Planning the Waypoint-Following Task for a Unicycle-like Robot in Cluttered Environments

Submitted: 15th October 2014; accepted: 8th January 2015

Tomasz Gawron, Maciej M. Michalek

DOI: 10.14313/JAMRIS_1-2015/10

Abstract:
The paper presents a two-stage, global planning algorithm for the waypoint-following task realized by a unicycle-like robot in a cluttered environment. It assumes motion execution with the VFO (Vector Field Orientation) controller. The planner is a result of a controller-driven design process and exploits particular properties of the VFO controller. Emphasis has been put on the plan safety in the sense of maximizing the distance from the obstacles. In the first stage of planning, an A-like pathfinding algorithm is used to find a safe geometric plan (i.e. a polyline) in a two-dimensional occupancy grid. During the second stage, a sequence of waypoint positions is selected from the geometric plan and reference orientations at the waypoints are planned. Orientation planning exploits properties of the VFO controller used for subsequent motion execution. Proposed two-stage algorithm admits changes of robot motion strategy (forward/backward movement) and has lower computational cost than the full configuration space search. Performance of the algorithm can be intuitively tuned with provided design parameters.*

Keywords: *VFO control, waypoint-following, unicycle-like robot, combinatorial search, controller-driven motion planning*

1. Introduction

Motion planning for systems with nonholonomic constraints is a topic recently explored by many researchers all around the world. Significance of this topic stems from the fact that many wheeled mobile robots and the vast majority of road vehicles are subject to nonholonomic constraints. Efficient and theoretically sound motion planning in the cluttered environment is paramount to effective automation of wheeled vehicles. While many of the existing motion planners achieve this goal to some extent, they all come with some limitations, which are outlined in the short survey of motion planners presented below. A complete survey of motion planning algorithms can be found in [4, 15, 17].

Availability of powerful computing units has stimulated the development of sampling-based motion planners. Among them, the most successful are Rapidly Exploring Random Trees [13, 16] and their variations referred to generally as Rapidly Exploring Dense Trees (see [17]). Those algorithms are applicable to a wide range of systems and perform well in high-dimensional configuration spaces with no efficient collision checking procedures available. This versatility is achieved by means of probabilistic sampling in the problem space and particular method of extending the tree of possible solutions. Unfortunately, those algorithms are non-deterministic, i.e. two runs with equal boundary conditions may not result in the same solution. Moreover, they are not strictly complete, but rather asymptotically complete, which means that there is no guarantee of finding a solution in finite time. Their Voronoi bias property leads to convergence problems in environments containing narrow passages and in the neighborhood of boundary conditions. Many of those problems can be fixed systematically (see [10]) or heuristically (see [1]) by making specific trade-offs. Most of these algorithms rely on numerical simulation of system's evolution for exploring the problem space and make use of no other system-specific information.

The other major class of motion planners is formed by grid-based algorithms, which model the problem space as a multidimensional lattice. That lattice is searched systematically for a feasible path with algorithms such as Dijkstra, Value Iteration, A* or their variations [2, 5]. Effectiveness of the planner depends on choice of the search space and specialization of the search algorithm. Many search spaces were proposed, some of them are: control spaces, configuration space grids with dynamic neighborhoods [8], state lattices constructed with motion primitives [25] and simplicial decompositions searched continuously [27]. Specialization of search algorithms is achieved mainly by redefinition of movement costs in the grid and simple customization of heuristic functions (see [14]). Some variations of grid-based planners include planners searching in the preprocessed decompositions of the problem space such as hierarchical grids, visibility graphs, etc. Grid-based planners are usually resolution complete, i.e. they will always find a solution in finite time if the grid resolution is sufficiently high. They are deterministic by construction and excel in low-dimensional problem spaces. Their convergence rate may be affected by topology of the environment, but not as much as in the case of sampling-based planners.

Some planners using genetic algorithms [3], mathematical programming [12] or other not mentioned before forms of global optimization can be found. The vast majority of motion planners is designed to generate a plan in the form of control signals (e.g. piecewise constant control actions or sinusoidal controls), parametric paths or trajectories. Only a handful of al-

gorithms employ a plan description in the form of waypoints [3, 11, 12, 23], despite its advantages such as lower memory requirements, human readability of the plan or opportunities for easier replanning and execution of the plan.

To achieve the aforementioned advantages, we propose a two-stage global controller-driven planner generating a sequence of waypoint positions and reference orientations at those positions dedicated for a unicycle driven by the VFO (Vector Field Orientation) control law. It is designed to exploit the specific beneficial properties of the VFO controller used for motion execution. During the first stage of planning, an occupancy grid of positions with robot direction-dependent neighborhood is searched with a orange specialized A*-like algorithm. Previous efforts such as [7] have shown that such specialization can lead to faster planning. In our case the specialization was performed in a controller-driven manner, i.e. with use of the knowledge about motion execution process specific to the VFO control law. Novel motion cost-to-go and cost-to-come definitions are proposed along with simple, yet effective neighborhood pruning rules. At the beginning of the second stage waypoint positions are chosen based on the results of the first stage. The second stage of planning is then completed by an analytic procedure computing reference orientations to ensure collision-free and sufficiently smooth execution of the waypoint sequence with the VFO controller. Resulting planner works in various environments and performs searches in only two-dimensional problem space as opposed to the three-dimensional configuration space of a unicycle. Moreover, it automatically plans changes of the motion strategy (forward/backward movement) and guarantees collision-free motion execution in the closed-loop system if the VFO controller is used. Performance of the planner can be customized to specific applications with

The paper is organized as follows. In Section 2 we state the problem under consideration. Section 3 contains information about the VFO controller necessary to understand the waypoint planning process. Section 4 provides a high level view of the planning process. Section 5 provides rationale behind choice of the search algorithm and an in-depth description of modifications resulting from specialization of the search. In Section 6, the second, semianalytic stage of planning is described. Section 7 contains a brief discussion of computational complexity issues. In Section 8 we present simulation results verifying feasibility of the proposed planner and finally we conclude with some closing remarks and future plans in Section 9. This work is an extension to the conference paper [6].

2. Problem Statement

The following assumptionsare made:

A1. The motion environment is structured, i.e. it is fully known before the planning process begins. Only static obstacles are considered.

A2. Robot's footprint is bounded by a rectangular area of width a_r and length b_r as shown in Fig. 1.

A3. The motion environment is described with a binary occupancy grid. Grid cells have the width and height of ϕ [m]. The grid has passages no narrower than $w_k = 2\left(\frac{\phi}{2} + o_r\right)$ [m], where $o_r \triangleq \sqrt{a_r^2 + b_r^2}$ is the radius of a bounding circle of the robot shown in Fig. 1.

A4. The robot has unicycle kinematics described by the equation:

$$\dot{\boldsymbol{q}}(t) = \begin{bmatrix} 1 & 0 \\ 0 & \cos\theta(t) \\ 0 & \sin\theta(t) \end{bmatrix} \boldsymbol{u}(t), \quad \boldsymbol{u}(t) \triangleq \begin{bmatrix} \omega(t) \\ v(t) \end{bmatrix}, \quad (1)$$

where t denotes time, $\dot{\boldsymbol{q}}(t)$ is the configuration velocity vector of the robot, $\boldsymbol{q}(t) \triangleq \begin{bmatrix} \theta & \boldsymbol{q}^{*\top}(t) \end{bmatrix}^\top$ describes robot's configuration, $\boldsymbol{q}^*(t) \triangleq \begin{bmatrix} x(t) & y(t) \end{bmatrix}^\top$ is the position vector, $\theta(t)$ denotes orientation angle of the robot, $x(t)$ and $y(t)$ are coordinates of the robot's guidance point as shown in Fig. 1 and $\boldsymbol{u}(t)$ is the vector of control signals consisting of the angular velocity $\omega(t)$ and longitudinal velocity $v(t)$ of point P.

A5. Motion execution is performed by the VFO controller in the version for the waypoint-following task described in details in [23].

A6. External disturbances and measurement noises can be neglected.

Assumption A1 seems restrictive at first glance, but it proves to be reasonable in many applications targeted by the presented planner. For example, robots inspecting buildings in dangerous conditions can be easily equipped with an up to date map of the building. It is also possible to periodically repeat the planning process with an updated environmental map. Passage width in assumption A3 is chosen in such a way that a narrowest passage in the occupancy grid can be traversed by the robot taking into account the enclosing circle of it's rectangular footprint and the discretization error introduced by the grid. The unicycle model from the assumption A4 serves as a generic robot-body kinematics for restricted-mobility vehicles. Thus, as a consequence, proposed algorithm could be applied to robots with more complex kinematics, e.g. car-like robots (see [22]). An application to skid-steering robots is also possible (see [20]). Assumption A5 is extensively discussed in Section 4.

Planning the waypoint-following task consists of finding three sequences:

$$Q \triangleq \{\boldsymbol{q}_{d1}; \boldsymbol{q}_{d2}; \ldots \boldsymbol{q}_{dN}\}, \quad (2)$$

$$\Sigma \triangleq \{\sigma_1; \sigma_2; \ldots \sigma_N\}, \quad (3)$$

$$M \triangleq \{\mu_1; \mu_2; \ldots \mu_N\}, \quad (4)$$

where $\boldsymbol{q}_{di} = \begin{bmatrix} \theta_{di} & x_{di} & y_{di} \end{bmatrix}^\top$, $i = 1, \ldots, N$ is an i-th waypoint configuration, Q is the set of waypoint configurations, Σ is the set of motion strategy variables (1 for forward motion, -1 for backward motion), and M is the set of waypoint relative directing coefficients. Motion strategy variables determine whether the particular waypoint should be realized by forward or

backward motion of the robot. Ability to plan changes in motion strategy helps in challenging environments, as will be shown in Section 8. The anticipated path drawn by the robot during motion execution is shaped by relative directing coefficients from set M, which influence the manner in which the robot approaches the waypoint (see Section 3). Sets Q, Σ and M should be such that motion execution of the plan with an initial condition $\boldsymbol{q}_0$ will be collision-free and will complete in finite time under assumptions A1-A6. Motion execution is collision free if $\forall t > 0 : \mathcal{C}_R^* \cap \mathcal{C}_{obs}^* = \emptyset$, where $\mathcal{C}_{obs}^*$ is the subset of position space occupied by obstacles according to the occupancy grid from assumption A3, and $\mathcal{C}_R^*$ is the subset of position space occupied by the enclosing circle of the robot's footprint according to assumption A2. Motion execution ends in finite time if

$$\exists\, t_N < \infty : \forall t \geq t_N \, ||\boldsymbol{q}_{dN}^* - \boldsymbol{q}^*(t)|| \leq \epsilon_N, \qquad (5)$$

where $||\cdot||$ denotes an Euclidean norm and $\epsilon_N > 0$ is a prescribed vicinity of zero.

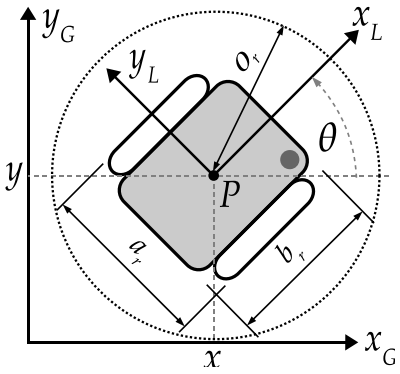

Fig. 1. A unicycle-like robot with a rectangular footprint $a_r \times b_r$ ***and guidance point P(x, y)***

3. A Brief Introduction to the VFO Control Law

The waypoint-following control task is treated as a sequence of set-point control tasks, as in [23]. The controller switches to the $(i+1)$-st waypoint when the robot enters an ϵ_i ball centered at the current waypoint, i.e. when $||\boldsymbol{q}_{di}^* - \boldsymbol{q}^*(t)|| \leq \epsilon_i$. Let us consider execution of a single motion segment - the movement between $(i-1)$-st and i-th waypoint.

The VFO control law is decomposed into the pushing control $v(t)$ and the orienting control $\omega(t)$. The orienting control aligns the configuration vector of the robot with the convergence vector field $\boldsymbol{h}_i(t)$. The pushing control moves the robot along the direction resulting from the influence of the orienting control on the system. Such separation results in the following control law for a unicycle during execution of the i-th motion segment:

$$\omega \triangleq h_{ai}(t) = k_a e_{ai}(t) + \dot{\theta}_{ai}(t), \qquad (6)$$

$$v \triangleq \sigma_i \bar{\rho}_i(t) \cos e_{ai}(t), \qquad (7)$$

$$\boldsymbol{h}_i(t) = \begin{bmatrix} h_{ai}(t) \\ \boldsymbol{h}_i^*(t) \end{bmatrix} \triangleq \begin{bmatrix} h_{ai}(t) \\ h_{xi}(t) \\ h_{yi}(t) \end{bmatrix}, \qquad (8)$$

$$e_{ai}(t) \triangleq \theta_{ai}(t) - \theta(t), \qquad (9)$$

$$\theta_{ai}(t) \triangleq \text{Atan2c}(\sigma_i h_{yi}(t), \sigma_i h_{xi}(t)) \in \mathcal{R}^1, \qquad (10)$$

$$\boldsymbol{h}_i^*(t) \triangleq k_p \boldsymbol{e}_i^*(t) + \boldsymbol{v}_i^*(t), \qquad (11)$$

$$\dot{\theta}_{ai} = \frac{\dot{h}_{yi} h_{xi} - \dot{h}_{xi} h_{yi}}{h_{xi}^2 + h_{yi}^2}, \quad h_{xi}^2 + h_{yi}^2 \neq 0, \qquad (12)$$

$$\boldsymbol{e}_i^*(t) \triangleq \boldsymbol{q}_{di}^* - \boldsymbol{q}^*(t), \qquad (13)$$

$$\boldsymbol{v}_i^*(t) \triangleq -k_p \mu_i \sigma_i ||\boldsymbol{e}_i^*(t)|| \begin{bmatrix} \cos(\theta_{di}) \\ \sin(\theta_{di}) \end{bmatrix}, \qquad (14)$$

$$\mu_i = \frac{\eta_i}{k_p}, \qquad (15)$$

$$\bar{\rho}_i(t) \triangleq \begin{cases} U_2 \frac{||\boldsymbol{h}_N^*(t)||}{||\boldsymbol{h}_N^*(t_{N-1})||} & \text{for } i = N, \\ U_2 & \text{otherwise}, \end{cases} \qquad (16)$$

where a design parameter $k_a > 0$ is the orienting control gain, θ_{ai} is the auxiliary orientation, e_{ai} denotes the auxiliary orientation error, $\bar{\rho}_i(t) \geq 0$ is the longitudinal velocity profile, t_i is the time of switching to the $(i+1)$-st waypoint, $U_2 = const > 0$ is a design parameter, $\boldsymbol{h}_i(t)$ is the convergence vector field with position components $h_{xi}(t)$, $h_{yi}(t)$ and an orientation component $h_{ai}(t)$, $\sigma_i \in \{1, -1\}$ is the decision variable determining the motion strategy (forward or backward movement), a design parameter $k_p > 0$ is the pushing control gain, $\boldsymbol{e}_i^*(t)$ denotes the position error, $\boldsymbol{v}_i^*(t)$ denotes the virtual velocity vector with a directing coefficient $\eta_i \in (0, k_p)$, and finally $\mu_i \in (0, 1)$ is the relative directing coefficient. Definition of the velocity profile $\bar{\rho}_i(t)$ is the same as the one proposed in [23]. It has no influence on the planning process described in this paper. Operator Atan2c : $\mathcal{R} \times \mathcal{R} \mapsto \mathcal{R}$ is a continuous version of the standard Atan2$(\cdot) : \mathcal{R} \times \mathcal{R} \mapsto (-\pi, \pi]$ function, which is defined in [21]. Auxiliary orientation θ_{ai} is determined by the position component $\boldsymbol{h}_i^*(t)$ of the convergence vector field.

Consider that $e_{ai}(t) = 0 \implies \theta(t) = \theta_{ai}(t)$. It means that when $e_{ai}(t) = 0$, the configuration velocity of the robot has the same direction and turn as the convergence vector field $\boldsymbol{h}_i(t)$. The purpose of the orienting control ω is to achieve such a convergence. It can be seen from the definition of ω that the auxiliary orientation error e_{ai} converges to zero exponentially fast depending on k_a. Value of the pushing control v depends on the application-specific velocity profile function $\bar{\rho}_i$, motion strategy σ_i and the auxiliary orientation error.

The relative directing coefficient μ_i changes the intensity of the so called *directing effect*. When the directing effect is intensified, the magnitude of virtual velocity $\boldsymbol{v}_i^*(t)$ increases and the auxiliary orientation $\theta_{ai}(t)$ converges faster to the waypoint orientation $\theta_{di}(t)$. This effect is shown in Fig. 2 and discussed further in [21]. The ability to shape the path drawn by the robot using the relative directing coefficient μ_i is crucial in the second stage of the proposed planning algorithm, which is described in Section 6.

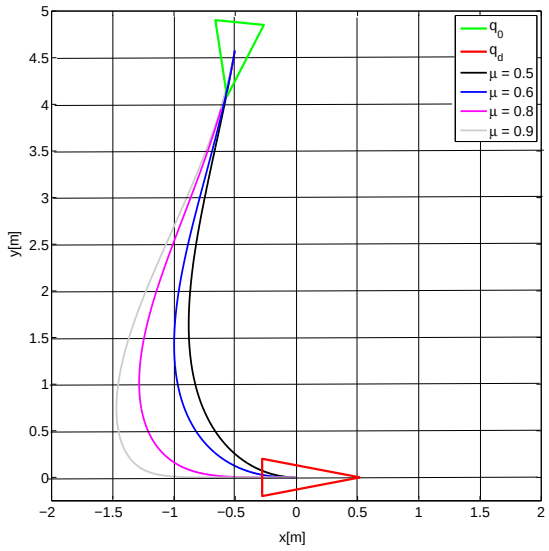

Fig. 2. Influence of the relative directing coefficient μ_i on the path drawn by the robot during the VFO set-point control process

4. A Controller-driven Planning Strategy

The motion plan is only as good as it's execution, since the only path we are really concerned with in practice is the path drawn by the robot, as opposed to the planned one. Controllers possessing specific properties, such as the VFO controller, can substantially reduce the complexity of the planning process by taking over some of the planner's responsibilities. This reduction in complexity of the solution is achieved in our case thanks to the directing effect mentioned earlier. A particular manner of approaching to the waypoint can be specified naturally without placing additional waypoints in the plan. In light of the above benefits, we propose a dedicated planner designed to exploit specific properties of the closed-loop system with the VFO controller, hence the name controller-driven planning strategy.

The controller-driven planning process shown in Fig. 3 proceeds as follows. First is the combinatorial stage, which consists of finding a collision-free path (in the sense of graph theory) in the occupancy grid representing the environment. Linear interpolation between positions of cells in the grid belonging to the solution of the search forms a piecewise rectilinear geometric path. That geometric path will be called the *geometric plan* in the sequel. While the algorithm of searching the occupancy grid for geometric plan is conceptually similar to many other planners, it is heavily specialized here to provide results most convenient to process during the second planning stage. Worth noting that the shape of the geometric plan implicitly determines the motion strategy (forward/backward movement) during execution of consecutive motion segments. Proposed controller-driven planner is somewhat similar to the hierarchical geometric motion planners. In both cases a geometric plan is generated as an intermediate step, as indicated by red rectangles in Fig. 3. Blue rectangles in Fig. 3 highlight areas using knowledge about the motion execution stage. In contrast to hierarchical geometric planners, proposed controller-driven planner utilizes the knowledge about the motion execution stage during the second stage of planning denoted by green rectangles in Fig. 3. As a consequence, generation of parametric reference paths is bypassed in our approach by generation of waypoints based upon results of the first stage of planning.

Analytic planning procedure from the second stage is responsible for ensuring that robot's motion during the waypoint-following task is collision-free and sufficiently smooth under the assumptions A5 and A6. Thanks to this stage, there is no need to perform any combinatorial search in the space of orientations. Nonholonomic constraints of a unicycle are taken into account only during the second stage of planning. As a consequence, cell processing during combinatorial search in the first stage is relatively fast and simple. This is crucial for planning performance, because combinatorial search accounts for the majority of time spent planning. Particular motion planning stages will be described in details in Sections 5 and 6, respectively.

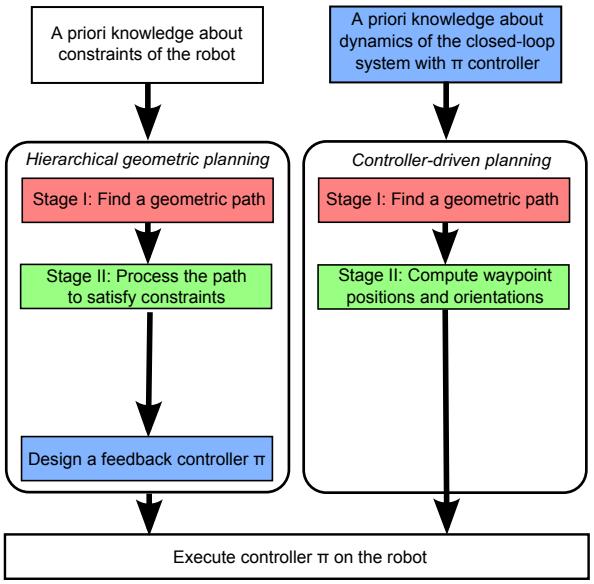

Fig. 3. Comparison of the hierarchical geometric motion planning and the proposed controller-driven planning approach

5. Stage 1: Planning Positions of the Waypoints

5.1. Combinatorial Search in the Occupancy Grid

Combinatorial search for the best geometric plan according to a specified edge cost is usually performed with a variation of the Value Iteration or Dijkstra algorithms. We assume that edge costs in the occupancy grid, i.e. costs of movement between it's cells will be positive. It means that a greedy variation of the Dijkstra algorithm, namely A*-like algorithm, can be used. There are specific requirements for the geometric plan. Namely, it must be relatively short and above

all safe, i.e. it should lie sufficiently far from the occupied grid cells. These requirements, along with guidelines specific to the second stage of motion planning, are reflected in the domain-specific modifications of the search algorithm described in Subsection 5.2.

Before describing proposed modifications, let us recall the basics of any A*-like algorithm. The goal of the search is to find a geometric plan connecting the start cell (represented by the variable "startNode") with the goal cell (represented by the variable "goalNode"). Coordinates of start cell and goal cell are obtained by quantizing the initial condition $\boldsymbol{q}_0$ and the goal configuration $\boldsymbol{q}_{dN}$ respectively. It is assumed that the quantization step is $\phi\,[\mathrm{m}]$. The search procedure guarantees eventual checking of all possible solutions by construction, because all encountered neighbors are placed in the ordered queue called the "priorityQueue" (lines 1 and 19) and processed in a best-first manner. It means that any A*-like algorithm is complete, regardless of the ordering of elements in the "priorityQueue". The key for ordering of unprocessed cells is the "EstimatedCost" variable (line 18) also denoted by f, which is defined as follows:

$$f \triangleq g + h. \tag{17}$$

It has two components. The first one – "CostToCome" (lines 14 and 17), also denoted by g, represents the movement cost of the best geometric plan connecting start cell to the currently processed cell. The second component is a result of the "heuristic" function (line 18), also denoted by h, representing the heuristic estimate of the movement cost resulting from the traversal of the remaining, yet unknown, portion of the geometric plan. Cost-to-come and estimated cost-to-go can be changed flexibly to obtain desired performance of the algorithm, as long as they are positive. Result of the search is a semioptimal geometric plan connecting the initial configuration $\boldsymbol{q}_0^*$ with the goal configuration $\boldsymbol{q}_{dN}^*$.

5.2. A specialized A*-like algorithm

Proposed search algorithm utilizes a particular definition of a neighbohood in the occupancy grid. It is compared with the standard omnidirectional neighborhood in Fig. 5. Let us define $\boldsymbol{\lambda}(c) \triangleq \begin{bmatrix}\lambda_x & \lambda_y\end{bmatrix}^\top$ as a position of cell's center in the configuration space. The neighborhood of a cell c is determined by a direction of the vector $\boldsymbol{d}_p \triangleq \begin{bmatrix}d_{px} & d_{py}\end{bmatrix}^\top = \boldsymbol{\lambda}(c) - \boldsymbol{\lambda}(p)$, where cell p is a predecessor of cell c in the current best path in the graph to currently processed cell c. Analogically to the definition of cell p, we define p_p as a predecessor of cell p. Neighbor positions and motion strategy (forward/backward movement) at those positions are directly dependent on current direction of the path. A set of cell neighbors $S \triangleq F \cup B$ contains neighbors $F \triangleq \{f_f, f_l, f_r\}$ attainable by moving forward in two diagonal directions and in straight direction relatively to the direction of $\boldsymbol{d}_p$. Neighbors $B \triangleq \{b_l, b_r\}$ attainable by moving backwards in two diagonal directions relatively to the direction of $\boldsymbol{d}_p$ are also included in the set of neighbors S. Neighbors attainable by straight backward movement are pruned to prevent the search from unnecessary generation of cycles in the geometric plan. Thanks to the analysis conducted in [7], neighbors attainable by movement in directions perpendicular to the direction of $\boldsymbol{d}_p$ are also pruned. Movement in those directions occurs in the geometric plan only when it is forced by an immediate neighborhood of an occupied cell while searching for the shortest geometric plan. As discussed in Subsection 5.1, in this application we are not concerned with the shortest geometric plans in the occupancy grid, but with ones relatively distant from the obstacles. Moreover, since assumption A3 holds, processed occupancy grid cannot contain passages so narrow that 90 degree turns could be forced. Thus, mentioned neighbors can be safely pruned without losing completeness of the search. Finally, proposed smaller neighborhood implies a 37.5 percent lower branching factor of the processed graph, since only 5 neighbors in the grid are considered instead of all 8 usually considered neighbors (see Fig. 5). Lower branching factor leads to lower computational cost of the search process.

Choice of the motion strategy during the combinatorial search is a consequence of assumed neighborhood definition. The search algorithm labels processed grid cells with an additional variable $\sigma \in \{1, -1\}$ signifying the motion strategy used while traversing the cell. The same cell can have two entries in the "priorityQueue", one for forward movement and one for backward movement. The current value of motion strategy is maintained during the search. It is assigned to every cell visited during the search. If a neighbor from the set B is chosen, current motion strategy changes to the opposite value. In other cases current motion strategy remains unchanged. Resulting geometric plan contains information about motion strategy for every relevant cell of the grid. This concept is easier to grasp, when one visualizes the search process as a search in two interconnected occupancy grids, as can be seen in Fig. 7.

Combinatorial search is performed in the domain of robot positions, but information about initial orientation θ_0 and goal orientation θ_{dN} can be used to choose a geometric plan most compliant with them. Generation of such plans helps in simplifying the second stage of planning and minimizing the auxiliary orientation error $e_a(0)$ at the beginning of motion execution. Simulation studies have shown that the following two simple techniques of neighborhood pruning are effective, outperforming even very elaborate redefinitions of motion cost functions. At the beginning of the combinatorial search one chooses $\boldsymbol{d}_p$ to satisfy $\mathrm{Atan2}\,(\boldsymbol{d}_{py}, \boldsymbol{d}_{px}) = \theta_0$. At the end of the combinatorial search all the neighbors $s \in S$ of the currently processed cell c satisfying $|\mathrm{Atan2}\,(s_y, s_x) - \theta_{dN}| > \frac{\pi}{4}$ are pruned, as can be seen in Fig. 6.

The last specialization of the search algorithm is a new definition of the motion costs, namely the cost-to-come g and the estimated cost-to-go h. They are de-

```
priorityQueue ← ∅
startNode ← mapToGraph(q*_0)
goalNode ← mapToGraph(q*_dN)
priorityQueue.Insert(startNode)
visit(startNode)
while priorityQueue ≠ ∅ do
    processedNode ← priorityQueue.RemoveFirst()
    if processedNode == goalNode then
        return reconstructPath(goalNode)
    markAsClosed(processedNode)
    for all neighbor ∈ neighbors(processedNode) do
        if (wasClosed(neighbor) then
            continue
        tentativeCost ← processedNode.CostToCome + neighbor.MovementCost
        if !priorityQueue.Contains(neighbor) || tentativeCost < neighbor.CostToCome then
            neighbor.Parent ← processedNode
            neighbor.CostToCome ← tentativeCost
            neighbor.EstimatedCost ← neighbor.CostToCome + heuristic(neighbor, goalNode)
            priorityQueue.Insert(neighbor)
return SearchFailed
```

Fig. 4. The iteration process of an A*-like algorithm

fined as follows:

$$h(c,p) \triangleq g(c,p)\left[s_f||\boldsymbol{q}^*_{dN} - \boldsymbol{\lambda}(c)|| - 1\right], \tag{18}$$

$$g(c,p) \triangleq m_c s_c\left[g(p,p_p) + ||\boldsymbol{\lambda}(c) - \boldsymbol{\lambda}(p)||\right], \tag{19}$$

$$s_f \triangleq \sqrt{\frac{\hat{D}}{\min(D)}}, \tag{20}$$

$$s_c \triangleq 1 + \frac{k_s}{\phi\exp(\min(D))}, \quad k_s \geq 1 \tag{21}$$

$$D \triangleq \{d_1, d_2, d_3, d_4\}, \tag{22}$$

$$\hat{D} \triangleq \frac{(d_1 + d_2 + d_3 + d_4)}{4}, \tag{23}$$

with

$$m_c \triangleq \begin{cases} 0.9 & \text{if c keeps current motion direction,} \\ 1.1 & \text{if c changes the motion strategy,} \\ 1.0 & \text{otherwise,} \end{cases} \tag{24}$$

where ϕ is the cell size, g is the cost-to-come from $\boldsymbol{q}^*_0$ to $\boldsymbol{\lambda}(c)$, h is the estimated cost-to-go from $\boldsymbol{\lambda}(c)$ to $\boldsymbol{q}^*_{dN}$. The safety coefficient s_c is a measure of cell safety, which is a nonlinear function of the minimum element of the set D. Set D contains distances to nearest occupied cells in the directions dependent on the vector $\boldsymbol{d}_p$, much like in the case of neighborhood described previously (see Fig. 5). $\hat{D}$ denotes an arithmetic mean of elements in D. Directional coefficient m_c slightly modifies the motion cost to favor geometric plans with lower number of edges and fewer motion strategy changes. It was chosen empirically, but it does not depend on topology of the grid. Safety coefficient s_c is designed in such a way, that cost-to-come g grows rapidly for cells lying near the obstacles. It can be seen from equations (18) and (20) that cost-to-come g is actually scaled by the value of safety coefficient s_c. It can be easily verified that $s_c|_{\min(D)=0} = 2$ and $s_c|_{\min(D)\to\infty} = 0$. Moreover, thanks to the term $\exp(\min(D))$, the safety coefficient s_c decreases fast and monotonically with respect to the shortest distance to an obstacle. Worth noting that it is crucial for s_c to be bounded as in the above definition. If the safety coefficient s_c could grow to arbitrary large values, the importance of safety in cost-to-come could not be controlled. The importance of plan safety can be tuned with the safety gain $k_s \geq 1$. Higher values of k_s result in longer, but safer paths. On the other hand, if k_s is so high, that the coefficient s_c is dominant in the cost f, geometric plan becomes unnecessarily long and number of processed cells rises significantly. The scaling coefficient s_f, dependent on the mean $\hat{D}$, is designed to amplify the effect of the safety coefficient s_c in wide passages of the environment.

Remark 1. *Tuning of the proposed A*-like search algorithm is performed only by changing the safety gain k_s and cell size ϕ. All other terms used in (14) are computed automatically. One could also experiment with different definitions of m_c.*

Definitions (18–23) are a result of extensive empirical studies providing insight into the interactions between motion costs and other components of the planner. Proposed specific dependency of both cost-to-come g and estimated cost-to-go h on the distance from obstacles results in a global influence of this distance on the choosen path, as opposed to local influence present in many existing algorithms (see [2]). It means that different (precisely: homotopically inequivalent) passages in the occupancy grid will be chosen, depending on their influence on safety of the geometric plan. The estimated cost-to-go h is inadmissible (see [9]) by design, i.e. it overestimates distance to the goal. Such a choice can be useful in some scenarios as indicated in [19, 24]. In this case, the heuristic function h was designed to better direct the search

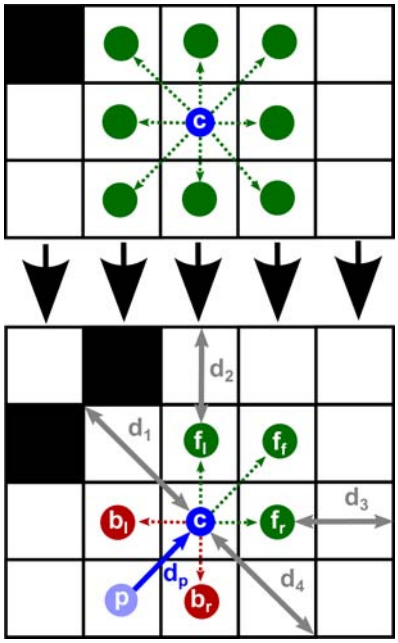

Fig. 5. Proposed direction-dependent neighborhood (on the bottom) in the occupancy grid in comparison to the standard neighborhood (on the top)

when the influence of the safety coefficient s_c is significant. In definition (18) estimated cost-to-go is scaled by the cost-to-come of a particular grid cell, hence it inherits the useful properties of the cost-to-come outlined above. This scaling results in inflation of the estimated cost-to-go for cells lying close to obstacles. Notice that the scaling coefficient $s_f \geq 1$ inflates the estimated cost-to-go for cells lying far from the Voronoi diagram of the environment. It means that in broad passages of the environment, where safety coefficient s_c is low, the heuristic function h remains highly dependent on the distance to obstacles with the help of scaling coefficient s_f. Such a design results in a lower number of processed cells than in the case of algorithms, which use information about distance from the obstacles only in the definition of cost-to-come.

Remark 2. *Proposed choice of inadmissible and inconsistent heuristic function h results in generation of semioptimal solutions in terms of a total cost-to-come. However the task at hand does not require optimal solutions, since we are concerned with finding sufficiently safe and short paths, as opposed to shortest or safest paths.*

Remark 3. *The A*-like search algorithm presented in the paper is complete regardless of chosen heuristic function h, since eventually all potential feasible solutions will be checked (see the discussion of forward search [17]). Worth noting that the map area processed by the search algorithm with our heuristic function is lower than the area processed by Dijkstra search. This indicates that the proposed heuristic function is indeed serving its purpose, which is to guide the search towards more promising grid cells.*

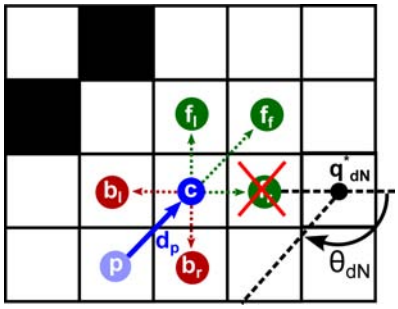

Fig. 6. Neighbor pruning based on the goal orientation

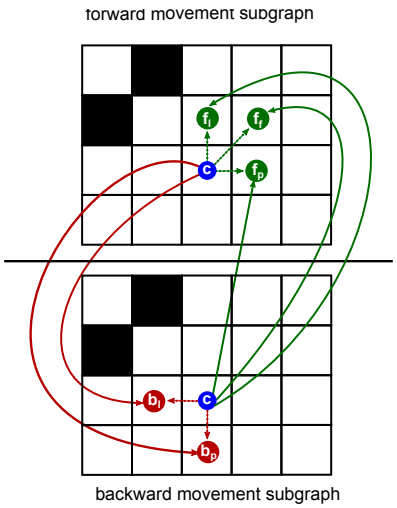

Fig. 7. Visualization of cells corresponding to different motion strategies in separate subgraphs

5.3. Processing the Results of Combinatorial Search

Thanks to modifications described in Subsection 5.2, generated geometric plan lies sufficiently far from the obstacles and has turning points in places, where the traversed passage changes significantly (i.e. when a sharp turn or a sudden transition to a narrow passage occurs). Moreover, the motion strategy can change only at turning points of the geometric plan. Given the above, we propose choosing every turning point of the geometric plan as a waypoint position. Waypoints corresponding to turning points of the geometric plan are augmented with additional intermediate waypoints placed on the polyline constituting the geometric plan in such a manner that subsequent waypoints lie no further away from each other than prescribed distance l. Motion strategy value assigned to a cell containing an i-th turning point of the geometric plan is assigned as a value of motion strategy σ_i for the corresponding waypoint and all the intermediate waypoints contained in the i-th segment of the geometric plan.

Remark 4. *Distance l should be tuned to achieve desired compromise between low waypoint count and increased safety of motion execution resulting from proximity of subsequent waypoints. Note that, while various strategies of adding intermediate waypoints could be employed, the waypoint sequence must contain all the turning points of the geometric plan.*

6. Stage 2: Planning the Reference Orientations

6.1. Recursive Computation of Reference Orientations

In the second stage of planning, an effective approach to planning desired reference orientations θ_{di} for waypoints proposed in [23] is utilized. The planning process begins from the goal configuration $\boldsymbol{q}_{dN}$ and proceeds back to the initial condition $\boldsymbol{q}_0$. Reference orientation θ_{di} of the i-th point should be chosen in such a way, that the robot will not turn abruptly after switching to execution of the $(i+1)$-st point. To achieve this, the auxiliary orientation $\theta_{ai}(t)$ should be continuous during the whole time of motion execution. Let us assume, for the purpose of subsequent considerations, that motion execution is perfect, i.e. $e_a \equiv 0$ and waypoints are switched precisely at reference positions, i.e. for $\epsilon_i \equiv 0$. Under those assumptions, auxiliary orientation will be continuous if $\theta_{di} = \theta_{ai+1}(t_i)$, where t_i denotes the time instant when switching to the $(i+1)$-st waypoint occurs. Considering that $\boldsymbol{q}^*(t_i) = \boldsymbol{q}^*_{di}$, the auxiliary orientation $\theta_{ai+1}(t_i)$ can be computed easily from (10) yielding the desired reference orientation θ_{di}. Computation of the auxiliary orientation $\theta_{ai+1}(t_i)$ requires knowledge of the relative directing coefficient μ_{i+1}. Description of how μ_{i+1} can be chosen is provided in the next subsection.

6.2. Determining the Relative Directing Coefficients

Effective choice of the relative directing coefficient μ_i is paramount to safety and smoothness of the resulting motion plan. It is evident, that if no nonholonomic constraints were present in the system, geometric plan could be used directly. The presence of kinematic constraints of the unicycle makes the geometric plan unfeasible, since the robot would have to stop and turn in place at every turning point of the plan.In other words, since continuity of the auxiliary orientation $\theta_{ai}(t)$ has to be ensured, the geometric plan can only be approximated by the robot's path. Relative directing coefficients μ_i should be chosen in such a way, that the resulting robot's path will be smooth enough to avoid intermediate stopping or abrupt turns. At the same time, robot's path cannot lie far from the geometric plan to avoid collisions.

Let us analyze the two extreme choices of μ_i. Figure 8 (blue path) illustrates the first choice $\mu_i = \mu_{ai}$, where μ_{ai} is a value of relative directing coefficient guaranteeing that:

$$\theta_{di-1} = \text{Atan2}\left(w_{yi-1}, w_{xi-1}\right), \tag{25}$$

$$\boldsymbol{w}_i = \begin{bmatrix} w_{xi} & w_{yi} \end{bmatrix}^\top \triangleq \boldsymbol{q}^*_{di} - \boldsymbol{q}^*_{di-1}, \tag{26}$$

where $\boldsymbol{w}_i$ is the position error after switching to the i-th waypoint assuming perfect motion realization. If the first motion segment is analyzed, $\boldsymbol{w}_i$ can be constructed easily by taking into account the initial orientation θ_0. Guaranteeing satisfaction of relation (25) results in a closest possible fit of the path drawn by the robot to the $(i-1)$-th segment of the geometric plan. The value of μ_{ai} can be computed analytically using the following formulas:

$$\mu_{ai} \triangleq \begin{cases} \mu_{min} & \text{if } \hat{\mu}_{ai} < \mu_{min}, \\ \mu_{max} & \text{if } \hat{\mu}_{ai} > \mu_{max}, \\ \hat{\mu}_{ai} & \text{otherwise}, \end{cases} \tag{27}$$

$$\hat{\mu}_{ai} \triangleq \frac{w_{yi} - w_{xi}\text{ctg}\left(\text{Atan2}\left(w_{yi-1}, w_{xi-1}\right)\right)}{||\boldsymbol{w}_i||}. \tag{28}$$

Additional limits $\mu_{min} < \mu_{ai} < \mu_{max}$ are imposed on the relative directing coefficient for two reasons. Firstly, for very high relative directing coefficients, maximal curvature of path drawn by the robot will be high and the path will be similar to a polyline. Secondly, μ_{min} imposes a minimal acceptable smoothness level of robot's motion. Equation (27) can be derived in two steps. Firstly the point $\boldsymbol{z}_i$ seen in Figs. 8–10 is computed using simple geometric reasoning and then using it's position, the length of vector $\boldsymbol{v}^*_i(t_{i-1})$ is computed (see Figs. 8-10). Secondly, $\hat{\mu}_{ai}$ is computed from Eq. (14) using the expression for $||\boldsymbol{v}^*_i(t_{i-1})||$ derived before.

While in the situation from Fig. 8 choosing $\mu_i = \mu_{ai}$ is acceptable, it often results in a poor fit of the robot's path to the i-th segment. Also, it makes no sense for the first motion segment. To achieve a closest possible fit of the robot's path to the i-th segment, one has to set μ_i to the lowest possible value μ_{min}. It can be seen in Fig. 10 (blue path) that such extreme choice reduces length of the virtual velocity $\boldsymbol{v}^*_i(t)$ to a very small value and effectively forces the robot to follow an almost straight line towards the next waypoint just like in the geometric plan.

We propose a choice resulting from a compromise between the two opposing criteria outlined above. It is illustrated in Fig. 11. The relative directing coefficient μ_i is computed as follows:

$$\mu_i \triangleq \frac{||\boldsymbol{w}_i||\mu_{min} + k_f||\boldsymbol{w}_{i-1}||\mu_{ai}}{||\boldsymbol{w}_i|| + k_f||\boldsymbol{w}_{i-1}||}, \tag{29}$$

where $k_f \in (0, \infty)$ is a design parameter signifying a the priority of the geometric plan during the computation of the relative directing coefficient. Lower values of k_f result in robot's paths following the geometric plan more closely, but less smoothly. Formula (29) is a simple weighted average with weights assigned according to the length of particular motion segments.

Choosing large value of k_f for traversing narrow passages could lead to robot's paths colliding with obstacles. To remedy this problem, we propose a computationally inexpensive collision test performed for every segment during the second stage of planning. It can be proven that when $e_{ai}(t) \equiv 0$, a sign of the angular velocity ω of the robot does not change during execution of a single motion segment. This fact implies that path drawn by the robot during the execution of the i-th motion segment is contained within a triangle constructed from the following points: beginning of the segment $\boldsymbol{q}^*_{di-1}$, end of the segment $\boldsymbol{q}^*_{di}$ and the point $\boldsymbol{z}_i$ shown in Figs. 8–10. That relation holds regardless of chosen relative directing coefficient μ_i.

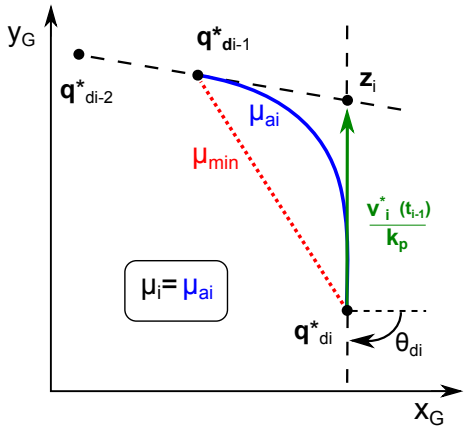

Fig. 8. Planning for the best fit of the path drawn by the robot in $(i-1)$-st motion segment

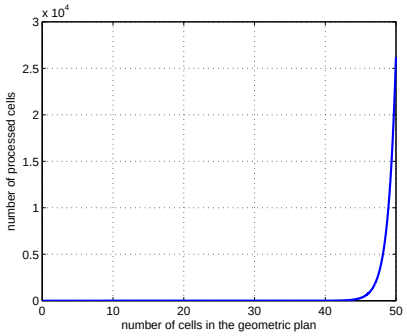

Fig. 9. Actual complexity of the proposed combinatorial search as a function of a geometric plan length

For every motion segment, after computing the relative directing coefficient μ_i, all cells of the occupancy grid lying inside of the mentioned triangle are checked for occupancy. If at least one cell is occupied, the algorithm falls back to choosing μ_{min} for this particular motion segment and the collision check is repeated. If the second collision check fails, the algorithm signals failure to generate a feasible motion plan. It means that the whole motion planning process is not complete for some unfavorable choices of the design parameters. Placement of additional waypoints mentioned in Subsection 5.3 and the specific definition of grid neighborhood from Subsection 5.2 helps with the aforementioned issue of collisions. They result in a more dense waypoint placement in narrow passages of the environment. Increased number of waypoints implies an increased number of motion segments and, as a consequence, increased number of relative directing coefficients μ_i that can be used to shape robot's motion in the second stage of planning. Worth noting, that the proposed strategy results in lower number of waypoints than an alternative strategy of placing them in the small fixed distance from each other. Plans with lower number of waypoints are obviously easier to process, store and execute.

Remark 5. *In practice, one should choose sufficiently low values of design parameters k_f , μ_{min} and distance l to guarantee that the path drawn by the robot lies sufficiently close to the geometric plan in targeted environments.*

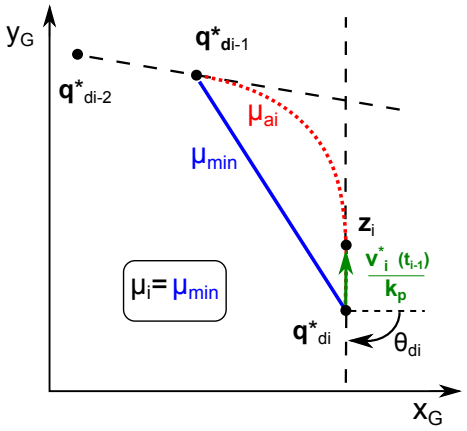

Fig. 10. Planning for the best fit of the path drawn by the robot in i-th motion segment

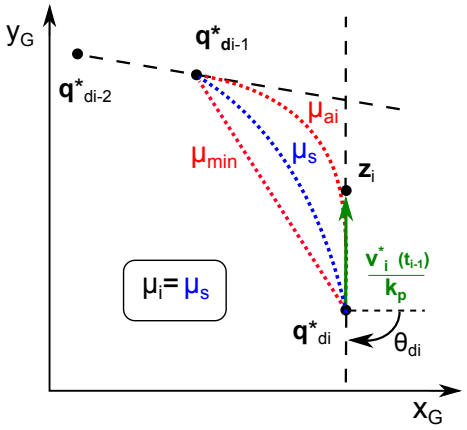

Fig. 11. Planning for a good fit of the path drawn by the robot in both $(i-1)$-st and i-th motion segment ($\mu_s = \frac{\mu_{ai}+\mu_{min}}{2}$)

7. Comments on Computational Complexity

The plot aggregating acquired simulation results in Fig. 9 shows how the proposed combinatorial search algorithm scales with the length of resulting geometric plan. It is evident that the number of processed cells scales exponentially. That result is congruent with theoretical computational complexity bounds of A*-like algorithms provided in [26]. Processing of a single cell happens in a constant time interval if one assumes that computational complexity of finding the set D is $\mathcal{O}(1)$. Such assumption is reasonable, since it is possible to find this set in amortized constant time (e.g. by means of caching).

The second stage of the planning process has the computational complexity $\mathcal{O}(N)$, where N is the number of waypoints in the plan. Despite that fact, the proposed algorithm may not be suitable for fast replanning in very big (i.e. bigger than 4000 cells) occupancy grids due to the nature of the combinatorial search procedure. In such applications proposed planner could be used with a hierarchical representation of the environment to mitigate the issue of long combinatorial searches.

8. Simulation Results

In this section results of selected simulations verifying effectiveness of the proposed planner are presented. The following notational convention has been assumed: triangles denote configuration of the robot at waypoints, green triangle corresponds to initial configuration $\boldsymbol{q}_0$, red triangle corresponds to goal configuration $\boldsymbol{q}_{dN}$, processed occupancy grid cells are marked with black „+" marks, finally colored curves show paths drawn by the robot resulting from simulated motion execution with the utilization of the VFO controller described in Section 3 and [23]. The following design parameters are common to all presented scenarios unless specified otherwise: $k_s = 1$, $k_p = 5$, $k_a = 10$, $a_r = 0.2\,\text{m}$, $b_r = 0.3\,\text{m}$, $\mu_{max} = 0.95$ and $\epsilon_i = 0.001\,m$. For scenarios S1-S6 $k_f = 5$ and $\mu_{min} = 0.2$ was chosen. For other scenarios design parameters were changed to $k_f = 10$ and $\mu_{min} = 0.3$. For maps used in scenarios S4-S6 resolution $\phi = 0.4\,\text{m}$ was chosen. Maps from other scenarios were processed with $\phi = 0.29\,\text{m}$.

Effectiveness of the algorithm is demonstrated in Fig. 12. Proposed combinatorial search procedure has processed only a fraction of unoccupied map cells. The global influence of narrow passages on the motion planning process can be seen in Fig. 14, scenario S4. Generated plan is longer, but safer than its shortest counterpart. Influence of the design parameters ϕ and k_f on the path drawn by the robot can be observed in Fig. 17. Changes in k_f have in fact the exact consequences discussed in Section 6. Repeated simulation of scenario S2 shown in Fig. 15 illustrates the influence of the safety gain k_s on planning of motion strategy (forward/backward motion). Additional changes in motion strategy (forward/backward motion) have been planned to achieve a safer traversal of a very narrow corridor towards the final waypoint. This scenario shows, that when motion safety is a priority, the safety gain k_s should be set to high values. Lowering the value of k_s significantly will lead to a situation where narrow passages will be chosen by the algorithm to optimize the length of the geometric plan. Scenario S7 in Fig. 13 contains a typical benchmark „U-shaped" obstacle (see [17]). A longer, but safer waypoint sequence avoiding narrow passages has been planned in this case. The „U-shaped" obstacle, which generates deadlocks for many algorithms based on potential fields, has been avoided here. Scenario S8 in Fig. 16 contains another benchmark environment shape, namely the „zig-zag" pattern (see [17,18]) designed to test algorithm's ability to plan smooth travel around corners intertwined with narrow passages. One can observe how the motion strategy (forward/backward motion) changes to traverse a particularly demanding corner and a smooth robot's path unattainable for many classic planners focusing on shortest paths. Scenario S9 in Fig. 18 shows performance of the proposed algorithm in an environment with topology similar the building of the Faculty of Electrical Engineering at Poznań University of Technology. The environment contains very narrow passages intertwined with long, wide passages to test performance of the planner in more spacious environments. Despite the fact that no design parameters were specially tuned for this task, it's execution succeeded.

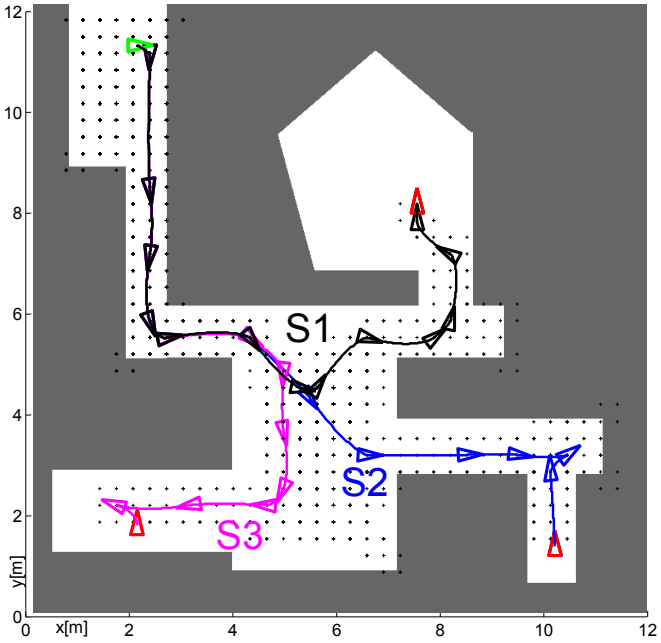

Fig. 12. Simulation scenarios S1, S2 and S3 with different initial and goal configurations

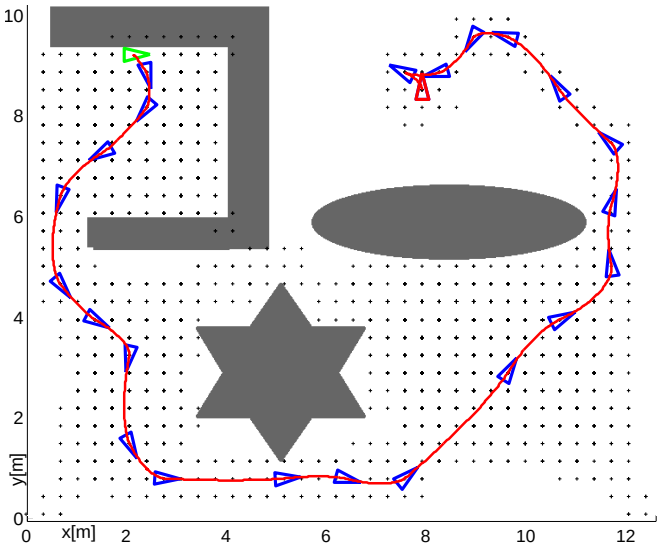

Fig. 13. Simulation scenario S7

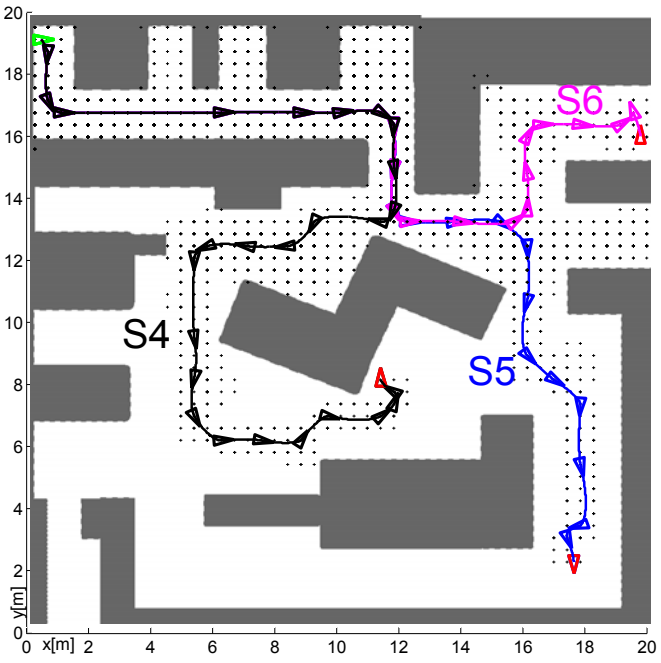

Fig. 14. Simulation scenarios S4, S5, and S6 with different initial and goal configurations

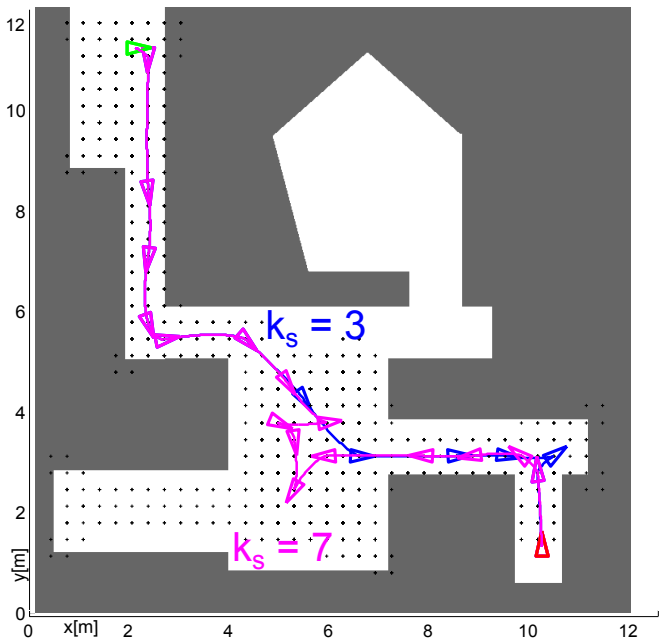

Fig. 15. Simulation scenario S2 for $k_s \in \{3, 7\}$

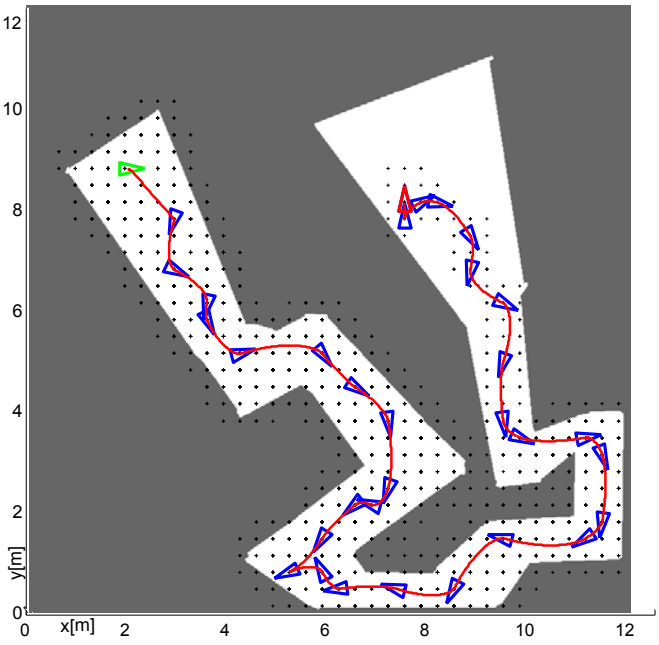

Fig. 16. Simulation scenario S8

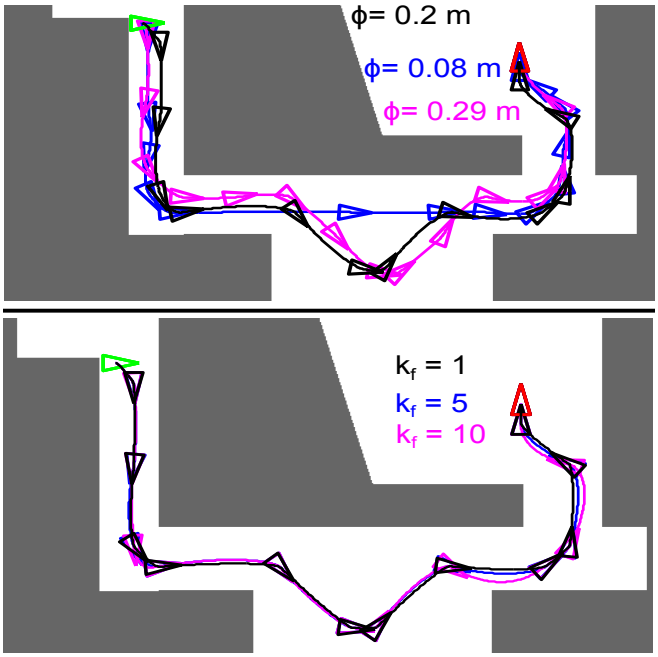

Fig. 17. The influence of design parameters ϕ and k_f on the plan and its execution

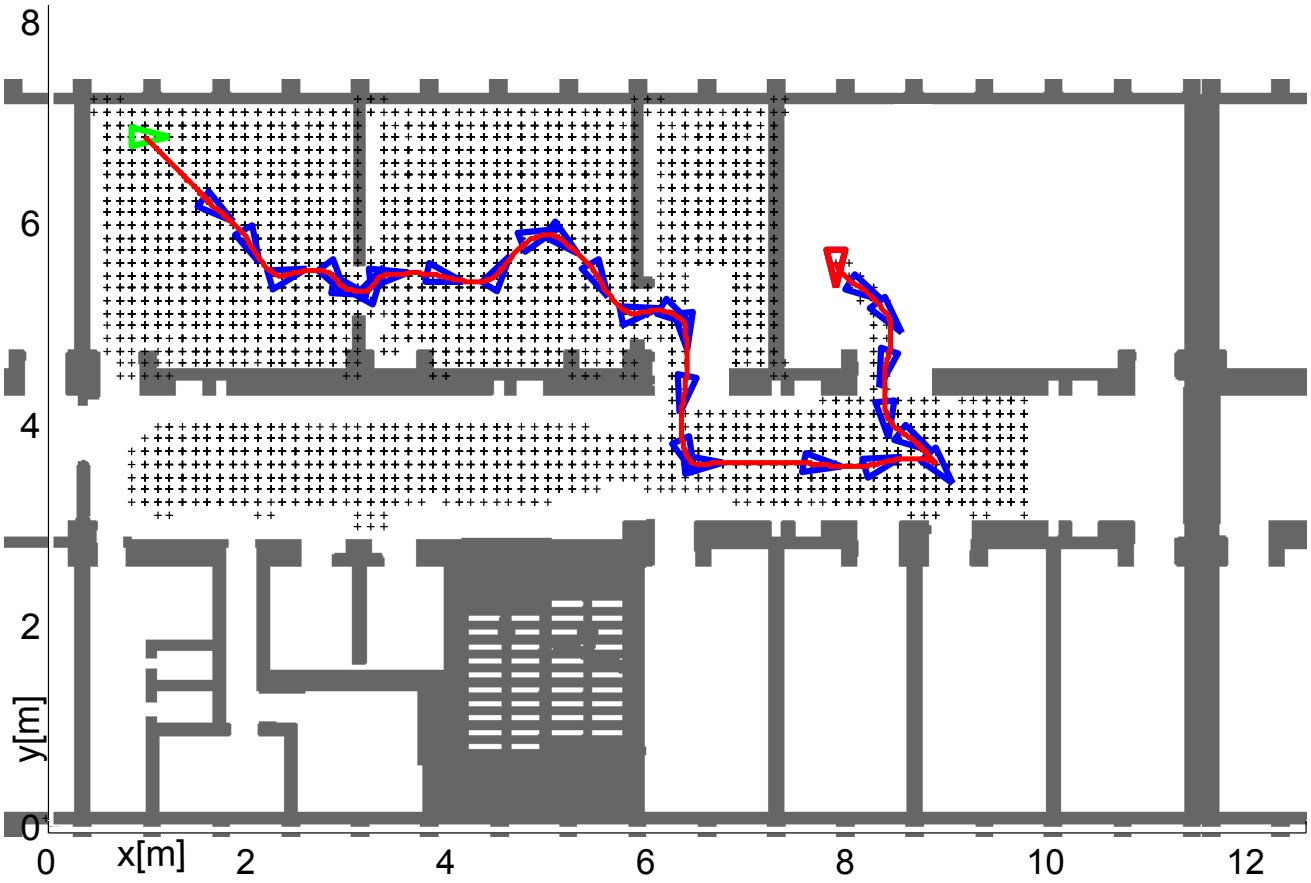

Fig. 18. Simulation scenario S9 utilizing the map of the building of Faculty of Electrical Engineering at Poznan University of Technology

9. Conlusions

In this paper a global two-stage motion planner for the waypoint-following task realized in a cluttered environment was proposed for a unicycle-like robot. Structure of the proposed solution is a result of a controller-driven approach to motion planning and particular beneficial properties of the VFO controller. Simulation studies have shown that the proposed planning algorithm can be effective for various topologies of the environment, leading to safe plans with maximized distances from obstacles. Performance of the planner can be adapted to a specific application intuitively by tuning the provided design parameters. Unlike many other planners presented to date, proposed solution directly takes into account dynamics of the closed-loop system evolving during motion execution stage. In the near future, authors are going to apply the presented controller-driven methodology to motion planning problems in the presence of motion curvature constraints imposed on the vehicle motion.

ACKNOWLEDGEMENTS

This work was supported by Polish National Center of Research and Development as the grant PBS1/A3/8/2012, research project No. 2224K from the Programme of Applied Research.

AUTHORS

Tomasz Gawron* – Chair of Control and Systems Engineering, Piotrowo 3A, Poland, Poznań, 60-965, e-mail: tomasz.gawron@doctorate.put.poznan.pl, www: www.tomaszgawron.pl.

Maciej M. Michalek – Chair of Control and Systems Engineering, Piotrowo 3A, Poland, Poznań, 60-965, e-mail: maciej.michalek@put.poznan.pl, www: www.put.poznan.pl/~maciej.michalek.

*Corresponding author

REFERENCES

[1] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions". In: *2011 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, San Francisco, CA, USA, 2011, 2640–2645, DOI: 10.1109/IROS.2011.6095077.

[2] M. Cikes, M. Dakulovic, and I. Petrovic, "The path planning algorithms for a mobile robot based on the occupancy grid map of the environment - A comparative study". In: *Proc. XXIII International Symposium on Information, Communication and Automation Technologies (ICAT)*, Sarajevo, Bosnia and Herzegovina, 2011, 1–8.

[3] T. Davies and A. Jnifene, "Multiple waypoint path planning for a mobile robot using genetic algorithms". In: *2006 IEEE Int. Conf. on Computational Intelligence for Measurement Systems and Applications*, La Coruńa, Spain, 2006, 21–26, DOI: 10.1109/CIMSA.2006.250741.

[4] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review", *IEEE Access*, vol. 2, 2014, 56–77.

[5] D. Ferguson and A. Stentz, "Field D*: An interpolation-based path planner and replanner". In: *Proc. of the Int. Symposium on Robotics Research (ISRR)*, San Francisco, CA, USA, 2005, 1926–1931.

[6] T. Gawron and M. Michałek. "Planowanie przejazdu przez zbiór punktów dla zadania zrobotyzowanej inspekcji". In: K. Tchoń and C. Zieliński, eds., *Problemy robotyki*, volume 194 of *Prace naukowe. Elektronika*, 35–44. Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2014. in Polish.

[7] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps". In: *25th Conference on Artificial Intelligence (AAAI-11)*, San Francisco, CA, USA, 2011, 1114–1119.

[8] D. D. Harabor and A. Grastien, "An optimal any-angle pathfinding algorithm". In: *2013 Int. Conf. on Automated Planning and Scheduling*, Rome, Italy, 2013, 308–311.

[9] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, 1968, 100–107.

[10] L. Janson and M. Pavone. "Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions". 2013 Int. Symp. on Robotics Research, Seoul, Korea, Available online: http://web.stanford.edu/ pavone/papers/ Janson.Pavone.ISRR13.pdf, Dec 2013.

[11] N. Jouandeau and Z. Yan, "Decentralized waypoint-based multi-robot coordination". In: *2012 IEEE Int. Conf. on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, Bangkok, Thailand, 2012, 175–178, DDOI: 10.1109/CYBER.2012.6392549.

[12] R. Junqueira Magalhaes Afonso, R. Kawakami Harrop Galvao, and K. Kienitz, "Waypoint trajectory planning in the presence of obstacles with a tunnel-milp approach". In: *2013 European Control Conference (ECC)*, Zurich, Switzerland, 2013, 1390–1397.

[13] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning", *Int. J. Rob. Res.*, vol. 30, no. 7, 2011, 846–894.

[14] R. Knepper and A. Kelly, "High performance state lattice planning using heuristic look-up tables". In: *2006 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Beijing, China, 2006, 3375–3380, DOI: 10.1109/IROS.2006.282515.

[15] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers: Norwell, MA, USA, 1991.

[16] S. M. Lavalle and J. J. Kuffner, "Rapidly-Exploring Random Trees: Progress and prospects". In: *Algo-*

rithmic and Computational Robotics: New Directions, 2000, 293–308.

[17] S. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.

[18] S. LaValle and M. Branicky. "On the relationship between classical grid search and probabilistic roadmaps". In: *Algorithmic Foundations of Robotics V*, volume 7 of *STAR*, 59–76. Springer Berlin Heidelberg, 2004.

[19] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality". In: *Advances in Neural Information Processing Systems 16*, British Columbia, Canada, 2004, 767–774.

[20] A. Mandow, J. L. Martinez, J. Morales, J. L. Blanco, A. Garcia-Cerezo, and J. Gonzalez, "Experimental kinematics for wheeled skid-steer mobile robots". In: *2007 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, San Diego, USA, 2007, 1222–1227.

[21] M. Michałek and K. Kozłowski, "Vector-Field-Orientation feedback control method for a differentially driven vehicle", *IEEE Transactions on Control Systems Technology*, vol. 18, no. 1, 2010, 45–65, DOI: 10.1109/TCST.2008.2010406.

[22] M. Michałek and K. Kozłowski, "Feedback control framework for car-like robots using the unicycle controllers", *Robotica*, vol. 30, 2012, 517–535.

[23] M. Michałek and K. Kozłowski, "Motion planning and feedback control for a unicycle in a way point following task: The VFO approach", *Int. J. Appl. Math. Comput. Sci.*, vol. 19, no. 4, 2009, 533–545, DOI: 10.2478/v10006-009-0042-2.

[24] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1984.

[25] M. Pivtoraiko and A. Kelly, "Differentially constrained motion replanning using state lattices with graduated fidelity". In: *2008 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Nice, France, 2008, 2611–2616.

[26] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Pearson Education, 2003.

[27] D. S. Yershov and S. LaValle, "Simplicial Dijkstra and A* algorithms for optimal feedback planning". In: *2011 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, San Francisco, CA, USA, 2011, 3862–3867.