

FAUBOT: Purposeful Navigation of a Robot in a Simulated Environment

Daniel Danner, Caroline Kaufhold, Peter Kranz, Rainer Müller, Sven Pfaller,
Christian Rieß, Elli Angelopoulou

Department of Computer Science, University of Erlangen-Nuremberg, Germany.
{sidadann, sicakauf, sipekran, sirrmuell, sisvpfal}@stud.informatik.uni-erlangen.de,
{riess, elli}@i5.informatik.uni-erlangen.de

Abstract—In the CiberMouse Contest a robot has to find in a simulated environment its way to a beacon and back to its initial position using data from different sensors. This paper presents the design of the FAUBOT and the challenges the team met when creating the robot. Particular algorithms and datastructures that are used include Brook’s subsumption architecture, the A* algorithm and a probability map of the world.

I. INTRODUCTION

The CiberMouse simulation environment is a platform to experiment with virtual robots and is for example used in the annual Ciber-Rato Contest [1]. Since the setup is bound to short simulation cycles, special attention has to be paid to real-time performance. The CiberMouse organization provides the technical framework to start right away with the development of algorithms. In general, the task is to direct a mouse through a maze to find a beacon. The mouse is modeled by a virtual robot that has sensors and actors. For orientation, the robot has four free-to-place obstacle sensors (the “eyes”) and a beacon sensor (the “nose”). For navigation, it has two individual controllable wheels. Also, for test and evaluation purposes, there is a GPS tracker installed to externally monitor the actual position. The task is not only to find the beacon but also to find the way back to the initial position (in the following referred to as “home”) and to be faster than the two other competing mice. Therefore, three tasks have to be solved:

- 1) The sensor data has to be handled properly and some movement strategy has to be developed to build a reliable map.

- 2) Once a (partial) map is available, a sophisticated pathfinding algorithm needs to be employed to find the way to the beacon and back.
- 3) The robot must not collide with walls or other competing robots.

This paper is structured as follows: First, we give an overview of the used algorithms. In section III the framework and our robot system architecture is described in detail. Several other aspects that came up during the development of the robot are addressed in section IV, as well as the overall system performance. We conclude with a short summary and an outlook.

II. SYSTEM OVERVIEW

The FAUBOT needs to explore the environment randomly as long as there is no available information about the beacon’s position. Until the beacon is found, the robot is continuously exploring the world. During these operations, a probabilistic map is built up based on the perceived sensor data. It contains not only supposed obstacles but also visited areas which will be assumed “safe for driving” later on. As soon as the beacon is detected, we rely on naive driving to the measured direction while obstacle avoidance is handled using a non-deterministic algorithm.

When the beacon has been reached, the FAUBOT uses a modified A* algorithm on the probabilistic map in order to compute the optimal known path home through the maze. As long as the robot moves along areas that are marked *visited*, we know that appearing obstacles are other robots. The avoidance

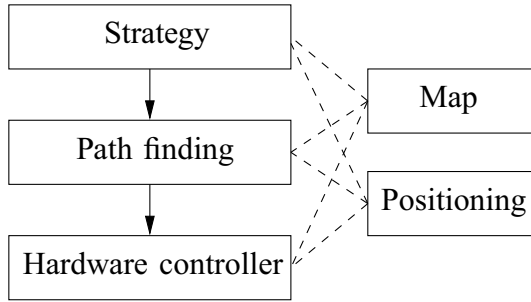


Fig. 1. System architecture: The components on the left form a hierarchical decision module. Map and positioning component provide the fundamental data for those decisions.

routine is a combination of the wall avoidance in the exploring phase and the A* algorithm.

III. SYSTEM ARCHITECTURE

The system architecture is split into three major parts, “strategy”, “path finding” and “hardware controller”, as shown in fig. 1. They build a vertical hierarchy of different abstraction levels over which sensor data and commands are passed upwards and downwards.

Sensor data is updated in discrete time units called “cycles”. The robot can read two sensors in every cycle, which demands a trade-off between the demand of different sensor data. After reading the sensors of the current cycle the components are updated with the new information, and start to perform the higher level computations. The first component to be considered is the strategy which issues commands at the highest level of abstraction. On the second level the path finding component executes the strategy’s commands by autonomously performing obstacle avoidance. The lowest level provides a basic sensor scheduling and the evaluation of the sensor data. Additionally, higher level components can always override decisions of the lower level components. This organization of tasks is similar to Brooks’ subsumption architecture [2].

Besides those three layers there are a “map” and a “positioning” component, which provide a representation of the world and the robot in it. Those are both independent of the operating hierarchy and compute the information based on the data we get from the sensors.

In the following we give a description of the different layers, their domains and interactions.

A. Strategy

The strategy component internally implements a finite state machine. It basically consists of three states that describe our main goals. At start-up the FAUBOT is not aware of the beacon position, so it has to *explore*, which is the first state. The strategy component selects waypoints where the bot should drive to in order to find the beacon. When the robot spots the target it switches to the next state, *drive to beacon*. The strategy decides between two alternatives:

- driving on a straight line towards the beacon. Note that once the strategy decided to head directly towards the beacon, the responsibility for driving along the line and to the assumed target position relies on the path finding component, including the handling of walls and obstacles.
- gathering more precise data about the beacon position. By collecting multiple measurements of the beacon sensor that are represented as lines between different robot positions and the beacon, we can compute where those lines intersect. The strategy then assumes that the beacon is located in the area where the intersections accumulate.

In order to get more quickly to the target, mostly the first alternative is chosen.

Once the beacon is reached, the strategy switches to the *return home* state. Based on the data collected so far, the strategy instructs the path finder component to drive to the start position, where internally an A* algorithm is used.

The whole state machine is modularly designed so states can easily be added or removed. For example, additional states can be implemented like a higher level strategy for the avoidance of other robots.

B. Path finding

The path finding component is comprised of the following three methods:

- **drive to point:** The path finding algorithm heads directly or in small slopes for a given (x, y) -coordinate. When an obstacle occurs it uses a modified *Wall Follower* algorithm by randomly avoiding it to the right or to the left. This method is used during the exploration phase, and when the assumed beacon position

could be sufficiently narrowed down by examination of multiple signals from the beacon.

- **drive to beacon:** As long as there were too few beacon signals received for narrowing down the beacons position, only an approximate direction towards the beacon is known. The algorithm leads then directly along the bearing. The obstacle avoidance is similarly done as in the first case. One addition is to look around for further beacon signals once the robot left the direct trail towards the beacon.
- **drive home:** The last method is returning to our initial starting position. As there is already at least one known complete path from the home to the beacon from the exploration phase, we use a modified A* algorithm [3] on the collected data to compute a way back. The algorithm does not use the full resolution of the internal map, but operates on median filtered cells that are larger than our internal representation. This ensures that the computation of the A* algorithm finishes within a few cycles, and brings some additional security distance to the walls, since the lower resolution slightly increases known obstacles.

The strategy component can at any time override the behaviour of the path finding component and issue modified or different commands.

C. Hardware controller

The hardware layer provides a reliable abstract sensor and actuator API for the higher level methods. This comprises

- **engine actuators:** It is only possible to set a certain amount of power to the engines, thus it is necessary to estimate the actual speed. The FAUBOT does this by applying the acceleration formula given in the rule set to the difference in the power supply between the last time step and the current one. In the development phase this formula was checked by comparing the current position plus the speed estimates against the GPS coordinates, which yielded sufficient results.
- **beacon sensor:** The beacon sensor has a significant latency. As soon as the beacon is detected this event is applied to a state in the past and stored. The last ten beacon sightings are

used to determine the area where the beacon is assumed to be located.

- **obstacle sensor:** The FAUBOT uses the standard distribution of the obstacle sensors around the robot, where the rear sensor is not used. When an obstacle is detected, an entry into the map is made based on our assumed position. In order to avoid collisions the front sensor is higher weighted than the side sensors.
- **compass sensor:** Similarly to the beacon the compass has a high latency. Thus the positioning using the compass can be used on data in the past, since the FAUBOT incorporates the compass data online, and does not stop and wait for new compass data to arrive. This causes a reevaluation of all positioning results in the past nine cycles, and therefore also a reevaluation of almost all other recent results.
- **ground sensor:** If the FAUBOT is driving directly towards the beacon, the ground sensor is activated and regularly queried.

Most work of the sensor data evaluation is about integrating the obstacle and compass sensor together with the engine actuators into a common, accurate map, thus the hardware controller is closely connected to the map and the positioning components.

D. Positioning component

A reliable estimation of the robot's position in the world is essential for all decisions. This task is implemented in the positioning component. Its computation is based on assumptions of the motor behaviour taken from the rule set. In order to minimize our calculation error the compass is used to adjust our angle and therefore our position. The high latency of the compass requires us to apply these corrections to data that were collected several cycles earlier.

E. Map

The internal representation of the world is built up in the map component. At first it gets the raw data from the sensors that were queried in the current cycle. These are preprocessed and added to the map of the world included into this component. In order to provide as reliable as possible information for the modules that control the actions of the FAUBOT, noise has to be taken under consideration.

This is done by normalizing the received sensor data using a Gaussian distribution. The area that is covered by a specific sensor is updated with certain probabilities for containing an obstacle or not. Additionally the map marks visited sites in the map as *free*. Whenever an obstacle shows up in free areas, it can not be a wall, but one of the other robots.

IV. EXPERIENCES

A. Visual debugging

One of the first things we built up during development was a graphical interface for allowing easy debugging of the various low-level components. It turned out to be very useful in comparing the assumed position with the GPS coordinates, visually verifying the probabilistic map against the actual maze, and for displaying the generated log messages in an integrated environment. We also built in a mode for manually controlling the FAUBOT through the map in order to evaluate the map's robustness with respect to drawing walls and the effect of passing competing robots. In fig. 2 an example of our graphical representation is given. The image on the bottom shows the given simulation environment. The image on the top shows a visualization of the generated probability map, where darker grey levels correspond to higher obstacle probabilities. In this specific case the robot obviously did not manage to get past the second barriers.

B. Development problems

We tried to use a similar approach for the computation of the beacon's position as we did for the obstacle detection. This required entering for every direction of the beacon's signal a 4-degree circle sector into the map within the maze's boundaries. Unfortunately we did not succeed to perform this computationally efficiently, thus we dropped that approach.

Our geometric solution of representing the detection of a beacon as a line between the robot and the beacon position seems to produce also acceptable results, though we believe the method described above leaves some room for improvements.

V. CONCLUSION

Looking back, the modular approach has been successful as the behaviour can be easily changed

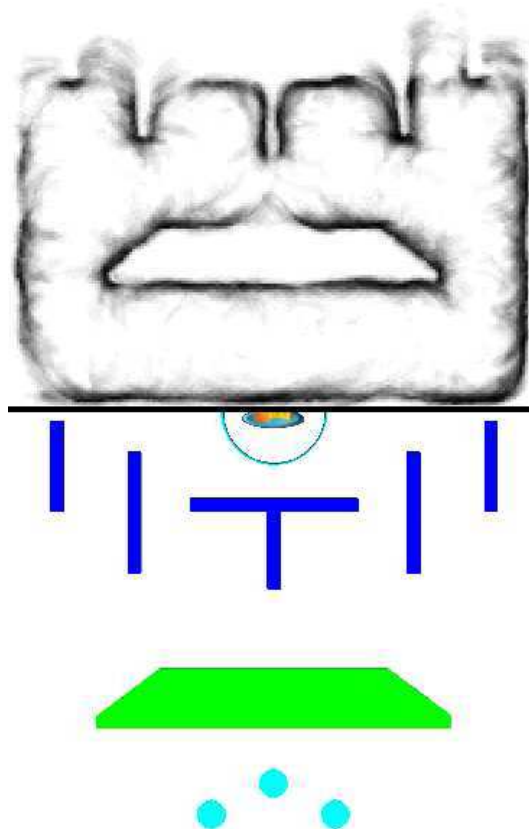


Fig. 2. Comparison between a given maze and the robot's representation of the world being partly explored. In this case the beacon was not yet considered.

in individual components. Additionally the graphical environment proved to benefit the development process, especially since the robot simulation must be written in comparably short time.

Currently our robot is fulfilling the basic requirements for the contest like finding the beacon and returning back home. We are sure several optimizations of our strategy or algorithms are possible and we will try to integrate them until the event. We are eager to test the performance of the FAUBOT in the CiberMouse competition at RTSS 2007.

REFERENCES

- [1] N. Lau, A. Pereira, A. Melo, and A. Neves e João Figueiredo, "Ciber-Rato: Um Ambiente de Simulação de Robots Móveis e Autónomos," in *Revista do DETUA*, vol. 3, no. 7, 2002, pp. 647–650.
- [2] R. A. Brooks, "A Robust Layered Control System for a Mobile Robot," in *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, March 1986, pp. 14–23.
- [3] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, July 1968, pp. 100–107.