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

Ran Zhao, Daniel Sidobre

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

Ran Zhao[1,2], Daniel Sidobre[1,2]
[1]*CNRS, LAAS, 7 Avenue du Colonel Roche, F-31400 Toulouse, France*
[2]*Univ. de Toulouse, LAAS, F-31400 Toulouse, France*
{*ran.zhao@laas.fr, daniel.sidobre@laas.fr* }

Final version published in 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO), Colmar, 2015, pp. 414-421

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)(LaValle and Kuffner, 2001), or probabilistic roadmaps (PRMs)(Kavraki et al., 1996) 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. The controller must also accept to switch from an initial trajectory to follow a new one.

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 (Fiorini and Shillert, 1998) and was later extended to the case of reactive collision avoidance among multiple robots in (van den Berg et al., 2011a),(van den Berg et al., 2011b). 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 (Dahl and Nielsen, 1989) and (Morenon-Valenzuela, 2006) are used in execution of

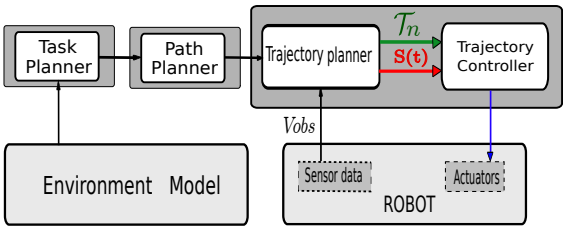


Figure 1: Trajectory planner serves as an intermediate level between the path planner and the low level controller. The planner can generate both new trajectories $\mathcal{T}_n$ and time-scaling functions $s(t)$

fast trajectories along a geometric path, where the motion is limited by torque constraints. (Szadeczky-Kardoss and Kiss, 2006) gives an on-line time-scaling methods based on the tracking error. In this study, we use the trajectory time-scaling schemes to avoid dynamic obstacles, which will not result in a change of path of the robot. Moreover, we apply a non-linear time-scaling to satisfy the kinematic constraints during the collision avoidance.

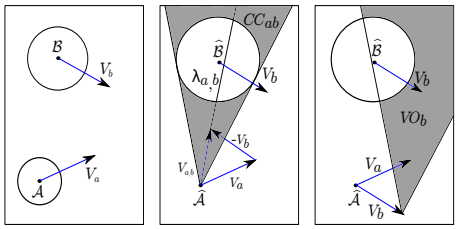
In this paper, we use a sequence of cubic polynomial functions to describe trajectories and we propose an efficient obstacle avoidance method in the trajectory level. We build an intermediate level between the path planner and the low level controller, which can read sensor information to generate non-linear time-scaling functions or 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 implement in real-time.

This rest of the paper is organized as follow: We present the trajectory representations and introduce notations in Section 2. We describe the obstacle avoidance algorithm in Section 3 and Section 4. Simulation results are highlighted in Section 5.

2 Notations and Representations

2.1 Trajectory Representations

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



(a) (b) (c)

Figure 2: (a) Two circular object with constant velocity. (b) The collision cone of object $\mathcal{A}$ and $\mathcal{B}$, any relative velocity lies in this cone will cause a collision. (c) The velocity obstacle VO_b on the absolute velocity of $\mathcal{A}$

then defined as:

$$\begin{aligned} \mathcal{T} : [t_I, t_F] &\longrightarrow \mathbb{R}^n \\ t &\longmapsto \mathcal{T}(t) \end{aligned} \tag{1}$$

where $\mathcal{T}(t) = X(t) = ({}_1X(t), {}_2X(t), \cdots, {}_nX(t))^T$ in Cartesian space from the time interval $[t_I, t_F]$ to $\mathbb{R}^n$.

We choose a particular series of 3^{rd} degree polynomial trajectories and name them as Soft Motion trajectories. A Soft Motion trajectory is a type V trajectory defined by Kröger(Kröger, 2010), which satisfies:

$$\begin{aligned} V(t) \in \mathbb{R}^n \quad |V(t)| \leq V_{max} \\ A(t) \in \mathbb{R}^n \quad |A(t)| \leq A_{max} \\ J(t) \in \mathbb{R}^n \quad |J(t)| \leq J_{max} \end{aligned} \tag{2}$$

where V, A and J represent the position, velocity, acceleration and jerk, respectively. $J_{max}, A_{max}, V_{max}$ are the kinematic constraints. Therefore, at a discrete instant, the trajectory can transfer the motion states while not exceeding the given motion bounds. The continuity class of the trajectory is $\mathcal{C}^2$. For the discussion of the next sections, a state of motion at an instant t_i is denoted as $M_i = (X_i, V_i, A_i)$.

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 $\mathcal{A}$ and $\mathcal{B}$, shown in Fig. 2(a) at time t_0, with constant velocity $\mathbf{v}_a$ and $\mathbf{v}_b$. Let circle $\mathcal{A}$ represent the robot, and $\mathcal{B}$ represent the obstacle. The point $\widehat{\mathcal{A}}$ is center of circle $\mathcal{A}$, $\widehat{\mathcal{B}}$ is mapped by enlarging $\mathcal{B}$ by the radius of $\mathcal{A}$, shown in Fig. 2(b). The *Collision Cone* $CC_{a,b}$ is then defined as the set of colliding relative velocities between $\widehat{\mathcal{A}}$ and $\widehat{\mathcal{B}}$:

$$CC_{a,b} = \left\{ \mathbf{v}_{a,b} \quad |\lambda_{a,b} \bigcap \widehat{\mathcal{B}} = \emptyset \right\} \tag{3}$$

where $\mathbf{v}_{a,b}$ is the relative velocity of $\widehat{\mathcal{A}}$ with respect to $\widehat{\mathcal{B}}$, $\mathbf{v}_{a,b} = \mathbf{v}_a - \mathbf{v}_b$, and $\lambda_{a,b}$ is the line of $\mathbf{v}_{a,b}$.

The shape of $CC_{a,b}$ is the planar sector with apex in $\widehat{\mathcal{A}}$, bounded by the two tangent from $\widehat{\mathcal{A}}$ to $\widehat{\mathcal{B}}$. 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 velocitiy of $\mathcal{A}$, This can be done simply by adding the velocity of $\mathcal{B}$:

$$VO_b = CC_{a,b} \oplus \mathbf{v}_b \tag{4}$$

where $\oplus$ is the Minkowski sum. When the velocity of robot $\mathcal{A}$ is included in the VO_b, 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 Collision Avoidance by Trajectory Time-Scaling

3.1 Time-Scaling Function

A time-scaling function $s = s(t)$ is an increasing function of time, because time cannot be reversed. Using the time-scaling function a new trajectory $\widetilde{\mathcal{T}}(t)$ can be defined such that $\widetilde{\mathcal{T}}(t) = \mathcal{T}(s)$. 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.

$$\widetilde{V}(t) = V(s)\dot{s} \tag{5}$$

$$\widetilde{A}(t) = A(s)\dot{s}^2 + V(s)\ddot{s} \tag{6}$$

$$\widetilde{J}(t) = J(s)\dot{s}^3 + 3A(s)\dot{s}\ddot{s} + V(s)\dddot{s} \tag{7}$$

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

Since time cannot reverse itself, $s(t)$ has to be a monotonic function. 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 Sidobre, 2005). We use the velocity of an Optimal Motion to represent the time variaiton. 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:

$$s(t) = \begin{cases} s(t)_i + \mathsf{A}t + \mathsf{J}\frac{t^2}{2}, & \mathsf{J} = -\mathsf{J}_{max} \\ s(t)_i + \mathsf{A}t, & \mathsf{J} = 0, \mathsf{A} = \mathsf{A}_{max} \\ s(t)_i + \mathsf{A}t + \mathsf{J}\frac{t^2}{2}, & \mathsf{J} = \mathsf{J}_{max} \end{cases} \tag{8}$$

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 $\mathsf{V}_{max} = 1$. 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: $|\widetilde{V}(t)| \leq V_{max}$, $|\widetilde{A}(t)| \leq A_{max}$, $|\widetilde{J}(t)| \leq J_{max}$. 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 VO_b : $\mathbf{v}_a \leq \mathbf{v}_{max}$, where $\mathbf{v}_{max}$ is the maximum velocity of $\mathcal{A}$ on the VO_b boundary. With the trajectory of robot, we could get the final value of the time-scaling function $s(t)_f$ which satisfy that: $\mathbf{v}_a(s) \leq \mathbf{v}_{max}$. Then we could define a Optimal Motion for the time-scaling function. The initial condition and final condition of the motion are $(0,0,s(t)_c)$ and $(1,0,s(t)_f)$, respectively. $s(t)_c$ 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:

$$\mathsf{J}(t) = \begin{cases} -\mathsf{J}_{max} \\ 0 \\ \mathsf{J}_{max} \end{cases} \qquad \mathsf{A}(t) = \begin{cases} -\mathsf{J}_{max}t \\ -\mathsf{A}_{max} \\ -\mathsf{A}_{max} + \mathsf{J}_{max}\frac{t^2}{2} \end{cases} \tag{9}$$

$$s(t) = \begin{cases} s(t)_c + \mathsf{A}(t)t - \mathsf{J}_{max}\frac{t^2}{2} & t \in [0, \frac{\mathsf{A}_{max}}{\mathsf{J}_{max}}] \\ s(t)_c + \mathsf{A}_{max}t & t \in [\frac{\mathsf{A}_{max}}{\mathsf{J}_{max}}, T_{op} - \frac{\mathsf{A}_{max}}{\mathsf{J}_{max}}] \\ s(t)_c + \mathsf{A}(t)t + \mathsf{J}_{max}\frac{t^2}{2} & t \in [T_{op} - \frac{\mathsf{A}_{max}}{\mathsf{J}_{max}}, T_{op}] \end{cases} \tag{10}$$

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

3.3 Multiple Obstacles Case

In the case of many 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 $\mathcal{A}$ and an obstacle $\mathcal{B}$ at time instant t. If the $d < \mathbf{v}_{a,b} * T_h$, we treat $\mathcal{B}$ as a imminent collision and the avoidance procedure will be first considered. If not, even $\mathbf{v}_a$ 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 $\mathcal{T}$ of robot $\mathcal{A}$, number of obstacles M, T_h

for $i = 0$ to M **do**
  Obtain the VO_i, $\mathbf{v}_a$ on $\mathcal{T}$ at time t
  if $\mathbf{v}_a \bigcap VO_i \neq \emptyset$ **then**
    Compute the relative velocity $\mathbf{v}_i$ between the robot and the obstacle i
    Compute the distance d_i between the robot and the obstacle i
    if $d_i < \mathbf{v}_i * T_h$ **then**
      Calculate the time-scaling factor $s(t)_{fi}$ for the single obstacle
    else
      $s(t)_f = s(t)_c$
    end if
  end if
end for
return the minimal factor $s(t)_{fmin}$, 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.

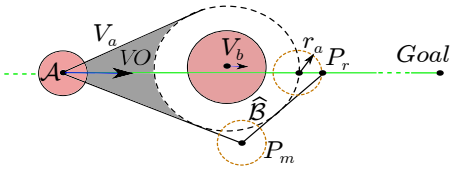


Figure 3: Choice of the middle and return waypoints for trajectory replanning

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 $\mathcal{A}$ and the obstacle $\mathcal{B}$ have the same path $\mathcal{P}(u)$ 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_r 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 $\mathcal{A}$ grazing $\mathcal{B}$. For sake of simplicity, we can enlarge the VO region slightly by increase the radius of $\widehat{\mathcal{B}}$ by a constant r (the black dashed circle in Fig. 3). In this study, we choose P_r as:

$$P_r = \widehat{\mathcal{B}} \bigcap \mathcal{P}(u)_{\mathcal{B}} \oplus r_a \frac{\mathbf{v}_{a,b}}{|\mathbf{v}_{a,b}|} \tag{11}$$

where $\mathcal{P}(u)_{\mathcal{B}}$ is part of the path $\mathcal{P}(u)$ that is behind the obstacle $\mathcal{B}$. Then the middle waypoint P_m is the point of intersection between the two tangents $\lambda_{\widehat{\mathcal{A}},\widehat{\mathcal{B}}}$ and $\lambda_{P_r,\widehat{\mathcal{B}}}$.

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 $\mathcal{T}_{ptp}$ obtained in the first step is feasible, but is not 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_r) and the smoothing at the intermediate via-point P_m. Firstly, the two straight-line trajectories $\mathcal{T}_{P_sP_m}$ and $\mathcal{T}_{P_mP_r}$ are computed respectively. Then we choose two points (P_{start}, P_{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_{start} and P_{end}. (l_{start}, l_{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_r, see Fig.4, $\overrightarrow{\mathbf{n}_i}$ being the normal unit vectors to the straight line P_sP_m, and $\mathcal{T}_s(t)$ the smoothed trajectory. Then the error ε can be defined as:

$$\varepsilon(t) = \min_{t\in[t_I,t_F]} ([\mathcal{T}_s(t) - P_m] \cdot \overrightarrow{\mathbf{n}_i}, [\mathcal{T}_s(t) - P_m] \cdot \overrightarrow{\mathbf{n}_{i+1}})$$
$$\varepsilon = \max(\varepsilon(t)) \tag{12}$$

To compute this error, we introduce another parameter ε_v, which is represented by the minimum distance between the vertex P_i and the trajectory $\mathcal{T}_s(t)$:

$$\varepsilon_v = \min_{t\in[t_I,t_F]} d(P_i, \mathcal{T}_s(t)) \tag{13}$$

As the start and end points of the blend we choose locate at the symmetric segments on each straight-line trajectory, the error $\mathcal{E}_v$ happens at the bisector of the angle α formed by the 3 adjacent points, Therefore,

$$\varepsilon_v = d(P_m, \mathcal{T}_{\frac{t_I+t_F}{2}}) \tag{14}$$

$$\varepsilon = \varepsilon_v * \sin\frac{\alpha}{2} \tag{15}$$

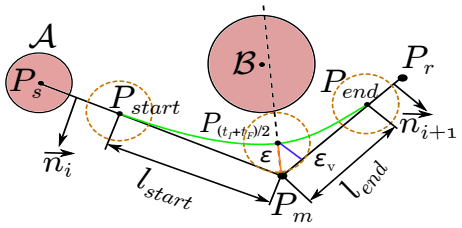


Figure 4: Error between the smoothed trajectory and pre-planed path

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

$$\varepsilon < d_{(p_m,\mathcal{B})} - r_{\mathcal{A}} \tag{16}$$

where $d_{(p_m,\mathcal{B})}$ is the distance between the point P_m and the obstacle $\mathcal{B}$. To achieve Eq. 16 we introduce a parameter δ, for a general case, it is defined as:

$$\delta = \frac{l_{start}}{|P_sP_m|} = \frac{l_{end}}{|P_mP_r|} \tag{17}$$

Thus $\delta \in [0,1]$. Then we can compute in a loop and get the δ_{max}.

4.2 Smooth Trajectory Generation

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

$$T_{imp} = max(f(M_s^j, M_e^j, J_{max}^j, A_{max}^j, V_{max}^j)) \quad j \in [1,n]$$

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_{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 $T_1 = T_2 = T_3 = \frac{T_{imp}}{3}$, 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 $T_{imp}/3$ must be a multiple of the period T, and in this study, T_{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 (J_2, T_1, T_2 and T_3):

$$A_e = J_3 T_3 + A_2 \tag{18}$$

$$V_e = J_3 \frac{T_3^2}{2} + A_2 T_3 + V_2 \tag{19}$$

$$X_e = J_3 \frac{T_3^3}{6} + A_2 \frac{T_3^2}{2} + V_2 T_3 + X_2 \tag{20}$$

$$T_{imp} = T_1 + T_2 + T_3 \tag{21}$$

where

$$\begin{aligned} A_2 =& J_2 T_2 + J_1 T_1 + A_s \\ V_2 =& J_2 \frac{T_2^2}{2} + (J_1 T_1 + A_s) T_2 + J_1 \frac{T_1^2}{2} + A_s T_1 + V_s \\ X_2 =& J_2 \frac{T_2^3}{6} + (J_1 T_1 + A_s) \frac{T_2^2}{2} + (J_1 \frac{T_1^2}{2} + A_s T_1 + V_s) T_2 \\ & + J_1 \frac{T_1^3}{6} + A_s \frac{T_1^2}{2} + V_s T_1 + X_s \end{aligned}$$

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

jerk-Bounded, Acceleration-Bounded, Velocity-Bounded Interpolants Now we derive the all-bounded trajectory given a fixed duration T_{imp}. 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_{imp} by computing the time-optimal trajectory on each aixs, 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_{imp} by unsaturated interpolants while maintaining the number of segments $\mathcal{N}_j$ on each aixs. We name it a *Slowing Down Motion*. Algorithm 2 shows the generation of all-bounded interpolants.

Algorithm 2 All-Bounded Interpolants Generation

Input: Motion states: M_s, M_e; number of DOFs: n; T_{imp}
Output: Jerk-Bounded, Acceleration-Bounded, Velocity-Bounded Interpolants

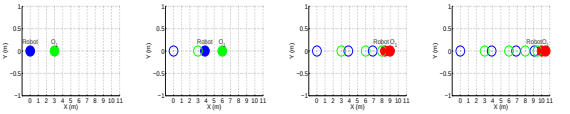
```
for j = 1 to n do
    Compute the time-optimal interpolants between M_s and M_e, then get the execution time T_j
    Get N_j and the execution time on each segment T_j^N
    if N_j = 0 then
        No motion on this axis, maintain the time-optimal interpolants
    else
        Enlarge T_j^N by T_j^N = T_j^N + (T_imp - T_j) / N_j
        Compute the new Jerk on each segment J_j^N
    end if
    Generate the interpolants with J_j^N and T_j^N
end for
```

5 Simulation Results

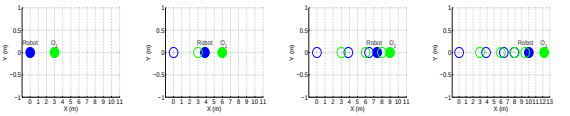
The initial trajectories from start $(0,0)$ to the goal $(10,0)$ for the robots were obtained by modeling the robots as unicycle. The maximum velocity, acceleration and jerk of the robot was set to 1.5 m/s, 3 m/s^2 and 9 m/s^3, respectively.

5.1 Single Obstacle Avoidance

Fig. 5 shows the simulation results for an obstacle moving along the path of the robot. The blue and

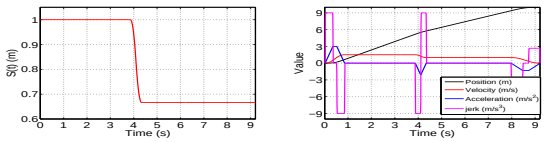


(a) t = 0 s (b) t = 3 s (c) t = 6 s (d) t = 7.51 s



(e) t = 0 s (f) t = 3 s (g) t = 6 s (h) t = 9.21 s

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.



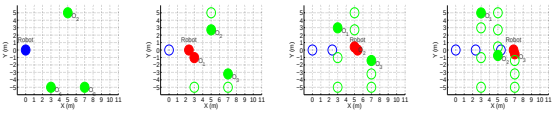
(a) (b)

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.

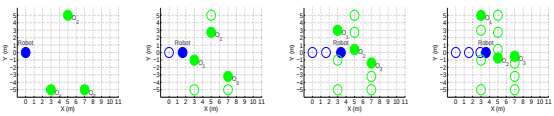
green circles represent the robot and the obstacle respectively. The situation shown in Fig. 5 takes place when the robot moves along a corridor. The obstacle moves at a velocity of $vx = 1\ m/s$ from the initial location $(3,0)$. As shown in Fig. 5(a)-5(d), the robot collides with the obstacle at 6-7.51 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-

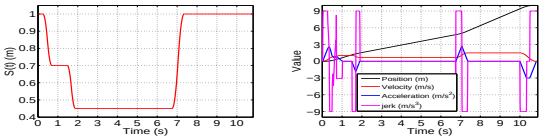


(a) t = 0 s (b) t = 2 s (c) t = 4 s (d) t = 5 s



(e) t = 0 s (f) t = 2 s (g) t = 4 s (h) t = 5 s

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



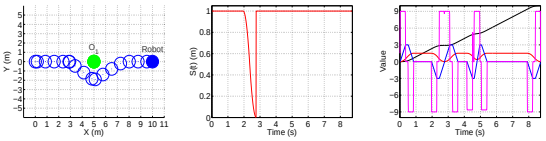
(a) (b)

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.

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. 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.



(a) (b) (c)

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.

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 $\mathcal{C}^2$ 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

Biagiotti, L. and Melchiorri, C. (2008). *Trajectory Planning for Automatic Machines and Robots*. Springer.

Broquère, X. and Sidobre, D. (2010). From motion planning to trajectory control with bounded jerk for service manipulator robots. In *IEEE Int. Conf. Robot. And Autom.*

Broquere, X., Sidobre, D., and Herrera-Aguilar, I. (2008). Soft motion trajectory planner for service manipulator robot. *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2808–2813.

Dahl, O. and Nielsen, L. (1989). Torque limited path following by on-line trajectory time scaling. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 1122–1128 vol.2.

Fiorini, P. and Shillert, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, 17:760–772.

Herrera, I. and Sidobre, D. (2005). On-line trajectory planning of robot manipulators end effector in cartesian space using quaternions. In *5th Int. Symposium on Measurement and Control in Robotics*.

Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580.

Kröger, T. (2010). *On-Line Trajectory Generation in Robotic Systems*, volume 58 of *Springer Tracts in Advanced Robotics*. Springer, Berlin, Heidelberg, Germany, first edition.

LaValle, S. M. and Kuffner, J. J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400.

Morenon-Valenzuela, J. (2006). Tracking control of on-line time-scaled trajectories for robot manipulators under constrained torques. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 19–24.

Szadeczky-Kardoss, E. and Kiss, B. (2006). Tracking error based on-line trajectory time scaling. In *Intelligent Engineering Systems, 2006. INES '06. Proceedings. International Conference on*, pages 80–85.

van den Berg, J., Guy, S., Lin, M., and Manocha, D. (2011a). Reciprocal n-body collision avoidance. In Pradalier, C., Siegwart, R., and Hirzinger, G., editors, *Robotics Research*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 3–19. Springer Berlin Heidelberg.

van den Berg, J., Snape, J., Guy, S., and Manocha, D. (2011b). Reciprocal collision avoidance with acceleration-velocity obstacles. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3475–3482.