*/

/*UART functions (For CMUCam)*/
/*Initialize UART0 to 115.2k baud*/
void uart0_init(void)
{
	UBRR0H = 0x00;
	UBRR0L = 16; //16 is for 115.2k if U2X=1, 0x33 = 51 for 38.4k
baud
	UCSR0A = 0x02;
	UCSR0B = 0x18; //Bit 4 is Rx enable, Bit 3 is Tx enable
	UCSR0C = 0x06; //Bit 6 = Mode (0=Ascync, 1=Sync),
}

/*Transmit a message over UART0 in the form of a character array*/
void UART0_TX(char message[MAX_MSG_SIZE])
{
	int t = 0;
	while ((t < (MAX_MSG_SIZE + 1)) & (message[t] != 0x00))
	{
		/*Wait for an empty transmit buffer*/
		while ((UCSR0A & _BV(UDRE0)) == 0)
			;
		UDR0 = message[t];
		t++;
	}
}

/*Receive a message*/
unsigned char UART0_RX(void)
{
	while(!(UCSR0A & (1<<RXC0)))
		;
	return UDR0;
}

/*CMUCam functions*/
void cmu_init(void)
{
	/*Reset*/
	UART0_TX("RS\r");
	ms_sleep(20);
```

```
	/*Poll Mode*/
	UART0_TX("PM 1\r");
	ms_sleep(20);
	/*Raw Output*/
	UART0_TX("RM 3\r");
	ms_sleep(20);
	/*Middle Mass On*/
	UART0_TX("MM 1\r");
	ms_sleep(20);
	/*Track with the full window*/
	UART0_TX("SW 1 1 80 143 \r");
	ms_sleep(20);
}

/*
** Get the mean (average) values of red, green, and blue (R, G, B)
** and store them in the global "CMUResponseBuffer"
*/
void CMU_GM(void)
{
	int i = 0;
	char tempChar;

	UART0_TX("GM\r");

	/*Read and discard the first 255 framing byte*/
	tempChar = UART0_RX();

	/*Read 7 byte long "type S" packet*/
	for(i=0;i<7;i++)
	{
		CMUResponseBuffer[i] = UART0_RX();
	}

	/*Trash the last 255 framing byte*/
	while(tempChar != ':')
	{
		tempChar = UART0_RX();
	}

	CMUResponseBuffer[i] = '\0';
}

/*Tracks a color*/
void CMU_TC(int Rmin, int Rmax, int Gmin, int Gmax, int Bmin, int Bmax)
{
	int i = 0;
	char tempChar, tempMessage[30];

	sprintf(tempMessage,"TC %i %i %i %i %i %i\r", Rmin, Rmax, Gmin,
Gmax, Bmin, Bmax);

	UART0_TX(tempMessage);

	tempChar = UART0_RX();

	/*Return a 9 byte long "type M" packet*/
```

```
	for(i=0;i<9;i++)
	{
		CMUResponseBuffer[i] = UART0_RX();
	}

	while(tempChar!= ':')
	{
		tempChar = UART0_RX();
	}

	CMUResponseBuffer[i] = '\0';
}

/*Convert raw CMUCam data into an integer*/
int binary2int(unsigned char binary_num)
{
	int result = 0;
	if(binary_num & 1)
		result +=1;
	if(binary_num & 2)
		result +=2;
	if(binary_num & 4)
		result +=4;
	if(binary_num & 8)
		result +=8;
	if(binary_num & 16)
		result +=16;
	if(binary_num & 32)
		result +=32;
	if(binary_num & 64)
		result +=64;
	if(binary_num & 128)
		result +=128;
	return result;
}

/*This function was not tested*/
/*uint8_t Read_Color(void)
{
	uint8_t i;
	uint8_t value = 0;

	for (i=0; i<4; i++)
	{
		ms_sleep(125);
		CMU_TC(115, 150, 15, 150, 17, 35);

		value = value + binary2int(CMUResponseBuffer[8]);
	}

	value = value/4;

	return value;
}
*/

/*Servo functions*/
```

```
/*Declarations*/
int L_new_speed = 0, R_new_speed = 0, L_old_speed = 0, R_old_speed = 0;
int increment = 1;

/*Increment the wheel speed as slowly as possible to preserve the
servo*/
void wheels(int L_desired_speed, int R_desired_speed)
{
      L_desired_speed *= 5;
      R_desired_speed *= 5; //Calibration

      while((L_new_speed != L_desired_speed) || (R_new_speed !=
R_desired_speed))
      {
            if (L_desired_speed > L_old_speed)
            {
                  L_new_speed = L_old_speed + increment;
            }
            else if (L_desired_speed < L_old_speed)
            {
                  L_new_speed = L_old_speed - increment;
            }

            if (R_desired_speed > R_old_speed)
            {
                  R_new_speed = R_old_speed + increment;
            }
            else if (R_desired_speed < R_old_speed)
            {
                  R_new_speed = R_old_speed - increment;
            }

            L_WHEEL = WHEEL_MID + L_new_speed; // L wheel adjustment
            R_WHEEL = WHEEL_MID + R_new_speed; // R wheel adjustment

            L_old_speed = L_new_speed;
            R_old_speed = R_new_speed;

            ms_sleep(3);
      }
}

/*Wheels functions*/
void stop(void)
{
      wheels(0,0);
}

void forward(int speed)
{
      wheels(speed*0.91, -speed); // Servos have to correct drift
(Hence, the "...*0.91")
}

void R_turn(int speed)
{
      wheels(speed, speed);
```

```
}

void L_turn(int speed)
{
      wheels(-speed, -speed);
}

void reverse(int speed)
{
      wheels(-speed, speed);
}

/*
** I use this function, but I am not sure if it works.
** I only saw right turns from this function so far.
*/
void rand_turn(int speed)
{
      if (TCNT0 & 1)
            speed = -speed;
      else ;

      wheels(speed, speed);
}

void dir_init(void)
{
      DDRDIR = 0; // Read off the DIR port
}

/*A delay to allow the servos to be centered (If needed)*/
void center(void)
{
      int i;
      for (i=0;i<8;i++)
      {
            PORTB ^= 0x01;
            ms_sleep(512);
      }
}

/*I don't use this. It turns off some sensors.*/
volatile int go_to_line = 0;

void deinit_IR_sonar(void)
{
      ADMUX &= ~(_BV(REFS0)); // Deselect AVCC (internal) for AD
voltage source, Only 8 bit A to D accuracy, and channel 0 (PORTF0) only
      ADCSRA = 0; // Turn off ADC

      DDRSRF = 0; // Disable trigger
      PORTSRF = 0;
      go_to_line = 1;
}

/*LCD Functions*/
void latch(void)
```

```
{
	PORTLCD |= ENABLE;
	PORTLCD &= ~ENABLE; // Latch data (strobe)
}

void lcd_command_nibble(uint8_t nibble)
{
	PORTLCD = nibble;
	latch(); // Send
}

void lcd_command(uint8_t byte)
{
	uint8_t temp = byte; // Save command
	temp &= 0xF0; // Peel off the four MSBs (Also, to make RS = 0)
	PORTLCD = temp;
	latch();
	ms_sleep(5); // Wait for LCD data to go
	temp = byte;
	temp <<= 4; // Shift the four LSBs up to the MSBs (Or, sending
position)
	temp &= 0xF0;
	PORTLCD = temp; // Load LSBs
	latch();
	us_48_sleep(1);
}

void lcd_clear_screen(void)
{
	/*Clear screen; cursor home*/
	ms_sleep(1);
	lcd_command(0x01);
	ms_sleep(2);
}

void lcd_init(void)
{
	DDRLCD = 0b11111100;
	PORTLCD = 0;

	ms_sleep(20); // For more than 15000 us = 15 ms

	/*Normal lcd initializations (For 4-bit mode)*/
	lcd_command_nibble(0x30);
	ms_sleep(5);
	lcd_command_nibble(0x30);
	us_48_sleep(3);
	lcd_command_nibble(0x30);
	ms_sleep(5);
	lcd_command_nibble(0x20);
	us_48_sleep(1);

	/*Two lines*/
	lcd_command(0x28);
	us_48_sleep(1);

	/*Display on; cursor on; blink on*/
```

```
	lcd_command(0x0F);
	us_48_sleep(1);

	/*Clear screen; cursor home*/
	lcd_clear_screen();
}

void lcd_out_char(uint8_t byte)
{
	uint8_t temp = byte; // Save command
	temp &= 0xF0; // Peel off the four MSBs
	PORTLCD = (temp | RS); // Signal a write to DDRAM for display
	latch();
	ms_sleep(5); // Wait for LCD data to go
	temp = byte;
	temp <<= 4; // Shift the four LSBs up to the MSBs (Or, sending
position)
	temp &= 0xF0;
	PORTLCD = (temp | RS); // Load LSBs
	latch();
	us_48_sleep(1);
}

/*
** This will not word wrap.
** Put in if-elses if you want word wrap.
*/
void lcd_out_string(char *s)
{
	while (*s) lcd_out_char(*s++);
}

/*
**	Function:		writeIntegerToLCD (lcd_out_int)
**	Parameters:		integer - the integer that will be written to
the LCD
**	Purpose:		Coverts a standard 16-bit int into the ASCII
**				representation of the number and writes
that number
**				to the LCD.
**				*** Note:   The maximum value that can be
written is
**					9999.  This is because
there is no
**					ten thousands place
support.
**	Returns			<none>
*/
void lcd_out_int(uint16_t integer)
{
	/*
	** Break down the original number into the thousands, hundreds,
tens,
	** and ones places and then immediately write that value to the
LCD
	*/
	uint8_t thousands = integer / 1000;
```

```
	lcd_out_char(thousands + 0x30); // 0x30 = zero in hexidecimal
format = 0b00110000 (in binary format)

	uint8_t hundreds = (integer - thousands*1000) / 100;
	lcd_out_char(hundreds + 0x30);

	uint8_t tens = (integer - thousands*1000 - hundreds*100 ) / 10;
	lcd_out_char(tens + 0x30);

	uint8_t ones = (integer - thousands*1000 - hundreds*100 -
tens*10);
	lcd_out_char(ones + 0x30);
}

/*
** I broke up my code into seperate while(1) loops to increase the
chances that Ant will navigate through my course.
** My demonstration was what I coded for. As a result, my code takes on
a role-playing game (RPG) feel.
** This RPG appearance was unintentional.
*/

/*
** The values stated below were found in the Harris rotunda of NEB.
** Do not use these values elsewhere.
*/

int main(void)
{
	DDRB |= 0x01; // Turn LED enable on

	/*Initializations*/
	timer_init();

	sei(); // Enable interrupts

	lcd_init();
	ad_init();
	srf_init();
	servo_init();
	uart0_init();
	cmu_init();
	dir_init();

	//center(); // Delay to center servos

	/*Take dummy reads*/
	CMU_TC(90, 180, 0, 90, 10, 70);
	ms_sleep(50);
	CMU_GM();

	lcd_clear_screen();
	lcd_out_int(5666);
	ms_sleep(512);

	lcd_clear_screen();
	lcd_out_string("I'm hungry");
```

```
	/*Test gripper*/
	open();
	grab();
	ms_sleep(512);
	open();

	uint16_t IR = ad_readn(2,4); // Dummy read: IR (channel 0, PORTF
bit 0), 4 times and average

	forward(33);

	/*First of multiple infinite loops*/
	while(1)
	{
		/*Check IR for obstacle or edge*/
		IR = ad_readn(2,4); // Read IR (channel 2) 4 times and
average

		/*
		** Warning: These IR functions will send Ant off the edge
if Ant turns
		** under seven inches away from the edge.
		*/
		if ((IR > 108) && (IR < 339))
		{
			ms_sleep(512);
			IR = ad_readn(2,4);
			/*Ignore black tape*/
			if (!((IR > 108) && (IR < 339)))
			{
				lcd_clear_screen();
				lcd_out_string("I saw a trail");
			}
			/*Avoid an edge in front of a chair or something*/
			else
			{
				lcd_clear_screen();
				lcd_out_string("Aaaah!");
				reverse(33);
				rand_turn(44);
				ms_sleep(760);
			}
		}
		else if (IR > 400)
		{
			ms_sleep(512);
			IR = ad_readn(2,4);
			/*Ignore a flash of light*/
			if (!(IR > 400))
				;
			/*Avoid an obstacle*/
			else if (IR > 460)
			{
				lcd_clear_screen();
				lcd_out_string("Missed an object");
				rand_turn(33);
```

```
                  }
            }
            /*Avoid an edge*/
            else if (IR < 50)
            {
                  lcd_clear_screen();
                  lcd_out_string("Aaaah!");
                  reverse(33);
                  rand_turn(44);
                  ms_sleep(760);
            }

            /*First of three (To increase detection probability) line
detection methods in this while(1) loop*/
            /*I should use case-switches instead of if-elses for
speed*/
            /*Left line-acquisition does NOT work*/
            if((PINDIR & 0x06) == 0x02) // Left line-acquisition
            {
                  keep_turning1:
                  while(!(PINDIR & 0x08))
                  {
                        reverse(20);
                  }
                  while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                  {
                        L_turn(33);
                  }
                  if (!(PINDIR & 0x08))
                        goto keep_turning1;
                  goto line_track;
            }
            else if((PINDIR & 0x06) == 0x04)
            {
                  while(!(PINDIR & 0x08))
                  {
                        forward(20);
                  }
                  while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                  {
                        R_turn(20);
                  }
                  lcd_clear_screen();
                  lcd_out_string("Following trail");
                  goto line_track;
            }
            else if(!(PINDIR & 0x06)) // I assumed Ant goes right more
often than not
            {
                  while(!(PINDIR & 0x08))
                  {
                        forward(20);
                  }
                  while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
```

```
                {
                        R_turn(20);
                        ms_sleep(512);
                }
                goto line_track;
        }

        uint8_t threshold = 7; // Point to turn robot (For sonar
range)
        ms_sleep(54); // Delay between sonar pulses
        /*Check left sonar for an object ahead*/
        uint8_t l_range = l_sonar();
        if (l_range < threshold)
        {
                lcd_clear_screen();
                lcd_out_string("Missed an object");
                R_turn(40);
        }

        /*Line Acquiring (2)*/
        if((PINDIR & 0x06) == 0x02)
        {
                keep_turning4:
                while(!(PINDIR & 0x08))
                {
                        reverse(20);
                }
                while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                {
                        L_turn(33);
                }
                if (!(PINDIR & 0x08))
                        goto keep_turning4;
                goto line_track;
        }
        else if((PINDIR & 0x06) == 0x04)
        {
                while(!(PINDIR & 0x08))
                {
                        forward(20);
                }
                while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                {
                        R_turn(20);
                }
                goto line_track;
        }
        else if(!(PINDIR & 0x06)) // I assumed Ant goes right more
often than not
        {
                while(!(PINDIR & 0x08))
                {
                        forward(20);
                }
                while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
```

```
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                {
                        R_turn(20);
                        ms_sleep(512);
                }
                lcd_clear_screen();
                lcd_out_string("Following trail");
                goto line_track;
        }

        ms_sleep(60); // Delay between sonar pulses
        uint8_t r_range = r_sonar();
        if (l_range < threshold && r_range < threshold)
        {
                lcd_clear_screen();
                lcd_out_string("Missed an object");
                rand_turn(40);
        }
        else if(r_range < threshold)
        {
                lcd_clear_screen();
                lcd_out_string("Missed an object");
                L_turn(40);
        }

        /*Clear ahead*/
        forward(33);

        /*Line Acquiring (3)*/
        if((PINDIR & 0x06) == 0x02)
        {
                keep_turning7:
                while(!(PINDIR & 0x08))
                {
                        reverse(20);
                }
                while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                {
                        L_turn(33);
                }
                if (!(PINDIR & 0x08))
                        goto keep_turning7;
                goto line_track;
        }
        else if((PINDIR & 0x06) == 0x04)
        {
                while(!(PINDIR & 0x08))
                {
                        forward(20);
                }
                while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                {
                        R_turn(20);
                }
                lcd_clear_screen();
```

```
                  lcd_out_string("Following trail");
                  goto line_track;
            }
            else if(!(PINDIR & 0x06)) // I assumed Ant goes right more
often than not
            {
                  while(!(PINDIR & 0x08))
                  {
                        forward(20);
                  }
                  while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                  {
                        R_turn(20);
                        ms_sleep(512);
                  }
                  lcd_clear_screen();
                  lcd_out_string("Following trail");
                  goto line_track;
            }
      }

      while(1)
      {
            line_acquire:
            // Line Acquiring only (To eliminate the possibility of
entering an undesired behavior)

            forward(33);

            if((PINDIR & 0x06) == 0x02)
            {
                  keep_turning10:
                  while(!(PINDIR & 0x08))
                  {
                        reverse(20);
                  }
                  while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                  {
                        L_turn(33);
                  }
                  if (!(PINDIR & 0x08))
                        goto keep_turning10;
                  goto line_track;
            }
            else if((PINDIR & 0x06) == 0x04)
            {
                  while(!(PINDIR & 0x08))
                  {
                        forward(20);
                  }
                  while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                  {
                        R_turn(20);
                  }
```

```
                  lcd_clear_screen();
                  lcd_out_string("Following trail");
                  goto line_track;
            }
            else if(!(PINDIR & 0x06)) // I assumed Ant goes right more
often than not
            {
                  while(!(PINDIR & 0x08))
                  {
                        forward(20);
                  }
                  while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                  {
                        R_turn(20);
                        ms_sleep(512);
                  }
                  lcd_clear_screen();
                  lcd_out_string("Following trail");
                  goto line_track;
            }
      }

      while(1)
      {
            line_track:
            /*
            ** PINDIR of 8 is the abitrator for stopping on a line.
            ** I added this mechanical stop because my CMUCam took a
long time to look for colors
            ** (Not that I blame my CMUCam).
            ** My robot did not continue line-tracking.
            */
            if ((PINDIR & 0x08) == 0x08)
            {
                  if ((PINDIR & 0x06) == 0x06)
                        reverse(18); // Off line completely
                  else if ((PINDIR & 0x06) == 0x02)
                        R_turn(15); // Right side off line
                  else if ((PINDIR & 0x06) == 0x04)
                        L_turn(15); // Left side off line
                  else if (!(PINDIR & 0x06))
                        forward(25); // On the line
            }
            else if (!(PINDIR & 0x08))
            {
                  stop();
                  CMU_GM(); // Dummy read
                  ms_sleep(1024); // Wait for transients to die out
(Very important)
                  CMU_GM(); // Get means (15 min, 240 max)
                  int RED = CMUResponseBuffer[1]; // Red mean
                  int GREEN = CMUResponseBuffer[2]; // Green mean
                  int BLUE = CMUResponseBuffer[3]; // Blue mean

                  /*
                  ** Look for Red candy bar
```

```
                  ** These ranges were determined at 4 PM in the Harris
rotunda of the New
                  ** Engineering Building on 04-23-06 for a Nutri-Grain
bar experimentally.
                  ** Do NOT use these values in other environments
because the values will change.
                  */
                  if ((RED > 80) & (RED < 175) & (GREEN > 41) & (GREEN
< 98) & (BLUE > 15) & (BLUE < 70))
                  {
                        ms_sleep(8);
                        /*Double-check*/
                        CMU_GM();
                        RED = CMUResponseBuffer[1];
                        GREEN = CMUResponseBuffer[2];
                        BLUE = CMUResponseBuffer[3];

                        if ((RED > 80) & (RED < 175) & (GREEN > 41) &
(GREEN < 98) & (BLUE > 15) & (BLUE < 70))
                        {
                              stop();
                              lcd_clear_screen();
                              lcd_out_string("Red is delicious");
                              grab(); // Pick up object
                              while(!(PINDIR & 0x08))
                              {
                                    forward(20);
                              }
                              goto end;
                        }
                        else ;
                  }
                  /*Look for blue - to avoid it*/
                  else if ((RED > 80) & (RED < 130) & (GREEN > 70) &
(GREEN < 125) & (BLUE > 87) & (BLUE < 132))
                  {
                        ms_sleep(8);
                        CMU_GM();
                        RED = CMUResponseBuffer[1];
                        GREEN = CMUResponseBuffer[2];
                        BLUE = CMUResponseBuffer[3];

                        if ((RED > 80) & (RED < 130) & (GREEN > 70) &
(GREEN < 125) & (BLUE > 87) & (BLUE < 132))
                        {
                              open();
                              lcd_clear_screen();
                              lcd_out_string("Blue is gross");

                              /*Turn around and reacquire line*/
                              reverse(33);
                              ms_sleep(1024);
                              R_turn(51);

                              reverse(33);
                              ms_sleep(1024);
```

```
                        goto line_acquire;
                    }
                }
                else // Nothing
                {
                    open();
                    lcd_clear_screen();
                    lcd_out_string("Where's food?");
                    lcd_clear_screen();
                    /*For testing: Display RGB values*/
                    lcd_out_string(" ");
                    lcd_out_int(binary2int(RED)); // Red mean
                    lcd_out_string(" ");
                    lcd_out_int(binary2int(GREEN));     // Green
mean
                    lcd_out_string(" ");
                    lcd_out_int(binary2int(BLUE)); // Blue mean
                    ms_sleep(4096);

                    reverse(33);
                    ms_sleep(1024);
                    R_turn(51);

                    reverse(33);
                    ms_sleep(1024);

                    goto line_acquire;
                }
            }
        }

        while(1)
        {
            end:
            // Assuming that a red Nutri-Grain bar was picked up
            /*Line-acquire*/
            forward(20);

            if((PINDIR & 0x06) == 0x02)
            {
                keep_turning13:
                while(!(PINDIR & 0x08))
                {
                    reverse(20);
                }
                while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                {
                    L_turn(33);
                }
                if (!(PINDIR & 0x08))
                    goto keep_turning13;
                goto finale;
            }
            else if((PINDIR & 0x06) == 0x04)
            {
                while(!(PINDIR & 0x08))
```

```
                  {
                        forward(20);
                  }
                  while((((PINDIR & 0x06) == 0x06) || ((PINDIR & 0x06)
== 0x04) || ((PINDIR & 0x06) == 0x02)) && (!(PINDIR & 0x08)))
                  {
                        R_turn(20);
                  }
            }
            else if((PINDIR & 0x06) == 0)
            {
                  goto finale;
            }
            else // Somehow Ant got off its trail
            {
                  stop();
                  lcd_out_string("No trail");
            }
      }

      while(1)
      {
            finale:
            /*Go around a curve (to the right) and drop off food*/
            if ((PINDIR & 0x08) == 0x08)
            {
                  if ((PINDIR & 0x06) == 0x06)
                        reverse(15); // Off line completely
                  else if ((PINDIR & 0x06) == 0x02)
                        R_turn(20); // Right side off line
                  else if ((PINDIR & 0x06) == 0x04)
                        L_turn(13); // Left side off line
                  else if (!(PINDIR & 0x06))
                        forward(28); // On the line
            }
            else if (!(PINDIR & 0x08))
            {
                  /*Ant arrived at drop-off point*/
                  stop();
                  open(); // Release object
                  reverse(35);
                  ms_sleep(4096);
                  stop();
                  lcd_clear_screen();
                  lcd_out_string("I'm home!");
                  ms_sleep(2048);

                  /*
                  ** Stay still until picked up.
                  ** Then avoid obstacles and edges but ignore trails.
                  ** Supervise Ant now.
                  */
                  while(PINDIR)
                        ;
                  goto avoidance;
            }
      }
```

```
	while(1)
	{
		avoidance:

		IR = ad_readn(2,4); // Read IR (channel 2) 4 times and
average

		if ((IR > 108) && (IR < 339))
		{
			ms_sleep(512);
			IR = ad_readn(2,4);
			/*Ignore black tape*/
			if (!((IR > 108) && (IR < 339)))
			{
				lcd_clear_screen();
				lcd_out_string("I saw a trail");
			}
			/*Avoid an edge in front of a chair or something*/
			else
			{
				lcd_clear_screen();
				lcd_out_string("Aaaah!");
				reverse(33);
				L_turn(40);
				ms_sleep(760);
			}
		}
		else if (IR > 400)
		{
			ms_sleep(512);
			IR = ad_readn(2,4);
			/*Ignore a flash of light*/
			if (!(IR > 400))
				;
			/*Avoid an obstacle*/
			else if (IR > 460)
			{
				lcd_clear_screen();
				lcd_out_string("Missed an object");
				rand_turn(33);
			}
		}
		/*Avoid an edge*/
		else if (IR < 50)
		{
			lcd_clear_screen();
			lcd_out_string("Aaaah!");
			reverse(33);
			L_turn(40);
			ms_sleep(760);
		}

		uint8_t threshold = 7; // Point to turn robot (For sonar
range)
		ms_sleep(54); // Delay between sonar pulses
		/*Check left sonar for an object ahead*/
```

```
            uint8_t l_range = l_sonar();
            if (l_range < threshold)
            {
                  lcd_clear_screen();
                  lcd_out_string("Missed an object");
                  R_turn(33);
            }

            ms_sleep(60); // Delay between sonar pulses
            uint8_t r_range = r_sonar();
            if (l_range < threshold && r_range < threshold)
            {
                  lcd_clear_screen();
                  lcd_out_string("Missed an object");
                  rand_turn(33);
            }
            else if(r_range < threshold)
            {
                  lcd_clear_screen();
                  lcd_out_string("Missed an object");
                  L_turn(33);
            }

            /*Clear ahead*/
            forward(33);
      }

      return 0;
}
```