Escaping from a Labyrinth with One-way Roads for Limited Robots

Bernd Brüggemann* Tom Kamphans† Elmar Langetepe†
*FKIE, FGAN e.V., Bonn, Germany
†Institute of Computer Science I, University of Bonn, Bonn, Germany

***Abstract*— In this paper, we consider the problem of navigating a robot with limited abilities concerning computing power, memory, and sensors through a labyrinth with one-way roads. Escaping from a labyrinth is a task which is widely explored. Many algorithms with different advantages and different areas of application are known. Usually, labyrinths are given as polygonal scenes or (directed) graphs. While scenes of the first type preserve the geometry of the 'real' environment, the latter preserve the connectivity, which is crucial if there are passages that can be traversed in only one direction (e.g., one-way roads). Our approach combines both advantages. In this work we present some properties of labyrinths with one-way roads. With the help of these properties, we were able to develop algorithms that allow a robot with very limited abilities to solve any 'fair' labyrinth.**

***Index Terms*— navigation, path planning, low budget robots**

I. Introduction

To navigate in an unknown environment is one of the most important tasks that autonomous robots have to solve. There are many different approaches to solve this problem. Among them are, for example, SLAM algorithms [1], which use probabilistic methods to build up a map to navigate.

Most algorithms that build a map at runtime, need robots with adequate memory, computing power, or advanced sensor techniques, such as laser sensors or video cameras with extensive image processing.

But sometimes it is necessary to use smaller and cheaper robots. For some reason, like cost or space, you might want to use very *limited robots*. Limited in terms of very little computing power, small amount of memory and very limited sensory (e.g. only touch sensors). Therefore, there are a lot of algorithms which move your robot where it wants to be, without needing a map. In fact, the robot does not need much more than a touch sensor and a compass. Examples are the BUG algorithms [2, 3, 4, 5] or the Pledge algorithm [6].

The Pledge algorithm solves every possible labyrinth; that is, a robot, placed in a labyrinth, will find the exit of the labyrinth, using the following simple rules:

- move straight forward until an obstacle is hit (*free movement*)
- turn right until the obstacle is at the left hand side, log the turn (*angle counter*)
- follow the obstacle, while logging every turn made
- if the angle counter is equal to zero, leave the obstacle and move straight forward

For a correctness proof see, for example, [6, 7]. Please note that the range for the angle counter values is $[0; -\infty[$; it does not compute modulo $360°$. It is not equal to the heading of the robot. If the robot has turned three times $360°$ to the right (i.e., the angle counter decreases by $1080°$), it needs to follow the obstacle until it performed three full clockwise loops and the angle counter gets zero.

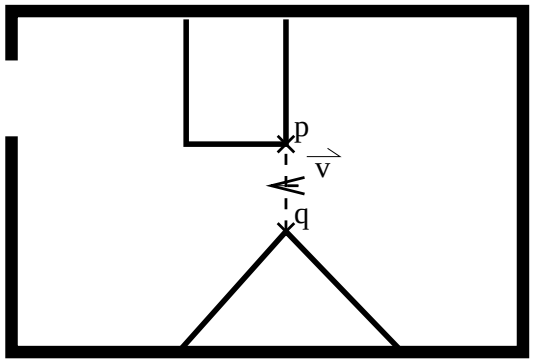

Fig. 1. A labyrinth with an one-way road. p and q have to be on the boundary of an obstacle. $\vec{v}$ gives the direction in which the robot may pass the one-way road.

The robot needs only a few abilities to follow these instructions: It needs touch sensors to detect an obstacle and to follow it. Further, it has to log the turns. Odometry will do fine here. Particularly as odometry does not need to be totally correct as shown in [8].

But as soon as there are other structures in the labyrinth the Pledge algorithm may fail. In this paper we analyze the effect of one-way roads in a labyrinth.

The idea of adding one-way roads to the environment is to model, for example, doors. Without an actuator, the robot may pass a door by pushing it. But arriving from the other side (i.e., when the robot has to pull the door), the way is blocked. Other examples for one-way roads a robot are ramps. Coming from one side, the robot can pass, but faced with the brink from the other side, the way is blocked.

In Section 2 we will present some properties of labyrinths with one-way roads. These properties help us to design algorithms to solve such labyrinths.

Two algorithms are presented in Section 3 and 4. For other solutions please refer to [9] and [10].

Section 5 shows an example implementation on a Khepera II robot.

II. Labyrinths with One-Way Roads

First, let us give a precise model for one-way roads:

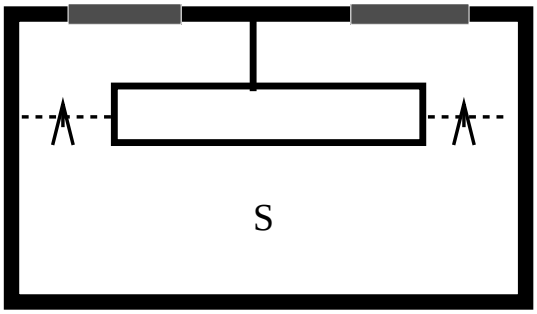

Fig. 2. An unfair labyrinth. The exit of this labyrinth is exactly one of the two grey boxes. The robot does not know which one. Even with a mapping algorithm, the robot cannot decide, which one-way road leads to exit and which one-way road leads to the dead end. It has to test one one-way road. But then the robot might be trapped. So there is no on-line algorithm which can deal with "unfair labyrinths".

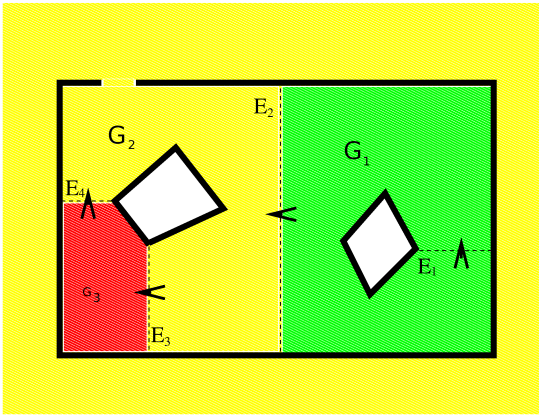

Fig. 3. A labyrinth with one-way roads can be partitioned into regions. Region G_2 is the outer region. Note that not every one-way road (here one-way road E_1) contributes to an region.

Definition 1 (one-way road): A one-way road is a line in the labyrinth that the robot can pass in one direction only. It is given by a line $\overline{pq}$ with points p, q on boundary of an obstacle, and by a vector $\vec{v}$, which indicates the direction in which the robot may pass the one-way road.

Thus, one-way roads in our labyrinth are simple lines instead of areas in the labyrinth. Save from the points p an q, no other point of the one-way road may lie on the boundaries of obstacles (for an example see Fig. 1). In addition, two one-way roads may not cross each other.

It may be impossible to leave a labyrinth with one-way roads: Imagine, for example, the exit may be blocked by a one-way road that leads inside the labyrinth, but not outside. But even if there is a path from the start to the exit, there may be dead ends behind one-way roads that trap the robot. Thus, we consider only *fair* labyrinths.

Definition 2 (fair labyrinth): In a fair labyrinth, there is a path from every point p to the exit.

This constraint is necessary, because even with the help of maps, in some labyrinths it is impossible to decide if an one-way road leads to a dead end. The robot may have to enter a trap before recognizing it, see Fig. 2.

The one-way roads induce—in a very natural way—a partition of the labyrinth into maximal path-connected components. We call these components *regions*.

Definition 3 (region): A region G is the maximum set of robot positions for which the following is true:
$\forall p_i, p_q \in G(p_i, p_j$: robot positions) exists a path from p_i to p_q that neither crosses an obstacle nor an one-way road.

Regions are a powerful tool to understand the behavior of robots in labyrinths and prove the correctness of labyrinth-solving algorithms. Figure 3 shows and an example of a labyrinth and its regions. Note that not every one-way road contributes to an region. Further, in every labyrinth there is exactly one *outer* region. The outer region is the area of the labyrinth from which the robot can escape without crossing any further one-way road.

Even if the labyrinth is fair, the Pledge algorithm may fail, see Fig. 4: Using the Pledge algorithm, the robot enters the one-way road, and gets trapped in an endless loop. If it would not pass the one-way road, it would escape. Thus, for a maze-solving algorithm it is necessary to decide, whether to enter or to bypass a one-way road.

Lemma 1: The Pledge algorithm solves every fair labyrinth with one-way roads, if the robot knows which one-way roads it has to pass and which ones not.

Proof: To escape from the labyrinth, the robot has to reach the unique outer region. Let this region be G_1. If the robot starts in G_1 and passes no one-way road, the Pledge algorithm will lead the robot to the exit. So let the robot start in region G_m. Then there is at least one sequence of regions $G_{m-1}, G_{m-2}, ..., G_1$ that the robot has to pass on its way from the start to the exit.

These regions are separated by one-way roads from each other. The robot starts inside G_m. It is allowed to pass only the one-way road that leads to G_{m-1}. So the Pledge algorithm will lead the robot to G_{m-1} (see above). Now, the robot may pass only the one-way road to G_{m-2}, and so on. If the robot passes only the one-way roads that leads it along this sequence—and no other one-way road—the Pledge algorithm will lead the robot to the exit. ■

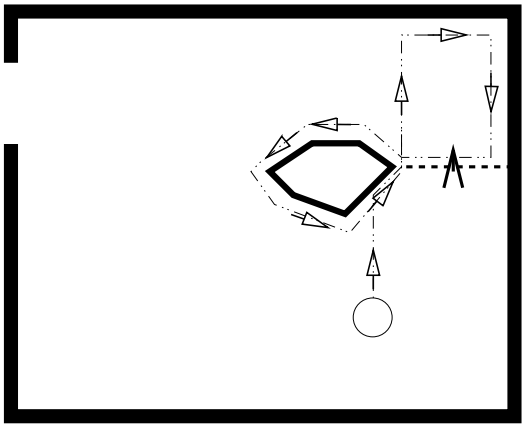

Fig. 4. The Pledge algorithm may fail in labyrinth with one-way roads

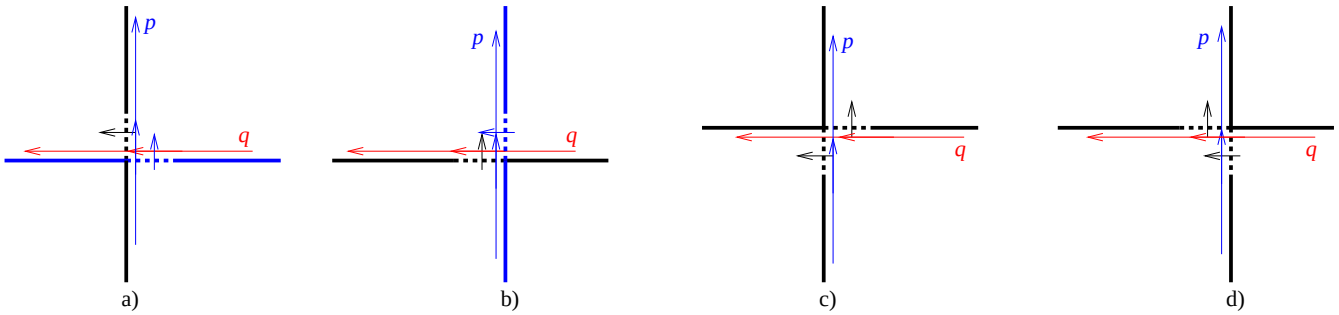

Fig. 5. The four possible configuration for intersections between two non-free movements. Only case a) and d) may be achieved by Pledge algorithm. Only in these cases the wall is at both parts of the path on the same side.

We call a set of instructions that tells the robot which one-way road the robot has to enter is called a *list of behavior*. Such a list of behavior can be given in two ways:

- Local list of behavior
- Global list of behavior

A local list of behavior tells the robot what it has to do at the **next** one-way road that it meets. It might be given periodically, such as the instruction: "Pass every second one-way road".

A global list of behavior is a mapping from the set of one-way roads to {"pass", "do not pass"}.

Figure 4 shows—beside the fact that the Pledge algorithm fails—another interesting property: in contrast to the path of a robot steered by the Pledge algorithm in simple labyrinths, the path in a labyrinth with one-way roads may have intersections.

Assume that a global list of behavior is used and that there is no change in this global list during runtime.

Lemma 2: Let the robot use a fixed, global list of behavior and let p and q be two segments of the robot's path. If p and q intersect each other then either p or q represents a free movement (that is, the angle counter is zero).

Proof: If both p and q are free movements, they cannot intersect: A movement is free when the angle counter is zero. Therefore, free movements are parallel and point in the same direction. Intersections involving at most one free movement may occur, as shown in Figure 4.

If there are intersections while both p and q are non-free movements, the robot must have an obstacle at its left-hand side. As the path intersects, the walls of the obstacle intersect, too. At the intersection point, parts of the walls have to be one-way roads. In Figure 5, all possible configurations are shown.

The cases b) and c) cannot be achieved by the Pledge algorithm, because the wall is at different sides of the robot. Using the Pledge algorithm, the robot always moves on the same side of the obstacles while performing a non-free movement. So only cases a) and d) can be achieved by the Pledge algorithm. Consider case a): Assume w.l.o.g. that p is reached before q. When p was reached, the vertical one-way road had not been passed. Now, when q is reached, the vertical one-way road has to be open. So the behavior for this one-way road has been changed, but this contradicts to our assumption that the list of behavior is fixed. The same arguments hold in case d). ■

From Lemma 2 we conclude another interesting property of labyrinths with one-way roads:

Proposition 1: Let the robot use the Pledge algorithm and a fixed, global list of behavior in a fair labyrinth with one-way roads. If the robot performs a closed, endless loop then there is exactly one point where the robot's path intersects itself.

Proof: (Sketch) If there are no intersections in the endless loop, the robot makes either a full counterclockwise or clockwise turn (i.e., $\pm 2\pi$) in every cycle. In the first case, the angle counter increases by $+2\pi$ per cycle; and, thus, it will eventually get zero. But then the robot leaves the obstacle and breaks the loop. In the second case, the counter decreases in every cycle. Thus, the robot is trapped in a courtyard and the labyrinth cannot be solved [6].

Now, let the robot move on an endless loop with intersections. If we cut the loop in the intersection points, we get several Jordan arcs, each of them increases or decreases the angle counter by 2π if the robot moves on the arc. Assume that the loop has a different number of counterclockwise and clockwise arcs: The angle counter will increase or decrease by 2π every time the robot surrounds the loop. Using the preceding arguments, such a loop cannot exist.

Therefore, there may be only loops with the same number of counterclockwise and clockwise arcs; that is, the angle counter in a fixed point on the loop does not change when the robot moves on the loop, see Figure 6. Loops driven by

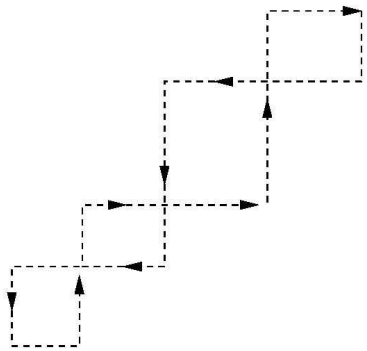

Fig. 6. An example for a loop with four Jordan arcs, two of them clockwise and two counterclockwise. This loop cannot be achieved by the Pledge algorithm because the free movements do not point in same direction.

the Pledge algorithm have a second property: After a free movement, there is always a right turn. Hence, after hitting an obstacle the robot either moves on a clockwise circle or it leaves the obstacle before it performs a full counterclockwise circle. Thus, a loop with n intersections yields n arcs where the robot turns clockwise (because there is a free movement for every intersection). So the robot needs additional n circles to preserve the angle counter values. But this is possible only for $n = 1$; that is, there is exactly one counterclockwise arc, one clockwise arc, and one intersection. ■

III. Control-Word Pledge

The control-word Pledge is designed to search for a local list of behavior. The idea of the control-word Pledge is adapted from [11]. Hemmerling has given an algorithm that allows a robot to reach any point in a labyrinth, equipped only with a compass that shows the direction of the target. The robot does not need to store anything connected with the labyrinth.

The control word in the control-word Pledge describes the local list of behavior. Therefore, the alphabet Σ consists of the following letters:

- "p": pass the one-way road
- "n": do not pass the one-way road

Every finite word, build with the help of Σ, matches a non-periodic local list of behavior. As additional function we need a third letter in Σ:

- "r": reset the angle counter

Obviously, "r" does not tell the robot how to pass a one-way road. An "r" always has to be followed by another letter. Also it is senseless to perform two (or more) "r" in a row.

Lemma 3: From every one-way road there is a word in Σ^* that leads the robot to the exit.

Proof: As shown in Lemma 1, for every start point there is always at least one global list of behavior that leads the robot to the exit. This list can easily be transformed into a local list of behavior. Further, every local list of behavior can be given by a word over Σ. ■

Of course the word, that leads the robot to the exit, is not known in the beginning and depends on the start point.

Theorem 1: For every fair labyrinth it exists an finite word w_{uni} over Σ^* that leads the robot from every one-way road to the exit.

Proof: From Lemma 3 we know that there is a word w_a that leads from one-way road A to the exit. Now, assume that robot starts at one-way road B. Performing the word w_a may lead the robot to the exit (and the algorithm terminates). More likely, it will move the robot to another one-way road, Q. This is the first one-way road for which the word w_a has no further command. If the robot resets now, we known that there is a word w_q that leads the robot starting in Q to the exit of the labyrinth.

So the word $w_{ab} = w_a + r + w_q$ leads from two one-way roads (A and B) to the exit. The word w_q assumes that the robot starts with an angle-counter value of zero. To ensure that this requirement is fulfilled we reset the angle counter before processing w_q.

Now, we consider a third one-way road C as start point. Steered by the word w_{ab}, the robot reaches the one-way road V, and so on. This can be done for each one-way road. This construction yields the word w_{uni} that leads the robot from every one-way road to the exit. ■

Of course, w_{uni} is not known a priori and, of course, w_{uni} is different for different labyrinths. But we know that it exists and we know that it is finite (because w_{uni} consists of a finite number of finite words). Therefore, it is possible to search systematically for it. That is, we enumerate—one by one—each word in Σ^* and evaluate it character by character in the following way:

Each time the robot driven by the Pledge algorithm reaches a one-way road, it behaves as the current character demands. At first the robot applies every word over Σ^* which consists of one letter. If the robot leaves the labyrinth, we are done. Otherwise the robot chooses every two-letter word and so on. By Theorem 1, the robot will find the exit at the latest when it finds the word w_{uni} (perhaps the robot finds the exit much earlier, but that would be by chance). The performance of our algorithm depends on the length of the word w_{uni}. Therefore, we give an upper bound for the length of a word from a single one-way road to the exit.

Lemma 4: The length of a control word that leads from one one-way road to the exit, is in $O(N)$ where N is the number of one-way roads.

Proof: The robot starts in the region G_m. The path to the exit is a sequence of regions $G_{m-1}, G_{m-2}..., G_1$. If the robot hits an one-way road that leads from G_i to G_{i-1} it will pass the road, otherwise it will not pass. So for every one-way road there has to be only one decision. Therefore, we need N letters either "p" or "n". As mentioned above it is reasonable to use one "r" in a row. So we can add at most N "r"'s. So one word consists of $2N$ letters and is in $O(N)$. ■

Theorem 2: At most $3^{O(N^2)}$ characters have to be tested before finding w_{uni}.

Proof: The word w_{uni} is a concatenation of N words with $2N$ letters, where N is the number of one-way roads. Therefore, w_{uni} has length at most $2N^2$.

This means that the robot has to check every word up to a length of $2N^2$ until it finds the exit. The total number of characters tested is:

$$\sum_{i=1}^{2N^2} 3^i = \frac{1}{2} \cdot 3^{(2N^2+1)} - \frac{3}{2} \in 3^{O(N^2)}$$

■

It is obvious that this algorithm is not usable in practice for a large numbers of one-way roads. The search for a global behavior lists might give a more efficient algorithm, but requires the capability of distinguishing one-way roads.

IV. Binary Pledge

As shown in section 2, there are lists of behaviors that lead the robot out of a labyrinth with one-way roads. The binary Pledge searches for an appropriate global list of behavior.

The idea of the binary Pledge is to map a list of behavior to a binary number: Every digit in the number determines the

behavior of one one-way road. '0' means 'do not pass' and '1' means 'pass'. Hence, every possible list is mapped to a N-digit binary number, where N is the number of one-way roads. Therefore, the algorithm searches for a global list of behavior. The one-way road E_1 is connected with the first digit of the binary number, E_2 is connected with the second, and so on.

Now, the robot has to test all possible binary numbers until it finds its way out. The robot no longer makes local decisions. Instead, it has to determine if the chosen binary number leads to the exit.

This decision is possible with the help of the one-way roads Using the one-way roads as landmarks, we can determine if the robot drives a loop or not. Sometimes it is acceptable that the robot drives in a loop, for example in a spiral-like environment. But sometimes a loop indicates that the robot cannot find the exit with the current binary number (i.e., the current global behavior), so the binary number has to be changed.

To decide whether a loop is an endless loop we use the angle counter. We observe the changes in the angle counter at the one-way roads over time. There are four possibilities how the counter changes:

- No angle-counter value changes
- Every angle-counter value on every one-way road becomes smaller over time
- At least at one one-way road the angle-counter value becomes bigger over time
- No angle-counter value becomes bigger but at least one becomes smaller

Note that such a statement is possible only when the robot has driven the loop at least two times.

A special movement in the Pledge algorithm is the free movement. To decide if the robot performs an endless loop every criteria above has different meanings if there was a free movement or not. So we have to check these two kinds of movements with every of the four criteria:

- There was at least one free movement in the loop
  - *No angle-counter value changed over time:* Because the Pledge algorithm is deterministic, the robot will perform the same loop on and on. The loop is endless. The robot drives a path with one intersection (see above).
  - *Every angle counter value on every one-way road becomes smaller over time:* Because the angle counter gets smaller, the free movement must vanish over time. This is not an endless loop.
  - *At least at one one-way road the angel counter value becomes bigger over time:* Because the angle counter cannot be positive, there will be a change in the path in time. This is not an endless loop.
  - *No angle counter value becomes bigger but at least one becomes smaller:* Same as former. The free movement will become impossible over time. The path will change so this is not an endless loop.
- There was no free movement in the loop
  - *No angle counter value changed over time:* In this case, the robot's path will also never change, so this path is an endless loop.
  - *Every angle counter value on every one-way road becomes smaller over time:* There is no free movement; therefore, the robot is trapped inside an inner courtyard. This is an endless loop.
  - *At least at one one-way road the angel counter value becomes bigger over time:* The angle counter will become zero, so a free movement will occur. This is not an endless loop.
  - *No angle counter value becomes bigger but at least one becomes smaller:* There is no free movement and no one will occur. This is an endless loop.

	no free movement	free movement
at least one angle counter becomes bigger	path will change	endless loop
every angle counter stay equal	endless loop	endless loop
every angle counter becomes smaller	endless loop	path will change
no angle counter becomes bigger, but at least one becomes smaller	endless loop	path will change

TABLE I

THE POSSIBLE CHANGES FOR THE ANGLE COUNTER OVER TIME AND THE CONCLUSIONS (I.E., ENDLESS LOOP OR NOT).

Therefore, we have found the criteria for a binary number to lead to the exit or not (see table I). So we can describe the algorithm:

1) move with Pledge algorithm until a one-way road is hit
2) store one-way road and angle counter value in list
3) test if robot has driven an loop
   - if not: continue with 4)
   - if there is an loop: Check if it is driven twice.
     - if not: continue with 4)
     - if driven twice, check if an endless loop criteria is found
       - ∗ if not: continue with 4)
       - ∗ if endless loop is found, increment the binary number, reset angle counter, and continue with 4)
4) handle one-way according to the current binary number. Continue with 1)

Although it is unknown how many one-way roads exist in the labyrinth, the robot can start with the assumption that there is a one-way road. It starts with the binary digit '0'. If there is no one-way road, the robot will escape (due to the correctness of the Pledge algorithm). If there is one, it also will escape (due to try to pass and try not to pass). Let the first one-way road hit be called E_1. If there are more one-way roads they will be added as additional digits to the binary number. Here an example:

The robot knows the one-way roads $E_1, ..., E_4$. Therefore, it has stored a 4-digit binary number to handle the list of behavior. Let the current binary number be *'0110'*, where the digits assigned to the one-way roads as follows: 'E_4, E_3, E_2, E_1'. If the robot hits an unknown one-way road, E_5, it adds it: 'E_5, E_4, E_3, E_2, E_1' and the current binary number is *'00110'*.

Fig. 7. The Khepera II with a CMUCam turret. The wires are only used to submit status messages. The whole computation is done on the Khepera itself.

Fig. 8. An example toy world labyrinth in which we tested the one-way road pledge. The marks on the floor represent the one-way roads. A red color indicates the side from which the robot may not pass. Other colors are used to identify each one-way road and the side, from which the robot may pass.

In comparison to the control-word Pledge the performance is much better. Obviously the maximal length of the binary number is N; therefore, the algorithm has to check at most $2^{(N+1)} - 1$ different binary numbers which means an considerable saving of time.

V. Implementation on Khepera II

The Khepera II, developed by the K-Team, matches our definition of a limited robot. It has only small computing capabilities due to limited memory and computing speed. Further, its sensory, eight proximity sensors with a range of approximate eight centimeters, limits the field of application.

The only extension we have done to the standard Khepera II is a turret with a CMUCam (see Fig. 7). This was necessary to give the robot the capability to distinguish between different one-way roads. It is not possible to do any image processing on the Khepera II because the transmitting of the picture into the robot's memory lasts about 10 seconds. So we used the camera build-in function to track a specific color (for color coding the one-way roads see Fig.8).

The experiments show that such a labyrinth could be solved by the Khepera II using the developed algorithms. Nevertheless, we see that the identification of the one-way roads with the help of color marks is faulty, so that the algorithm may fail in some situations. On the other hand, errors from the proximity sensors (here used as touch sensors) and odometry errors showed no effect on the ability of the algorithm to solve the labyrinth.

VI. Conclusions

In this paper we considered labyrinths with one-way roads. First, we examined some properties of such labyrinths. With the help of these properties we have shown that it is possible to design algorithms that lead the robot to the exit. Further we believe that these properties may help us to examine other classes of algorithms. For example, we might consider the influence of one-way roads to BUG-like algorithms.

We developed two algorithms to solve every fair labyrinth with one-way roads and prove their correctness. We considered a robot with very limited abilities. However, we assume that the sensors are error-free. A future work might be to consider error-prone sensors (particularly the one-way identification) and determine in which ways the algorithms will be affected.

We have shown that it is possible to expand a well-known algorithm such as the Pledge algorithm to solve tasks which the original algorithm cannot solve, without loosing the ability to prove its correctness. Future work may concern to expand it to solve more complex, maybe real world environments.

References

[1] J. J. Leonard and H. F. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," 1991, pp. 1442–1447 vol.3.

[2] N. Rao, S. Kareti, W. Shi, and S. Iyenagar, "Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms," 1993.

[3] A. Sankaranarayanan and I. Masuda, "A new algorithm for robot curvefollowing amidst unknown obstacles, and a generalization of maze-searching." in *Proceedings of 1992 IEEE International Conference on Robotics and Automation*, 1992, pp. 2487–2494.

[4] A. Sankaranarayanan and M. Vidyasagar, "A new path planning algorithm for a point object amidst unknown obstacles in a plane." in *Proceedings of IEEE Conference on Robotics and Automation*, 1990, pp. 1930–1936.

[5] V. J. Lumelsky and A. A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment." in *IEEE transactions on Automatic control*, 1986, pp. 1058–1063.

[6] H. Abelson and A. diSessa, *Turtle Geometry*. MIT Press, Cambridge, MA, 1980.

[7] R. Klein, *Algorithmische Geometrie*. Springer, 2005.

[8] T. Kamphans and E. Langetepe, "The pledge algorithm reconsidered under errors in sensors and motion," in *Proc. of the 1th Workshop on Approximation and Online Algorithms, Lecture Notes Comput. Sci 2909, Springer*, 2003, pp. 165–178.

[9] B. Brüggemann, "Entkommen aus unbekannten Labyrinthen mit Einbahnstrassen," Master's thesis, Rheinische Friedrich Wilhelms Universität Bonn, 2006.

[10] B. Brüggemann, T. Kamphans, and E. Langetepe, "Leaving an unknown maze with one-way roads," in *Abstracts 23nd European Workshop Comput. Geom., Graz, Austria*, 2007, pp. 90–93.

[11] A. Hemmerling, "Navigation without perception of coordinates and distances," Berkeley, CA, Tech. Rep. TR-93-018, 1993.