A blind-spot-aware optimization-based planner for safe robot navigation

Kenny Schlegel[1], Peter Weissig[1] and Peter Protzel[1]

***Abstract*— Safe mobile robot navigation should consider not only collision avoidance with current obstacles but also include non-visible areas (to which we refer as blind spots) and the resulting risk of collision with hidden moving objects (e.g. people). Such capability is important for mobile robots operating in environments shared with humans - for instance a shopping assistant robot in a supermarket. This work aims to extend an existing motion planner for mobile robots (the Time Elastic Band planner) by including blind spots. As a result, the final planner does not only consider static and visible dynamic obstacles, but handles blind spots, too. To identify such blind spots, we define and use *critical corners* that imply them. Hence, our contributions in this paper are creating a critical corner detector, which operates on laser scan data, and the extension of a factor-graph-based path planner. We evaluate the proposed method standalone and in our simulation environment of a supermarket. It can be seen that the implementation is capable of detecting and dealing with blind spots. Finally, we provide source code for both the detector and the planner extensions.**

I. INTRODUCTION

The presence of mobile robots in human environments requires carefully developed methods and algorithms for navigation. Particularly important is not only collision-free navigation but also human-aware motion planning. A sophisticated application scenario of such mobile robots in dynamic environments is a shopping assistant robot as in [1]. Since this topic is part of an ongoing research project, we will concentrate on this domain in the present paper, but the proposed approach is also transferable to other applications. Such a shopping assistant robot can make people's lives easier - for instance, it works as a guide, gives information or collects articles. For these tasks, the robot, shown in Fig. 1 as a simulation, needs to satisfy several requirements: (1) it has to reach its target position as fast as possible, (2) it must not collide with any object, and (3) it should move as socially acceptable as possible, which is known as human-aware motion planning. The first and the second point are well studied and the literature provides several methods. The third one is more complex because the attribute *socially acceptable* is open to interpretation. An important component is, for example, the implementation of *visible* dynamic obstacle avoidance (e.g. moving people). It consists of tracking the person and predicting their future trajectory, which should not be crossed. However, a system that attempts to satisfy all three navigation requirements is described in our previous work [1], where a major component is the Time Elastic Band (TEB)-planner from [2] and [3] – this planner incorporates static and visible dynamic obstacles.

Supported by the Federal Ministry of Education and Research, Germany.
[1]all authors are with Faculty of Electrical Engineering, Chemnitz University of Technology, Germany *firstname.lastname*@etit.tu-chemnitz.de

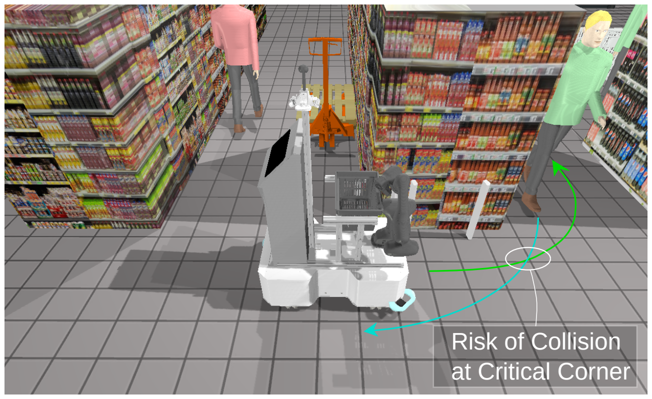

Fig. 1. Visualization of a critical corner within our supermarket simulation. There is a risk of collision since the robot follows the green and the person the light-blue path. Through the occlusion by the shelf, they do not see each other.

One more important aspect of the third navigation requirement is the incorporation of *non-visible* dynamic obstacles. In an environment with *non-visible* dynamic objects, there is the additional risk of collisions at blind spots (also known as occluded areas). Such danger zones may arise if the sensor's field of view is limited and objects occlude some areas where a hidden dynamic obstacle (person) can suddenly appear into the robot's field of view. Especially in a supermarket environment, there are many occlusions, e.g. at the shelves' corners. A visualization of such a case can be seen in Fig. 1: it contains a scene from the simulation presented in [1] with the shopping assistant robot and some people. The robot wants to follow the green and the human the blue path. The shelf creates a blind spot – there may be a person entering the robot's visible field – such a surprising situation can lead to a collision and the human can be scared off. Avoiding such situations is particularly important concerning the navigation requirements (2) and (3). To detect such blind spots, we define a critical corner that implies a beginning corresponding blind spot. This simplifies the problem description and solution implementation, because it reduces the unknown area of a blind spot to a clear point – the critical corner. We use the measurement principle of raw sensor data of a laser scanner to detect these critical corners because it provides a sequence of ordered points.

As in section II, the literature shows either some approaches for handling blind spots with possible non-visible dynamic obstacles, or for incorporating visible dynamic obstacles, like walking people. Hence, with our work, we are trying to close this gap and describe a solution to address both navigation situations within one planner. Since the factor-graph-based TEB-planner for static and visible dynamic obstacle avoidance performs well in [1], we extend

this approach in the present paper to deal with critical corners and incorporate possible non-visible dynamic obstacles. Such a planner that satisfies all necessary navigation requirements is currently missing in the literature. To sum up, the major contributions in this paper are:
1.) We describe a concrete and simple algorithm in section III-A for detecting critical corners that work directly on the robot's laser scans and is based on a ray-tracing approach.
2.) We develop an extension to the factor-graph based TEB-planner from [3] in section III-B to incorporate critical corners.

The critical corner detector and the planner are evaluated within the section IV. Finally, we provide the source code[1] of our implementation.

II. RELATED WORK

The literature provides several approaches dealing with occlusion during navigation. A seminal analysis of human-aware navigation methods can be found in [4], where some approaches are discussed that deal with occlusion, hidden zones and limited visibility. For instance, the authors in [5] focus on a motion planner dealing with social acceptance. They used the definition of hidden zones, which means that the robot is hidden from the human perception by an obstacle. Such a situation can lead to a surprise of humans and should be avoided. It is achieved through adapting the costmap by adding hidden zones as additional costs. An A* planner was then used to calculate the optimal path to avoid surprising situations. Our approach differs by switching from the human perspective to the robot perspective by asking: Which area is occluded from the robot's field of view? An example of such a perspective of the robot is given in [6]. The approach takes occlusions into account caused by objects and defines a collision and an emergency zone. The former zone enables it to distinguish between obstacles and other objects, and the latter defines the zone whose size corresponds to the stopping distance. If the detector finds an occlusion, the speed is reduced to decrease the size of the emergency zone. However, the path is always fixed during navigation – this prevents, for example, changing the path to increase speed and reducing the time to reach the target. Additionally, the planner is not able to incorporate visible dynamic obstacles (adapt the path according to the predicted obstacle's trajectory in the future).

Another approach, which additionally modifies the path, is described in [7]. In this paper, the authors define shadowing corners (similar to our *critical corners*), which represent the outer bounds of an object causing a shadow (occlusion). The visibility polygon algorithm from [8] calculates these shadowing corners. Based on the shadowing corners and an initial trajectory, a maximum velocity profile is computed. The initial trajectory is randomly deformed and evaluated with the newly calculated maximum speed to modify the speed and trajectory near occluded areas. After a few iterations, the time-optimal path is found. However, this approach is also not able to take into account visible dynamic obstacles, which is particularly important for a shopping robot, for example.

Chung et al. [9] provide a method for navigation in environments with occluded regions caused by a limited field of view. Similar to [7], they also deal with path planning and speed control. All occluded areas are calculated with a ray-tracing method and are transformed into risky regions mainly located around corners. The combination with additional calculated speed constraints (depending on the distance to the risk corners) results in a map with the safe speed in risk areas. A gradient-based route planning method and the DWA (Dynamic Windows Approach) algorithm for reactive planning were used based on this map. This approach takes into account non-visible dynamic obstacles but not visible dynamic obstacles in path adaptation with respect to their predicted paths.

In addition to mobile robotics, there are also contributions in the field of autonomous driving. For instance [10], [11] and [12], which describe approaches to occlusion-aware risk assessment. Since these methods are more adapted to urban scenarios and road geometry (with streets and lanes), they do not fit well with our problem at hand. Furthermore, these approaches are rather an evaluation of risky occluded areas than a motion planner. Only the approach of [13] from the field of autonomous driving is more related to our problem. There, the authors explained a method of maximizing the visibility (reduce blind spots) of sensors while overtaking a car. They also incorporate dynamic obstacles (e.g., a car on the opposite lane) with a conditional state machine that decides to wait until the car disappears. However, such hard-decided handling of visible dynamic obstacles does not fit our requirements of adapting the path concerning moving obstacles.

Finally, some of the listed methods, such as [6], [7] or [9], offer a solution for dealing with critical corners, but they are specifically adapted to their task and not open to solving additional problems such as visible dynamic obstacle avoidance (the TEB-planner can solve this part as in [1]). Such functionality is important in terms of human-aware navigation. That is why we extend the existing TEB local planner to a closed solution that solves the problem of visible and non-visible dynamic obstacles avoidance (at critical corners). Currently, the TEB planner can incorporate visible dynamic obstacles by their poses and velocity but does not consider potential risks at critical corners.

III. METHOD

The following section is divided into two parts, the detector and the trajectory planning.

[1]Source code and supplemental material can be found on our website `https://www.tu-chemnitz.de/etit/proaut/cc`

A. *Critical-Corner-Detector*

1) *Problem Definition:*

The desired behavior of a detector is that it finds possible critical corners and has a simple logic to reduce false positives. For the former, we are interested in an approach that does not require a special pre-processed representation and works directly on raw laser scan data. The latter means that critical corners which create blind spots outside the area of interest (e.g. behind the robot's direction of movement) should not be considered and neglected – this is important for better navigation behavior. Such false positives depending on the robot's geometry and sensor positions.

As shown in the Fig. 1, our robot is equipped with 2D laser scanners to provide a 360 degrees laser scan (a more detailed hardware setup can be found in [1]). We use this kind of sensors to detect critical corners for two reasons: (1) A laser scanner has a high measuring accuracy and a manageable data size, which is beneficial in terms of real-time capability. (2) Based on the measurement principle of a laser scanner (ordered sequence of beams), possible occlusions can be easily detected. The idea behind point (2) can be seen in the laser scan visualizations in Fig. 2: Since the point measurements within the 2D plane have the same coordinate frame and the laser beams are equally distributed, jumps between neighboring scans indicate a blind spot. Such blind spots are closely related to critical corners, which may imply a risk of collision.

As mentioned in Section II, the authors from [7] use the visible polygon algorithm from [8] to detect such critical corners. However, this method requires a polygon or line description of the objects as input and then starts a ray-tracing method. Since we would have to convert the current environmental perception into polygons or lines and then calculate the visible polygon, we instead used a method that works completely with the robot's raw input data based on a ray-tracing method similar to [14]. There, the authors define an occlusion as a point where the consecutive range values are discontinuous.

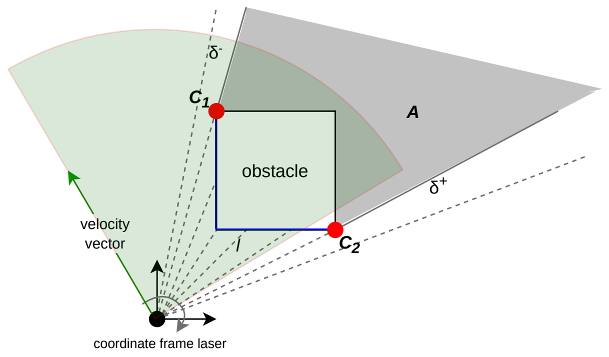

Fig. 2. Visualization of the definition of a critical corner **C**. The blue line represents the contour that caused the blind spot **A**. δ^+ and δ^- define the positive and the negative jump between two neighboring scan points. The dotted lines represent the laser beams that are evaluated clockwise. The green area shows the field of attention corresponding to the velocity vector to reduce false-positive detection.

For further explanations and more clarity, we define a critical corner in the 2D laser-scanner domain as follows:

Definition 3.1 (Critical Corner): A critical corner is a point **C** *in the robot's 2D environment that indicates the start of a blind spot* **A**. *The size of this area* **A** *is roughly related to the size of the closed contour l, which consists of consecutive points with small distances to each other, connected to the point* **C**. *A necessary condition of the point* **C** *is that there must be a jump δ between neighboring measurements (consecutive points of the laser scan). The sufficient condition of the point* **C** *is a minimal occlusion by the contour l.*

Fig. 2 illustrates definition 3.1 with a given obstacle and the corresponding contour l, two critical corners **C**, the approximate blind spot area **A** and the laser beams (dotted lines), which have the same origin. It can be seen that there are two possible types of jumps that correspond to critical corners: a positive (δ^+) and negative (δ^-). This definition of positive and negative jumps is based on the clockwise evaluation of the beams and corresponds to going from near to far and vice versa.

The green area in Fig. 2 represents the simple logic to reduce false positives. It corresponds to the velocity vector (green arrow) of the robot. If there is a critical corner (e.g., **C2** in Fig. 2) that is outside the area of interest (colored in green), it will be deleted and not be considered for future path planning, because the robot does not move towards the blind spot.

2) *Possible Solution:*

Based on the previous section, it is possible to implement a detector – algorithm 1 provides pseudo-code. The input is a sequence of clockwise-ordered laser scan points. Additionally, some parameters are important for adjusting the algorithm:

δ_{th}	the threshold of a jump between neighboring scans (a distance greater than δ_{th} between two points indicates a critical corner)
δ_{tol}	distance tolerance of neighboring points corresponding to a contour (a distance greater than δ_{tol} indicates a break of the contour)
l_{min}	minimum contour length causing occlusion in combination with a jump – it correlates with an area that is large enough to hide an obstacle

Using these parameters and the described detection principle, the proposed method may also have some limitations depending on the particular application: (1) If we consider only the occlusion length l_{min} with a contour, a special case of false-positive detection may occur. This means that an obstacle may have a small blind spot but its contour is considered long enough, e.g. because its surface is highly sloped. (2) Gaps within long contours can produce two critical corners if enough occlusion is before and after the gap. There is no test to distinguish between a huge (real) and small (neglectable) gab. The latter is also equal to false positives. Additional case discrimination for (1) and (2) increase the complexity and can be neglected due to the very rare occurrence. In the worst case, depending on the parameterization, the algorithm adds false-positive detections whereby the robot moves slower than necessary. False-negative results do not occur in the example presented, but this also depends on the application and parameterization of the detector.

However, for each laser scan measurement, the distance to the neighboring point is calculated (compare to line 4 in algorithm 1 – the angle of the laser scanner α_{scan} between the measurements is used). Next, positive and negative jumps must be detected. Particularly important is detecting the jump with the minimum occlusion length l_{min}. This minimum occlusion length will be computed in lines 8-13, accumulating the distances of neighboring points if they are not greater than δ_{tol}. Combining the jumps with the occlusion length requires different cases for both positive and negative jumps. A positive jump (according to Fig. 2) must take into account the contour, which creates the occlusion and occurs before it (lines 5-7). Conversely, the same applies to the negative jump - it evaluates the contour after the jump appears (lines 14-21). Finally, line 24 filters the critical corners, if they do not appear within the area of interest.

Algorithm 1 Detection Algorithm for critical corners.

```
Input: Laser-Scan of size $n$ with distances $D_i$, angle between scanning
    points $\alpha_{scan}$, jump threshold parameter $\delta_{th}$, tolerance parameter $\delta_{tol}$,
    minimum occlusion length $l_{min}$, robot's velocity vector $\overrightarrow{v}$
Output: Critical corners point cloud $CC$
 1: $occlusion \leftarrow 0$
 2: $negativFlag \leftarrow false$
 3: for $i = 2 : n$ do
 4:     $inbetweenDist \leftarrow \sqrt{D_i^2 + D_{i-1}^2 - 2 \cdot D_i \cdot D_{i-1} \cdot \alpha_{scan}}$   ▷ distance
    between laser points
 5:     if $(D_i - D_{i-1}) > \delta_{th}$ and $occlusion > l_{min}$ then   ▷ positiv jump
 6:         $CC.append(P_{i-1})$
 7:     end if
 8:     if $inbetweenDist < \delta_{tol}$ then   ▷ accumulate occlusion
 9:         $occlusion += inbetweenDist$
10:     else
11:         $occlusion \leftarrow 0$
12:         $negativFlag \leftarrow false$
13:     end if
14:     if $(D_{i-1} - D_i) > \delta_{th}$ then   ▷ negativ jump
15:         $negativFlag \leftarrow true$
16:         $tempPoint \leftarrow P_i$
17:     end if
18:     if $negativFlag$ and $occlusion > l_{min}$ then
19:         $CC.append(tempPoint)$
20:         $negativFlag \leftarrow false$
21:     end if
22: end for
23: for $c$ in $CC$ do   ▷ delete CC outside area of interest
24:     if $|\measuredangle(\overrightarrow{v}, c)| > 90^\circ$ then
25:         CC.delete(c)
26:     end if
27: end for
28: return $CC$
```

To use the described method and implement it on a real robot, we used ROS (Robotic Operating System). We created a node that uses laser scans as input and outputs the corresponding critical corners of the current scene. Since we operate within a simulation as in [1] and did a measurement of a real supermarket with another robot [15], we can quickly test the described method with simulated and real data. Fig. 3 provides some visualized results of the implemented detector with simulated laser scans. It shows the tested scenario within the simulation environment on the left side and the corresponding critical corners on the right side. It can be seen that the algorithm can detect all possible critical corners (corners of shelves), even if there is a person (left side of the center corridor) which also produces jumps in the laser scan. The prevention of such false positives (caused by people or other small obstacles) is achieved through the defined minimum occlusion length. Since the robot stands still, no critical corners are deleted to prevent false positives.

The Fig. 4 shows the detector's output with real-world laser scans. It can be recognized that the detector can find the critical corners (shown as red spheres) correctly, even if there is a person or are some cluttered contours – for instance caused by pallets.

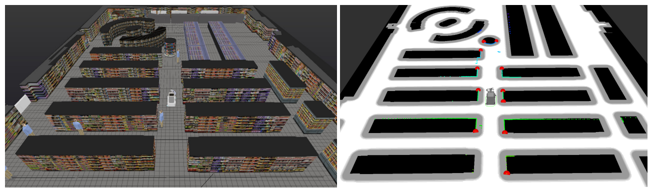
Fig. 3. Scenario within the simulated supermarket. Left: The simulation environment. Right: Visualization of the detected critical corners in red and the corresponding laser scans in green.

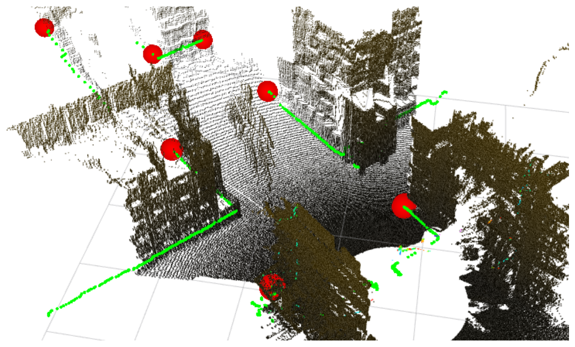
Fig. 4. Visualization of the detection node output with real laser scan data in a supermarket. For a better visibility, we added the appropriate 3D laser-scan as dark shaded points, which indicates the shelves and corridor. Detected critical corners are shown as red spheres. The 2D laser scan is colored in green.

B. Planner

1) Basic TEB Planner:

The TEB-planner [2], [3] is mainly a local planner[2], optimizing an initial (rough) plan, which might come from a global planner (e.g., an A* planner). In its core, the TEB-planner creates a factor graph where the robot poses are represented as nodes (variables) – the trajectory results from the consecutive poses. Therefore, all constraints – like maximum velocity, minimum distance to obstacles or holonomy – are formulated as edges (also called factors) within the factor graph. To get an optimal trajectory, the whole graph will be

[2]For completeness it should be mentioned that the TEB-planner can be used standalone, only passing a start and a goal pose to it [16].

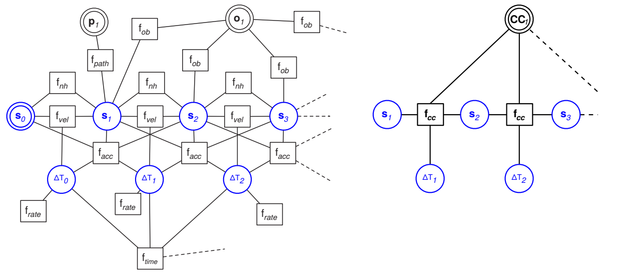

Fig. 5. Left: A possible factor graph without critical corners (based on Fig. 2b of [3]). Right: Example usage of our new multi-edge factor. Both: Rectangles represent edges (factors) and circles represent nodes (variables). Double circles are used for constant variables. For easier comparison basic nodes, which correspond to the trajectory, are colored in blue.

optimized regarding the minimum penalty of all constraints and the minimum total travel time or distance. The g2o framework [17] is used to solve this optimization problem.

The left part of Fig. 5 gives an overview of the factor graph used in the TEB planner. Circles like s_n represent the planned poses, p_n the path waypoints (given from a global planner) and o_n the obstacle position. Some common factors (rectangles) are visualized in Fig. 5 and represent the constraints like velocity, distance to obstacles, etc. All factors are mainly based on a general penalty function given in equation 1 that penalizes violation of a constraint. There, x_r denotes the penalty-bound, S represents a scaling factor, n the polynomial order and ε a small translation value.

$$e_\Gamma(x, x_r, \varepsilon, S, n) \simeq \begin{cases} \left(\frac{x-(x_r-\varepsilon)}{S}\right)^n & \text{if } x > x_r - \varepsilon \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Currently, the TEB planner [2], [3] has implemented constraints such as minimum distance to an obstacle or a fixed maximum velocity. A specific velocity constraint correlated with the distance to a critical corner (can be seen as a point obstacle) is missing. Therefore we need to define a new bound for the penalty function in equation 1, which leads to a new edge f_{cc} in the factor graph (right part of Fig. 5) – the critical corner edge.

2) Proposed Extension:

The new critical corner edge should slow down the robot if it is close to a critical corner. For this, it needs the distance between the robot and the critical corner. Thus, the new edge connects the position of the selected critical corner CC_j and the related pose of the robot x_i. Additionally, it needs to know the current velocity, which will be indirectly done by also including the next robot pose x_{i+1} and the time difference between both poses ΔT_i. The right part of Fig. 5 shows two applied factors of this new edge type.

With the basic idea of the edge functionality, we can define the exact relation between the velocity limit v_{limit} (the bound of the penalty function) and the distance to the critical corner d_{CC} in equation 2. Intuitively, v_{limit} is set equal to the given robot velocity cc_{vel}, if d_{CC} is equal predefined distance cc_{dist} – it can be seen as a minimum distance to a critical corner passed at maximum speed. The parameters cc_{vel} and cc_{dist} are indirectly relate to a worst-case time, in which

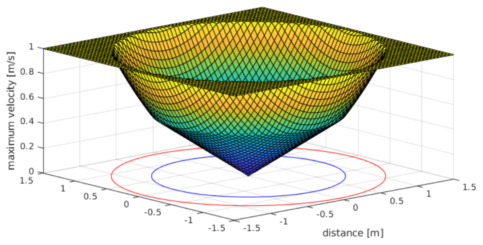

Fig. 6. Example of the allowed velocity limit where the critical corner is at $(0,0)$. The blue circle corresponds to the critical distance cc_{dist} where the velocity limit is equal to the critical velocity cc_{vel}. Within the blue circle, the critical corner is in full effect – limiting the velocity linear to the distance from the critical corner. Between the blue and the red circle, the critical corner will have some effect and the velocity limit will increase quadratically with the distance. At the red circle, the allowed velocity is equal to the global maximum velocity.

the robot might crash into something unseen. Based on this assumption, the robot can get closer to the critical corner if the worst-case time is guaranteed not to decrease. Hence, within the range of cc_{dist}, the edge limits the velocity linear based on the distance d_{CC}. To avoid that the robot is going full speed just a bit outside the range, a power function with a steeper slope is applied. For simplicity, we choose the square function, but others like the cubic do the same trick. The described velocity constraint v_{limit} depending on the distance to the critical corner d_{CC} can be visualized as seen in Fig. 6.

$$v_{limit}(d_{CC}) = cc_{vel} \cdot \left(\frac{d_{CC}}{cc_{dist}}\right)^\kappa, \kappa = \begin{cases} 1 & \text{if } d_{CC} < cc_{dist} \\ 2 & \text{otherwise} \end{cases} \quad (2)$$

After defining the new velocity constraint corresponding the distance to critical corners, we can create the cost function f_{cc} as follows (S and n are equal 1):

$$f_{CC} = e_\Gamma\left(v_j, v_{limit,j}\left(d_{CC,j}\right), \varepsilon, S, n\right) \quad (3)$$

It describes the penalty of a given velocity v_j and the corresponding velocity limit $v_{limit,j}$ (bound of the constraint) at the j_{th} pose of the factor graph. In Fig. 5 on the right side the factor f_{cc} is directly described with equation 3.

3) Fixing time differences in the TEB-planner: This section presents a minor issue with the original TEB planner and is not directly related to the critical corner extension. Nevertheless, we consider it worth mentioning for readers who work intensively with the TEB planner.

While extending and analyzing the TEB-planner, we recognized the difficulty to compare different trajectories visually. This is caused by the internal logic which automatically adds or removes poses before calling the optimizer to solve the factor graph. The described effect can be seen in Fig. 7 on the left side.

We provide a solution that tackles the problem by only changing the time differences[3]. We use the same thresholds for removing poses and for not changing anything. We

[3]See our pull request 263 within the original TEB planner repository `https://github.com/rst-tu-dortmund/teb_local_planner/pull/263`

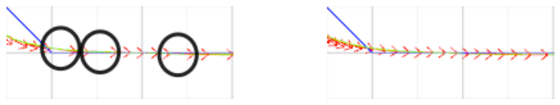
Fig. 7. Comparing the spacing of adjacent poses. Left: due to the internal logic of the TEB-planner, the poses have sometimes greater spacing. Right: our implementation solve the inequality. This was necessary to allow easier visual comparison between different generated trajectories.

also keep the insertion between adjacent poses, but only apply it when the time difference is twice the desired value. Additionally, we modify time differences in the remaining cases – we set the time difference to the desired value and add the missing time to the next pose, if possible. As a result, the optimizer can solve the factor graph without inequality-spaced poses.

IV. EXPERIMENTS

The following section presents two different show-cases: The first one represents standalone experiments of the planner extension (obstacles, critical corners and target are predefined - no sensors are used) and the second one deals with practical scenarios within our simulation environment and compares driving with and without considering critical corners. In summary: section IV-A focuses on the planning algorithm itself and section IV-B deals with both the detector and the planner under realistic conditions.

For our experiments with the navigation algorithm we used the following values of parameters:

- Original TEB-planner - used default parameters except:

no_inner_iterations	= 10
no_outer_iterations	= 10 (3 in simulation)
dt_hysteresis	= 0.05

- Critical Corner Extension:

dt_force_equal	= true
critical_corner_dist (cc_{dist})	= 1
critical_corner_vel (cc_{vel})	= 0.5
critical_corner_inclusion_dist	= 2
weight_cc	= 1

- Critical Corner Detector:

l_{min}	= 0.8
δ_{th}	= 1
δ_{tol}	= 0.4

We chose the planner parameter values cc_{dist} and cc_{vel} empirically. The parameter *critical_corner_inclusion_dist* describes the radius around a critical corner in which it is considered in the factor graph. The parametrization of the detector is based on the assumption that a person has roughly a diameter of 0.8 meters – it is approximated by the minimum occlusion length l_{min}. All remaining parameters are selected empirically, too.

A. Planner Standalone

We perform standalone experiments to evaluate the general performance of our extension to the TEB-planner. The aim is to determine whether the planner takes critical corners into account and adjusts the trajectory. In the following, we define two possible scenarios and discuss their results:

1) The robot turns into a supermarket corridor.
2) The robot's goal is straight ahead in a supermarket corridor and it crosses a junction.

In both scenarios, the planning algorithm should be able to handle critical corners by adjusting the trajectory.

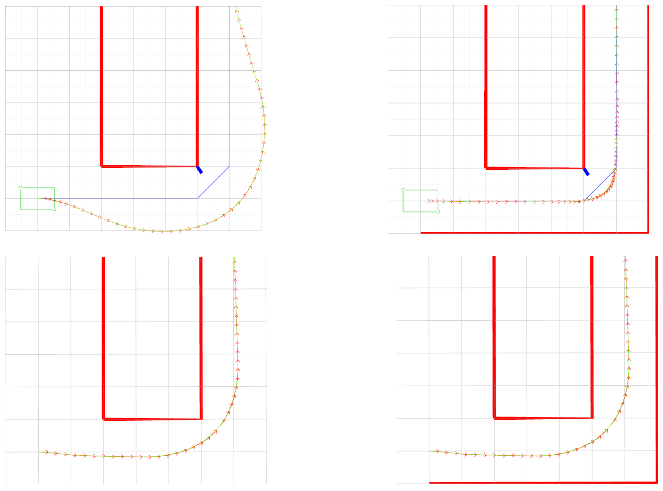
Fig. 8. Top-Left: Pass a critical corner and turn into a corridor without an additional wall. Top-Right: Pass a critical corner and turn into a corridor with a surrounding wall. Red lines indicate the obstacle's boundary, blue dots represent the critical corners and the red arrows show the single poses of the robot (distance between them relates to the corresponding velocity). Bottom-Left and Bottom-Right: original TEB-planner without considering critical corners.

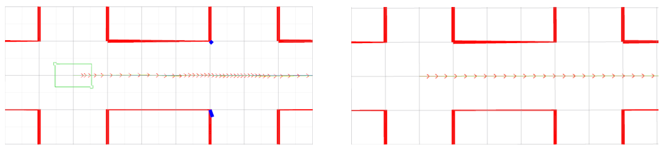
Fig. 9. Crossing a junction with two critical corners. The meaning of the colours is as in Fig. 8. Left: TEB-planner dealing with critical corner. Right: TEB-planner without considering critical corners.

Fig. 8 shows the results of scenario 1) in two possible variations. Both have an obstacle in the middle (e.g. a shelf), but they differ in the surrounding free space: In both left pictures of Fig. 8 is no other obstacle and the robot has as much space as possible during the turn. Therefore, the planner that considers critical corners pushes the poses away from the corner and the resulting trajectory becomes a curve (distance between the poses corresponds to the velocity because the time interval is the same). This happens because when the robot is closer to the critical corner, it is not allowed to drive fast. Since the execution time should be as short as possible, the distance between poses and critical corners increases to ensure higher speed. The lower picture on the left side represents the original planning behavior without taking critical corners into account. It shows a trajectory without sufficient distance to the corner at almost maximum speed (equidistant arrows).

A different situation is in the upper right image of Fig. 8, where an additional wall limits the free space, and the planner cannot increase the distance between the poses and the critical corner. In that case, the resulting trajectory

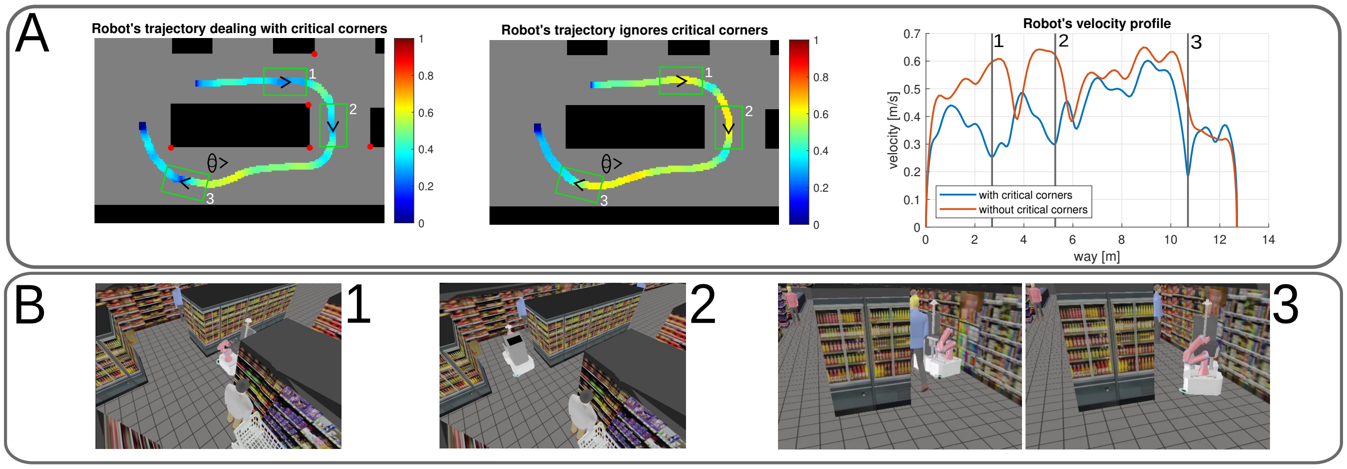

Fig. 10. **A:** Picture on the left and middle is the visualization of the robot's driven trajectories with and without handling critical corners. The colors along the trajectory represent the corresponding velocity (color bar in m/s). Dark areas represent shelves of the supermarket and light-gray areas free space (corridors). The green rectangles represent the robot's dimension in three different risky situations. Near pose number 3, there is a walking person that represents a dynamic obstacle. The red circles visualize critical corners which are in effect. The picture on the right shows the velocity profiles of the two different planning methods with the corresponding three risky situations (vertical lines). **B:** Stop-motion-pictures of the simulation at the three risky situations. Point 3 is shown with two pictures to visualize the consideration of the person as well.

comes closer to the corner while reducing the velocity. Such behavior increases the safety of a mobile robot, especially in environments with humans. The original planner in the lower right image does not satisfy this behaviour because it does not take blind spots into account.

The result of scenario 2) can be seen in Fig. 9, which also represents a common situation in a supermarket or other human environments. As in the second variation of the first scenario in Fig. 8 (with the additional wall), the planner cannot increase the distance between the poses and the critical corners. Furthermore, two corners influence the planning – one from the left and one from the right side. The result is a reduction of speed when passing these corners. The right part of Fig. 9 shows the planning result of the original TEB planner without critical corners. There, the robot is moving at maximum speed even when it crosses the junction with possible blind spots.

Finally, the results of the standalone experiments indicate the correct handling of critical corners by the TEB-planner. The next section will evaluate the detector and the planning algorithm together in a realistic simulation environment.

B. *Simulated Scenarios*

After evaluating the planning algorithm's basic functionality within the previous section, we combine the planner and the critical corner detector in a realistic scenario to analyze the overall performance. The difference to the section IV-A is that the planner takes critical corners into account even if they could disappear after passing the corner. When the robot is moving and critical corners are cleared (e.g. by the detector), the driven trajectory may differ from the planned path. Therefore, this section evaluates this more complex scenario.

Since we developed a simulation in [1], we can perform repeatable experiments in a standardized and consistent environment. This is important for a fair comparison of different navigation methods – in our case: The comparison of the TEB-planner with and without handling critical corners. For qualitative comparison, we define the start and goal position within the simulation and record the robot's poses over time. To demonstrate the ability to deal with visible dynamic obstacles as well, we added a walking person in the simulation that crosses the robot's path.

Fig. 10 contains the visualized results of two representative experiments with the appropriate velocity profile and stop motion pictures of the simulation environment. For intuitive understanding, we colored the individual positions based on the corresponding velocity. Hence, each colored point in the graphic represents the position of the robot during navigation, the color itself indicates the absolute value of the velocity (going from blue to red is going from slow to fast). Furthermore, we highlighted three different robot's position (green numbered rectangles) that represents situations of potential collisions with non-visible dynamic obstacles. Those positions are also marked in the velocity profile on the two lower graphs in Fig. 10.

Firstly, the images show that the TEB planner can generally handle visible dynamic obstacles – near position 3, a walking person within the simulation crosses the robot's path and the robot considers it without reducing speed (based on the person's predicted future path). Secondly, it can be seen that our proposed method produces safer trajectories compared to the planner without critical corners. At each of the three marked points, the planner with critical corners decreases the speed to reduce the collision risk with potential non-visible dynamic obstacles. In particular, the velocity profiles show that the original TEB planner does not consider critical corners and has high velocities at the important locations. On the website where we provide the source code, there are also videos of the simulated scenarios for better understanding.

V. CONCLUSIONS

Prior work has developed several possible strategies for solving the occlusion problem as described in this paper (critical corners imply occlusions). These approaches differ in adapting the environment's map, the robot's velocity or the trajectory itself. Such methods basically provide the functionality to incorporate non-visible dynamic obstacles but are not able to adapt their path to visible dynamic obstacles as in the TEB planner. In this work, we have closed the gap of a planner that considers visible and non-visible dynamic obstacles and modified the common TEB-planner, capable of dealing with static and dynamic obstacles, by extending its functionality to include critical corners (and the resulting blind spots). In addition to this extension, we also created a detector for critical corners, which indicates such blind spots, based on given raw laser scans and ray tracing methods. Experiments show that our method can handle occlusions and increase the safety of a mobile robot in both theoretical evaluations (section IV-A) and realistic experiments (section IV-B).

Either the planner tries to increase the distance to a critical corner (it improves the visibility and avoids surprising moments) or the planner decreases the velocity if the free space is limited. Future work includes transferring and evaluating the method to another domain with a different unstructured environment (not a supermarket).

REFERENCES

[1] K. Schlegel, P. Neubert, and P. Protzel, "Building a navigation system for a shopping assistant robot from off-the-shelf components," in *Towards Autonomous Robotic Systems (TAROS)*, 2020. [Online]. Available: https://www.tu-chemnitz.de/etit/proaut/publications/schlegel_2020.pdf

[2] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," *Robotik 2012*, pp. 74–79, 2012.

[3] ——, "Efficient trajectory optimization using a sparse model," in *2013 European Conference on Mobile Robots*. Barcelona, Catalonia, Spain: IEEE, Sep. 2013, pp. 138–143. [Online]. Available: http://ieeexplore.ieee.org/document/6698833/

[4] T. Kruse, A. Pandey, R. Alami, and A. Kirsch, "Human-Aware Robot Navigation: A Survey," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1726–1743, 2013. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01684295/document

[5] E. A. Sisbot, K. F. Marin-Urias, R. Alami, and T. Siméon, "A human aware mobile robot motion planner," in *IEEE Transactions on Robotics*, vol. 23, no. 5, oct 2007, pp. 874–883.

[6] M. Sadou, V. Polotski, and P. Cohen, "Occlusions in obstacle detection for safe navigation," in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2004, pp. 716–721.

[7] K. M. Krishna, R. Alami, T. Simeon, K. M. Krishna, R. Alami, and T. Simeon, "Safe proactive plans and their execution," *Robot. Auton. Syst.*, vol. 54, no. 3, pp. 244–255, 2006.

[8] S. Suri and J. O'Rourke, "Worst-case optimal algorithms for constructing visibility polygons with holes," *Proceedings of the 2nd Annual Symposium on Computational Geometry, SCG 1986*, pp. 14–23, 1986.

[9] W. Chung, S. Kim, M. Choi, J. Choi, H. Kim, C. B. Moon, and J. B. Song, "Safe navigation of a mobile robot considering visibility of environment," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 3941–3950, 2009.

[10] P. F. Orzechowski, A. Meyer, and M. Lauer, "Tackling Occlusions Limited Sensor Range with Set-based Safety Verification," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-Novem, pp. 1729–1736, 2018.

[11] M. Lee, K. Jo, and M. Sunwoo, "Collision risk assessment for possible collision vehicle in occluded area based on precise map," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-March, pp. 1–6, 2018.

[12] M. Y. Yu, R. Vasudevan, and M. Johnson-Roberson, "Occlusion-aware risk assessment for autonomous driving in urban environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2235–2241, 2019.

[13] H. Andersen, W. Schwarting, F. Naser, Y. H. Eng, M. H. Ang, D. Rus, and J. Alonso-Mora, "Trajectory optimization for autonomous overtaking with visibility maximization," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-March, pp. 1–8, 2018.

[14] H. G. Jung, Y. H. Cho, P. J. Yoon, and J. Kim, "Scanning laser radar-based target position designation for parking aid system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 3, pp. 406–424, 2008.

[15] S. Lange and et al., "Two autonomous robots for the dlr spacebot cup -lessons learned from 60 minutes on the moon," *International Symposium on Robotics, ISR*, 2016.

[16] C. Rösmann, F. Hoffmann, and T. Bertram, "Planning of multiple robot trajectories in distinctive topologies," in *2015 European Conference on Mobile Robots (ECMR)*. Lincoln, United Kingdom: IEEE, Sep. 2015, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/7324179/

[17] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*. Shanghai, China: IEEE, May 2011, pp. 3607–3613. [Online]. Available: http://ieeexplore.ieee.org/document/5979949/