12 Lab 4 CS 123 Computation Lab III Spring 2010
Directions and Problems

12.1 Overview

At this point, we've seen many examples of how computer programs are used to do time simulations of various physical phenomena. Another important application of computer programming is control of devices. We've constructed a Maple package that supports simulation of driving a car under computer control. The programming for the control is similar to but not the same as the NXT robots you built earlier this year. The simulator world does not have the same difficulties with wheel slippage and motor imprecision that a real robot would have which makes it easier to develop and debug more complicated programs because there are fewer causes of error to deal with. However, the "simulator world" is less realistic in that a program that successfully navigates the simulator world may still have problems with a real-world robot. Building a more realistic simulation is feasible but we have left those details because getting good control of a car even in the idealized world of this simulator is hard enough for a two hour lab. This lab presents an experience with controller programming, using an API (application programming interface). It gives you a chance to practice your software development skills by controlling a car simulator that behaves suspiciously like a NXT robot. However, since it's a simulation, you don't have to spend a lot of time with mechanical breakdowns or resets.

12.2 Pre-lab preparation

1. Reading: chapters 19 and 20. Review older chapters and labs as needed.

2. Read the lab directions and download the simulator api. See whether you can run it in your own copy of Maple following the directions for installation of the API contained in the lab directions below.

5. Take the pre-lab quizlet 4 at the CS 123 Maple TA web site. You should do quizlet 4 before lab to be prepared for the lab activities.

12.3 Introduction to the problem setting

The Defense Advanced Research Projects Agency DARPA has in the past few years sponsored a contest for "autonomous vehicles" -- cars that drive themselves. There are no humans inside the cars, but there is a "chase car" nearby with an emergency cut-off switch just in case. The driving range for the contests in progressive years has becoming increasingly more difficult, from open countryside, to urban traffic navigation. You can read about the 2007 contest at http://www.darpa.mil/GRANDCHALLENGE/ . That site has links to previous years' contests. Here are some pictures from the contest:

<table>
<thead>
<tr>
<th>Scenes from a race</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.jpg" alt="Picture 1" /></td>
</tr>
<tr>
<td><img src="image2.jpg" alt="Picture 2" /></td>
</tr>
<tr>
<td><img src="image3.jpg" alt="Picture 3" /></td>
</tr>
</tbody>
</table>
There are many as-yet unsolved aspects to building a controller for a car that can deal with "real world" traffic, which is why DARPA funded it as a significant science and engineering challenge to universities. The winning team from Carnegie Mellon in 2007 won $2,000,000 for placing first. In this lab, we look at some aspects of the computing involved. Since good driving depends on road, vehicle, and traffic conditions, it's clear that any autonomous vehicle could not function on a fixed strategy. In other words, it would have to carry on-board some computer programming.

The purpose of this lab is for you to practice with using what's been covered in the past 12 labs to handle some autonomous vehicle scenarios. We introduce a simulator to develop simple driving software, and an Application Programming Interface (API) -- a package of procedures that simplify the process of writing programs that control the car.

12.4 Problems

Problem 1

Let's see a quick demo of the simulator. Download the lab 4 materials from the class website. You should see several Maple worksheets that you will use for various parts of the lab. (a) Open and execute the worksheet named Lab4Tutorial1.mw. Within the worksheet, you should see lines of code that:

1. Establish initial values for Maple variables used in the simulation. This initialization must be each time you want to use the simulator, much as you have done actions such as \( t := \text{table}() \); each time before you execute a loop to accumulate results.

2. A coordinate for a target for the car to get to.

3. Establishing the limits of the "playing field" for the simulator. You can visualize this as a wall of specified dimensions that the car will run into if it goes too far.

4. Writing a Maple procedure that describes actions that the car will take. Note that the sample program, run1a, contains a few actions from the car simulator package \textit{move}, and \textit{turn}. The actions are performed with the help of loops (for multiple moves). Note that the car is pinkish when it is running but turns gold when it finishes. This is what it does when it stops next to a target square (blue). The car should automatically stop when it is pointing at and next to a target.

(b) Open and execute the worksheet named Lab4Tutorial2.mw. This worksheet explores the state table, which explains the position and condition of the car at points of time.

(c) Open and execute the worksheet named Lab4Tutorial3.mw. Study the program and the state table generated by it.

Then answer the following questions:

Suppose we run a simulation and notice the following entry in the state table after we do so:

\[ 6 = [3,2, \pi/2, \text{CarSimulator}:-\text{CARFINAL}] \]

(i) What is the coordinates of the car at that point?
(ii) Is the car (as indicated by the arrow) pointing right, left, up, or down?

(iii) What do you think the 7th entry of the state table (the one that will be 7=\ldots\) will be like? (Yes, this is a trick question -- but I guess this admission means that it isn't really.)

(d) **Open the worksheet** named Lab4Problem1.mw **Solve** the problem in it, which asks you to write a program to move a car forward three steps and then play the animation of it. To get points for this part, you should be able to demonstrate the program you wrote and explain in your own words what it does.

To get credit for this problem, you should be able to demonstrate your simple program for (d) and explain you understand it, such as the details of how you developed it, and what the various instructions in the program are doing.

**Problem 2**

The CarSimulator package includes some help pages on the various features of the package. Activate Maple on-line Help. In the "Table of Contents" pane on the left hand side you should see a folder for the CarSimulator. (Note: CarSimulator help should be available after you have done part 1, because some of the actions in Part 1 tell Maple to look in the current directory for help. If not, open again one of the Tutorial worksheets from problem 1.1 and then activate Maple help again.)
The documentation for this was written by course staff and included in the CarSimulator.hdb file that you downloaded with the other lab files.

(a) **Open and execute** Lab4Tutorial4.mw. Study the control program which puts the car through some moves.

(b) **Open and complete** the programming found in Lab4Problem2.mw.

To get credit for this problem, you need to demonstrate the program you wrote for (b) and be able to explain what it is doing and how you did it.

**Problem 3**

(a) **Open** Lab4Tutorial5.mw and **execute** the code block in it. After observing what happens and consulting the CarSimulator help documentation to better understand the code, answer the following questions:

(i) What color is the car when it is in the following car states: CARNORMAL, CARFINAL, CARBUMPED, CARSTOPPED? (ii) Describe what happens to the car if it is facing a wall and you tell it to move forward?

(iii) Suppose the car is in the following configuration:

![Car Configuration Diagram]

(It is facing upwards, and is adjacent to an obstacle. You then do isTouching(Pi/2, 'stateOfCar'); According to the documentation, isTouching should return true or false. What does it do in this case? What do you think the value of the variable stateOfCar will be?)

(b) Solve the programming problem in Lab4Problem3.mw.

In order to get credit for this problem, you should be able to answer the questions of part (a) plus related ones associated with the CarSimulator API, and explain your working program for part (b).
Problem 4

Solve the programming problem in Lab4Problem4.mw.

To get credit for this problem, you should be able to demonstrate your working program, be able to explain what issues you faced in developing it, and that you understand how the program works and how it solves the problem.

Problem 5

Solve the programming problem defined in the worksheet LabProblem5.mw.

You should be able to demonstrate your solution, explain how you developed the solution, and how the program solves the problem.

12.5 Afterword: Comparison of NXT and Maple as programming languages for control

NXT for Lego robots and the CarSimulator package in Maple present different approaches to writing computer programs that control a device. Mindstorms' visual interface allows for "wired together" actions configured by point-and-click menu selection that move a robot. The Maple CarSimulator uses textual specification for actions and configuration. Repetition (handled by "for" in Maple) and conditional decision-making (handled by if or while) are present in both languages as a way of expressing control. A significant difference is that the car simulator takes a "move one unit at a time" approach to motion whereas control of a real robot typically engages motion or a motor continuously until another command disengages the action.

The textual interface tends to be preferred by programmers for larger control programs. It is usually faster to enter control programs (or to modify them) through textual entry than through clicking -- assuming that you are reasonably proficient at typing and meeting the syntactic exactitude required when communicating in a programming language. It also tends to be easier to do transformations of textual programs than visual programs. For example, if you decided that you wanted to modify a procedure so that all the turns to the right are replaced by turns to the left, you could find and modify all relevant instances of turn textual search and replacement. Doing a similar transformation in a visual program can often be much more laborious.

The Maple CarSimulator does not control a real robotic car. Nevertheless it and other software control simulators have their use in engineering applications because it's usually much faster to develop ideas and principles using simulator results, than it is to set up and test a physical device. For example, in a single lab period, you've been able to get in many more test runs with the simulator than were possible with running a physical robot. Even if the simulation is not perfectly realistic, there are typically many problems that can be worked out using them.

Textually-entered languages such as Java, C or Python can be used to program NXT robots. Maple could be as well. For the reasons we have mentioned, extended projects with robots such as solving and writing the solution to Sudoku puzzles (http://www.gizmowatch.com/entry/lego-mindstorms-nxt-robot-tailored-to-solve-sudoku-puzzles/) or manipulating and solving a Rubik's Cube (http://www.youtube.com/watch?v=3QpEG27Gt4) would be more quickly developed in such languages rather than the visual language introduced in ENGR 102 at Drexel.