Lab 02

Avoid-Obstacle and Random Wander

Reading: *Ch. 3 of text*
Read this entire lab procedure before starting the lab.

Purpose: An essential characteristic of an autonomous robot is the ability to navigate in an environment safely. The purpose of this lab is to develop random wander and obstacle avoidance behaviors for the MegaBot. The design of your program should use *subsumption architecture* where layer 0 of your control architecture will be the *collide* and *run away* behaviors to keep the robot from hitting obstacles. Layer 1 will be the *random wander* behavior which moves the robot a random distance and/or heading every n seconds.

Objectives: At the conclusion of this lab, the student should be able to:

- Mount infrared and/or sonar sensors on the robot
- Acquire and use data from all of the robot's range sensors
- Write random wander and obstacle avoidance behaviors on the MegaBot using sensor feedback and a bang-bang or proportional controller
- Use modular programming to implement subsumption architecture or a state machine on the MegaBot
- Move the robot safely in an environment with obstacles

****

Equipment: base robot
4 to 6 Infrared and/or ultrasonic sensors (analog and/or digital) sensors
3 LEDs
Tape measurer

Theory:

Last week, you created the first two primitive motion behaviors on the robot: *Go-To-Goal* and *Go-To-Angle*. It is important to safely navigate a cluttered environment so this week you will implement the *Avoid-Obstacle* and *Random Wander* behaviors. Each week, you will create more functionality and behaviors for your robot so you should design the system to be as modular as possible in order to re-use algorithms and code and build on what you have done before. One way to do this is to use the subsumption architecture with layers where the most basic layer is motion control and obstacle avoidance.

The Sharp IR sensor will return an analog value that is proportional to the distance to an object based upon time of flight for infrared light. The analog ultrasonic sensor emits a sound (ping) and measures the time of flight for the sound to return to calculate distance. The digital ultrasonic sensor uses a trigger pin and echo to perform the same calculation. As with all IR and sonar; the ambient light, color, texture, material, and angle of incidence determine how much energy is returned and this as well as specular reflection may affect accurate measurements.

Obstacle avoidance will be based upon sensor feedback. The IR or sonar sensor is used to detect the distance to an obstacle and the error between the desired and actual distance will be used to control the robot wheels. The robot collide behavior will stop the robot if the distance to the obstacle is within a given error band, otherwise the robot continues to move forward. This type of control is BANG-BANG or ON-OFF control because either the controller affects the robot motors or it does not. The robot run away behavior will use the sensor data to create a polar plot and the robot will move away proportional to the distance to the object as well as the direction of the obstacle. This type of control is called proportional

control because the change in robot motion is affected by the magnitude of the error. Figure 1 shows an example of the feedback control system for the robot obstacle avoidance behavior.

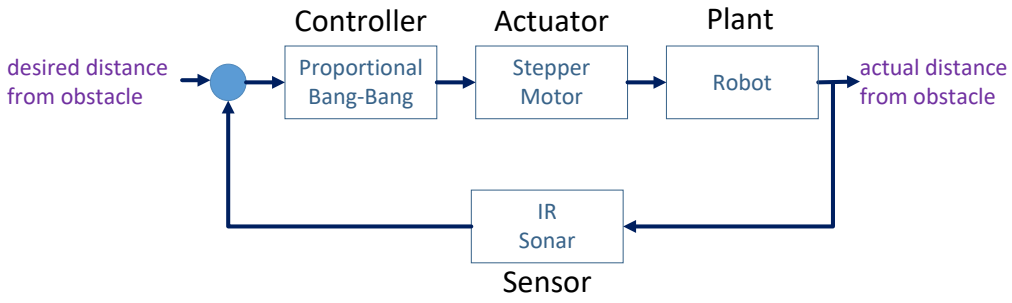


Figure 1: Obstacle Avoidance Feedback Control System

References:

Use the following references to help you implement the various requirements in the lab procedure.

Sharp IR Sensor Library: http://playground.arduino.cc/Main/SharpIR

Parallax Ping Sonar: http://playground.arduino.cc/Code/NewPing
https://www.arduino.cc/en/Tutorial/Ping
http://playground.arduino.cc/Code/PingInterruptCode

HC-SR04 Sonar: http://www.instructables.com/id/Simple-Arduino-and-HC-SR04-Example/
http://www.instructables.com/id/Hack-an-HC-SR04-to-a-3-pin-sensor/

PID Tuning: https://youtu.be/nvAQHSe-Ax4

Timers & Interrupts: http://arduinoinfo.mywikis.net/wiki/Timers-Arduino#Arduino_Timers_and_Interrupts

Interrupts: http://playground.arduino.cc/Code/Interrupts
http://playground.arduino.cc/Code/Timer1

Feedback Systems: An Introduction for Scientists and Engineers,
Karl J. Astrom and Richard M. Murray
http://www.cds.caltech.edu/~murray/amwiki/index.php/Main_Page
Tuning of a PID controller using Ziegler-Nichols Method
https://www.scribd.com/document/233690247/Tuning-of-a-PID-controller-using-Ziegler-Nichols-Method

Pre-Lab:

- Create the state diagram and state transition table for the obstacle avoidance behavior with 3 states (random wander, avoid obstacle, go to goal).
- Simulate the collide behavior in Tinkercad or Dawson Virtual Robot Lab

****

LAB PROCEDURE

****

Part I – Mount IR and Sonar Sensors to Robot

1. Mount at least 4 IR sensors to the front, back, left, and right side of the robot. Note they may already be mounted. The IR sensor is an analog sensor so you should connect the signal wire (white) to one of the analog pins on the microcontroller (A0-A15). Connect the black wire to the ground buss (blue line) on the breadboard. Connect the red wire to the 5V buss (red line closest to power switch) on the breadboard. Figure 2 shows the pinout for the Sharp IR sensor. ***BE CAREFUL! If you connect the sensor to the wrong power buss you could damage it!***

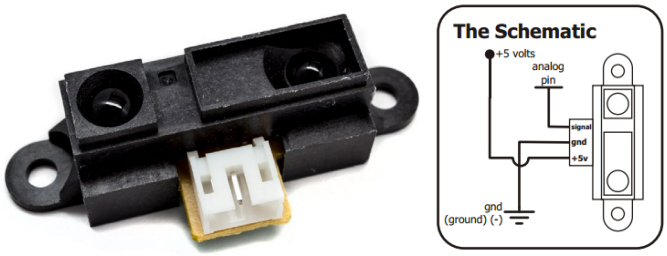


Figure 2: Sharp IR Sensor Pin Out

2. Attach at least one digital sonar sensor to your robot and connect the white signal line to one of the analog pins on the microcontroller (A0-A15). *Note that you can use the analog pins whether you have a digital or analog sonar sensor.* **Connect the black wire to the ground buss (blue line) on the breadboard. Connect the red wire to the 5V buss (red line closest to power switch) on the breadboard. BE CAREFUL! If you connect the sensor to the wrong power buss you could damage it!** Figure 3 shows the pinout for the various sonar sensors. For now, use masking tape to attach the sensor, after you are sure where you want to mount it, go see the ECE technicians to permanently mount it if you don't have a screwdriver, screws, or the required holes in the chassis. If you have a sensor with 4 pins, simply tie the trigger and echo pins together. You may want to put the sonar to point over the front corners of the robot to cover the blind spots.

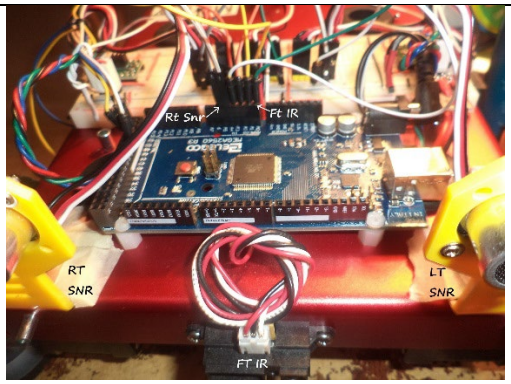
a. Maxbotix Analog

b. Parallax Ping Digital

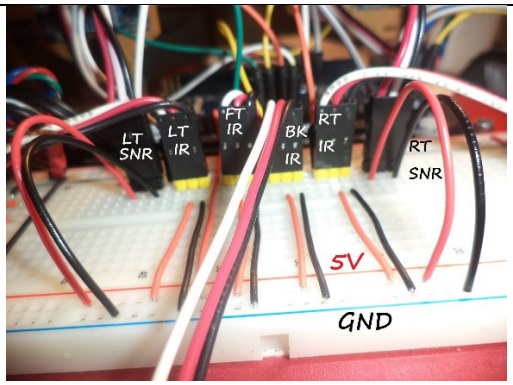
c. HC-SR04 Digital

Figure 3: Sonar Sensor Types

Figure 4 shows an example of how to mount and wire the IR and sonar sensors to the robot. You have the flexibility of changing the mounting location as well as the analog pins where you attach the sensors.



a. Sensors Front



b. Sensors Back

Figure 4: Mounting IR and Sonar Sensors

Part II – Sensor and State Machine Sample Code

1. Review each of the following libraries (Libraries Folder) Install the necessary libraries by doing the following:
   a. Sketch->Include Libraries->Manage Libraries->Search for *PinChangeInt*->click Install
   b. Sketch->Include Libraries->Manage Libraries->Search for *TimerOne*->click Install
   c. Sketch->Include Libraries->Manage Libraries->Search for *NewPing*->click Install
   d. There are also other libraries available in the lab folder and Arduino IDE, try them out to see which ones work best for your sensors and code.
2. Review each of the sample programs Lab 02 code folder. Read all comments to completely understand how to use the sensors with an interrupt to create a state machine on the robot.
   a. *RobotTimerInterrupt.ino* - uses a timer interrupt to update IR and sonar data to use for obstacle avoidance on the robot. Uses NewPing library to read sonar.
   b. *RobotInterrupt.ino* - uses a pin change interrupt with delays to update sonar data based upon falling edge from ping echo and then calls obstacle avoidance routine.
   c. *DigitalSonar.ino* - uses NewPing library to read sensor data without an interrupt.
   d. *StateMachine.ino* - template for creating a state-machine or behavior-based control architecture.
   e. *StateMachineExample2.ino* - template that uses switch case to create a state machine, used for a different system so you will have to modify to work with your robot.

Part III –Infrared Range Sensor Calibration

1. Write a program to read the analog value from the sensor by using
```
int value = 0;
value = analogRead(SensorPin);
```
2. Since there will be some error in the IR sensor, it is a good rule of thumb to take an average of sensor readings as opposed to using one data point. Modify your program to take an average of every 5 sensor readings by using.
```
int value = 0;
for (int i = 0; i < 4 ; i++) {
   value = value + analogRead(SensorPin);
}
value=value/5;
```
3. Next modify the program to display the IR sensor data on the serial monitor. In order to do this add `Serial.begin(9600);` in the setup() function. Next, add `Serial.println(value);` after you find the average of the 5 IR sensor readings. In order to open the serial monitor, click the hour glass on the top right of the IDE. ***Note that you can also use a library function to do this but you need to determine how it works. Try to use the SharpIR sensor library.***
4. Measure the analog value returned from the sensor for distances from 1” to 20” (see Table 1). Recall that sensors have a range and dead zone so it may not be accurate at certain distances at all.
5. Plot this data in Excel and use curve fitting to find the equation to linearize the data. See an example of how to do this at the following link: https://acroname.com/articles/linearizing-sharp-ranger-data

Equation: _____

6. Since all 4 of the IR sensors on your robot may not be the same, repeat steps 4 and 5 for all 4 sensors. ***You must include a table of the data in your lab memo. You must include a plot of the analog and distance data in your lab memo. You must include the equation found in your lab memo.***

7. Modify the program to output inches using the equation you found and create a table of measured versus actual distance for 1" to 20" with percent error (see Table 1). ***You must include this table in your lab memo. Your table should include data from all 4 sensors.*** Please use the following formula for percent error.

$$\%error = \frac{meas - actual}{actual}\ x\ 100$$

Table 1: Sharp IR Sensor Data

	Front IR			Rear IR			Left IR			Right IR		
Actual Inches	Analog Value	Measured Inches	% error	Analog Value	Measured Inches	% error	Analog Value	Measured Inches	% error	Analog Value	Measured Inches	% error
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

8. ***Address in your memo the accuracy and range for each of your 4 IR sensors***

Part IV – Sonar Range Sensor Calibration

1. Write a program to read the digital value from the digital sensor by using the following code in a loop.

```
long value;
pinMode(ftSonarPin, OUTPUT);//set the PING pin as an output
```

```
digitalWrite(ftSonarPin, LOW);//set the PING pin low first
delayMicroseconds(2);//wait 2 us
digitalWrite(ftSonarPin, HIGH);//trigger sonar by a 2 us HIGH PULSE
delayMicroseconds(5);//wait 5 us
digitalWrite(ftSonarPin, LOW);//set pin low first again
pinMode(ftSonarPin, INPUT);//set pin as input with duration as reception
time
value = pulseIn(ftSonarPin, HIGH);//measures how long the pin is high
```

2. Once again you may want to use the average of 5 sensor readings to reduce error. ***Note that you can also use a library function to do this, but you need to determine how it works.***
3. Record the data from 1" to 20" in a table similar to Table 1 that must be included in your lab memo.
4. Plot this data in Excel and use curve fitting to find the equation of the line. You must include a plot of the analog/digital and distance data in your lab memo. You must include the equation found in your lab memo. Note that unlike the IR sensor, the sonar sensor data equation should be close to linear.

   Equation: _____

5. Modify the program to output inches using the equation you found and create a table of measured versus actual distance for 1" to 20" with percent error (see Table 1). ***Include the table in the appendix of your lab 2 memo.***
6. Repeat steps 4 – 6 for the number of sonar sensors you have mounted to your robot. ***Include a discussion about the sonar range and accuracy in your lab memo.***
7. ***NOTE IF YOU DON'T GET GOOD CONSISTENT DATA FROM THE SONAR, PLEASE TRY TO USE THE NewPing.h library available in the Arduino IDE to collect data.***

Part V – Avoid-Obstacle Behavior (Layer 0)

Behavior-based programming uses primitive behaviors as modules for control. Primitive behaviors are concerned with achieving or maintaining a single, time-extended goal. They take inputs from sensors (or other behaviors) and send outputs to actuators (or other behaviors). Figure 5 shows the robot primitives for a behavior-based control architecture. Figure 5 is the Avoid-Obstacle behavior and Layer 0 of the subsumption architecture.

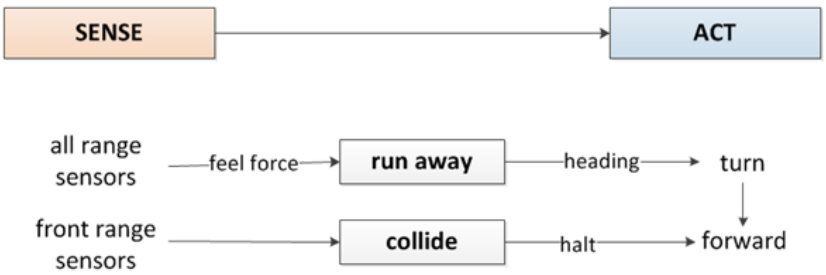


Figure 5: Level 0 – Obstacle Avoidance

In this lab, you will create two behaviors: *avoid-obstacle*, and *random wander*. The obstacle avoidance behavior will use either *collide* or *run away* primitive behaviors. In the *collide* behavior, the robot will drive forward and stop when an obstacle is detected and continue moving forward when the object is removed. To demonstrate the *collide* behavior, the robot should drive forward until it encounters an obstacle and stop without hitting it. Think of the collide behavior as the aggressive kid who comes

close but then stops short from touching the object. Turn on the RED LED when the robot executes the aggressive kid behavior.

In the *run away* behavior, the robot will move forward or sit still and when an obstacle is detected, move away proportional to where the obstacle is felt (feel force). The robot's motion will be based upon the potential fields concept. For the *run away* behavior, create a plot of the sensor readings and use the sum to create a repulsive vector to turn the robot away (i.e. *feel force*). The robot should turn by some angle proportional to the repulsive vector and continue moving. To demonstrate the *run away* behavior, the robot should sit in the middle of the floor until an object gets close and then move the opposite direction to get away based upon the potential field. Think of this behavior as the shy kid who does not want any object to get too close to him. It is possible to also show run away for an aggressive kid who moves forward and then moves away when an object gets close. It should be clear during the demonstration what type of behavior the robot is executing. Turn on the YELLOW LED when the robot executes the shy kid behavior.

Part VI – Random Wander (Layer 1)

In a random wander behavior, the robot will move in a random pattern when no obstacles are present. Create a random wander routine that the robot uses to explore the room. This can be done by generating a random number that represents the robot's heading, steps, distance, or motor speed every n seconds. You have the flexibility of using any combination of these values to make the robot explore the environment. To demonstrate this behavior, set the robot on the floor and execute the random motion. Turn on an LED to indicate when the robot is randomly wandering. Turn on the GREEN LED when the robot executes the random wander behavior.

Part VII – Subsumption Architecture – Smart Wander Behavior (Layers 0 and 1)

In this section, you will use the subsumption architecture to create a *smart wander* behavior (see Figure 6). The perceptual schema is *feel force* and the motor schemas are *run away* and *collide*. The primitive behaviors are also called *run away* and *collide* and the two together make the abstract behavior, *obstacle avoidance*. The second layer of the architecture is the *wander* module created in part 3. The wander module passes the heading to the *avoid* behavior which combines the feel force and wander heading to determine the direction the robot should turn to move away from obstacle. Note that the power of subsumption architecture is that output of the higher level subsumes the output from the lower level. The avoid module supp also can suppress the output from runaway and replace it to make the robot turn. There is sample code provided with the lab 2 files to help develop the structure for a behavior-based architecture state machine.

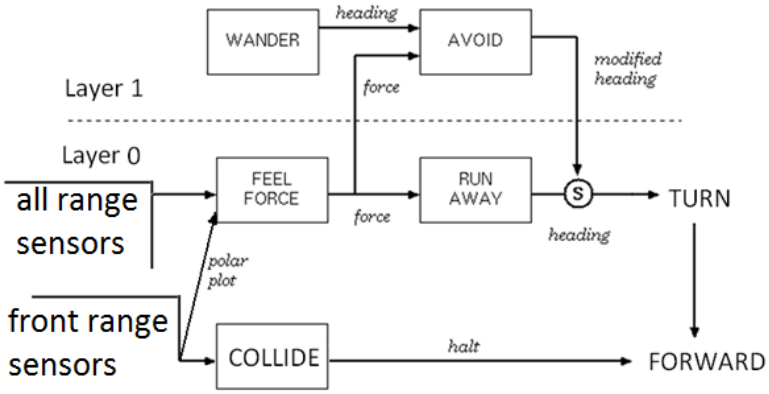


Figure 6: Smart Wander Behavior

Now improve the random wander routine by integrating obstacle avoidance (collide and run away). The robot should wander randomly until an obstacle is encountered. The robot should *run away* from the obstacle and continue to wander. The robot's heading from the *wander* behavior should be modified based upon the force from the range sensors and then turn and move from the obstacle. The *avoid* module in Layer 1 combines the *FeelForce* vector with the *Wander* vector. The *Avoid* module then subsumes the heading from the Run Away module and replaces it with the modified heading as input to the *Turn* module. The power in this type of architecture is the flexibility in the execution based upon the use of inhibition and suppression. Figure 7 provides an example of the robot's sample motion for the Avoid-Obstacle behavior.

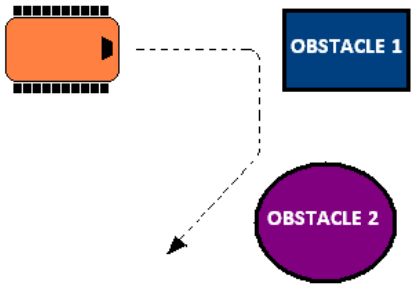


Figure 7: Subsumption Architecture – Sample Robot Motion

Your program should provide a method to get the robot 'unstuck' if it approaches any local minima points (i.e. oscillates between two obstacles). Your program should be as modular as possible with multiple subroutines and behaviors that will be integrated in subsequent programs. Devise a method to test and confirm that your program works correctly and present the results in the laboratory memo.

Part VIII – Integrate Avoid-Obstacle and Go-To-Goal Behaviors (Layer 2)

An alternate environment exploration technique would be to send the robot to a specific goal point or via points. The final step in the procedure will be to integrate the Avoid-Obstacle and Go-To-Goal Behaviors. The robot should attempt to drive to a goal position and stop within a given error. If an obstacle is encountered, the robot should navigate around the obstacle until there is a direct line of sight to the goal position and then continue moving to the goal position. Note that this behavior would have to subsume the *random-wander* behavior.

It is necessary to use potential fields to solve this problem by finding the sum of the goal attraction vector and the obstacle repulsion vector. It is important to keep track of the robot's state and position as it avoids the obstacle in order to calculate the new vector to the goal and also to identify when to abandon the obstacle and make a straight path to goal. This is also referred to as potential field navigation. Figures 8 and 9 is an example of using potential fields to move a robot toward a goal while avoiding obstacles. Figure 10 shows an example of the state machine that can be used to switch between avoid-obstacle and go-to-goal. Turn on RED, YELLOW, and GREEN LEDs when the robot is moving toward the goal.

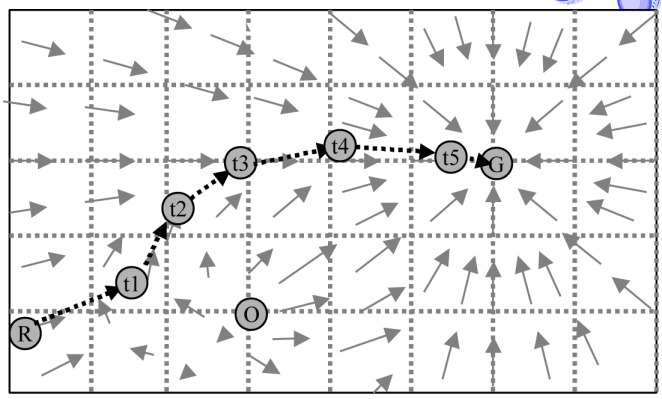


Figure 8: Potential Fields Method for Obstacle Avoidance

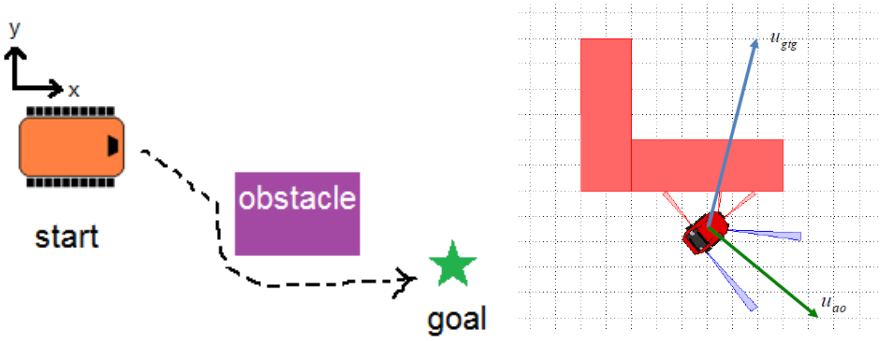


Figure 9: Avoid-Obstacle & Go-To-Goal Behaviors

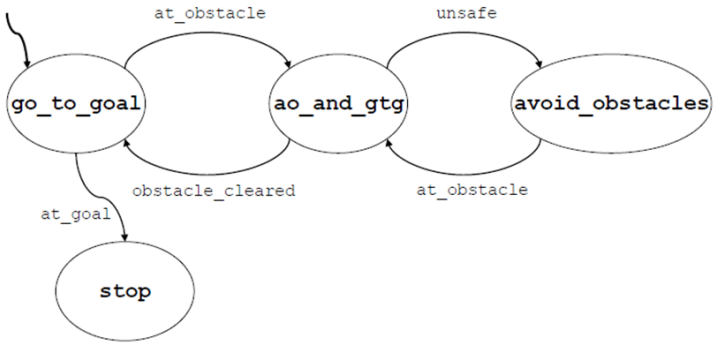


Figure 10: Avoid-Obstacle & Go-To-Goal State Machine

Submission Requirements:

Software Design Plan

For each lab you will submit a software design plan which may be in the form of pseudocode, a flowchart, state machine or subsumption architecture. The plan will be graded and you will get feedback on implementation before designing the full system.

Demonstration:

Bring your robot fully charged to class every day! Plug it in overnight.

During the demonstration, you will show each layer of the architecture separately.

- For layer 0, the robot should demonstrate shy kid and aggressive kid, separately. Please review the lab procedure if you don't recall what this means.

- For layer 1, to demonstrate random wander, the robot should turn at a random heading and move forward periodically.
- To demonstrate the *Avoid* behavior, the robot should wander in the environment and halt when it "collides" with an obstacle, or modify the heading when it encounters and obstacle and then run away. The robot should give some type of audible and/or visual signal when an obstacle is encountered. Use RED, GREEN or YELLOW LEDs as described in the lab procedure.
- The final component of the demonstration involves giving the robot a goal position and the robot should move to this location while also avoiding obstacles. The robot should turn on the RED, YELLOW, and GREEN LED when it is moving toward the goal.

Program:

In subsequent weeks you will reuse this code thus your code should follow proper programming techniques such as detailed commenting and be as modular as possible where behaviors and reactive rules are separate functions.

Use the following guidelines for creating your code.

- Code has a header comment with
- Program Name
    - Author Names
    - Date Created
    - Overall program description
    - Summary of key functions created with a description of each
- Code is modular with multiple functions called from the loop function. Functions have logical names that describe their functionality.
- Code is organized in a logical fashion with global and local variables with intuitive names.
- Code has block comments to introduce and describe each function and sufficient in line comments to describe how key parts of the code work.
- All functions and variables should have logical names that describe what they do or the value they hold. There should be no magic numbers or numeric values not related to a constant or variable to make it clear what it does.
- In general, someone unfamiliar with your project should be able to read your code and understand the functionality based upon your comments and naming convention.

Memo Guidelines:

Please use the following checklist to insure that your memo meets the basic guidelines.

✓ Format
- Begins with Date, To , From, Subject
- Font no larger than 12 point font
- Spacing no larger than double space
- Includes handwritten initials of both partners at the top of the memo next to the names
- Written as a paragraph not bulleted list

✓ Writing
- Memo is organized in a logical order
- Writing is direct, concise and to the point
- Written in first person from lab partners
- Correct grammar, no spelling errors

✓ Content
- Starts with a statement of purpose
- Discusses the strategy or pseudocode for implementing the robot paths (may include a flow chart)

- Discusses the tests and methods performed
- States the results including error analysis
- Shows data tables with error analysis and required plots or graphs
- Answers all questions posed in the lab procedure
- Clear statement of conclusions
- Include software design plan in the appendix (reference design plan in the text)
- Include screen shot of simulator results or code in the appendix and reference in text

Questions to Answer in the Memo:

1. What was the general plan you used to implement the random wander and obstacle avoidance behaviors?
2. How did you create a modular program and integrate the two layers into the overall program?
3. Did you use the contact and IR sensors to create redundant sensing on the robot's front half.
4. How could you create a smart wander routine to entirely cover a room?
5. What kind of errors did you encounter with the obstacle avoidance behavior?
6. How could you improve the obstacle avoidance behavior?
7. Were there any obstacles that the robot could not detect?
8. Were there any situations when the range sensors did not give you reliable data?
9. How did you keep track of the robot's states in the program?
10. Did the robot encounter any "stuck" situations? How did you account for those?
11. How did you keep track of the goal position and robot states as it integrated avoid-obstacle and go-to-goal behaviors?
12. What should the subsumption architecture look like for the addition of the go-to-goal and avoid-obstacle behaviors?
13. Compare the performance of your robot to the theory and software design plan you made.

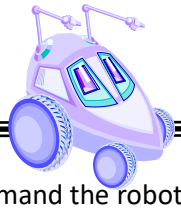
Grading Rubric:

The lab is worth a total of 30 points and is graded by the following rubric.

Points	Demonstration	Code	Memo
Excellent 100%	Excellent work, the robot performs exactly as required	Properly commented with a header and function comments, easy to follow with modular components	Follows all guidelines and answers all questions posed
Average 75%	Performs most of the functionality with minor failures	Partial comments and/or not modular with objects	Does not answer some questions and/or has spelling, grammatical, content errors
Poor 50%	Performs some of the functionality but with major failures or parts missing	No comments, not modular, not easy to follow	Multiple grammatical, format, content, spelling errors, questions not answered
Missing	Meets none of the design specifications or not submitted	Not submitted	Not submitted

Upload Details:

You must submit you properly commented code to Moodle DropBox and lab memo (worksheet) to GradeScope by **11:59 pm on Sunday**. Your code should be modular with functions and classes in order to



make it more readable. You should use wireless communication to command the robot and indicate the robot state during program execution.