Predictive Collision Avoidance for the Dynamic Window Approach

Marcell Missura Maren Bennewitz

***Abstract*— Foresighted navigation is an essential skill for robots to rise from rigid factory floor installations to much more versatile mobile robots that partake in our everyday environment. The current state of the art that provides this mobility to some extent is the Dynamic Window Approach combined with a global start-to-target path planner. However, neither the Dynamic Window Approach nor the path planner are equipped to predict the motion of other objects in the environment. We propose a change in the Dynamic Window Approach—a dynamic collision model—that is capable of predicting future collisions with the environment by also taking into account the motion of other objects. We show in simulated experiments that our new way of computing the Dynamic Window Approach significantly reduces the number of collisions in a dynamic setting with nonholonomic vehicles while still being computationally efficient.**

I. Introduction

There is a rising demand for a new class of robots—autonomous robots—that are able to safely locomote in human environments and perform a large variety of tasks. A few prominent examples are self-driving cars, package delivery drones, and domestic service robots. The demand for efficient and collision-free locomotion in such dynamic settings is only partially met by what navigational software can do today. Most of the state-of-the-art motion planners still regard obstacles in the environment as stationary.

The state of the art constitutes of shortest path finding in combination with a low-level motion controller for path tracking and collision avoidance. The work of Willow Garage [1] describes an extensive test of this control paradigm. The used shortest path finder is a vanilla A* algorithm in an occupancy map. An intermediate target along this path then serves as a local goal for the Dynamic Window Approach (DWA) that computes velocity commands for the robot. This navigation stack is widely available in ROS [2].

The DWA [3] uses circular arcs to approximate the otherwise complex trajectories of nonholonomic vehicles. After systematic sampling of such arcs, the best of them is elected according to an objective function that evaluates progress towards the target and the clearance from obstacles. The selected arc then yields the next velocity command that is sent to the robot, and the entire process is reiterated. The Global Dynamic Window Approach [4] uses an NF1 navigation function to guide the DWA. The NF1 function is a precomputed lookup table of the length of the shortest path from every cell of an occupancy grid to the target. The NF1 table captures obstacles only as a static snapshot and it is too costly to be updated at a high frequency. Ögren et al. [5] harnessed the global DWA scheme presented above and proved with a few modifications in a model predictive control setting convergence to the goal.

Both authors are with Humanoid Robots Lab, University of Bonn, Germany

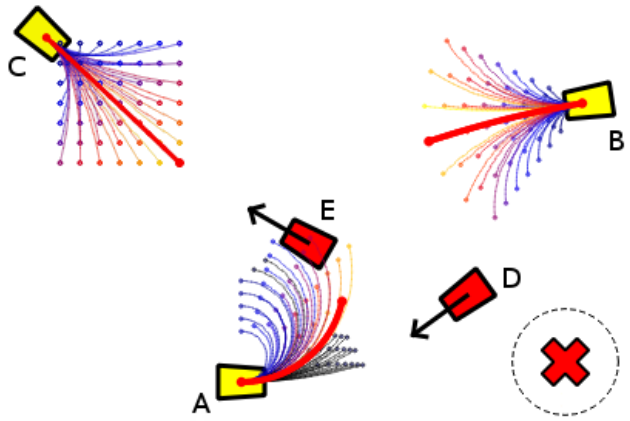
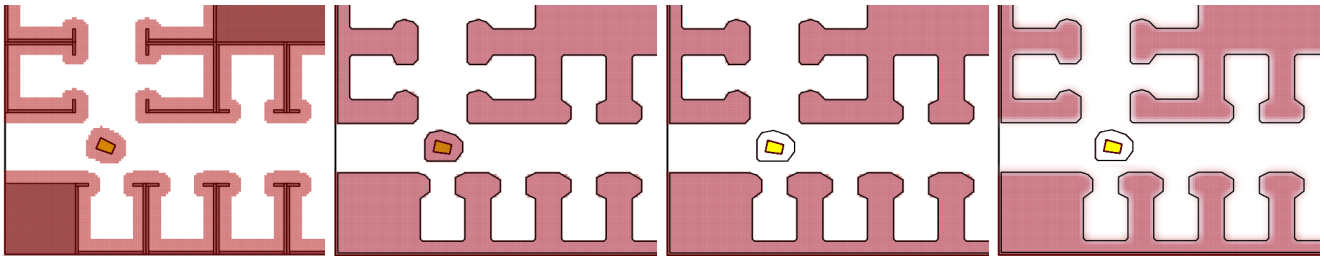


Fig. 1: Nonholonomic agent A detects future collisions with agent D and predictively adjusts its course while heading for the target location marked with the red cross. The sampled trajetories of agent A do not indicate a collision with agent E, since it will have moved out of the way by the time agent A reaches the end of the trajectories. Agent B is using circular trajectories, agent C is using a holonomic model.

Earlier versions of navigational software relied on the force field method [6] [7]. Obstacles in the environment exert a repulsive force on the agent while the target is an attractor. However, the purely reactive nature of this approach renders foresighted motion out of reach. This was followed by a trend heavily based on different flavours of the A* algorithm. Some approaches rely on heuristic search of control actions for cars [8] and for quadrocopters [9]. Computational feasibility is achieved by a discretization of the state space rather than the control space with the help of motion primitives that move the vehicle between known points on a lattice. Another class of planners attempts to refine an A* path to a dynamically feasible motion trajectory on the fly. Quintic splines [10], B-splines [11], Bezier curves [12], or elastic bands [13] are used in a manner of online optimization. While the path to begin with is collision free, the refined trajectory is not necessary so, especially not if the obstacles move between updates. In all of the aforementioned methods moving obstacles are assumed to be stationary. When the motion plan is replanned with a high enough frequency, it may be sufficient for obstacle avoidance at low velocities, but foresighted evasive maneuvers are not possible.

With the advent of deep learning, in particular in combination with reinforcement learning, machine learned controllers are on the rise that can account for motion. In [14], an end-to-



(a) Dilation (b) Contour detection (c) Removal (d) Blur

Fig. 2: The steps of our assumed perception pipeline that are needed to compute grid and polygonal representations of static and dynamic objects. The yellow object is a robotic agent in motion. a) First, an occupancy map is computed from the raw sensor data and dilated by the radius of the robot. b) Using contour detection, polygons are computed from the occupancy map. c) Cells that belong to moving objects are removed, but the polygons remain. d) A blur filter smoothes the inflated occupancy map.

end motion planner was learned for small, static maps using the raw sensor data as input, the motor controls as output, and a ROS path planner as the expert teacher. In [15] and [16] obstacles were represented by their position, velocity, and radius before being fed as input into a network. The fact that the variable number of obstacles is not a fixed sized input is handled by an LSTM layer. This system appears to work well and it does account for the motion of obstacles, but it is limited to small circular obstacles so far. In [17] inverse reinforcement learning was used to identify the parameters of human trajectories that were modeled by cubic splines. The result was a predictive system with human-like behavior.

An approach to obstacle avoidance most related to ours that explicitly models the motion of other objects is based on the concept named Reciprocal Velocity Obstacles [18], [19], [20]. Velocity obstacles are geometric regions in velocity space that describe the set of velocities an agent A is not permitted to use in order to avoid collision with agent B within a fixed time window. This geometric formulation allows efficient computation of velocity commands using a linear program. However, a number of assumptions need to be made that may impair the performance. The agent and the obstacles are assumed to be holonomic, both travel with a constant velocity, and all participants make an effort to avoid a collision using the same controller. Nonholonomic control [20] is achieved by shifting the reference point of the vehicle away from the center (and thereby also enlarging the effective radius) to a point that can be fully controlled.

Our method makes the constant velocity assumption only for the obstacles in the environment which are modeled as moving polygons with a velocity attached. For the agent we use trajectories that approximate accelerated nonholonomic motion and intersect these with the moving polygons. The output of our controller are nonholonomic accelerations rather than velocity, which results in smoother steering and physical feasibility. We implemented two versions of the DWA in this setting, one using the traditional circular arcs, and another where we convert to a holonomic model and exploit its simplicity. In both versions, we predict collision points between the trajectory of the vehicle and the edges of the moving polygons in order to assess the quality of the sampled trajectories, and select the best one to derive the next incremental control.

II. World Modeling

For the obstacle avoidance concept presented in this contribution to work as intended, we make the following assumptions. We assume that we have a dilated and blurred occupancy map available that contains only stationary objects. Furthermore, stationary and moving objects are both represented as polygons as well. The processing steps of our assumed perception pipeline, which for now only exists as a mock-up in simulation, are sketched in Figure 2.

An occupancy grid can be directly obtained from sensor data by marking blocked cells that fall onto objects in the sensor range of the robot. A dilation filter can then be used to expand clusters of blocked cells by the radius of the robot (or a little more). After the obstacles have been inflated this way, the robot can be regarded as a point. Polygons can be gained from the dilated occupancy map using contour detection. We assume that by associating polygon observations over multiple frames, we are able to measure the velocities of the polygons and can thus group them into two classes: stationary and moving. The cells corresponding to moving polygons are removed from the occupancy grid. A simple box blur filter then smoothes the remaining occupancy grid. Dilation, blurring, and contour detection algorithms are readily available in the OpenCV library. The association of polygons over multiple frames can be handled by the Hungarian Algorithm and their tracking, i.e., velocity estimation, can be done by Kalman filtering.

Our system exploits each part of this hybrid world representation in a specific way in order to be maximally efficient. The blurred occupancy grid is used for fast collision checking with stationary objects. The blurring results in careful driving with a comfortable clearance from the obstacles that can be sacrificed when having to move through narrow gaps. The polygons of the stationary objects are used for shortest path finding in a polygonal scene with the Minimal Construct algorithm [21], a shortest path planner designed for polygonal scenes. Finally, the moving polygons are used for dynamic collision checking with our modified DWA.

III. The Dynamic Window Approach

The core philosophy of the DWA algorithm is to sample a set of control parameters and to predict their outcome when applied to the robot in its current state of motion, assuming

the controls stay constant for a short time. The resulting trajectories are then evaluated according to an objective function that captures obstacle clearance and progress towards a target. The controls of the best trajectory elected by the objective function are then passed on to the robot to produce the next motion increment. This entire process is repeatedly executed while the robot is in motion. Figure 1 shows an illustration of a robotic agent using the DWA.

A. Trajectory Sampling

In order to model the nonholonomic motion of wheeled robots, we regard the unicycle model. The state of the unicycle is defined by the vector $\mathbf{s}_u = (x, y, \theta, v, \omega)$, where (x, y) are the Cartesian coordinates, θ is the heading of the robot, v is the linear velocity, and ω is the angular velocity. If we assume constant accelerations a_v and a_ω to be the control parameters where a_v is the linear acceleration and a_ω is the angular acceleration, then the motion of the unicycle is described by the differential equation

$$(\dot{x}, \dot{y}, \dot{\theta}, \dot{v}, \dot{\omega}) = (v\cos\theta, v\sin\theta, \omega, a_v, a_\omega). \tag{1}$$

The integration of this equation [22] [23] is not well suited for fast collision checking. The classic version of the DWA [3] samples pairs of linear and angular velocities (v, ω) as controls that describe circular arcs, which are only a special case of nonholonomic trajectories, but they can be used for the approximation of a future state

$$\mathbf{s}_u(v, \omega, t) = \mathbf{s}_{u_0} + \begin{pmatrix} \frac{v}{\omega}(\sin(\theta + \omega t) - \sin(\theta)) \\ \frac{v}{\omega}(\cos(\theta + \omega t) - \cos(\theta)) \\ \omega t \\ 0 \\ 0 \end{pmatrix}, \tag{2}$$

and they can be intersected with polygons quite efficiently. We adopt the circular arcs in our implementation, but we maintain acceleration control parameters by creating the following relation. We generate a set of control pairs

$$\begin{aligned} C = \{(a_{v_i}, a_{\omega_j}) | i, j \in \{0, ..., N-1\}, \\ a_{v_i} = -A + i\frac{2A}{N-1}, \\ a_{\omega_j} = -B + j\frac{2B}{N-1}\}, \end{aligned} \tag{3}$$

where $A = 20\,\frac{m}{s^2}$ is the bound of the linear acceleration, $B = 10\,\frac{rad}{s^2}$ is the bound of the angular acceleration, and N is the number of samples per dimension. We set $N = 7$ and obtain 49 control pairs in total. We then convert the acceleration pairs to a set of velocities

$$\begin{aligned} C' = \{(v_i, \omega_j) | i, j \in \{0, ..., N-1\}, \\ v_i = v_0 + \delta a_{v_i} T, \\ \omega_j = \omega_0 + \delta a_{\omega_j} T\}, \end{aligned} \tag{4}$$

where (v_0, ω_0) is the current velocity of the robot, $T = 300\,ms$ is the total time horizon of the DWA, and $\delta = 0.5$ is a tuning parameter that determines at which portion of $[v_0, v_T]$ and $[\omega_0, \omega_T]$ the velocities are taken. When we elect the best circular arc (v_k, ω_l) with the objective function, we can relate to the assigned acceleration (a_{v_k}, a_{ω_l}) and use it to control the vehicle. Note that we also enforce a velocity bound $V = 5\,\frac{m}{s}$ by setting the accelerations to zero, if the robot is already at the velocity limit.

B. Objective Function

Each of the velocity pairs in C' is evaluated with respect to the objective function

$$\begin{aligned} F(v, \omega) = \alpha\,\mathrm{gridclearance}(v, \omega) \\ + \beta\,\mathrm{polygonclearance}(v, \omega) \\ + \gamma\,\mathrm{progress}(v, \omega). \end{aligned} \tag{5}$$

Our objective function F consists of three parts, including two types of clearance functions. The grid clearance function

$$\begin{aligned} \mathrm{gridclearance}(v, \omega) = -\max\{\mathrm{grid}(\mathbf{s}_u\,(v, \omega, t_k))\,| \\ k \in \{1, ..., K\}, t_k = k\frac{T}{K}\} \end{aligned} \tag{6}$$

captures the proximity of static obstacles that are represented by our blurred occupancy map. It is computed by predicting the location of the controlled agent using Eq. (2) at times t_k sampled from the interval $[0, T]$ and evaluating the occupancy grid in these points. Since we dilate the occupancy grid, obstacles are relatively thick and a low number of samples is sufficient. In our implementation, we use only $K = 2$ samples—one in the middle of the trajectory and one at the end point. The evaluation of the occupancy map with the $\mathrm{grid}()$ function amounts to a table lookup that returns a blurred occupancy value in the $[0, 1]$ interval.

The polygonal clearance function $\mathrm{polygonclearance}()$ returns the collision time $t_c \in [0, T]$, which indicates at what time in the future the first collision will occur with any of the edges of the moving polygons. If no collision occurs at all, or the first collision occurs after the DWA time ($t_c > T$), t_c is set to T. The details of the polygonal clearance function are explained in Sec. III-C after the presentation of the objective function. The collision time is then converted to a clearance indicator $\frac{t_c}{T} \in [0, 1]$.

The progress indicator

$$\mathrm{progress}(v, \omega) = 1 - \frac{|\mathbf{s}_u\,(v, \omega, T) - g|}{\arg\max_{(v_i, \omega_j) \in C'} |\mathbf{s}_u\,(v_i, \omega_j, T) - g|} \tag{7}$$

computes the normalized Euclidean distance between the end point of a trajectory and a target location g that is obtained from the path planner. Each component of the objective function is a normalized value in the $[0, 1]$ range. This makes it easy to weight them with the parameters α, β, and γ in Eq. (5). We set $\alpha = 0.8$, $\beta = 1.0$, and $\gamma = 0.5$.

Note the difference between our objective function and the one originally suggested in [3]. We do not use the heading or the velocity of the robot to measure progress, but simply use the Euclidean distance to an intermediate target instead. Our grid clearance is essentially the same as in [3], except that we take the maximum of only two samples. The predictive component for the polygon clearance is new.

Furthermore, in the classic version, inadmissible trajectories are discarded before being evaluated by the objective function. We argue that it is harmful to discard inadmissible trajectories, because cases may occur where choosing an inadmissible trajectory is the only option. For example, if an object suddenly appears in front of the agent that has not been sensed before—this could be a child chasing a ball that just rolled out onto the street—and a collision seems unavoidable, it is still better to aim for the best inadmissible trajectory that maximizes the time until collision and possibly also minimizes damage by allowing more time to decelerate, than the controller not knowing what to do at all. Instead of the admissibility check, we differentiate between trajectories that are in collision, i.e., $t_c < T$, and ones that are not. We elect the best candidate according to Eq. (5) only among the trajectories that are not in collision. If there is no such trajectory, then we set $\gamma = 0$ and choose the best inadmissible trajectory with the largest clearance.

C. Dynamic Collision Checking

The polygonal clearance function $\mathrm{polygonclearance}()$ used in Eq. (5) computes the time t_c of a future collision between the agent traveling along the circular arc implied by the unicycle state $\mathbf{s}_u = (x, y, \theta, v, \omega)$ and a set of moving polygons. To determine the earliest time of a collision, or whether a collision occurs at all, an intersection has to be computed with every edge of every polygon in the set.

Let $(p, q) = (p_x, p_y, q_x, q_y)$ be a polygon edge defined by its end points and $v_P = (v_{P_x}, v_{P_y})$ the velocity of the polygon P. To compute a dynamic intersection with this edge, we first rectify the task by centering the coordinate system on the center of the circular arc, rotating around the center such that the edge becomes horizontal, and normalizing by the radius of the arc. The center of the arc is given by

$$c = \begin{pmatrix} x \\ y \end{pmatrix} + R\left(\theta + \frac{\pi}{2}\right) \begin{pmatrix} r \\ 0 \end{pmatrix}, \tag{8}$$

where R is a rotation matrix and $r = \frac{v}{\omega}$ is the radius of the arc. Let $\psi = \arctan\left(\frac{q_y - p_y}{q_x - p_x}\right)$ be the angle of the polygon edge with respect to the x-axis. Then the rectified edge and its velocity are

$$(p', q') = \frac{1}{r} R(-\psi)\,(p - c, q - c), \tag{9}$$

$$v'_p = \frac{1}{r} R(-\psi)\, v_p. \tag{10}$$

Furthermore, we need the orbit angle $\phi = \theta - \psi - \mathrm{sgn}(r)\frac{\pi}{2}$ of the position of the robot around the arc center c with respect to the horizontal. This is illustrated in Figure 3. Then, the time t_c of collision between the edge and the robot is the root of the equation

$$\sin(\phi + \omega t_c) - v'_{P_y} t_c - p'_y = 0. \tag{11}$$

Unfortunately, we cannot solve for the root algebraically. Instead, we split up the arc at $\pm\frac{\pi}{2}$ into y-monotonic segments and use three iterations of the Illinois algorithm to find the root of each segment in order. If a $t_c < T$ is found, we need

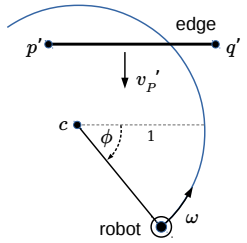


Fig. 3: Dynamic collision check between the moving rectified polygon edge (p', q') with velocity v'_p and a circular trajectory around the center c.

to check if $p'_x < \cos(\phi + \omega t_c) - v'_{p_x} t_c < q'_x$ to make sure that the collision point is between the left and right boundaries of the rectified edge, assuming $p'_x < q'_x$. We can abort as soon as we find the first root. A bounding box test is effective at quickly discarding cases that cannot possibly intersect before the numerical method is run. In all discarded cases, and also if $t_c > T$, we set $t_c = T$.

IV. Holonomic Model

The need for a numerical method to intersect circular arcs with moving edges is unsatisfying. Thus, we also consider a computationally efficient model—a holonomic one [24].

A. Trajectory Sampling

Holonomic motion in a 2D plane is characterized by the differential equation

$$(\dot{x}, \dot{y}, \dot{v}_x, \dot{v}_y) = (v_x, v_y, a_x, a_y), \tag{12}$$

where the state of motion is $\mathbf{s}_h = (x, y, v_x, v_y)$ with (x, y) the Cartesian coordinates in the plane and (v_x, v_y) the respective velocities along the x and y axes. This system does not have an explicit orientation, even though the direction of the motion $\theta = \arctan(\frac{v_y}{v_x})$ does imply a heading. The control parameters are the accelerations (a_x, a_y). This time, we can easily integrate the equation of motion (12) and obtain the state prediction function

$$\mathbf{s}_h(a_x, a_y, t) = \begin{pmatrix} x + v_x t + \frac{1}{2} a_x t^2 \\ y + v_y t + \frac{1}{2} a_y t^2 \\ v_x + a_x t \\ v_y + a_y t \end{pmatrix}. \tag{13}$$

The sampling of controls for the holonomic model in a DWA fashion becomes

$$\begin{aligned} C = \{(a_{x_i}, a_{y_j}) | i, j \in \{0, ..., N-1\}, \\ a_{x_i} = -A + i\frac{2A}{N-1}, \\ a_{y_j} = -A + j\frac{2A}{N-1}\}, \end{aligned} \tag{14}$$

where A and N are the same parameters as before.

B. Objective Function

The objective function remains the same as equation (5), except that $F(a_x, a_y)$ now evaluates holonomic acceleration inputs. The state prediction function that the $\mathrm{gridclearance}()$ and $\mathrm{progress}()$ components use is now Eq. (13) instead of Eq. (2), i. e., $\mathbf{s}_h$ instead of $\mathbf{s}_u$.

C. Dynamic Collision Checking

The most intriguing part of the holonomic model is the easiness of the implementation of the $\mathrm{polygonclearance}()$ function. Again, an intersection has to be computed for every polygonal edge to find the earliest time of collision t_c. Let $(p, q) = (p_x, p_y, q_x, q_y)$ be a polygon edge and $v_P = (v_{P_x}, v_{P_y})$ the velocity of the polygon P. In the holonomic case, we can subtract v_P from the velocity of the robot in order to regard the polygon edge as stationary and set $\left(v'_x, v'_y\right) = \left(v_x - v_{P_x}, v_y - v_{P_y}\right)$. We then express the equation of a collision with a line in a normal form

$$y(t) = ax(t) + b. \tag{15}$$

and set $a = \frac{q_y - p_y}{q_x - p_x}$ and $b = p_y - ap_x$. Then, we use Eq. (13) with (v'_x, v'_y) to expand Eq. (15) and set $t = t_c$ to obtain

$$y + v'_y t_c + \frac{1}{2} a_y t_c^2 = a\left(x + v'_x t_c + \frac{1}{2} a_x t_c^2\right) + b. \tag{16}$$

Solving for t_c yields the time of collision

$$t_c = \frac{(v'_y - av'_x)}{(a_y - aa_x)} \pm \sqrt{\frac{(v'_y - av'_x)^2}{(a_y - aa_x)^2} - \frac{2(y - ax - b)}{(a_y - aa_x)}}. \tag{17}$$

From the two possible solutions of Eq. (17), we are interested in the smaller positive one. If it exists and $t_c < T$, we check if $p_x < x + v'_x t_c + \frac{1}{2} a_x t_c^2 < q_x$, again assuming $p_x < q_x$. If this condition is true, we identified a collision time t_c. Otherwise—also when there is no solution—we set $t_c = T$ and assume no collision.

D. Conversion

In order to harness the holonomic model to control a nonholonomic vehicle, a conversion must be made. The conversion from the unicycle input state $\mathbf{s}_u = (x_u, y_u, \theta_u, v_u, \omega_u)$ to a holonomic state $\mathbf{s}_h = (x_h, y_h, v_{x_h}\, v_{y_h})$ is given by $\mathbf{s}_h = (x_u, y_u, \cos(\theta_u)\, v_u, \sin(\theta_u)\, v_u)$. Then, $\mathbf{s}_h$ can be used to execute the holonomic DWA. This yields the holonomic acceleration (a_x, a_y) that maximizes the objective function. However, it needs to be converted back to a reasonable nonholonomic control signal (a_v, a_ω). Since $v_u(t) = \sqrt{(v_{x_h} + a_x t)^2 + (v_{y_h} + a_y t)^2}$, it follows that $a_v = \frac{d}{dt} v_u(t)$, i. e.,

$$a_v = \frac{v_{x_h} a_x + v_{y_h} a_y}{\sqrt{v_{x_h}^2 + v_{y_h}^2}}. \tag{18}$$

The angular acceleration a_ω is a little bit more tricky. Since $\theta_u(t) = \arctan(\frac{v_{y_h} + a_y t}{v_{x_h} + a_x t})$ is the implied direction of the holonomic motion, the instantaneous angular velocity must be $\frac{d}{dt}\theta_u(t)$, i. e.,

$$\omega_h = \frac{v_{x_h} a_y - v_{y_h} a_x}{v_{x_h}^2 + v_{y_h}^2}. \tag{19}$$

However, since (a_x, a_y) is the output of the DWA and it is not continuous, ω_h is not continuous either. We obtain a_ω by computing

$$a_\omega = \frac{\omega_h - \omega_u}{\sigma}, \tag{20}$$

with $\sigma = 0.01\,\mathrm{s}$, the time period of the main control loop, and ω_u, the current angular velocity of the nonholonomic vehicle. We then bound the acceleration parameters using the bounds A, B, and V. This way, we obtain a controller that steers as fast as it can towards the angular velocity the DWA suggests, but maintains continuous curvature.

V. Experimental Results

We have performed extensive simulation experiments in order to test our implementation of the DWA. We used two different maps that are shown in Figure 4. The map on the left is void of static obstacles and isolates the capability of the controllers to deal with moving obstacles. The map on the right is an office environment with narrow passages that are more difficult to navigate. The accompanying video[1] gives a visual impression of these experiments.

We successively filled the maps with up to ten agents. The agents were started in a random configuration and commanded to drive to a goal location. Whenever an agent reached its goal, it randomly picked a new goal and continued driving. One agent was chosen to test four different versions of obstacle avoidance—an arc-based DWA with predictive collision avoidance and without, and a holonomic DWA with predicitve avoidance and without. All other agents were always driven by an arc-based predictive controller. We simulated ten runs of five minutes with each type of controller and each number of agents in the map, i. e., testing one controller amounted to 500 minutes simulation time. Whenever we switched the type of controller, we reseeded the random number generator with the same seed so that each controller would be faced with the same initial configuration and with the same sequence of goal locations. We recorded the number of times the observed agent collided with something, the number of targets it reached, and the average runtime of the DWA controllers. We computed the average and the standard deviation of these figures over the ten runs each controller was evaluated for.

The results are shown in Figure 5. The predictive and static versions of the holonomic controllers are paired in blue color, the arc-based controllers in orange and red color. In the first row, we can observe that in both maps the predictive controllers reach more goals than their static counter parts and the difference is increasing with the number of agents in

[1] Video: https://youtu.be/Y14CAtCNBDE

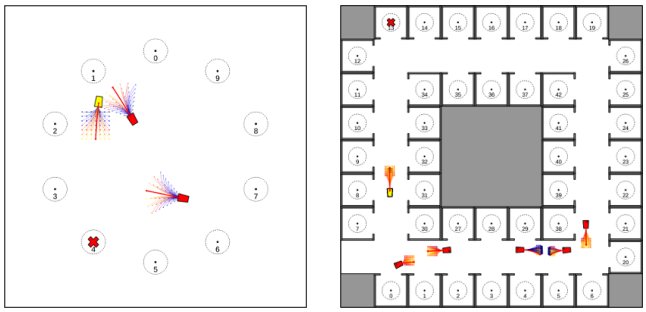


Fig. 4: The maps that were used in our experiments. Left: Void map. Right: Office map.

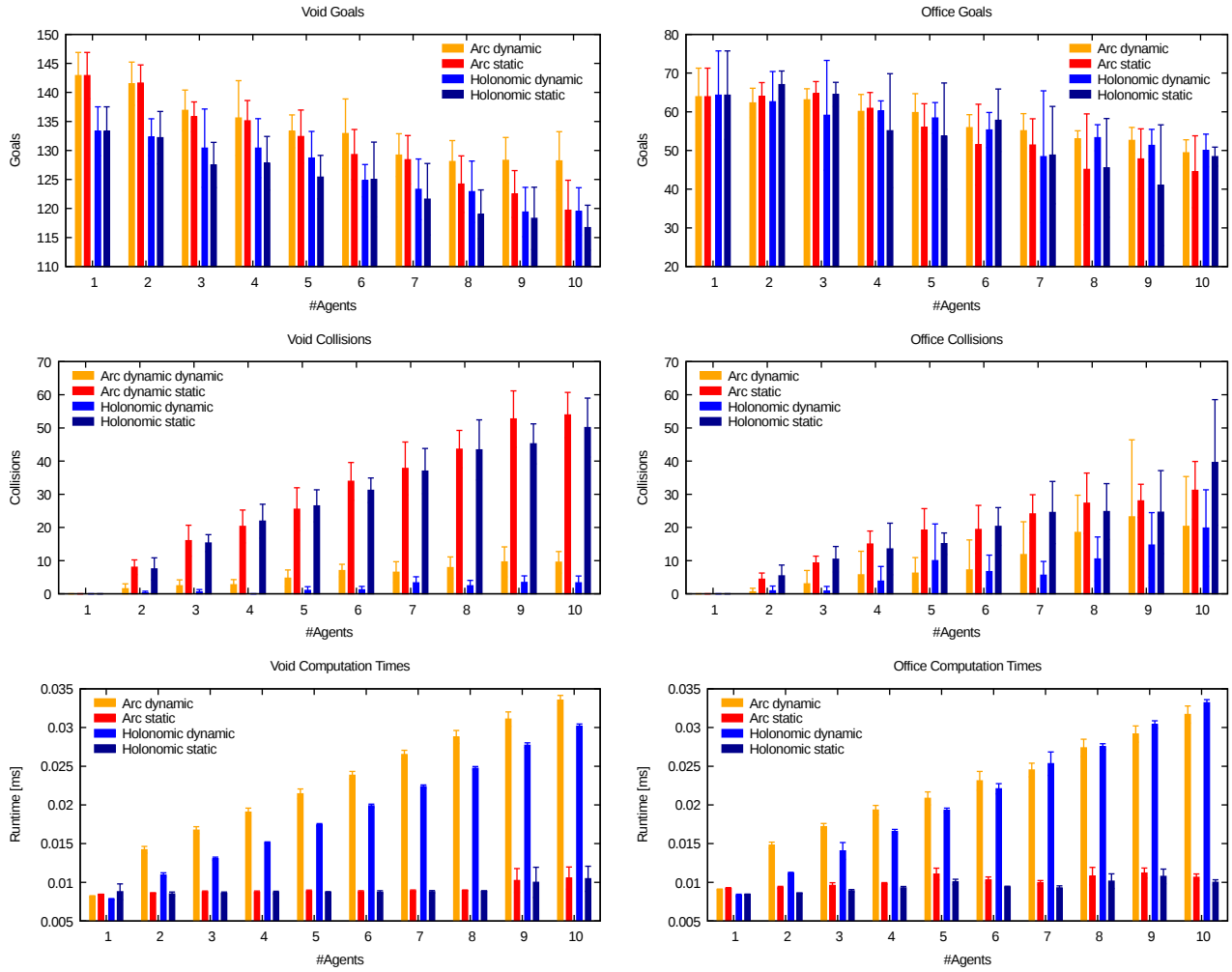


Fig. 5: Data recorded during our experiments (mean and standard deviation). The left column shows the results obtained in the void map, the right column the results of the office map (see maps in Fig. 4). Most notably, the number of collisions is drastically reduced when using predictive collision avoidance considering the motion of dynamic obstacles.

the map. The arc-based controllers appear to perform better than the holonomic ones in terms of reaching goals.

In the second row, we can see that the predictive versions dramatically reduce the number of collisions. The remaining collisions typically occur in situations where agents approach each other so closely that they enter each others' inflation zones where the DWA is not effective. We switch to a force field method in this case that attempts to push the vehicles apart, but it also induces turning that may result in light touches between the agents. In terms of collision avoidance, the holonomic controller seems to perform a bit better. All four controllers are able to navigate the office map without a single collision as long as there are no other agents.

The computation times shown in the third row are in the order of tens of microseconds. The static runtimes remain constant while the runtimes of the dynamic controllers scale linearly with the number of moving obstacles present. Most interestingly, despite the numeric method that was needed to compute intersection with the arcs, arc-based collisions can be computed just as fast as holonomic ones. We attribute this phenomenon to the rectification step that rotates the polygon edges into a horizontal position. This way, the bounding box checks are more effective at discarding non-colliding cases since they only need to check a box against a line as opposed to the holonomic case, where the bounding box of the trajectory is checked against the bounding box of a slanted line. We can expect to still be in the millisecond range when extrapolating to 1000 obstacles, which would undoubtedly saturate the sensory range of a robot.

VI. Conclusions

We have introduced a new version of predictive a DWA controller that explicitly models moving objects as polygons with velocities and computes future collisions. In our experiments, we demonstrated that the predictive nature of our DWA implementation is highly effective at reducing collisions and improves the all around navigational performance at very little computational cost. We compared two distinct types of motion models—circular arcs and a holonomic model—and found no significant difference to report. In future work, we intend to adopt a precise nonholonomic model [22] into our framework and to evaluate our system in real-world scenarios.

REFERENCES

[1] E. Marder-Eppstein, E. Berger, T. Foote, B.P. Gerkey, and K. Konolige. The office marathon: Robust navigation in an indoor office environment. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[2] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Robotics*, 2009.

[3] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, Mar 1997.

[4] Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 341–346, 1999.

[5] P. Ögren and N. E. Leonard. A convergent dynamic window approach to obstacle avoidance. *IEEE Transactions on Robotics*, 21(2):188–195, April 2005.

[6] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505, March 1985.

[7] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, Oct 1992.

[8] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

[9] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. *CoRR*, abs/1709.05401, 2017.

[10] B. Lau, C. Sprunk, and W. Burgard. Kinodynamic motion planning for mobile robots using splines. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2009.

[11] Vladyslav C. Usenko, Lukas von Stumberg, Andrej Pangercic, and Daniel Cremers. Real-time trajectory replanning for mavs using uniform b-splines and 3d circular buffer. *CoRR*, abs/1703.01416, 2017.

[12] Sterling McLeod and Jing Xiao. Real-time adaptive non-holonomic motion planning in unforeseen dynamic environments. In *IROS*, pages 4692–4699. IEEE, 2016.

[23] Alonzo Kelly and Bryan Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *I. J. Robotics Res.*, 22(7-8):583–602, 2003.

[13] C. Rösmann, F. Hoffmann, and T. Bertram. Kinodynamic trajectory optimization and control for car-like robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, pages 5681–5686, Sept 2017.

[14] Mark Pfeiffer, Michael Schaeuble, Juan I. Nieto, Roland Siegwart, and Cesar Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. *CoRR*, abs/1609.07910, 2016.

[15] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P. How. Socially aware motion planning with deep reinforcement learning. In *IROS*, pages 1343–1350. IEEE, 2017.

[16] Michael Everett, Yu Fan Chen, and Jonathan P. How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. *CoRR*, abs/1805.01956, 2018.

[17] Henrik Kretzschmar, Markus Spies, Christoph Sprunk, and Wolfram Burgard. Socially compliant mobile robot navigation via inverse reinforcement learning. *I. J. Robotics Res.*, 35(11):1289–1307, 2016.

[18] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. pages 1928–1935, 05 2008.

[19] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In Cédric Pradalier, Roland Siegwart, and Gerhard Hirzinger, editors, *Robotics Research*, pages 3–19, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[20] Jamie Snape, Jur van den Berg, Stephen J. Guy, and Dinesh Manocha. Smooth and collision-free navigation for multiple robots under differential-drive constraints. In *IROS*, pages 4584–4589. IEEE, 2010.

[21] M. Missura, D. D. Lee, and M. Bennewitz. Minimal construct: Efficient shortest path finding for mobile robots in polygonal maps. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2018 (to appear).

[22] M. Missura and S. Behnke. Efficient kinodynamic trajectory generation for wheeled robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

[24] M. Missura, D. D. Lee, O. von Stryk, , and M. Bennewitz. The synchronized holonomic model: A framework for efficient motion generation. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2017).