Final Report

Courier-Bot

Kristopher Brosch

EEL 4665/5666
Intelligent Machines Design Laboratory

Instructors:
Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

TAs:
Josh Weaver
Ryan Stevens
Tim Martin
Devin Hughes
Sean Frucht

Table of Contents

1 Abstract

Courier-Bot is an autonomous wheeled robot that wanders its environment, grabs objects that it finds with its gripper, then brings those objects to different places based on their color. Courier-Bot is a complex system made up of a mobile platform, a mechanical gripper, sensors, infrared beacons, objects, and programming that allows Courier-Bot to execute his objectives.

2 Executive Summary

Courier-Bot is an autonomous robot that wanders his environment, looking for objects. When he finds one, he grabs it, then searches for an infrared beacon associated with the color of the object he found. When he finds the beacon, Courier-Bot releases the object, then goes in search of more objects. Courier-Bot consists of many components that allow him to perform these functions.

Courier-Bot's mobile platform is driven by two motors, and has a ball-caster third contact point. Courier-Bot has a unique gripper design that allows it to grab and move objects. Sonar rangefinders enable Courier-Bot to avoid obstacles and to detect objects to grab with its gripper. An analog infrared beacon finder circuit that enables Courier-Bot to distinguish between three infrared beacons was designed for Courier-Bot, as were the beacons he searches for. Courier-Bot uses an Atmel XMEGA microcontroller to control all of its components. Courier-Bot also has a second microcontroller that drives a graphical LCD that shows Courier-Bot's face. Courier-Bot's programming includes behaviors that enable him to perform his objectives.

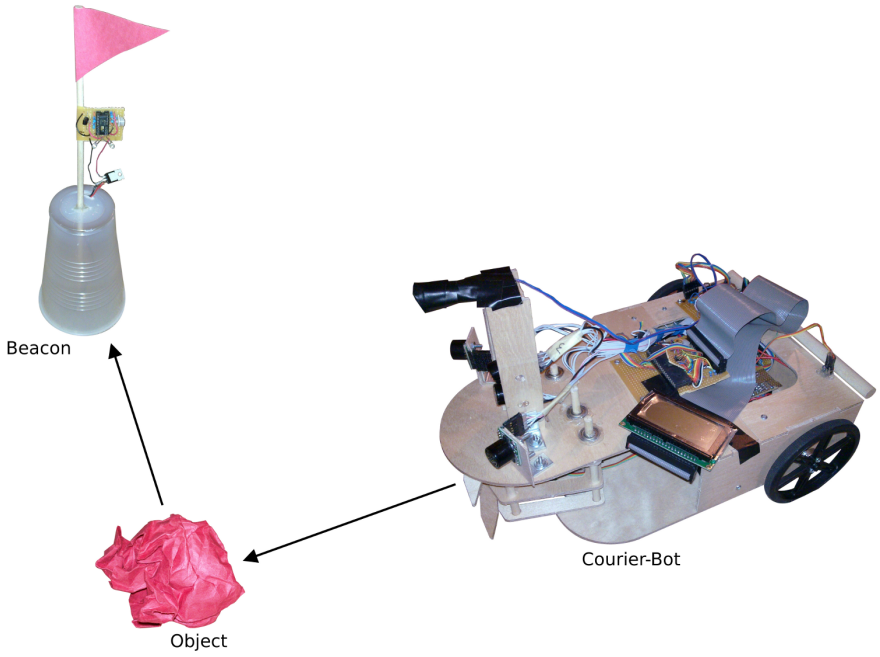

Figure 1: Courier-Bot System

3 Introduction

The objective of this project was to create a robot with some mechanical complexity, with some electrical complexity, and with an interesting behavior. These objectives lead to the creation of a robot that will find objects and carry them to different locations based on their color. The robot, named Courier-Bot, was designed to move about its environment, detect items it can carry, then pick up items one at a time and move them to different target locations marked with beacons. Different beacons are associated with different colored objects. Courier-Bot achieves mechanical complexity with his gripper, which allows him to grab and move objects. Courier-Bot achieves electrical complexity with his infrared beacon system, and the interfacing of a microcontroller with all of his electrical sensors and components. Courier-Bot achieves interesting behaviors through his programming, which uses his mechanical components and his electrical components. This paper further describes the requirements of Courier-Bot, and the methods of meeting those requirements. First, Courier-Bot as a system is discussed, and its component parts are described. Next, the paper details Courier-Bot's platform, its mechanical components, its sensors, and its programming.

4 Integrated System

Courier-Bot wanders its environment, finds objects, and moves them to different beacons based on their color. The overall system includes Courier-Bot itself, as well as the beacons, which designate the locations Courier-Bot will bring different colored objects. Courier-Bot consists of a mobile platform which includes a gripping mechanism that is used to carry objects, and the appropriate sensors that enable it to perform obstacle avoidance, object detection, and to find the beacons. The beacons consist of LEDs and circuits to modulate the LEDs at different frequencies in order to enable Courier-Bot to differentiate them from one another. Figure 1 shows several components of the overall Courier-Bot system: Courier-Bot itself, the mobile robot, an object it will pick up, and the beacon it will take the object to. Figure 2 shows how the various electronic components of Courier-Bot are interconnected in his platform.

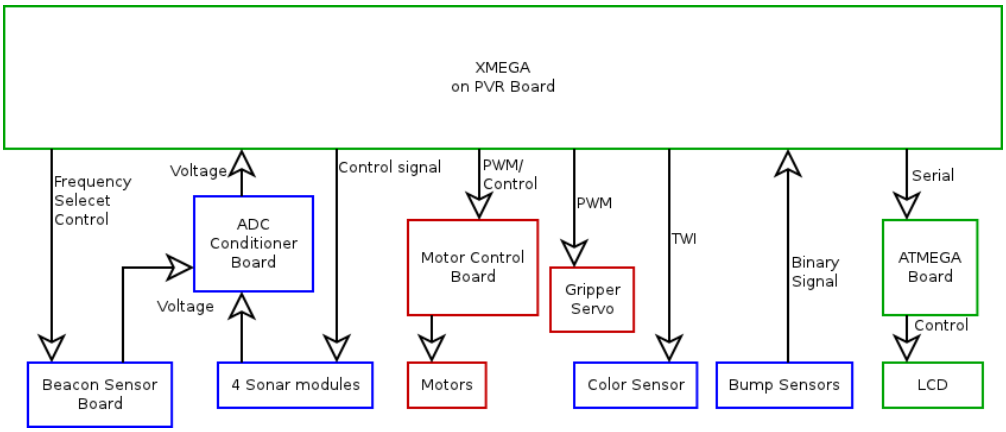

Figure 2: Courier-Bot Electronic System

4.1 Microcontroller

Courier-Bot is controlled by an Atmel AVR ATxmega128A1 microcontroller. The Atxmega128A1 is an 8/16 bit microcontroller from Atmel that can run at 32MHz [2] [3]. The microcontroller is on a Pridgen Vermeer Robotics (PVR) printed circuit board, which provides convenient connections for most of its pins. Courier-Bot's XMEGA runs at 32MHz on an internal clock. The XMEGA executes a main loop with a regular period, which repeatedly updates sensor data, uses behavior code to decide what motor and servo actions to take based on that sensor data, and updates the motor and servo driving parameters. A listing of Courier-Bot's code is included in Appendix A.

4.2 LCD

Courier-Bot has a 128-pixel by 64-pixel graphical LCD, on which Courier-Bot shows his face, as well as his current behavior. The LCD control is handled by an AVR ATmega324. The ATMEGA contains the bitmaps of ASCII characters, and the bitmaps that make up Courier-Bot's face, and code that handles drawing them to, and refreshing the LCD. Courier-Bot's face is shown in Figure 3. The XMEGA communicates text to display, as well as what face to display and which direction to "look" to the ATMEGA via one of its USARTs. All of the ATMEGA code is in the appendix, starting with Listing 16 on page 39, and the XMEGA's code to communicate with the ATMEGA is in Listings 8 and 9, starting on page 34.

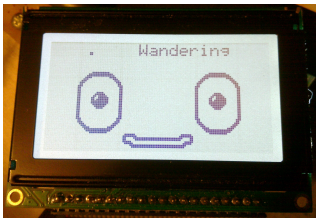

Figure 3: Courier-Bot's LCD displaying his face

4.3 Power

Courier-Bot is powered by a pack of six rechargable AA-batteries. The PVR board includes voltage regulators, which provide a 3.3 volt supply for the XMEGA, sonar sensors, and color sensor, and 5 volt supplies for the servo, motors, ATMEGA, LCD, and beacon sensor.

4.4 Objects and Beacons

The beacons are astable 555-timer circuits, which each drive infrared LEDs at a specific frequencies. The beacons, as well as the beacon sensor, are described in detail in section 7.3. The objects that Courier-Bot finds and carries to beacons are balls of colored paper.

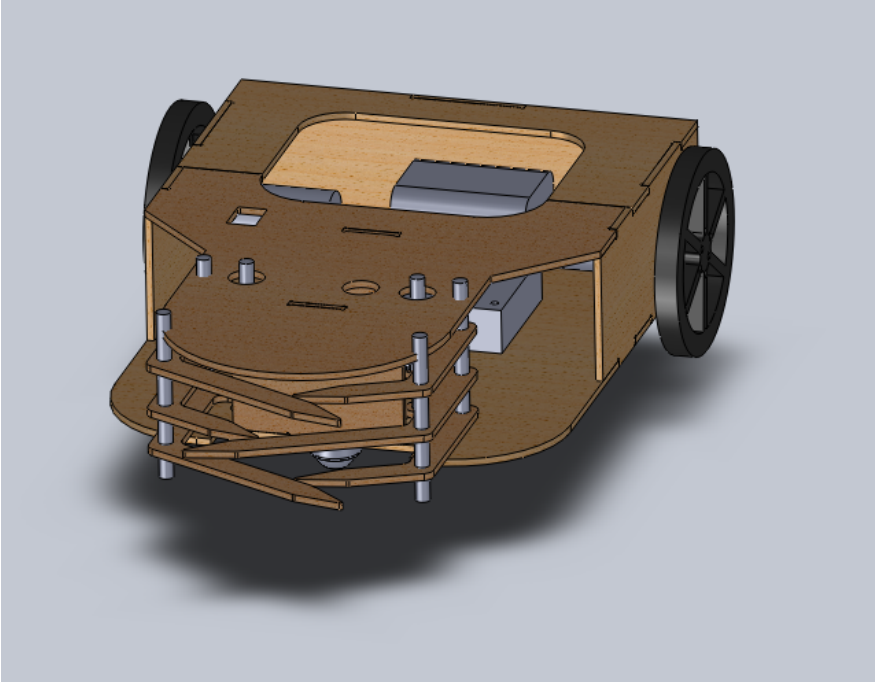

Figure 4: Courier-Bot Platform CAD Drawing

5 Mobile Platform

Courier-Bot's mobile platform is required to enable Courier-Bot to move freely in an open, flat environment, to perform obstacle avoidance, and to pick up, carry, and release small objects. Courier-Bot's mobile platform is designed to be adept at moving at a reasonable speed in an unhazardous environment.The robot has a two-wheel platform that enables it to move forwards and backwards, and turn in place, by changing the speed and direction of the two wheels independently. A ball caster provides a third contact point with the ground to provide stability.

In order to accommodate all the electronics and the batteries that must be carried by Courier-Bot, the platform is designed in the back like a box; most of the electronics are housed inside, while some boards are mounted on top. The bottom front of Courier-Bot is dedicated to his gripper mechanism. The color sensor is mounted inside the gripper, facing down. Above the gripper, on the top platform, are the sonar sensors used for obstacle avoidance and object detection, and the beacon sensor.

5.1 Design and Construction

Courier-Bot's mobile platform was designed using the *Solid Works* CAD software. Figure 4 shows the CAD drawing. This design was used to cut the wood pieces that make up Courier-Bot's body and gripper. Later, other elements were added, including the wood to mount the sonars and beacon sensor, and the bump sensors. The final version of Courier-Bot's platform is visible in Figure 1.

6 Actuation

Courier-Bot's platform has two wheels. Courier-Bot is able to turn each wheel forwards or backwards. It has motors to turn its wheels. Courier-Bot also has the ability to grab, hold, and release small objects. Courier-Bot uses a gripping mechanism mounted to the front of its platform to pick up objects. The gripping mechanism is actuated using a servo.

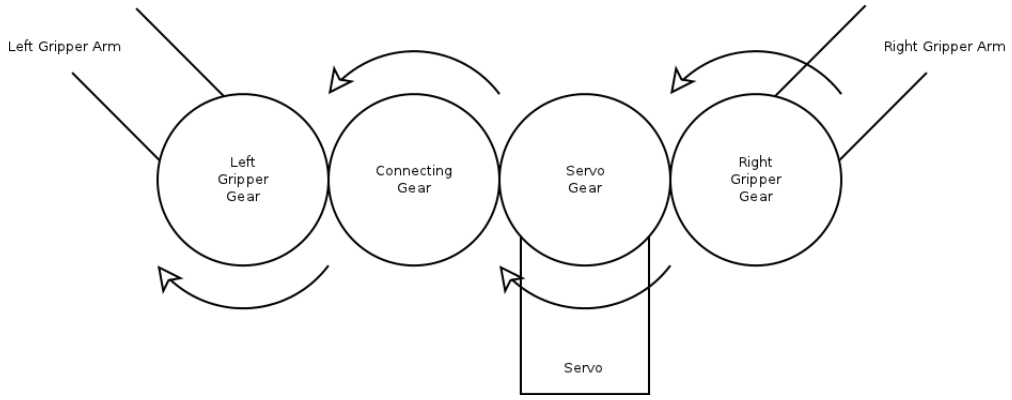

Figure 5: Diagram of Gripper Actuation (viewed from above)

6.1 Wheel Motors

The motors used to drive Courier-Bot's wheels are Pololu 47:1 gearmotors. They have a stall torque rating of 50oz-in and run at 120rpm, with no load [7]. Courier-Bot's wheels are Pololu plastic wheels which have a 9cm diameter [8]. This means that Courier-Bot could theoretically move at a speed of $120rpm*(\pi*0.9m)*(\frac{1min}{60sec}) = 5.65m/s$ if there were no load on the motors, but in practice, the robot moves much slower depending on the surface it is driving on.

The motors are driven using an L298N, which is an H-Bridge motor driver IC [10]. Courier-Bot's microcontroller is used to generate a PWM signal for each motor on the L298N enable pins, which translates into rough control of speed and torque. The microcontroller also drives the H-Bridge inputs, which enable it to drive the motors forward, in reverse, or to force them to stop.

Courier-Bot's behavior software generates speed and direction control information for the motors. This control information is not used directly, however. Instead, the motor control values are filtered by averaging the past 16 values, and the averaged value is used to set the PWM parameters. This prevents the motor control signals from changing too quickly. All the motor control code is in Listings 10 and 11, starting on page 34.

6.2 Gripper

Courier-Bot's gripper is designed to close around light objects so that Courier-Bot can move them around. The gripper consists of six cut pieces of wood connected to shafts, which are actuated using a servo motor. The servo is a Hitec HS-635HB, which is rated for 83oz-in of torque [9], which is more than strong enough for grabbing paper objects.

The servo has a gear connected directly to its shaft, which directly drives a gear connected to the right gripper shaft, and which also drives an intermediary gear that drives the left gripper shaft. In this configuration, a clockwise turning of the servo motor (when viewed from above) corresponds to closing the gripper claw; a counter-clockwise turning corresponds to opening the gripper claw. Figure 5 shows this layout. In the figure, the circles are gears, and the arrows show the direction the gears turn when the gripper is closing.

When being actuated by the microcontroller, the speed of the gripper is limited by repeatedly commanding the servo to go to slightly different angles. The servo is controlled using a PWM signal generated by the microcontroller; the PWM signal corresponds to an angle which the servo should turn to. The servo control code is in Listings 12 and 13, starting on page 37.

7 Sensors

Courier-Bot utilizes several types of sensors in order to achieve its goals. Courier-Bot needs sensors to allow it to perform obstacle avoidance and to detect objects to grab. The robot needs a sensor to detect the

color of carried objects. The robot also needs sensors to detect beacons, identify them, and determine their location relative the robot. Courier-Bot uses sonar rangefinders for obstacle avoidance and object detection. The robot uses a color sensor, mounted in its gripping mechanism to detect the color of carried objects. It uses a custom-built beacon sensor to detect and locate LED beacons. There are also two bump sensors on Courier-Bot.

7.1 Obstacle Avoidance and Object Detection

Courier-Bot has four sonar rangefinder sensors mounted on it, which it uses for obstacle avoidance and object detection. Three of the sonar modules are used for obstacle avoidance. One faces left, one right, and one center, all mounted on top of the front of the platform. The fourth sonar also faces front, but it is angled down. Courier-Bot detects objects using the two forward facing sonars. When an object is short enough that it is not detected by the obstacle-avoidance forward facing sonar, but it is detected by the downward-tilted sonar, Courier-Bot tries to pick it up.

The sonars that are used are Maxbotix LV-MaxSonar-EZ0 and LV-MaxSonar-EZ3 sensors. The EZ0 sensors have a wider beam width [5], and are used for the left and right sensors. The EZ3 sensors, with a narrower beam width [6] are used for the front and downward facing sensors. The sonars are configured to sample one at a time, round-robin in order to prevent the sensors from interfering with one another while they operate. The Maxbotix sonars are designed to be able to operate this way. In order to facilitate the round-robin continuous sampling, cables connect each sonar to the next in sequence. An I/O pin on the microcontroller is used to start the first sonar reading; after that, when one sonar module finishes, it signals the next to start. The last sonar's transmit pin is connected to the first sonar's receive pin through a resistor so that the microcontroller may drive the receive pin to start the chain; once it does this during startup, the microcontroller releases the line by configuring that pin as an input.

The sonar modules report a distance measurement by producing an analog voltage. This voltage can be in the full range of 0 volts to the supply voltage (3.3 volts). Unfortunately, the XMEGA microcontroller's analog to digital converter can only handle inputs up to the supply voltage minus 0.6 volts, and ignores even the top 5% of this range [1]. To get around this limitation, a circuit was designed to produce an appropriate voltage reference signal for the XMEGA, and to lower the voltages from the sonars. This circuit is shown in Figure 10, in the appendix.

7.2 Color Sensor

Courier-Bot's color sensor is used to determine the color of an object Courier-Bot is carrying. The sensor is a ADJD-S371 chip, which is a 10-bit red, green, blue, and clear color sensor [4]. This chip is interfaced with Courier-Bot's microcontroller with an I^2C-style interface, using the XMEGA's TWI module. The microcontroller can set configuration registers, start sensor readings, and read 10-bit red, green, blue, and clear result values using the I^2C-style interface. The code used to interface with and use the color sensor is in Listings 4 and 5, starting on page 28. The color sensor is mounted on the inside of Courier-Bot's gripper, facing downward. A bright white LED is also mounted next to the color sensor, in order to provide consistent lighting for the sensor, which enables the sensor to provide more consistent readings.

The ADJD-S371 color sensor needs to be calibrated. It is done so by setting calibration registers on the chip. Calibration of Courier-Bot's sensor was done one time by taking sensor readings and adjusting the calibration register settings until each of red, green, and blue sensor values could be made to change over their full range with the bright white LED providing lighting. These chosen values are in the source code file colorsensor.h, which can be found on page 28.

Courier-Bot samples takes a color sensor reading just after it grabs an object. Since Courier-Bot looks for red, green, and blue objects, it finds the maximum of the red, green, and blue sensor readings, and assumes it is carrying an object of that color.

7.3 Infrared Beacon Sensor

The beacons and beacon sensor used on Courier-Bot were custom-designed for Courier-Bot. The beacons contain infrared LEDs, with circuitry to make them blink at different frequencies. The sensor on the robot

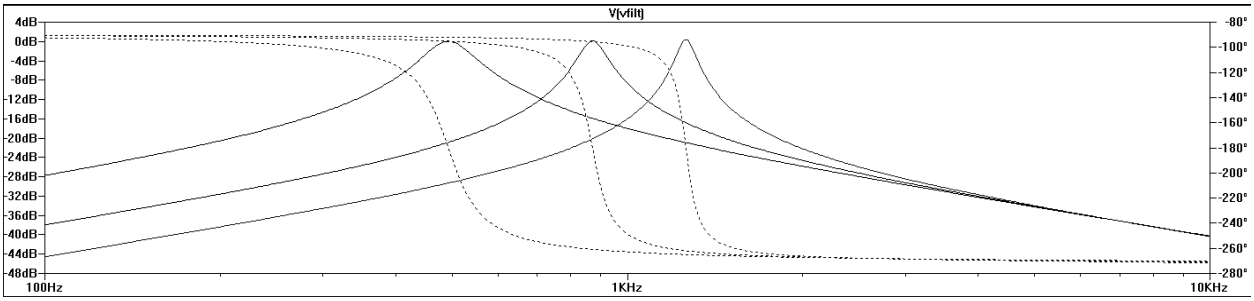

Figure 6: Beacon-detector filter response

consists of a photodiode, which detects the infrared transmissions from the beacons, and a tunable bandpass filter which allows the robot to differentiate the signals from the different beacons. The photodiode in the detector circuit is directional, so when mounted on the front of the robot, it allows the robot to detect and identify beacons directly in front of it.

There are three beacons, designed to oscillate at about 500Hz, about 880Hz, and about 1.3KHz. These frequencies were chosen for convenience of design of the detector circuit, and so that there is not significant interference between beacons due to fundamental or harmonic frequencies of the beacons.

The beacons consist of an astable 555-timer circuit, the output of which switches a transistor that drives two infrared LEDs. This circuit is shown in Figure 8, in the appendix. The 555-timer circuit oscillates at a frequency determined by two resistors and a capacitor: $f = \frac{1}{0.693C_1(R_1+2R_2)}$; it is high for $0.693(R_1 + R_2)C_1$ seconds, and low for $0.693R_2C_1$ seconds [11]. There are three beacons, each with component values as shown in Figure 8, but with differing values of R_2. These values should theoretically be about 144k, 81.5k, and 55k ohms for the desired frequencies respectively according to the above equations. Experimentally, the values required were smaller when the circuit was implemented on a breadboard due primarily to built-in capacitance of the breadboard. When implemented on perfboard, a 10k potentiometer was added in series with R_2 as a variable resistance to tune the oscillating frequency.

The beacon sensor circuit is shown in Figure 9, in the appendix. The input to this circuit is the signal from a photodiode. The photodiode is reverse-biased, and generates a small current when exposed to infrared light. This current is converted to a significant voltage using a large (4.7 meg-ohm) series resistor. After a voltage follower, the signal passes through a second-order bandpass filter, and then a peak-detector circuit, the output of which is read by an A/D input on the robot's microcontroller.

The bandpass filter utilizes the multiple feedback topology. It is made tunable by switching in and out R_{3A}, R_{3B}, and R_{3C}. Different resistor values in this location change the filter's center frequency. They also change the Q-value of the filter, but with the chosen frequencies, the varying Q is acceptable. Figure 6 shows the magnitude and phase response of the filter portion of the circuit over a range of input frequencies for the three different values of R_3. Each of the three resistors is connected to a general purpose I/O pin of the robot's microcontroller. The microcontroller sets the desired frequency by making two pins high-impedance (configuring them as inputs), and driving the third pin low; this effectively removes two resistors from the circuit and grounds the third. The microcontroller can therefore tune the filter to the beacon the robot needs to detect, then read the value on the A/D input. Two diodes are used in the peak detector portion of the circuit to lower the bias of the signal to a level convenient for the microcontroller's A/D input.

The microcontroller reads the beacon sensor using an A/D input. The code used to interpret information from the IR beacon sensor is in Listings 6 and 7, starting on page 31.

7.4 Bump Sensors

Courier-Bot has two bump sensors, on the rear of the platform, which are used when moving to detect when Courier-Bot has backed up into an obstacle. Both bump sensors are connected to a wooden bumper which spans the width of the back of the robot. A bump on the left or right triggers one sensor, while a bump on the center or on the entire back of the robot triggers both. The bump sensors are of very simple design; they are simply switches which are connected to ground and to XMEGA inputs with internal pull-up resistors.

The XMEGA pins are configured to be inverting, so pressing the bump sensors results in a '1' being read by the microcontroller. The used to configure and read the bump sensors are in Listings 14 and 15, starting on page 39.

8 Behaviors

Courier-Bot wanders its environment, looking for objects to carry, then picks up objects and carries them to different locations based on their color. Courier-Bot uses six distinct behaviors in total to achieve these actions.

8.1 Behavior Descriptions

Courier-Bot's behaviors are *Waiting*, *Wandering*, *Grabbing*, *Searching*, *Releasing*, and *Obstacle-Avoiding*. The waiting behavior is only used when Courier-Bot is first powered-on, and waits for a bump-sensor to be pressed before moving, and is less important than the other behaviors. The obstacle-avoidance behavior is a special case because it is never explicitly used as the primary behavior, but it is deferred to in the wandering and searching behaviors when there are obstacles to avoid. All of the behavior code is in courierbot.c, which is included in Listing 1 in the appendix, on page 13.

8.1.1 Waiting

Courier-Bot starts up *Waiting*. In this state, Courier-Bot will stand still until one of his bump sensors is pressed. Once this happens, Courier-Bot will move on to *Wandering*, and follow the behavior loop depicted in Figure 7.

8.1.2 Wandering

In the *Wandering* state, Courier-Bot mostly defers to *Obstacle-Avoiding*, but occasionally turns a random amount in a random direction. Courier-Bot also checks the sonar sensors while *Wandering*. If an object is detected, Courier-Bot moves to *Grabbing*.

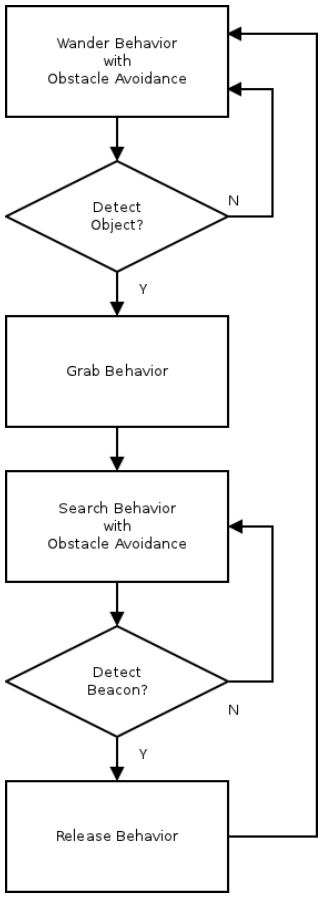

Figure 7: Flowchart of Courier-Bot's Main Behaviors

8.1.3 Grabbing

The *Grabbing* behavior consists of a series of sequential steps. Courier-Bot stops, opens its gripper, moves forward, closes its gripper (hopefully around an object), moves backwards, then stops and changes to the *Searching* behavior.

8.1.4 Searching

In the *Searching* behavior, Courier-Bot will defer to *Obstacle-Avoiding*, but often turns around when not deferring to *Obstacle-Avoiding* in an attempt to find the correct beacon. Courier-Bot records the beacon it is searching for by checking the color sensor when the *Searching* state is first entered. When the correct color sensor is found while *Searching*, Courier-Bot smiles (on the LCD) and moves to the *Releasing* behavior.

8.1.5 Releasing

The *Releasing* behavior is similar to *Grabbing*. Courier-Bot stops, then opens it gripper, moves backward, stops, closes its gripper, turns around, then switches back to the *Wandering* behavior, in search of a new object.

9 Experimental Layout and Results

Courier-Bot has been tested in several environments, and works at least moderately well in carpeted, hard-floored, bright, and dim environments. When on a rougher surface, such as a carpet floor, the extra load on the motors and friction on the caster cause Courier-Bot to move slowly, but when on a smooth floor, Courier-Bot moves rather quickly. Moving on a rougher floor can actually be an advantage, because when Courier-Bot moves too quickly, it is worse at detecting objects, and is especially worse at detecting beacons.

Lighting conditions also affect Courier-Bot's performance. Bright or dim lighting can impact the performance of the color sensor. Courier-Bot has a bright white LED light providing lighting for the color sensor. Before this LED was added, the color sensor was unreliable, and lighting was a major problem. Since it has been added, however, the color sensor is much more robust, and behaves well in a range of lighting conditions. Lighting can also affect the performance of the infrared beacon sensor. Too much stray infrared light, or even visible light can cause the sensor to be noisy or saturated, causing it to not detect the beacons. A light shield was added to the photodiode in the sensor to block light except that coming from directly in front of Courier-Bot at the height of the beacons, which helps considerably.

Overall, Courier-Bot's performance is good. The robot's actuation works very well, and it executes its behaviors perfectly, but its sensors are less robust. The sonar used to detect objects often does not detect them, and the beacon sensor often does not detect beacons very well, especially when the robot is moving fast, but given enough time in a closed environment, Courier-Bot will eventually find objects and beacons, and perform all of its behaviors.

10 Conclusion

Courier-Bot is a complex system made up of many parts. Each part was designed and tested in the course of making Courier-Bot. Courier-Bot's physical platform, including the mechanical gripper, was designed using CAD software, the parts were cut, and Courier-Bot was assembled. Courier-Bot's XMEGA microcontroller was interfaced with an ATMEGA, which was programmed to control a graphical LCD. Courier-Bot's microcontroller was interfaced electronically with the motors and servo, and was then programmed to control them. Sonar rangefinders and bump sensors were added to Courier-Bot, and Courier-Bot was taught obstacle avoidance. Infrared beacons were designed, and an analog beacon sensor was designed and interfaced with Courier-Bot's microcontroller. A color sensor was added and interfaced with Courier-Bot's microcontroller. Finally, Courier-Bot was programmed to perform several behaviors which allow it to find objects and carry them to different places based on their color.

Courier-Bot works well, but could be improved. Specifically, Courier-Bot's weaknesses are his difficulty in object detection and finding beacons depending on the environment. The sonar sensor and beacon detector sensor work well, but could be better used in the system. Specifically, the wandering and searching behaviors could be modified to produce movements that let Courier-Bot better use the sensor data it can acquire.

IMDL has been a rewarding class. Courier-Bot has been a success in that it is a complicated system that has been successfully designed and constructed and that works. It is also a success in that it represents a great deal of learning about mechanical design, robotics, and a great deal of execution of knowledge in electrical engineering.

References

[1] Atmel, "AVR1300: Using the Atmel AVR XMEGA ADC", datasheet, 2010, `http://www.atmel.com/dyn/resources/prod\_documents/doc8032.pdf`

[2] Atmel, "AVR XMEGA A1 Device Datasheet", datasheet, 2010, `http://www.atmel.com/dyn/resources/prod\_documents/doc8067.pdf`

[3] Atmel, "AVR XMEGA A Manual", datasheet, 2010, `http://www.atmel.com/dyn/resources/prod\_documents/doc8077.pdf`

[4] Avago, "ADJD-S371-QR999 Miniature Surface-Mount RGB Digital Color Sensor Module", datasheet, 2007.

[5] MaxBotix, "LV-MaxSonar-EZ0 High Performance Sonar Range Finder", datasheet, 2007, http://www.maxbotix.com/uploads/LV-MaxSonar-EZ0-Datasheet.pdf

[6] MaxBotix, "LV-MaxSonar-EZ3 High Performance Sonar Range Finder", datasheet, 2007, http://www.maxbotix.com/uploads/LV-MaxSonar-EZ3-Datasheet.pdf

[7] Pololu, Gearmotor Specifications Page http://www.pololu.com/catalog/product/1585/specs

[8] Pololu, Wheel Specifications Page http://www.pololu.com/catalog/product/1435/specs

[9] Servo City, HS-635B Specifications Page http://servocity.com/html/hs-635hb_high_torque.html

[10] STMicroelectronics, L298, datasheet, 2000, http://www.st.com/stonline/books/pdf/docs/1773.pdf

[11] Tony van Roon, "555 Timer Tutorial", 1995, http://www.sentex.ca/~mec1995/gadgets/555/555.html

A Program Code

The following source code files are included in this appendix:

Listing 1: code/courierbot/courierbot.c

```
1  #include <avr/io.h>
2  #define F_CPU 32000000UL
3  #include <util/delay.h>
4  #include <avr/interrupt.h>
5  #include "lcdurt.h"       // LCD control via USART to atmega32
6  #include "servo.h"        // Servo driver
7  #include "motor.h"        // Motor driver
8  #include "adc.h"          // Driver for everything ADC
9  #include "ir.h"           // Driver for IR beacon sensor
10 #include "switches.h"     // Bump sensors & other switches
11 #include "colorsensor.h"  // Color sensor
12
13 // Uncomment any one of these to enable test code
14 //#define SERVO_TEST
15 //#define COLOR_SENSOR_TEST
16 //#define SPECIAL_SENSOR_TEST
17 //#define SONAR_TEST
18
19 // Uncommenting this line will enable a more complex display
20 // during normal operation
21 //#define DEBUG_DISPLAY
22
23 #define DBGPORT PORTQ
24 #define DBGPIN  1
25
```

```
26 #define CLAW_SERVO 5
27
28 #define SONAR_THRESHOLD 0x1400 // 5120
29 #define SONAR_HIGHER_THRESHOLD 0x1900 // 6400
30 #define SONAR_LOWER_THRESHOLD 0x1300 //4864
31
32 #define WAIT_BEHAVIOR 0
33 #define OBSTACLE_AVOID_BEHAVIOR 1
34 #define SIMPLE_OBSTACLE_AVOID_BEHAVIOR 2
35 #define WANDER_BEHAVIOR 3
36 #define GRAB_BEHAVIOR 4
37 #define RELEASE_BEHAVIOR 5
38 #define SEARCH_BEHAVIOR 6
39
40 char *behaviornames = "ZOVWGRS";
41 char *longbehaviornames[] = {"Waiting", "Obst. Avd.", "S Obst. Avd.",
42                              "Wandering", "Grabbing", "Releasing", "Searching"};
43
44 #define STARTING_BEHAVIOR WAIT_BEHAVIOR
45
46 #define TICK_TIMER TCE1
47 #define TICK_VECT TCE1_CCA_vect
48 // This makes it one tick per millisecond
49 #define CLOCKS_PER_TICK 32000
50 // 50 millisecond period
51 #define MAIN_LOOP_TICKS 10
52
53 // ----- Global variables -----
54
55 // Current behavior to exhibit
56 uint8_t behavior;
57
58 // A timer tick. Updated in timer interrupt;
59 // reset in main.
60 volatile uint8_t tickval;
61
62 // Sensor data
63 uint16_t lsonar, rsonar, csonar, dsonar;
64 uint8_t switchData;
65 char ircolor; // 'r', 'g', 'b', or '\0' based on sensors
66
67 // Whether we're carrying something
68 uint8_t carrying;
69
70 // What color the search behavior is looking for
71 char searchcolor;
72
73 // What color the sensor sees right now
74 char sensorcolor;
75
76 // State of PRNG
77 unsigned long randnext = 1;
78
79 // LCD Face stuff
80 char lookdir = 'F';
81 char emotion = 'H';
82
83
84 // ----- ISRs -----
85
86 ISR(TICK_VECT) {
87   tickval++;
88 }
89
90 // ----- Helper functions -----
91
92 // Get a 16-bit random number
93 // This is a relatively well-known
```

```
94 // pseudo-random number generator algorithm.
95 uint16_t myrand() {
96   randnext = randnext * 1103515245 + 12345;
97   return((unsigned)(randnext >> 16));
98 }
99
100 void mysrand(unsigned long seed) {
101   randnext = seed;
102 }
103
104 // Get a single random bit
105 uint8_t randBit() {
106   return (pollADC(1) >> 4) & 1;
107 }
108
109 // Format sonar data in a short, fuzzy, human-readable way
110 void formatSonarDisplay(char *sonardisplaybuf) {
111   if(lsonar < SONAR_LOWER_THRESHOLD) {
112     sonardisplaybuf[0] = 'X';
113   } else if(lsonar < SONAR_THRESHOLD) {
114     sonardisplaybuf[0] = 'x';
115   } else if(lsonar < SONAR_HIGHER_THRESHOLD) {
116     sonardisplaybuf[0] = '.';
117   } else {
118     sonardisplaybuf[0] = ' ';
119   }
120
121   if(csonar < SONAR_LOWER_THRESHOLD) {
122     sonardisplaybuf[1] = 'X';
123   } else if(csonar < SONAR_THRESHOLD) {
124     sonardisplaybuf[1] = 'x';
125   } else if(csonar < SONAR_HIGHER_THRESHOLD) {
126     sonardisplaybuf[1] = '.';
127   } else {
128     sonardisplaybuf[1] = ' ';
129   }
130
131   if(dsonar < SONAR_LOWER_THRESHOLD) {
132     sonardisplaybuf[2] = 'X';
133   } else if(dsonar < SONAR_THRESHOLD) {
134     sonardisplaybuf[2] = 'x';
135   } else if(dsonar < SONAR_HIGHER_THRESHOLD) {
136     sonardisplaybuf[2] = '.';
137   } else {
138     sonardisplaybuf[2] = ' ';
139   }
140
141   if(rsonar < SONAR_LOWER_THRESHOLD) {
142     sonardisplaybuf[3] = 'X';
143   } else if(rsonar < SONAR_THRESHOLD) {
144     sonardisplaybuf[3] = 'x';
145   } else if(rsonar < SONAR_HIGHER_THRESHOLD) {
146     sonardisplaybuf[3] = '.';
147   } else {
148     sonardisplaybuf[3] = ' ';
149   }
150
151   sonardisplaybuf[4] = 0;
152 }
153
154
155 // ----- Behavior loops -----
156
157 // Wait until bump sensor is hit, then wander.
158 void waitLoop() {
159   releaseMotors();
160   emotion = 'N';
161   if(switchData & (BLBUMP_PIN | BRBUMP_PIN)) {
```

```
162     enableMotors();
163     emotion = 'H';
164     behavior = WANDER_BEHAVIOR;
165   }
166 }
167
168 // Works surprisingly well.     [not used]
169 void simpleObstacleAvoidLoop() {
170   if(lsonar < SONAR_THRESHOLD) {
171     setMotor(RIGHT_MOTOR, -MOTOR_MAX_SPEED);
172   }
173   else {
174     setMotor(RIGHT_MOTOR, MOTOR_MAX_SPEED);
175   }
176   if(rsonar < SONAR_THRESHOLD) {
177     setMotor(LEFT_MOTOR, -MOTOR_MAX_SPEED);
178   }
179   else {
180     setMotor(LEFT_MOTOR, MOTOR_MAX_SPEED);
181   }
182 }
183
184 // More complex.        [not used]
185 void obstacleAvoidLoop() {
186   static uint16_t turnLeftC = 0;
187   static uint16_t turnRightC = 0;
188   static uint16_t backupC = 0;
189   static uint8_t confuseda = 0;
190   static uint8_t confusedb = 0;
191
192   // Interpret sensor data
193
194   // Center or both left & right; backup then turn random direction
195   if(switchData & BRBUMP_PIN) {
196     backupC = 0;
197     turnLeftC = 50;
198     turnRightC = 0;
199   } else if(switchData & BLBUMP_PIN) {
200     backupC = 0;
201     turnLeftC = 0;
202     turnRightC = 50;
203   } else if((csonar < SONAR_THRESHOLD) ||
204             ((lsonar < SONAR_THRESHOLD) && (rsonar < SONAR_THRESHOLD))) {
205     backupC = 125;
206     if(randBit()) {
207       turnLeftC = 75;
208       turnRightC = 0;
209     } else {
210       turnLeftC = 0;
211       turnRightC = 75;
212     }
213   } else if(lsonar < SONAR_THRESHOLD) {
214     confuseda = 1;
215     turnRightC = 50;
216   } else if(rsonar < SONAR_THRESHOLD) {
217     confusedb = 1;
218     turnLeftC = 50;
219   }
220
221   if(confuseda && confusedb) {
222     turnLeftC = 600;
223     turnRightC = 0;
224     confuseda = 0;
225     confusedb = 0;
226   }
227
228   // Act
229   if(backupC > 0) {
```

```
230     setMotor(BOTH_MOTORS, -MOTOR_MAX_SPEED);
231     backupC--;
232   } else if(turnLeftC > 0) {
233     setMotor(LEFT_MOTOR, -MOTOR_MAX_SPEED);
234     setMotor(RIGHT_MOTOR, MOTOR_MAX_SPEED);
235     turnLeftC--;
236   } else if(turnRightC > 0) {
237     setMotor(LEFT_MOTOR, MOTOR_MAX_SPEED);
238     setMotor(RIGHT_MOTOR, -MOTOR_MAX_SPEED);
239     turnRightC--;
240   } else {
241     confuseda = 0;
242     confusedb = 0;
243     setMotor(BOTH_MOTORS, MOTOR_MAX_SPEED);
244   }
245 }
246
247 // Based on obstacleAvoidLoop
248 // This isn't a loop, but it reccomends motor values
249 // for doing obstacle avoidance. It is used as a fallback
250 // behavior for other behaviors.
251 void obstacleAvoid(int16_t *lrec, int16_t *rrec, int8_t *avoiding) {
252   static uint16_t turnLeftC = 0;
253   static uint16_t turnRightC = 0;
254   static uint16_t backupC = 0;
255   static uint8_t confuseda = 0;
256   static uint8_t confusedb = 0;
257
258   // Interpret sensor data
259
260   if(switchData & BRBUMP_PIN) {
261     backupC = 0;
262     turnLeftC = 20;
263     turnRightC = 0;
264   } else if(switchData & BLBUMP_PIN) {
265     backupC = 0;
266     turnLeftC = 0;
267     turnRightC = 20;
268   } else if((csonar < SONAR_THRESHOLD) ||
269             ((lsonar < SONAR_THRESHOLD) && (rsonar < SONAR_THRESHOLD))) {
270     backupC = 100;
271     if(randBit()) {
272       turnLeftC = 50;
273       turnRightC = 0;
274     } else {
275       turnLeftC = 0;
276       turnRightC = 50;
277     }
278   } else if(lsonar < SONAR_THRESHOLD) {
279     confuseda = 1;
280     turnRightC = 20;
281   } else if(rsonar < SONAR_THRESHOLD) {
282     confusedb = 1;
283     turnLeftC = 20;
284   }
285
286   if(confuseda && confusedb) {
287     turnLeftC = 60;
288     turnRightC = 0;
289     confuseda = 0;
290     confusedb = 0;
291   }
292
293   // Act
294   *avoiding = 1;
295   if(backupC > 0) {
296     *lrec = -MOTOR_MAX_SPEED;
297     *rrec = -MOTOR_MAX_SPEED;
```

```
298       backupC--;
299     } else if(turnLeftC > 0) {
300       *lrec = -MOTOR_MAX_SPEED;
301       *rrec = MOTOR_MAX_SPEED;
302       turnLeftC--;
303       lookdir = 'R';
304     } else if(turnRightC > 0) {
305       *lrec = MOTOR_MAX_SPEED;
306       *rrec = -MOTOR_MAX_SPEED;
307       turnRightC--;
308       lookdir = 'L';
309     } else {
310       confuseda = 0;
311       confusedb = 0;
312       *lrec = MOTOR_MAX_SPEED;
313       *rrec = MOTOR_MAX_SPEED;
314       *avoiding = 0;
315       lookdir = 'F';
316     }
317 }
318 
319 // Wander, pick things up if we see them
320 void wanderLoop() {
321   int16_t ool, oor;
322   int8_t avoiding, investigating;
323   static int8_t turnLeftC = 0, turnRightC = 0;
324   static int16_t wanderC = 0;
325   static uint8_t seeit = 0;
326 
327   obstacleAvoid(&ool, &oor, &avoiding);
328 
329   // Turn when not avoiding, every once in a while
330   if(wanderC != 0) {
331     wanderC--;
332   } else if((turnLeftC == 0) && (turnRightC == 0) && (!avoiding)) {
333     if(randBit()) {
334       turnLeftC = myrand() & 0x3F;
335       turnRightC = 0;
336     } else {
337       turnLeftC = 0;
338       turnRightC = myrand() & 0x3F;
339     }
340     wanderC = myrand();
341   }
342 
343   // See the object?
344   investigating = 0;
345   if((csonar > SONAR_HIGHER_THRESHOLD) &&
346      (dsonar < SONAR_THRESHOLD) &&
347      (lsonar > SONAR_THRESHOLD) &&
348      (rsonar > SONAR_THRESHOLD)) {
349     investigating = 1;
350     seeit++;
351   } else {
352     seeit = 0;
353   }
354 
355   if(seeit > 5) {
356     if(carrying == 0) {
357       // Grab it!
358       behavior = GRAB_BEHAVIOR;
359     } else {
360       // Carrying something already. Avoid it!
361       if(randBit()) {
362         turnLeftC = 30;
363         turnRightC = 0;
364       } else {
365         turnLeftC = 0;
```

```
366         turnRightC = 30;
367       }
368     }
369   }
370
371   if(investigating) {            // Don't avoid if we think there's an object here
372     setMotor(BOTH_MOTORS, MOTOR_MAX_SPEED); // Full speed ahead! (into the object)
373   } else if(avoiding) {          // Defer to avoidance
374     setMotor(LEFT_MOTOR, ool);
375     setMotor(RIGHT_MOTOR, oor);
376   } else {
377     if(turnLeftC > 0) {         // Avoidance for carrying or wander
378       setMotor(LEFT_MOTOR, -MOTOR_MAX_SPEED);
379       setMotor(RIGHT_MOTOR, MOTOR_MAX_SPEED);
380       turnLeftC--;
381     } else if(turnRightC > 0) {
382       setMotor(LEFT_MOTOR, MOTOR_MAX_SPEED);
383       setMotor(RIGHT_MOTOR, -MOTOR_MAX_SPEED);
384       turnRightC--;
385     } else {                    // Default moving forward
386       setMotor(LEFT_MOTOR, MOTOR_MAX_SPEED);
387       setMotor(RIGHT_MOTOR, MOTOR_MAX_SPEED);
388     }
389   }
390 }
391
392 // Search for a beacon
393 void searchLoop() {
394   int16_t ool, oor;
395   int8_t avoiding;
396   static int16_t countb = 100;
397   static int8_t running = 0;
398
399   obstacleAvoid(&ool, &oor, &avoiding);
400
401   if(running == 0) {
402     // Only pick the color to look for NOW
403     searchcolor = sensorcolor;
404     running = 1;
405   }
406
407   emotion = 'N';
408   if(ircolor == searchcolor) {
409     // Show happy face when we see the beacon
410     emotion = 'H';
411
412     // Assume this means we've reached the beacon...
413     if((csonar < SONAR_HIGHER_THRESHOLD) && (dsonar < SONAR_THRESHOLD)) {
414       running = 0;
415       behavior = RELEASE_BEHAVIOR;
416       return;
417     }
418
419     // See the beacon... go for it!
420     if(!avoiding) {
421       setMotor(LEFT_MOTOR, MOTOR_MAX_SPEED);
422       setMotor(RIGHT_MOTOR, MOTOR_MAX_SPEED);
423       return;
424     }
425
426   } else if(avoiding) {              // Defer to avoidance
427     setMotor(LEFT_MOTOR, ool);
428     setMotor(RIGHT_MOTOR, oor);
429
430   } else {
431
432     // Every so often just turn around
433     countb--;
```

```
434     if(countb == 0) {
435       countb = 800;
436     } else if(countb < 300) {
437       setMotor(LEFT_MOTOR, -MOTOR_MAX_SPEED);
438       setMotor(RIGHT_MOTOR, MOTOR_MAX_SPEED);
439     } else {
440       setMotor(BOTH_MOTORS, MOTOR_MAX_SPEED);
441     }
442
443   }
444 }
445
446 // These macros make it a lot cleaner and easier to write
447 // these behavior loop functions when the behavior consists of
448 // a sequence of smaller behaviors that last a certain
449 // number of ticks, then leads to a different behavior.
450 #define LOOPSEQ_A( num, length ) if(seqno == (num)) {           \
451                                    if(count < (length)) {
452
453 #define LOOPSEQ_B( num, length )   count++;                     \
454                                    }                            \
455                                    else {                       \
456                                      count = 0; seqno++;        \
457                                    }                            \
458                                  } else if(seqno == (num)) {    \
459                                    if(count < (length)) {
460
461 #define LOOPSEQ_C( bhvr )          count++;                     \
462                                    } else {                     \
463                                      count = 0; seqno = 0;      \
464                                      behavior = (bhvr);         \
465                                    }                            \
466                                  }
467
468 // Grab an object
469 void grabLoop() {
470   static uint16_t count = 0;
471   static uint8_t seqno = 0;
472
473   LOOPSEQ_A(0, 1)
474     releaseMotors();                                            // Stop
475   LOOPSEQ_B(1, 360)
476     setServoAngle(CLAW_SERVO, 90 - ((int16_t)count / 2));       // Open
477   LOOPSEQ_B(2, 1)
478     enableMotors();                                             // Forward
479     setMotor(BOTH_MOTORS, MOTOR_MAX_SPEED);
480   LOOPSEQ_B(3, 150)
481                                                                 // Moving forward
482   LOOPSEQ_B(4, 1)
483     releaseMotors();                                            // Stop
484   LOOPSEQ_B(5, 300)
485     setServoAngle(CLAW_SERVO, 90 - ((int16_t)(300-count) / 2)); // Close
486   LOOPSEQ_B(6, 1)
487     carrying = 1;                                               // Back
488     enableMotors();
489     setMotor(BOTH_MOTORS, -MOTOR_MAX_SPEED);
490   LOOPSEQ_B(7, 100)
491                                                                 // Moving backward
492
493   LOOPSEQ_B(8, 1)
494     setServoAngle(CLAW_SERVO, 75);                              // Loosen claw
495
496   LOOPSEQ_C(SEARCH_BEHAVIOR)                                    // Go to SEARCH
497 }
498
499 // Release an object
500 void releaseLoop() {
501   static uint16_t count = 0;
```

```
502   static uint8_t seqno = 0;
503
504   LOOPSEQ_A(0, 1)
505     releaseMotors();                                          // Stop
506   LOOPSEQ_B(1, 300)
507     setServoAngle(CLAW_SERVO, 90 - ((int16_t)count / 2));     // Open
508   LOOPSEQ_B(2, 1)
509     enableMotors();                                           // Go back
510     setMotor(BOTH_MOTORS, -MOTOR_MAX_SPEED);
511   LOOPSEQ_B(3, 150)
512                                                               // Moving backward
513   LOOPSEQ_B(4, 1)
514     releaseMotors();                                          // Stop
515   LOOPSEQ_B(5, 300)
516     setServoAngle(CLAW_SERVO, 90 - ((int16_t)(300-count) / 2)); // Close
517   LOOPSEQ_B(6, 1)
518     carrying = 0;                                             // Turn
519     enableMotors();
520     setMotor(LEFT_MOTOR, -MOTOR_MAX_SPEED);
521     setMotor(RIGHT_MOTOR, MOTOR_MAX_SPEED);
522   LOOPSEQ_B(7, 200)
523                                                               // Turning
524
525   LOOPSEQ_B(8, 1)
526     setServoAngle(CLAW_SERVO, 75);                            // Loosen claw
527
528   LOOPSEQ_C(WANDER_BEHAVIOR)                                  // Go to WANDER
529 }
530
531
532 // ----- Main -----
533
534 int main() {
535   uint8_t i, ii;
536   uint16_t displayctr;
537   uint16_t rir, gir, bir;
538   uint16_t colorr, colorg, colorb, colorc;
539   char lastlookdir, lookbuf[4];
540   char lastemotion, emotionbuf[4];
541   char sonardisplaybuf[5];
542
543   cli();
544
545   // Set main clock frequency to 32MHz
546   DFLLRC32M.CTRL = 1; // Turn on 32MHz DFLL
547   CCP = CCP_IOREG_gc; // Security
548   OSC.CTRL |= (OSC_RC32KEN_bm | OSC_RC32MEN_bm); // Turn on 32.786kHz and 32 MHz oscillators
549   while((OSC.STATUS & (OSC_RC32KRDY_bm | OSC_RC32MRDY_bm)) !=
550         (OSC_RC32KRDY_bm | OSC_RC32MRDY_bm)); // Wait for them to stabilize
551   CCP = CCP_IOREG_gc; // Security
552   CLK.CTRL = 1; // Select 32MHz clock
553   CLK.PSCTRL = 0; // Don't divide peripheral clock
554
555   // Enable all interrupt priority levels
556   PMIC.CTRL |= 0x7;
557
558   // Set up debug port
559   DBGPORT.DIR |= DBGPIN;
560   DBGPORT.OUT |= DBGPIN;
561
562   // Set up tick interrupt
563   TICK_TIMER.CTRLA = RTC_PRESCALER_DIV1_gc;
564   TICK_TIMER.CTRLB = 0;
565   TICK_TIMER.PER = CLOCKS_PER_TICK;
566   TICK_TIMER.INTCTRLB = 1;
567   tickval = 0;
568
569   // Set up USART for LCD
```

```
570   lcdurtInit();
571
572   // Set up servos
573   initServos();
574
575   // Set up motors
576   initMotors();
577
578   // Set up ADC
579   initADC();
580
581   // Set up IR sensor
582   initIR();
583   rir = 0;
584   gir = 0;
585   bir = 0;
586   ircolor = 0;
587   searchcolor = 'r';
588
589   // Set up switches
590   initSwitches();
591
592   // Set up color sensor
593   initColorSensor();
594   colorr = 0;
595   colorg = 0;
596   colorb = 0;
597   colorc = 0;
598   colorSensorLEDon();
599
600   // Set up sonars
601   // Wait for sensors to warm up
602   _delay_ms(300);
603   // Start them up, then back off
604   PORTQ.DIR |= 0x02;
605   PORTQ.OUT &= 0xFD;
606   _delay_ms(1);
607   PORTQ.OUT |= 0x02;
608   _delay_ms(1);
609   PORTQ.OUT &= 0xFD;
610   PORTQ.DIR &= 0xFD;
611
612   // Close claw
613   setServoAngle(CLAW_SERVO, 90);
614
615   // Start out not carrying anything
616   carrying = 0;
617
618   // LCD Face stuff
619   lastlookdir = lookdir;
620   lastemotion = emotion;
621
622   // Seed random number generator
623   mysrand(pollADC(1));
624
625   // Take a deep breath...
626   _delay_ms(1000);
627
628   sei();
629
630   behavior = STARTING_BEHAVIOR;
631
632   // *** Start motors ***
633   setMotor(BOTH_MOTORS, MOTOR_MAX_SPEED);
634   enableMotors();
635
636   DBGPORT.OUT &= ~DBGPIN;
637   displayctr = 0;
```

```
638
639
640   // *** Test code ***
641
642 #ifdef SERVO_TEST
643   setServoAngle(CLAW_SERVO, -60);
644   _delay_ms(2000);
645   setServoAngle(CLAW_SERVO, 0);
646   _delay_ms(2000);
647   setServoAngle(CLAW_SERVO, 90);
648   _delay_ms(2000);
649   setServoAngle(CLAW_SERVO, 0);
650   _delay_ms(2000);
651   setServoAngle(CLAW_SERVO, -60);
652   _delay_ms(2000);
653   setServoAngle(CLAW_SERVO, 0);
654   _delay_ms(2000);
655   setServoAngle(CLAW_SERVO, 90);
656   _delay_ms(2000);
657   setServoAngle(CLAW_SERVO, 75);
658   _delay_ms(2000);
659 #endif
660
661 #ifdef COLOR_SENSOR_TEST
662   releaseMotors();
663   ii = 0;
664   behavior = GRAB_BEHAVIOR;
665   while(1) {
666
667     //pollColorSensor(&colorr, &colorg, &colorb, &colorc);
668     sensorcolor = updateColorSensor(&colorr, &colorg, &colorb, &colorc);
669
670     if(displayctr++ > 5) {
671       lcdPrintf("%4hx%4hX%4hx%4hx.%c.%02hhx",
672           colorr,
673           colorg,
674           colorb,
675           colorc,
676           sensorcolor,
677           ii++);
678       displayctr = 0;
679     }
680
681     /*
682     // Repeat GRAB -> WAIT -> RELEASE -> GRAB -> WAIT -> RELEASE
683     if(behavior == GRAB_BEHAVIOR) {
684       grabLoop();
685     } else if(behavior == RELEASE_BEHAVIOR) {
686       releaseLoop();
687     } else if(behavior == SEARCH_BEHAVIOR) {
688       behavior = WAIT_BEHAVIOR;
689     } else if(behavior == WANDER_BEHAVIOR) {
690       behavior = GRAB_BEHAVIOR;
691     } else {
692       behavior = WAIT_BEHAVIOR;
693       waitLoop();
694     }
695     */
696
697     updateMotors();
698
699     // Wait until it is time for the next loop
700     while(1) {
701       cli();
702       if(tickval >= MAIN_LOOP_TICKS) {
703         tickval = 0;
704         break;
705       }
```

```
706       sei();
707       for(i = 0; i < 10; i++) {
708         asm volatile("nop");
709       }
710     }
711     sei();
712   }
713 #endif
714
715 #ifdef SPECIAL_SENSOR_TEST
716   releaseMotors();
717   ii = 0;
718   emotion = 'T';
719   while(1) {
720
721     ircolor = updateIRs(&rir, &gir, &bir);
722
723     if(ircolor != 0) {
724       emotion = 'H';
725     } else {
726       emotion = 'T';
727     }
728
729     if(displayctr++ > 5) {
730       lcdPrintf("%4hx %4hX %4hx %c %02hhx",
731           rir,
732           gir,
733           bir,
734           (ircolor == 0) ? '_' : ircolor,
735           ii++);
736
737       if(lastemotion != emotion) {
738         lastemotion = emotion;
739         emotionbuf[0] = 'E';
740         emotionbuf[1] = emotion;
741         emotionbuf[2] = '\n';
742         emotionbuf[3] = 0;
743         lcdSendStr(emotionbuf);
744       }
745
746       displayctr = 0;
747     }
748
749     updateMotors();
750
751     // Wait until it is time for the next loop
752     while(1) {
753       cli();
754       if(tickval >= MAIN_LOOP_TICKS) {
755         tickval = 0;
756         break;
757       }
758       sei();
759       for(i = 0; i < 10; i++) {
760         asm volatile("nop");
761       }
762     }
763     sei();
764   }
765 #endif
766
767 #ifdef SONAR_TEST
768   releaseMotors();
769   ii = 0;
770   while(1) {
771
772     lsonar = pollADC(1);
773     csonar = pollADC(2);
```

```
774     dsonar = pollADC(3);
775     rsonar = pollADC(4);
776 
777     ircolor = updateIRs(&rir, &gir, &bir);
778     sensorcolor = updateColorSensor(&colorr, &colorg, &colorb, &colorc);
779 
780     if(displayctr++ > 5) {
781       lcdPrintf("%3hhd%3hhd%3hhd%3hhd %02x%c%c%c%02hhx",
782           lsonar >> 8,
783           csonar >> 8,
784           dsonar >> 8,
785           rsonar >> 8,
786           switchData,
787           ((ircolor == 0) ? '_' : ircolor),
788           sensorcolor,
789           '?',
790           ii++);
791       displayctr = 0;
792     }
793 
794     updateMotors();
795 
796     // Wait until it is time for the next loop
797     while(1) {
798       cli();
799       if(tickval >= MAIN_LOOP_TICKS) {
800         tickval = 0;
801         break;
802       }
803       sei();
804       for(i = 0; i < 10; i++) {
805         asm volatile("nop");
806       }
807     }
808     sei();
809   }
810 #endif
811 
812 
813   // *** Main loop ***
814 
815   ii =0;
816   while(1) {
817     // Get sensor data
818 
819     // Sonars
820     lsonar = pollADC(1);
821     csonar = pollADC(2);
822     dsonar = pollADC(3);
823     rsonar = pollADC(4);
824 
825     // Switches
826     switchData = pollSwitches();
827 
828     // Color sensor
829     sensorcolor = updateColorSensor(&colorr, &colorg, &colorb, &colorc);
830 
831     // IR beacon sensor
832     ircolor = updateIRs(&rir, &gir, &bir);
833 
834     // Update display
835     if(displayctr++ > 5) {
836 #ifdef DEBUG_DISPLAY
837       lcdPrintf("%3hhd%3hhd%3hhd%3hhd %02x%c%c%c%02hhx",
838           lsonar >> 8,
839           csonar >> 8,
840           dsonar >> 8,
841           rsonar >> 8,
```

```
842           switchData,
843           ((ircolor == 0) ? '_' : ircolor),
844           sensorcolor,
845           '?',
846           ii++);
847       displayctr = 0;
848 #else
849       formatSonarDisplay(sonardisplaybuf);
850 
851       if(behavior == SEARCH_BEHAVIOR) {
852         if(searchcolor == 'r') {
853           lcdPrintf("%s %s - RED", sonardisplaybuf, longbehaviornames[behavior]);
854         } else if(searchcolor == 'g') {
855           lcdPrintf("%s %s - GREEN", sonardisplaybuf, longbehaviornames[behavior]);
856         } else if(searchcolor == 'b') {
857           lcdPrintf("%s %s - BLUE", sonardisplaybuf, longbehaviornames[behavior]);
858         }
859       } else {
860         lcdPrintf("%s %s", sonardisplaybuf, longbehaviornames[behavior]);
861       }
862 #endif
863       if(lastlookdir != lookdir) {
864         lastlookdir = lookdir;
865         lookbuf[0] = 'L';
866         lookbuf[1] = lookdir;
867         lookbuf[2] = '\n';
868         lookbuf[3] = 0;
869         lcdSendStr(lookbuf);
870       }
871       if(lastemotion != emotion) {
872         lastemotion = emotion;
873         emotionbuf[0] = 'E';
874         emotionbuf[1] = emotion;
875         emotionbuf[2] = '\n';
876         emotionbuf[3] = 0;
877         lcdSendStr(emotionbuf);
878       }
879       displayctr = 0;
880     }
881 
882     // Execute current behavior
883     switch(behavior) {
884 
885       case OBSTACLE_AVOID_BEHAVIOR:
886         obstacleAvoidLoop();
887         break;
888 
889       case SIMPLE_OBSTACLE_AVOID_BEHAVIOR:
890         simpleObstacleAvoidLoop();
891         break;
892 
893       case WANDER_BEHAVIOR:
894         wanderLoop();
895         break;
896 
897       case GRAB_BEHAVIOR:
898         grabLoop();
899         break;
900 
901       case RELEASE_BEHAVIOR:
902         releaseLoop();
903         break;
904 
905       case SEARCH_BEHAVIOR:
906         searchLoop();
907         break;
908 
909       default:
```

```
910         waitLoop();
911
912     }
913
914     // Let motor driver do filtering
915     updateMotors();
916
917     // Wait until it is time for the next loop
918     DBGPORT.OUT |= DBGPIN;
919     while(1) {
920       cli();
921       if(tickval >= MAIN_LOOP_TICKS) {
922         tickval = 0;
923         break;
924       }
925       sei();
926       for(i = 0; i < 10; i++) {
927         asm volatile("nop");
928       }
929     }
930     sei();
931     DBGPORT.OUT &= ~DBGPIN;
932   }
933 }
```

Listing 2: code/courierbot/adc.h

```
1 #ifndef ADC_H
2 #define ADC_H
3
4 void initADC();
5 uint16_t pollADC(uint8_t id);
6
7 #endif
```

Listing 3: code/courierbot/adc.c

```
1  #include <avr/io.h>
2  #include <avr/pgmspace.h>
3  #include <stddef.h>
4  #include "adc.h"
5
6
7  void initADC() {
8    PORTA.DIR = 0; // All port A is inputs
9    // Set ADC control registers
10   ADCA.CTRLB = 0x06; // Unsigned, NO free-run, 12-bit, left-adjusted
11   ADCA.REFCTRL = 0x20; // Port A reference voltage; Bandgap & Temp DISABLED
12   ADCA.EVCTRL = 0x00; // Sweep only channel 0 on event or free-run mode; events unused
13   ADCA.PRESCALER = 2; // Peripheral clk / 16 (fastest - 2MHz at sys clk = 32MHz)
14   // Set Channel control registers
15   ADCA.CH0.CTRL = 0x01; // Gain=1; Use external inputs
16   ADCA.CH0.INTCTRL = 0; // No interrupts
17
18   ADCA.CTRLA = 1; // Enable the ADC (do this last)
19 }
20
21 // Poll ADC values. Results are 12-bit, left-aligned
22 uint16_t pollADC(uint8_t id) {
23   uint16_t result;
24   if((id < 1) || (id > 15))
25     return 0xFFFF;
26   if(id < 8) {
27     ADCA.CH0.MUXCTRL = (id << 3);
28     ADCA.CH0.CTRL |= 0x80; // Start conversion
29     while((ADCA.CH0.INTFLAGS & 0x01) != 0x01); // Wait for conversion
30     ADCA.CH0.INTFLAGS |= 1; // Clear interrupt flag (Yes, really. Intuitive!)
```

```
    result = ADCA.CH0.RES;
    if(result > 210)
      result -= 210;
    else
      result = 0;
    return result;
  } else {
    return 0xFFFF; // For now...
  }
}
```

Listing 4: code/courierbot/colorsensor.h

```
#ifndef COLORSENSOR_H

#define COLORTWI TWIF
// This is the 7-bit address (0x74) shifted left by 1
// so that the R/W bit can be OR'd in
#define COLORSENSORADDR 0xE8
#define COLORPORT PORTF
#define COLORLEDPIN 4

// These set relative gains.
// Higer CAP value leads to lower reading.
// Higher INT value leads to higer reading.
#define CAPRVAL 15
#define CAPGVAL 10
#define CAPBVAL 3
#define CAPCVAL 14
#define INTRVAL 0x1FF
#define INTGVAL 0x1FF
#define INTBVAL 0x1FF
#define INTCVAL 0x1FF


#define CS_CTRL            0x00
#define CS_CONFIG          0x01
#define CS_CAP_RED         0x06
#define CS_CAP_GREEN       0x07
#define CS_CAP_BLUE        0x08
#define CS_CAP_CLEAR       0x09
#define CS_INT_RED_LO      0x0A
#define CS_INT_RED_HI      0x0B
#define CS_INT_GREEN_LO    0x0C
#define CS_INT_GREEN_HI    0x0D
#define CS_INT_BLUE_LO     0x0E
#define CS_INT_BLUE_HI     0x0F
#define CS_INT_CLEAR_LO    0x10
#define CS_INT_CLEAR_HI    0x11

#define CS_DATA_RED_LO     0x40
#define CS_DATA_RED_HI     0x41
#define CS_DATA_GREEN_LO   0x42
#define CS_DATA_GREEN_HI   0x43
#define CS_DATA_BLUE_LO    0x44
#define CS_DATA_BLUE_HI    0x45
#define CS_DATA_CLEAR_LO   0x46
#define CS_DATA_CLEAR_HI   0x47

#define CS_OFFSET_RED      0x48
#define CS_OFFSET_GREEN    0x49
#define CS_OFFSET_BLUE     0x4A
#define CS_OFFSET_CLEAR    0x4B

void initColorSensor();
void pollColorSensor(uint16_t *red, uint16_t *green, uint16_t *blue, uint16_t *clear);
char updateColorSensor(uint16_t *rr, uint16_t *gr, uint16_t *br, uint16_t *cr);
#define colorSensorLEDon() (COLORPORT.OUT |= COLORLEDPIN)
```

```
#define colorSensorLEDoff() (COLORPORT.OUT &= (~COLORLEDPIN))

#endif
```

Listing 5: code/courierbot/colorsensor.c

```
#include <avr/io.h>
#include "colorsensor.h"

uint8_t readCsReg(uint8_t addr) {
  uint8_t data;
  // Wolud need to wait for the bus to be idle if there
  // were other masters on the bus

  // Send start condition, write, and color sensor address
  COLORTWI.MASTER.ADDR = COLORSENSORADDR;
  // Assume no contention; wait for write interrupt flag
  while((COLORTWI.MASTER.STATUS & 0x40) == 0);
  //if((COLORTWI.MASTER.STATUS & 0x10) != 0)            << Would check for error

  // Send address of register
  COLORTWI.MASTER.DATA = addr;
  // Assume the sensor won't NACK; wait for flag
  while((COLORTWI.MASTER.STATUS & 0x40) == 0);
  //if((COLORTWI.MASTER.STATUS & 0x10) != 0)            << Would check for error

  // Send repeated start condition, read, and color sensor address
  COLORTWI.MASTER.ADDR = (COLORSENSORADDR | 0x01);

  // Wait for Read _or_ Write flag (write flag would mean error;
  // watch for it at the very least just to avoid hanging).
  while((COLORTWI.MASTER.STATUS & 0xC0) == 0);
  //if((COLORTWI.MASTER.STATUS & 0x10) != 0)            << Would check for error

  // Get the data
  data = COLORTWI.MASTER.DATA;
  // Send NACK and stop condition
  COLORTWI.MASTER.CTRLC |= 0x04;
  COLORTWI.MASTER.CTRLC |= 0x03;

  return data;
}

void writeCsReg(uint8_t addr, uint8_t value) {
  // Wolud need to wait for the bus to be idle if there
  // were other masters on the bus

  // Send start condition, write, and color sensor address
  COLORTWI.MASTER.ADDR = COLORSENSORADDR;
  // Assume no contention; wait for write interrupt flag
  while((COLORTWI.MASTER.STATUS & 0x40) == 0);
  // Send address of register
  COLORTWI.MASTER.DATA = addr;
  // Assume the sensor won't NACK; wait for flag
  while((COLORTWI.MASTER.STATUS & 0x40) == 0);
  // Send data to write to register
  COLORTWI.MASTER.DATA = value;
  // Assume the sensor won't NACK; wait for flag
  while((COLORTWI.MASTER.STATUS & 0x40) == 0);
  // Send stop condition
  COLORTWI.MASTER.CTRLC |= 0x03;
}

void initColorSensor() {
  // Set up pin directions
  COLORPORT.DIR |= COLORLEDPIN;

  // Set up TWI
```

```
63    COLORTWI.MASTER.BAUD = 155; // Corresponds to 100kHz when system clk = 32MHz
64    COLORTWI.MASTER.CTRLA |= TWI_MASTER_ENABLE_bm; // Enable the TWI master module
65    COLORTWI.MASTER.STATUS = TWI_MASTER_BUSSTATE_IDLE_gc; // Force bus status to idle
66
67    // Send configuration values
68    writeCsReg(CS_CAP_RED,   CAPRVAL);
69    writeCsReg(CS_CAP_GREEN, CAPGVAL);
70    writeCsReg(CS_CAP_BLUE,  CAPBVAL);
71    writeCsReg(CS_CAP_CLEAR, CAPCVAL);
72    writeCsReg(CS_INT_RED_LO,   (INTRVAL & 0xFF));
73    writeCsReg(CS_INT_RED_HI,   (INTRVAL >> 8));
74    writeCsReg(CS_INT_GREEN_LO, (INTGVAL & 0xFF));
75    writeCsReg(CS_INT_GREEN_HI, (INTGVAL >> 8));
76    writeCsReg(CS_INT_BLUE_LO,  (INTBVAL & 0xFF));
77    writeCsReg(CS_INT_BLUE_HI,  (INTBVAL >> 8));
78    writeCsReg(CS_INT_CLEAR_LO, (INTCVAL & 0xFF));
79    writeCsReg(CS_INT_CLEAR_HI, (INTCVAL >> 8));
80  }
81
82  // This function takes a long time to run... too long for calling every loop
83  void pollColorSensor(uint16_t *red, uint16_t *green, uint16_t *blue, uint16_t *clear) {
84    uint8_t lo, hi, r;
85
86    // Request sensor read
87    writeCsReg(CS_CTRL, 1);
88
89    // Wait for sensor data
90    r = 1;
91    while(r != 0) {
92      r = readCsReg(CS_CTRL);
93    }
94
95    lo = readCsReg(CS_DATA_RED_LO);
96    hi = readCsReg(CS_DATA_RED_HI);
97    *red = ((uint16_t)hi << 8) | lo;
98    lo = readCsReg(CS_DATA_GREEN_LO);
99    hi = readCsReg(CS_DATA_GREEN_HI);
100   *green = ((uint16_t)hi << 8) | lo;
101   lo = readCsReg(CS_DATA_BLUE_LO);
102   hi = readCsReg(CS_DATA_BLUE_HI);
103   *blue = ((uint16_t)hi << 8) | lo;
104   lo = readCsReg(CS_DATA_CLEAR_LO);
105   hi = readCsReg(CS_DATA_CLEAR_HI);
106   *clear = ((uint16_t)hi << 8) | lo;
107 }
108
109 // This updates one of red, green blue, or clear, round-robin, OR just sets up/waits for data
110 // when it is called.
111 // You'd have to call this AT LEAST (possibly more than) 10 times to get all new data....
112 char updateColorSensor(uint16_t *rr, uint16_t *gr, uint16_t *br, uint16_t *cr) {
113   static uint8_t lo, hi, r = 1, mode = 0;
114   static uint16_t red, green, blue, clear;
115   char sensorcolor;
116
117   switch(mode) {
118   case 0:                           // Request a read
119     writeCsReg(CS_CTRL, 1);
120     mode++;
121     break;
122
123   case 1:                           // Is the sensor done reading?
124     r = readCsReg(CS_CTRL);
125     if(r == 0) {
126       mode++;
127     }
128     break;
129
130   case 2:                           // Get RED LOW
```

```
    lo = readCsReg(CS_DATA_RED_LO);
    mode++;
    break;

  case 3:                           // Get RED HIGH
    hi = readCsReg(CS_DATA_RED_HI);
    red = ((uint16_t)hi << 8) | lo;
    mode++;
    break;

  case 4:                           // Get GREEN LOW
    lo = readCsReg(CS_DATA_GREEN_LO);
    mode++;
    break;

  case 5:                           // Get GREEN HIGH
    hi = readCsReg(CS_DATA_GREEN_HI);
    green = ((uint16_t)hi << 8) | lo;
    mode++;
    break;

  case 6:                           // Get BLUE LOW
    lo = readCsReg(CS_DATA_BLUE_LO);
    mode++;
    break;

  case 7:                           // Get BLUE HIGH
    hi = readCsReg(CS_DATA_BLUE_HI);
    blue = ((uint16_t)hi << 8) | lo;
    mode++;
    break;

  case 8:                           // Get CLEAR LOW
    lo = readCsReg(CS_DATA_CLEAR_LO);
    mode++;
    break;

  default:                          // Get CLEAR HIGH
    hi = readCsReg(CS_DATA_CLEAR_HI);
    clear = ((uint16_t)hi << 8) | lo;
    mode = 0;
    break;
  }


  if((red >= green) && (red >= blue)) {
    sensorcolor = 'r';
  } else if((green >= red) && (green >= blue)) {
    sensorcolor = 'g';
  } else {
    sensorcolor = 'b';
  }

  *rr = red;
  *gr = green;
  *br = blue;
  *cr = clear;

  return sensorcolor;
}
```

Listing 6: code/courierbot/ir.h

```
#ifndef IRH
#define IRH

#include <avr/io.h>

```

```
#define IRPORT PORTJ
#define IRR_PIN 0x80
#define IRG_PIN 0x40
#define IRB_PIN 0x20
#define IR_AD_ID 5

#define IRLISTN 10
#define IR_THRESHOLD 48000

void initIR();
void setIR(uint8_t pin);
#define pollIR() pollADC(IR_AD_ID)
uint16_t pollIRpin(uint8_t pin);
char updateIRs(uint16_t *rr, uint16_t *gr, uint16_t *br);

#endif
```

Listing 7: code/courierbot/ir.c

```
#include "ir.h"
#include "adc.h"

// I shouldn't have to do this here, but the
// compiler might compile this file first and
// get the frequency wrong
#define F_CPU 32000000UL
#include <util/delay.h>

void initIR() {
  IRPORT.DIR &= ~(IRR_PIN | IRG_PIN | IRB_PIN);
  IRPORT.DIR |= IRR_PIN;
  IRPORT.OUT &= ~IRR_PIN;
}

void setIRfast(uint8_t pin) {
  IRPORT.DIR &= ~(IRR_PIN | IRG_PIN | IRB_PIN);
  IRPORT.DIR |= pin;
  IRPORT.OUT &= ~pin;
}

void setIR(uint8_t pin) {
  setIRfast(pin);
  _delay_ms(20); // It takes some time for the output to stabilize
}

// This polls one of red green or blue IR values round-robin
// each time it is called - you need to call it 3 times to
// get all new data.
// This function now also interprets the data, and
// just returns a char that is 'r', 'g', 'b', or 0.
// It also keeps track of the last IRLISTN values
// and takes a vote using them to get the return value.
char updateIRs(uint16_t *rr, uint16_t *gr, uint16_t *br) {
  // Start out on RED 'cause the initIR defaults to
  // configuring for red.
  static uint8_t whichir = IRR_PIN;
  static uint16_t r,g,b;
  static char returncolor;
  char thiscolor;
  static char irlist[IRLISTN];
  uint8_t rv, gv, bv, nv, i;
  static uint8_t irlistindex = 0;

  switch(whichir) {
    case IRR_PIN:
      r = pollIR();
      whichir = IRG_PIN;
      setIRfast(whichir);
```

```
50        break;
51      case IRG_PIN:
52        g = pollIR();
53        whichir = IRB_PIN;
54        setIRfast(whichir);
55        break;
56      default:
57        b = pollIR();
58        whichir = IRR_PIN;
59        setIRfast(whichir);
60        break;
61    }
62
63    if((r > IR_THRESHOLD) || (g > IR_THRESHOLD) || (b > IR_THRESHOLD)) {
64      if((r == g) && (g == b)) {
65        // They're all the same. Could be saturated.
66        //thiscolor = 0;
67        thiscolor = returncolor;
68      } else if((r >= g) && (r >= b)) {
69        thiscolor = 'r';
70      } else if((g >= r) && (g >= b)) {
71        thiscolor = 'g';
72      } else {
73        thiscolor = 'b';
74      }
75    } else {
76      thiscolor = 0;
77    }
78
79    *rr = r;
80    *br = b;
81    *gr = g;
82
83    irlist[irlistindex] = thiscolor;
84    irlistindex++;
85    if(irlistindex >= IRLISTN) {
86      irlistindex = 0;
87    }
88
89    rv = 0; gv = 0; bv = 0; nv = 0;
90    for(i = 0; i < IRLISTN; i++) {
91      switch(irlist[i]) {
92        case 'r':
93          rv++;
94          break;
95        case 'g':
96          gv++;
97          break;
98        case 'b':
99          bv++;
100         break;
101       default:
102         nv++;
103         break;
104     }
105   }
106
107   if( (rv > gv) && (rv > bv) && (rv > nv) ) {
108     returncolor = 'r';
109   } else if( (gv > rv) && (gv > bv) && (gv > nv) ) {
110     returncolor = 'g';
111   } else if( (bv > gv) && (bv > rv) && (bv > nv) ) {
112     returncolor = 'b';
113   } else {
114     returncolor = 0;
115   }
116
117   return returncolor;
```

```
}

// Takes some time due to the delay....
// It would be good to split this up and
// put it on a timer interrupt or something...
uint16_t pollIRpin(uint8_t pin) {
  setIR(pin);
  return pollIR();
}
```

Listing 8: code/courierbot/lcdurt.h

```
#ifndef LCDURT_H
#define LCDURT_H
#include <avr/io.h>

// USART for LCD control
#define LCDURT USARTF1
#define LCDPORT PORTF
#define LCDOUTBITS 0x80

void lcdurtInit();
void lcdSendStr(char *s);
void lcdPrintf(char *fmt, ...);

#endif
```

Listing 9: code/courierbot/lcdurt.c

```
#include <stdio.h>
#include <stdarg.h>
#include "lcdurt.h"

void lcdurtInit() {
  // Set up USART
  // Set clock to 38400 baud
  LCDURT.BAUDCTRLA = 51;
  LCDURT.BAUDCTRLB = 0;
  LCDPORT.DIR |= LCDOUTBITS; // TX pin is output
  LCDURT.CTRLA = 0x00; // No interrupts
  LCDURT.CTRLB = 0x18; // Enable TX and RX
  LCDURT.CTRLC = 0x03; // 8-bit, 1 stop, no parity, async
}

// For sending COMMANDS, not just text
void lcdSendStr(char *s) {
  do {
    // Wait for transmitter to be ready
    while((LCDURT.STATUS & USART_DREIF_bm) == 0);
    // Send data
    LCDURT.DATA = *s;
  } while (*(s++) != 0);
}

void lcdPrintf(char *fmt, ...) {
  char buf[52];
  va_list ap;
  va_start(ap, fmt);
  vsnprintf(&(buf[1]), 49, fmt, ap);
  buf[0] = 'S';
  buf[50] = '\n';
  buf[51] = 0;
  lcdSendStr(buf);
}
```

Listing 10: code/courierbot/motor.h

```
#ifndef MOTOR_H
#define MOTOR_H

#define MOTORPORT PORTC
#define MOTORTC0 TCC0

#define MOTOR_PINS 0x3F
// PWM on ENA and ENB pins
#define MOTOR_ENA_PIN 1
#define MOTOR_ENB_PIN 2
// C & D pins determine direction
#define MOTOR_C1_PIN 4
#define MOTOR_D1_PIN 8
#define MOTOR_C2_PIN 16
#define MOTOR_D2_PIN 32

// Motor IDs
#define LEFT_MOTOR 1
#define RIGHT_MOTOR 2
#define BOTH_MOTORS 3

// Motor directions
#define MOTOR_FD 1
#define MOTOR_BK 2

// Motor speed scale.
// This depends on the PWM frequency
// +/- MOTOR_MAX_SPEED should fit in
// a signed 16-bit int (-32768..32767)
#define MOTOR_MAX_SPEED 1600

// This is the size of the running-average
// filter on motors. Higher = slower response.
// K = 2**MOTOR_K_EXP
// e.g. MOTOR_K_EXP = 3, so K = 8
#define MOTOR_K_EXP 4
#define MOTOR_K (1 << MOTOR_K_EXP)

void initMotors();
void releaseMotors();
void enableMotors();
void updateMotors(); // Should be called at a regular short interval
void setMotor(uint8_t motorid, int16_t dirspeed); // Gets applied on updateMotors()
                        // dirspeed is between -MOTOR_MAX_SPEED and +MOTOR_MAX_SPEED
#endif
```

Listing 11: code/courierbot/motor.c

```
#include <avr/io.h>
#include "motor.h"

// Using a PWM frequency of 20kHz (period = 50uS) for motors

// Globals
int16_t lhist[MOTOR_K], rhist[MOTOR_K], lcur, rcur;
uint8_t hist_index;

void setMotorDir(uint8_t motorid, uint8_t direction);
void setMotorSpeed(uint8_t motorid, uint16_t speed);

void initMotors() {
  uint8_t i;

  // Make motor pins outputs
  MOTORPORT.DIR |= MOTOR_PINS;
  // Set clock prescalers at CLK/1
  MOTORTC0.CTRLA = RTC_PRESCALER_DIV1_gc;
  // Set wave-generation mode to Single-Slope PWM
```

```
21    // and enable waveform outputs
22    // (Top = PER; Update on BOTTOM; OVFIF/Event on BOTTOM)
23    MOTORTC0.CTRLB = TC0_CCBEN_bm | TC0_CCAEN_bm | TC_WGMODE_SS_gc;
24    // Set period = 50uS (for clk = 32MHz; prescaler=1)
25    MOTORTC0.PER = MOTOR_MAX_SPEED;
26
27    // By default, motors off
28    MOTORTC0.CCA = 0;
29    MOTORTC0.CCB = 0;
30
31    // Motors not enabled
32    MOTORTC0.CTRLB &= (~TC0_CCAEN_bm);
33    MOTORTC0.CTRLB &= (~TC0_CCBEN_bm);
34    MOTORPORT.OUT &= (~MOTOR_ENA_PIN);
35    MOTORPORT.OUT &= (~MOTOR_ENB_PIN);
36    // Motor direction set to forward
37    MOTORPORT.OUT &= (~MOTOR_C1_PIN);
38    MOTORPORT.OUT |= MOTOR_D1_PIN;
39    MOTORPORT.OUT &= (~MOTOR_C2_PIN);
40    MOTORPORT.OUT |= MOTOR_D2_PIN;
41
42    hist_index = 0;
43    for(i = 0; i < MOTOR_K; i++) {
44      lhist[i] = 0;
45      rhist[i] = 0;
46    }
47    lcur = 0;
48    rcur = 0;
49  }
50
51  void releaseMotors() {
52    MOTORTC0.CTRLB &= (~TC0_CCAEN_bm);
53    MOTORTC0.CTRLB &= (~TC0_CCBEN_bm);
54    MOTORPORT.OUT &= (~MOTOR_ENA_PIN);
55    MOTORPORT.OUT &= (~MOTOR_ENB_PIN);
56    lcur = 0;
57    rcur = 0;
58  }
59
60  void enableMotors() {
61    MOTORTC0.CTRLB |= TC0_CCAEN_bm;
62    MOTORTC0.CTRLB |= TC0_CCBEN_bm;
63  }
64
65  void updateMotors() {
66    static int32_t lsum, rsum;
67
68    // lsum and rsum are kept as the running
69    // sum of all the values in lhist and rhist,
70    // respectively
71    lsum -= lhist[hist_index];
72    rsum -= rhist[hist_index];
73    lhist[hist_index] = lcur;
74    rhist[hist_index] = rcur;
75    lsum += lcur;
76    rsum += rcur;
77
78    // This is hist_index++ mod length of histories
79    hist_index = (hist_index + 1) & (MOTOR_K-1);
80
81    // Update left motor
82    if(lsum > 0) {
83      setMotorDir(LEFT_MOTOR, MOTOR_FD);
84      setMotorSpeed(LEFT_MOTOR, lsum >> MOTOR_K_EXP);
85    } else {
86      setMotorDir(LEFT_MOTOR, MOTOR_BK);
87      setMotorSpeed(LEFT_MOTOR, (-lsum) >> MOTOR_K_EXP);
88    }
```

```
89    // Update right motor
90    if(rsum > 0) {
91      setMotorDir(RIGHT_MOTOR, MOTOR_FD);
92      setMotorSpeed(RIGHT_MOTOR, rsum >> MOTOR_K_EXP);
93    } else {
94      setMotorDir(RIGHT_MOTOR, MOTOR_BK);
95      setMotorSpeed(RIGHT_MOTOR, (-rsum) >> MOTOR_K_EXP);
96    }
97  }
98
99  void setMotorDir(uint8_t motorid, uint8_t direction) {
100   if((motorid == LEFT_MOTOR) || (motorid == BOTH_MOTORS)) {
101     if(direction == MOTOR_FD) {
102       MOTORPORT.OUT &= (~MOTOR_C1_PIN);
103       MOTORPORT.OUT |= MOTOR_D1_PIN;
104     } else {
105       MOTORPORT.OUT |= MOTOR_C1_PIN;
106       MOTORPORT.OUT &= (~MOTOR_D1_PIN);
107     }
108   }
109   if((motorid == RIGHT_MOTOR) || (motorid == BOTH_MOTORS)) {
110     if(direction == MOTOR_FD) {
111       MOTORPORT.OUT &= (~MOTOR_C2_PIN);
112       MOTORPORT.OUT |= MOTOR_D2_PIN;
113     } else {
114       MOTORPORT.OUT |= MOTOR_C2_PIN;
115       MOTORPORT.OUT &= (~MOTOR_D2_PIN);
116     }
117   }
118 }
119
120 void setMotorSpeed(uint8_t motorid, uint16_t speed) {
121   if((motorid == LEFT_MOTOR) || (motorid == BOTH_MOTORS)) {
122     MOTORTC0.CCA = speed;
123   }
124   if((motorid == RIGHT_MOTOR) || (motorid == BOTH_MOTORS)) {
125     MOTORTC0.CCB = speed;
126   }
127 }
128
129 void setMotor(uint8_t motorid, int16_t dirSpeed) {
130   if((motorid == LEFT_MOTOR) || (motorid == BOTH_MOTORS)) {
131     lcur = dirSpeed;
132   }
133   if((motorid == RIGHT_MOTOR) || (motorid == BOTH_MOTORS)) {
134     rcur = dirSpeed;
135   }
136 }
```

Listing 12: code/courierbot/servo.h

```
1  #ifndef SERVO_H
2  #define SERVO_H
3
4  #define SERVOPORT PORTD
5  #define SERVOTC0 TCD0
6  #define SERVOTC1 TCD1
7
8  void initServos();
9  void setServoAngle(uint8_t servoid, int16_t angle);
10
11 #endif
```

Listing 13: code/courierbot/servo.c

```
1  #include <avr/io.h>
2  #include "servo.h"
```

```
3
4  // Note that all servos use PWM with a frequency
5  // of 50Hz (period of 20mS).
6
7  void initServos() {
8    // Make servo pins (bits 0-5) outputs
9    SERVOPORT.DIR |= 0x3F;
10   // Set clock prescalers at CLK/256
11   SERVOTC0.CTRLA = RTC_PRESCALER_DIV256_gc;
12   SERVOTC1.CTRLA = RTC_PRESCALER_DIV256_gc;
13   // Set wave-generation mode to Single-Slope PWM
14   // and enable waveform outputs
15   // (Top = PER; Update on BOTTOM; OVFIF/Event on BOTTOM)
16   SERVOTC0.CTRLB = TC0_CCDEN_bm | TC0_CCCEN_bm | TC0_CCBEN_bm |
17                    TC0_CCAEN_bm | TC_WGMODE_SS_gc;
18   SERVOTC1.CTRLB = TC0_CCBEN_bm | TC0_CCAEN_bm | TC_WGMODE_SS_gc;
19   // Set period = 20mS (for clk = 32MHz; prescalers=256)
20   SERVOTC0.PER = 6249; // is this right?
21   SERVOTC1.PER = 6249;
22   // Set default pulse widths at about 1500usecs
23   SERVOTC0.CCA = 187;
24   SERVOTC0.CCB = 187;
25   SERVOTC0.CCC = 187;
26   SERVOTC0.CCD = 187;
27   SERVOTC1.CCA = 187;
28   SERVOTC1.CCB = 187;
29 }
30
31 // Angle ranges from -180 to +180
32 void setServoAngle(uint8_t servoid, int16_t angle) {
33   // This is a derivation of the math for below
34   // (32000000 / 256) * (pulse width in seconds)
35   // = 125000 * pw(secs)
36   // = 125 * pw(msecs)
37   // = 125 * pw(usecs) * 1000
38   // = .125 * pw(usecs)
39   // = pw(usecs) / 8
40   // = pw(usecs) >> 3
41   // pulse width ranges from 600uS to 2400uS
42   // pw(usecs) = ((angle+180) * 5) + 600
43
44   uint16_t pwus; // Pulse-width in microseconds
45   uint16_t x;
46
47   if(angle < -180) {
48     angle = -180;
49   } else if(angle > 180) {
50     angle = 180;
51   }
52
53   pwus = ((angle + 180) * 5) + 600;
54   x = pwus >> 3;
55
56   switch(servoid) {
57     case 0:
58       SERVOTC0.CCA = x;
59       break;
60     case 1:
61       SERVOTC0.CCB = x;
62       break;
63     case 2:
64       SERVOTC0.CCC = x;
65       break;
66     case 3:
67       SERVOTC0.CCD = x;
68       break;
69     case 4:
70       SERVOTC1.CCA = x;
```

```
      break;
    case 5:
      SERVOTC1.CCB = x;
      break;
  }
}
```

Listing 14: code/courierbot/switches.h

```
#ifndef SWITCHES_H
#define SWITCHES_H

#define SWITCHPORT PORTH
#define BLBUMP_PIN 1
#define BLBUMP_PINCTRL SWITCHPORT.PIN0CTRL
#define BRBUMP_PIN 2
#define BRBUMP_PINCTRL SWITCHPORT.PIN1CTRL
#define FLBUMP_PIN 4
#define FLBUMP_PINCTRL SWITCHPORT.PIN2CTRL
#define FRBUMP_PIN 8
#define FRBUMP_PINCTRL SWITCHPORT.PIN3CTRL

void initSwitches();
uint8_t pollSwitches();

#endif
```

Listing 15: code/courierbot/switches.c

```
#include <avr/io.h>
#include "switches.h"

void initSwitches() {
  // Switches are inputs
  SWITCHPORT.DIR &= ~(BLBUMP_PIN | BRBUMP_PIN | FLBUMP_PIN | FRBUMP_PIN);
  // Invert inputs and enable pull-up resistors
  BLBUMP_PINCTRL = PORT_INVEN_bm | PORT_OPC_PULLUP_gc;
  BRBUMP_PINCTRL = PORT_INVEN_bm | PORT_OPC_PULLUP_gc;
  FLBUMP_PINCTRL = PORT_INVEN_bm | PORT_OPC_PULLUP_gc;
  FRBUMP_PINCTRL = PORT_INVEN_bm | PORT_OPC_PULLUP_gc;
}

uint8_t pollSwitches() {
  return SWITCHPORT.IN;
}
```

Listing 16: code/lcdcontrol/face.h

```
#ifndef FACE_H
#define FACE_H

// Face constants
#define EYEW 26
#define EYEH 36
#define LEYEX 17
#define LEYEY 18
#define REYEX 85
#define REYEY 18
#define LPCX 25
#define LPCY 30
#define RPCX 93
#define RPCY 30
#define LPRX 30
#define LPRY 30
#define RPRX 98
#define RPRY 30
#define LPLX 20
```

```
#define LPLY 30
#define RPLX 88
#define RPLY 30
#define PUPILW 10
#define PUPILH 10
#define MOUTHX 44
#define MOUTHY 54
#define MOUTHW 40
#define MOUTHH 10

void look(uint8_t direction);
void emote(uint8_t e);
void drawFace(uint8_t *buf);

#endif
```

Listing 17: code/lcdcontrol/face.c

```
#include <stdint.h>
#include "face.h"
#include "graphics.h"
#include "sprites.h"

// F = forward; L = left; R = right
uint8_t lookDirection;
// N = neutral; H = happy; S = sad; O = strain
uint8_t emotion;

void look(uint8_t direction) {
  lookDirection = direction;
}

void emote(uint8_t e) {
  emotion = e;
}

void drawFace(uint8_t *buf) {
  // Eyes
  if(emotion == 'T') {
    drawBmp(buf, eyeTiredBMP, LEYEX, LEYEY, EYEW, EYEH, 0, 0);
    drawBmp(buf, eyeTiredBMP, REYEX, REYEY, EYEW, EYEH, 1, 0);
  } else if(emotion == 'O') {
    drawBmp(buf, eyeTightBMP, LEYEX, LEYEY, EYEW, EYEH, 0, 0);
    drawBmp(buf, eyeTightBMP, REYEX, REYEY, EYEW, EYEH, 1, 0);
  } else {
    switch(lookDirection) {
      case 'L':
        // Left eye
        drawBmp(buf, eyeBMP, LEYEX, LEYEY, EYEW, EYEH, 0, 0);
        drawBmp(buf, pupilBMP, LPLX, LPLY, PUPILW, PUPILH, 0, 0);
        // Right eye
        drawBmp(buf, eyeBMP, REYEX, REYEY, EYEW, EYEH, 0, 0);
        drawBmp(buf, pupilBMP, RPLX, RPLY, PUPILW, PUPILH, 0, 0);
        break;
      case 'R':
        // Left eye
        drawBmp(buf, eyeBMP, LEYEX, LEYEY, EYEW, EYEH, 0, 0);
        drawBmp(buf, pupilBMP, LPRX, LPRY, PUPILW, PUPILH, 0, 0);
        // Right eye
        drawBmp(buf, eyeBMP, REYEX, REYEY, EYEW, EYEH, 0, 0);
        drawBmp(buf, pupilBMP, RPRX, RPRY, PUPILW, PUPILH, 0, 0);
        break;
      default:
        // Left eye
        drawBmp(buf, eyeBMP, LEYEX, LEYEY, EYEW, EYEH, 0, 0);
        drawBmp(buf, pupilBMP, LPCX, LPCY, PUPILW, PUPILH, 0, 0);
        // Right eye
        drawBmp(buf, eyeBMP, REYEX, REYEY, EYEW, EYEH, 0, 0);
```

```
51          drawBmp(buf, pupilBMP, RPCX, RPCY, PUPILW, PUPILH, 0, 0);
52          break;
53      }
54    }
55    // Mouth
56    switch(emotion) {
57      case 'H':
58        drawBmp(buf, mouth_smileBMP, MOUTHX, MOUTHY, MOUTHW, MOUTHH, 0, 0);
59        break;
60      case 'S':
61        drawBmp(buf, mouth_smileBMP, MOUTHX, MOUTHY, MOUTHW, MOUTHH, 0, 1);
62        break;
63      case 'T':
64        drawBmp(buf, mouth_smileBMP, MOUTHX, MOUTHY, MOUTHW, MOUTHH, 0, 1);
65        break;
66      case 'O':
67        drawBmp(buf, mouth_neutralBMP, MOUTHX, MOUTHY, MOUTHW, MOUTHH, 0, 0);
68        break;
69      default:
70        drawBmp(buf, mouth_neutralBMP, MOUTHX, MOUTHY, MOUTHW, MOUTHH, 0, 0);
71    }
72  }
```

Listing 18: code/lcdcontrol/font.h

```
1  #ifndef FONT_H
2  #define FONT_H
3
4  extern uint8_t fontBlock[];
5  extern uint8_t font[][8];
6
7  #endif
```

Listing 19: code/lcdcontrol/font.c

```
1  #include <avr/pgmspace.h>
2  #include "font.h"
3
4  uint8_t fontBlock[] PROGMEM = {
5    0b11111000,
6    0b11111000,
7    0b11111000,
8    0b11111000,
9    0b11111000,
10   0b11111000,
11   0b11111000,
12   0b00000000,
13 };
14
15 // ASCII from 0x20 to 0x7E
16 // Pretty much the same as
17 // the standard HD44780
18 // font. This was a pain.
19
20 uint8_t font[][8] PROGMEM = {
21   {
22     0b00000000,
23     0b00000000,
24     0b00000000,
25     0b00000000,
26     0b00000000,
27     0b00000000,
28     0b00000000,
29     0b00000000
30   },
31   {
32     0b00100000,
```

```
    0b00100000,
    0b00100000,
    0b00100000,
    0b00100000,
    0b00000000,
    0b00100000,
    0b00000000
  },
  {
    0b01010000,
    0b01010000,
    0b01010000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000
  },
```

Note a large portion of font.c was ommited here to save space, and because it does not provide interesting or relevant information.

Listing 20: code/lcdcontrol/font.c

```
  {
    0b01000000,
    0b00100000,
    0b00100000,
    0b00010000,
    0b00100000,
    0b00100000,
    0b01000000,
    0b00000000
  },
  {
    0b01101000,
    0b10110000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000,
    0b00000000
  }
};
```

Listing 21: code/lcdcontrol/graphics.h

```
#ifndef GRAPHICS_H
#define GRAPHICS_H

#include <stdint.h>

// I/O for LCD
#define LCD_DATA_PORT   PORTC
#define LCD_DATA_PIN    PINC
#define LCD_DATA_DDR    DDRC
#define LCD_CTL_PORT    PORTB
#define LCD_CTL_DDR     DDRB
#define LCD_E           1
#define LCD_DI          2
#define LCD_RW          4               // Don't turn this high!
#define LCD_RST         8
#define LCD_CS2         16
#define LCD_CS1         32
#define LCD_CTL_ALL     0x3F

void drawBmp(uint8_t *buf, uint8_t *bmp, uint8_t x, uint8_t y, uint8_t w, uint8_t h,
```

```
21              uint8_t mirrorlr, uint8_t mirrorud);
22 void drawText(uint8_t *buf, char *text, uint16_t x, uint16_t y);
23 void printCenteredText(uint8_t *buf, char *text);
24 void lcd_write_byte(uint8_t isdata, uint8_t chipno, uint8_t data);
25 void printBuf(uint8_t *buf);
26
27 #endif
```

Listing 22: code/lcdcontrol/graphics.c

```
1  #include <avr/pgmspace.h>
2  #include <avr/io.h>
3  #include "graphics.h"
4  #include "font.h"
5  #include "lcdcontrol.h"
6
7  // Draw a bitmap, pixelwise
8  void drawBmp(uint8_t *buf, uint8_t *bmp, uint8_t x, uint8_t y, uint8_t w, uint8_t h,
9               uint8_t mirrorlr, uint8_t mirrorud) {
10   uint8_t j,k,d,ej,ek;
11   uint16_t bmpi, bufi;
12
13   bmpi = 0;
14   for(k = 0; k < h; k++) {
15     for(j = 0; j < w; j++) {
16       if(mirrorlr)
17         ej = w-j-1;
18       else
19         ej = j;
20       if(mirrorud)
21         ek = h-k-1;
22       else
23         ek = k;
24       bmpi = k*((w+7)/8) + (j/8);
25       d = pgm_read_byte(&(bmp[bmpi])) & (0x80 >> (j%8));
26       bufi = (((y+ek)/8) * LCD_W) + (x+ej);
27       if(d)
28         buf[bufi] |= (0x80 >> (7-((y+ek)%8)));
29       else
30         buf[bufi] &= ~(0x80 >> (7-((y+ek)%8)));
31     }
32   }
33 }
34
35 void drawText(uint8_t *buf, char *text, uint16_t x, uint16_t y) {
36   uint16_t xx = x;
37   uint16_t yy = y;
38   char c;
39   while(*text) {
40     if(*text == '\n') {
41       xx = x;
42       yy += 8;
43     } else {
44       c = *text;
45       if((c > 0x1F) && (c < 0x7F)) {
46         drawBmp(buf, font[c-0x20], xx, yy, 6, 8, 0, 0);
47       } else {
48         drawBmp(buf, fontBlock, xx, yy, 6, 8, 0, 0);
49       }
50       xx += 6;
51     }
52     text++;
53   }
54 }
55
56 // prints up to one line of centered text near the top of the LCD
57 void printCenteredText(uint8_t *obuf, char *text) {
58   char buf[(LCD_W/6) + 1];
```

```
59    char *x;
60    uint16_t i, w;
61
62    // Clear any previous text
63    for(i = 0; i < LCD_W; i++) {
64      obuf[i] = 0;
65    }
66
67    // Remove newlines and restrict number of characters
68    x = text;
69    w = 0;
70    for(i = 0; (i < (LCD_W/6)) && (*x != 0); i++) {
71      if(*x != '\n') {
72        buf[i] = *x;
73        w += 6;
74      }
75      x++;
76    }
77    // Null terminate
78    buf[i] = 0;
79    // Draw text
80    drawText(obuf, buf, (LCD_W/2)-(w/2), 1);
81  }
82
83  // isdata = 1 for data, 0 for instruction
84  // chipno = 0 for left half, 1 for right half
85  void lcd_write_byte(uint8_t isdata, uint8_t chipno, uint8_t data) {
86    uint8_t ctlport, i;
87
88    ctlport = LCD_CTL_PORT;
89    // Clear all bits initially
90    ctlport &= ~LCD_CTL_ALL;
91    // Don't reset
92    ctlport |= LCD_RST;
93    // select instruction / data
94    if(isdata)
95      ctlport |= LCD_DI;
96    // select LCD half
97    if(chipno == 0) {
98      ctlport |= LCD_CS2;
99    } else {
100     ctlport |= LCD_CS1;
101   }
102   // leave R/W low
103
104   LCD_CTL_PORT = ctlport;
105
106   // Make data port output
107   LCD_DATA_DDR = 0xFF;
108   LCD_DATA_PORT = data;
109   // Toggle enable
110   LCD_CTL_PORT |= LCD_E;
111   asm volatile("nop\nnop\nnop\n");
112   LCD_CTL_PORT = ctlport;
113   asm volatile("nop\nnop\nnop\n");
114   // Make data port input, to be safe
115   LCD_DATA_DDR = 0;
116   // Delay loop
117   for(i = 0; i < 10; i++) {
118     asm volatile("nop\nnop\nnop\n");
119   }
120 }
121
122 void printBuf(uint8_t *buf) {
123   uint16_t x,y,index;
124   uint8_t half, page, lcdi;
125
126   for(half = 0; half < 2; half++) {
```

```
    for(page = 0; page < 8; page++) {
      // Set LCD's Y=0
      lcd_write_byte(0,half,0x40);
      // Set LCD's X=page
      lcd_write_byte(0,half,0xB8 | page);
      for(lcdi = 0; lcdi < 64; lcdi++) {
        x = (64 * half) + lcdi;
        y = page * 8;
        index = ((y/8) * LCD_W) + x;
        lcd_write_byte(1,half,buf[index]);
      }
    }
  }
}
```

Listing 23: code/lcdcontrol/lcdcontrol.h

```
#ifndef LCDCONTROL_H
#define LCDCONTROL_H

#define LCD_W 128
#define LCD_H 64
uint8_t lcdbuf[LCD_W * ((LCD_H+7)/8 + 1)];

#endif
```

Listing 24: code/lcdcontrol/lcdcontrol.c

```
#include <avr/io.h>
#define F_CPU 8000000UL  // 8 MHz
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <stdint.h>

#include "lcdcontrol.h"
#include "font.h"
#include "sprites.h"
#include "graphics.h"
#include "face.h"

// Command queue
volatile uint8_t cmdq[256];
volatile uint8_t cmdqStart, cmdqEnd;

// USART RX interrupt handler
ISR(USART1_RX_vect) {
  if(cmdqEnd != (cmdqStart - 1)) {
    cmdq[cmdqEnd] = UDR1;
    cmdqEnd++;
  }
}

void serial_sendByte(uint8_t data) {
  while((UCSR1A & 0x20) == 0);
  UDR1 = data;
}

void sleep() {
  sei();
  set_sleep_mode(SLEEP_MODE_IDLE);
  sleep_mode();
}

void mainLoop() {
  uint8_t cmdqs, cmdqe, cmdqlen, done, thisByte, mode, cmd, texti;
  char textbuf[(LCD_W/6) + 2];
```

```
40
41    mode = 0;
42    cmd = ' ';
43    texti = 0;
44
45    while(1) {
46      sleep();
47      done = 0;
48      while(!done) {
49        cli();
50        cmdqs = cmdqStart;
51        cmdqe = cmdqEnd;
52        sei();
53        // Works since queue length == 255
54        cmdqlen = cmdqe - cmdqs;
55        if(cmdqlen == 0) {
56          done = 1;
57          break;
58        }
59        thisByte = cmdq[cmdqs];
60        cli();
61        cmdqStart++;
62        sei();
63        serial_sendByte(thisByte); // echo
64
65        if(mode == 0) {
66          if((thisByte == 'E') || (thisByte == 'L')) {
67            cmd = thisByte;
68            mode = 1;
69          } else if(thisByte == 'S') {
70            texti = 0;
71            mode = 2;
72          }
73        } else if(mode == 1) {
74          if(cmd == 'E')
75            emote(thisByte);
76          else if(cmd == 'L')
77            look(thisByte);
78          drawFace(lcdbuf);
79          printBuf(lcdbuf);
80          mode = 0;
81        } else if(mode == 2) {
82          if((texti == (LCD_W/6)) ||
83             (thisByte == 0) ||
84             (thisByte == '\n') ||
85             (thisByte == '\r')) {
86            textbuf[texti] = 0;
87            printCenteredText(lcdbuf, textbuf);
88            printBuf(lcdbuf);
89            mode = 0;
90          } else {
91            textbuf[texti++] = thisByte;
92            mode = 2;
93          }
94        }
95      }
96    }
97  }
98
99  int main() {
100   uint16_t i;
101
102   cli();
103
104   // Set up LCD
105   // Set data port as INPUT (for now)
106   LCD_DATA_DDR = 0;
107   // Output control bits
```

```
  LCD_CTL_DDR |= LCD_CTL_ALL;
  // Reset
  LCD_CTL_PORT &= ~LCD_CTL_ALL;
  _delay_ms(1);
  LCD_CTL_PORT |= LCD_RST;

  lcd_write_byte(0,0,0x3F); // Turn on left half
  lcd_write_byte(0,1,0x3F); // Turn on right half
  lcd_write_byte(0,0,0xC0); // Set start=0
  lcd_write_byte(0,1,0xC0); // Set start=0

  // Set up USART
  //DDRD &= 0xFE; // RXD0 is input
  //DDRD |= 2;    // TXD0 is output
  DDRD &= 0xFB; // RXD1 is input
  DDRD |= 0x08; // TXD1 is output
  // Baudrate = 38,400 for clock speed = 8MHZ
  UBRR1H = 0;
  UBRR1L = 12;
  UCSR1B = 0x98; // Enable TX, RX; enable RX interrupt
  UCSR1C = 0x06; // 8-bit, NP, async
  cmdqStart = 0;
  cmdqEnd = 0;

  // Clear LCD bufer
  for(i = 0; i < sizeof(lcdbuf); i++) {
    lcdbuf[i] = 0;
  }

  // Save power
  ACSR = 0x80; // Turn off analog comparator

  sei();

  printCenteredText(lcdbuf, "Hello, World!");
  look('F');
  emote('H');
  drawFace(lcdbuf);
  printBuf(lcdbuf);

  // Never returns
  mainLoop();

}
```

Listing 25: code/lcdcontrol/sprites.h

```

#ifndef SPRITES_H
#define SPRITES_H
#include <avr/pgmspace.h>

extern uint8_t eyeBMP[];
extern uint8_t pupilBMP[];
extern uint8_t mouth_smileBMP[];
extern uint8_t mouth_neutralBMP[];
extern uint8_t eyeTiredBMP[];
extern uint8_t eyeTightBMP[];

#endif
```

Listing 26: code/lcdcontrol/sprites.c

```

#include <avr/pgmspace.h>
#include "sprites.h"

```

```
5
6  uint8_t eyeBMP[] PROGMEM = {
7  0b00000000, 0b11111111, 0b11000000, 0b00000000,
8  0b00000011, 0b11111111, 0b11110000, 0b00000000,
9  0b00000111, 0b11111111, 0b11111000, 0b00000000,
10 0b00001111, 0b00000000, 0b00111100, 0b00000000,
11 0b00011100, 0b00000000, 0b00001110, 0b00000000,
12 0b00111000, 0b00000000, 0b00000111, 0b00000000,
13 0b01110000, 0b00000000, 0b00000011, 0b10000000,
14 0b01100000, 0b00000000, 0b00000001, 0b10000000,
15 0b11100000, 0b00000000, 0b00000001, 0b11000000,
16 0b11000000, 0b00000000, 0b00000000, 0b11000000,
17 0b11000000, 0b00000000, 0b00000000, 0b11000000,
18 0b11000000, 0b00000000, 0b00000000, 0b11000000,
19 0b11000000, 0b00000000, 0b00000000, 0b11000000,
20 0b11000000, 0b00000000, 0b00000000, 0b11000000,
21 0b11000000, 0b00000000, 0b00000000, 0b11000000,
22 0b11000000, 0b00000000, 0b00000000, 0b11000000,
23 0b11000000, 0b00000000, 0b00000000, 0b11000000,
24 0b11000000, 0b00000000, 0b00000000, 0b11000000,
25 0b11000000, 0b00000000, 0b00000000, 0b11000000,
26 0b11000000, 0b00000000, 0b00000000, 0b11000000,
27 0b11000000, 0b00000000, 0b00000000, 0b11000000,
28 0b11000000, 0b00000000, 0b00000000, 0b11000000,
29 0b11000000, 0b00000000, 0b00000000, 0b11000000,
30 0b11000000, 0b00000000, 0b00000000, 0b11000000,
31 0b11000000, 0b00000000, 0b00000000, 0b11000000,
32 0b11000000, 0b00000000, 0b00000000, 0b11000000,
33 0b11000000, 0b00000000, 0b00000000, 0b11000000,
34 0b11100000, 0b00000000, 0b00000001, 0b11000000,
35 0b01100000, 0b00000000, 0b00000001, 0b10000000,
36 0b01110000, 0b00000000, 0b00000011, 0b10000000,
37 0b00111000, 0b00000000, 0b00000111, 0b00000000,
38 0b00011100, 0b00000000, 0b00001110, 0b00000000,
39 0b00001111, 0b00000000, 0b00111100, 0b00000000,
40 0b00000111, 0b11111111, 0b11111000, 0b00000000,
41 0b00000011, 0b11111111, 0b11110000, 0b00000000,
42 0b00000000, 0b11111111, 0b11000000, 0b00000000
43 };
44
45 uint8_t pupilBMP[] PROGMEM = {
46 0b00011110, 0b00000000,
47 0b00111111, 0b00000000,
48 0b01101111, 0b10000000,
49 0b11001111, 0b11000000,
50 0b11011111, 0b11000000,
51 0b11111111, 0b11000000,
52 0b11111111, 0b11000000,
53 0b01111111, 0b10000000,
54 0b00111111, 0b00000000,
55 0b00011110, 0b00000000
56 };
57
58 uint8_t mouth_smileBMP[] PROGMEM = {
59 0b01111110, 0b00000000, 0b00000000, 0b00000000, 0b01111110,
60 0b11111111, 0b00000000, 0b00000000, 0b00000000, 0b11111111,
61 0b11000011, 0b11111111, 0b11111111, 0b11111111, 0b11000011,
62 0b11000001, 0b11111111, 0b11111111, 0b11111111, 0b10000011,
63 0b11111000, 0b00000000, 0b00000000, 0b00000000, 0b00011111,
64 0b01111100, 0b00000000, 0b00000000, 0b00000000, 0b00111110,
65 0b00001111, 0b11111111, 0b11111111, 0b11111111, 0b11110000,
66 0b00000111, 0b11111111, 0b11111111, 0b11111111, 0b11100000,
67 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
68 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000
69 };
70
71
72 uint8_t mouth_neutralBMP[] PROGMEM = {
```

```
0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00111111, 0b11111111, 0b11111111, 0b11111111, 0b11111100,
0b11111111, 0b11111111, 0b11111111, 0b11111111, 0b11111110,
0b11100000, 0b00000000, 0b00000000, 0b00000000, 0b00000111,
0b11100000, 0b00000000, 0b00000000, 0b00000000, 0b00000111,
0b01111111, 0b11111111, 0b11111111, 0b11111111, 0b11111110,
0b00111111, 0b11111111, 0b11111111, 0b11111111, 0b11111100,
0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000
};

uint8_t eyeTiredBMP[] PROGMEM = {
0b00000000, 0b11111111, 0b11000000, 0b00000000,
0b00000011, 0b11111111, 0b11110000, 0b00000000,
0b00000111, 0b11111111, 0b11111000, 0b00000000,
0b00001111, 0b00000000, 0b00111100, 0b00000000,
0b00011100, 0b00000000, 0b00001110, 0b00000000,
0b00111000, 0b00000000, 0b00000111, 0b00000000,
0b01110000, 0b00000000, 0b00000011, 0b10000000,
0b01100000, 0b00000000, 0b00000001, 0b10000000,
0b11100000, 0b00000000, 0b00000001, 0b11000000,
0b11000000, 0b00000000, 0b00000000, 0b11000000,
0b11000000, 0b00000000, 0b00000000, 0b11000000,
0b11000000, 0b00000000, 0b00000000, 0b11000000,
0b11000000, 0b00000000, 0b00000000, 0b11000000,
0b11000000, 0b00000000, 0b00000000, 0b11000000,
0b11000000, 0b11111111, 0b11000000, 0b11000000,
0b11001111, 0b11111111, 0b11111100, 0b11000000,
0b11111111, 0b11111111, 0b11111111, 0b11000000,
0b11111000, 0b11111111, 0b11000111, 0b11000000,
0b11000000, 0b11111111, 0b11000000, 0b11000000,
0b11000000, 0b01111111, 0b10000000, 0b11000000,
0b11000000, 0b00111111, 0b00000000, 0b11000000,
0b11000000, 0b00011110, 0b00000000, 0b11000000,
0b11000000, 0b00000000, 0b00000000, 0b11000000,
0b11000000, 0b00000000, 0b00000000, 0b11000000,
0b11000000, 0b00000000, 0b00000000, 0b11000000,
0b11000000, 0b00000000, 0b00000000, 0b11000000,
0b11100000, 0b00000000, 0b00000001, 0b11000000,
0b01100000, 0b00000000, 0b00000001, 0b10000000,
0b01110000, 0b00000000, 0b00000011, 0b10000000,
0b00111000, 0b00000000, 0b00000111, 0b00000000,
0b00011100, 0b00000000, 0b00001110, 0b00000000,
0b00001111, 0b00000000, 0b00111100, 0b00000000,
0b00000111, 0b11111111, 0b11111000, 0b00000000,
0b00000011, 0b11111111, 0b11110000, 0b00000000,
0b00000000, 0b11111111, 0b11000000, 0b00000000
};

uint8_t eyeTightBMP[] PROGMEM = {
0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00001000, 0b00000000, 0b00000000, 0b00000000,
0b00001100, 0b00000000, 0b00000000, 0b00000000,
0b00001100, 0b00000000, 0b00000000, 0b00000000,
0b00001110, 0b00000000, 0b00000000, 0b00000000,
0b00001111, 0b00000000, 0b00000000, 0b00000000,
0b00000111, 0b10000000, 0b00000000, 0b00000000,
0b00000111, 0b10000000, 0b00000000, 0b00000000,
0b00000011, 0b11000000, 0b00000000, 0b00000000,
0b00000011, 0b11100000, 0b00000000, 0b00000000,
0b00000001, 0b11110000, 0b00000000, 0b00000000,
0b00000000, 0b11111100, 0b00000000, 0b00000000,
0b00000000, 0b11111111, 0b00000000, 0b00000000,
0b00000011, 0b11111111, 0b11110000, 0b00000000,
0b00001111, 0b11111111, 0b11111100, 0b00000000,
0b00011111, 0b11111111, 0b11111110, 0b00000000,
```

```
0b00111111, 0b11111111, 0b11111111, 0b00000000,
0b00111111, 0b11111111, 0b11111111, 0b00000000,
0b00111111, 0b11111111, 0b11111111, 0b00000000,
0b00011111, 0b11111111, 0b11111110, 0b00000000,
0b00001111, 0b11111111, 0b11111100, 0b00000000,
0b00000011, 0b11111111, 0b11110000, 0b00000000,
0b00000000, 0b11111111, 0b00000000, 0b00000000,
0b00000000, 0b11111100, 0b00000000, 0b00000000,
0b00000001, 0b11110000, 0b00000000, 0b00000000,
0b00000011, 0b11100000, 0b00000000, 0b00000000,
0b00000011, 0b11000000, 0b00000000, 0b00000000,
0b00000111, 0b10000000, 0b00000000, 0b00000000,
0b00000111, 0b10000000, 0b00000000, 0b00000000,
0b00001111, 0b00000000, 0b00000000, 0b00000000,
0b00001110, 0b00000000, 0b00000000, 0b00000000,
0b00001100, 0b00000000, 0b00000000, 0b00000000,
0b00001100, 0b00000000, 0b00000000, 0b00000000,
0b00001000, 0b00000000, 0b00000000, 0b00000000,
0b00000000, 0b00000000, 0b00000000, 0b00000000,
0b00000000, 0b00000000, 0b00000000, 0b00000000
};
```

B Circuit Diagrams

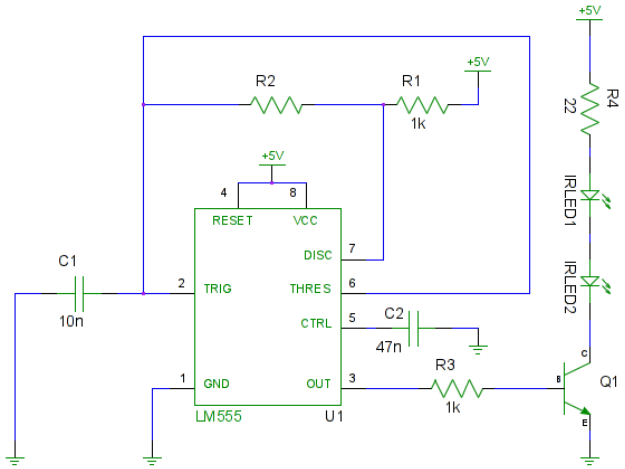

Figure 8: Beacon circuit

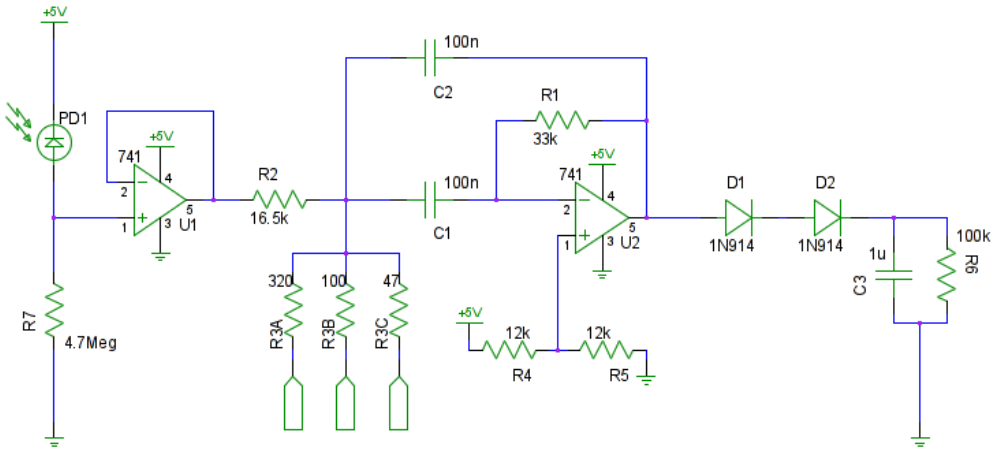

Figure 9: Beacon-detector circuit

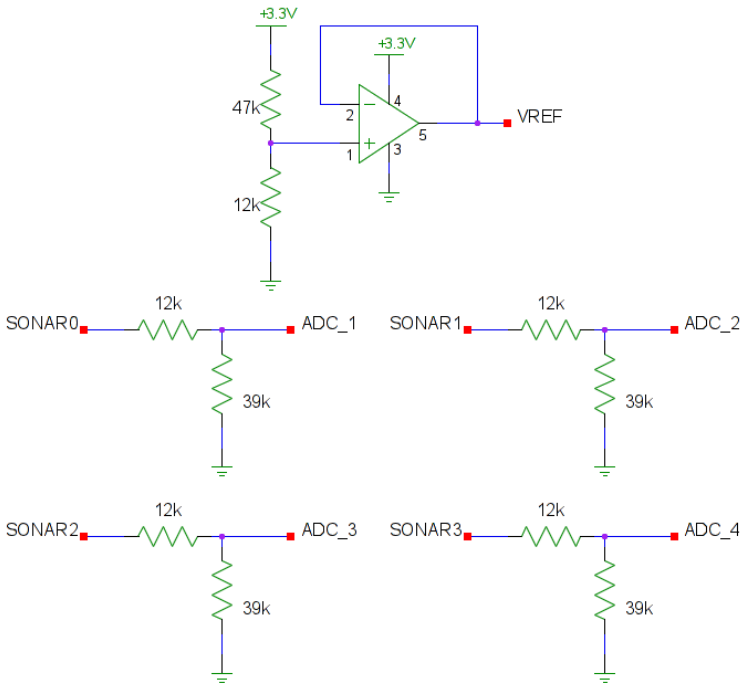

Figure 10: Voltage conditioning circuit for sonars and ADC