Bio-Inspired Artificial Intelligence

Laboratory exercise: Average landmark vector navigation

Steffen Wischmann and Thomas Schaffter

10.11.2009

1 Goal

The goal of this exercise is to systematically investigate a foraging strategy, that is inspired by insect navigation, with respect to its robustness to varying environmental and sensory conditions. You will apply your knowledge from the statistics lecture to a specific example that you learned about in the Biomimetic robots lecture. You will make technically sound conclusions about the robustness of this so called Average Landmark Navigation model [3, 2] (for how, for instance, ants perceive landmarks see [1]).

By the end of this exercise you should have learned something about:

- How to systematically test behavior under varying conditions:
  - position of the robot in the environment
  - level of sensory noise
  - number of landmarks
  - robustness to missing landmarks
- How to use Matlab for experimental data analysis:
  - linear regressions
  - correlation analysis
  - significance tests

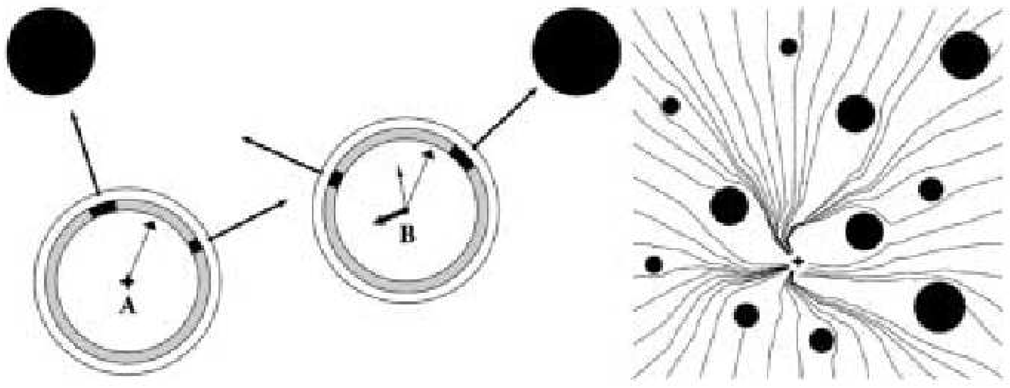

Figure 1: Average landmark navigation. Left: Homing mechanism of the ALV model. Landmarks are shown as black circles. The target position is marked with a cross (**A**). Each gray ring represents the horizontal portion of the landmark panorama as perceived from the position in the center of the ring. Vectors attached to the outer ring depict landmark vectors. **A**: The AL vector of the target location is computed from the average of the landmark vectors and stored in memory (vector in the center). **B**: The difference of the AL vectors of current location (thin vector, small head) and target location (thin vector, wide head) gives the home vector (thick vector). Left: Performance of the ALV model in a situation with 11 landmarks.

2 Theory

2.1 Average landmark navigation

The following description of the average landmark navigation model is an excerpt of from [3].

Figure 1 (left) visualizes the homing mechanism of the ALV model. A unit vector points from the position of the agent towards each detected landmark feature, in this case the center of black sectors in the horizontal view; these vectors are called “landmark vectors”. Their average - the AL vector - is an unambiguous signature for each location. The AL vector of the target location is stored. On the return journey, the agent follows a continuously updated home vector given by the difference between current and stored AL vectors. Note that the ALV model requires some kind of external reference to align the views or vectors to the same compass direction. The formal description of the ALV model presumes that the axes of the agent’s coordinate system are aligned with the corresponding axes of the world coordinate system. The positions of n landmark points in the plane are given by $x_i, i = 1, \cdots, n$. From each agent position x in the plane, a landmark vector $L_i(x)$ with unit length points towards landmark i:

$$L_i(x) = \frac{x_i - x}{||x_i - x||} \tag{1}$$

Visibility of all landmarks from all points is presumed in this description. The AL vector $A(x)$ of position x is expressed as the average of landmark vectors:

$$A(x) = \frac{1}{n} \sum_{i=1}^{n} L_i(x) \tag{2}$$

Given a target position x_0, a home vector field $H(x)$ can be computed by substracting the AL vector of the target location $A(x_0)$ from the AL vector field $A(x)$:

$$H(x) = A(x) - A(x_0) \tag{3}$$

In the homing process, the agent follows $H(x)$ to return to the target location x_0 (the constant c determines the speed):

$$\dot{x} = cH(x) \tag{4}$$

Despite its parsimony, the ALV model successfully copes with complex environments with a high number of landmarks that can even be partly covering each other. In the following experiments we will test this statement not only to landmark failures but also to sensory noise and different starting conditions.

3 Experiments

Download the file *091110_averageLandmarkNavigation_exercise.zip*. Unzip it, open Matlab and change into the according folder as your working directory.

3.1 Experiment 1: Influence of starting position

Open the file *exercise1.m*. Press the run button. What you can see in the figure (Figure 2):

- Green spots mark the landmarks
- The asterisk marks the target location

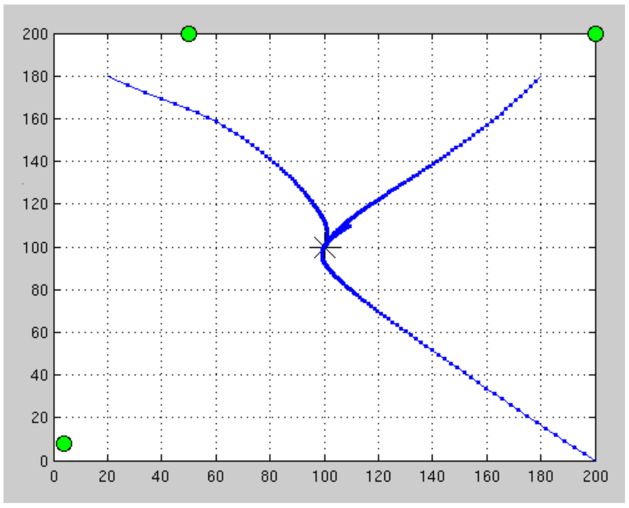

Figure 2: Your simulation environment. Landmarks are the big green dots. The robot trajectory to a given target location (black asterisk) from different starting positions is shown as blue dots and lines.

- The blue dots and lines mark the trajectory of the robot

You can put the robot in different starting locations by adding a position by following the syntax of:

```
r=addStartPosition(r, [x y]);
```

You can add as many starting positions as you want and wherever you want (note, that positions outside $[0; 200]$ are not displayed by default, you need to zoom out of the figure if you want to see them). Your task is now to investigate whether the final distance to the target depends on the initial distance of the robot to the target. For each starting position the initial distance and the final distance is written to the file *log_diffStartingPositions.txt*. Change the name of the file by changing the variable *fileName* if you want to try out different sets of starting positions. Note that the simulation will run faster if you set the variables *DRAW_SINGLESTEPS* and *DRAW_TRAJECTORIES* to 0.

- Create a set of initial starting conditions that will be suitable for further statistical analyzes.

After you evaluated several initial positions, open the file *plotExcersise1.m* and run it. The plot shows you the final distance in relation to the initial distance.

Now analyze the data as follows:

- Do a linear regression and plot the result together with the raw data (hint: use the matlab functions *polyfit* and *polyval*).
- Find out whether or not the two variables are statistically significantly correlated (hint: use the matlab function *corr*).
- Does your sample size influence the result? (try small and large sample sizes by varying the number of starting positions)

3.2 Experiment 2: Sensory noise

3.2.1 Noise while searching

Real sensors, either of insects or robots, are always noisy and don't give a precise value as we dealt with in the previous experiment. In the following we will apply different noise levels to our model.

Open the file *exercise2.m.* Press the run button. You see two trajectories of the robot searching for the target from a random starting position. In the first case the sensors are perfect, in the second case a gaussian noise with a standard deviation of 0.1 (i.e., $\approx 10\%$) is applied to each sensor reading (i.e., the vector to the landmarks). Rerun the experiment a few times and observe how the noise influences the robot's trajectory.

In the next steps we want to systematically investigate the influence of sensory noise with respect to the performance of the robot in finding the target.

- Change the *searchNoise* variable so that it contains different levels of noise (the noise should be kept between $[0.0; 1.0]$).
- Set *DRAW_SINGLESTEPS=0* and *DRAW_TRAJECTORIES=1*.
- Rerun the experiment and visually observe how different noise levels influence the trajectory of the robot.
- Increase the number of *NUMBER_OF_TRIES* to 3.
- The results of your runs are saved in the file *log_diffSearchNoise.txt* where the first column gives you the noise level and the subsequent columns tell you the final distance of the robot to the target for each try. You can change the *fileName* if you want to try different sets of experiments.

Now we want to apply some statistics:

- Open the file *plotExcersise2.m*. It will read the *log_diffSearchNoise.txt* file you just created and plot the following information:
  - The first plot shows you the result of each single experiment and a linear regression as you know it from Experiment 1.
  - The second plot visualizes your data in form of boxplots.
  - The third plot shows you the average performance and the standard deviation for each noise level.
- What can you tell from these plot about the difference between the different noise levels?
- Extend the script below the line

  ```
  %test for significant differences
  ```

  To include a significance tests for the different noise levels (hint: use the *ttest* matlab function). Note that your data has been transformed. Now all results for a particular noise level are in one column of *data*.
- How well do the p-values correspond with you visual observation of the box-plots?
- Change now the *NUMBER_OF_TRIES* in *exercise2.m* to somewhere between 3 and 20. Rerun the experiment and the analysis. Why is this important for statistical analysis?
- What else can you change to improve strengthen your statistical analysis (remember what you've just learned in Experiment 1)?

3.2.2 Noise while memorizing

We now want to investigate how sensory noise during the memorizing process (i.e., when the target vector is calculated) influences the performance of our robot. Open the file *exercise2b.m*. Press the run button. You see two trajectories of the robot searching for the target from a random starting position. In the first case the sensors are perfect, in the second case a gaussian noise with a standard deviation of 0.1 (i.e., $\approx 10\%$) is applied to each sensor reading but now only during the memorizing process (for simplicity we keep the noise during searching constant at 0.0). Rerun the experiment a few times and observe how the noise influences the robot's trajectory.

- change the *memoryNoise* variable so that it contains different levels of noise (the noise should be kept between $[0.0; 1.0]$
- Continue as in the previous experiment.
- For statistical analysis reuse the *plotExcersise2.m* script. Note, that the default filename for this experiment is *log_diffMemoryNoise.txt*.
- Extend the script to compare the results with the results of the previous experiment.
- What can you say about the difference between sensory noise during memorizing and sensory noise during searching.

3.3 Experiment 3: Number of landmarks

Now we want to investigate the influence of the number of available landmarks on the performance of the robot.

- Open the file *exercise3.m*.
- Choose an appropriate noise level for *searchNoise* and *memoryNoise* based on your experience from Experiment 2 (e.g., 0.05).
- Choose an appropriate number for *NUMBER_OF_TRIES* (e.g., 10)
- Now systematically vary the number of landmarks by using the function:

  ```
  w=addLandmark(w, [4 8], 0)
  ```

  this would set a landmarks at position $(4, 8)$. (keep the last argument at 0 for now). Keep the position of the landmarks within $[0; 200]$ otherwise you need to zoom out in the trajectory plot to see them.
- Each time you run the experiment with a different number of landmarks a line is added to the file *log_diffNoOfLandmarks.txt* giving the performance for this condition of each trial (if you want to reset everything, just delete this file and start anew).
- Open the file *plotExcersise3.m*. It will plot your data in a way you are used to from the previous experiment (note that each condition correspond to the experiment you conducted with different numbers of landmarks in sequential order).

- How does the number of landmarks influence performance?
- Vary now the level of sensory noise and investigate how the number of landmarks influence the results compared to what you found out in the previous experiment (note that you need move or delete the log file if you start a new set of experiments).

3.4 Experiment 4: Missing landmarks

Now we want to investigate the robustness of the behavior if landmarks are missing during the search behavior. This can happen, for instance, by occlusion or because landmarks are not within the visual field of the robot anymore.

- Open the file *exercise4.m*
- When adding landmarks by

  ```
  w=addLandmark(w, [4 8], 0)
  ```

  the last argument of this function determines whether or not the robot can see this landmark when it is searching for the target (set this value to 1 if you want this particular landmark to not be visible during searching, otherwise set it to 0).
- Landmarks that are not visible during search (but were visible during the memorizing process) appear red in the trajectory plot.
- Compare different set of landmarks with and without removal of one or more landmarks by using the script *plotExcersise3.m* which you know already from the previous experiment (note that you need to change the *fileName* to *log_diffSearchMissingLandmarks.txt*).
- What can you say about the robustness of the behavior with respect to the number of available landmarks and the number of missing landmarks?

Of course landmarks can also be missing while memorizing which are present while searching.

- Proceed as above, but now make landmarks invisible during memorizing by changing the last argument of the following function to 2

```
    w=addLandmark(w, [4 8], 2)
```

Landmarks that are missing during memorizing but present during searching are marked yellow in the trajectory plot.

- Change the *fileName* variables in the scripts appropriately if you test different sets of experiments but want to keep your old data.

You can now investigate different sets of combinations of landmarks missing during memorizing or/and during searching.

References

[1] Paul Graham and Ken Cheng. Ants use the panoramic skyline as a visual cue during navigation. *Current Biology*, 19(20):R935 – R937, 2009.

[2] Dimitrios Lambrinos, Ralf Möller, Thomas Labhart, Rolf Pfeifer, and Rüdiger Wehner. A mobile robot employing insect strategies for navigation. *Robotics and Autonomous Systems*, 30(1-2):39 – 64, 2000.

[3] Ralf Möller. Insect visual homing strategies in a robot with analog processing. *Biological Cybernetics*, 83(3):231–243, August 2000.