Grid-Based Navigation for Autonomous, Mobile Robots

Carsten BUSCHMANN, Florian MÜLLER and Stefan FISCHER

Institute of Operating Systems and Networks, Technical University of Braunschweig
Braunschweig, Germany, e-mail: (buschmann, fmueller, fischer)@ibr.cs.tu-bs.de

Abstract - **Navigation is a major challenge for autonomous, mobile robots. The problem can basically be divided into positioning and path planning. In this paper we present an approach which we call grid-based navigation. Though we also propose a scheme for path finding, we focus on positioning. Our approach uses minimal environmental infrastructure and only two light sensors on the mobile device itself. Starting out from a predefined tile and orientation in the grid, the mobile robot can autonomously head for destination tiles in the grid. On its way it determines the current location in the grid using a finite state machine by picking up line-crossing events with its sensors.**

1 Introduction

A key ability needed by an autonomous, mobile robot is the possibility to navigate through the space. The problem can basically be decomposed into positioning and path planning. Though we also propose a scheme for the latter, we clearly focus on the detection of the current position by monitoring grid crossing events by using two light sensors.
Especially if the robot is severely resource-constrained, simple schemes are favourable to elaborated algorithms. Rather simple sensors and actuators as well as a limited computing platform also demand simple, robust techniques due to inaccuracy and the lack of resources.
The paper is structured as follows: In section 2, we give an overview over the environment that the robot which is described in the following section needs for navigation. In section 4 we present the underlying algorithm and discuss various details. After that the software structure of the implementation is presented. The paper is concluded by a summary and an outlook to future work.

2 Environment

The robot navigates on a grid (see Figure 1) which regularly divides the ground into square tiles that are identified with Cartesian coordinates. As mentioned earlier, the robot starts out from a predefined position (e.g. the centre of tile 0,0) and orientation (e.g. North, which means looking up the y-axis). It is sufficient that the full grid is rectangular and all tiles are the same size. It is not required that the robot knows the dimension (n,m) of the grid. It is satisfactory to only request navigation to tiles that really exist. Heading from tile to tile, the robot distinguishes eight driving directions: north, north-east, east, south-east, south and so on. Depending on the size of the robot and the sensitivity of the

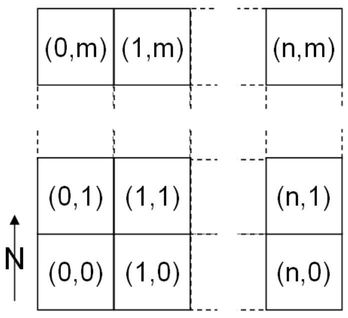


Figure 1: The grid

luminance sensors, different tile sizes and contrasts between plane and grid are possible. We used a grid made out of black self-adhesive tape of about 2 cm width on white ground with tiles of about 45 by 45 cm.

3 The Robot

The robot requires only very basic sensing and actuation capabilities. All that is needed are two ground-observing light sensors attached in a line orthogonal to the driving direction at the bottom of the robot, and two individually controllable wheels. For precise positioning (see Figure 3) the light sensors (A) are attached between the wheels (B) to make sure the sensor readings represent the wheels' location. As shown in section 4, the distance between wheels and sensors should be as small as possible to facilitate the alignment when crossing the grid orthogonally. At the back, the robots rests on a ball (C) that is spherical seated, allowing the robot to turn on the spot. The distance between the two light sensors should be no smaller than double the grid line width. This ensures that orthogonal grid line crossings where both light sensors get dark simultaneously (see Figure 2a) can be securely distinguished from diagonal crossings at the intersection of grid lines (see Figure 2b). If the sensors are located too close to each other, misinterpretations may appear (see Figure 2c).

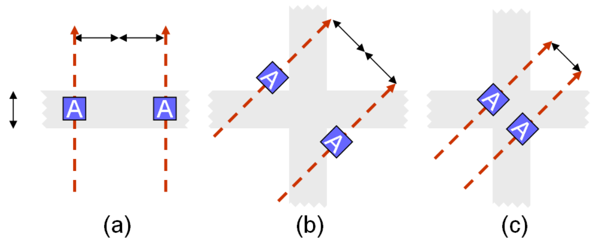


Figure 2: Consequences of different distances between the light sensors

Basically only binary sensor readings are required; in order to cope with a wider spectrum of readings, an initial calibration on light and dark ground can be used to determine a threshold between the two by calculating the average of both readings. Figure 3 to Figure 5 show the robot we used for experimenting built out of the LEGO Mindstorms Robotics Invention System 2.0 [1].
It does not only offer a cheap possibility to experiment with various robots, it also allows for focusing our work on algorithmic aspects without putting much effort into mechanical construction. The entire navigation and positioning code executes on the yellow RCX control unit (D in Figure 5). The PDA shown on the robot in Figure 4 serves solely as a communication gateway to a Wireless LAN Network [2].
Nevertheless, the Robotics Invention Systems also yields some disadvantages. The RCX is extremely resource constrained. It offers only 32kB of memory firmware, program code and data. In addition, engines and sensors are rather imprecise, e.g. the driving speed of the robot heavily varies with the remaining battery capacity. All this demands simple, robust algorithms for navigation.

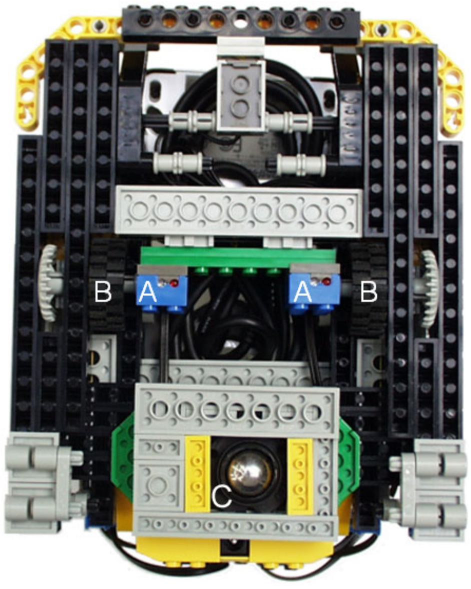


Figure 3: Bottom view of the robot with light sensors (A), wheels (B) and ball (C)

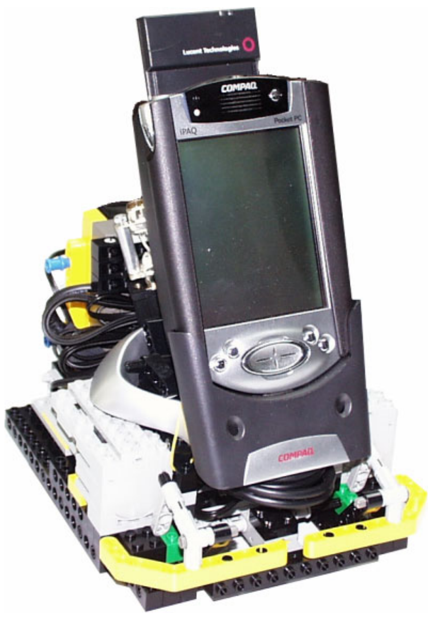


Figure 4: Robot with PDA for WLAN communication

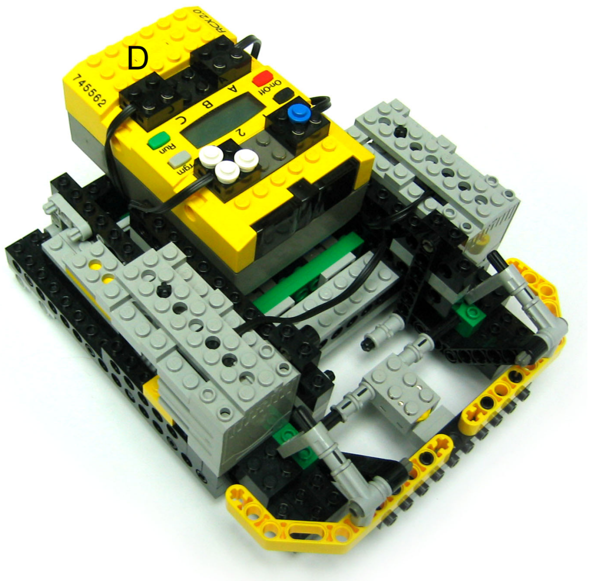


Figure 5: Top view of the robot including the yellow RCX control unit (D)

4 Algorithm

The process of driving to a destination tile always follows the same 5-step-algorithm: (1) calculate the next tile to go to, (2) turn towards the direction of the next tile, (3) start driving and wait for line-crossing events to happen, (4) thereby decide which tile the robot arrived in, and (5) finally determine whether the robot has reached the destination tile. The 5-step-algorithm is also visualized in Figure 6.
Determining the next tile is done via a breadth-first search: starting from the current tile, the coordinates of all adjacent tiles that have not been visited on the way to the current destination are

stored into a vector together with information on its predecessor. If the destination tile is not among the vector entries, again each entry's all adjacent and not yet considered tiles are stored together with their predecessor. The algorithm terminates when the destination tile is in one of the vectors. The path to that destination tile can then be derived using the chain of predecessors stored together with the coordinates. This rather naive approach has potential to be optimized. An example would be to employ branch-and-bound-techniques. Nevertheless, in large grids the applicability of such tree-based techniques is limited. In this case greedy algorithms might be an alternative. However, such path finding issues were not the focus of our research.

Once a path has been derived, the robot turns towards the direction of the next tile (which can be calculated from the current and next coordinates) by turning both wheels in different directions for a predefined time corresponding to 45 degrees (repeatedly if necessary). Because wheels and sensors are attached in a line, the light sensors will not "move" but only rotate.

The robot then starts driving, waiting for events to occur. We distinguish light sensor events and timeout events. A series of sensor events corresponding to the crossing of the grid) is always concluded by a timeout event. If no sensor events have occurred for a certain time period, it is assumed that the robot has reached the next tile. The length of the interval depends on the size of the tiles, the robot dimensions,

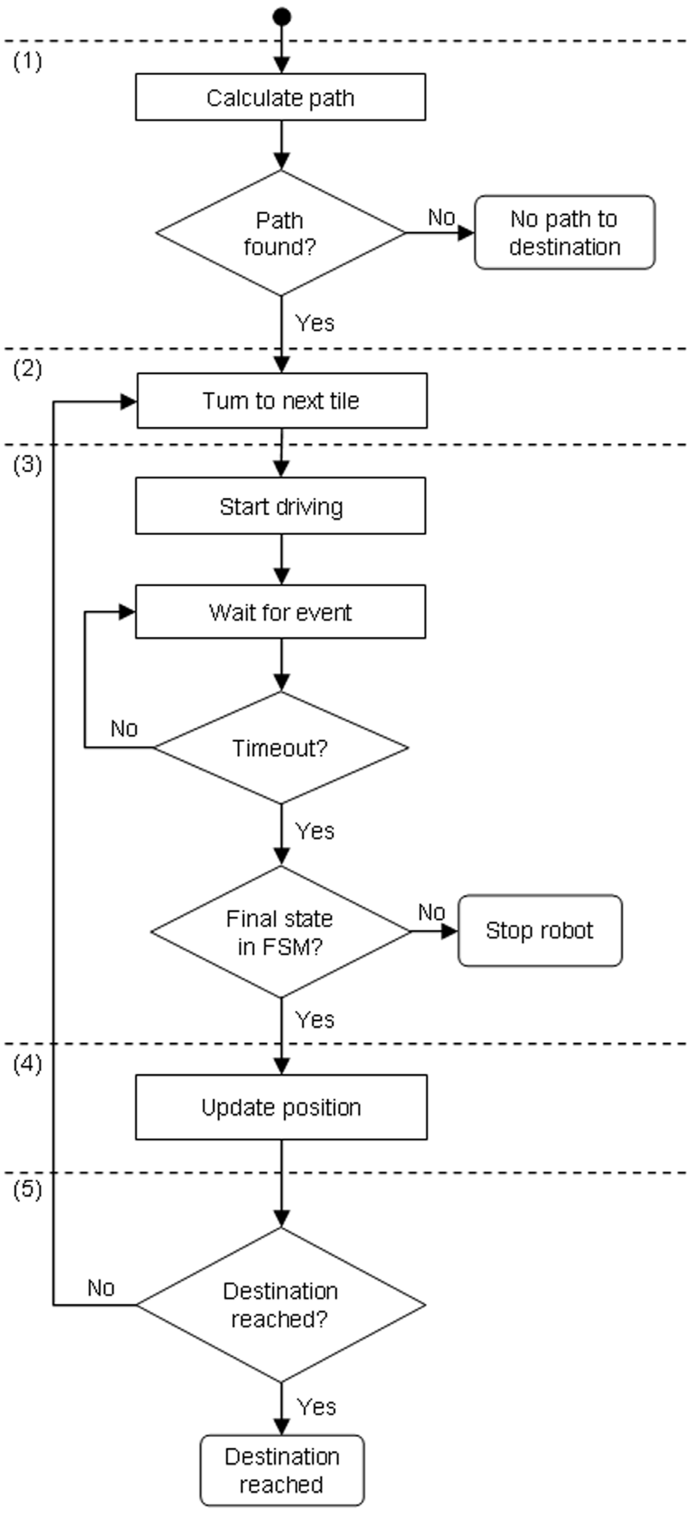


Figure 6: The 5-step-algorithm

path deviation and speed. If it is chosen too long (and tiles are small), the robot might already have started to cross the next lines at the opposite side of the new tile. If it is chosen too short, the timeout might occur during the crossing of the grid. For our implementation, we chose a timeout interval of four seconds.

Number, kind and order of sensor events determine which tile the robot drove to. Due to path deviations due to wheel slip, different engine powers etc. the robot does not necessarily arrive at the tile it was actually heading for.

Basically two ways of crossing the grid can be distinguished: the change to a tile in horizontal (East, West) or vertical (North, South) direction which we will call orthogonal crossing and the change to a tile at north-east, north-west south-east or south-west which we will call diagonal crossing for the rest of this paper.

Determining orthogonal crossings is relatively easy: after both sensors have gone dark and light again once and nothing happened afterwards for a while, the robot has reached the next tile. From the order in which the two events occurred a first indication can be drawn on the robot's deviation from the wanted direction (orthogonal to the grid line).

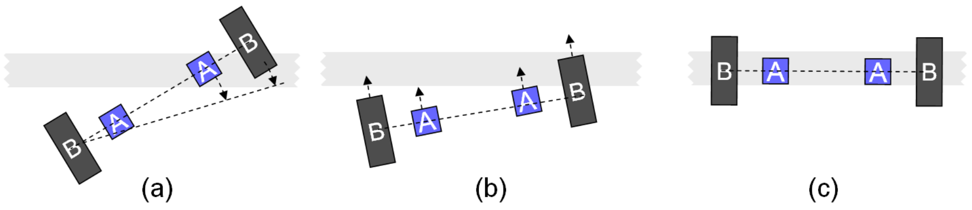


Figure 7: Robot alignment at orthogonal crossings

We use this fact to adjust the driving direction when crossing the line (see Figure 7). When the robot is heading for an orthogonal line crossing and the first light sensor event occurs, we stop the robot. Let us assume that it was the right sensor that got dark first (a). We then turn the corresponding (right) wheel back to make the robot turn until the according light sensor gets off the line again. The robot then again approaches the line until one light sensor recognizes it (b). This procedure is repeated until both light sensors are on the line simultaneously. By doing so the robot almost perfectly aligns with the grid line (c). Please note that this procedure is only invoked if the robot intentionally performs an orthogonal crossing.

When heading for a tile in diagonal direction (e.g. north-east), things are more complicated because five different cases can occur (see Figure 8). Depending on the position in the current tile, driving diagonally can lead to arriving in one out of three tiles that can be distinguished by the order in which the events occur. Please note that the upper left arrow and the lower right arrow denote special cases of an orthogonal crossing: though intending to cross horizontally, the robot effectively performs an orthogonal crossing. Unfortunately, this can only be detected after the robot arrived in tile 1 or 3 respectively.

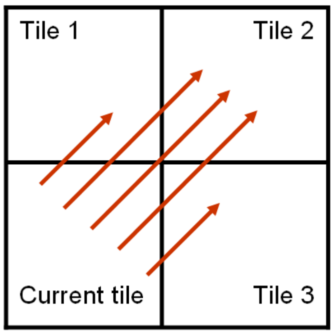
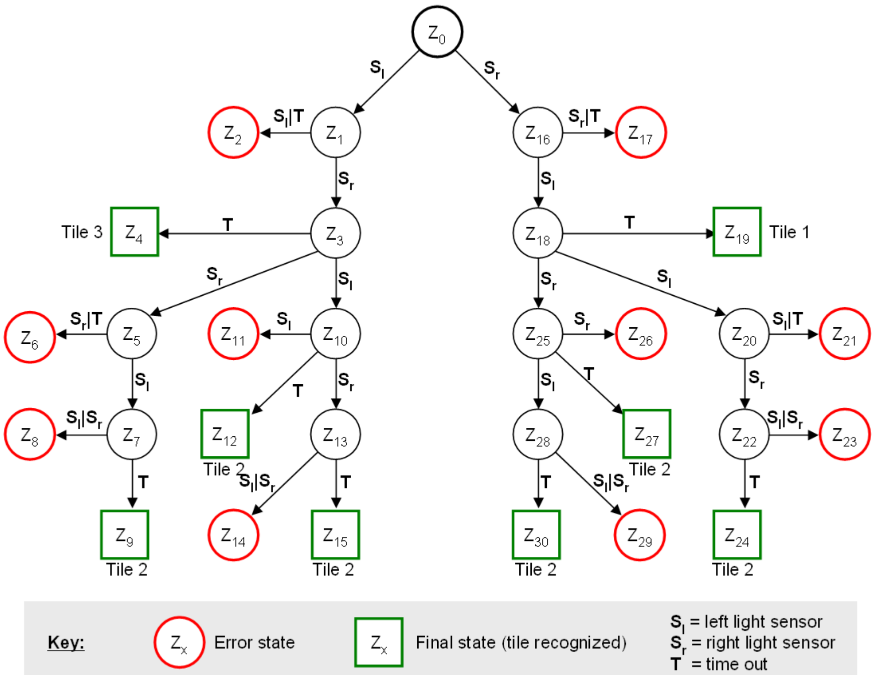


Figure 8: Possible cases when crossing diagonally


Z0
Sl
Sr
Z1
Z2
Sl|T
Z16
Z17
Sr|T
Z3
Z4
Tile 3
T
Z18
Z19
Tile 1
Z5
Z6
Sr|T
Z7
Z8
Sl|Sr
Z9
Tile 2
Z10
Z11
Z12
Tile 2
Z13
Z14
Z15
Tile 2
Z25
Z26
Z27
Tile 2
Z28
Z29
Z30
Tile 2
Z20
Z21
Sl|T
Z22
Z23
Sl|Sr
Z24
Tile 2
Key:
Zx Error state
Zx Final state (tile recognized)
Sl = left light sensor
Sr = right light sensor
T = time out