University of Tennessee, Knoxville
Trace: Tennessee Research and Creative Exchange

University of Tennessee Honors Thesis Projects | University of Tennessee Honors Program

5-2012

Line Following Navigation

Nicole Marie Pennington
npennin2@utk.edu

Follow this and additional works at: http://trace.tennessee.edu/utk_chanhonoproj

Part of the Robotics Commons

Recommended Citation
Pennington, Nicole Marie, "Line Following Navigation" (2012). *University of Tennessee Honors Thesis Projects.*
http://trace.tennessee.edu/utk_chanhonoproj/1554

This Dissertation/Thesis is brought to you for free and open access by the University of Tennessee Honors Program at Trace: Tennessee Research and Creative Exchange. It has been accepted for inclusion in University of Tennessee Honors Thesis Projects by an authorized administrator of Trace: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

My responsibility to the IEEE Robotics Team was to design and develop a navigation scheme with the assistance of Chad Armstrong. I was relied upon to choose a sensory/navigation method, implement an appropriate sensor layout, and integrate that sensory method into a navigation algorithm, which I also needed to develop.

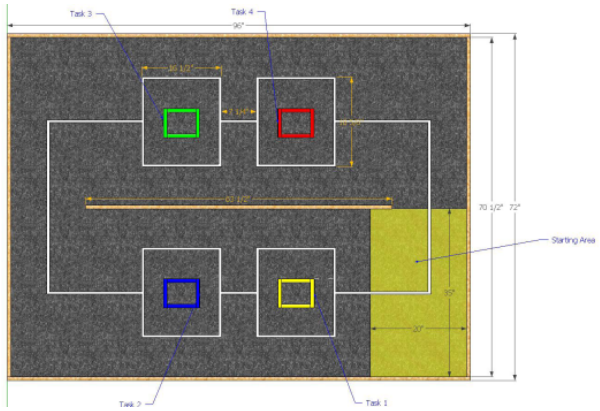
I. Navigation Method

The robotics competition required an autonomous robot to navigate its way around a course. There were several ways to approach this problem, but I narrowed down my options to the following:

- Image capture and processing using a webcam, android device camera, or Gameboy camera
    - Complex, expensive
- Magnetometer and encoders
    - Navigation based on angle and distance from starting location
- Line following using infrared emitter and detector line sensors
    - Various examples available from previous years
    - Economical, reliable

While some combination of these may have provided the best navigation, I decided upon a simple line following scheme for several reasons:

1. Time constraint. We began working on this robot in January and thus only had 2.5 months to work on it. Navigation would ultimately rely heavily on the size, shape, and mobility of the robot, so it needed to have a working robot upon which to test. This meant that, realistically, I would have to wait until there was at least a chassis and simple motor functions available. Given this, I knew the most simple navigation implementation had the greatest chance to be done in time to compete.
2. Simplicity. Line following was a simple and reliable scheme, with sensors available which needed little to no input processing. Essentially, I could order the sensors, solder on leads, connect them to the arduino, and instantly begin detecting lines.
3. Course Design. The course had a simple rectangular layout with well-defined white lines on a dark grey carpet. The starting area (shown in yellow) included plenty of space for the robot to start on the line. This layout was conducive to line following in that it guaranteed successful navigation around the course as long as the robot could stay on the line (or at least get back to it).



II. Line Sensors

I decided to use the QRE1113 Line Sensor Breakout from Sparkfun, which came in an analog and digital variety, as explained below. These were reliable, cheap sensors with the following important characteristics:

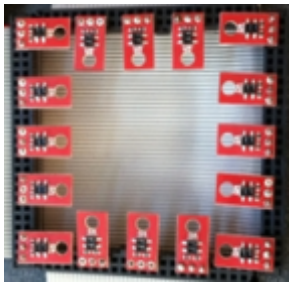
- Dimensions: 0.3" x 0.5" x 0.1" (without header pins installed)
- Operating voltage: 5.0 V
- Supply current: 25 mA
- Optimal sensing distance: 0.125" (3 mm)
- Maximum recommended sensing distance: 0.375" (9.5 mm)

A. Analog Line Sensor

The analog line sensor was available to us from the start from a previous year's robot. This was the first sensor I used in testing, and it proved to have a very simple reading scheme in that it was a simple function call in the code. In this sensor, a voltage change due to the amount of infrared light reflected back to the sensor varies the output of the sensor from 0 to 1023, where 0 is white and 1023 is black (no light). This was the first sensor chosen for use in the robot, and part of the decision to use an Arduino Mega board was that it had 16 analog ports available for use, which fit our orientation design decision (see Sensor Orientation).

B. Digital Line Sensor

The digital line sensor was initially decided against due to its much more involved sensor reading. The output of each sensor needed to be processed in a separate function before the sensor's reading could be determined. This was due to the fact that its reading was measured based on the discharge time of a capacitor. Even so, it was found that the digital sensor generally had better discrimination between the carpet and line in that there was a greater threshold between the two. Eventually, we decided it was necessary to add more sensors to our orientation because of the robot's large size, and since the analog ports were all in use, we added 4 digital sensors to the design.



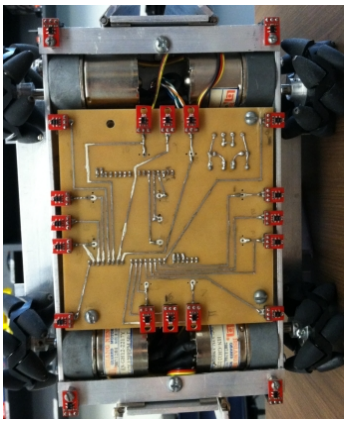
III. Sensor Orientation

The orientation of the line sensors was another important consideration. The initial movement scheme for the robot included lateral and backward movements, so I wanted a symmetric orientation to allow easy implementation of navigation in various directions. I also knew we needed several sensors spread across each side in order for the robot to successfully follow the course lines without losing them. With this in mind, I decided upon an initial sensor orientation consisting of 16 analog sensors in a tight square formation, as shown here. We actually saw other teams using this sensor orientation design at the competition.

Later, it became clear that this orientation would not be sufficient for successful course navigation for several reasons. One reason was the size of our robot. For this orientation to be effective, it needed to have small (1/2") spacing between the sensors to ensure that at least two sensors would be reading the 3/4" course line. However, it also needed to be close enough to the front, back, and sides of the robot to ensure that the robot could detect when it had reached a "T" in the line before a box without overshooting that line too far.

Another major factor was the speed of our robot. It was generally moving fast enough that the robot would shoot well over the line before the sensors had time to register the line and send communication to the motors to stop. Because of this, we knew that we needed sensors at the very front and back of the robot.

Our final design, shown here, was placed on a custom-etched board and included 16 analog sensors and 4 digital sensors. This design was essentially an expanded version of the square, but proved to have enough coverage for effective navigation and line following. The far corner sensors were used to detect the "T"s at boxes, the inner corner sensors were used to detect the end of a turn, and the inner front sensors were used during deployment. The side sensors were used to detect when the robot reached a 90 degree turn. The symmetry allowed the robot to easily switch from moving forwards to moving backwards.



IV. Navigation Code

A. Programming Interface

The team met several times during the semester to discuss and decide upon an API for our various components. Our modular design efforts necessitated communication to ensure that the robot would come together nicely in the end. The motor movement API needed to include an interface that the navigation controller could use to indicate a direction of movement, a speed, and in the case of turns, a harshness of turning. See Ammar's work on the motor controller API.

This robot also required a deployment API, since we were using servos to deploy the sensors. Deployment was one of the last things added to the robot before competition, so the API needed to be integrated quickly. The following functions were provided by Ryan Young for use with the deployment servos:

load() - opens the chambers to allow for sensor loading
grasp() - grips the loaded sensor
drop() - allows the bottom sensor to drop to the ground
release() - opens the deployment chamber wider to ensure that the sensor will be left at the box
reload() - drops the top sensor to the bottom chamber so that it is ready to be deployed
relax() - pulls both top and bottom chambers closed after deployments to ensure that they will not be in the way

B. Finite-State Machine Design

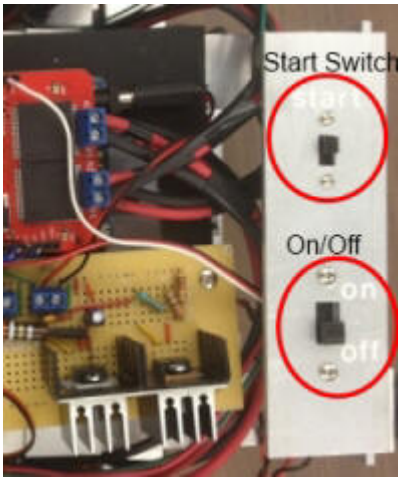
Our navigation scheme was a finite-state machine design, such that the robot existed only in the a set number of states. This was implemented very easily within the main loop() function of the Arduino as follows:

```
void loop(){
   readSensors();
   condition();
   if(following) line_follow();
}
```

This very simple loop was an elegant design which polled the sensors, checked the line conditions, and followed the line in one continuous loop. As you can see, line following could also be turned off. This was useful when navigating around a box, since the robot had very little space to travel before the next turn and did not need to be trying to correct itself in that short range. The sensor reading function polled all sensors and assigned a binary value based on a threshold determined for the line.

The condition function was actually a function pointer, implemented such that each state of the robot was looking for a specific sensor to read the line before it changed to the next state. These various functions were active during specificed segments of the course and turned line following on and off as needed. For example, there was a reached_t() condition which polled the far corner sensors until one of them read a line. The robot began in this state, since it needed only to follow the starting line until it reached the "T" at the first box. A count of the boxes was also kept, so that the first four boxes initiated a deployment function while the subsequent trips to each box

caused the robot to simply poll the deployed sensors in order to make a decision on which way to go around the box.

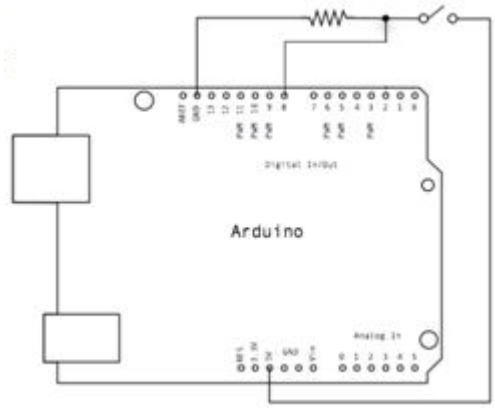


V. Miscellaneous

A. Start Button

Early on in our testing, we encountered an issue with motor communication in which the robot would continue its last movement indefinitely after being turned off then back on. To fix this, we had to connect our laptop to the arduino and open the serial monitor, which reset the boards. This became a tedious extra step in testing, so we decided to create a second button in addition to the On/Off button, which we called the Start button. After flipping the On switch, the arduinos were receiving power but would not move until the Start switch was also flipped.

This was achieved by including "while(digitalRead(buttonPin) == LOW);" in the setup() function of the robot's primary controller. Our buttonPin was digital pin 12 on the Arduino Mega and was connected according to the schematic on the left.



We also decided later on to integrate the delivery system such that the first time we flipped the Start switch, the robot would hold open its deployment servos to allow for easy sensor loading. The next time the Start switch was flipped, the servos would close, holding the sensors in place and ready. A third flip of the Start switch then caused the robot to begin navigating the course.

B. Sensor Test Code

I also implemented test code for the sensors, which was essentially just a serial print that would display the values of the sensor readings with some small delay to allow time for them to be understood. With this code, we could connect our laptop to the Arduino and move the robot slowly across the line, watching the sensor values on the serial monitor to ensure that they were all functioning properly. This test code also served to give us a good threshold value upon which to base our decision of what is line and what is carpet.

C. Turning Calibration

Many, many hours were spent in the lab fine-tuning the pivot turns. Each left/right turn needed a special pivot point in order to ensure that the turn was executed smoothly. This process became rather tedious, since the harshness of turns was largely influenced by the power being sent to the motors.