Navigation of a Kinematically Restricted Wheelchair by the Parti–Game Algorithm

J. Kollmann, A. Lankenau, A. Bühlmeier, B. Krieg–Brückner, T. Röfer
Bremer Institut für sichere Systeme, Technologie Zentrum Informatik, FB 3 Informatik
Universität Bremen
Postfach 330440, D–28334 Bremen
{joergk, alone, andreas, bkb, roefer}@informatik.uni–bremen.de

7/4/97

Abstract

This paper describes a technique of adaptive state–space partitioning for mobile robot navigation. Following this approach, the controller of a wheelchair robot adaptively discretises its state–space and learns which partitions are connected and which are not.

Due to the wheelchair's kinematic restrictions, the occuring problems differ from those of holonomic research robots. Several successful experiments were performed one of which is presented in full details.

1 Introduction

Up to now, a variety of approaches to robot navigation have been developed. State of the art are autonomous robots that explore the environment, building a metrical map or some kind of topological map. This map can then be used for path planning.

Strict separation between map-building and path planning is dangerous: a representation of space can be chosen that is not adapted to the specific environment and the robot's limited manoeuvring capabilities. Typically, parts of an environment are poorly structured and do not have to be represented in detail.

Proc. AISB workshop on "Spatial Reasoning in Mobile Robots and Animals", Manchester 1997.
Technical Report Series, Department of Computer Science, Manchester University, ISSN 1361 - 6161.
Report number UMCS-97-4-1. http://www.cs.man.ac.uk/csonly/cstechrep/titles97.html

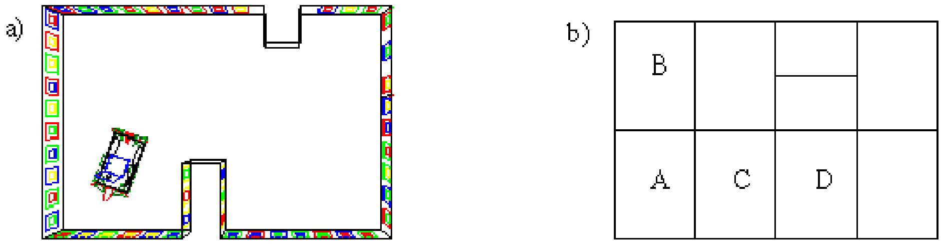


Figure 1: Illustration of the partitioning of the parti–game algorithm. a) Visualisation of the real world scene by SimRobot [4]. b) The corresponding state–space as learned by parti–game after three solved tasks.

Other sections pose bigger problems and should result in a higher resolution of spatial knowledge. This paper demonstrates how such an adaptive representation can be learned by the robot.

The goal of the work presented here is both, to develop new navigation techniques *and* to build an experimental wheelchair that is able to navigate autonomously. Results as described below provide the basis for future products that can ease the daily life of many handicapped or elder people. The parti–game algorithm [3] has been used to solve various tasks, including the navigation of a mobile robot and the control of a multi–joint robot arm [1]. However, as far as we know, these implementations have only been tested in simulations.

2 The Algorithm

The parti–game algorithm was developed for path planning in multidimensional state–spaces. Planning in the state–space avoids the problem of exactly locating the obstacles in the real world, i.e. neither a metrical map nor a CAD representation is required. Moreover, employing the state–space offers the advantage that for an execution of movements no additional transformation between the real world and the state–space is needed.

The only knowledge that the robot has to acquire is to learn whether it is possible to transform its state from state A into state B without a collision occuring. Because of the continuity of the state–space, it has to be discretised by partitioning into a finite number of cells, with each cell representing a set of states. To navigate in an environment with many obstacles, the model of the real world or the state–space has to be decomposed to a very high degree. The higher the number of obstacles close to the robot, the more precisely its position should be known. Thus, the state–space has to be partitioned to a resolution as high as the "difficult" areas require. Unfortunately, such a uniform partitioning yields an enormous amount of cells in the state–space — even in areas where there are no obstacles. The basic idea of the parti–game algorithm is to avoid this completely inefficient static decomposition in favour of an adaptive partitioning of the state–space.

At first, the whole state–space consists of a single cell that is at least as large as the scene's base. The size of the environment is the only information that the algorithm needs to initialise the state–space.

The robot "erfährt" the neighbourhood relationships between cells when learning that there is a way from cell A to the adjacent cell B but there is no direct way from cell C to cell D (cf. Figure 1b). This pun with the word "erfahren" (German for "to experience" as well as "to drive around") describes rather appropriately the robot's learning behaviour. More precisely stated, the algorithm initially assumes that there is an action which transforms the robot's state into one that is in the goal cell. When the robot hits an obstacle (i.e. there is something within a radius of less than e.g. 40 cm), this assumption turns out to be wrong. In this case (i.e. the robot detected a collision when performing the planned action), this experience is added to the *knowledge base*. Under certain circumstances, which are explained below, the algorithm decides that the actual resolution is not adequate to the problem and refines the partitioning. The *knowledge base* in combination with the partitioning of the state–space enables the algorithm to find a path from any start situation to any goal state by driving from one cell centre to the next.

Besides the employment of a multi–dimensional state–space, parti–game is based on some principles and requires some prior conditions which are described next.

2.1 Prerequisites

The parti–game algorithm relies on three additional modules:

Self–localisation. The robot's current state vector must be provided to the controller to determine the corresponding cell.

Greedy controller. Assuming that there is an obstacle–free trajectory from one state to another, there has to be a controller that is able to perform these "direct" movements [3].

Collision–detection. Because of the algorithm's ability to learn the topology of its environment from scratch, a reliable mechanism for collision detection has to be implemented.

More details on the implementation of these prerequisites are given in section 3.

2.2 Major Principles

Two important concepts that distinguish the parti–game algorithm are the *partitioning strategy* (i.e. the decision when to divide which cells) and the *worst–case assumption*. They are explained in some detail below.

Partitioning Strategy

If there is no way out of a cell (with regard to current partitioning and knowledge), this cell is called a *direct losing cell*. This especially implies that the robot cannot reach the goal cell from a direct losing cell (cf. Figure 2a).

In spite of the fact that transitions are possible between some cells, these cells may nonetheless be losing cells. This is the case if such a set of cells does not have any connection to at least one cell with a known path to the goal. Such a cluster of cells from which the robot cannot escape, is called a set of *indirect losing cells*.

To summarise, from an indirect losing cell there are only transitions to direct or indirect losing cells, as shown in Figure 2a. In this state–space, solid lines represent cell borders between cells, for which the robot learned that there is no possible transition. For the dashed lines either no knowledge has been gathered so far[1] or the robot has already successfully performed the corresponding movement. If the robot is placed in a losing cell, the resolution has to be refined.

The question asking which cells are to be divided, is one of the design decisions when implementing the parti–game algorithm. According to the principle of adaptive decomposition, only those cells should be divided which contribute to reaching the goal. The following approach has been introduced in [3]. The set of cells which have to be partitioned consists of all losing cells neighbouring non losing cells and these non losing cells themselves. As depicted in Figure 2b, there is one continuous line dividing the state–space into two parts. Obviously this "wall" has to be broken to find a new way to the goal cell. This is done by refining the resolution in the regions covering this "wall". Therefore the cells mentioned above are divided. Even though the upper left cell is a direct loser cell, it is not relevant for the problem considered. This is because it has not to be visited on the way from start to goal cell. Therefore it will not be divided. In contrast to that, the starting cell in the lower left corner remains untouched although it is a losing cell on the robot's path. This is intuitively reasonable because a division of this cell will not yield any useful results. The knowledge is only acquired when trying to solve a certain task. Consequently this knowledge is particularly valuable for this specific task. This differs from map–building algorithms that are based on random exploration. If there is a path to the goal, it will be found after sufficient partitioning using the technique described here.

[1] and therefore — as mentioned above — the optimistic case of a possible transition is assumed

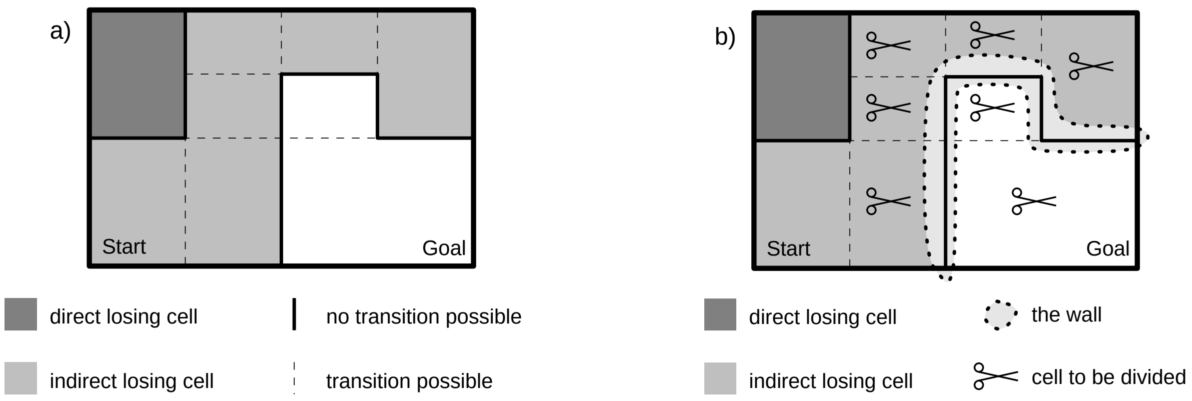


Figure 2: *Concept of losing cells:* Figure a) shows a state–space consisting of nine cells, one of which is a direct losing cell and six are indirect losing cells. Figure b) shows the "wall" along which the cells are divided.

Worst–Case Assumption

If the robot failed to drive from one position in a cell to a neighbouring cell, it is assumed that it is impossible to reach this adjacent cell from any position within the starting cell. This *worst–case assumption* prevents the robot from trying this cell transition until one of the two cells is partitioned.

The question whether or not it is a good idea to combine the optimistic attitude about the environment (i.e. assuming that transistions that have not been tried yet are possible) with this restrictive worst–case thinking (i.e. failing once means failing forever) will be discussed here.

The approach of the adaptive decomposition of the state–space suggests that states grouped together in one cell will not necessarily be of the same kind (i.e. allowing the goal cell either to be reached or not). This is what the stepwise refinement of the partitioning of the state–space is all about: to improve the decomposition in a way that as few cells as possible contain clusters of differently typed states.

Knowing that there will always be some inhomogeneous cells, a decision has to be made as to which of the different types of states in a certain cell should serve as *the* type of this cell. In other words, if the robot succeded in reaching a neighbouring cell N from one state in a cell A and failed to do so from another state in the same cell A, should it avoid A next time heading for N or should it hope that it will work again?

Obviously, the optimistic approach is prone to result in a behaviour of the robot that includes many stops and retreats. A handicapped person in a wheelchair should not be forced to cope with such inappropriate behaviour. Consequently we implemented Moore's and Atkeson's (cf. [3]) worst–case idea.

2.3 Extensions to the Parti–Game Algorithm

Our extensions to the original parti–game algorithm (cf. [3]) are described below.

Path–finding

A means has to be provided that enables the algorithm to decide which cell to head for in order to find an optimal path. In this context, the *transition costs* assessing a single cell transition and the *distance from the goal*, calculated as the sum of the costs of the involved transitions, are especially relevant. In [3] the authors propose to compute a path's length by counting the cell transitions between the start and the goal state (i.e. each cell transisition causes "costs" of one unit). In this approach paths involving larger cells are prone to be considered short, even though they are physically long.

Therefore, we use the real world distance between two positions to rate the cell transitions (i.e. the transition costs are equal to the euclidian distance between the corresponding cell centres if relevant experiences have not been made yet). If the robot already tried a certain transition, the corresponding costs are equal to the distance that had to be driven practically.

However besides the intended effect, this extension to the original parti–game algorithm caused some trouble. When assigning different costs to cell transitions it is no longer possible to calculate a cell's distance from the goal by means of the dynamic programming technique that is used in [3]. As indicated above, that approach employs a cost function. It states that the distance of a cell to the goal is either zero (i.e. for the goal cell itself) or $1+d$, where d is the distance from the most favourable neighbouring cell to the goal. The most favourable neighbouring cell is selected with regard to the worst–case assumption as explained above. Initially the distances of all cells are assumed to be zero. Iterating this cost–function n times for all n cells will result in a situation in which every cell has a properly calculated distance. Only for losing cells would the distance to the goal still increase when iterating the process one more time. Hence, these losing cells can be determined by this algorithm.

Simply replacing 1 in the expression $1 + d$ by the real world distance between the centres of the two involved cells fails: if relatively high costs for some transition of a cell meet very low costs for other transitions out of the same cell, the losing cell criterion does not work any longer.

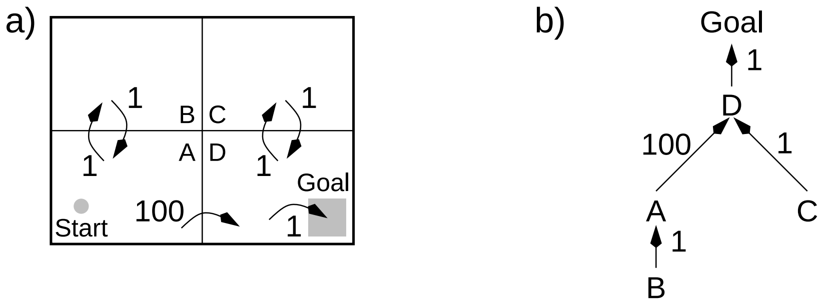


Figure 3: Figure a) shows a situation in which the robot is to drive from the starting point in the lower left corner to the goal cell. The knowledge gathered so far is depicted by arrows. Note that there is no transition possible between cells B and C. Figure b) shows the corresponding "shortest path tree". See text for more explanation.

Figure 3a shows a simple state–space with only one path from the start to the goal. Unfortunately this path includes a very expensive transition (A to D). High costs may have been caused by curving movements for example. On the other hand the transition from cell A to B is extremely cheap. Therefore the iterative process will oscillate between these two cells for a long time, because no detection of cycles is included. As a result, the determination of losing cells fails.

This is why another path–finding algorithm has been used. It calculates a kind of a *shortest path tree*, known from graph theory as Dijkstra algorithm (cf. e.g. [2]). The edges in Figure 3b represent cell transitions with the corresponding costs, whereas the vertices stand for the cells themselves. A shortest path tree describes the shortest path from a particular vertex to any other one in the graph.

Fortunately this approach is very suitable — with the exception that it uses the wrong orientation. It calculates shortest paths from *one cell to all* other cells. Instead, parti–game needs to find the paths from *any cell to one* goal cell. The required alteration is straight forward: just interpret the tree "upside down" by assigning the experienced distance between cell X and Y to the edge between Y and X (cf. Figure 3b). The construction of such a shortest path tree works as follows: First, the distances to the goal of each cell is set to infinity — except for the goal cell itself (distance=0). All these cells will be considered in the first run of the algorithm. The one with the least distance is chosen and extracted from the set of cells. Its distance is assumed to be correct. For all remaining cells, it is checked whether any knowledge has been gathered so far that is relevant with regard to the chosen cell. If so, the distance of such a cell is set to the sum of the distance to the goal of the chosen cell and the transition costs to the chosen cell. Thus, after the first run the distance to the goal of the nearest cell is calculated correctly. Then, by induction, after n runs the distance of all n cells is correct.

Finally, determining losing cells is also very easy: all cells whose distance is still infinity (the initial value) are losing cells.

Local subgoals depending on the task

Even though the paths are planned from one cell centre to the next, the real goal that the algorithm tells the greedy controller to head for, is a different. It proved useful to move each subgoal slightly towards the robot depending on the next subgoal. In this way the robot avoids moving too far into an intermediate cell.

Neighbourhood relationships

Additionally to avoid zigzag trajectories, the neighbourhood relationships were extended. Two cells are adjacent if they have one common edge or are connected diagonally.

3 Parti–Game Implementation on the Wheelchair

The special difficulties that arose due to kinematic restrictions and how we addressed them are now described. A more detailed presentation of the prerequisites mentioned in section 2.1 is also included.

3.1 Problems in Navigating a Wheelchair

In contrast to many experimental platforms in robotics, a typical wheelchair as used here can only follow curved trajectories. Besides this, the possible movements with regard to a particular position depend on the current orientation of the vehicle. Often reaching certain positions is only possible by shunting. Furthermore, our long–term intention (to use the autonomous wheelchair to support handicapped or elder people in daily life) has to be kept in mind when navigating. The handicapped user probably cannot be expected to sit in a wheelchair that drives backwards in serpentines from the kitchen to the living room. Therefore, in our implementation of the parti–game algorithm the robot moves forward if possible, and avoids long curves. Natural beings underlie similar restrictions and have preferred movement directions. An additional reason to have biased movements is that the arrangement of sensors are typically not evenly placed in all directions.

3.2 Design of the State–Space

The essential problem for the parti–game algorithm — caused by such kinematic restrictions — arises, if a state–space is chosen that does not match the robot's degrees of freedom. In this case it is not always possible to change the robot's state from one state–space cell to a neighbouring one in a single move. Unfortunately, it is not easy to find a state–space that complies to a wheelchair and has a bijective mapping to the real state of the robot. To solve this problem, a simple state–space is used, but the greedy controller receives information about which steps are appropriate to circumvent kinematic restrictions.

The state of our wheelchair includes its position in (x,y)–coordinates, orientation, steering, driving direction and speed. Only the (x,y)–coordinates are used in the state–space. In addition, for each intermediate target an orientation is calculated. This is done by planning ahead and calculating the next target assuming that the current step is successfully performed. Then the orientation is determined simply by finding the direction from the actual target to the next one. All other values are left to the greedy controller, which knows when to drive backwards and how to use steering to change the orientation. The low–dimensional state–space has the advantage of faster learning of which trajectories to take, because less cell partitions are needed to find a collision–free path. The disadvantage is that no learning takes place of how to master a trajectory, i.e. no information about the performed actions and manoeuvres is stored.

3.3 Greedy Controller

The greedy controller receives a target state and has to convert the actual state to this target. If the target lies inside a narrow corridor in front of or behind the wheelchair — relative to its actual orientation — it can be reached by simply moving with a slight steering; this is the original simple state translation which a "greedy"–controller should perform. If the target lies outside the tolerance region, it is reached by a combination of two turns and a straight movement. With the first turn the wheelchair is directed to its target, then in a straight line it drives towards the target. The second turn is done to reach the correct orientation at the target position. Depending on the current situation, the turns can be made forward or backward. For example, if the robot reached an obstacle in the last move, it may be neccessary to move backward in order to get away from the obstacle. The straight line is always a forward movement. In theory, using these combinations every position is reachable. To minimise deviations, a controller was implemented which checks the position during movement and adjusts the steering if needed. It also avoids abrupt changes from a curve to a straight movement. Smooth transitions are realised instead.

3.4 Self–Localisation

Self–localisation is achieved using dead–reckoning and a video camera that detects landmarks. In the current set–up, predefined landmarks at positions that are known to the self–localisation module are employed (cf. Figure 1a). Through calculation of the landmarks' width and height, the actual position and orientation relative to the room is computed. In addition, odometer sensors are used to measure the movement of the wheels. To avoid accumulating errors in dead–reckoning, it is regularly corrected by new video–calculated positions. Future experimental set–ups will include cameras mounted on pan–tilt heads to track non–artificial landmarks.

3.5 Collision–Detection

Another prerequisite for parti–game is an ability of the greedy controller to detect collisions. This is achieved by a "binary sensor shield" consisting of bumpers, ultrasonic and infrared sensors. If any obstacle is detected within a certain range in the direction of the current movement, including swinging out motions, it is considered an "anticipated collision". This is reported back to the parti–game algorithm, which then calculates a new target position.

4 Experimental Results

Experiments with the wheelchair were performed in a lab environment as well as under presentation conditions with people crowded around the scene. The room's size, in which the wheelchair navigated, was $7m \times 5m$. The wheelchair was able to learn to navigate between five arbitrarily chosen positions. Even when confronted with sensor inaccuracy and communication delays in early versions of the hardware/software architecture, the algorithm successfully partitioned the state–space. Furthermore, experiments revealed the algorithm's robustness against inaccurate sensor–values, e.g. when recovering from wrong positions provided by the self–localisation module due to low quality colour perception.

If the collision detection module fails by wrongly signaling obstacles, the worst–case assumption seems to be disadvantageous at first glance. But the algorithm is robust against this kind of errors because it recovers from the incorrect knowledge. This happens when no other path can be found and the partitioning is refined. In this case, the old experiences are forgotten.

In the following, one complete adaptation process is described in detail.

4.1 Example of an Exploration

In this experiment the wheelchair had to learn to navigate in a room with two obstacles, shown as the two grey boxes in Figures 4 to 7. Note that the location of these real world obstacles is not explicitly learned by the algorithm. They are projected into the state–space for explanation purposes only. The dashed lines in these figures connect the points at which new decisions had to be made. The the real trajectories of the wheelchair may differ from these lines.

At the beginning of the first run there is no knowledge available about the environment (Figure 4, a). The first move is to directly reach the target. In the first step (a) the current intermediate target (depicted as a hollow arrow) is equal to the final target of the task. The robot starts at the position and orientation represented by the filled arrow. This attempted move results in a collision with an obstacle (b) and as a consequence the state–space is partitioned. Then an intermediate target is chosen, taking the centre point of the upper cell and moving it a certain amount towards the actual position. This is performed according to the procedure described in section 2.3. This new target in the upper half of the

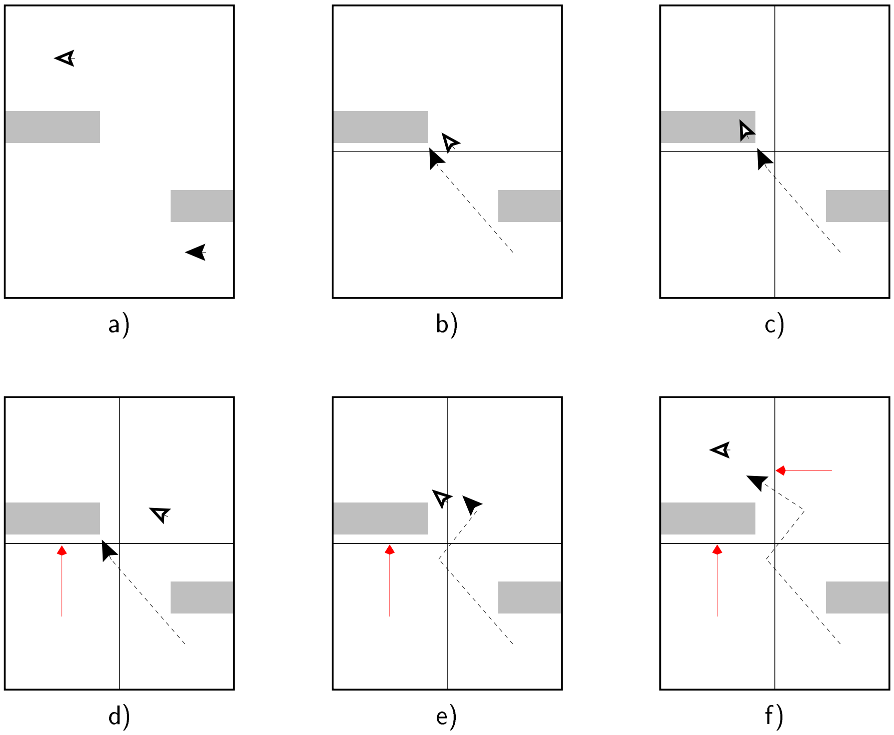


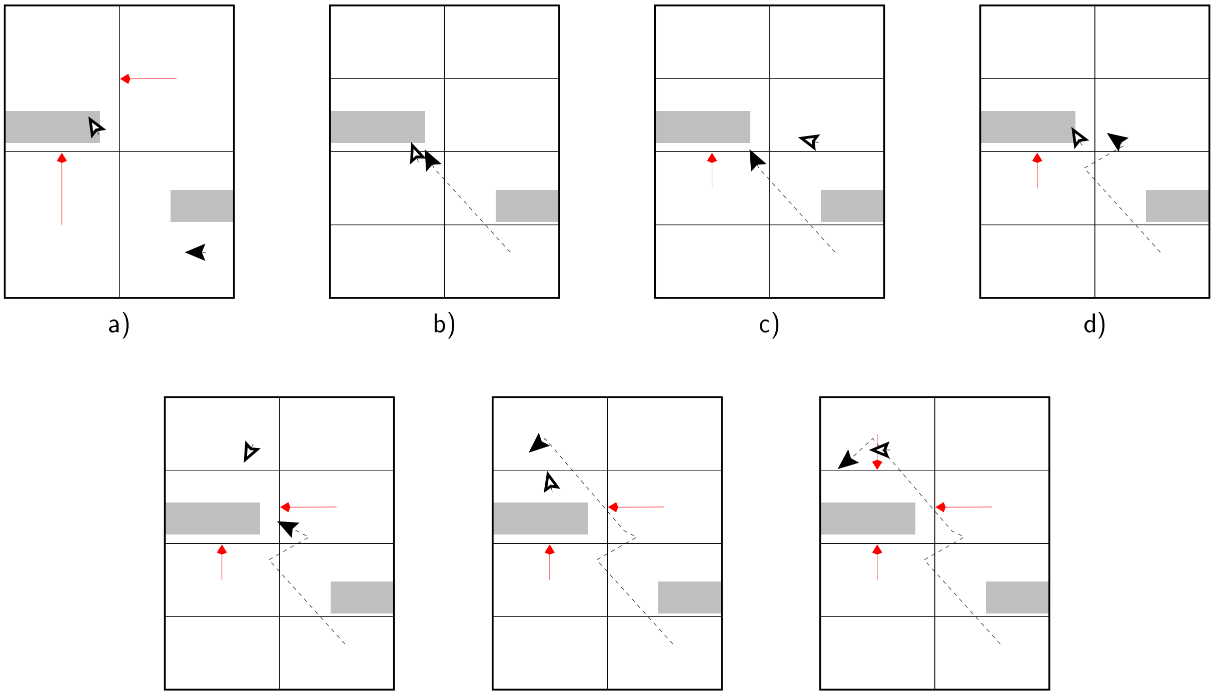
Figure 4: The first run: Learning of the basic partitioning.

room is still not reachable, so the state partitions are refined again (b→c). After that the neighbouring cells are headed for one after another (c, d). In this process, experiences made with impossible cell transitions — because of obstacles — are visualised by small arrows pointing into the direction of the impossible transition. At last a way to the target is found (f), although in this move the wheelchair came too close to the obstacle and was rated as an almost collision. Because of this, it memorised not to try this way again. In this first run the basic partitioning of the environment is learned.

In the second run the starting position is set back to the lower right (Figure 5, a). The knowledge learned in the first run permits only one possibility: to directly head for the cell containing the target. After failing, the partitioning has to be refined again (b). Now the procedure of trying all neighbouring cells is repeated (c, d, e, f). After reaching the cell of the target state, parti-game decides to head for the target using a detour instead of taking an expensive turn. This fails again (g), because the outer left wall is hit. Fortunately now the target state can be reached simply by driving backwards. In the second run all neccessary knowledge was learned to handle the upper left region of the state–space.

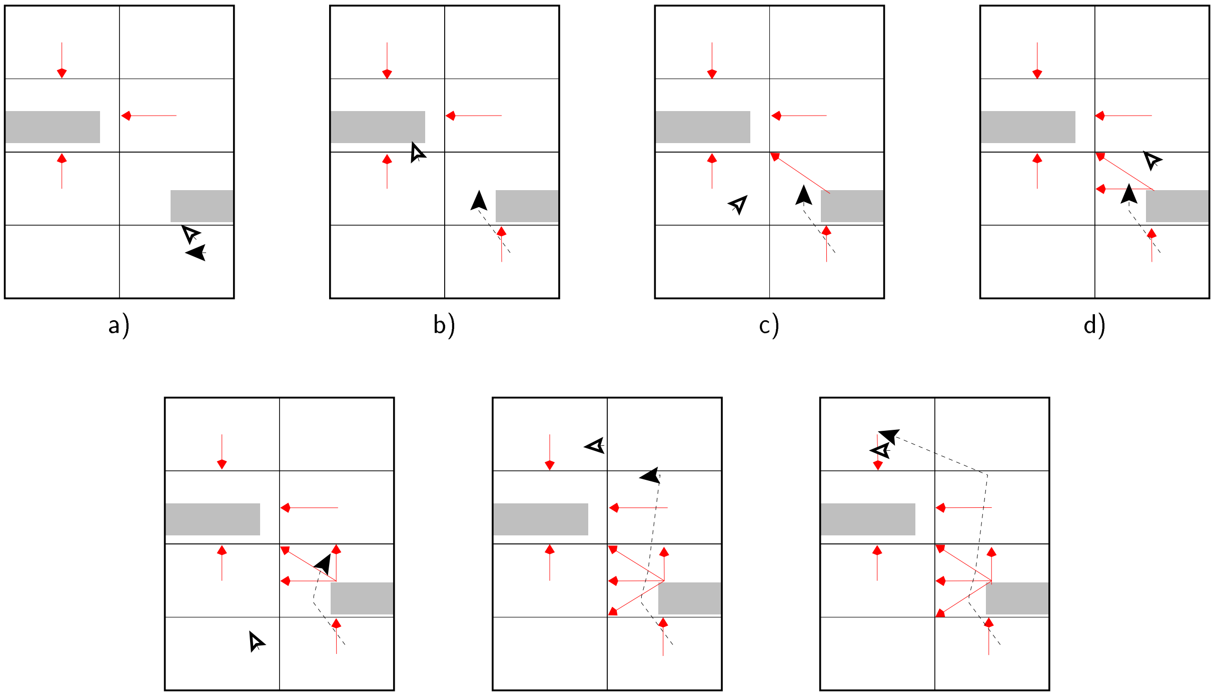
At the beginning of the third run (Figure 6, a), no knowledge exists about the lower right region, because of the last partitioning. Again, the direct way is chosen. Now, the wheelchair comes to an unfavourable position on the left side of the obstacle (b). It is hard to leave this position because the wheelchair's back wheels swing out in curves. Finally, after some movements it gets away from the obstacle (c, d, e, f, g). In (e, f) the wheelchair tries to go to the lower left cell, because this is the last remaining possibility of the knowledge base. To achieve this, extensive movements are needed. Accidentially, after getting away from the obstacle, these movements lead to the upper right cell when parti-game decides to choose a new target. This way, finally it manages to reach the target. In this run, the knowledge needed to navigate in the lower right region was learned. Now after three learning cycles, enough knowledge has been acquired to avoid all obstacles when driving from the lower right to the upper left of the room.

The last run demonstrates a collision–free navigation through the parcours, using the experience gained before (Figure 7). In each step, the favourable next cell on the way to the target is chosen avoiding all cells in which a collision occured in the past. After four steps, the target state is reached without colliding with an obstacle and without modifying the knowledge base.



a) b) c) d) e) f) g)

Figure 5: The second run: Learning in the upper left area.



a) b) c) d) e) f) g)

Figure 6: The third run: Learning in the lower right area.

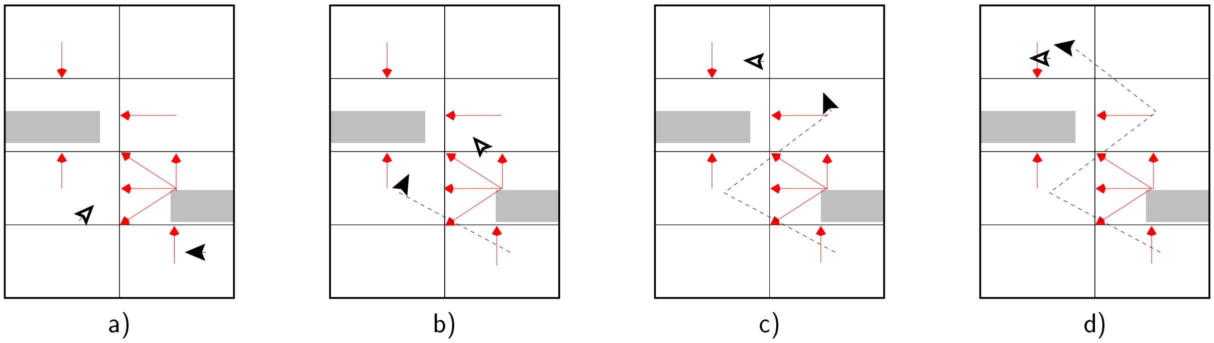


Figure 7: The fourth run: The wheelchair reaches the goal without any collisions or relearning.

5 Conclusion and Future Work

Based on the work of Moore and Atkeson [3], the parti–game algorithm has been adapted and extended to control a real wheelchair. Therefore different extensions had to be made to the algorithm including a new concept of costs and extended cell neighbourhood relationships. The parti–game algorithm has proved to be able to navigate a wheelchair under lab conditions, but problems occured that inhibit the application as a product in daily life. This is partly caused by the dependence on several additional modules like collision–detection and self–localisation.

Future work will include the development of an alternative state–space design that matches the wheelchair's degrees of freedom more closely. This can reduce the amount of the greedy controller's assumptions and improve the robot's behaviour in critical regions. In addition, aspects of dynamic obstacles will be examined. One idea is to initially learn the state–space representing the static parts of the environment, e.g. walls, doors and cupboards. Any further experiences are considered to be temporary and can be forgotten simply by falling back into a system state of the previously learned static knowledge.

Furthermore the results will be improved when porting the parti–game algorithm to a new, more sophisticated wheelchair with enhanced sensing facilities.

6 Acknowledgements

The semi–autonomous wheelchair has been developed in the student project “Sensomotorik Autonomer Systeme” (SAUS) and the Graduiertenkolleg “Raumorientierung und Handlungsorganisation autonomer Systeme”, supported by the DFG (Deutsche Forschungsgemeinschaft) and the Freie Hansestadt Bremen. The authors especially thank Ch. Herwig, M. Heger from the Graduiertenkolleg and all members of SAUS, without whose help this work would not have been possible.

References

[1] Heger, M. (1996). Risk–Averse Reinforcement Learning (PhD thesis). ZKW–Bericht 5/96. Zentrum für Kognitionswissenschaften, Universität Bremen. ISSN 0947-0204.

[2] Jungnickel, D. (1990). Netzwerke und Algorithmen. 2. Auflage. BI-Wissenschaftsverlag Mannheim.

[3] Moore, A. & Atkeson, C. (1995). The Parti–Game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State–Spaces. Machine Learning 21, pp. 199–233. Kluwer Academic Publishers, Boston.

[4] Siems, U., Herwig, Ch. & Röfer, T. (1994). SimRobot — Ein Programm zur Simulation sensorbestückter Agenten in einer dreidimensionalen Umwelt. ZKW–Bericht 1/94. Zentrum für Kognitionswissenschaften, Universität Bremen. ISSN 0947-0204.