Dynamic Obstacle Avoidance using Online Trajectory Time-scaling and Local Replanning

Ran Zhao\textsuperscript{1,2} and Daniel Sidobre\textsuperscript{2,3}
\textsuperscript{1}CNRS, LAAS, 7 Avenue du Colonel Roche, F-31400 Toulouse, France
\textsuperscript{2}Univ. de Toulouse, LAAS, F-31400 Toulouse, France
\textsuperscript{3}Univ. de Toulouse, UPS, LAAS, F-31400 Toulouse, France

Keywords: Obstacle Avoidance, Velocity Obstacles, Trajectory Time-scaling, Local Replanning.

Abstract: In various circumstances, planning at trajectory level is very useful to generate flexible collision-free motions for autonomous robots, especially when the system interacts with humans or human environment. This paper presents a simple and fast obstacle avoidance algorithm that operates at the trajectory level in real-time. The algorithm uses the Velocity Obstacle to obtain the boundary conditions required to avoid a dynamic obstacle, and then adjust the time evolution using the non-linear trajectory time-scaling scheme. A trajectory local replanning method is applied to make a detour when the static obstacles block the advance path of the robot, which leads to failure of implementing time-scaling approach. Cubic polynomial functions are used to describe trajectories, which brings sufficient flexibility in terms of providing higher order smoothness. We applied this algorithm on reaching tasks for a mobile robot. Simulation results demonstrate that the technique can generate collision-free motion in real time.

1 INTRODUCTION

To achieve a large variety of tasks in interaction with human or human environments, autonomous robots must have the capability to quickly generate collision-free motions. Significant research has been performed in the modeling of the path planning problem. Sample-based planners such as rapidly-exploring random trees (RRTs)\cite{LaValle2001}, or probabilistic roadmaps (PRMs)\cite{Kavraki1996} generate the motion as collision-free paths, which the robot is expected to follow. They are often fast, but they generate a global path using an environmental model and update the planned path when the planned path is blocked by unmapped obstacles. As a result, they can not deal with unknown environments with a large number of dynamic obstacles.

To make the robot more reactive, it is reasonable to replace paths by trajectories as the interface between planners and controllers, and to add a trajectory planner as an intermediate level in the software architecture. To react to environment changes, the trajectory planning must be done in real time. Meanwhile, the robot needs to guarantee the human safety and the absence of collision. So the model for trajectory must allow fast computation and easy communication between the different components, including path planner, trajectory generator, collision checker and controller. To avoid the replan of an entire trajectory, the model must allow slowing down or deforming locally a trajectory.

To assure the collision safety of an autonomous robot in dynamic environment, the velocity of obstacles should be considered when planning the robots trajectory. The concept of Velocity Obstacle for obstacle avoidance was proposed in\cite{Fiorini1998} and was later extended to the case of reactive collision avoidance among multiple robots in\cite{vanDerBerg2011a,vanDerBerg2011b}. Velocity obstacles (VOs) represent a subset of the velocity space where a mobile robot and a dynamic obstacle collide in the future when the mobile robot moves at a velocity of the VO.

Trajectory time-scaling methods are generally used to adjust the speed or torque while maintaining the tracking path. The trajectory time-scaling schemes proposed in\cite{Dahl1989} and\cite{Morenon2006} are used in execution of fast trajectories along a geometric path, where the motion is limited by torque constraints.\cite{Sza2006} gives an on-line time-scaling methods based on the tracking error. In this study, we
use the trajectory time-scaling schemes to avoid dy-
namic obstacles, which will not result in a change of
path of the robot. Moreover, we apply a non-linear
time-scaling function to replan new trajectories for
avoiding dynamic obstacles, as shown in Fig. 1. The
advantage of our algorithm is more reactive to
dynamically changing environments and temporally
distributes the computations, making it feasible to im-
plement in real-time.

The rest of the paper is organized as follow: We
present the trajectory representations and introduce
notations in Section 2. We describe the obstacle
avoidance algorithms in Section 3 and Section 4. Sim-
ulation results are highlighted in Section 5.

2 NOTATIONS AND
REPRESENTATIONS

2.1 Trajectory Representations

Let $C = \mathbb{R}^n$ denote the $n$-dimensional configura-
tion space. A trajectory $T$ can be a direct function of time
or the composition $P(u(t))$ of a path $P(u)$ and a func-
tion $u(t)$ describing the time evolution along this path.
The background trajectory materials are summarized in
the books from Biagiotti (Biagiotti and Melchiorri, 2008) and Kroger (Kroger, 2010). A trajectory $T$ is then defined as:

$$T : [t_1, t_F] \longrightarrow \mathbb{R}^n$$

where $T(t) = X(t) = (X(t), 2X(t), \cdots, nX(t))^T$ in
Cartesian space from the time interval $[t_1, t_F]$ to $\mathbb{R}^n$.

2.2 Velocity Obstacle (VO)

A VO (Fiorini and Shillert, 1998) represents a set of
velocities that create an obstacle collision. Consider
two circular objects $A$ and $B$, shown in Fig. 2(a) at
time $t_0$, with constant velocity $v_a$ and $v_b$. Let circle
$A$ represent the robot, and $B$ represent the obstacle.
The point \( \hat{A} \) is center of circle $A$, $\hat{B}$ is mapped by
enlarging $B$ by the radius of $A$, shown in Fig. 2(b). The Collision Cone $CC_{a,b}$ is then defined as the set of
colliding relative velocities between $\hat{A}$ and $\hat{B}$:

$$CC_{a,b} = \left\{ v_{a,b} \mid \lambda_{a,b} \cap \hat{B} = \emptyset \right\}$$

where $v_{a,b}$ is the relative velocity of $\hat{A}$ with respect to
$\hat{B}$, $v_{a,b} = v_a - v_b$, and $\lambda_{a,b}$ is the line of $v_{a,b}$.

The shape of $CC_{a,b}$ is the planar sector with apex
in $\hat{A}$, bounded by the two tangent from $\hat{A}$ to $\hat{B}$. Any
relative velocity lies in this cone will cause a collision.
To deal with multiple obstacles, it is interesting to
consider the VO of the absolute velocity of $A$. This
can be done simply by adding the velocity of $\mathcal{B}$:

$$\mathcal{V}_O = C_{a,b} \oplus v_b$$  \hspace{1cm} (4)

where $\oplus$ is the Minkowski sum. When the velocity of robot $\mathcal{A}$ is included in the $\mathcal{V}_O$, the robot will collide with obstacle $\mathcal{B}$ in the future. In the case of multiple obstacles, the collision-free velocity can be obtained by selecting a velocity that is not contained in any VO. Fig. 2(c) shows an illustration of an VO.

3 COLISION AVOIDANCE BY TRAJECTORY TIME-SCALING

3.1 Time-scaling Function

A time-scaling function $s = s(t)$ is an increasing function of time. Using the time-scaling function a new trajectory $T(t)$ can be defined such that $T(t) = T(s)$. Applying the time-scaling function does not change the path taken by the robot, but brings the following changes in the velocity and acceleration profile of the trajectory.

$$\tilde{v}(t) = v(s) \hat{s}$$  \hspace{1cm} (5)

$$\tilde{\dot{A}}(t) = A(s) s^2 + V(s) \hat{s}$$  \hspace{1cm} (6)

$$\tilde{\ddot{J}}(t) = J(s) s^3 + 3A(s) s \hat{s} + V(s) \hat{s}$$  \hspace{1cm} (7)

where $s(t)$ scales up the velocity and acceleration for $s(t) > 1$ and scales it down for $s(t) < 1$.

In this current work, we use the concept of the Optimal Motion to describe the time variation. The Optimal Motion is a motion with jerk, acceleration and velocity constraints successively saturated (Herrera and Sidoère, 2005). We use the velocity of an Optimal Motion to represent the time variation. In this case, we are able to control the time change by defining the maximum jerk $J_{max}$, acceleration $A_{max}$ and velocity $V_{max}$, thus the motion of the robot can be bounded, e.g. the kinematic constraints ($J_{max}$, $A_{max}$, $V_{max}$) of the robot motion are linked to the Coefficient of the time-scaling function ($J_{max}$, $A_{max}$, $V_{max}$). So we use a non-linear time-scaling function as follow:

$$s(t) = \begin{cases} 
    s(t) + J_{max} t + J_{max}^2 t^2, & J = -J_{max} \\
    s(t) + A_{max}, & J = 0, A = A_{max} \\
    s(t) + A_{max} t + J_{max}^2 t^2, & J = J_{max}
\end{cases}$$ \hspace{1cm} (8)

In this study, we assumed that a mobile service robot should not increase its speed to avoid moving obstacles, because it could cause some people anxiety. Thus $V_{max} = 1$. It also should be noted that the acceleration/de-acceleration produced by the scaling transformation must respect the acceleration and jerk bounds and hence the following inequalities are obtained: $|\tilde{v}(t)| \leq V_{max}$, $|\tilde{\dot{A}}(t)| \leq A_{max}$, $|\tilde{\ddot{J}}(t)| \leq J_{max}$. With the framework for constructing scaling function in place, the collision avoidance problem becomes of choosing what scaling function to use for the robot at any given instant.

3.2 Single Obstacle Case

The case of a robot avoiding a moving obstacle is first considered. This is illustrated in Fig. 2(c) which shows robot $\mathcal{A}$ in collision course with passively moving obstacle $\mathcal{B}$. Collision is avoided if the velocity of $\mathcal{A}$ is out of $\mathcal{V}_O : v_a \leq v_{max}$, where $v_{max}$ is the maximum velocity of $\mathcal{A}$ on the $\mathcal{V}_O$ boundary. With the trajectory of robot, we could get the final value of the time-scaling function $s(T_f)$ which satisfy that: $v_a(s) \leq v_{max}$. Then we could define the Optimal Motion for the time-scaling function. The initial condition and final condition of the motion are $0, 0, s(t_f)$ and $1, 0, s(t_f)$, respectively. $s(t_f)$ is the current time-scaling factor when the robot start to accelerate/decelerate. $[0, T_{op}]$ is the time interval of the time-scaling function, where $T_{op}$ is the execution time of the Optimal Motion . So $J, A$ and $s(t)$ are piecewise functions:

$$J(t) = \begin{cases} -J_{max} & t \in (0, \frac{T_{op}}{2}] \\
0 & t \in \left[\frac{T_{op}}{2}, \frac{T_{op}}{2} + \frac{A_{max}}{2J_{max}}\right] \\
-J_{max} + \frac{A_{max} t}{2} & t \in \left[\frac{T_{op}}{2} + \frac{A_{max}}{2J_{max}}, T_{op}\right]
\end{cases}$$  \hspace{1cm} (9)

$$A(t) = \begin{cases} 0 & t \in (0, \frac{T_{op}}{2}] \\
-A_{max} & t \in \left[\frac{T_{op}}{2}, \frac{T_{op}}{2} + \frac{A_{max}}{2J_{max}}\right] \\
-A_{max} + \frac{A_{max} t}{2} & t \in \left[\frac{T_{op}}{2} + \frac{A_{max}}{2J_{max}}, T_{op}\right]
\end{cases}$$  \hspace{1cm} (10)

Then from the Eqs 5-10, we can calculate the maximum coefficients, $J_{max}$ and $A_{max}$, of the time-scaling function.

3.3 Multiple Obstacles Case

In the case of multiple obstacles, it may be useful to prioritize the obstacles so that those with imminent collision will take precedence over those with long time to collision. We use the concept of imminent (Fiorini and Shillert, 1998) which is a collision between the robot and an obstacle if it occurs at some time $t < T_h$, where $T_h$ is a suitable time horizon, selected based on system dynamics, obstacle trajectories, and the computation rate of the avoidance maneuvers. To account for imminent collisions, we compute the distance $d$
between the robot $A$ and an obstacle $B$ at time instant $t$. If the $d < v_{a,b} * T_h$, we treat $B$ as a imminent collision and the avoidance procedure will be first considered. If not, even $v_a$ lies inside the VO region, the old time evolution will be maintained. If there exist many imminents, the minimal time-scaling factor $s(t)_f$ will be applied to the original trajectory. Algorithm 1 shows the overall pseudocode of collision avoidance for multiple obstacles.

Algorithm 1: Trajectory time-scaling Based Collision Avoidance.

Input: a trajectory $T$ of robot $A$, number of obstacles $M$, $T_h$
1: for $i = 0$ to $M$ do
2: Obtain the $VO_i$, $v_a$ on $T$ at time $t$
3: if $v_a \cap VO_i \neq \emptyset$ then
4: Compute the relative velocity $v_i$ between the robot and the obstacle $i$
5: Compute the distance $d_i$ between the robot and the obstacle $i$
6: if $d_i < v_a * T_h$ then
7: Calculate the time-scaling factor $s(t)_fi$ for the single obstacle
8: else
9: $s(t)_f = s(t)_c$
10: end if
11: end if
12: end for
13: return the minimal factor $s(t)_f min$, then compute the coefficients $J_{max}$ and $A_{max}$

4 COLLISION AVOIDANCE BY TRAJECTORY LOCAL REPLANNING

The global path planning method considers the surrounding environment knowledge, and then attempts to optimize the path. However, some problems are existing in this method such as data are incomplete, the real environment situations are unexpected and the real-time operation of the robot.

Autonomous robots often operate in human environment where the obstacles may block the robot’s path. For example, an obstacle may have the same path with the robot but with a very low velocity, or an obstacle moves on the robot’s path and then halts there. In the first case, the time-scaling function will return a extreme small value, which leads to a quite slow robot motion. In the second case, the time-scaling function will stop the robot and goal position cannot be reached. Thus, the robot needs to be able to replan quickly as the knowledge of the environment changes. This can be done simply by adding a middle waypoint where the robot can pass through to avoid the obstacle. Fig. 3 illustrates a situation of the first case. Robot $A$ and the obstacle $B$ have the same path $P(u)$ while $V_b$ is small. To replan a new trajectory, we need to find a middle waypoint $P_m$ to make a detour, and a return point $P_f$ to go back the original path.

To avoid the obstacle, the middle waypoint must be out of VO. Waypoint on the boundaries of VO would result in $A$ grazing $B$. For sake of simplicity, we can enlarge the VO region slightly by increase the radius of $B$ by a constant $r$ (the black dashed circle in Fig. 3). In this study, we choose $P_f$ as:

$$P_f = \hat{B} \left( P(u)_B \ominus ra \right) \left( \frac{v_{a,b}}{|v_{a,b}|} \right)$$

where $P(u)_B$ is part of the path $P(u)$ that is behind the obstacle $B$. Then the middle waypoint $P_m$ is the point of intersection between the two tangents $\lambda_{\hat{B},\hat{B}}$ and $\lambda_{P_f,\hat{B}}$.

4.1 Trajectory Generation Through Waypoints

Now the problem becomes to generate a trajectory that traverse these waypoints. We consider a solution to plan point-to-point (straight-line) trajectories that halt at each via-point where the direction of the path changes. Then we smooth the edges of the trajectory to produce smoother motion defined by kinematics conditions.

The first step is to convert a piecewise linear path to a time-optimal trajectory that stops at each waypoint. The trajectory along a straight-line path should be a phase-synchronized motion(Broquere et al., 2008). To achieve that, we compute the final time for each dimension. Considering the largest motion time, we readjust the other dimension motions to this time. Time adjusting is done by decreasing linearly $J_{max}, A_{max}, V_{max}$. In other words, the motion consumes minimum time for one direction. At other directions, the motions are conditioned by the minimum one.

The point-to-point (straight-line) trajectory $T_{p2p}$ obtained in the first step is feasible, but is not sat-
is satisfactory because the velocity varies greatly at each waypoint, which stops the motion. These stops can be avoided by drawing shortcuts between random points on the trajectory. Without loss of generality we consider three adjacent points \((P_s, P_m, P_f)\) and the smoothing at the intermediate via-point \(P_m\). Firstly, the two straight-line trajectories \(T_{P_mP_e}\) and \(T_{P_mP_s}\) are computed respectively. Then we choose two points \((P_{\text{start}}, P_{\text{end}})\) based on the parameter \(l\), which is the distance from the point \(P_m\) on the trajectory. Then we compute a trajectory to connect \(P_{\text{start}}\) and \(P_{\text{end}}\). The two straight-line trajectories \((l_{\text{start}}, l_{\text{end}})\) are used to denote the two distances and they defines the limits of the smoothed area. The possible choices of these two points are infinite, and the resulting trajectories can vary greatly due to the different choices. In this study, we choose the two points considering the distance between the trajectory and the obstacle. We discuss how to choose the points in this paragraph and then detail the smooth trajectory generation in the next part.

Considering the case of three adjacent points \(P_s, P_m, P_f\), see Fig.4. \(\mathbf{n}_i\) being the normal unit vectors to the straight line \(P_iP_m\), and \(T_i(t)\) the smoothed trajectory. Then the error \(\varepsilon\) can be defined as:

\[
\varepsilon(t) = \min_{t \in [l_p,l_f]} ( |T_i(t) - P_m| \cdot \mathbf{n}_i, |T_i(t) - P_m| \cdot \mathbf{n}_{i+1} )
\]

\[
\varepsilon = \max(\varepsilon(t))
\]

(12)

To compute this error, we introduce another parameter \(\varepsilon_v\), which is defined by the minimum distance between the vertex \(P_i\) and the trajectory \(T_i(t)\):

\[
\varepsilon_v = \min_{t \in [l_p,l_f]} d(P_i, T_i(t))
\]

(13)

As the start and end points of the blend we choose locate at the symmetric segments on each straight-line trajectory, the error \(\varepsilon_v\) happens at the bisector of the angle \(\alpha\) formed by the 3 adjacent points, Therefore,

\[
\varepsilon_v = d(P_m, T_{l_p+\varepsilon_v})
\]

(14)

\[
\varepsilon = \varepsilon_v \cdot \sin \frac{\alpha}{2}
\]

(15)

To avoid the collision, a simple possibility is to change the distance \(l\) to make the error satisfy that:

\[
\varepsilon < d(\rho_n, B) - r_A
\]

(16)

where \(d(\rho_n, B)\) is the distance between the point \(P_m\) and the obstacle \(B\). To achieve Eq. 16 we introduce a parameter \(\delta\), for a general case, it is defined as:

\[
\delta = \frac{l_{\text{start}}}{|P_sP_m|} - \frac{l_{\text{end}}}{|P_mP_f|}
\]

(17)

Thus \(\delta \in [0,1]\). Then we can compute in a loop and get the \(\delta_{\text{max}}\).

4.2 Smooth Trajectory Generation

In this part, we describe how to generate a smooth motion to connect \(P_{\text{start}}\) and \(P_{\text{end}}\). The motion state of these two points are \(M_s\) and \(M_e\) respectively. Since each axis variable is assumed to be independent, the minimum execution time between \(M_s\) and \(M_e\) is determined by the slowest single-axis trajectory. Then the motion on the rest of the axis must be interpolated to this imposed time which is called \(T_{\text{imp}}\). More precisely, \(f(M_s, M_e, J_{\text{max}}, A_{\text{max}}, V_{\text{max}})\) is used to compute the time of the time-optimal interpolation between motion states \(M_s\) and \(M_e\) under the kinematic bounds \(J_{\text{max}}, A_{\text{max}}\) and \(V_{\text{max}}\). Thus,

\[
T_{\text{imp}} = \max(f(M_j, M_{\varepsilon j}, J_{\text{max}}, A_{\text{max}}, V_{\text{max}})) \quad j \in [1,n]
\]

The interpolation problem becomes one of building a motion with predefined time. We propose three methods by computing different parameters of the trajectory.

Three-segment Interpolants. If we consider states \(M_s\) and \(M_e\) defined by a starting instant \(t_s\) and an ending instant \(t_e\), the starting and ending situations to be connected are: \((X_s, V_s, A_s)\) and \((X_e, V_e, A_e)\). An interesting solution to connect this portion of trajectories is to define a sequence of three trajectory segments with constant jerk that bring the motion from the initial situation to the final one within time \(T_{\text{imp}}\). We choose three segments because we need a small number of segments and there is not always a solution with one or two segments.

The system to be solved is then defined by 13 constraints: the initial and final situations (6 constraints), the continuity in position velocity and acceleration for the two switching situations and time. Each segment of a trajectory is defined by four parameters and time.

If we fix the three durations \(T_1 = T_2 = T_3 = \frac{T_{\text{imp}}}{3}\), we obtain a system with 13 parameters where only the three jerks are unknown (Broquère and Sidobre, 2010). As the final control system is periodic with period \(T\), the time \(T_{\text{imp}}/3\) must be a multiple of the period \(T\), and in this study, \(T_{\text{imp}}\) is chosen to be a multiple of \(3T\).
Three-segment Interpolants with Bounded Jerk. The three-segment interpolants solves the problem of trajectory generation with fixed duration for each segment. However, it cannot be guaranteed that the computed jerk is always bounded. Here, we introduce a variant three-segment method with defined jerk.

Same as the three-segment method, the system is also defined by 13 constraints. With the variant method, however, we fix the jerks on the first and third segments as $|J_1| = |J_3|$, which have the value bounded within the kinematic constraints. Then, the unknown parameters in the system are $J_2$ and the three time durations. Thus we obtain a system of four equations with four parameters ($T_1$, $T_2$ and $T_3$):

$$A_x = J_3 T_3 + A_2$$  \hspace{1cm} (18)

$$V_x = J_3 T_3^2 / 2 + A_2 T_2 + V_2$$  \hspace{1cm} (19)

$$X_x - J_3 T_3^3 / 6 + A_2 T_2^2 / 2 + V_2 T_2 + X_2$$  \hspace{1cm} (20)

$$T_{tmp} = T_1 + T_2 + T_3$$  \hspace{1cm} (21)

where

$$A_2 = J_2 T_2 + J_1 T_1 + A_x$$

$$V_2 = J_2 T_2^2 / 2 + (J_1 T_1 + A_x) T_2 + J_1 T_1^2 / 2 + A_x T_1 + V_1$$

$$X_2 = J_2 T_2^3 / 6 + (J_1 T_1 + A_x) T_2^2 / 2 + (J_1 T_1^2 / 2 + A_x T_1 + V_1) T_2$$

$$+ J_1 T_1^3 / 6 + A_x T_1^2 / 2 + V_1 T_1 + X_1$$

To choose the values of jerks on each dimension, we resort to the velocities $V_x$ and $V_c$. The jerks are fixed by $J_1 = -J_3 = J_{max}$ when $V_x - V_c > 0$, and by $J_1 = -J_3 = J_{max}$ when $V_x - V_c < 0$. If $V_x - V_c = 0$, we compare the values of $A_x$ and $A_c$ instead.

Jerk-bounded, Acceleration-bounded, Velocity-bounded Interpolants. Now we derive the all-bounded trajectory given a fixed duration $T_{tmp}$. The method in the previous paragraph can directly bound the jerk, but have to readjust the jerk values by a predefined resolution to bound the velocity and acceleration. As we detect the longest execution time $T_{tmp}$ by computing the time-optimal trajectory on each axis, the jerk is saturated and the acceleration and velocity may be saturated, depending on different cases. Thus, we can extend the duration of all axis (except the one with the longest duration) to $T_{tmp}$ by unsaturated interpolants while maintaining the number of segments $N_j$ on each axis. We name it a Slowdown Down Motion. Algorithm 2 shows the generation of all-bounded interpolants.

Algorithm 2: All-Bounded Interpolants Generation.

**Input:** Motion states: $M_x, M_c$; number of DOFs: $n$; $T_{tmp}$

**Output:** Jerk-Bounded, Acceleration-Bounded, Velocity-Bounded Interpolants

1. **for** $j = 1$ to $n$ **do**
2. Compute the time-optimal interpolants between $M_x$ and $M_c$, then get the execution time $T_j$
3. Get $N_j$ and the execution time on each segment $T_j^{N}$
4. **if** $N_j = 0$ **then**
5. No motion on this axis, maintain the time-optimal interpolants
6. **else**
7. Enlarge $T_j^{N}$ by $T_j^{N} = T_j^{N} + T_{tmp} - T_j$
8. Compute the new Jerk on each segment $J_j^{N}$
9. **end if**
10. Generate the interpolants with $J_j^{N}$ and $T_j^{N}$
11. **end for**

Figure 5: A robot avoids an obstacle that is moving along the same trajectory as the robot (a)-(d): The original trajectory collides with the obstacle at 6-7.51 s, (e)-(h): The scaled collision-free trajectory.

Figure 6: Time-scaling function and scaled trajectory of Fig. 5. (a): Time-scaling function, (b): position, velocity, acceleration and jerk profile of the scaled trajectory along X axis.

5 SIMULATION RESULTS

The initial trajectories from start (0,0) to the goal (10,0) for the robots were obtained by modeling the robots as unicycle. The maximum velocity, acceleration and jerk of the robot was set to 1.5 m/s, 3 m/s$^2$ and 9 m/s$^3$, respectively.
5.1 Single Obstacle Avoidance

Fig. 5 shows the simulation results for an obstacle moving along the path of the robot. The blue and green circles represent the robot and the obstacle respectively. The situation shown in Fig. 5 takes place when the robot moves along a corridor. The obstacle moves at a velocity of \( v_x = 1 \) m/s from the initial location \((3,0)\). As shown in Fig. 5(a)-5(d), the robot collides with the obstacle at \( t = 6.75 \) s. Fig. 5(e)-5(h) demonstrates the time-scaled results. Obstacle avoidance was achieved by decelerating the robot. The duration of the trajectory was extended to \( 9.21 \) s. Fig. 6 illustrates the time-scaling function and scaled trajectory along the X axis. Results show that the robot decelerated to a safety velocity very fast (in less than \( 1 \) s), and all kinematic variables were well bounded during the construction of the scaled trajectory.

5.2 Multiple Obstacles Avoidance

Fig. 7 depicts a multiple collision situation. The obstacles \( O_1, O_2 \) and \( O_3 \) moved at a velocity of \( vy = 2 \) m/s, -1.15 m/s and 0.9 m/s, and started at the location \((3,5), (5,5), \) and \((7,5)\), respectively. The robot collided with the three obstacles at \( t = 2 \) s, \( t = 4 \) s and \( t = 5 \) s respectively when it followed the original trajectory. The trajectory was scaled at \( t = 0.3 \) s to pass obstacle \( O_1 \) through the trajectory, and was scaled at \( t = 1.5 \) s to avoid \( O_2 \). After the second time-scaling function, the VO became an empty set, then the system switch to the original trajectory by setting \( s(t) = 1 \). Fig. 8 shows the time-scaling function and the time-scaled trajectory.

5.3 Trajectory Replanning

Fig. 9 describes the obstacle avoidance using trajectory replanning. The time-scaling function returned to 0 because the velocity of the robot always lied in VO.

### Table: Simulation Results

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Position (m)</th>
<th>Velocity (m/s)</th>
<th>Acceleration (m/s²)</th>
<th>Jerk (m/s³)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>4.5</td>
<td>4.5</td>
<td>4.5</td>
<td>4.5</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 7: A robot avoids multiple obstacle. (a)-(d): The original trajectory collides with all the obstacles, (e)-(h): The scaled collision-free trajectory.

Figure 8: Time-scaling function and scaled trajectory of Fig. 7. (a): Time-scaling function, (b): position, velocity, acceleration and jerk profile of the scaled trajectory along X axis.

Figure 9: Simulation of trajectory local replanning. (a): The final path of the robot, (b): Time-scaling function, (c): The kinematic profile of the replanned trajectory along X axis. The system replanned a new trajectory when \( s(t) = 0 \) and \( s(t) \) was set to 1 immediately while the new trajectory was executed. Fig. 9(a)-9(c) illustrates the robot path, time-scaling function and the kinematic variables, respectively.

6 CONCLUSIONS AND FUTURE WORKS

This paper presents a simple and fast obstacle avoidance algorithm that operates at the trajectory level in real-time. It uses the trajectory time-scaling functions and trajectory replanning scheme to compute \( C^3 \) trajectories that are bounded in velocity, acceleration and jerk. The proposed algorithm is based on the use of pre-planned trajectories. The Velocity Obstacle concept was used to obtain the boundary conditions required to avoid a dynamic obstacle. The simulation results validate the effectiveness of this algorithm to deal with typical collision states within a short period of time. Future work will include extending the trajectory time-scaling scheme to the robot manipulators. Real-time collision-free trajectory planning is more complex for many-DOFs manipulators because time-scaling functions should be considered for each joint. Velocity Obstacle are not suitable anymore and new criterions for collision boundary conditions must be implemented.
REFERENCES


