FFR 125, Autonomous agents, 2016: Home problem 1

General instructions. READ CAREFULLY!

Home problem 1 consists of three parts. Problems 1.1 and 1.2 are mandatory, problem 1.3 is voluntary (but check the requirements for the various grades on the web page). **Note!** Before submitting your solutions and your report, make sure to download and *read carefully* the checklist for home problem submission, available at

`www.me.chalmers.se/~mwahde/courses/aa/2016/Checklist_HPSubmission.pdf`

When programming, you should use Matlab. Make sure to follow the Matlab coding standard, available at

`www.me.chalmers.se/~mwahde/courses/aa/2016/MatlabCodingStandard.pdf`

During correction of the home problems, it will be assumed that you have read and followed the information in the two documents listed above. If you have any questions regarding the coding standard or the submission procedure, please ask *before* submitting your solutions.

You should provide a single report (for the whole problem set) in the form of a PDF file. In the case of analytical problems, make sure to include *all the relevant 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. You should write the report on a computer, preferably using LaTeX (see `www.miktex.org`). Scanned (or photographed) handwritten pages are *not* allowed.

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.

Make sure to send the solutions and your report before the (strict) deadline. If the solutions are submitted after the deadline, only the mandatory problems (1.1 and 1.2) will be corrected. The submission time will be taken as the time when your last submission e-mail is *received*. Thus, make sure to include all solutions (as a single zip file, which should contain your report and the Matlab code) when sending the e-mail.

Make sure to have some (time) margin when submitting the solutions. Submitting a few seconds before midnight on the 11th *is not recommended.*

Deadline: 20160211, 23.59.59

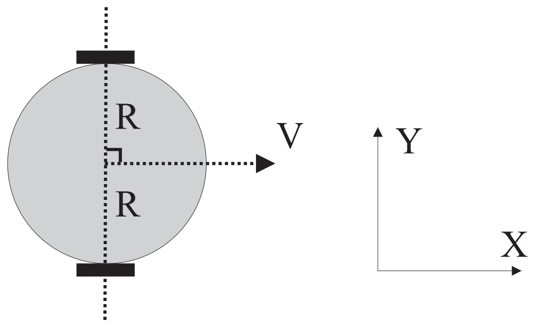

Figure 1: The differentially steered robot considered in Problem 1.1.

Problem 1.1, 4p, Basic kinematics

If the wheel speeds and the radius (R) of a two-wheeled differentially steered robot are known, its position can be obtained using the kinematic equations derived in Chapter 2. Consider a case where the robot starts at $(x, y) = (0, 0)$, heading in the positive x direction ($\varphi = 0$) as shown in Fig. 1.

(a) (2p) Assuming that the wheel speeds vary as

$$v_L(t) = v_0(t/t_1), \tag{1}$$

$$v_R(t) = v_0(t/t_2), \tag{2}$$

where v_0, t_1, and t_2 are constants, derive (analytically, i.e. without using a computer!) a **general** expression for the position (x, y) and direction of heading φ for the robot, as functions of time, and plot the resulting trajectory in your report, for $t \in [0, 10]$ for the case $R = 0.25$ (m), $v_0 = 1.0$ (m/s), $t_1 = 10$ (s) and $t_2 = 5$ (s).

(b) (2p) Write a Matlab program for numerical integration of the kinematic equations (2.7)-(2.9) in Chapter 2. Consider a case in which the wheel speeds are given by

$$(v_L(t), v_R(t)) = \begin{cases} (0, v_0\frac{t}{t_1}) & \text{if } 0 \leq t \leq t_1 \\ (v_0\frac{t-t_1}{t_2}, v_0) & \text{if } t_1 < t \leq t_2. \end{cases} \tag{3}$$

Assuming that the radius of the robot is equal to 0.25 m, determine (numerically, using your program) the position (x, y) and heading angle φ of the robot at $t = t_2$ for $v_0 = 1$ (m/s), $t_1 = 3$ (s), $t_2 = 10$ (s). Also, plot the trajectory of the robot in your report. The interface to your Matlab program (which you should submit) *should* be exactly as follows

```
>> [x y phi] = IntegratePosition(v0,t1,t2);
```

where (x, y) and φ are the final values (not the entire path) of the position and the heading of the robot, respectively. (In older versions of Matlab, you may have to write `[x, y, phi]` instead of `[x y phi]`. This is of course allowed).

Note: You should write your own function for integrating the kinematic equations. It is sufficient to use the simplest form of numerical integration, i.e. forming the integral (of a general function $f(t)$) by summing terms of the form $f(t)\Delta t$, where Δt is the time step, which should be sufficiently small (0.01 s or less, in this case) to maintain accuracy.

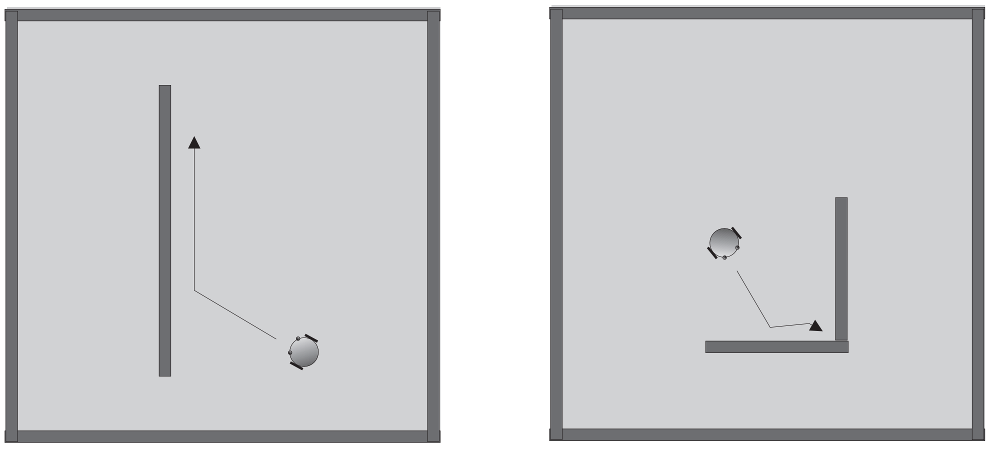

Figure 2: Illustration of the behaviors in problem 1.2.

Problem 1.2, 3p, Simple behaviors

Behaviors are often generated using an FSM (`if-then-else`-rule) representation, as described in Chapter 5.

(a) Starting from ARSim available at

`www.me.chalmers.se/˜mwahde/courses/aa/2016/ARSim_HP1.2.zip`

write a *wall-following* behavior, using the FSM structure described in Chapter 5. When placed in an 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. 2. 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 FSM 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. 2. (2p)

For each problem ((a) and (b)) you should only modify (and hand in) the following files: `CreateBrain.m`, `BrainStep.m`, and `TestRunRobot.m`. All other submitted Matlab files will be ignored. In `TestRunRobot`, you may modify the number of IR sensors, and their range. However, the sensor range should be *at most* 1.0 m, and the arena size should be at least 4x4 meters. You should generate a suitable arena (with both internal and external walls, as shown in the figure above). You may *not* modify the noise levels in the sensors and actuators. Also, it is *not* allowed to use odometers and compasses; only IR sensors are allowed in this problem, and you may only use the `Reading` variable of the sensors.

As in Chapter 5, all parameters and variables of the robotic brain should be defined in `CreateBrain`. You should also describe the behaviors in your report, with a figure illustrating each behavior, along with a description of how the behavior is intended to work.

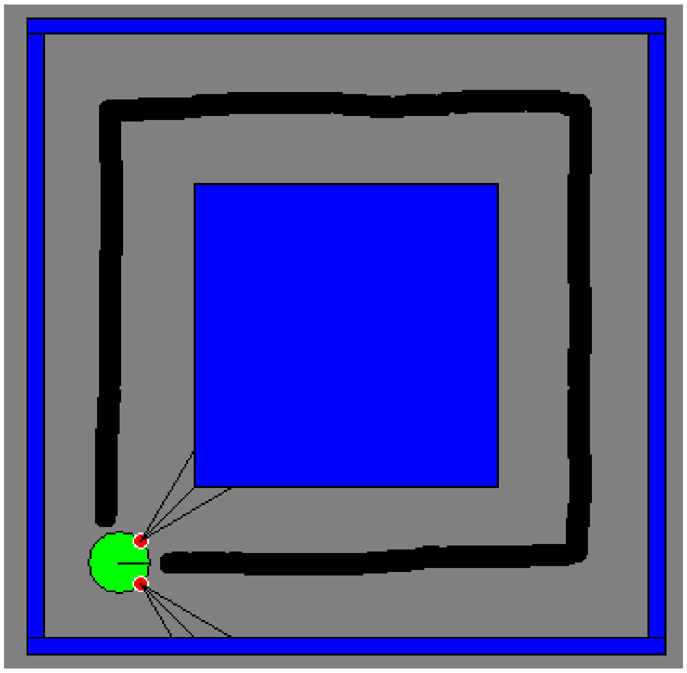

Figure 3: Illustration of the desired path (thick black line) for Problem 1.3.

Problem 1.3, 3p, Navigation using IR sensors and odometry

In this problem, a simple navigation problem will be considered, in which a robot uses odometry to estimate its pose, and IR sensors for detecting obstacles (and for positioning, see below).

You should write a robotic brain (in ARSim) that allows a robot to move twice around the arena, from a given starting point (-1.5, -1.5), and a given heading (0). You *should* start from the setup given at

`www.me.chalmers.se/˜mwahde/courses/aa/2016/ARSim_HP1.3.zip`

You may modify the number of simulation steps in `TestRunRobot`, but apart from that, you may *only* modify the `CreateBrain` and `BrainStep` functions. It is thus *not* allowed to modify the noise levels in sensors and actuators etc.

All variables (except local variables in `BrainStep`, see below) and parameters of the robotic brain *should* be introduced in the `CreateBrain` function (see the examples at the end of Chapter 5). Your `BrainStep` function should contain an FSM, i.e. a sequence of `if-then-else`-rules (again, see the two examples in Chapter 5), with clear and descriptive variable names. You may introduce local variables in the `BrainStep` function, but keep in mind that their values will not be available when the function is called again.

Your robot should try to follow the trajectory shown in the figure above. You may *not* (in this problem) recalibrate the odometry (except for the initial calibration, that occurs upon initialization). Because of the odometric drift, the odometry cannnot be used for absolute positioning. However, odometry *can* be used for making, say, a 90-degree turn, since the odometric drift over a single, brief motion is generally negligible. Thus, even though the absolute angle will be incorrect after a few turns, the relative angle (i.e. the angle after a turn minus the angle before the turn) will, in general, be rather accurate. You probably wish to introduce variables (in `CreateBrain`) for storing the initial pose (position and heading) before every maneuver in the simulation.

You can also use the IR sensors for orienting the robot correctly before and after turning (and, possibly, during the motion). For example, if the robot is placed at a 45 degree angle in front of a corner (where two walls meet at a right angle), the two IR sensors should, of course, give roughly equal readings (with slight variations due to noise). In order to minimize the effect of the noise during positioning (using the IR sensors), you may wish to briefly stop the robot in a given position, and then form (for example) a running time average of the IR sensor readings, which can be used to assess whether the robot's heading is as desired. Note that similar principles can also be used while moving in the (narrow) corridors: If the robot moves in the center of the corridor, the two IR sensors should give roughly equal readings.

The robot should go around the arena **twice** (not more, not less), and then stop as close to the initial pose (position and heading) as possible. Note: You may wish to measure the IR readings for a while (averaging to remove the effects of noise), and then store the values obtained for use at the end of the motion (if so, introduce the corresponding variable(s) in the `CreateBrain` function, as described above). Note that one lap around the arena requires four right-angle turns.

Note (Important!): When using the IR sensors, you may only make use of the fuzzy reading obtainable from the `Reading` field in the IR sensor `struct`. Specifically, you may *not* make use of the ray readings (which are only a computational tool used when forming the reading). **Students making use of the ray readings will get 0p on the problem, regardless of the performance of the robot**.

In your report, you should describe the behavior in detail. Furthermore, as for the program code, you should submit (only) the `CreateBrain` and `BrainStep` functions. Do *not* change the names of the `CreateBrain.m` and `BrainStep.m` files. All other submitted Matlab files will be ignored. Important: In your report, specify the maximum number of time steps (`maxSteps`) to be used in `TestRunRobot`, but do not include the `TestRunRobot.m` file. `maxSteps` should be sufficiently large to allow the robot to complete two laps around the arena, before stopping, as described above.