Gradient descent dynamic window approach to the mobile robot autonomous navigation.

Hugo POUSSEUR*a) Non-member, Alessandro CORREA-VICTORINO*b) Non-member

Avoiding obstacles is a key feature in a vehicle autonomous navigation methodology. The dynamic window approach (DWA), which has been proposed for several decades, has emerged as a responsive navigation methodology suitable reactive for obstacle avoidance.

In the initial approach of the dynamic window obstacle avoidance strategy, the optimization of an objective function is realized with an exhaustive computation which can be costly in computational time. This is not useful in a real-time scenario where an autonomous vehicle needs to avoid obstacles in urban or road velocity conditions. The improved run time execution also makes for a less abrupt and more comfortable driving experience.

In this paper, we revise the DWA methodology, implementing a new method that forces us to redefine the objective function differently, defining it as a loss function. In order to apply a gradient descent to optimize the convergence.

We use DWA to control our autonomous cars, but to verify the correctness of our optimization, we optimized DWA in its original context, i.e., to control a robot in an unknown environment with obstacles to visit given positions. We tested our approach, in simulation on ROS and on a real Turltebot robot.

Keywords: Optimisation Control, DWA, Gradient Descent, ROS

1. Introduction

The safe navigation of these autonomous vehicle requires perceiving the surrounding environment, interacting with any spatial and temporal changes in this environment, and rescheduling its mission according to this dynamic. A first required functionality is the autonomous realization of movements, avoiding obstacles, in an a priori unknown environment. The dynamic window approach (DWA), originally proposed by Dieter Fox [5] in 1997, is mobile robot autonomous navigation methodology that could be useful to the reactive local navigation of autonomous robotic vehicles [4].

The lane keeping module of our autonomous vehicles is based on an implementation of the DWA including deep learning models. In order to optimize the computations performed by our autonomous module, we try to optimize each part, including the computations of the different deep learning models, but also the computations performed by the DWA. We noticed that, the initial approach of DWA requires us to discretize a search space and to find the optimal solution among this space by applying brute force. As an example, in the default implementation proposed by ROS, the search space is of the order of 600. This means that the optimal solution can only be defined after 600 calculations, each of these calculations is independent. It is important to note that the implementation of this strategy on an autonomous car, obliges us because of the physical capacities to define an even more important space and consequently requires more calculations. And at high speed, it is important to be able to define a precise command quickly to avoid any accident.

Since the creation of DWA, new approaches have emerged. These different approaches [2] [6], adapt the initial approach to new robots, adapting DWA to holonomic vehicles. Indeed, the solution initially proposed is only applicable to non-holonomic robots. The authors propose to adapt the approach to holonomic robot by adapting the objective function. Other searches allow making DWA tractable and convergent, inspired by a model predictive control (MPC) and control Lyapunov function (CLF) frameworks proposed by Primbs [7]. The idea is to avoid local minima that would move the robot away from its target and the risk of it getting stuck. Other searches focused on the adaptation of the DWA according to its environment by using notions of reinforcement learning [3] or fuzzy logic [10]. This new approach is called Adaptive Dynamic Window Approach, the weights used by the objective function are defined in real-time in according the topology of the environment. In practice, we notice that the choice of weights is dependent on the situation, on the topology of the obstacles. These adaptations define the best path in all situations and does not get stuck.

Our approach proposed is to replace the initial search of the optimal velocity by a gradient descent in order to converge more quickly to an optimal solution. This paper focuses on the optimization of the DWA in the initial context, i.e. the robot must be able to evolve in an environment with obstacles in order to reach given positions.

The paper is structured as follows, the Section 2 introduces main concepts of the origin DWA. The Section 3 defines our modification about the initial objective function. The Section 4 shows the implementation of our solution on virtual and real robot and the Section 5 discusses the results of the

a) Correspondence to: hugo.pousseur@hds.utc.fr
b) Correspondence to: alessandro.victorino@hds.utc.fr
* Université de technologie de Compiègne, CNRS, Heudiasyc (Heuristics and Diagnosis of Complex Systems), CS60319 - 60203 Compiègne Cedex

experiments.

2. The Original DWA Method

This method provides obstacle avoidance and is classified as an online collision avoidance strategy, i.e., this strategy is based on the current state of the robot (i.e., current robot dynamics, velocity and acceleration). This method takes in consideration physic limitations of the robot, maximum acceleration/deceleration, in order to define the controls achievable by the robot.

The method is divided in two steps, first we define the velocities considered as admissible by the robot at the current time t, and second we define the one among these velocities considered as the best.

2.1 Search space For a given moment, T_t, the current velocity of the robot can be annotated like: (v_t, w_t). The set of achievable velocities at time t is limited by the physical restrictions of the robot (acceleration and braking) and by the safety of the robot (this set excludes all velocities that could collide the robot with an obstacle). The dynamic window (denoted by V_t) represents the velocity reachable at the moment T_t in a time interval d_t, this window is centered around the current velocity.

2.2 Objective Function Once the search space defined, the velocity (v^*, w^*) considered as optimal is defined in this space, $(v^*, w^*) \in V_t$. The optimal solution must allow to reach the objective by avoiding the obstacles as well as possible, these conditions are translated by an objective function allowing to attribute to any velocity a value quantifying the quality of the solution. The equation 1 defines the objective function used.

$$\begin{aligned} G(v,w) =& \alpha.heading(v,w) \\ &+ \beta.dist(v,w) \\ &+ \gamma.velocity(v,w) \end{aligned} \tag{1}$$

The velocity selected, is the couple $(v, w) \in V_t$ who maximizes the objective function:

$$(v^*, w^*) = \underset{(v,w)\in V_t}{\arg\max}(G(v,w)) \tag{2}$$

2.2.1 Target Heading: *heading*(v, w) This function is fundamental in the success of the objective, quantifies whether the robot performs the task it is supposed to perform, i.e. to get to a given position. The heading function defines the alignment of the robot with the target position, if the robot applies the velocity (v, w) tested.

2.2.2 Clearance: *dist*(v, w) The DWA must control the robot towards a target while avoiding obstacles, the dist function quantifies the distance between the robot and obstacles once the robot applies the velocity (v, w). The value of the function is high if the robot is far from obstacles, otherwise it is low.

2.2.3 Velocity: *velocity*(v, w) This function ensures that the robot is always moving at the desired speed. This encourages the robot not to get stuck and to keep moving.

3. From Objective Function To Loss Function

3.1 Motivation As explain above (Section 2.2), the optimal solution for a t moment is defined in attributing an objective value at each velocity tested. This step requires discretizing the initial search space. Thus, our initial continuous search space is now defined by a matrix $MV \in \mathcal{M}_{n,m}(\mathbb{R})$. The

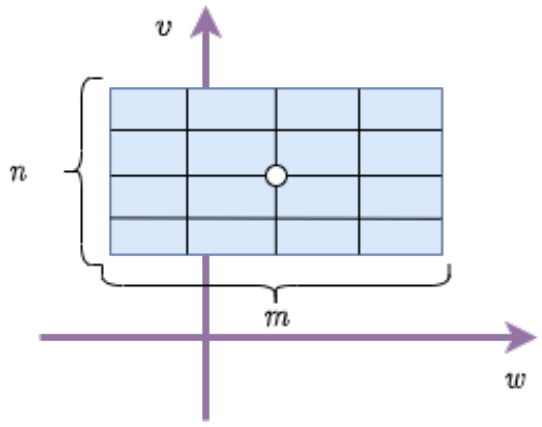

Fig. 1. Discretizing annotation.

figure 1 explains the discretizing process.

The search for the optimal solution thus consists in calculating for each $a_{ij} \in MV$ the value of the objective function. Each of these calculations is independent, i.e. no subcalculation can be used to calculate the objective value of another velocity. In this case we can define the complexity of the search to equal to $O(n * m)$. This complexity reflects a relationship between complexity and search quality. In the case where n and m are small then the search is fast, but the solution will be problematically inaccurate because of a too large discretization step. In the opposite case, the solution will be accurate but will require a longer execution time.

We want to find a method to avoid this brute force approach which forces us to perform several independent calculations. Our idea is to try to replace this approach by a convergent approach.

3.2 Gradient Descent Approach Our idea is to convert the initial objective function into a loss function. That means that the optimal solution minimize the function instead of maximize. Additionally, if this loss function is convex we can apply a gradient descent in order to converge to the minimum global avoiding to brute force. Let γ the learning rate and L a function, defined and differentiable in a neighbor of the point w, then the gradient descent is defined by:

$$w_{n+1} = w_n - \gamma * L(w_n) \tag{3}$$

So our approach is to change the objective function to a convex function. To create the loss function, we can use the following property of the convex function:

> if $w_1, ..., w_n \geq 0$ and $f_1, ..., f_n$ are all convex, then so $w_1 f_1 + ... + w_n f_n$ is convex.

Our idea is to exploit this property and find for each initial sub-function of the DWA an equivalent convex loss function. The loss function can then be defined as follows:

$$\begin{aligned} L(v,w) =& \alpha.heading_{loss}(v,w) \\ &+ \beta.dist_{loss}(v,w) \\ &+ \gamma.velocity_{loss}(v,w) \end{aligned} \tag{4}$$

Contrary to the initial definition the optimal solution, (v^*, w^*), is the velocity belonging to the search space, V_t, that minimizes the loss function.

$$(v^*, w^*) = \arg\min_{(v,w)\in V_t}(L(v, w)) \tag{5}$$

3.3 Constraint The Gradient Descent The initial DWA approach limits the search space by velocities reachable in dt time and that avoid obstacles. So we can't apply a gradient descent until the optimal solution, because we can't be sure that the optimal solution is defined in the search space.

Our approach includes this same concept of search space, the gradient descent is realized until the solution belongs to search space. The search space is defined from the current robot speed. So that the current speed is included in this space ($(v_t, w_t) \in V_{t+1}$). We can then include this notion of search space by applying the gradient descent until we are outside this space.

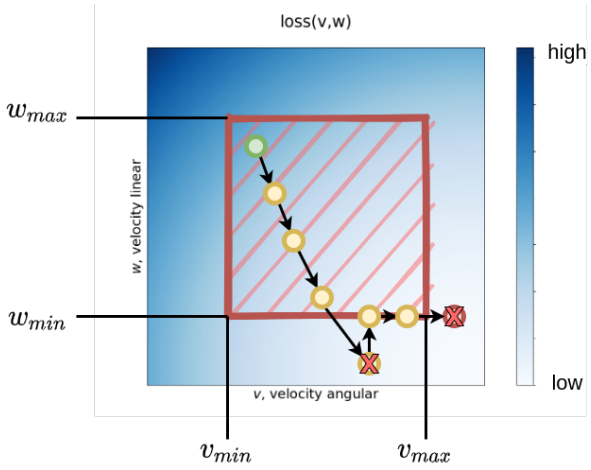

Fig. 2. Descent gradient constraint explication.

The following Figure 2 shows this limitation. The hashed rectangle represents the search space at time t, the green point (first point) represents the current speed where the gradient descent starts, the yellow points are found by the gradient descent, as shown the 5^{th} point is outside the search space, so we stop the gradient descent on the variable w, and place the point on the boundary. Until the gradient descent is off the side of v.

This restriction is based on the sign of the gradient as well as the current value. From this information, we can restrict the descent in the following way:

- $\partial L(v) < 0 \wedge v > v_{max}$ set to $v = v_{max}$
- $\partial L(v) > 0 \wedge v < v_{min}$ set to $v = v_{min}$
- $\partial L(w) < 0 \wedge w > w_{max}$ set to $w = w_{max}$
- $\partial L(w) > 0 \wedge w < w_{min}$ set to $w = w_{min}$

Once the descent is limited on an axis, then on this variable is defined as non-trainable, so the descent evolves only on the last remaining axis.

3.4 Loss Functions Definitions The following subsections will introduce new definitions of sub-functions used inside our loss function. The loss function reflects a relevant solution (v_i, w_j) if the value of the loss function evaluated at that velocity (v_i, w_j) is small.

3.4.1 Heading Loss Function As explained before (section 2.2.1), the heading function allows us to quantify if the computed velocity allows us to orient the robot in the alignment of the target to visit.

From the angle with the target and the interval time dt used we can define the w^* the optimal angular velocity. By this way we can define the $dist_{loss}$ as:

$$\begin{aligned} heading_{loss}(v, w) &= \frac{1}{2}(w - w^*)^2 \\ &= \frac{1}{2}\left(w - \left(arctan\left(\frac{y_{goal} - y_t}{x_{goal} - x_t}\right) - \theta_t\right) * \frac{1}{dt}\right)^2 \end{aligned} \tag{6}$$

Where (x_{goal}, y_{goal}) defines the position of the target, (x_t, y_t) the position of the robot at the current moment and θ_t the current orientation of the robot. From the loss function we can deduce the gradient as:

$$\frac{\partial heading_{loss}}{\partial v} = 0 \tag{7}$$

$$\frac{\partial heading_{loss}}{\partial w} = \left(w - \left(arctan\left(\frac{y_{goal} - y_t}{x_{goal} - x_t}\right) - \theta_t\right) * \frac{1}{dt}\right) \tag{8}$$

3.4.2 Dist Loss Function The $dist_{loss}$ function is used to move the robot away from obstacles around it. The definition of this function is divided in two steps, first we have to define the zones in which the robot can evolve safely (i.e. does not collide with obstacles). From this area we can define at a time t the angle allowing to place the robot in a safety position close to the current robot angle.

Safe Zone Definition: This function is based on our concept of safe zone. A safe zone represents area where the robot can move safely. This approach takes inspiration from Vector Field Histogram methods [1].

A safe zone is delimited by two angles where for all $\theta \in [\theta_{begin}, \theta_{end}]$ defining safe angles for the robot. The figure 3 shows three zones, represented by two hashed lines, the crossed out rectangles represent the obstacles.

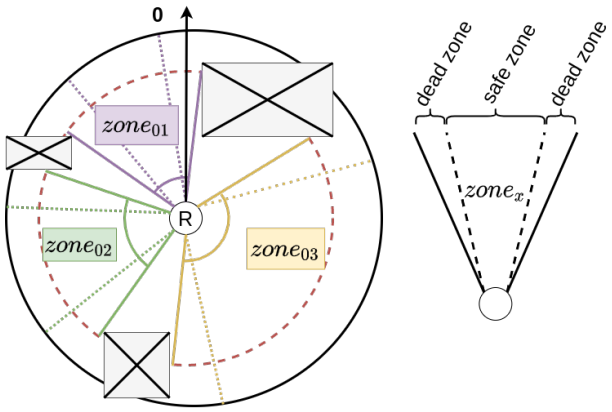

Fig. 3. Zone explication.

In first time, we create clear zones from LiDAR data. A clear zone is defined by two angles, where there is no one obstacle between them.

The safe zone represents the clear zone without collision risk, i.e. safe zone includes robot dimension constraint to remove areas where the robot may collide with obstacles. Our idea is to remove areas from the clear zone, in which the robot collides with obstacles, called dead zone. The figure 4 represents these different zones. The dead zone depends on

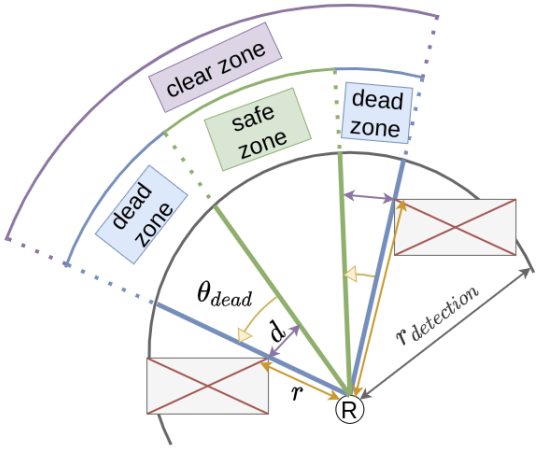

Fig. 4. Variables of safe zone representation.

the variable d representing the distance expected between the robot and the obstacle. We can define the variable as:

$$d = d_{radius} + d_{\epsilon} \tag{9}$$

Where d_{radius} the radius of the robot base and d_{ϵ} is a security distance. The dead angle depends on the distance d expected and the radius r between the robot and the obstacle. We can define the θ_{dead} by the following relation:

$$\theta_{dead} = 2 * arcsin\left(\frac{d}{2 * r}\right) \tag{10}$$

Once the dead angles are calculated, we can deduct them to define the safe zones. Which one z_i defined by $z_i = [\theta_{begin}, \theta_{end}]$

Loss Definition: Once the safety zones are defined, we can define for the moment t, the optimal angle to avoid the obstacles. These zones are defined from the robot's LiDAR, i.e. $\theta_{robot} = 0$ (the angles are relatives to the robot). The optimal angle is the angle defined in safe zone closed to the angle robot (θ_{robot}). There are two principals cases, the θ_{robot} is defined outside any zones, $\forall z_i = [\theta_{begin,i}, \theta_{end,i}], \theta_{robot} \notin z_i$. In this case the optimal solution is the angle inside a zone and close to the θ_{robot} (Figure 5, case 01). Otherwise, the θ_{robot} is defined in a zone safe then the angle is the optimal angle (Figure 5, case 02 and case 03). We can generalize the approach by the following relation:

$$\theta^* = \underset{\theta \in \bigcup[\theta_{begin,i}, \theta_{end,i}]}{\arg\min} \; (dist_{angle}(\theta, \theta_{robot})) \tag{11}$$

Let θ^*_{direct} the absolute difference direct between θ^* and θ_{robot}, otherwise, $\theta_{indirect}$ difference indirect. Then the $\theta^*_{final} = min(\theta^*_{direct}, \theta^*_{indirect})$

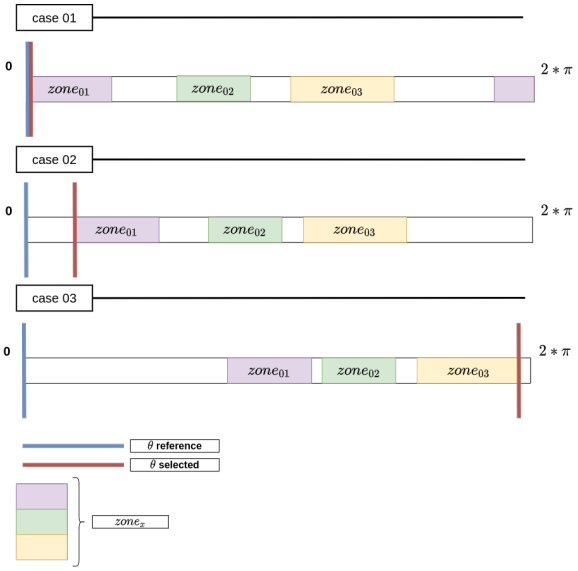

Fig. 5. Variables of safe zone representation.

$$\begin{aligned} dist_{loss}(v, w) &= \frac{1}{2}(w - w^*)^2 \\ &= \frac{1}{2}\left(w - \frac{\theta^*_{final}}{dt}\right)^2 \end{aligned} \tag{12}$$

from the $dist_{loss}$ function we can define, the gradient of this function as:

$$\frac{\partial dist_{loss}}{\partial v} = 0 \tag{13}$$

$$\frac{\partial dist_{loss}}{\partial w} = \left(w - \frac{\theta^*_{final}}{dt}\right) \tag{14}$$

3.4.3 Velocity Loss Function Unlike the previous functions, the $velocity_{loss}$ function does not require pre-calculation, we can directly define the translation velocity expected as the velocity optimal v^*, this velocity is defined by the user.

$$velocity_{loss}(v, w) = \frac{1}{2}(v - v^*)^2 \tag{15}$$

$$\frac{\partial velocity_{loss}}{\partial v} = (v - v^*) \tag{16}$$

$$\frac{\partial velocity_{loss}}{\partial w} = 0 \tag{17}$$

3.5 Loss function Based on the above definitions, the final loss function, L, is defined as follows:

$$\begin{aligned} L(v, w) = \alpha &\left(* \frac{1}{2}\left(w - \left(arctan\left(\frac{y_{goal} - y_t}{x_{goal} - x_t}\right) - \theta_t\right) * \frac{1}{dt}\right)^2 \right) \\ &+ \beta * \left(\frac{1}{2}\left(w - \frac{\theta^*_{final}}{dt}\right)^2\right) \\ &+ \gamma * \left(\frac{1}{2}(v - v^*)^2\right) \end{aligned}$$

(18)

Then the gradient of the L function is defined by:

$$\frac{\partial L}{\partial v} = (v - v^*) \tag{19}$$

$$\frac{\partial L}{\partial w} = \left(w - \left(arctan\left(\frac{y_{goal} - y_t}{x_{goal} - x_t}\right) - \theta_t\right) * \frac{1}{dt}\right) + \left(w - \frac{\theta^*_{final}}{dt}\right) \tag{20}$$

4. Methodology Validation

We have tested and evaluated the proposed methodology in virtual simulated environment and in real experimentation on board a real mobile robot. Our solution has been implemented on ROS[†], in this way is easy to test our approach on the Gazebo simulator and on real robot. The package uses the Tensorflow library[††] allowing to exploit gradient descent definition as well as gradient descent optimizers.

4.1 Simulation Experimentation In the simulation, the robot evolves inside a virtual world with obstacles. The simulation uses a virtual robot Turtlebot, so it is possible to exploit our simulation tests on a real robot afterwards. Before to start the simulation, we define a sequence of targets to visit.

The Figure 6 shows the path realized by the robot during the simulation, squares represent obstacles, dots represent target to visit, and the path represents the positions of the robot during the simulation. The robot uses the ROS package including our gradient descent approach. Like the original approach, we have to define in advance the parameters used in the loss function. This assignment is done empirically. The test has been realized with parameters defined in the table 1. During the simulation, for t moment, a descent is computing in order to define the optimal control. The descent resumes until variables are inside the search space and that the descent does not go up. The figure 7 shows the number of iterations used per descent during the simulation.

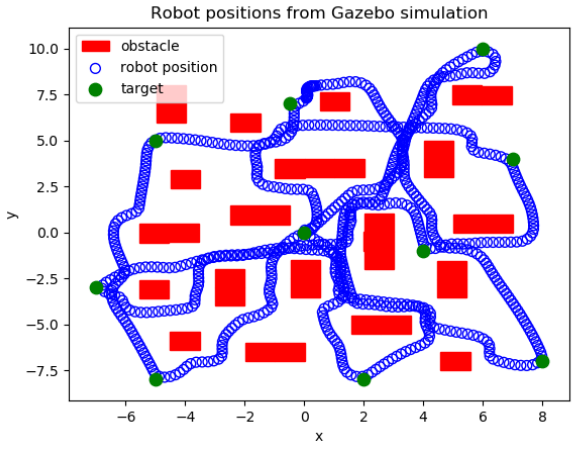

Fig. 6. Simulation experience, path made by the robot.

[†] Official website: https://www.ros.org/
[††] Official website: https://www.tensorflow.org/

variable	w_heading	w_dist	w_velocity	learning_rate
value	0.2	1.0	0.1	0.05

Table 1. Hyperparameters values

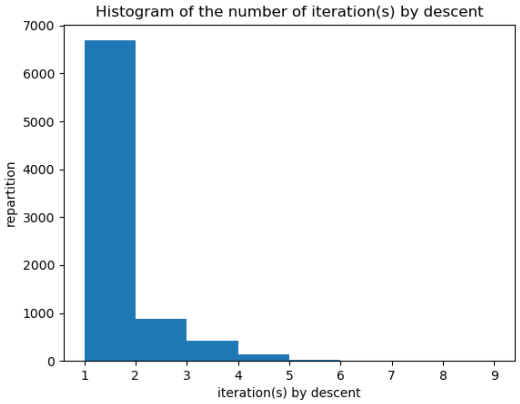

Fig. 7. Simulation experience, histogram of the distribution of the number of iterations used per descent.

4.2 Real Robot Experimentation The integration of our approach was designed to fit easily into Turtlebot robot. Our integration was tested in a joint experiment with a master student. In this experiment, the robot must reach a set of points while avoiding obstacles, a drone must be able to follow the robot's trajectory using visual feedback.

The Figure 8 shows the setup of our experience, where blue trashcans represent obstacles. The Figure 9 shows trajectory the robot during the experience, the crosses represent positions targets and dots represent obstacles.

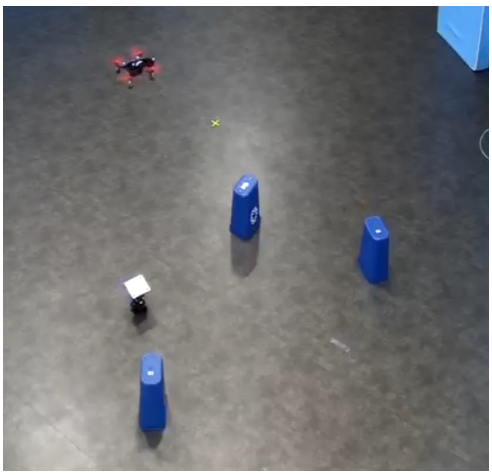

Fig. 8. Drone follower experience, picture of the scene.

5. Discussion

The first experience, inside the simulation, shows that our solution can drive a robot in an environment with obstacles. The robot managed to evolve to reach the different objectives while avoiding the obstacles on its way. Like the original approach, we defined the weights that best fit our situation, and we defined the learning rate used by the gradient descent.

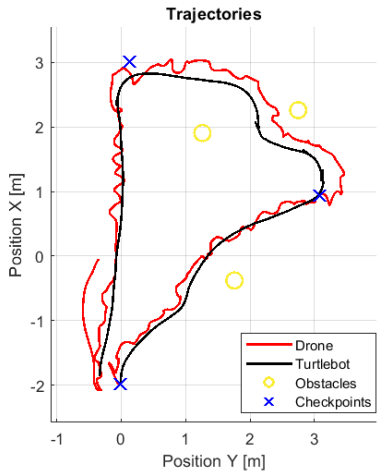

Fig. 9. Drone follower experience, drone and robot positions graphic.

The goal of our approach is to reduce the execution time, the Figure 7 shows us the number of iterations used per descent during the experience. We notice that in most cases, we only need one iteration to control the robot.

In the initial approach, the search space is defined using hyperparameters that define how to discretize the space. It is difficult to be objective about the comparison, but for comparison, we can use the default settings used by the ROS DWA library[†]. In this case, the search space is defined by a space of size 600, which means that the optimal solution requires 600 iterations per calculation. We can compare that our solution avoids many operations compared to the optimal solution; indeed, according to our experience, only ≈ 1.33 iterations are needed on average per calculation.

The second experiment proves that we can use our approach in a real situation with a real robot. This experiment showed that the robot was able to move in different positions while avoiding obstacles.

6. Conclusion

We tested a new approach to reduce the computation time in the initial context of the dynamic window approach. This approach has been successful because it has shown, in simulation and in real life, its efficiency and its ability to reduce the computation time. We plan to implement this same solution in our solution for driving autonomous cars, where the goal here is no longer to go to a set of given positions but to keep the vehicle in the center of the lane while avoiding obstacles on the lane. We must then redefine our loss function according to this new objective.

6.1 Limitation As with the original approach, because of the hyperparameters, this strategy is rigid with respect to the situation, depending on each situation optimal weights can be set. The definition of the learning rate is also done upstream, it is defined empirically with respect to a set of tests.

In our approach, we define an optimal angle at time t to avoid obstacles. This approach ignores solutions that are also potentially good.

[†] DWA ROS documentation: http://wiki.ros.org/dwa_local_planner

6.2 Perspective Different studies [10] [3] propose to vary the weights used by the objective function, we can apply these studies to our loss function so that the weighting of the weights is no longer dependent on a specific situation but has the ability to adapt to its environment.

Our approach adds a new hyperparameter, the learning rate. This variable is defined empirically by making several tests, we can in an analogous way to the weights of the loss function, define the learning rate by using information on the environment. Another approach is to look if the loss function evolves very little between two iterations. If the variation is small, we can possibly make a slaving of the learning rate according to the number of iterations performed during the previous calculation. In this way, it will be possible to make the number of iterations per calculation oscillate around a new parameter fixed beforehand, defining an instruction of the number of iterations to be carried out per iteration.

We can optimize the gradient descent by using optimizes [9], for example we can use a gradient descent with momentum [8] to accelerate the convergence.

Moreover, find a new way to define relevant solutions that avoid obstacles (including a redefinition of the dist function), providing a set of possible solutions rather than a single solution. However, this redefinition may well include the notion of zones defined in this article.

Acknowledgment

Thanks to Emanuele VENZANO for his involvement in the experiment with the drone/robot.

References

(1) Johann Borenstein, Yoram Koren, et al. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

(2) Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings 1999 ieee international conference on robotics and automation (Cat. No. 99CH36288C)*, volume 1, pages 341–346. IEEE, 1999.

(3) Lu Chang, Liang Shan, Chao Jiang, and Yuewei Dai. Reinforcement based mobile robot path planning with improved dynamic window approach in unknown environment. *Autonomous Robots*, 45(1):51–76, 2021.

(4) Danilo Alves de Lima and Alessandro Corrêa Victorino. A visual servoing approach for road lane following with obstacle avoidance. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 412–417. IEEE, 2014.

(5) Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

(6) Marcell Missura and Maren Bennewitz. Predictive collision avoidance for the dynamic window approach. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8620–8626. IEEE, 2019.

(7) Petter Ogren and Naomi Ehrich Leonard. A tractable convergent dynamic window approach to obstacle avoidance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 595–600. IEEE, 2002.

(8) Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

(9) Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

(10) Daniel Teso-Fz-Betoño, Ekaitz Zulueta, Unai Fernandez-Gamiz, Aitor Saenz-Aguirre, and Raquel Martinez. Predictive dynamic window approach development with artificial neural fuzzy inference improvement. *Electronics*, 8(9):935, 2019.