*Advanced Control for Robotics and Autonomous Systems-Research Article*

INTERNATIONAL JOURNAL OF ADVANCED ROBOTIC SYSTEMS

*International Journal of Advanced Robotic Systems*
January-February 2019: 1–12
© The Author(s) 2019
DOI: 10.1177/1729881419826857
journals.sagepub.com/home/arx

# Improving the Hybrid A* method for a non-holonomic wheeled robot

**Dušan Nemec, Michal Gregor, Emília Bubeníková, Marián Hruboš and Rastislav Pirník**

**Abstract**

This article proposes and completely describes a modification of the Hybrid A* method used for navigation of a non-holonomic mobile wheeled robot. Our modification allows straightforward multi-criterial adjustment of the algorithm according to the desired behavior considering not only traveled distance but also time, changing of direction, stopping, going backwards while avoiding obstacles. The obstacle avoidance algorithm evaluates the danger of collision smoothly (not-binarily) using danger fields. Such behavior reflects human-like sensing of danger—the closer to the obstacle the robot is, the higher is the danger of collision. A modified uniform state expansion method has been used to cover the state space of the robot more uniformly providing the possibility of precise near-target navigation. A greed factor has been introduced to decrease the computational time and improve the real-time performance of the algorithm.

**Keywords**

Non-holnomic robot, heuristics, navigation, path-planning, obstacle avoidance

Date received: 24 October 2018; accepted: 4 January 2019

Topic: Mobile Robots and Multi-Robot Systems
Topic Editor: Andrey V Savkin
Associate Editor: Bin He

## Introduction

The navigation of the mobile robot is a dynamic control process during which the control algorithm computes actions which lead the mobile robot to the target. The position of the robot and the target can be defined by three degrees of freedom (3 DoF)—Cartesian coordinates $x$, $y$, and direction $\psi$. The non-holonomic robot has non-zero minimal turning radius $R_{\min}$—its 3 DoF are not fully controllable. To connect the current position and the target by a physically achievable trajectory, the robot cannot travel straight to the target due to its physical constraints. More constraints are introduced by obstacles. To take all these factors into account, it is possible to use a form of a Model Predictive Controller (MPC).[1] This type of the controller can also be used in many non-robotic applications (see e.g. Hoy et al.[2]). To intercept the paths of the moving obstacles (e.g. humans in the working area), Bayesian representation of the obstacles has been used.[3]

One of the great methods of navigation for wheeled mobile robots is a Vector Field Histogram (VFH) combined with the A* searching algorithm (abbreviated as VFH*). The method was originally proposed by Ulrich and Borenstein.[4] The concept of the method is described by following steps:

- The current position of the robot is expanded by admissible actions into new projected positions.
- Each new position is evaluated with respect to (w.r.t.) the target and the previous position. The

Department of Control and Information Systems, Faculty of Electrical Engineering, University of Žilina, Žilina, Slovak Republic

**Corresponding author:**
Marián Hruboš, Department of Control and Information Systems, Faculty of Electrical Engineering, University of Žilina, Univerzitná 8215/1, 01026 Žilina, Slovak Republic.
Email: marian.hrubos@fel.uniza.sk

position that is closer to the target and requires less maneuvering, has a lower cost. If the new position collides with an obstacle, the position is rejected. The total cost of the position is a sum of the previously spent cost and the estimated further cost.
- Positions are recursively explored up to a given depth or until the target is reached. The A* searching algorithm is used to achieve optimal behavior. Searching is optimal and complete if the estimated cost is always smaller or equal to the true cost.
- The first action along the found trajectory is physically applied to the robot and the whole process repeats. The navigation is local (does not consider the whole world but only limited surroundings) and real-time.

The original paper[4] proposed the following cost function for the first expansion

$$g(\boldsymbol{r}_1) = \mu_1 \cdot \Delta(\psi_1, \psi_t) + \mu_2 \cdot \Delta(\psi_1, \psi_0) + \mu_3 \cdot \Delta(\psi_1, \psi_{d,n-1}) \qquad (1)$$

and for deeper expansions

$$g(\boldsymbol{r}_i) = \lambda^{i-1}[\mu_1' \cdot \max(\Delta(\psi_i, \psi_t), \Delta(\psi_e, \psi_t)) + \mu_2' \cdot \Delta(\psi_i, \psi_0) + \mu_3' \cdot \Delta(\psi_i, \psi_{i-1})] \qquad (2)$$

where $i$ is the depth of the expansion, $\psi_i$ is the direction at position $\boldsymbol{r}_i$, $\psi_t$ is the direction to the target, $\psi_0$ is the current direction, $\psi_{d,n-1}$ is the previously decided direction, $\psi_e$ is the effective direction

$$\psi_e = \text{atan2}(y_i - y_{i-1}, x_i - x_{i-1}) \qquad (3)$$

The delta operator over directions is defined by formula

$$\Delta(\psi_a, \psi_b) = \min(|\psi_a - \psi_b|, |\psi_a - \psi_b - 2\pi|, |\psi_a - \psi_b + 2\pi|) \qquad (4)$$

A heuristic estimation of the future cost is

$$h(\boldsymbol{r}_i) = \lambda^i[\mu_1' \cdot \Delta(\psi_e, \psi_t) + \mu_2' \cdot \Delta(\psi_t, \psi_A) + \mu_3' \cdot \Delta(\psi_t, \psi_{i-1})] \qquad (5)$$

Coefficient $\mu_1$ increases orientation of the robot to the target, coefficients $\mu_2$ and $\mu_3$ increase the smoothness of the trajectory. Coefficient $\lambda$ is the discount factor and improves the overall behavior of the searching algorithm.[4]

The main drawbacks of the original VFH* algorithm are:

- The target is defined only by direction (as the target is in infinity) within the method.
- The original cost function does not consider the distance travelled by the robot.
- Each state used in prediction is expanded only to different directions, but not to different distances (see Figure 1). Therefore, the state space is not covered uniformly.

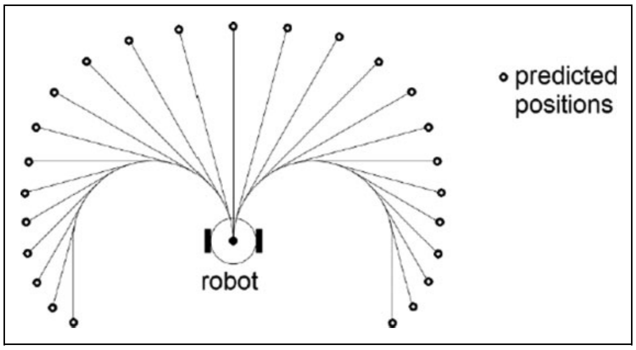

**Figure 1.** Exploration of the nearby robot's positions.

Other navigational methods are mainly based on grids. Chung and Huang described a navigation algorithm which represents the world as a 2-D grid of occupied or non-occupied cells. The occupancy grid can be obtained simultaneously from the readings of the robots' sensors.[5] Many different sensors can be used to detect the surrounding obstacles and the robot's own position (e.g. encoders, ultrasonic sensors, electronic compass, etc.).[6] Szabo[7] described a method for integration of the sensors' readings by an event representation. The final path obtained by A* searching contains only eight possible directions alongside the grid or diagonally.[8] Daniel et al. introduced the Theta* algorithm that also uses a grid, but applies an any-angle search algorithm, which allows connection of more distant grid vertices by a straight line (line of sight). The obtained trajectory is more smooth and efficient than only 8-directional trajectory.[9] Yap et al. proposed the Block A* method that utilizes a data base of local distances between blocks. The data base contains more than eight nearest blocks; therefore, it is more efficient than A* or Theta*.[10]

To overcome the non-holonomic nature of the planned grid trajectories (not smooth), the Stanford University team building robot Junior for DARPA Urban Challenge[11] developed algorithm named Hybrid A*. The algorithm uses continuous states within 4-D state space (two horizontal coordinates, orientation, and direction of motion—forward or reverse). The surroundings of the best node were explored in six defined directions (combinations of forward/backward throttle and left/straight/right steering), which results in six child states. Each child state is assigned to one horizontal grid cell. To improve the efficiency of the expansion, two different heuristic functions have been used to evaluate each state: the first considering non-holonomic nature of the robot and the second considering obstacles. It results in fewer nodes being required to be explored during navigation. The resultant path is smoothed by Conjugate Gradient filtering.[12]

## Uniform expansion method

Using the original version of VFH*, the local maneuvering of the robot to a nearby target defined by all three

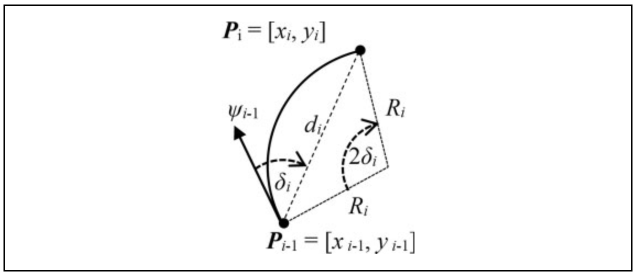

**Figure 2.** Approaching the nearby position in the *i*th expansion depth.

coordinates is not working perfectly. One of the reasons is that the expansion uses constant travel distance (the original histogram used in VFH considered only directions). If the target is closer than the expansion distance, the robot will oscillate. The Hybrid A* algorithm is a good starting point in solving the problem.

To cover the state space $\langle x, y, \psi \rangle$ more uniformly, we have to discretize the state space into a grid. The grid size corresponds to the tolerance of the whole navigation algorithm. The position of the robot, when snapped to the grid is

$$[x, y, \psi] = [n_x G_{xy}, n_y G_{xy}, n_\psi G_\psi] \tag{6}$$

where $G_{xy}$ is a translational grid cell size, $G_\psi$ is a rotational grid cell size, and $n_x$, $n_y$, $n_z$ are integers. Introducing the grid also enables fast searching for already visited positions within the A* searching algorithm, which avoids oscillations and loops during the search.

The exploration of the near positions around the currently best position is done reversely: first, we set the desired position and then we compute the required action to achieve that position. We assume that the robot's steering control does not need to be continuous (the curvature of the trajectory may change rapidly). The scenario is illustrated in Figure 2, note that direction (angle) $\psi_{i-1}$ is represented by a directional arrow.

The length and the direction of the distance vector $\boldsymbol{P}_i\boldsymbol{P}_{i-1}$ are (in that order)

$$d_i = |\boldsymbol{d}_i| = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{7}$$

$$\psi_{di} = \operatorname{atan2}(y_i - y_{i-1}, x_i - x_{i-1}) \tag{8}$$

The angle $\delta_i$ is equal to

$$\delta_i = (\psi_{di} - \psi_{i-1}) - 2\pi \left\| \frac{\psi_{di} - \psi_{i-1}}{2\pi} \right\| \tag{9}$$

where $\|x\|$ denotes the nearest integer to a given real number $x$. The distance $d_i$ is:

$$d_i = 2R_i \sin\delta_i \tag{10}$$

where $R_i$ is the turning radius. Then we obtain

$$R_i = \frac{d_i}{2\sin\delta_i} \tag{11}$$

If we assume that the vehicle has two control variables: throttle $T$ and steering $S$, both are within range $\langle -1, 1 \rangle$. Steering is then

$$S_i = \frac{R_{\min}}{R_i} = \frac{2R_{\min}\sin\delta_i}{d_i} \tag{12}$$

where $R_{\min}$ is the minimal turning radius of the robot. The length of the trajectory is

$$L_i = \begin{cases} d_i & \text{if } |\delta_i| < \varepsilon, \\ d_i \dfrac{|\delta_i|}{\sin|\delta_i|} & \text{if } \varepsilon \le |\delta_i| \le \dfrac{\pi}{2}, \\ -d_i \dfrac{(\pi - |\delta_i|)}{\sin(\pi - |\delta_i|)} & \text{if } \dfrac{\pi}{2} < |\delta_i| \le (\pi - \varepsilon), \\ -d_i & \text{if } |\delta_i| > (\pi - \varepsilon) \end{cases} \tag{13}$$

where $\varepsilon$ is the numerical precision of the data type used in the implementation (e.g. $\varepsilon = 10^{-6}$). A negative length $L_i$ means going backwards.

Within one step of the algorithm, it is reasonable to explore up to the maximal achievable distance, which corresponds to full throttle

$$L_{\max} = v_{\max}\Delta t \tag{14}$$

where $v_{\max}$ is the maximal speed of the robot and $\Delta t$ is the time period of one exploration step. The throttle is then

$$T_i = \frac{L_i}{L_{\max}} \tag{15}$$

The final direction of the robot after the turn is

$$\psi_i = \psi_{i-1} + \frac{L_i}{R_i} = \psi_{i-1} + \frac{2L_i\sin\delta_i}{d_i} \tag{16}$$

If the computed throttle $T_i$ or steering $S_i$ is outside the $\langle -1, 1 \rangle$ boundary, the point is unreachable. Figure 3 shows reachable positions within distance $L_{\max} = \frac{\pi}{2}R_{\min}$(quarter turn). The horizontal grid cell size was set to $G_{xy} = L_{\max}/10$.

To speed up the calculation, it is possible to precompute a set of possible actions $[T, S]$ that will later be used for exploration.

## *Distance between the robot and the obstacle*

Evaluation of collisions requires the distance between the robot's projection into given position $\boldsymbol{P}_i$ and the surrounding obstacles. In many cases, the robot and the obstacles have rectangular horizontal footprint. More complex shapes can be computed as a combination of rectangles. The problem of finding the distance to the nearest obstacle is therefore transformed to computing the distance between two arbitrary rotated rectangles. The distance between rectangles is the minimum of distances between all corners of one rectangle and sides of the other rectangle. Figure 4 illustrates the whole situation.

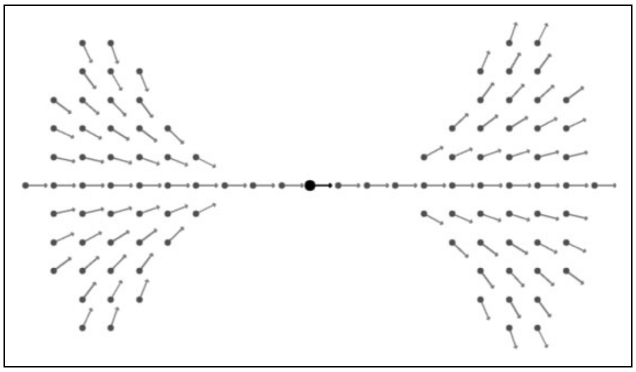

**Figure 3.** Reachable positions from the current position within one exploration step.

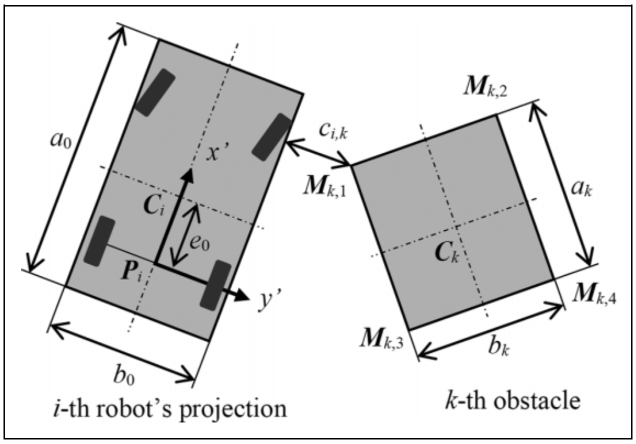

**Figure 4.** Minimal distance between the robot and the obstacle.

Position of the robot's origin is shifted from the rectangle's centrum $\boldsymbol{C}_i$ to the centrum of the non-controlled axle $\boldsymbol{P}_i$ by dislocation $e_0$. Dimensions of the robot are $a_0 \times b_0$, dimensions of the $k$th obstacle are $a_k \times b_k$. The position of the robot's geometrical centrum is

$$\boldsymbol{C}_i = \boldsymbol{P}_i + e_0 \cdot [\cos\psi_i \sin\psi_i]^{\mathrm{T}} \tag{17}$$

Positions of the obstacle's corners are

$$\boldsymbol{M}_{k,n} = \boldsymbol{C}_k + \frac{1}{2}\begin{bmatrix} \cos\psi_k & -\sin\psi_k \\ \sin\psi_k & \cos\psi_k \end{bmatrix} \cdot \begin{bmatrix} \pm a_k \\ \pm b_k \end{bmatrix} \tag{18}$$

where $\psi_k$ is the orientation (angle of rotation) of the obstacle and $n = 1, 2, \cdots 4$ is the index of the corner. The positions of the obstacle's corners in robot's local coordinate system $\langle x', y' \rangle$ are

$$\boldsymbol{M}'_{k,n} = \begin{bmatrix} x'_{k,n} \\ y'_{k,n} \end{bmatrix} = \begin{bmatrix} \cos\psi_i & \sin\psi_i \\ -\sin\psi_i & \cos\psi_i \end{bmatrix} \cdot (\boldsymbol{M}_{k,n} - \boldsymbol{P}_i) \tag{19}$$

The distances between the obstacle's corner and the robot are

$$d_{k,n} = \min(|x'_{k,n} - e_0| - a_0, |y'_{k,n}| - b_0) \tag{20}$$

Using similar approach, we obtain positions of the robot's corners $\boldsymbol{N}_{i,n}$

$$\boldsymbol{N}_{i,n} = \boldsymbol{C}_i + \frac{1}{2}\begin{bmatrix} \cos\psi_i & -\sin\psi_i \\ \sin\psi_i & \cos\psi_i \end{bmatrix} \cdot \begin{bmatrix} \pm a_i \\ \pm b_i \end{bmatrix} \tag{21}$$

The positions of the robot's corners w.r.t. the obstacle are

$$\boldsymbol{N}'_{i,n} = \begin{bmatrix} x'_{i,n} \\ y'_{i,n} \end{bmatrix} = \begin{bmatrix} \cos\psi_k & \sin\psi_k \\ -\sin\psi_k & \cos\psi_k \end{bmatrix} \cdot (\boldsymbol{N}_{i,n} - \boldsymbol{C}_k) \tag{22}$$

The distances $d_{i,n}$ between the robot's corners and the obstacle are

$$d_{i,n} = \min(|x'_{i,n}| - a_0, |y'_{i,n}| - b_0) \tag{23}$$

The distance between the $i$th robot's projection and the $k$th obstacle is the minimum of all eight distances

$$c_{i,k} = \min(d_{k,1}, d_{k,2}, d_{k,3}, d_{k,4}, d_{i,1}, d_{i,2}, d_{i,3}, d_{i,4}) \tag{24}$$

Clearance $c_i$ is the minimum of $c_{i,k}$ across all obstacles $k$.

If the obstacle or robot is significantly larger than the grid cell, the proposed method provides significant speedup of the clearance calculation compared to widely used computation of clearance using distances between occupied cells. It also allows more precise maneuvering, since each object may occupy its border cells only partially and the robot and the obstacle may occupy the same cell safely without collision. If the true horizontal footprint of the robot is not rectangular, we use minimal area bounding rectangle.

The reliability and safety of the proposed method depends on the way how the position of the obstacle is obtained. The obstacle may be detected by the robots' local sensors, or, in case of multiple robots, the information may be provided by communication among them. Then, the most critical factor is the safety of the communication, which has to be evaluated separately.[13]

## *The cost of single exploration step*

To find the optimal trajectory, the A* method expands a non-opened node with the smallest (best) total cost during each iteration. The total cost $f_i$ is a sum of the cost $g_i$ spent by reaching the $i$th node and the estimated future cost $h_i$ that has to be spent to reach the target. The spent cost $g_i$ increases with distance, changing of the direction, changing of the speed, or going backwards.

*The cost of distance.* The cost of distance is proportional to the length of the trajectory

$$g_{Li} = K_L \cdot |L_i| \qquad [K_L] = \mathrm{m}^{-1} \tag{25}$$

where $K_L$ is the (constant) cost per meter.

*The cost of changing the direction.* If it is required to maintain a smooth trajectory, the cost of changing the heading may be introduced. The cost is proportional to the arc angle

$$g_{Si} = K_S \cdot |\psi_i - \psi_{i-1}| = K_S \cdot \left| \frac{2L_i \sin\delta_i}{d_i} \right| \qquad [K_S] = \text{rad}^{-1} \tag{26}$$

where $K_S$ is the (constant) cost per radian.

*The cost of stopping the robot.* To avoid too many reversal points (point where the vehicle has to stop and change the polarity of throttle), we introduce the cost of stopping

$$g_{Ti} = K_T \cdot \text{neg}(T_{i-1} \cdot T_i) \qquad [K_T] = 1 \tag{27}$$

where $T_{i-1}$ is the previous throttle, $K_T$ is a fixed cost per one reversal point and neg$(x)$ is the following function

$$\text{neg}(x) = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{if } x \geq 0 \end{cases} \tag{28}$$

*The cost of going backwards.* For certain mobile robots, it is not convenient to go backwards (e.g. due to worse maneuvering abilities). The cost of going backwards is

$$g_{bi} = -K_b \cdot \text{neg}(L_i) \cdot L_i \qquad [K_b] = \text{m}^{-1} \tag{29}$$

where $K_b$ is the (constant) cost per one meter of going backwards.

*The cost of danger.* The most crucial part of the navigation is avoiding collisions with obstacles. Chen and Zhang[14] proposed a method for estimation of the distance between moving non-holonomic robots. Since the estimation of the position is not absolutely precise, collision detection should not be binary (the real position of the robot could collide with an obstacle while the estimated position does not collide). Therefore, we have introduced a danger field, which is derived from the potential field navigation method. The danger field defines a dimensionless danger of collision as a function of the robot's position $[x,\ y,\ \psi]$. The function is as follows

$$D_i = \left[ \frac{c_1 - c_0}{\max(c_i - c_0, \varepsilon)} \right]^{\kappa} \tag{30}$$

where $c_i$ is a clearance (minimal distance) between the robot projected to position $\boldsymbol{P}_i$ and the surrounding obstacles, $c_0$ is a safety border in meters around each obstacle, $c_1$ is a normalization constant which corresponds to the distance at which the danger is equal to 1. Coefficient $\kappa$ is the steepness of the danger function (we use $\kappa = 2$) and $\varepsilon$ is a very small positive number (prevents division by zero). If any obstacle moves, the danger field function also changes due to the changes of clearance.

The cost of danger is then

$$g_{Di} = K_D \cdot D_i \qquad [K_D] = 1 \tag{31}$$

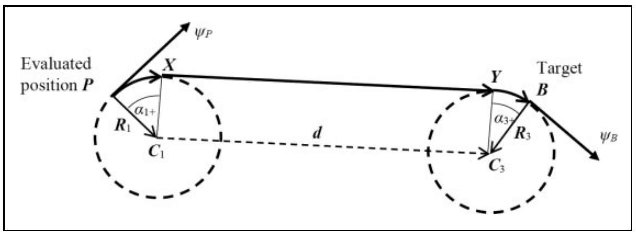

**Figure 5.** Trajectory pattern, forward trajectory.

where $K_D$ is the weight (dimensionless constant) of the danger. If the $K_D$ is higher, the robot is "more afraid" of collision.

*The cost of time.* The same trajectory can be achieved by a smaller number of large steps (higher speed) or by a larger number of small steps (small speed). It is desirable to select the faster trajectory, so we have introduced the cost of time. Each time step has the cost proportional to parameter $K_t$

$$g_{ti} = K_t \Delta t \qquad [K_t] = \text{s}^{-1} \tag{32}$$

The overall estimated spent cost is then given by the sum of all partial costs and the cost $g_{i-1}$ spent for reaching the position $P_{i-1}$

$$g_i = g_{i-1} + g_{Li} + g_{Si} + g_{Ti} + g_{bi} + g_{Di} + g_{ti} \tag{33}$$

## Estimation of future cost

In this chapter, we will solve the problem of the minimal distance which has to be run by a non-holonomic wheeled robot to move from one position to another. The space of all possible trajectories is too large to search and contains discontinuities; therefore, it is necessary to choose a certain pattern of the trajectory.

### Possible trajectories

We have chosen the trajectory pattern shown in Figure 5 which consists of an initial arc $\boldsymbol{PX}$, then followed by a straight line $\boldsymbol{XY}$, and finished by another arc $\boldsymbol{YB}$. To keep the trajectory length minimal, the radius of the initial and the final arc turns is $R_{\min}$. Such trajectory has been proved by Dubins to be the shortest.[15] These trajectories are commonly referred to as Dubins paths. Due to the motion planning purposes, also reversing should be considered. By modifying the Dubins principle, Reeds and Shepp[16] introduced a more complex method using at most five segments of arcs and/or straight lines. This article proposes the usage of a three-segment curve but also considers reversing.

First, we define process variables (we omit the index $i$ of the position $\boldsymbol{P}_i$ and of other variables):

- $\boldsymbol{P} = [x_P, y_P]$ the evaluated horizontal position,
- $\psi_P$ the evaluated direction,
- $\boldsymbol{B} = [x_B, y_B]$ the target horizontal position,

- $\psi_B$ the target direction,
- $\boldsymbol{R}_1$ the radial vector of the initial turn (perpendicular to the $\psi_P$),
- $\boldsymbol{R}_3$ the radial vector of the final turn (perpendicular to the target $\psi_B$),
- $\boldsymbol{d}$ the distance vector between the centrums of the turns,
- $\psi_d$ the direction of vector $\boldsymbol{D}$,
- $L_1$ the length of the initial arc $\boldsymbol{PX}$,
- $L_2$ the length of the straight line $\boldsymbol{XY}$,
- $L_3$ the length of the final arc $\boldsymbol{YB}$,
- $\alpha_1$ the initial arc angle,
- $\alpha_3$ the final arc angle,
- $s_1$ the polarity of the initial arc ($1$ = counterclockwise, $-1$ = clockwise),
- $s_3$ the polarity of the final arc ($1$ = counterclockwise, $-1$ = clockwise).

According to Figure 5, the centrum of the initial arc is

$$\boldsymbol{C}_1 = \boldsymbol{P} + s_1\boldsymbol{R}_1 = P + s_1 R_{\min}[-\sin\psi_P, \cos\psi_P] \tag{34}$$

analogically, the centrum of the final turn is

$$\boldsymbol{C}_3 = \boldsymbol{B} + s_3 R_{\min}[-\sin\psi_B, \cos\psi_B] \tag{35}$$

The distance vector between the centrum of the turns is then

$$\boldsymbol{d} = \boldsymbol{C}_3 - \boldsymbol{C}_1 = [d_x, d_y] \tag{36}$$

and its direction is

$$\psi_d = \text{atan2}(d_y, d_x) \tag{37}$$

In the following sections, we will compute all possible trajectories matching the given trajectory pattern.

*The same polarity of turns.* First, we will discuss the case when both turns have the same polarity

$$s_1 = s_3 \tag{38}$$

The arc angles are then (according to Figure 5)

$$\alpha_{1+} = s_1(\psi_d - \psi_P) \tag{39}$$

$$\alpha_{3+} = s_3(\psi_B - \psi_d) \tag{40}$$

Positive sign in indices denotes that the straight segment of the trajectory is traveled forwardly. Polarities $s_{-1}$ and $s_3$ were introduced, because we want to avoid negative angles. To keep the arc angles within the interval $[0, 2\pi)$, we remove the period

$$\alpha_k \leftarrow \alpha_k - 2\pi\left\lfloor\frac{\alpha_k}{2\pi}\right\rfloor \tag{41}$$

where $\lfloor x \rfloor$ denotes the nearest smaller integer to a given number $x$. A similar relation is valid for all used angles. Positions $\boldsymbol{X}$ and $\boldsymbol{Y}$ can be reached not just by turning by angles $\alpha_{1+}$ and $\alpha_{3+}$, respectively, but also by complementary angles $(2\pi - \alpha_{1+})$ and $(2\pi - \alpha_{3+})$ in reverse manner (displayed by the dashed line).

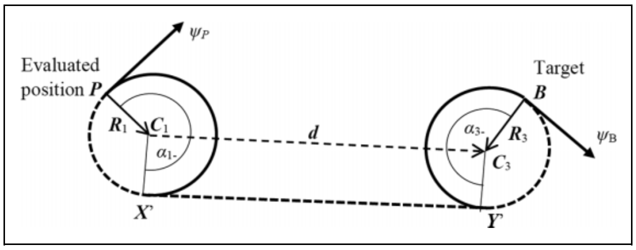

**Figure 6.** The same polarity of turns, reverse trajectory.

There is also a possibility of travelling the straight line reversely (see Figure 6).

The arc angles in the reverse case are

$$\alpha_{1-} = s_1(\psi_d + \pi - \psi_P) \tag{42}$$

$$\alpha_{3-} = s_3(\psi_B - \psi_d - \pi) \tag{43}$$

Operation (41) has to be applied to both angles (42) and (43) to keep them within the interval $[0, 2\pi)$. The length of the straight element is

$$d = \sqrt{d_x{}^2 + d_y{}^2} \tag{44}$$

To denote all possible trajectories in a compact way, we will denote each trajectory as a vector $[L_1, L_2, L_3]$. The negative sign will mean that the robot should travel backwards. Using this notation, the possible trajectories for the case of the same polarity of turns are

$$\begin{array}{rcl}
[R_{\min}\alpha_{1+}, & d, & R_{\min}\alpha_{3+}] \\
[-R_{\min}(2\pi - \alpha_{1+}), & d, & R_{\min}\alpha_{3+}] \\
[R_{\min}\alpha_{1+}, & d, & -R_{\min}(2\pi - \alpha_{3+})] \\
[-R_{\min}(2\pi - \alpha_{1+}), & d, & -R_{\min}(2\pi - \alpha_{3+})] \\
[R_{\min}\alpha_{1-}, & -d, & R_{\min}\alpha_{3-}] \\
[-R_{\min}(2\pi - \alpha_{1-}), & -d, & R_{\min}\alpha_{3-}] \\
[R_{\min}\alpha_{1-}, & -d, & -R_{\min}(2\pi - \alpha_{3-})] \\
[-R_{\min}(2\pi - \alpha_{1-}), & -d, & -R_{\min}(2\pi - \alpha_{3-})]
\end{array} \tag{45}$$

*The opposite polarity of turns.* The case when initial and final turns have opposite polarities is shown in Figure 7, the reverse variant is in Figure 8.

The arc angles are

$$\beta_{1+} = s_1(\psi_D - \psi_P) + \delta \tag{46}$$

$$\beta_{3+} = s_3(\psi_B - \psi_D) + \delta \tag{47}$$

where the angle $\delta$ (according to the figure) is

$$\delta = \text{asin}\left(\frac{R_{\min}}{d/2}\right) = \text{asin}\left(\frac{2R_{\min}}{d}\right) \tag{48}$$

The complementary arc angles (dashed arcs) are computed in the same way as in the previous case. From equation (48), it is clear that the trajectory exists only if

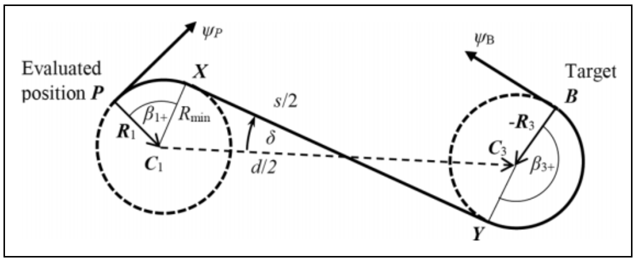

**Figure 7.** Opposite direction of turns, forward trajectory.

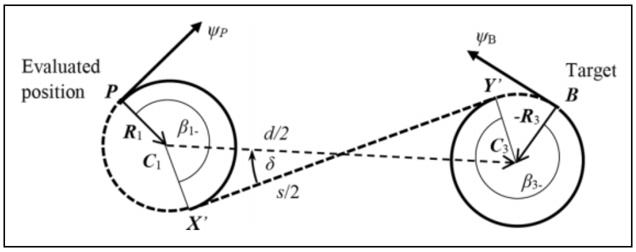

**Figure 8.** Opposite direction of turns, reverse trajectory.

$$R_{\min} < \frac{d}{2} \tag{49}$$

The length of the straight section of the trajectory is

$$s = \sqrt{{d_x}^2 + {d_y}^2 - 4{R_{\min}}^2} \tag{50}$$

Like in the case when both circles have the same polarity, there is also the possibility to travel reversely (see Figure 8).

The corresponding arc angles are

$$\beta_{1-} = s_1(\psi_D + \pi - \psi_P) - \delta \tag{51}$$

$$\beta_{3-} = s_3(\psi_B - \psi_d - \pi) - \delta \tag{52}$$

After removing the period from all arc angles using equation (41), we obtain another set of possible trajectories (valid for the case of opposite arc polarities)

$$\begin{array}{rcl}
[R_{\min}\beta_{1+}, & s, & R_{\min}\beta_{3+}] \\
[-R_{\min}(2\pi - \beta_{1+}), & s, & R_{\min}\beta_{3+}] \\
[R_{\min}\beta_{1+}, & s, & -R_{\min}(2\pi - \beta_{3+})] \\
[-R_{\min}(2\pi - \beta_{1+}), & s, & -R_{\min}(2\pi - \beta_{3+})] \\
[R_{\min}\beta_{1-}, & -s, & R_{\min}\beta_{3-}] \\
[-R_{\min}(2\pi - \beta_{1-}), & -s, & R_{\min}\beta_{3-}] \\
[R_{\min}\beta_{1-}, & -s, & -R_{\min}(2\pi - \beta_{3-})] \\
[-R_{\min}(2\pi - \beta_{1-}), & -s, & -R_{\min}(2\pi - \beta_{3-})]
\end{array} \tag{53}$$

## Computing the minimal cost of travelling to the target

There are 4 possible combinations of arc polarities

$$[s_1, s_3] \in \{[-1, -1], [-1, 1], [1, -1], [1, 1]\} \tag{54}$$

For each combination, we obtain eight possible trajectories (using equation (45) for the same polarities and equation (53) for the opposite polarities). The full set contains 4 × 8 = 32 possible trajectories matching the defined pattern. Each trajectory has its cost, which reflects the requirements of the application. Generally, the cost increases with the distance, changing of the heading, stopping the robot, or going backwards. Note that we do not evaluate danger (since it must be evaluated at each point of the trajectory). The formulas for predicted future costs are similar to the formulas for the spent cost (equations (25) to (33))

$$h_{Li} = K_L(|L_{1i}| + |L_{2i}| + |L_{3i}|) \tag{55}$$

$$h_{Si} = K_S \frac{(|L_{1i}| + |L_{3i}|)}{R_{\min}} \tag{56}$$

$$h_{Ti} = K_T[\text{neg}(T_i \cdot L_{1i}) + \text{neg}(L_{1i} \cdot L_{2i}) + \text{neg}(L_{2i} \cdot L_{3i})] \tag{57}$$

$$h_{bi} = K_b[\text{neg}(L_{1i}) \cdot L_{1i} + \text{neg}(L_{2i}) \cdot L_{2i} + \text{neg}(L_{3i}) \cdot L_{3i}] \tag{58}$$

$$h_{ti} = K_t \frac{(|L_{1i}| + |L_{2i}| + |L_{3i}|)}{v_{\max}} \tag{59}$$

where $T_i$ is the throttle which was used to reach position $P_i$ (corresponds to the forward speed of the robot). The cost of time is computed as if the speed was maximal during the whole trajectory (minimal time). The overall estimated future cost is then given by the sum of all costs

$$h_i = h_{di} + h_{Si} + h_{Ti} + h_{bi} + h_{ti} \tag{60}$$

The estimated future cost is computed for each of the 32 possible trajectories and the best trajectory is selected.

## The greed factor

If the original A* searching algorithm is used, the total cost of any predicted position $P_i$ is

$$f_i = g_i + h_i \tag{61}$$

The navigation algorithm finds the optimal trajectory but it examines too many nodes which results in poor real-time performance. To speed up the calculation and searching, we have introduced the *greed factor* $\gamma$. It sacrifices a small portion of the optimal (minimal cost) to decrease the number of nodes required to be expanded during search. The modified total cost function replaces equation (61)

$$f_i = (1 - \gamma)g_i + h_i \tag{62}$$

When the greed factor is equal to 1, the searching does not consider the spent cost $g_i$ that results in greedy search.

Each predicted state of the robot will be represented by the structure *Node*:

```
structure Node
  . [$x$, $y$, $\psi$] -- position
  . [$T$, $S$] -- action
  . parent -- previous node
  . $g$ -- spent cost
  . $h$ -- estimated future cost
  . $f$ -- total cost
end
```

The cost spent from the starting state into predicted state is computed using function *spent_cost* shown in Algorithm 1: The estimated cost from predicted state into next state is computed by the function *future_cost* described in Algorithm 2:

**Algorithm 1.** function spent_cost (*node*, *obstacles*)

```
Algorithm 1: function spent_cost(node, obstacles)
  $L_i$ ← node.T * $v_{max}$ * $\Delta t$
  $\psi_i$ ← node.$\psi$
  $\psi_{i-1}$ ← node.parent.$\psi$
  $T_i$ ← node.T
  $T_{i-1}$ ← node.parent.T

  compute $D_i$ from node.[$x$, $y$, $\psi$], obstacles    -- eq. (30)
  compute $g_L$ from $L_i$, $K_L$                           -- eq. (25)
  compute $g_S$ from $\psi_i$, $\psi_{i-1}$, $K_S$                   -- eq. (26)
  compute $g_T$ from $T_i$, $T_{i-1}$, $K_T$                   -- eq. (27)
  compute $g_b$ from $L_i$, $K_b$                           -- eq. (29)
  compute $g_D$ from $D_i$, $K_D$                           -- eq. (31)
  compute $g_t$ from $\Delta t$, $K_t$                          -- eq. (32)
  return (node.parent.g + $g_L$ + $g_S$ + $g_T$ + $g_b$ + $g_D$ + $g_t$)
```

**Algorithm 2.** function future_cost (*node*, *target*)

```
Algorithm 2: function future_cost(node, target)
  trajectories ← empty list
  $P$ ← node.[$x$, $y$]
  $B$ ← target.[$x$, $y$]
  $\psi_P$ ← node.$\psi$
  $\psi_B$ ← target.$\psi$

  for $s_1$ from {-1, 1}
      for $s_3$ from {-1, 1}
          compute $d_x$, $d_y$ from $s_1$, $s_3$, $P$, $B$             -- eq. (34)-(36)
          compute $\psi_d$ from $d_x$, $d_y$                      -- eq. (37)

          if $s_1$ = $s_3$ then
              -- same polarity of turns
              compute $\alpha_{1+}$, $\alpha_{3+}$ from $\psi_P$, $\psi_d$, $\psi_B$         -- eq. (39)-(41)
              compute $\alpha_{1-}$, $\alpha_{3-}$ from $\psi_P$, $\psi_d$, $\psi_B$         -- eq. (42), (43), (41)
              compute all [$L_1$, $L_2$, $L_3$] from $\alpha_{1+}$, $\alpha_{3+}$, $\alpha_{1-}$, $\alpha_{3-}$, $d_x$, $d_y$  -- eq.
(44), (45)
         else
              -- opposite polarity of turns
              compute $\beta_{1+}$, $\beta_{3+}$ from $\psi_P$, $\psi_d$, $\psi_B$         -- eq. (46)-(48), (41)
              compute $\beta_{1-}$, $\beta_{3-}$ from $\psi_P$, $\psi_d$, $\psi_B$         -- eq. (51), (52), (41)
              compute all [$L_1$, $L_2$, $L_3$] from $\beta_{1+}$, $\beta_{3+}$, $\beta_{1-}$, $\beta_{3-}$, $d_x$, $d_y$   -- eq.
(50), (53)
         end if
         insert all [$L_1$, $L_2$, $L_3$] into trajectories
      end for
  end for

  best ← Inf
  for each [$L_1$, $L_2$, $L_3$]  in trajectories
      compute $h$ from $L_1$, $L_2$, $L_3$, node.T                -- eq. (55)-(60)
      if $h$ < best then
          best ← $h$
      end if
  end for
  return best
```

**Algorithm 3.** function expand(*node, target, obstacles*)

```
Algorithm 3: function expand(node, target, obstacles)
    compute actions from node.ψ    -- eq. (7)-(15) - may be precomputed
    for each action in actions
         pos ← predict(node.[x, y, ψ], action, Δt)    -- predict the new
position

        child ← new Node
        child.[x, y, ψ] ← pos
        child.[T, S] ← action
        child.parent ← node

        if (explored not contains child) then
            -- node was not discovered before
            child.g ← spent_cost(child, obstacles)
            child.h ← future_cost(child, target)
            child.f  ← (1 - greed) * child.g + child.h
            insert child into opened
            insert child into explored
        end if
    end for
```

**Algorithm 4.** function navigate (*from, target, obstacles, $T_0$, $S_0$*)

```
Algorithm 4: function navigate(from, target, obstacles, T0, S0)
  opened ← empty list
  explored ← empty list
  actions ← empty list

  start.pos ← from
  start.parent ← none
  start.action ← [T0, S0]
  start.g ← 0  -- no cost is spent in start node
  start.h ← future_cost(start, target)
  start.f ← (1 - greed) * start.g + start.h
  insert start into opened

  while opened not empty
     best ← node from opened with lowest f
     remove best from opened

     if best.pos = target then
        -- found the solution, reconstruct the path
        repeat
           insert best.action into actions front
           best ← best.parent
        until best is none
        return actions

     else
        -- solution not found yet
        expand(best, target, obstacles)
     end if
  end while

  return actions   -- empty list
```

Exploration of the child states is accomplished by the function *expand* shown in Algorithm 3:

Finally, the function *navigate* implements A* searching with greed factor (see Algorithm 4):

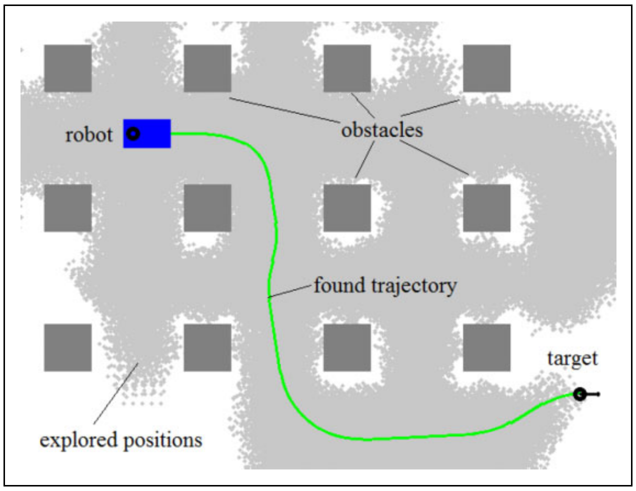

**Figure 9.** The simulated world and the optimal trajectory found (300,260 nodes has been explored).

## Experimental results

The proposed system has many adjustable parameters. To obtain results, that are comparable across experiments, we have used simulation in the simulated world (see Figure 9). The parameters of the robot were as follows: $v_{\max} = 100$ pixels s$^{-1}$, $R_{\min} = 50$ pixels, the tolerance of reaching the target (also the grid cell size): 10 pixels/5°. The resolution of the visualization was set to 640 × 480 pixels. The simulation was implemented in the OpenCV framework using the C++ programming language. The algorithm was evaluated on a standard PC with double-core Intel Pentium G870 3.10 GHz.

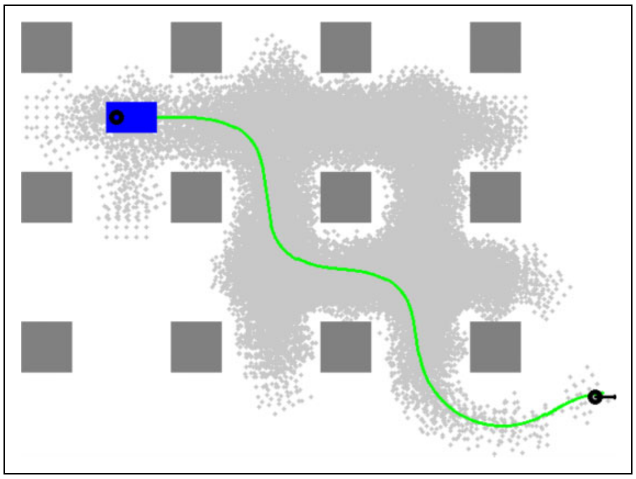

**Figure 10.** Map of danger.

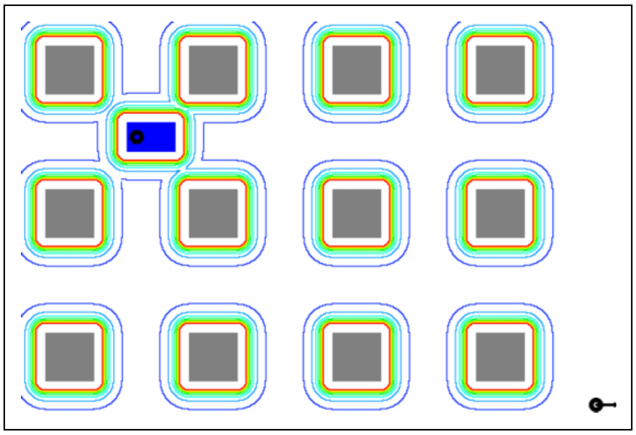

**Figure 11.** A suboptimal path computed using a greed factor $\gamma = 0.4$ (only 49,240 nodes has been explored).

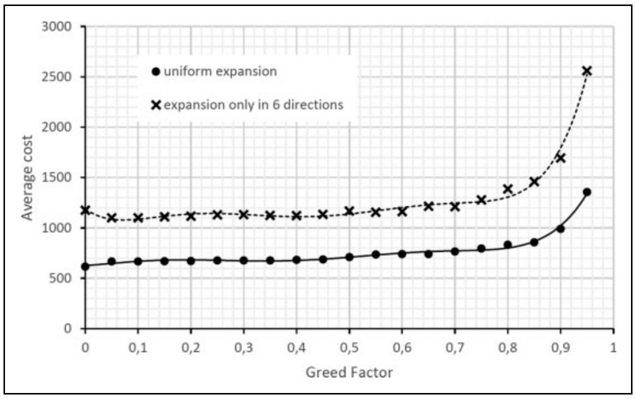

**Figure 12.** Average cost of the trajectory versus greed factor.

The world contains 12 static obstacles which makes it difficult for the robot to choose the optimal trajectory (many trajectories have the same cost).

We have evaluated the cost of the found trajectory, the computation time, and the count of explored nodes as a function of the greed factor. All the experiments were conducted with the same pseudorandom set of 1000 targets and 20 different settings of the greed factor (20,000 experiments). The cost constants which describe the properties of the optimal trajectory were set to the following values: the prediction step $\Delta t = 480$ ms, the cost of distance $K_L = 1$/pixel, the cost of steering $K_S = 20$ rad$^{-1}$, the cost of stopping $K_T = 100$, the cost of going backwards $K_b = 0.3$, the cost of danger $K_D = 200$, the cost of time $K_t = 1$ s$^{-1}$. The parameters of the danger function were as follows: the safety border $c_0 = 5$ pixels, the unit distance $c_1 = 15$ pixels, the exponent $\kappa = 2$ (Figure 10 shows a map of danger around the obstacles and the robot itself).

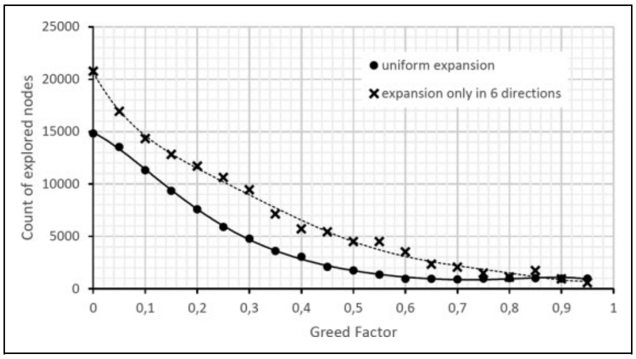

**Figure 13.** Average count of explored nodes versus greed factor.

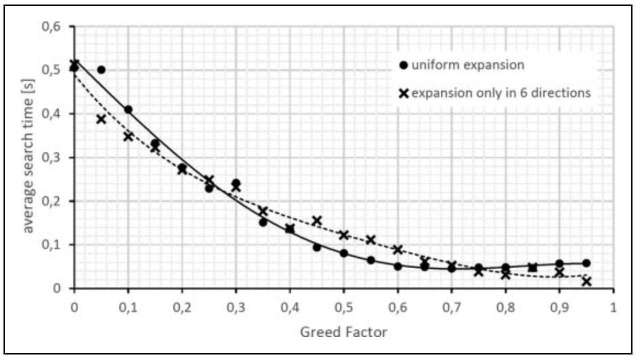

**Figure 14.** Average computation time versus greed factor.

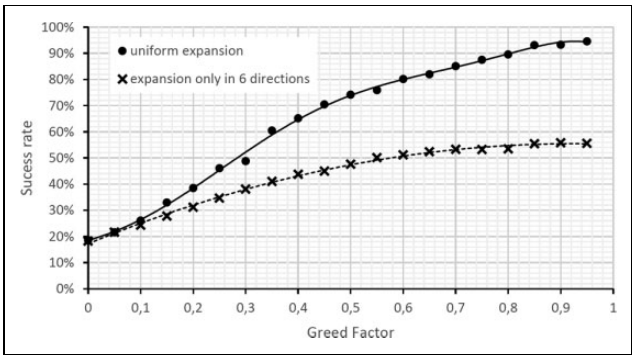

**Figure 15.** Probability of finding the trajectory within timeout (1 s) versus greed factor.

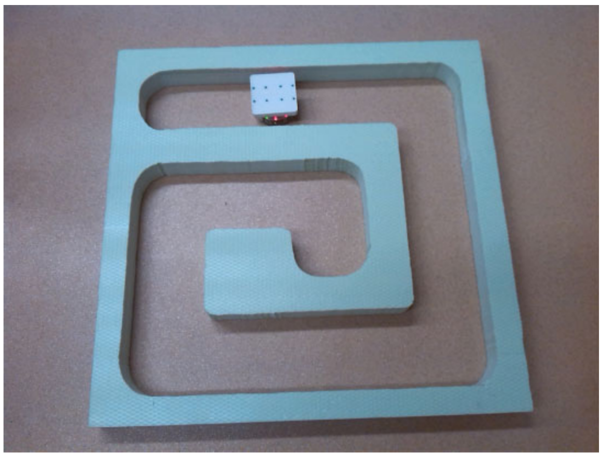
**Figure 16.** Testing environment with e-puck robot.

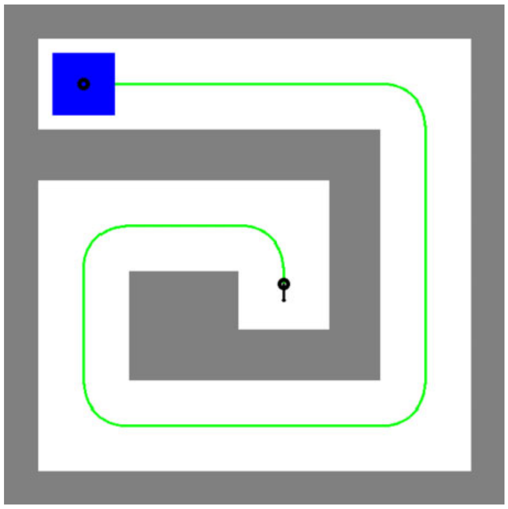
**Figure 17.** Designed trajectory.

To limit the simulation time, each experiment was limited to 1 s of computational time. If the algorithm did not find the solution within the given time limit, it was considered as unsuccessful. Such targets were removed from the set of targets. The remaining experiments were averaged for the same setting of the greed factor across all targets.

To demonstrate the importance of the uniform expansion method, we compare its performance with the state expansion in six directions only, which was proposed by the original Hybrid A* method.

As predicted, a higher greed factor allows the method to find the trajectory faster but the obtained trajectory is slightly suboptimal (compare Figures 9 and 11). Figure 12 shows the relation between the average cost per one target and the greed factor. With the greed factor from the range of 0 to 0.8, the cost of the trajectory is increasing only slightly (the greed factor $\gamma = 0.8$ caused an average increase of the cost by 35%). Further increase of the greed factor causes rapid increase of the cost (highly suboptimal behavior).

Figure 13 shows the average count of nodes which had to be explored to find the solution. With the greed factor from 0 to 0.3, the average count of nodes decreases almost linearly, and for a greed factor above 0.6, it is almost constant.

Computational time is closely related to the count of explored nodes, therefore the relation in Figure 14 is similar to the one in Figure 13. A higher greed factor means lower computational time. For a greed factor above 0.5, the average computational time was below 100 ms.

Since we require the algorithm to operate in real-time, the computation time was limited to 1 s. Figure 15 shows the probability of finding the trajectory within the given time limit. Without the greed factor, solutions for only 20% of the targets were found on time. For a greed factor above 0.5, more than 70% of the solutions were found on time. Note that using only six directions for expansion significantly increases the count of explored nodes, thus finding the solution times out in many cases. Figure 14 shows average of successful searches, therefore, it does not reflect poor probability of finding the solution in case of the six-directional expansion.

The algorithm has been evaluated in real world using e-puck robot (e-puck is a small mobile robot developed by GCtronic).[17,18] The testing environment with the e-puck robot inside can be seen in Figure 16. The environment has been modeled also in the control program (see Figure 17). The robot accomplished to pass the proposed trajectory while its position has been estimated using onboard odometers.

## Conclusion

Our proposed method improves the evaluation method of the Hybrid A* algorithm. First, we have introduced a uniform state expansion method which improves searching speed and decreases count of nodes needed to be explored. Then, we have modified and simplified the heuristic method used to estimate the future cost required to reach a given target. The navigation algorithm considers not only the length of the trajectory but also reversing, going backwards, changing the direction, the danger of collision, and the time of travel. Each feature is penalized by a separate parameter which allows simple adjustment of the behavior of the algorithm according to the requirements of any application. The evaluation of danger is not binary as used in many implementations of Hybrid A* (e.g. studies by Kurzer[1,19]), but smooth, which reflects the limited precision of the robot's localization system.

To speed up the real-time computations, we have used the greed factor which allows us to set the right balance between the required computational power and the cost of the projected trajectory. Our experiments show that a greed factor around 0.5 decreases the cost of the obtained trajectory only by 16% but decreases the computational time 5 times, which greatly improves the real-time performance of the algorithm.

## Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

## Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work has been supported by the Educational Grant Agency of the Slovak Republic KEGA, within the projects 014ŽU-4/2018 and 016ŽU-4/2018.

## ORCID iD

Marián Hruboš https://orcid.org/0000-0002-3871-1145

## References

1. Kurzer K. Hybrid A* Path Planner for the KTH Research Concept Vehicle, 2015. https://github.com/karlkurzer/path_planner (accessed 29 May 2018). GitHub.
2. Hoy M, Matvev A, and Savkin A. Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. *Robotica* 2015; 33(3): 463–497.
3. Hrbček J and Šimák V. Implementation of multi-dimensional model predictive control for critical process with stochastic behavior. In: Zheng T (ed) *Advanced model predictive control*. Rijeka, Croatia: IntechOpen, 2011, p. 19.
4. Ulrich I and Borenstein J. VFH*: local obstacle avoidance with look-ahead verification. In: *Proceedings 2000 ICRA. Millennium conference. IEEE International conference on robotics and automation. Symposia Proceedings*, 24–28 April 2000, San Francisco, CA, USA, pp. 2505–2511. San Francisco, CA, USA: IEEE.
5. Chung SY and Huang HP. Predictive navigation by understanding human motion patterns. *Int J Adv Robot Syst* 2011; 8(1): 13.
6. Gomez C, Hernandez AC, Crespo J, et al. A topological navigation system for indoor environments based on perception events. *Int J Adv Robot Syst* 2017; 14(1): 12.
7. Szabo R. Topological navigation of simulated robots using occupancy grid. *Int J Adv Robot Syst* 2004; 1(4): 6.
8. Choset H, Lynch K, Hutchinson S, et al. *Principles of robot motion: theory, algorithms, and implementations*. Cambridge Center, Cambridge, MA: MIT Press, 2005.
9. Daniel K, Nash A, Koenig S, et al. Theta*: any-angle path planning on grids. *J Artif Int Res* 2010; 39: 533–579.
10. Yap P, Burch N, Holte R, et al. Block A*: database-driven search with applications in any-angle path-planning. In: *AAAI'11 proceedings of the twenty-fifth AAAI conference on artificial intelligence*, 07–11 August 2011, San Francisco, California. pp. 120–125. San Francisco, California: AAAI Press.
11. Buehler M, Iagnemma K, and Singh S (eds). Junior: the Stanford entry in the urban challenge. In: *The 2005 DARPA grand challenge: the great robot race*. Germany: Springer-Verlag Berlin Heidelberg, 2006, p. 31. DOI: 10.1002/rob.20258.
12. Dolgov D, Thrun S, Montemerlo M, et al. Path planning for autonomous vehicles in unknown semi-structured environments. *Int J Robot Res* 2010; 29(5): 485–501.
13. Franeková M and Rástočný K. Safety evaluation of fail-safe fieldbus in safety related control system. *J Elect Eng* 2011; 61(6): 350–356.
14. Chen W and Zhang T. An indoor mobile robot navigation technique using odometry and electronic compass. *Int J Adv Robot Syst* 2017; 14(3): 15.
15. Dubins LE. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *Am J Math* 1957; 79(3): 497–516.
16. Reeds JA and Shepp LA. Optimal paths for a car that goes both forwards and backwards. *Pacific J Math* 1990; 145(2): 367–393.
17. Bernabeu EJ, Valera A, and Gomez-Moreno J. Distance computation between non-holonomic motions with constant accelerations. *Int J Adv Robot Syst* 2013; 10(9): 15.
18. Nolfi S and Mirolli M (eds) Evolution of communication and language in embodied agents. New York: Springer, 2009, p. 4.
19. Kurzer K. *Path planning in unstructured environments: a real-time Hybrid A* implementation for fast and deterministic path generation for the KTH research concept vehicle [thesis]*. School of Industrial Engineering and Management (ITM), 2016, p. 63. http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1057261&dswid=-1770 (accessed 27 May 2018).