Chalmers University of Technology
Department of Applied Mechanics

Home problems, set 1, FFR125/FIM760 Autonomous agents 2008

General instructions

Home problems set 1 consists of three parts. The maximum number of points is 10, and a minimum score of 4 points is required in order to pass. Incorrect problems will *not* be returned for correction, so make sure to check your solutions and programs carefully before submitting them. In order to maximize the utility of your efforts, *carefully* follow the instructions given below:

1. Hand in your solutions, via email to `krister.wolff@chalmers.se`, no later than 17.00 on 2008-02-15. Late hand-in will result in a deduction of points!

2. Make sure to include your name and (if available) civic registration number (personnummer).

3. Read the problem formulations carefully, to make sure that you provide clear answers to all questions.

4. The solutions should be given in the form of a brief report in PDF, PS, MS Word, or ODT format, for each problem. The programming code alone is *not* to be considered as a report!

5. In analytical problems make sure to include all the steps of the calculation in your report, so that the calculations can be followed. Providing only the answer is *not* sufficient. Whenever possible, use symbolical calculations as far as possible, and introduce numerical values only when needed. All mathematical expressions in the report *should* be formatted (using e.g. MS equation editor).

6. In problems requiring programming, it should *not* be necessary to edit the programs. Programs that do not function or require editing to function will result in a deduction of points.

7. Make your *own* solutions! You may, of course, discuss the problems with other students. However, each student *must* hand in his or her *own* solution. In obvious cases of plagiarism, points will be deducted from all students involved.

8. Collect all your files in *one* zip file which, when opened, generates one subdirectory for each problem, i.e. `Problem1.1, Problem1.2, Problem1.3`.

Deadline: 20080215, at 17.00!

GOOD LUCK!

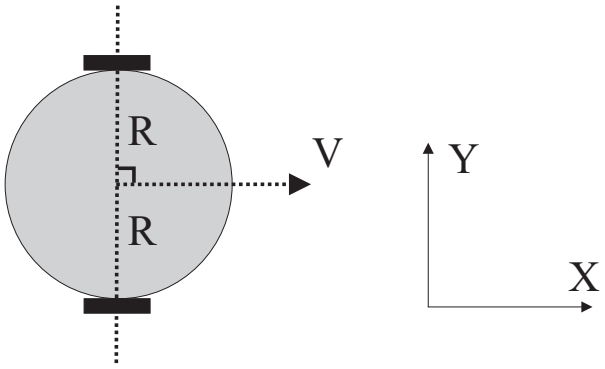

Figure 1: The differentially steered robot considered in Problem 1.1

Problem 1.1, 4p, Basic kinematics

Consider the two-wheeled differentially steered robot shown in Fig. 1.

(a) (2p) If the wheel speeds and the radius (R) of the robot are known, its position can be obtained using the kinematic equations derived in Chapter 1. Consider a case where the robot starts at $(x, y) = (0, 0)$, heading in the positive x direction ($\varphi = 0$) as shown in the figure. Assuming that the wheel speeds are constant and given by

$$v_L(t) = v_1, \tag{1}$$

$$v_R(t) = v_2, \tag{2}$$

derive a general expression for the position (x, y) and direction of heading φ for the robot, as functions of time, and show that the resulting trajectory is circular (if $v_1 \neq v_2$). What is the radius of the circular trajectory?

(b) (2p) Write a Matlab program for numerical integration of the kinematic equations (1.18) in Chapter 1. Consider a case in which the wheel speeds are given by

$$(v_L(t), v_R(t)) = \begin{cases} (\frac{t}{t_1}, \frac{t}{t_2}) & \text{if } 0 \leq t \leq t_1, \\ (1, \frac{t}{t_2}) & \text{if } t_1 < t \leq t_2. \end{cases} \tag{3}$$

Assuming that the radius of the robot is equal to 0.5 m, determine (numerically) values of t_1 and t_2 that will place the robot at $(x, y) = (1.93, -2.16)$ at $t = t_2$ given that the robot starts (at $t = 0$) in $(x, y) = (0, 0)$ with $\varphi = 0$. What is the final value of φ? Also, plot the trajectory $(x(t), y(t))$ for the robot.

The interface to your Matlab program (which you should submit along with your report), called `IntegratePosition` should be as follows

```
>> [x, y, phi] = IntegratePosition(t1,t2);
```

where (x, y) and φ are the final values of the position and the heading of the robot, respectively. (The "," on the left hand side might not be needed in some versions of Matlab).

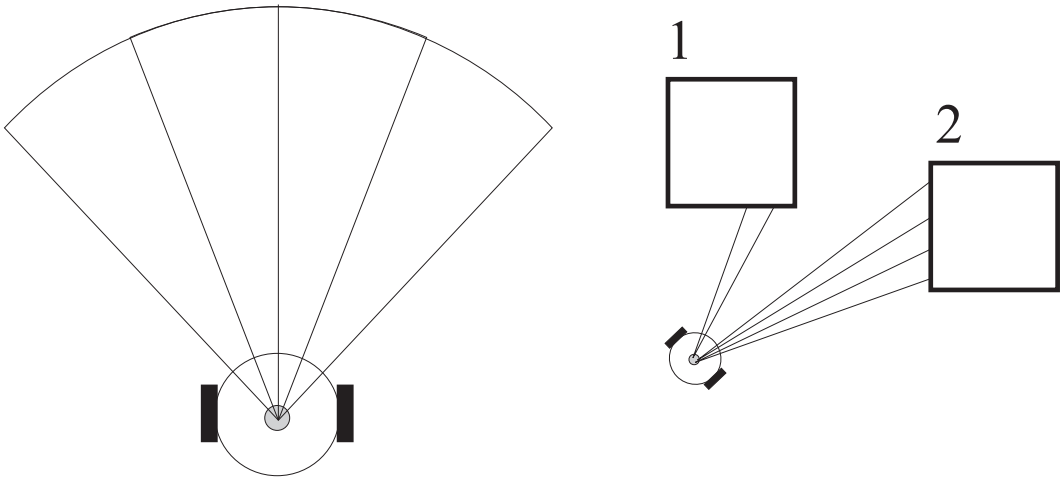

Figure 2: A two-dimensional laser range finder.

Problem 1.2, 3p, Simulation of laser range finders

A laser range finder is a sensor that uses time-of-flight measurements of reflected laser beams in order to obtain a distance map over a large range of angles. Both 3D and 2D laser range finders exist, but here we will only work with the 2D version. Consider the robot shown (from above)in Fig. 2. A 2D laser range finder, with opening angle γ is mounted (centrally) on top of the robot. The laser range finder sends out N rays, separated by the angle $\delta\gamma = \gamma/(N-1)$. The range of the sensor is L. Thus, for objects located at a distance $r \leq L$ (along a ray), the sensor returns r. If no object is detected along a ray, (i.e. if $r > L$), the sensor returns a 0. Hence, the complete reading of the laser range finder will be a vector with N elements, containing distance values for the N rays.

(a) Write Matlab functions implementing this type of laser range finder. First, write a function for creating the sensor, namely:

```
lrf = CreateLaserRangeFinder(name,relativeangle,size,nr,openingangle,range,L);
```

where `relativeangle` determines the forward direction of the sensor (i.e. the direction of the central ray, if N is odd), `size` the size of the visual representation (in Matlab), and `nr` is the number of rays. Next, write a function for reading the sensor, namely

```
lrf = GetLaserRangeFinderReading(lrf, arena);
```

The sensor should send out `nr` rays, and obtain distance readings as described above (note: you do *not* need to model the time-of-flight - it is sufficient simply to measure directly the distance to the nearest object along the ray). You may use the functions `GetDistanceToLineAlongRay` and `GetDistanceToNearestObject` implemented in `ARSim`. Finally, modify the Matlab functions `InitPlot` and `ShowRobot` so that your laser range finder is plotted as a filled circle, and the rays as lines, visible only for rays such that $r < L$.

(b) Dissect the functions `GetDistanceToNearestObject` and `GetDistanceToLineAlongRay`, and explain (by drawing appropriate figures and explaining the equations) *in detail* how they work.

(c) Finally, use your laser range finder together with the functions available in `ARSim` to obtain the readings of the laser range finder in the situation shown in the right panel of the figure. The robot is located at $(0, 0)$, with its forward direction pointing in the direction $\varphi = 45$ degrees, and the central ray of the laser range finder pointing in the same direction. The lower left corner of arena object 1 is located at $(-0.5, 2.0)$. The lower left corner of arena object 2 is located at $(3.5, 1.0)$. Both objects are squares with side length $a = 1.5$m. Let the opening angle of the sensor be 45 degrees, and assume that the number of rays is equal to 9 and that the range L is equal to 5.0 m. Note that the figure is schematic (not to scale). Do not forget to submit the Matlab files, as described in the general instructions above.

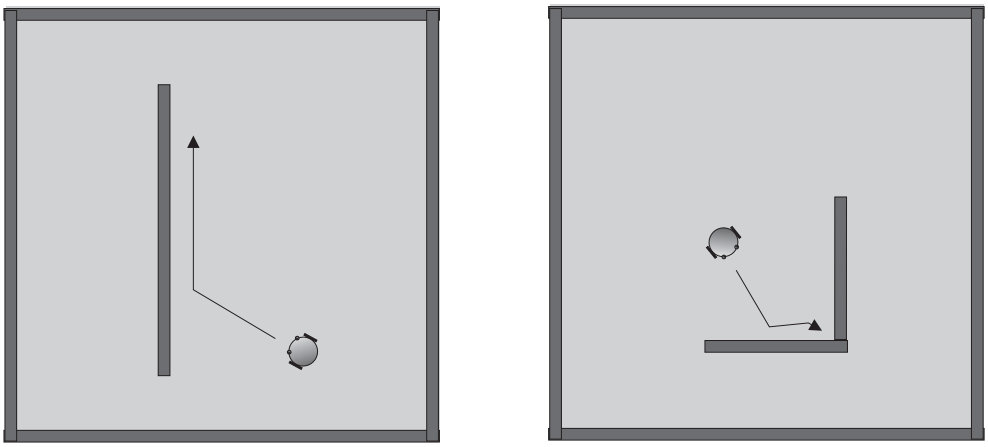

Figure 3: Illustration of the behaviors in problem 1.3.

Problem 1.3, 3p, Simple hand-coded behaviors

Simple behaviors for autonomous robots can sometimes be generated by hand. In such cases, a behavioral representation that is easy to interpret is, of course, chosen. In this problem, we will use `if-then-else`-rules as the behavioral representation. Thus, each behavior should consist of conditions (involving e.g. sensor readings) of the general form

```
if (CONDITION)
 <ACTION1>
elseif (CONDITION2)
 <ACTION2>
else
 <ACTION3>
end
```

This is just an example: Of course, the number of actions can vary from case to case, as can the complexity of the conditions. It is also possible, in principle, to have several layers of nested `if-then-else`-clauses.

(a) Use a behavioral representation of the kind above to generate the behavior *wall-following*. When placed in a arena, containing only walls (i.e. no other obstacles) the robot should first find a wall, and then follow it, as exemplified in the left panel of Fig. 3. No particular action need be taken when the robot reaches the end of the wall (i.e. you do *not* need to implement collision avoidance). (1p)

(b) Again using an `if-then-else`-representation, generate a behavior for *corner-positioning* (you may, of course, start from the behavior generated in (a)). In this problem, the robot should find a wall, follow it until it encounters a perpendicular wall (assuming that walls are placed in a perpendicular fashion in the arena), and *then* adjust its heading so that the front of the robot points directly towards the corner, i.e. at a 45 degree angle (or as close as possible) to either wall. The behavior is illustrated in the right panel of Fig. 3. (2p)

For these problems you *should* download and start from `ARSim` (unchanged) from the web page, and you should modify (and hand in) the following files: `CreateBrain.m`, `BrainStep.m` and `TestRunRobot.m`. These files will be copied into `ARSim`, and the submitted behaviors will then be tested. Any number of variables may be introduced in `CreateBrain` and `BrainStep`. You should also submit a report describing your behaviors, with a figure illustrating each behavior, along with a description of how the behavior is intended to work. Note: you may modify the robot in `TestRunRobot` e.g. by adding (or removing) IR sensors. However, the sensor range should be *at most* 0.8m, and the arena size should be at least 4x4 meters. Note also that the only information that may be used (from an IR sensor) is the fuzzy reading obtained from the variable `Reading` contained in the IR sensor. Also, it is *not* allowed to use odometers and compasses; only IR sensors are allowed in this problem.