INTECH
open science | open minds

OPEN ACCESS
ARTICLE

International Journal of Advanced Robotic Systems

Distance Computation Between Non-Holonomic Motions with Constant Accelerations

Regular Paper

Enrique J. Bernabeu[1,*], Angel Valera[1] and Javier Gomez-Moreno[1]

1 Instituto Universitario de Automática e Informática Industrial. Universitat Politècnica de València, Valencia, España
* Corresponding author E-mail: ebernabe@isa.upv.es

Received 30 May 2012; Accepted 12 Jun 2013

DOI: 10.5772/56760

Abstract A method for computing the distance between two moving robots or between a mobile robot and a dynamic obstacle with linear or arc-like motions and with constant accelerations is presented in this paper. This distance is obtained without stepping or discretizing the motions of the robots or obstacles. The robots and obstacles are modelled by convex hulls. This technique obtains the future instant in time when two moving objects will be at their minimum translational distance - i.e., at their minimum separation or maximum penetration (if they will collide). This distance and the future instant in time are computed in parallel. This method is intended to be run each time new information from the world is received and, consequently, it can be used for generating collision-free trajectories for non-holonomic mobile robots.

Keywords Continuous Distance Computation, Gilbert-Johnson-Keerthi (GJK) Algorithm, Mobile Robots, Non-Holonomic Motions, Continuous Collision Detection

1. Introduction

Detecting a collision in motion planning is still an open research area in robotics. Nowadays, powerful motion planners are developed, where collision tests are an unavoidable step and represent, in general, a decisive time-consuming part of the whole planning algorithm.

A recent example and with important social impact is shown by [1,2]. An estimated motion for an obstacle and a desired one for the robotized car Boss are stepped. Next, collision tests between the configurations of both objects at each considered time instant are run. The objects are modelled by boxes or circles. This collision-detection technique has several limitations, as shown by [3]. Nevertheless, this approach is common in the literature in order to detect collisions between mobile objects [4].

An extensive group of collision-detection methods is 'continuous collision detection' (CCD). In general, these methods provide, when there is a collision, the instant in time of the first contact [3,5–10]. In particular, when no collision is presented, the instant in time when both objects are at their closest position is returned by [10].

Another remarkable collision-detection technique is 'reciprocal velocity obstacle' (RVO) [11]. RVO is an oscillation-free navigation method for multiple agents with similar behaviours. RVO is an extension of the collision-detection technique 'vehicle obstacle' (VO) [12].

The contribution of this paper consists in obtaining the future instant in time when a robot and an obstacle or two robots, both in motion, will be at their minimum translational distance (MTD) of separation (if they will not collide) or penetration (if they will collide). The penetration distance verifies the definition given by [13].

Given that motions are not stepped, this method might be classified as a CCD technique, but really it is much more. When two robots or a robot and an obstacle will collide, the main advantage of computing the MTD of penetration and the associated instant in time versus the first time of contact is expressed in terms of generating a collision-avoidance trajectory. From the maximum penetration positions, a contact (or with a desired separation) position for the involved objects is obtained by translating, for instance, one of the robot's position by using the translational vector from the computed MTD [13]. In forcing this robot to be in this new position at the instant in time of the MTD, a new collision-free trajectory is proposed for that robot.

The robots and obstacles are modelled by bi-dimensional convex hulls. A trade-off between conservativeness in the models and computation costs has been set.

Considering the previous work in [14] as a collision detector, the main contributions of this paper are twofold: arc-like motions are now considered, and the involved robots and obstacles follow motions with non-null linear or angular accelerations.

The method in this paper is fast enough to be run as frequently as new information from the sensor system is received and, consequently, it is intended to be used as a module for collision detection and avoidance in trajectory planning algorithms dealing with non-holonomic robots.

For this paper, some experiments and simulations were run. The method was tested on the LEGO NXT Mindstorms platform. The velocity and acceleration of each robot is assumed to be measurable or estimable.

The main contributions of this paper with respect to [15] are an improved version of the support function, which is explained in Section 4, and a deepest analysis of our technique by applying it to real and simulated robots.

For convenience, some notations that will be used in this paper are listed here:

A,B: two mobile objects, modelled respectively, by a s-tope;
$S^A(t)$, $S^B(t)$: the sets of circles (s-topes) that describe the motions of objects A, B for $t\in[t_s,t_s+\Delta t]$;
$s_i^A(t)=(c_i^A(t),r_i^A)$; $s_j^B(t)=(c_j^B(t),r_j^M)$: any circle in $S^A(t)$, $S^B(t)$;
$S^A(t_s)$, $S^B(t_s)$: the start positions of the motions $S^A(t)$, $S^B(t)$;
$S^M(t)$: the Minkowski difference set between two motions;
$s_{ij}^M(t)=(c_{ij}^M(t),r_{ij}^M)$: any circle or element in set $S^M(t)$;
$p^M(\lambda)$,$p_i^M(\lambda)$ or $p_{ij}^M(\lambda)$ for $\lambda\in[0,1]$: the trajectory followed by a centre $c_{ij}^M(t)$ for $t\in[t_s,t_s+\Delta t]$;
O: the origin point;
$O^\perp$, O_c, $p^M(\lambda_c)$, $p_i^M(\lambda_c)$ or $p_{ij}^M(\lambda_c)$: the point in the trajectory $p^M(\lambda)$, $p_i^M(\lambda)$ or $p_{ij}^M(\lambda)$ which is the closest to O;
d_O^M: the maximum approach (separation or penetration) achieved by two objects in motion.
t_O^M: the future instant in time when two objects in motion will be at their maximum-approach positions;

2. Review of the GJK algorithm

Gilbert, Johnson and Keerthi [16] presented an algorithm, referred to as 'GJK', for computing, with linear complexity, the separation distance between two static polytopes.

A polytope is the convex hull of a finite set of points. The convex hull of the set $\Theta=\{p_0,p_1,\ldots,p_{n-1}\}$, $p_i\in\Re 3\ \forall i$, contains infinite points p that verify:

$$\left\{p:p=p_0+\sum_{i=1}^{n-1}\lambda_i\cdot(p_i-p_0),p_i\in\Theta,\ \lambda_i\in[0,1],\sum_{i=1}^{n-1}\lambda_i\le 1\right\} \quad (1)$$

The order of a polytope is the number of points in set Θ.

The GJK algorithm obtains the separation between the origin point O and the Minkowski difference of the involved polytopes. The Minkowski difference between polytopes A, B, defined by the sets $P^A=\{a_i\}$, $P^B=\{b_j\}$ with $a_i,b_j\in\Re^3$, i=0,1,…,n–1 j=0,1,…,m–1, is also a polytope defined by a set of $n\times m$ points $P^{A-B}=\{a_i-b_j: a_i\in P^A,\ b_j\in P^B,\ \forall i,j\}$.

In the GJK algorithm, set V_k always contains from one to four points of set P^{A-B}. Initially, some points from P^{A-B} are randomly assigned to V_k. Next, an iterative process starts.

The first step, called the 'sub-distance algorithm', consists of computing the distance from O to the polytope defined by the points in V_k. This distance is obtained by projecting O onto the mentioned polytope. This projected point is called '$O^\perp$'. The points in V_k, which are not required to describe the face, edge or vertex where $O^\perp$ is, are removed from V_k. In particular, after this step, if the set V_k contains four points because O is inside the polytope defined by V_k, then the GJK algorithm will finish immediately without returning a distance. This situation means that polytopes A and B are colliding. The GJK algorithm does not compute a penetration distance.

If the sub-distance algorithm finally returns a distance, the next step in the GJK algorithm consists of selecting the closest point in P^{A-B} to O in the direction $-O^{\perp}$. This point, $s_{A-B}(-O^{\perp})$, is found by applying the support h_{A-B} and mapping s_{A-B} functions:

$$\begin{aligned} h_{A-B}(-O^{\perp}) &= \max_{i,j}\left\{(a_i-b_j)\cdot(-O^{\perp}),\ a_i-b_j\in P^{A-B},\forall i,j\right\} \\ s_{A-B}(-O^{\perp}) &\in P^{A-B} : h_{A-B}(-O^{\perp}) = s_{A-B}(-O^{\perp})\cdot(-O^{\perp}) \end{aligned} \quad (2)$$

Now, if $||O^{\perp}||^2+h_{A-B}(-O^{\perp})=0$ (final condition) is true, then the GJK algorithm ends [16]. Otherwise, point $s_{A-B}(-O^{\perp})$ is added to V_k and the GJK algorithm iterates once more.

The GJK algorithm has a linear complexity $O(n+m)$, because the Minkowski difference is not computed before running the GJK algorithm. This is a consequence of the definition of the support function, since it verifies $h_{A-B}(-O^{\perp})=h_A(-O^{\perp})+h_B(O^{\perp})$.

The GJK algorithm finishes in less than five or six iterations even when dealing with high-order polytopes [16].

The GJK algorithm was updated in [17] to compute the separation or penetration distance for polytopes and spherically-extended polytopes (s-topes) [18].

3. Review of continuous distance computation for robots following linear motions with null accelerations

A technique for obtaining the future instant in time when a mobile robot and a dynamic obstacle will be at their minimum translational distance (MTD) is shown by [19]. Robots and obstacles are modelled indistinctively by polytopes or s-topes and follow linear motions at a constant speed. The distance and the instant in time are computed without stepping any robot or obstacle's motion.

An s-tope is the convex hull of a finite set of spheres - circles if bi-dimensional - $S=\{s_0,s_1,\ldots,s_{n-1}\}$ with $s_i=(c_i,r_i)$, where c_i is the centre and r_i is the radius. An s-tope contains an infinite set of swept spheres s (or circles) expressed by:

$$\left\{s=(c,r):\ c=c_0+\sum_{i=1}^{n-1}\lambda_i(c_i-c_0),\ \ r=r_0+\sum_{i=1}^{n-1}\lambda_i(r_i-r_0),\ \ s_i=(c_i,r_i)\in S,\ \lambda_i\in[0,1],\ \sum_{i=1}^{n-1}\lambda_i\le 1\right\} \quad (3)$$

If the radii r_i are zero, then (3) matches with the polytope definition (1) (i.e., a polytope is a particular case of an s-tope).

Let A and B be mobile robots or obstacles whose positions at t_s are given respectively by the sets of circles $S^A(t_s)=\{s_0^A(t_s),s_1^A(t_s),\ldots,s^A_{n-1}(t_s)\}$ and $S^B(t_s)=\{s_0^B(t_s),s_1^B(t_s),\ldots,s^B_{m-1}(t_s)\}$ where $c_i^A(t_s),c_j^B(t_s)\in\Re^2$ and r_i^A, $r_j^B\in\Re$ are, respectively, the centres and radii $\forall i,j$. The A and B constant velocities are given by the vectors v_A, $v_B\in\Re^2$.

Each one of the infinite intermediate positions of the objects A and B, $c_i^A(t)$ and $c_j^B(t)$, from t_s to $t_s+\Delta t$ are parameterized by $\lambda\in[0,1]$ as:

$$\begin{aligned} &c_i^A(t)=c_i^A(t_s)+\lambda\cdot\Delta t\cdot v_A;\ c_j^B(t)=c_j^B(t_s)+\lambda\cdot\Delta t\cdot v_B;\ \forall i,j \\ &\forall t: t=t_s+\lambda\cdot\Delta t:\ \ t\in[t_s,t_s+\Delta t]\ \text{ and }\lambda\in[0,1] \end{aligned} \quad (4)$$

The GJK-based algorithm in [19] computes the future instant in time when A and B will be at their MTD as the distance between O and the Minkowski difference of all the infinite intermediate positions of A and B.

The Minkowski difference between all the A and B infinite intermediate positions for $t\in[t_s,t_s+\Delta t]$, called '$S^M(t)$', is defined by the set of $n\times m$ circles $\{s_{ij}^M(t)\}$. $s_{ij}^M(t)$ is parameterized by $\lambda\in[0,1]$ as:

$$\{s_{ij}^M(t)\}=\left\{(c_{ij}^M(t),r_{ij}^M):\ c_{ij}^M(t)=c_i^A(t)-c_j^B(t),\ r_{ij}^M=r_i^A+r_i^B;\ \forall i,j,\ t=t_s+\lambda\cdot\Delta t,\ \lambda\in[0,1]\right\} \quad (5)$$

$c_i^A(t)$, $c_j^B(t)$ are defined in (4). Each circle $s_{ij}^M(t)$ for all $t\in[t_s,t_s+\Delta t]$ sweeps an area consisting of a rectangle whose ends are capped off with circles. This geometrical figure is referred to as a 'stadium' in [20]. Next, $S^M(t)$ is formed by $n\times m$ stadiums, and each one is described by three parameters: a start point $c_{ij}^M(t_s)=c_i^A(t_s)-c_j^B(t_s)$, a radius $r_{ij}^M=r_i^A+r_j^B$ and a linear axis $p^M(\lambda)\in\Re^2$. $p^M(\lambda)$ is parametrically defined by $\lambda\in[0,1]$ as:

$$\begin{aligned} p^M(\lambda)&=c_i^A(t_s+\lambda\Delta t)-c_j^B(t_s+\lambda\Delta t)-(c_i^A(t_s)-c_j^B(t_s))= \\ &=\lambda\cdot\Delta t(v_A-v_B) \end{aligned} \quad (6)$$

All the axes of the stadiums have the same length $||p^M(1)||=\Delta t\cdot||v_A-v_B||$. Figure 1 shows the Minkowski difference $S^M(t)$ with 4×2 stadiums from two constant-speed and linear motions.

The GJK-based algorithm in [19] mainly requires the definition of a new sub-distance algorithm and the support and mapping functions.

The sub-distance algorithm receives one or two stadiums in set V_k and computes the distance from O to the closest stadium. The distance between O and the stadium with the start point $c_{ab}^M(t_s)=c_a^A(t_s)-c_b^B(t_s)$ and the radius r_{ab}^M is determined by obtaining $O^{\perp}$ (i.e., by projecting O onto the stadium's axis). Therefore:

$$\begin{aligned} &O^{\perp}=c_a^A(t_s)-c_b^B(t_s)+p^M(\lambda_c) \\ &\text{with }\lambda_c=-\left((c_a^A(t_s)-c_b^A(t_s))\cdot p^M(1)\right)\Big/\|p^M(1)\|^2 \end{aligned} \quad (7)$$

Finally, the sub-distance algorithm returns the parameters $O^{\perp}$ and λ_c and rejects (if it receives two) the furthest stadium (i.e. this stadium is removed from V_k).

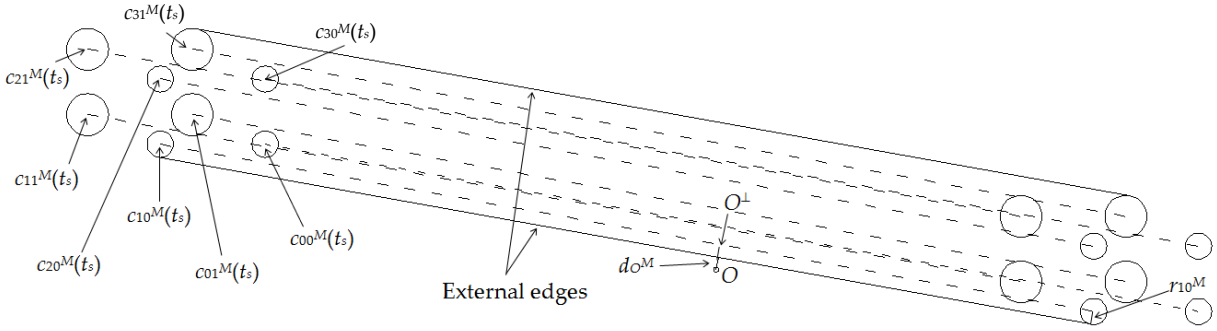

Figure 1. Eight stadiums in the Minkowski difference $S^M(t)$ from two constant-speed and linear motions. For clarity, only the axes (dashed lines) and the capping circles of the stadiums are depicted. $c_{10}^M(t_s)$, $c_{31}^M(t_s)$ are the start points of the two stadiums that hold the external edges of $S^M(t)$.

The support h_M and mapping s_M functions determine the closest stadium in $S^M(t)$ in the direction given by $-O^\perp$:

$$\begin{aligned} h_M(-O^\perp)&=\max_{i,j}\left\{c_{ij}^M(t_s)\cdot(-O^\perp)+r_{ij}^M\cdot|\,|O^\perp|\,|,\ (c_{ij}^M(t_s),r_{ij}^M)\in S^M(t)\right\} \\ s_M(-O^\perp)&=(c_s,r_s)\in S^M(t):h_M(-O^\perp)=-c_s\cdot O^\perp+r_s\cdot|\,|O^\perp|\,| \end{aligned} \quad (8)$$

where $s_M(-O^\perp)$ contains the start point of the stadium that generates the maximum in $h_M(-O^\perp)$.

The complexity of the algorithm in [19] is linear since $h_M(-O^\perp)=h_A(-O^\perp)+h_B(O^\perp)$ is true.

This GJK-based algorithm needs less than four iterations to find the instant in time, t_O^M, when A and B will be at their MTD, called 'd_O^M'. t_O^M and d_O^M are obtained from the distance between O to the closer external edge of $S^M(t)$ [19] (see figure 1), and mathematically by:

$$\begin{aligned} t_O^M&=t_s+\lambda_c\cdot\Delta t \\ d_O^M&=|\,|O^\perp|\,|-r_{ab}^M \end{aligned} \ ;\ \text{with } t_O^M\in[t_s,t_s+\Delta t] \text{ and } \lambda_c\in[0,1] \quad (9)$$

Note that r_{ab}^M is finally the radius of the closest external stadium to O (r_{ab}^M is r_{10}^M in figure 1).

Nevertheless, when the sub-distance algorithm receives two stadiums, and O is inside the area delimited by the axes of these stadiums, then the distance, d_O^M, between the inner O and the external edge is reformulated as $d_O^M=-(|\,|O^\perp|\,|+r_{ab}^M)$, where $O^\perp$ is the projection of O onto the axis of the stadium in V_k that is the closest to O. d_O^M is converted into a negative number and the radius has to be added, since it represents a translational distance of penetration [13]. t_O^M is computed as indicated by (9).

4. Continuous distance computation between two mobile objects with constant acceleration

A technique for computing the future instant in time when two mobile objects (robots and obstacles) are at their minimum translation distance (MTD) of separation or penetration is introduced in this section. The mentioned instant in time and MTD are computed without steeping any involved motion.

An object's motion is characterized by its initial position, its start and goal times (time span), its initial velocity, its constant acceleration, and its linear or arc-like (non-holonomic) path.

Three situations are considered in this paper: where the objects follow linear motions (case called LL); where one object follows an arc-like and where the other one follows a linear motion (AL); and where both objects follow arc-like motions (AA).

4.1 Problem Formulation

The s-topes for modelling the robots and obstacles and their motions are formally defined in this subsection.
Let us consider two mobile objects A and B, each modelled respectively by a s-tope. The A and B positions at the start time t_s are given by $S^A(t_s)$ and $S^B(t_s)$ respectively. $S^A(t_s)$ and $S^B(t_s)$ are defined by the set of circles $S^A(t_s)=\{s_0^A(t_s),s_1^A(t_s),\ldots,s_{n\_1}^A(t_s)\}$ and $S^B(t_s)=\{s_0^B(t_s),s_1^B(t_s),\ldots,s_{m\_1}^B(t_s)\}$ where $c_i^A(t_s)$, $c_j^B(t_s)\in\Re^2$ and r_i^A, $r_j^B\in\Re$ are the centres and radii of the circles $s_i^A(t_s)$ and $s_j^B(t_s)$, $\forall$i,j, respectively.

Motions are considered from t_s to $t_s+\Delta t$, where Δt is a time horizon.

When A and B are following linear motions, their initial velocities - i.e., at t_s - are $v_A(t_s)$ and $v_B(t_s)\in\Re^2$, respectively, and their constant accelerations are a_A, $a_B\in\Re$.

If A and B are following arc-like motions centred at c_A and $c_B\in\Re^2$, respectively, then each $c_i^A(t_s)$, $c_j^B(t_s)$, $\forall$i,j can be rewritten in polar coordinates by using the arc radius, ρ^A, ρ^B - i.e., its respective distance to the arc centre - and its initial angular position, $\theta_i^A(t_s)$, $\theta_j^B(t_s)$, as:

$$\begin{aligned} c_i^A(t_s)&=c_A+\rho_i^A(\cos(\theta_i^A(t_s)),\sin(\theta_i^A(t_s))) \\ c_j^B(t_s)&=c_B+\rho_j^B(\cos(\theta_j^B(t_s)),\sin(\theta_j^B(t_s))) \end{aligned} \ ;\ \forall i,j \quad (10)$$

A and B's angular speed at t_s are, respectively, $\omega_A(t_s)$ and $\omega_B(t_s)\in\Re$. Their constant angular accelerations are α_A and $\alpha_B\in\Re$, respectively.

Each one of the infinite intermediate positions of A from t_s to $t_s+\Delta t$ is parameterized by $\lambda\in[0,1]$ as:

$$c_i^A(t)=c_i^A(t_s)+\lambda\cdot\Delta t\cdot v_A(t_s)+0.5\cdot\lambda^2\cdot\Delta t^2\cdot a_A\cdot\hat{v}_A(t_s)\ ;\ \forall i \quad (11)$$

$$c_i^A(t)=c_A+\rho_i^A\left(\cos(\theta_i^A(t)),\sin(\theta_i^A(t))\right)\ ;\quad \forall i$$
$$\text{with } \theta_i^A(t)=\theta_i^A(t_s)+\lambda\cdot\Delta t\cdot w_A(t_s)+0.5\cdot\lambda^2\cdot\Delta t^2\cdot\alpha_A \quad (12)$$

$\forall t: t=t_s+\lambda\cdot\Delta t$, $\lambda\in[0,1]$, $t\in[t_s,t_s+\Delta t]$ and $\hat{v}_A(t_s)=v_A(t_s)/||v_A(t_s)||$. A is following a linear motion in case (11) and an arc-like motion in (12). (11) and (12) are analogously modified for B.

A constraint is contemplated from the motions in (11) and (12). A motion with a change from forward to backwards, and from counter-clockwise to clockwise, or vice versa, is not considered. In this case, the motion is properly divided.

The proposed GJK-based algorithms deal with $S^M(t)$, with $t\in[t_s,t_s+\Delta t]$. $S^M(t)$ is the Minkowski difference between all the A and B infinite intermediate positions while A and B are following their respective motions. $S^M(t)$ has been defined in (5) and, independently of the involved motions, $S^M(t)$ is defined by $n\times m$ stadiums, and each stadium is described by its start point $c_i^A(t_s)-c_j^B(t_s)$, a radius $r_{ij}^M=r_i^A+r_j^B$ and an axis $p^M(\lambda)\in\Re^2$ with $\lambda\in[0,1]$.

The future instant in time when A and B are located at their MTD of separation or penetration is obtained by computing the distance between O and $S^M(t)$ (specifically, from O to its closer external edge of $S^M(t)$).

4.2 Dealing with Two Straight-Line Motions (LL)

Two GJK-based algorithms are introduced in this sub-section. They deal with mobile objects (robot or obstacle) A and B with linear motions and constant accelerations. The *LL-GJK* algorithm obtains the future instant in time, t_O^M, when A and B will be at their MTD of separation, d_O^M, or it returns a failure if A and B will collide at $t\in(t_s,t_s+\Delta t]$. In case of a collision, the *LL$_{in}$-GJK* algorithm is then run and returns the future instant in time t_O^M when A and B will be at their MTD of penetration, d_O^M.

The Minkowski difference between these two motions is $S^M(t)$, $t\in[t_s,t_s+\Delta t]$, and it has $n\times m$ stadiums whose axes, $p^M(\lambda)$, are parabolic with (see (6)):

$$p^M(\lambda)=\lambda\cdot\Delta t\left(v_A(t_s)-v_B(t_s)\right)+ \\ +0.5\cdot\lambda^2\cdot\Delta t^2\left(a_A\cdot\hat{v}_A(t_s)-a_B\cdot\hat{v}_B(t_s)\right) \quad (13)$$

Any GJK-based algorithm to design requires: a sub-distance algorithm, the support and mapping functions, and a final condition.

The sub-distance algorithm computes the distance, d_O, between O and the axis of the stadium whose start point is in V_k. Let us consider the stadium with a start point $c_a^A(t_s)-c_b^B(t_s)$, then such a distance d_O is determined by finding the parameter λ_c that verifies:

$$d\left\|c_a^A(t_s)-c_b^B(t_s)+p^M(\lambda)\right\|/d\lambda\ =\ 0 \quad (14)$$

λ_c is found by applying the root-finding technique, termed the 'Secant method' [21]. Experimentally, $\lambda_0=0.45$ and $\lambda_1=0.55$ have been confirmed as good choices as initial values for the Secant method. The searching accuracy has been set to 10^{-6}.

After finding λ_c, $O_c\in\Re^2$ (the axis's closest point to O) and d_O are obtained as:

$$O_c=c_a^A(t_s)-c_b^A(t_s)+p^M(\lambda_c)\ ;\quad d_O=||O_c|| \quad (15)$$

Set V_k contains as maximum of two start points (stadiums). Only when the *LL-GJK* algorithm is being run and the set V_k contains two stadiums, the sub-distance algorithm first checks whether O is inside the area delimited by the axes of these two stadiums. If the result of this test is true, A and B will collide [16,17,19], and the *LL-GJK* algorithm finishes immediately with a failure and returns the set V_k. It the result of this test is false, the sub-distance algorithm returns the distance, d_O, from O to the closest stadium in V_k together with the corresponding parameters λ_c, O_c.

When the *LL$_{in}$-GJK* algorithm is being run, the sub-distance algorithm returns the distance from O to the furthest stadium d_O, if more than one in V_k and λ_c, O_c.

In any case, the sub-distance algorithm rejects and removes from V_k the stadium whose distance to O has not been returned.

The support h_M and mapping s_M functions are used for finding the furthest stadium in a given direction $\eta\in\Re^2$. This stadium is theoretically the candidate to be the closest stadium to O.

The GJK-based algorithm in [19] deals with stadiums whose axes are straight lines. Moreover, and for this reason, the support function in (8) works properly. Note that this support function uses the start points of the involved axes, $c_{ij}^M(t_s)$.

Now, we are dealing with non-convex (parabolic) axes. Therefore, the support function in (8) is not valid here.

This difficulty is overcome by applying the support function to different points of a given axis instead of just to the start point. These points are characterized by the golden ratio parameter r, with $r=(\sqrt{5}-1)/2$ [21].

Let $c_a^A(t_s)-c_b^B(t_s)$ be the start point of a stadium with a radius $r_a^A+r_b^B$ and axis $p^M(\lambda)$ with $\lambda\in[0,1]$, the support function h_M is applied to the points in the axis:

$$c_i^A(t_s)-c_j^B(t_s)+p^M(\mu)\ ;\quad \text{with } \mu=0,1-r,r,1,\lambda_c;\ \lambda_c\in[0,1] \quad (16)$$

Then, the support function h_M is defined as:

$$h_M(\eta,\mu)=\max_{i,j}\left\{\left(c_i^A(t_s)-c_j^B(t_s)+p^M(\mu)\right)\cdot\eta+(r_i^A+r_j^B)\cdot|\,|\eta|\,|\right\} \quad (17)$$
$$\text{with } (c_i^A(t_s),r_i^A)\in S^A(t_s);\ \ (c_j^B(t_s),r_j^B)\in S^B(t_s)$$

The parameter λ_c to be used in (17) is that returned by the last execution of the sub-distance algorithm.

The mapping function $s_M(\eta,\mu)$ contains the start point of the stadium that generates the maximum in $h_M(\eta,\mu)$. Consequently, according to μ, $s_M(\eta,\mu)$ represents up to five different stadiums. The stadium with the minimum distance to O is selected and called $s_M(\eta)$ in the *LL-GJK* algorithm. Conversely, $s_M(\eta)$ contains the stadium with the maximum distance to O for the *LL$_{in}$-GJK* algorithm. In any case, the other stadiums are rejected.

These new h_M and s_M functions are an improvement of those introduced in [15].

If the *LL-GJK* algorithm is being run, then η is substituted by $-O_c$ in (17). Conversely, if the *LL$_{in}$-GJK* algorithm is being run, then O_c is used in (17) instead of η.

The *LL-GJK* and *LL$_{in}$-GJK* algorithms finish when the stadium in $s_M(\eta)$ is the same as that represented by $s_M(\eta,\lambda_c)$ - i.e., $s_M(\eta)=s_M(\eta,\lambda_c)$ - and a final condition is also verified. Therefore, the *LL-GJK* algorithm finishes successfully, after obtaining the separation distance from O to the closest external edge in $S^M(t)$ (see figure 1), when the conditions $s_M(-O_c)=s_M(-O_c,\lambda_c)$ and $g_M(-O_c,\lambda_c)=0$ are true, with:

$$g_M(-O_c,\lambda_c)=|\,|O_c|\,|^2-\text{radius}(s_M(-O_c,\lambda_c))+h_M(-O_c,\lambda_c) \quad (18)$$

Given that the *LL$_{in}$-GJK* algorithm deals with O inside the area delimited by the axes from $S^M(t)$, it finishes when the conditions $s_M(O_c)=s_M(O_c,\lambda_c)$ and $\hat{g}_M(O_c,\lambda_c)=0$ are true, with:

$$\hat{g}_M(O_c,\lambda_c)=-|\,|O_c|\,|^2-\text{radius}(s_M(O_c,\lambda_c))+h_M(O_c,\lambda_c) \quad (19)$$

The function *radius* returns the radius of the stadium represented by $s_M(\eta,\lambda_c)$. The final conditions g_M and $\hat{g}_M$ are updated from their equivalent in [19].

If the *LL-GJK* algorithm finishes with a failure (*A* and *B* will collide), it returns a set V_k containing two stadiums. Next, the *LL$_{in}$-GJK* is run twice to obtain the MTD of penetration and the associated instant in time. Each execution is started by providing as initial set, namely an element from the mentioned set V_k.

Each execution of the *LL$_{in}$-GJK* algorithm returns a parameter λ_c and a negative distance $d_O{}^{M'}$ (computed as indicated in the last paragraph of Section 3). The maximum of these two negative distances holds $d_O{}^M$ (i.e., the MTD of penetration between *A* and *B*). From the execution that gives value to $d_O{}^M$, the future instant in time $t_O{}^M$ is calculated by using the returned λ_c as indicated by (9).

The parameter O_c returned by both the *LL-GJK* and *LL$_{in}$-GJK* algorithms holds a translational vector (i.e., if, for instance, the position of *A* at $t_O{}^M$ is translated as $d_O{}^M$ in the direction O_c, then *A* and *B* will be in contact at $t_O{}^M$).

For clarity, the *LL-GJK* and *LL$_{in}$-GJK* algorithms are presented in pseudocode.

A discrete motion representation of a mobile robot *A* and a dynamic obstacle *B*, both following linear motions with constant accelerations, is shown in figure 2.a. The positions where *A* and *B* are at their MTD of penetration, are depicted in red. The Minkowski difference $S^M(t)$ between both continuous motions is shown in figure 2.b. *A* is modelled by a fourth-order s-tope (polytope) and *B* is a second-order s-tope. $d_O{}^M$ and $t_O{}^M$ have been obtained in 5.1 µs in an Intel® Core™ i5 650 processor at 3.2 GHz.

Input: $S^A(t_s)$, $S^B(t_s)$, t_s, Δt, $p^M(\lambda)$, $r=(\sqrt{5}-1)/2$
Output: (λ_c, $t_O{}^M$, $d_O{}^M$,O_c,V_k) or (*failure*, V_k)

1: k=0, $V_k=\{c_0{}^A(t_s)-c_0{}^B(t_s),r_0{}^A+r_0{}^B\}$ with $(c_0{}^A(t_s),r_0{}^A)\in S^A(t_s)$,
and $(c_0{}^B(t_s),r_0{}^B)\in S^B(t_s)$
2: **do**
3: (λ_c,d_O,O_c,V_k,*O_inside*) ← *sub-distance_algorithm*(V_k)
4: **if** *O_inside* **then return**(*failure*,V_k)
5: *compute* $h_M(-O_c,\mu)$, $s_M(-O_c,\mu)$ for μ=0,1−r,r,1,λ_c
6: $s_M(-O_c)$=*select_closest_stadium_to_O*($s_M(-O_c,\mu)$)
7: **if** $s_M(-O_c)=s_M(-O_c,\lambda_c)$ & $g_M(-O_c,\lambda_c)=0$ **then** *exitloop* **endif**
8: $V_{k+1}=V_k\cup\{s_M(-O_c)\}$
9: k=k+1
10: **while** *true*
11: $t_O{}^M=t_s+\lambda_c\cdot\Delta t$; $d_O{}^M=d_O-(r_p{}^A+r_q{}^B)$
where $V_k=\{c_p{}^A(t_s)-c_q{}^B(t_s),r_p{}^A+r_q{}^B\}$;
with $(c_p{}^A(t_s),r_p{}^A)\in S^A(t_s)$, $(c_q{}^B(t_s),r_q{}^B)\in S^B(t_s)$
12: **return**(λ_c, $t_O{}^M$, $d_O{}^M$, O_c, V_k)

Algorithm 1. *LL-GJK* algorithm

Input: $S^A(t_s)$, $S^B(t_s)$, t_s, Δt, $p^M(\lambda)$, $r=(\sqrt{5}-1)/2$, and a one-element set V_{in}

Output: $(\lambda_c, d_{O^M}, O_c, V_k)$

1: k=0, V_k=V_{in}
2: **do**
3: $(\lambda_c, d_O, O_c, V_k) \leftarrow$ *sub-distance_in_algorithm*(V_k)
4: *compute* $h_M(O_c,\mu)$, $s_M(O_c,\mu)$ for μ=0,1−r,r,1,λ_c
5: $s_M(O_c)$=*select_furthest_stadium_to_O*$(s_M(O_c,\mu))$
6: **if** $s_M(O_c)$=$s_M(O_c,\lambda_c)$ & $\hat{g}_M(O_c,\lambda_c)$=0 **then** *exitloop* **endif**
7: V_{k+1}=$V_k \cup \{s_M(O_c)\}$
8: k=k+1
9: **while** *true*
10: d_{O^M}=−$(d_O+(r_{p^A}+r_{q^B}))$ with $V_k=\{c_{p^A}(t_s)-c_{q^B}(t_s), r_{p^A}+r_{q^B}\}$; and $(c_{p^A}(t_s), r_{p^A}) \in S^A(t_s)$, $(c_{q^B}(t_s), r_{q^B}) \in S^B(t_s)$
11: **return**$(\lambda_c, d_{O^M}, O_c, V_k)$

Algorithm 2. *LLin-GJK* algorithm

4.3 Dealing with Arc-like and Straight-line Motions (AL)

Now, a mobile robot A is following an arc-like motion while the other mobile robot or obstacle B follows a linear motion. These motions are described in subsection 4.1.

The future instant in time, when the mobile A and B will be at their MTD of separation, is obtained by applying the *AL-GJK* algorithm. If it fails, because A and B will collide at a time instant $t \in (t_s, t_s+\Delta t]$, then the *ALin-GJK* algorithm is run to obtain their MTD of penetration and the corresponding instant in time.

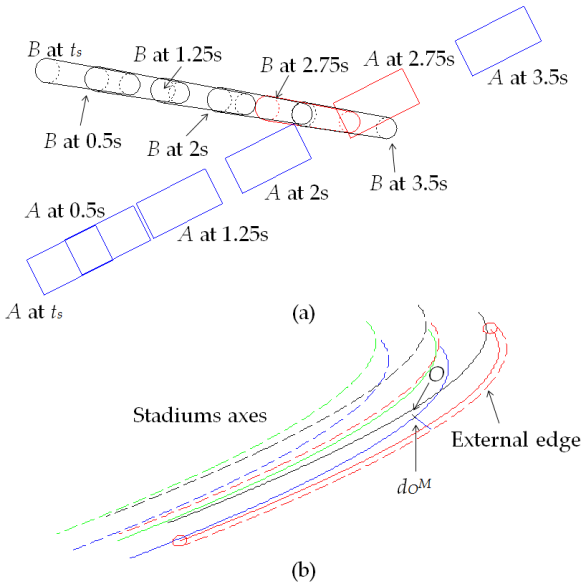

Figure 2. The distance between two mobile objects A and B following linear motions, with $||v_A(t_s)||$=2.2 m/s, a_A=1 m/s², $||v_B(t_s)||$=3 m/s, a_B=−0.5 m/s², t_s=0s and Δt=5s. (a) The A and B positions are only depicted at t_s, 0.5s, 1.25s, 2s, 2.75s and 3.5s. The positions at t_{O^M}=2.75s where A and B are at their MTD of penetration, d_{O^M}, are in red. (b) The Minkowski difference between the A and B motions. $S^M(t)$ has eight stadiums. For clarity, only the $S^M(t)$ external edge close to O, its associated capping circles, its distance, d_{O^M}, to O, and all the axes, are depicted.

As already mentioned, this future instant in time and the corresponding MTD are obtained by computing the distance from O to the closest external edge of the Minkowski difference of the involved motions.

The *AL-GJK* and *ALin-GJK* algorithms are, respectively, analogous to the *LL-GJK* and *LLin-GJK* algorithms. Only the subtle differences are pointed out here.

In accordance with the motion definition in (11) and (12), the Minkowski difference $S^M(t)$ between both motions has $n \times m$ stadiums whose axes are now cycloid-like. Furthermore, there are n different cycloid-like axes. Each of these n axes $p_i^M(\lambda) \in \Re^2$ $\forall$i is described by $\lambda \in [0,1]$ as:

$$p_i^M(\lambda)=\rho_i^A\left(\cos(\theta_i^A(t)),\sin(\theta_i^A(t))\right) - \lambda\cdot\Delta t\cdot v_B(t_s) - \\ -0.5\cdot\lambda^2\cdot\Delta t^2\cdot a_B\cdot\hat{v}_B(t_s);\ \ i=0,1...,n-1 \quad (20)$$

$\theta_i^A(t)$ depends upon λ - see (12) - with t=t_s+$\lambda\cdot\Delta t$. Let us consider a stadium described by $c_{a^A}(t_s)-c_{b^B}(t_s)$, $r_{a^A}+r_{b^B}$ and $p_{a^M}(\lambda)$ with $(c_{a^A}(t_s), r_{a^A}) \in S^A(t_s)$ and $(c_{b^B}(t_s), r_{b^B}) \in S^B(t_s)$. The sub-distance algorithm obtains the desired distance by finding the λ_c that minimizes $||c_A-c_{b^B}(t_s)+p_{a^M}(\lambda)||$ by solving:

$$d\left\|c_A-c_b^B(t_s)+p_a^M(\lambda)\right\| \Big/ d\lambda = 0 \quad (21)$$

λ_c is then found by applying the Secant method to (21), but this method will work properly if there is one minimum in $||c_A-c_{b^B}(t_s)+p_{a^M}(\lambda)||$. Given that the axes of the stadiums are cycloid-like, if A's angular displacement is lower than π, then $||c_A-c_{b^B}(t_s)+p_{a^M}(\lambda)||$ for all $\lambda \in [0,1]$ contains, in the worst case, one maximum and one minimum (apart from the extremes of the search interval). Consequently, if such a condition is false, then the A and B motions are properly divided before running the *AL-GJK* and *ALin-GJK* algorithms.

The support function h_M is also required to be modified as:

$$h_M(\eta,\mu)=\max_{\forall i,j}\left\{\left(c_A-c_j^B(t_s)+p_i^M(\mu)\right)\cdot\eta+(r_i^A+r_j^B)\cdot||\eta||\right\} \\ \text{with } \mu=0,1-r,r,1,\lambda_c;\ \ \lambda_c\in[0,1] \quad (22)$$

An example of the execution of these algorithms is shown in figure 3. A is modelled by a fourth-order s-tope (polytope), while B is a second-order s-tope. d_{O^M} and t_{O^M} have been obtained in 10.2 μs in an Intel® Core™ i5 650 processor at 3.2 GHz.

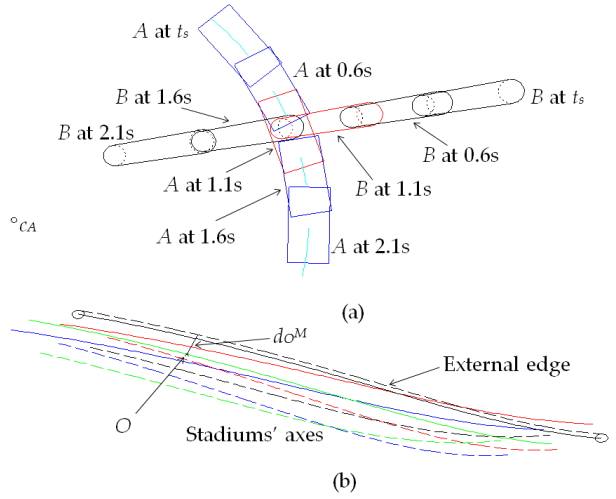

Figure 3. The distance between A and B. A follows an arc-like motion. B follows a linear motion. The motions are defined by $\omega_A(t_s)$=−18°/s, α_A=−0.5°/s², $||v_B(t_s)||$=3 m/s, a_B=1 m/s², t_s=0 s and Δt=5 s. (a) The A and B positions are stepped at t_s, 0.6s, 1.1s, 1.6s and 2.1s. The positions where A and B are at their MTD of penetration, do^M, are in red with to^M=1.1s. (b) The Minkowski difference between the A and B motions. $S^M(t)$ has eight stadiums and four different cycloid-like axes. For clarity, only the $S^M(t)$ external edge close to O, its associated capping circles, its distance, do^M, to O, and all the axes, are depicted.

4.4 Dealing with Two Arc-like Motions (AA)

In this subsection, a mobile robot A and a robot or obstacle B are following arc-like motions with constant angular accelerations. These motions have been described in subsection 4.1.

The future instant in time, when the mobile A and B will be at their MTD, is also obtained by computing the distance from O to the closest external edge of the Minkowski difference of the involved motions.

The *AA-GJK* algorithm computes the future instant in time when A and B will be at their MTD of separation. If this algorithm fails - i.e., if it detects that A and B will collide at $t\in(t_s,t_s+\Delta t]$ - then the *AA$_{in}$-GJK* algorithm computes the future instant in time when both objects will be at their MTD of penetration. These algorithms are analogous to the previous ones. Only the differences are shown here.

The axes of the stadiums in $S^M(t)$ are now rose-like (a rhodonea curve) [22]. $S^M(t)$ has $n\times m$ different axes $p_{ij}^M(\lambda)\in\Re^2\ \forall i,j$. These axes are parameterized by $\lambda\in[0,1]$ as:

$$p_{ij}^M(\lambda)=\rho_i^A(\cos(\theta_i^A(t)),\sin(\theta_i^A(t)))-\rho_j^B(\cos(\theta_j^B(t)),\sin(\theta_j^B(t)))$$
$$t=t_s+\lambda\cdot\Delta t\ ;\ t\in[t_s,t_s+\Delta t] \quad (23)$$

where $\theta_i^A(t)$ and $\theta_j^B(t)$ are described by (12).

Let $c_a^A(t_s)-c_b^B(t_s)$, $r_a^A+r_b^B$ and $p_{ab}^M(\lambda)$ be a stadium in $S^M(t)$ with $(c_a^A(t_s),r_a^A)\in S^A(t_s)$ and $(c_b^B(t_s),r_b^B)\in S^B(t_s)$. The sub-distance algorithm computes the distance from O to the mentioned stadium by finding λ_c that minimizes $||c_A-c_B+p_{ab}^M(\lambda)||$, i.e., by applying the Secant method to:

$$d\left\|c_A-c_B+p_{ab}^M(\lambda)\right\| \Big/ d\lambda \ = 0 \quad (24)$$

If A and B's angular displacements are lower than π, then $||c_A-c_B^B(t_s)+p_a^M(\lambda)||$ with $\lambda\in[0,1]$ contains, in the worst case, a maximum and a minimum (apart from the extremes of the interval of the search). If the mentioned condition is not verified, then any involved arc-like motions have to be divided before running any algorithm.

The support function h_M has to be updated to:

$$h_M(\eta,\mu)=\max_{\forall i,j}\left\{\left(c_A-c_B+p_{ij}^M(\mu)\right)\cdot\eta+(r_i^A+r_j^B)\cdot|\,|\eta|\,|\right\}$$
$$\text{with } \mu=0,1-r,r,1,\lambda_c;\quad \lambda_c\in[0,1] \quad (25)$$

An example dealing with a mobile robot A and a mobile obstacle B with arc-like motions and with constant angular accelerations is shown in figure 4. Its MTD of separation, do^M, and the corresponding instant in time, to^M, have been obtained in 9.7 μs in an Intel® Core™ i5 650 processor at 3.2 GHz.

5. Algorithm analysis and experimental results

All the support functions in this paper verify:

$$h_M(\eta,\mu)=h_{S^A(t)}(\eta,\mu)+h_{S^B(t)}(-\eta,\mu) \quad (26)$$

A's motion is described by n stadiums, while B's motion is defined by m stadiums. For this reason, the support functions can be applied separately to A's motion (i.e., $h_{S^A(t)}(\eta,\mu)$) and to B's motion (i.e., $h_{S^B(t)}(-\eta,\mu)$).

The condition in (26) is true and is proved for the case of A and B following, respectively, arc-like and linear motions. The proof is entirely similar to the other two cases.

In accordance with the definition of A's motion in (12), then:

$$h_{S^A(t)}(\eta,\mu)=\max_{\forall i}\left\{\left(c_A+\rho_i^A\left(\cos(\theta_i^A(t)),\sin(\theta_i^A(t))\right)\right)\eta+r_i^A|\,|\eta|\,|\right\} \quad (27)$$

with $\theta_i^A(t)=\theta_i^A(t_s)+\mu\cdot\Delta t\cdot w_A(t_s)+0.5\cdot\mu^2\cdot\Delta t^2\cdot\alpha_A$.

According to the definition of B's motion in (11), then:

$$h_{S^B(t)}(-\eta,\mu)=$$
$$\max_{\forall j}\left\{\left(c_j^B(t_s)+\mu\cdot\Delta t\cdot w_B(t_s)+0.5\mu^2\Delta t^2\alpha_B\right)(-\eta)+r_j^B|\,|\eta|\,|\right\} \quad (28)$$

Adding (27) and (28), the h_M definition given by (22) is reached. $p_i^M(\mu)$ verifies the definition in (20).

The condition in (26) has an important consequence: the Minkowski difference $S^M(t)$ does not need to be computed before running any of the *LL-GJK*, *LL$_{in}$-GJK*, *AL-GJK*, *AL$_{in}$-GJK*, *AA-GJK* or *AA$_{in}$-GJK* algorithms. Therefore, the complexity of all these algorithms is linear, namely O(n+m) instead of O(n×m)

These algorithms have been implemented in *C* and run in an Intel® Core™ i5 650 processor at 3.2 GHz. S-topes *A* and *B*, and their motions, have been randomly generated in order to analyse all of these algorithms. The s-tope orders, n and m, have been fixed in order to consider the following situations: n+m= 10, 50, 100, 250, 500, 1,000, 1,500 and 2,000. Approximately 5,000 different experiments have been run.

It is important to note that when the two involved s-topes do collide, then the corresponding *LL$_{in}$-GJK*, *AL$_{in}$- GJK* and *AA$_{in}$-GJK* algorithms will be run twice, returning two penetration distances. Next, the collision cases have a significant influence on the analysis of the algorithms.

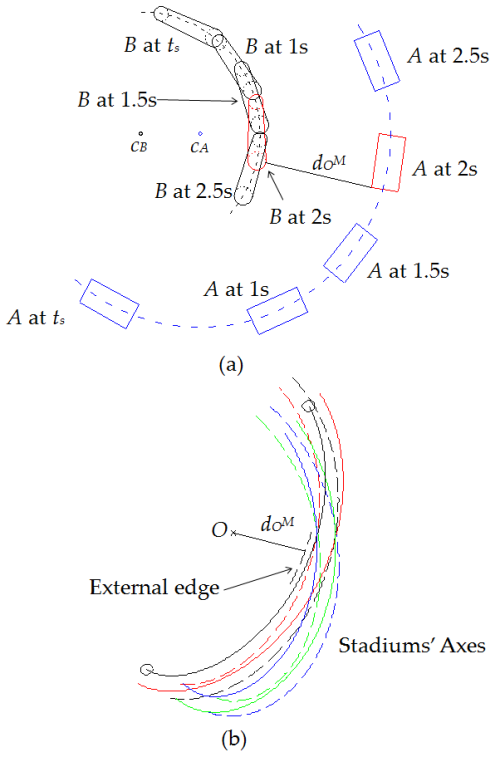

Figure 4. Distance between *A* and *B* with arc-like motions. The motions are described by $\omega_A(t_s)$=50°/s, α_A=5°/s^2, $\omega_B(t_s)$=−30°/s, a_B=−2.5°/s^2, t_s=0s and Δt=3s. (a) The *A* and *B* positions are stepped at t_s, 1s, 1.5s, 2s and 2.5s. The positions at t_{O^M}=2s, where *A* and *B* are at their MTD of separation, d_{O^M}, are in red. (b) The rose-like axes of the eight different stadiums in $S^M(t)$ and the distance d_{O^M} (at a different scale). For clarity, only the $S^M(t)$ external edge close to *O*, its associated capping circles, its distance, d_{O^M}, to *O*, and all the axes, are depicted.

The runtime of the algorithms per computed distance is shown in figure 5. The linear complexity of these algorithms is verified in figure 5. The sub-distance algorithm (the Secant method) requires more time when dealing with arc-like motions and, for this reason, the *LL-GJK* and *LL$_{in}$-GJK* algorithms present a minor computational cost.

The total number of iterations for all the algorithms is convex, as with the original GJK algorithm [16]. Figure 6 shows the average number of iterations per distance. The number of collision cases affects the linearity in the number of iterations.

The average number of iterations in the Secant Method for finding a minimum is shown in figure 7. The results in figure 7 show that the number of iterations in the Secant-method procedure is also convex.

Sometimes, the interval of searching in the Secant method contains a maximum and a minimum; moreover, if the Secant method first finds a maximum, then it is started again in order to search for the desired minimum, increasing the total number of iterations in the procedure. This situation is presented randomly as a consequence of how the data for this analysis has been created. Consequently, the experiments where a maximum is found by the Secant method have not been considered in the analysis shown in figures 5, 6 and 7.

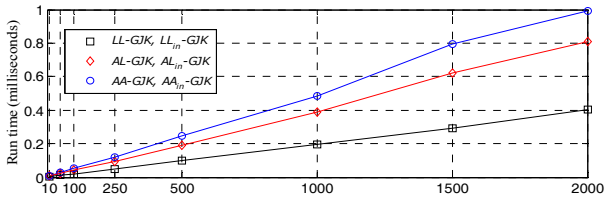

Figure 5. Computational cost of the algorithms by distance.

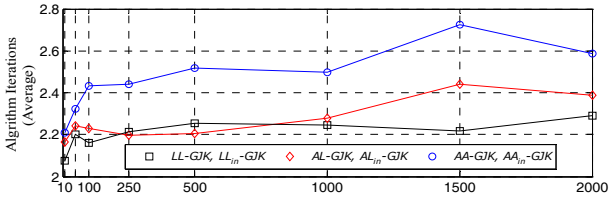

Figure 6. Average number of iterations in the algorithms.

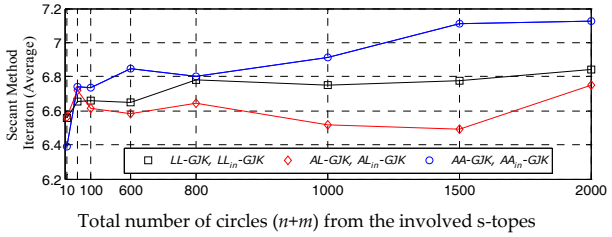

Figure 7. Average number of iterations in the Secant method, with the searching accuracy set to 10^{-6}.

In order to validate and analyse the proposed algorithms, two wheeled mobile robots have been used. These robots are based on the LEGO NXT Mindstorms platform. They are differential vehicles, and so they use independent velocities in both left and right wheels to move in the 2D plane.

The control unit of the Lego robots is based on an ARM7 microcontroller. This 32 bit CPU provides most of the control logic for the robot, including analogue-to-digital converters, timers, Bluetooth and USB communications ports, 256 Kbytes of FLASH memory and 64 Kbytes of static RAM. The actuators of these robots are high quality permanent-magnet dc motors that can provide torques of about 16.7 N.cm working at 117 rpm.

Different development tools can be chosen for programming the NXT microcontroller. In this work, RobotC has been used. It is a powerful programming language based on C that works in a Windows environment. RobotC is a cross-platform language that also allows for the debugging of the robot's applications in real-time.

Using RobotC, a pure-pursuit control algorithm for the path following was implemented. This algorithm follows any kind of path given by a series of points. In this case, linear and arc-like paths were specified.

Assuming that the initial position prior to the movement is known, the actual robot position can be estimated by using the local information of the robot motion (the wheels' velocities) obtained from several sensors in order to calculate the distance travelled from the initial point. This procedure has the benefit of a fast response time but also a disadvantage: between two position estimations, an error (between the actual and the estimated positions) is accumulated over time. Due to this, and after some navigation time, the position estimation can be very different from the actual value.

As it is important to have good local position estimation, the several sensors available on the robot - measuring the variables associated with the motion - should be used to increase the measurement accuracy, reduce the measurement noise and correct the deviation of the actual position value. In this case, the problems about how to integrate the different sensors into a single measurement that can be used by the control algorithm - taking into account the different accuracy and noise levels of the sensors or else how to determine which information should be discarded and which should be used to perform control - arise.

One of the most well-known and efficient techniques for data fusion is the Kalman Filter (KF) [23-24].

In this work, a linear KF has been used to obtain the global position of the robot. The implemented technique performs the sensor fusion locally by means of the wheel encoders (to measure the displacement of the left and right wheels), a gyroscope (to obtain the robot's angular velocity), a compass (to measure the heading angle) and two accelerometers placed above each wheel.

The main advantage of the linear KF proposed is its low computational cost. Because it uses small-sized matrices to obtain the Kalman gain, it can be calculated in real-time in the LEGO control unit.

Three different experiments were run. Each LEGO robot, L_A and L_B, has been simply modelled by a circle (a first-order s-tope). L_A and L_B's radii are r^A=110 and r^B=140 mm. Each radius is 25 mm greater than necessary for security.

Two linear motions with constant acceleration are considered in the first experiment (LL). L_A and L_B's motions' parameters are $c_A(t_s)$=(933,400), $v_A(t_s)$=(–58,0), a_A=0.42 mm/s^2, $c_B(t_s)$=(400,1051.6), $v_B(t_s)$=(0,–57), a_B=0.95 mm/s^2 with t_s=0 s and Δt=11 s. L_A and L_B's positions at t_s are given by $c_A(t_s)$ and $c_B(t_s)$, respectively. Figure 8 shows three plots: the centres' abscissa, their ordinates and the distance between them minus L_A and L_B's radii. The to^M and do^M obtained from the algorithm and the experiment differ, respectively, 55 ms and 5.5 mm. As the robots collide, the experimental MTD of penetration was obtained by running one of the robots with a delay. See figure 9. Figure 10 shows the control actions for the motors of robot L_B. Because the robot executes a linear trajectory with constant acceleration, the left and right wheels' control actions increase uniformly with the same slope.

In the AL experiment, L_A follows a linear motion with $c_A(t_s)$=(597,400), $v_A(t_s)$=(–60,0) and a_A=0.42 mm/s^2, while L_B follows an arc-like motion centred at (400,400) with an arc radius of 400 mm, $\theta^B(t_s)$=–90°, $w_B(t_s)$=24.2°/s and α_B=0.79°/s^2. $\theta^B(t_s)$ holds L_B's initial angular position. The time parameters are t_s=0 s and Δt=5 s. L_A and L_B's positions and their respective distances are shown in figure 11. The to^M and do^M obtained from the algorithm and the experiment differ, respectively, 98 ms and 3.9 mm. Some snapshots of this experiment are shown in figure 12. Figure 13 shows the control actions of robot L_A for this experiment.

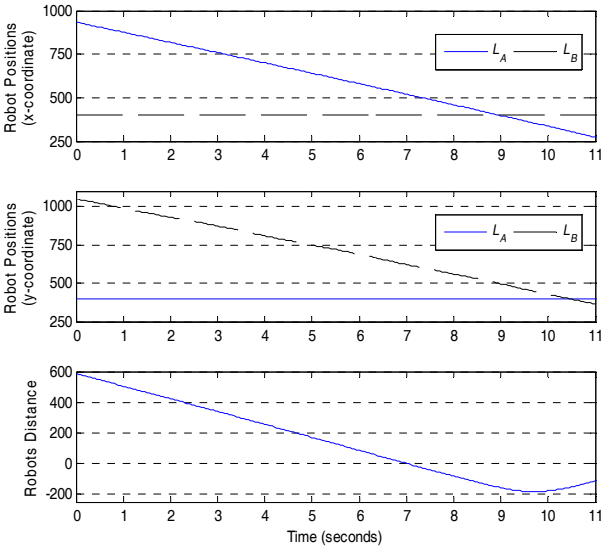

Figure 8. Experiment LL. The robots' positions and their distances (in millimetres) t_s=0 s. to $t_s+\Delta t$=11 s.

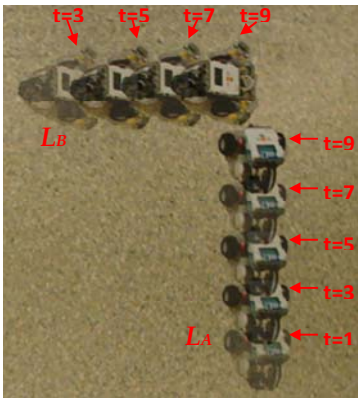

Figure 9. Experiment LL until collision. L_A's motion is delayed two seconds for appreciating the maximum penetration instant.

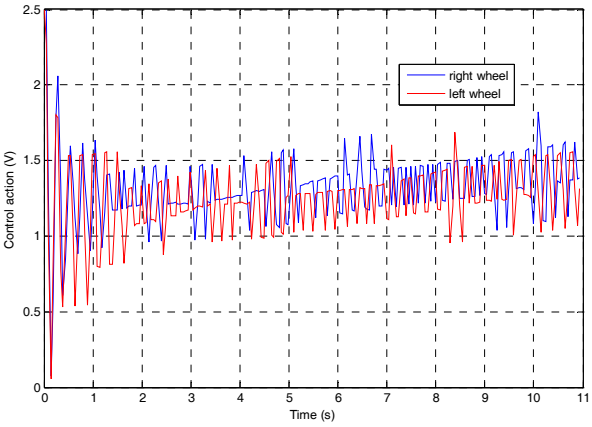

Figure 10. Control actions of the L_B robot's motors for the LL experiment.

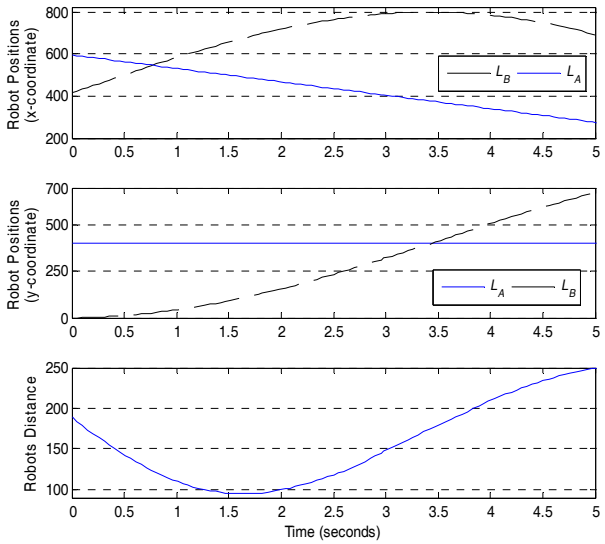

Figure 11. Experiment AL. The robots' positions and their distances (in millimetres) t_s=0 s. to $t_s+\Delta t$=5 s.

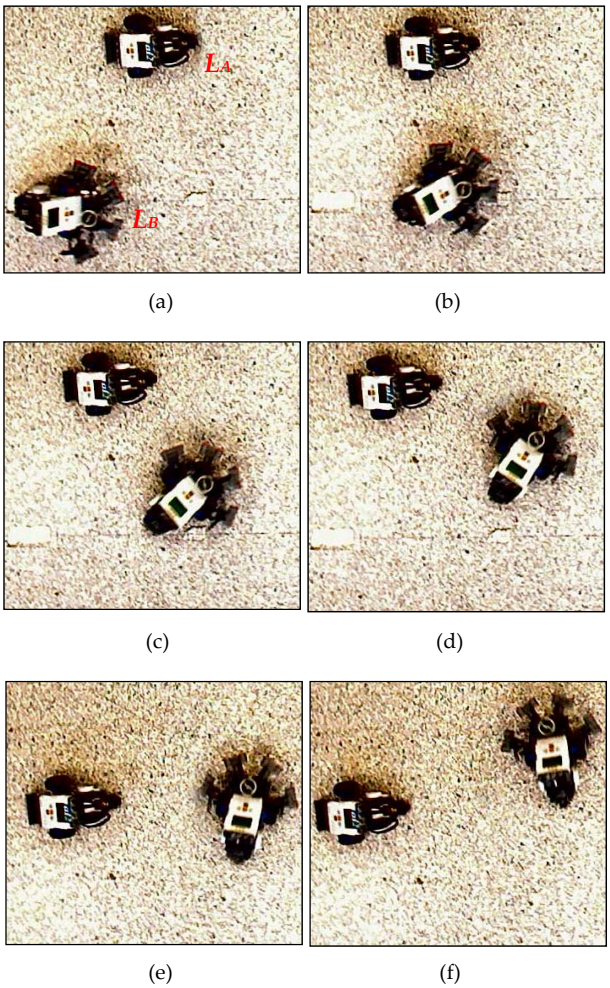

Figure 12. Snapshots from the AL experiment. The robots' positions at: (a) t_s=0 s, (b) 1 s, (c) 1.75 s (minimum separation) (d) 2.5 s, (e) 3.25 s and (f) 4 s

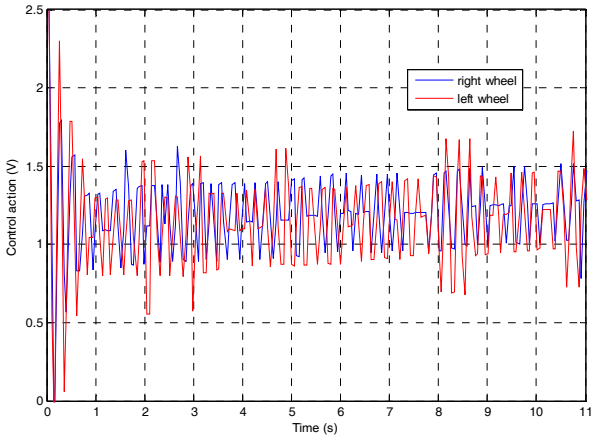

Figure 13. Control actions of the L_B robot's motors for the AL experiment.

In the AA experiment, robot L_A follows an arc-like motion centred at (1250,200) with an arc radius 200 mm, $\theta^A(t_s)$= –147.5°, $w_A(t_s)$= –19.4°/s and α_A=–0.21°/s². Robot L_B follows the arc-like motion shown in the AL experiment. The time parameters are t_s=0 s and Δt=6 s. L_A and L_B's positions and their respective distances are show in figure 14. The to^M and do^M obtained from the algorithm and the experiment differ, respectively, 75 ms and 0.9 mm. The experiment results are shown in figure 15. Figure 16 shows the control actions for the motors of robot L_B. Because the robot executes a circular (arc-like) trajectory with constant acceleration, the left and right wheels' control actions increase uniformly, but in this case the slopes of each action are different.

A simulation involving five different robots with non-holonomic motions is also shown. Robot R_1 is modelled by a circle and follows a linear motion with $c_{R1}(t_s)$=(19.5,45.6), r^{R1}=7 mm, $v_{R1}(t_s)$=(4.4,8.9), $||v_{R1}(t_s)||$=9.95 mm/s and a_{R1}=0.4 mm/s². Robot R_2 also follows a linear motion, but it is modelled by a fourth-order s-tope whose position S^{R2} at t_s is $S^{R2}(t_s)$={(169.6,180.3), (176.4,176.5), (181.5,185.1) and (174.6, 189.1)}. The radii of the circles in $S^{R2}(t_s)$ are, respectively, 2, 2, 3 and 3 mm. R_2's initial velocity and acceleration are $v_{R2}(t_s)$=(–10.3,–17,4), $||v_{R2}(t_s)||$=20.25 mm/s and a_{R2}=–0.8 mm/s². Robots R_3, R_4 and R_5 follow arc-like motions. R_3's motion is centred at (1,0) and it is modelled by a second-order s-tope. The two circles' centres are given in polar coordinates as $\theta_0^{R3}(t_s)$=14.4° where the arc radius ρ_0^{R3}=170 mm, and $\theta_1^{R3}(t_s)$=9.3° where ρ_1^{R3}=170.7 mm. The radii of the two circles are, respectively, 5 and 3 mm. R_3's initial angular speed and acceleration are $w_{R3}(t_s)$=6.1°/s and α_{R3}=–0.25°/s². R_4's motion is centred at (–115,120) and it is modelled by a third-order s-tope. R_4's position at t_s is given by $\theta_0^{R4}(t_s)$=15.5° where ρ_0^{R4}=240 mm, $\theta_1^{R4}(t_s)$=18.2° where ρ_1^{R4}=233.9 mm, and $\theta_2^{R4}(t_s)$=18.1° where ρ_2^{R4}=246.8 mm. The radii of the circles are, respectively, 3, 0, and 0 mm. R_4's initial angular speed and acceleration are $w_{R4}(t_s)$=–2.7°/s and α_{R4}=–0.1°/s². Finally, R_5's motion is centred at (210,210) and modelled by a fourth-order s-tope. R_5's position at t_s is given by $\theta_0^{R5}(t_s)$=–169.2° where ρ_0^{R5}=192 mm, $\theta_1^{R5}(t_s)$=–173.9° where ρ_1^{R5}=192.7 mm, $\theta_2^{R5}(t_s)$=–173.7° where ρ_2^{R4}=203.6 mm, and $\theta_3^{R5}(t_s)$=–169.2° where ρ_3^{R5}=203 mm. The radii of the four circles are 0 mm. R_5's initial angular speed and acceleration are $w_{R5}(t_s)$=4°/s, α_{R5}=0.15°/s². The time parameters are t_s=0 s and Δt=12 s.

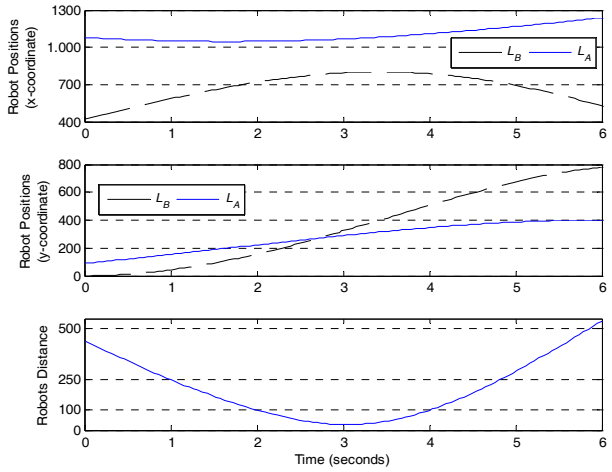

Figure 14. Experiment AA. The robots' positions and their distances (in millimetres) t_s=0 s. to t_s+Δt=6 s.

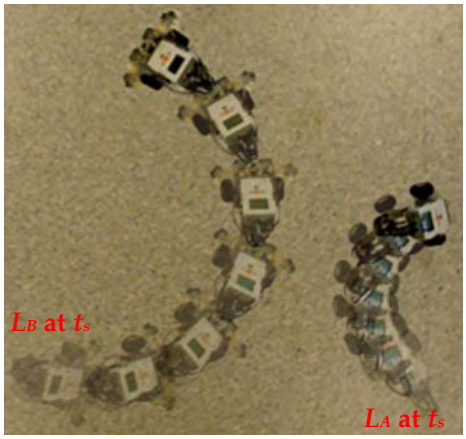

Figure 15. Experiment AA. L_A and L_B are shown at t_s=0, 1, 2, 3 (minimum separation), 4, 5 and 6s.

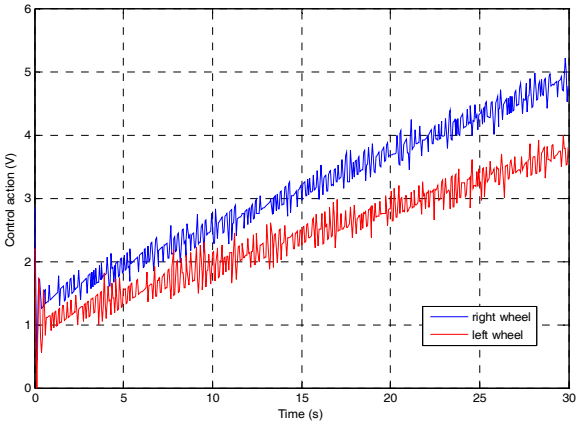

Figure 16. Control actions of robot L_B's motors for the AA experiment.

For clarity, the positions of the involved mobile robots have been stepped into two different figures (see figures 17.a and 17.b).

For each pair of robot motions, the future instant in time, to^M, when both robots will be at their MTD, do^M, has been computed. The results are shown in a matrix. See (29). The upper triangular submatrix contains the obtained do^M. The lower triangular submatrix shows the corresponding to^M. In this way, for instance, the MTD between robots R_2 and R_4 is 6.63 mm and it is presented at 5.36 s.

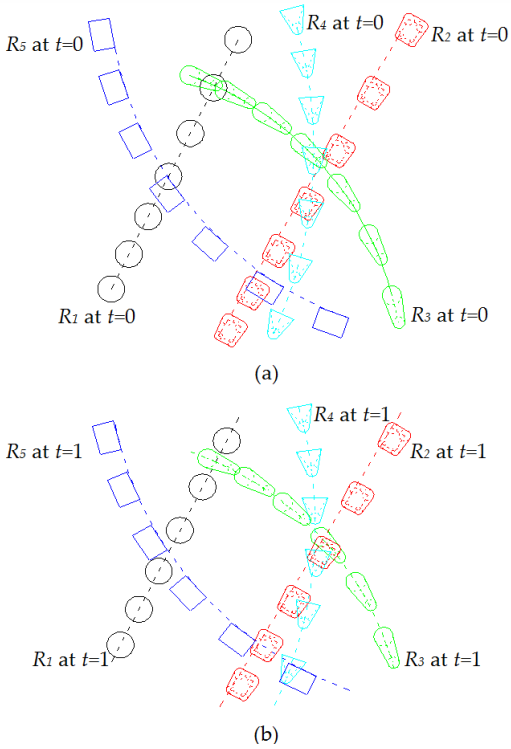

Figure 17. Simulation with five robots. (a) The robots' positions at t=0, 2, 4, 6, 8, 10 and 12 s are shown. (b) The positions at t=1, 3, 5, 7, 9 and 11 s are shown.

$$\begin{array}{c} \\ R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \end{array} \begin{array}{c} \begin{array}{ccccc} R_1 & R_2 & R_3 & R_4 & R_5 \end{array} \\ \begin{bmatrix} & 56.24 & -8.36 & 56.81 & -13.15 \\ 6.96 & & -14.98 & 6.63 & -15.06 \\ 10.26 & 4.72 & & -11.1 & 59.05 \\ 7.27 & 5.36 & 5.93 & & -10.33 \\ 5.44 & 9.5 & 7.23 & 10.88 & \end{bmatrix} \end{array} \quad (29)$$

The distances between each pair of robots while they are following their respective motions is shown in figure 18. Observing this figure, robot R_1 collides with R_3 and R_5; R_2 also collides with R_3 and R_5; R_3 collides with R_4; and R_4 collides with R_5 (see the negative distances in figure 18).

The evolution of the linear speed (in mm/s) of robots R_1 and R_2, and the angular speed (degree per second) of R_3, R_4 and R_5 is shown in figure 19.

6. Discussion

Our proposed collision detection technique is compared with some representative continuous collision detection (CCD) techniques.

Comparing our algorithms with the reciprocal volume object (RVO) in [11], the RVO is a robust collision-avoidance technique based on the VO concept [12]. VO contains the set of all the velocities of a robot that will result in a collision with another robot. Determining VO implies the computation of the Minkowski difference of the involved objects. Our technique does not compute the Minkowski difference, otherwise its complexity would be O(n×m) instead of O(n+m). The RVO considers neither explicitly non-holonomic motions nor the current agents' accelerations. RVO has been applied to thousands of disc-shaped agents, while our technique is suitable for robots modelled by convex-hulls defined by thousands of spheres (circles).

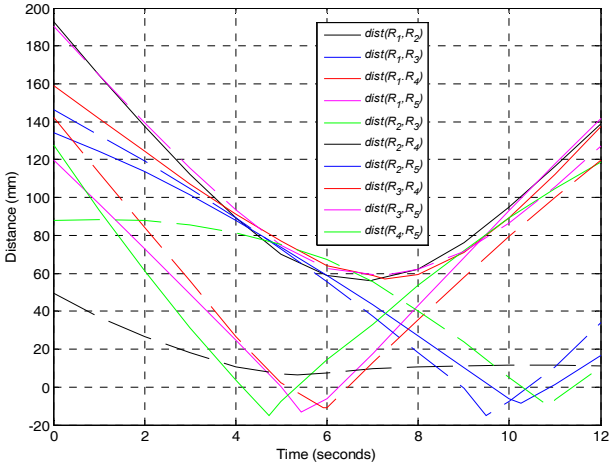

Figure 18. Distances between each pair of robots for $t \in [0,12]$.

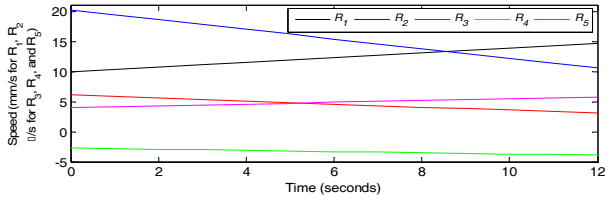

Figure 19. Speed evolution of the robots in motion for $t \in [0,12]$.

The work in [9] is also a CCD technique. The objects are modelled by swept sphere volumes and follow translational and rotational motions with constant velocities. The method returns the first time of contact if the objects collide. If not, the minimum separation is calculated. The method requires a separation distance computation function and a motion bound calculation. The method assumes that one of the objects is fixed (without motion) and computes a lower time bound. The mobile object is advanced according to the mentioned lower bound. This method does not compute penetration distances and does not consider trajectories with non-null accelerations.

The first contact time, the contact positions and the normal contact between two mobile rigid objects which are going to collide are obtained in [5]. The technique relies upon the effective interpolation of interval arithmetic and hierarchies of oriented bounding boxes. This CCD method finds the first time of contact by applying collision tests between object features (vertex, edge and face). The features are in motion and, iteratively, the time interval is reduced until the discovery of the instant in time when they are in contact. This technique does not consider non-null acceleration motions and is not intended to find the minimum separation when the objects do not collide. The method applies the same test each time with a minor interval of time.

7. Conclusions

This paper has given a method for more than detecting a collision between two mobile robots or between a mobile robot and a dynamic obstacle without stepping their motions. Specifically, this method obtains the future instant in time when two mobile objects will be at their minimum translational distance of separation or penetration (if collision). The mentioned translational distance and instant in time are computed in parallel. These results have been returned by certain proposed algorithms with linear complexity.

The involved robots and obstacles are modelled by spherically-extended polytopes (convex hulls). Their motions are non-holonomic (linear or arc-like) with constant accelerations. The positions of the robots or obstacles are assumed to be measurable and their motions (path, speed and acceleration) are estimable.

Some simulations and experiments with real robots have been run to conclude that the method is fast, robust, convex in the number of iterations, and accurate.

Additionally, our method is so fast that it can be run as frequently as any new information from the sensor system is received.

A direct extension of this work consists of updating these algorithms to compute, in the case of collision, the first time of contact. Another future and challenging work will be to deal with three-dimensional motions.

8. Acknowledgments

This work was partially funded by the Spanish government CICYT projects: DPI2010-20814-C02-02, and DPI2011-28507-C02-01.

9. References

[1] Ferguson D, Howard T.M, Likhachev M (2008) Motion Planning in Urban Environment: Part I, IEEE/RSJ Int. conf. on intell. robots and systems. pp. 1063-1069.

[2] Urmson C, Anhalt J, Bagnell D, Baker C, et al. (2008) Autonomous Driving in Urban Environments: Boss and the Urban Challenge. Journal of field rob. 25(8). pp. 425-466.

[3] Schwarzer F, Saha M, Latombe J-C (2005) Adaptive Dynamic Collision Checking for Single and Multiple Articulated Robots in Complex Environments. IEEE trans. on robotics 21(3). pp. 338-353.

[4] Jimenez P, Thomas F, Torras C (2001) 3D Collision Detection: A Survey. Comput. graph. 25. pp. 269-285.

[5] Redon S, Kheddar A, Coquillart S, (2002) Fast Continuous Collision Detection Between Rigid Bodies. Computer graphic forum 21(3). pp. 279-288.

[6] Canny J (1986) Collision Detection for Moving Polyhedra. IEEE trans. pattern anal. machine intell. 8(2). pp. 200–209.

[7] Choi Y-K, Wang W, Liu Y, Kim M-S (2006) Continuous Collision Detection for Two Moving Elliptic Disks. IEEE trans. robotics, 22(2). pp. 213-224.

[8] Buss S.R (2005) Collision Detection with Relative Screw Motion. The visual computer 21. pp. 41–58.

[9] Tang M, Kim Y.J, Manocha D (2009) C2A: Controlled Conservative Advancement for Continuous Collision Detection of Polygonal Models. IEEE Int. conf. on robotics and autom. pp. 849-854.

[10] Chakraborty N, Peng J, Akella S, Mitchell J.E (2008) Proximity Queries Between Convex Objects: An Interior Point Approach for Implicit Surfaces. IEEE trans. on robotics 24(1). pp. 211-220.

[11] Berg J.v-D, Lin M, Manocha D (2008) Reciprocal Velocity Obstacles for Real-time Multi-agent Navigation. IEEE Int. conf. on robotics and autom. pp. 1928-1935.

[12] Fiorini P, Shiller Z (1998) Motion Planning in Dynamic Environment Using Vehicle Obstacle. Int. journal of robotic research 17(7). pp. 760-772.

[13] Cameron S, Culley R.K (1986) Determining the Minimum Translational Distance between Two Convex Polyhedral. IEEE Int. conf. on robotics and autom. pp. 591-596.

[14] Bernabeu E.J (2009) Fast Generation of Multiple Collision-free and Linear Trajectories in Dynamic Environments. IEEE trans. on robotics 25(4). pp. 967-975.

[15] Bernabeu E.J (2010) Continuous Distance Computation for Planar Non-holonomic Motions with Constant Accelerations, IEEE Int. conf. on robotics and autom. pp. 4028-4034.

[16] Gilbert E.G, Johnson D.W, Keerthi S.S (1988) A Fast Procedure for Computing the Distance between Complex Objects in Three-dimensional Space. IEEE journal robot. and autom. 4(2). pp. 193-203.

[17] Bernabeu E.J, Tornero J (2002) Hough Transform for Distance Computation and Collision Avoidance. IEEE trans. on robotics and autom. 18(3). pp. 393-398.

[18] Hamlin G.J, Kelley R.B, Tornero J (1992) Efficient Distance Calculation Using Spherically-extended

Polytope (s-tope) Model. IEEE Int. conf. on robotics and autom. pp. 2502-2507.
[19] Bernabeu E.J, Tornero J, Tomizuka M (2001) Collision Prediction and Avoidance amidst Moving Objects for Trajectory Planning Applications. IEEE Int. conf. on robot. and automat. pp. 3801-3806.
[20] http://www.mathworld.wolfram.com/Stadium.html
[21] Mathews J.H (1987), Numerical Methods for Computer Science, Engineering and Mathematics. Prentice Hall.
[22] http://www.mathworld.wolfram.com/Rose.html
[23] Welch G, Bishop G (2007) An Introduction to the Kalman Filter. University of North Carolina at Chapel Hill, http://www.cs.unc.edu/~welch/kalman/
[24] Simon D (2006) Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches. John Wiley & Sons.