Self-Localisation Revisited

Joscha Bach and Michael Gollin

Humboldt University of Berlin, Department for Computer Science
{bach, gollin}@informatik.hu-berlin.de

Abstract. We discuss different approaches of self-localisation in the Simulation League. We found that the properties of the soccer server's quantization function tend to produce artifacts with the common approaches, which we try to deal with using a new method: We simulate the player's position, and dynamically correct this estimate with a gradient descent function by the minimal amount necessary to make it consistent with the perceived flag positions (where we allow for error margins according to the quantization function). It can be shown that self-localisation using the new method is not only relatively exact (between 6.6% and 35% smaller average error than the 'Nearest Flag' algorithm) but also yields much more consistency than the other approaches. ...

1 Introduction

The soccer agents in the simulation league of RoboCup [5] do not have the ability to perceive their environment (the field with players and ball) as a whole. Rather, the soccer server sends 'sense data' in regular intervals, relative to the agent's position and restricted to a certain viewing angle. Frequency of sense data and angle are inversely related: for the available viewing angles of 45, 90 and 180 degrees, data arrive at intervals of 75 ms, 150 ms and 300 ms, respectively [4]. (It is possible to receive data without distance information at a faster rate, but we do not take this into account here.) Whenever the agent gets no sense data during a simulation step, its position should be simulated with respect to its movement actions. To maintain a world model, the agent has to analyze this data, update the positions of 'seen' objects and perhaps simulate the positions of hidden objects. While there are no restrictions on the internal format of the data, most teams opted for an absolute representation in Cartesian coordinates. Self-localisation is crucial with this approach. Usually, the position is derived from the fixed objects in the field: there are 55 flags and four lines (the latter ones are helpful in determining the direction the agent is facing). Objects within the visible sector are determined by an angle relative to the agent's viewing direction (± 0.5 degrees) and the distance. The soccer server adds noise on the latter using a quantization function; closer objects are more accurately seen than distant ones.

Several ways of computing the position of the agent lend themselves to the task, for example:

A. Birk, S. Coradeschi, and S. Tadokoro (Eds.): RoboCup 2001, LNAI 2377, pp. 251–256, 2002.

- With known viewing direction, the position can be calculated from any seen flag. Since closer flags are more accurate, the nearest one should be used. This method (called "Nearest Flag") is both fast and easy to implement and is used for instance by the CMU-98 team. [6]
- Use a weighted sum of the positions derived from all visible flags. This method has been used by our team, AT Humboldt 98. [3]
- Triangulate from two or more visible flags. To reduce the computational complexity, not all flags should be combined. Rather, one might choose to look only at the two closest flags. It is also possible to use heuristics to determine reliable pairs of flags and weight the result. [2]
- The visible flags may be fed into a neural network that delivers the agent's position and viewing angle as output. We have tried this successfully, but we do not consider it an improvement over the algorithmic methods.
- Other learning methods, like Case Based Reasoning, can do the job as well. Again, while it works, the implementation overhead does not appear to be justified by the result. [7]
- It is possible to compute the maximum positional error for each seen flag using the quantization function, and therefore an area where the agent could be possibly located. By intersecting these areas, an 'error figure' can be determined. Whenever this intersected space is small enough, the position of the agent can be calculated with high accuracy. We use a simplified approach that makes use of this property.

2 Quantization and Consistency

To add noise on the sense data, the soccer server makes use of a special function, which we call Noise here (in the soccer server, this takes place in two steps):

$$\text{Noise}(d) = \frac{1}{10}\text{rint}\left(10e^{\text{rint}\left(\frac{\log(d+EPS)}{qstep}\right)qstep}\right) \tag{1}$$

where d is the distance of the object, *qstep* determines the quantization level (0.01 for fixed objects) and *EPS* is an offset of almost zero (to prevent undefined values from the log function, when d is zero. *Noise* returns the original distance with an added (quantization) noise component, with one decimal place.

The noise produced by this quantization function is very different from common kinds of noise (like, for instance, from sensor input). The values are discrete and not Gaussian distributed, that is, the likelyhood for a value to be close to the exact value is just as high as the likelyhood of it being located at the fringes of the error limit. *Noise* tends to produce artifacts, especially for distant, slowly moving objects, which seem to move in small, erratic jumps or even appear to change their direction temporarily (though within the error limit, of course). This may lead to difficulties, when localisation algorithms do not take the resolution of *Noise* into account; a weighting of values might not be enough.

The positional errors in subsequent self-localisations do not remain constant or, which is worse, add up surprisingly often. The quantization function actively

supports agents in beliefs about a jerking, wobbling world around them. This can render any attempt to determine the speed of objects impossible, if positional differences from one step to the next are used, which is often necessary. [1]

These pitfalls of a wobbling world model could be completely avoided by the introduction of a localized world model, where data is maintained relatively to the agent's position. But since the positions of fixed objects are known, and for reasons of joint planning and communication, a global world model relative to the field is immensely practical. The prospect of maintaining two world model components seemed not very elegant to us, and we dropped the idea. Instead, we thought of ways of increasing the accuracy of self-localisation and thereby reducing the possible cumulation of errors, and about steadying the algorithm.

3 The Gradient Descent Algorithm

The maximum positional error of each flag f_i is linearly dependent on its distance to the player and can be approximated as

$$\delta_{max} = 0.0045 d_i + 0.05 \tag{2}$$

where d_i is the seen distance of f_i. Thus, the visible distance determines a circular ring of possible positions around the flag. (If angles were taken into account, it would be a ring segment. However, we found it unnecessary to consider angles at all.) If several flags are seen, the intersections of their respective rings define an area of possible residence (the 'error figure'), such that for all points p within the area can be stated

$$\forall f_i : \left|\overrightarrow{p, f_i}\right| \leq \delta_{max} \tag{3}$$

where is the distance between p and f_i. Furthermore, we can guarantee that

$$\exists p : \forall f_i : \left|\overrightarrow{p, f_i}\right| \leq \delta_{max} \tag{4}$$

Theoretically, this area might be disconnected (imagine the case of two flags), but this almost never happens. Nevertheless, we will have to keep this in mind. The size of the error figure determines the maximum accuracy with that the agent's position can be determined from the available sense data. It might be a good idea to localize the agent in the middle of this figure (for instance, by finding the smallest δ to keep condition (4) satisfied and use the corresponding p as position of the agent, provided there is only one). Actually, this yields an average positional error of less than 5 cm at a visible angle of 180 degrees. However, for smaller visible sectors, the error figure tends to be much bigger and the result becomes inaccurate.

If position of the agent in the previous simulation step were known, we could estimate the current position more precisely than we could extract it from the sense data. But since errors from simulated movements accumulate, we have to integrate sense data into the calculation. Ideally, what we want is a p that satisfies [4] and is as close as possible to our estimate. (This constraint helps

also in the case of a disconnected or cavernous error figure.) Thus, quantization noise will only affect the resolution, but not the consistency of the localisation.

A suitable p might be found by generating a 'force field' that pulls our estimate towards the error figure. Imagine, all the flags were pulling or pushing the estimated position in the direction of their respective error limit. Note that there is no 'force' whenever the estimate is located within the error limits of the flag. Whenever there is a resultant force different from zero, there is a point in the direction of this force that is closer to the desired p than the current estimate. Of course, we can not simply add the individual forces to get the length of the vector towards this point. We have to make sure that three flags pulling into one direction do not produce three times the necessary force! Instead, we may use the according directional component of the strongest force pulling into the desired direction. To produce faster convergence, it is also acceptable to move in the direction and by the amount of the strongest individual force. This leads to the following simple algorithm:

1. Estimate the current position from previous position and movement actions. If previous positions are not available, use any other self-localisation method.
2. For all visible flags, compute the error limit using (2).
3. For the current estimate, calculate the distances to all visible flags (the real flag positions are known). Calculate the 'force vectors', which are given by the difference between seen distance and calculated distance. The error limit has to be subtracted from each; only forces over this threshold are considered.
4. If there is no force left, or the maximum number of iterations is reached, return the current estimate.
5. Otherwise, move the estimate by the longest force vector, continue with 3.

Since the algorithm sometimes shows asymptotic behaviour, the number of iterations must be limited (for instance, to 10 or 20). One could also increase the respective forces by a small amount to speed up the convergence and escape the asymptote. There is little danger in using only a small number of iterations, because the process will continue in the next simulation step at the position of the current estimate.

4 Results

When comparing the gradient descent algorithm with other methods, we have to bear in mind that data does not arrive in every simulation step (except when the visible angle is restricted to 45 degrees), so all methods have to use a certain amount of simulation. If sense data was to arrive in every simulation step at a visible angle of 180 degrees, the accuracy of our method would vastly increase.

We have tested the algorithms under what we hope are realistic conditions by repeatedly randomly placing an agent on the field and performing straight dashes until the border of the field is reached.

4.1 Accuracy

To determine the accuracy of the algorithm, we compare it to the 'nearest flag' method (CMU-98), the previous AT Humboldt algorithm and a heuristic triangulation algorithm that has been implemented as part of a Diploma Thesis at our department. [2]

Table 1. Average positional errors (in m), standard deviations and maximum errors at different visibility angles (10^7 samples)

	Algorithm:	Gradient	Nearest Flag	ATH 98	Triangulation
180 degrees	Av. Error	0.086	0.123	0.139	0.104
	Std. Dev.	0.101	0.144	0.162	0.122
	Maximum	0.569	0.744	0.743	2.179
90 degrees	Av. Error	0.083	0.127	0.146	0.130
	Std. Dev.	0.097	0.155	0.179	0.177
	Maximum	0.711	0.852	0.885	3.643
45 degrees	Av. Error	0.142	0.152	0.155	0.272
	Std. Dev.	0.182	0.192	0.195	0.353
	Maximum	2.218	1.247	1.027	5.173

At a visible angle of 180 degrees and a sense data frequency of 300 ms, our gradient descent algorithm is 30% better than the 'nearest flag' method. We found maximum values not very informative, since they often occur only rarely (for the triangulation method, only two out of one million values were in the 2 m range). We list the standard deviations to give an idea of the distribution.

At a visible angle of 90 degrees, data arrive in 2 out of 3 simulation steps. The average error is 35% better than for 'nearest flag'. At a visible angle of 45 degrees (sense data in every simulation step), few flags are visible at a time. Because our prototypical implementation does not check whether the number of visible flags allows reliable self-localisation with the gradient descent, the maximum positional error grows more than proportionally.

At an average error of 14.2 cm, the gradient descent algorithm is still slightly more (6.6%) accurate than the 'nearest flag' algorithm.

4.2 Consistency

As mentioned before, we consider consistency (i.e. only small differences between subsequent simulation steps between simulation and perception) as even more important than accuracy. The gradient descent algorithm yields indeed better consistency than the competitors.

While the other tested algorithms tend to produce bigger positional differences with growing unreliability of the input values, the gradient descent algorithm is relatively robust. Consistency appears to be almost independent from

Table 2. Average positional differences between simulation steps in m, standard deviations and maximum differences, at different visibility angles (10^7 samples)

	Algorithm:	Gradient	Nearest Flag	ATH 98	Triangulation
180 degrees	Pos. Diff.	0.050	0.064	0.056	0.061
	Std. Dev.	0.099	0.125	0.106	0.119
	Maximum	0.636	0.980	0.620	2.246
90 degrees	Pos. Diff.	0.058	0.113	0.179	0.130
	Std. Dev.	0.089	0.160	0.117	0.207
	Maximum	0.629	0.962	0.761	5.546
45 degrees	Pos. Diff.	0.051	0.189	0.131	0.322
	Std. Dev.	0.079	0.226	0.148	0.460
	Maximum	1.300	1.307	0.961	7.384

the number of visible fixed objects, and the values for the average positional differences are between 22% (at 180 degrees) and 73% (at 45 degrees) better than for 'nearest flag'.

References

1. Badjonski, M., Schröter, K., Wendler, J., Burkhard, H.-D.: Learning of Kick in Artificial Soccer. Pre-Proceedings of the fourth RoboCup Workshop, Melbourne (2000)
2. Meinert, T., Sander, G.: Der Intention-Agent. Diplomarbeit, Humboldt-Universität zu Berlin (2001)
3. Müller-Gugenberger, P., Wendler, J.: AT Humboldt 98: Design, Implementierung und Evaluierung eines Multiagentensystems fr den RoboCup-98 mittels einer BDI-Architektur. Diplomarbeit, Humboldt-Universität zu Berlin (1998)
4. Noda, I: Web Site: RoboCup Soccer Server. http://ci.etl.go.jp/~noda/soccer/server/ (March 2001)
5. Website: RoboCup Federation. http://www.robocup.org (March 2001)
6. Stone, P.: Layered Learning in Multi-Agent Systems. PhD Thesis, Carnegie Mellon University (1998)
7. Wendler, J., Brüggert, S., Burkhard, H.-D., Myritz, H.: Fault-tolerant self localization by case-based reasoning. In T. Balch, P. Stone, G. Kraetzschmar (eds.): Proceedings of the fourth RoboCup Workshop Melbourne (2000) 82–89