Chalmers University of Technology
Department of Applied Mechanics

Home problems, set 2, FFR125/FIM760 Autonomous agents 2008

General instructions

Home problems set 2 consists of four parts. The maximum number of points is 15, and a minimum score of 6 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 2008-03-10. 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. `Problem2.1, Problem2.2, Problem2.3, Problem2.4`.

Deadline: 20080310, at 17.00!

GOOD LUCK!

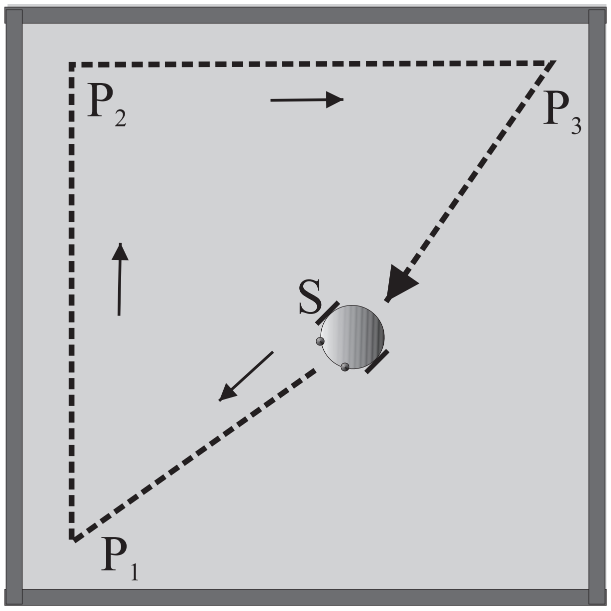

Figure 1: A robot navigating in a simple arena.

Problem 2.1, 4p, Navigation behavior

Consider the simple arena shown in Fig. 1. Using odometry, the robot shown in the figure is supposed to navigate from an *arbitrary* starting point S (see below) to the lower left corner (P_1), then to the upper left corner (P_2), then to the upper right corner (P_3), and, finally, return to the starting point S. The robot *should* recalibrate its odometry at P_1, P_2, and P_3, using the IR sensor readings (see below) and, possibly, the compass to estimate the position and heading, as accurately as possible, at these points. Implement the navigation behavior using IF-THEN-ELSE rules!

Note:

1. The arena should only contain walls (no obstacles), and its size should be 4x4 m.
2. The physical parameters of the robot (e.g. its size) should equal the default values in `ARSim`.
3. You should add (and use!) an odometer with the following settings:

   ```
   odometer = CreateOdometer('odometer',0.001,1.0);
   ```

   and a compass with the settings

   ```
   compass = CreateCompass('compass',0.005);
   ```

   Only the estimates provided by the odometer and the compass may be used by the robot during navigation. It is not allowed to use the actual position and heading of the robot.
4. You may add up to 6 standard IR sensors. Their range should be less than 0.5m, and their opening angle should be at least 0.5 radians and at most 1.5 radians. The `nr` property (of the IR sensors) should be at least 3. Only the contents of the `reading` variable of each IR sensor may be provided to the robotic brain. It is *not* allowed to use ray lengths etc.

5. For each run of the program, the starting point S (position *and* heading) should be chosen randomly, such that the distance between the robot and the nearest wall is 0.5m or more.

For this problems you *should* download and start from `ARSim, v.1.1.8` from the web page, and you should modify (and hand in) *only* the following files: `CreateBrain.m`, `BrainStep.m` and `TestRunRobot.m`. All other submitted Matlab files will be ignored. The three modified files will be copied into `ARSim`, and the submitted behavior 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 the behavior as a finite state machine, along with a description of how the behavior is intended to work.

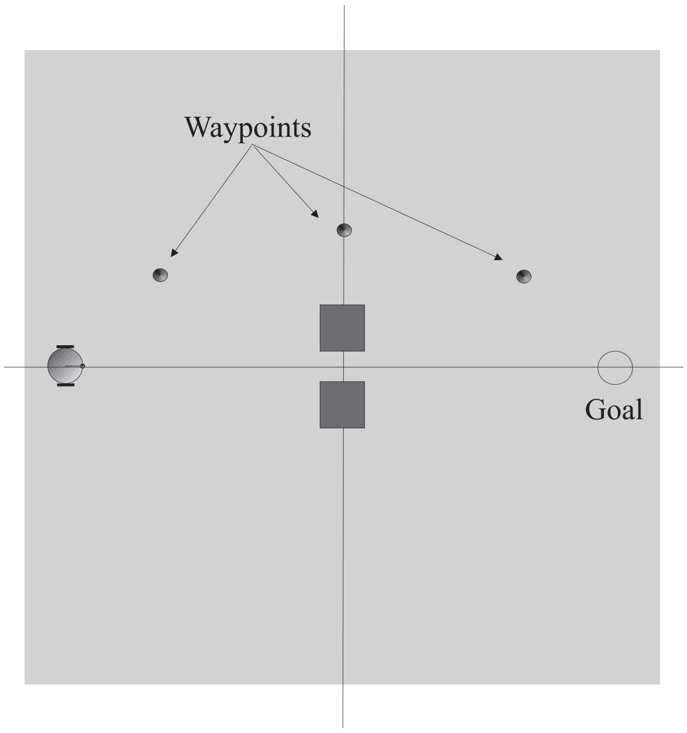

Figure 2: The arena used in problem 2.

Problem 2.2, 4p, Potential field navigation

Potential field navigation is a cooperative method for selecting behaviors (or, rather, actions) in robot navigation. Consider a robot, namely the standard robot in `ARSim` ($M = 3.0$ kg etc.), placed in the arena (without walls) shown in Fig. 2.

The task of the robot is to navigate from its starting point at $\mathbf{x} = (-3, 0)$ (heading in the positive x-direction, as shown in Fig. 2) to a goal located at $\mathbf{x} = (3, 0)$, with the potential

$$V_g = k_g(\mathbf{x} - \mathbf{x}_g)^2, \tag{1}$$

where $\mathbf{x}$ is the position of the robot and $\mathbf{x}_g$ is the position of the goal. There are two quadratic obstacles in the arena, each with side length 0.5 m. The potentials for the obstacles are given by

$$V_{o,i} = k_o \mathrm{e}^{-\frac{(\mathbf{x}-\mathbf{x}_{o,i})^2}{w_o^2}}, \quad i = 1, 2, \tag{2}$$

where k_o and w_o are positive constants, and $\mathbf{x}_{o,i},\ i = 1, 2$ denotes the location of the obstacles. The center of the upper obstacle is located at $(0.0, 0.4)$ and the center of the lower obstacle is located at $(-0.4, 0.0)$. Since the passage between the obstacles is too narrow for the robot to pass, some waypoints are introduced in the arena, namely at $(-2, 1)$ (waypoint 1), $(0, 1.5)$ (waypoint 2), and $(2, 1)$ (waypoint 3). The potentials of the waypoints are given by

$$V_{p,i} = -k_p \mathrm{e}^{-\frac{(\mathbf{x}-\mathbf{x}_{p,i})^2}{w_p^2}}, \quad i = 1, 2, 3 \tag{3}$$

where k_p and w_p are positive constants, and $\mathbf{x}_{p,i},\ i = 1, 2, 3$ denotes the location of waypoint i. Waypoint i should be removed if the distance $|\mathbf{x} - \mathbf{x}_{p,i}|$ is smaller than δ (a constant).

Implement the following navigation method for the robot, by modifying `CreateBrain.m`, `BrainStep.m`, and `TestRunRobot` (and *only* these files): After T seconds of navigation (see below), the robot should set its motor signals σ_L and σ_R to 0. Next, it should read off its location (you may use the exact position of the robot, without noise), determine the requested direction φ_{ref} (given by the normalized negative gradient of the potential field), set the motor signals as $\sigma_L = -\sigma_R = \pm\sigma_0$ (where σ_0 is a constant and the choice of sign ($\pm$) depends on the desired direction of rotation) and rotate until its direction is close to φ_{ref} (the exact direction of the robot may be used). Next, the robot should set the torque values to $\sigma_L = \sigma_R = \sigma_1$, where σ_1 is a constant (i.e. it should move along an asymptotically straight line) for another T seconds, at which point the whole procedure (stopping, determining the desired direction, rotating) should be repeated again etc. There is no time limit for the orientation procedure, but each navigation period, during which $\sigma_L = \sigma_R = \sigma_1$, should be of length T, as indicated above.

Determine values of the constants k_g, k_o, w_o, k_p, w_p, T, δ, σ_0, and σ_1 so that the robot can navigate safely from start to goal. Of course, there is no unique solution. You *may* use `ERSim` to find the parameter values, but using `ERSim` is *not* a requirement. In your report, you should list the parameter values, as well as the method you used for finding them. You should also submit `CreateBrain.m`, `BrainStep.m`, and `TestRunRobot.m`, which will be copied into `ARSim` for testing. You do not need to display the waypoints and the goal graphically.

Note! The current version of `ARSim` requires that a sensor should be present in the robot, at least if one wants to store the motion of the robot. In this problem, no sensors are needed. However, in order to be able to solve the problem by modifying only the three files listed above, you should add a dummy sensor, e.g. an IR sensor (with a single ray, say), whose readings are not used.

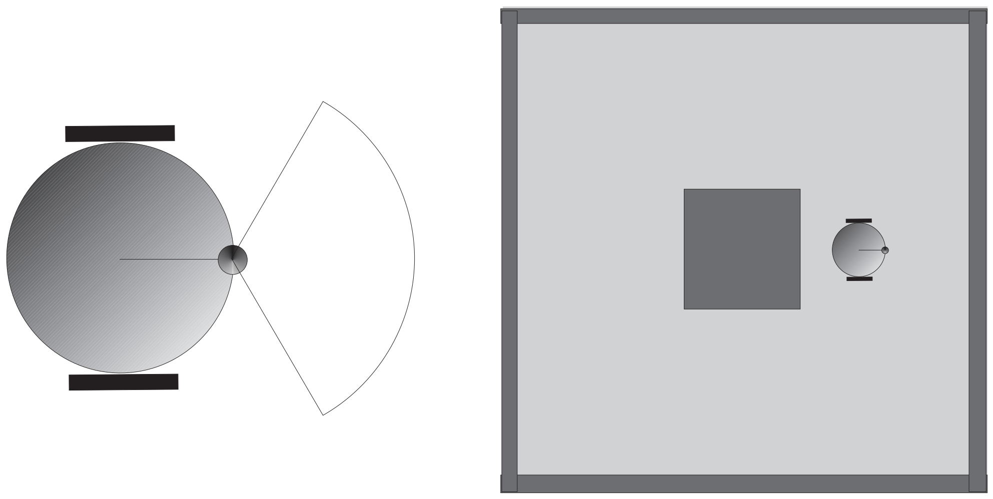
Figure 3: Robot and arena for problem 2.3.

Problem 2.3, 4p, Behavioral organization using the UF method

The utility function (UF) method can be used for behavioral organization in autonomous robots. Consider the case of a simple exploration robot, taken as the standard robot in `ARSim` (with $M = 3.0$ kg, radius $= 0.20$ m etc.), but with a single, wide-angle IR proximity sensor (opening angle $2\pi/3$ radians, range $= 0.8$ m), located on the edge of the robot's circular body, as shown schematically in the left panel of Fig. 3. Use `nr` $= 7$ rays in the IR sensor.

The size of the arena is 4×4 m (the walls should be defined as in the default `TestRunRobot` in `ARSim`), with a centrally located, quadratic obstacle with side length $L = 1.0$ m. The robot starts at $(x, y) = (1, 0)$, heading in the positive x-direction.

(a) Provide the robot with the two behaviors *straight-line navigation* (B1), in which the motor signals σ_L and σ_R sent to the two wheels are set to equal values, $\sigma_L = \sigma_R = 0.5$, and *obstacle avoidance* (B2), in which the two motor signals are set to the values $\sigma_L = -0.7, \sigma_R = 0.7$.

Assume that the state of the robot is given by the reading s of the proximity sensor, and an internal abstract variable x which is identically 0 in B1, and equal to 1 (constant) in B2. Assume further that the utility functions are given by

$$U_1 = a_0 + a_1 s + a_2 s^2, \tag{4}$$

for B1 and

$$U_2 = b_0 + b_1 x, \tag{5}$$

for B2. Implement (by hand) behavior selection based on the UF method, with B_1, B_2, U_1, and U_2 defined as above, by modifying the files `CreateBrain.m`, `BrainStep.m`, and `TestRunRobot.m` (and *only* these files).

(b) Next, let the fitness of the robot be the distance travelled in B1, i.e.

$$f = \sum_{i=1}^{N} \Delta S_i, \tag{6}$$

where N is the number of time steps in the simulation. $\Delta S_i = 0$ if B2 is active and equal to $(\Delta x_L + \Delta x_R)/2$ if B1 is active, where $x_L = v_L \Delta t$ and $x_R = v_R \Delta t$ are the distances travelled by the left and right wheel, respectively, in time step i, measured *exactly*, i.e. noisy odometry should *not* be used. Set the time step length Δt to 0.01s. You may use the exact readings of the wheel speeds, as obtained from the `robot` variable in `BrainStep.m`.

Find values of the parameters a_0, a_1, a_2, b_0, and b_1 which will allow the robot to move as far as possible (i.e. to get as high fitness as possible) during a 30 s simulation, without colliding with the walls or the obstacle (collisions should lead to termination of the evaluation) in the arena shown in the right panel of Fig. 3. For full points, the robot should obtain a fitness value of at least 10.0.

You *may* use `ERSim` to evolve the best set of parameters, but it is also allowed simply to search manually (or by any other method of your choice) through the five-dimensional space spanned by the parameters. In your report, make sure to include a description of the procedure used for determining the parameter values (as well as the actual *values*, of course). Furthermore, make sure to hand in `CreateBrain.m`, `BrainStep.m`, and `TestRunRobot.m`, with the best values of the parameters a_0, a_1, a_2, b_0, and b_1 (These files will be copied into `ARSim`, for testing the behavioral organization system).

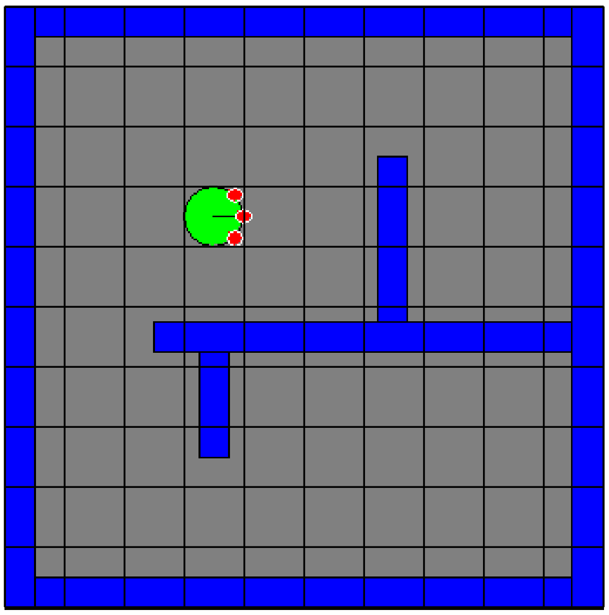

Figure 4: The robot and the arena, on which the occupancy grid is projected.

Problem 2.4, 3p, Exploration behavior

An important problem in robotics concerns mapping, i.e. making a robot generate a spatial representation of its surroundings. In order to be able to generate a map, the robot must efficiently explore its surroundings. Your main task in this problem is therefore to design a robotic brain that generates explorative behavior for the robot in the ARSim v1.1.8 package.

A common representation of maps in robotic mapping is the *occupancy grid* framework: A (virtual) grid of cells is projected onto the robot arena, where each cell is of a specific size. In this particular case, all grid cells are 0.4×0.4 m. Consider Fig. 4. Each cell of the grid is associated with a state, describing the status of the corresponding part of the arena, such as *free*, *unknown* or *occupied* (hence the name occupancy grid).

In this problem you should implement a robust (meaning that it should actively avoid collisions) exploration behavior for the simulated robot. That is, given an arbitrary starting point (x, y, ϕ), where ϕ denotes the orientation, the robot should visit as many free arena cells as possible in a specific amount of time, here set to 30 s (3,000 time steps) of simulated time. The actual position of the robot is taken as the position of its center of mass. In order to solve this problem you *should* use a special version of ARSim v.1.1.8, available for download at the following web address:

`www.am.chalmers.se/~wolff/AA/ARSimExamHP2.4.zip`

Except for the files `CreateBrain.m` and `BrainStep.m` (in which you should implement the robotic brain), you *must not* alter *any part* of *any file* in the ARSim package. The robot has three IR sensors, an odometer, and a compass. You are free to use any of these components in the robotic brain, but *no* settings should be altered in the specification of the robot and its sensors!

The explorative behavior may be hand-coded or evolved. However, the generated brain should be specified completely in the files `CreateBrain.m` and `BrainStep.m`, which you should submit along with your answers to the home exam. Furthermore, *no* pre-specified map of the arena may be used with the robot, *neither* should the robotic brain be supplied with any information about the robot's *true* position at any time! However, you may use the map created by the robot as it is being generated (using only the odometric readings to determine the robot's position) However, all grid cells *must* first be initialized to the *unknown* state every time the program is started.

Your score will be based on the number of cells visited by your robot, averaged over five runs. In each re-revaluation, the robot will be started at a (note!) *random* initial position, and with a random heading. When running your program for evaluation, the examiner will simply put your versions of the files `CreateBrain.m` and `BrainStep.m` in the ARSim directory, and type `TestRunRobot` at the MATLAB prompt. No manual editing of any file should be necassary in order to run your program!

In addition to the files `CreateBrain.m` and `BrainStep.m`, you should submit a (brief) description of the brain.