Transactions on Information and Communications Technologies vol 1, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

RobotSIM: An environment for obstacle avoidance control based on a potential field method

G. Gini & G.P. Picco

Department of Electronics and Information, Politecnico di Milano, Italy

ABSTRACT

We present a simulation system that moves a mobile robot in a world cluttered with obstacles using a potential field function. Following the method proposed by Bruce Krogh we take into account if the current position of the controlled system as well as its current velocity. The method relies on a double strategy for control: a time-optimal control, based on the bang-bang control, is performed in absence of obstacles, while a potential field control, enriched with some heuristics, is used when obstacles are present. The mathematical formulation is made in such a way that the first control strategy can be regarded as a subclass of the more general second strategy.

1. INTRODUCTION

In order to achieve the ultimate goal of creating autonomous robots, Robotics Science has to face with the basic problem of controlling robot motion. Unfortunately, this kind of task belongs to the class of problems we can efficiently perform as humans, but we can hardly take a robot to do.

The Obstacle Avoidance Control (OAC) problem deals with these issues. In particular, we can state the basic motion planning problem as in [3] "Given an initial position and orientation and a goal position and orientation of the robot in the workspace, generate a path specifying a continuous sequence of positions and orientations of the robot avoiding contact with the obstacles, starting at the initial position and orientation, and terminating at the goal position and orientation. Report failure if no such path exists."

This problem has been approached in a variety of ways. Following [2] we have three major classes:

- hypothesis and trial
- free space
- penalty functions

The first two classes generally require a huge amount of computational resources because of the non-trivial mathematics found in the algorithms, and moreover they need an a-priori description of the workspace in order to plan collision-free paths: so they are generally not suitable for real-time robot control.

By converse, the penalty function approach starts from another perspective; trajectory computation is not obtained handling the whole set of obstacles by means of functions that output a path, instead every obstacle has features that drive the robot trough a safe path. We can look at this approach as a shift from a centralized point of view to a distributed point of view, from global methods to local methods.

Among penalty function approaches, perhaps the most significant are potential field methods that rely on a basic idea borrowed from Classical Mechanics: think about associating mathematically defined force fields to relevant entities, i.e. obstacles and goal points In particular, an attractive force field is associated to goal point and repulsive force fields to obstacles. If we suppose that our robot is capable of feeling the sum of this forces, and of reacting to follow the minimum energy state, we own a system that is automatically driven by obstacles. We do not need an a-priori knowledge of world, because now we can compute the current forces configuration at every step, performing the appropriate action.

Several mathematical definitions of such force field have been proposed in literature, through the use of the notion of potential: perhaps the most known, and the only one implemented on a real robot manipulator system, has been proposed by Khatib[4].

Our system is a robot motion simulator, i.e. given a description of the workspace and initial and final positions for the robot, it shows a collision-free path through obstacles. It is based on a potential field approach slightly different from the traditional approach found in Khatib. Traditional potential field methods rely on potential functions generating a repulsive force inversely proportional to the distance from obstacle. This is a simple but not satisfying solution for many cases: for example, when the system is traveling away from the obstacle but very near to it, a high repulsive force is generated without need.

This drawback is eliminated by the method proposed by Bruce Krogh in [5]: it takes into account for the potential definition the current position of the controlled system as well as its current velocity. This can be regarded as a shift from a static view of the problem to a dynamic one. The method relies on a double strategy for control: a time-optimal control is performed in absence of obstacles, based on the bang-bang control described in [1], while a potential field control, enriched with some heuristics, is used when obstacles are present. The mathematical formulation is made in such a way that the first control strategy can be regarded as a subclass of the more general second strategy. A major drawback of potential field methods is the local minima problem, i.e. the system finds a solution (energy minimum in the physical analogy) that is not the desired solution (absolute energy minimum). Krogh's method offers heuristics to get the solution, but this is left unbounded from the rest of the method, allowing the user to substitute the proposed heuristic with a better strategy, if found.

We think Krogh's method is suitable for both mobile robots and manipulators, assuming the correct approximation, and it supports sensors integration and real-time control. Moreover, we have originally reinterpreted Krogh's method, developing a simulator that is constructed in such a way that other potential field methods can be put in the environment with few effort.

2. OBSTACLE AVOIDANCE CONTROL

The OAC finds its place among the problems to be solved to obtain autonomous robots able to accomplish tasks and to manage uncertainty and imprecision. Beside being an interesting study, autonomous robots will find a place in many practical tasks, for instance in dangerous environments. *Motion planning* is an important part in the autonomy of robots, because the ability to move in a safe and intelligent way is recognized as a fundamental ability. OAC is a big part of robot motion planning, and deals with the problem of moving the robot among obstacles to reach in a safe way its destination.

For autonomous robots the problem is getting computations in a short time and managing an unknown world. Here good candidates are local methods, and in particular the *potential field methods*. The basic idea is to associate forces to obstacles and to the goal: repulsive forces to obstacles, attractive forces to the goal position, so the robot can be attracted by its destination while being repulsed away from obstacles. In this formalization OAC can be solved with methods borrowed from physics.

The main drawback of the potential field methods is the existence of local minima. In those positions the robot is unable to continue its trajectory to the goal. The use of local knowledge, on the other end, makes the system able to take decisions in a short time on a reduced information. Usually potential field methods transform the robot into a point, situated in the robot hand or in the gravity center of the mobile robot. This assumption semplifies the computations and is often adequate in mobile robots.

2 1 - BASIC NOTATIONS IN ROBOTSIM

We formulate now the OAC problem as managed by Robotsim. We make the following assumptions:

- we consider 2D problems in a plane;
- the robot is a point;
- obstacles are only convex;
- we control the acceleration u, where $\|\mu\| \leq \alpha$ holds, and where a is the maximum acceleration we can get from the control system.
- the robot starts and stops with null velocities;
- we do not introduce costs, and we do not compute optimum control.

RobotSIM uses another approximation on the obstacles: they can be only of rectangular shape, and have any position and orientation. So the real shape of the obstacles is ignored, and we only make use of the approximating rectangle.

The strategy followed is choosing in any moment the control to avoid near obstacles. Choosing the control time by time is useful because the system does not need a knowledge of the world. So this strategy can find local solutions, because it is unable to look ahead, and remain there without getting out. For this reason we have to add heuristics to guide the system out of local minima.

The basis of the method consists in associating force fields (vectors) to any object in the world. If c is the acceleration vector, the equation describing the force fields is:

$$\vec{c} = \vec{c}_o + \gamma \vec{c}_g$$

where c is the acceleration vector, c_0 is the sum of the repulsive forces of the obstacles, c_g is the attractive force of the goal.

This is the basic idea of the 'impedance control' and is used by other approaches based on 'penalty function'. The potential fields, introduced in Physics, are defined as inversely proportional to the distance from the obstacle; so the potential can be used as a 'penalty function', that grows near the obstacle and reduces far away. We use this definition also to define the repulsive vector, that needs some more care because we want to take into account more than the nearest obstacle. As proposed in Krogh we define instead a 'Generalized Potential Field' (GPF). This provides a function using the distance from the obstacle as well as the velocity of the robot toward the obstacle; the function should be null when the robot is not moving towards obstacles. The definition of this potential is given in the case of the ideal plane. The GPF is defined for every obstacle and contributes to the definition of the repulsive vector.

Let us call:

A_i: the obstacle for which we calculate the GPF
x: the cartesian coordinates of the robot
p_i: the point of A_i nearest to x
d_i: the distance of p_i from x
v_i: the module of the velocity component along the segment x - p_i

The GPF depends on d_i and v_i, and in some way also from the maximum acceleration. It is different from 0 only if $v_i>0$.

At time t_0 with velocity v_i we want to compute the time needed to obtain the null velocity of the robot. The minimum and the maximum times are called *'minimum avoidance time'* τ_i and *'maximum avoidance time"* T_i

$$\tau_i = \frac{v_i(t_0)}{\alpha} \qquad T_i = \frac{2d_i}{v_i(t_0)}$$

The second definition is based on the idea that the robot travels for d_i and goes back to the departure with null velocity. These definitions are based on classic cinematics.

$$x = x_0 + vt \qquad v = v_0 + at$$

We can define the *'reserve avoidance time'* as the difference between τ_i and T_i. The *GPF is defined as the inverse of the reserve avoidance time:*

$$P_i(x,v) = \begin{cases} 0 & \text{if } v_i \le 0 \\ \dfrac{1}{T_i - \tau_i} = \dfrac{\alpha v_i}{2d_i\alpha - v_i^2} & \text{if } v_i > 0 \end{cases}$$

The GPF goes to infinity near the obstacle, and is 0 when no collisions are detected; moreover it grows also with the velocity. The GPF is undefined when the maximum avoidance time is less than the minimum avoidance time, because in this case no control is able to avoid the obstacle.

The repulsive vector c_0, generated by N obstacles, is the gradient of GPF with respect to the position:

$$\vec{c}_{o,i} = -\nabla_x P_i = \begin{cases} 0 & \text{if } v_i \le 0 \\ \dfrac{2P_i^2}{v_i}\vec{n}_i & \text{if } v_i > 0 \end{cases}$$

$$\vec{c}_o = \sum_{i=1}^{N} \vec{c}_{o.i}$$

The GPF is computed by Krogh using all the obstacles that are visible from the current position. We should now define the attractive component of the acceleration, in such a way that the time to go from departure to goal is minimum when no obstacles are found. In this case we can use the bang-bang control [1], that makes use of a double integrator.

2.2 CONTROL WITHOUT OBSTACLES

2.2.1 Classic formulation

The double integrator model is used to represent the movement of objects in a world without friction, where the motion equations are described through:

$$F(t) = m\ddot{y}(t)$$

The transfer function is:

$$G(s) = \frac{y(s)}{F(s)} = \frac{1}{ms^2}$$

Let be u(t), defined as u(t)=F(t)/m, the control; the Newton law becomes:

$$\ddot{y}(t)=u(t)$$

We represent the system as state variables:

$$x_1(t)=y(t)$$
$$x_2(t)=\dot{y}(t)$$

and we get:

$$\begin{cases}\dot{x}_1(t)=x_2(t)\\ \dot{x}_2(t)=u(t)\end{cases}$$

The control should be constrained; according to [1] $\|u(t)\|\le 1$, according to [4] any predefined value is acceptable. The control problem is so stated:

Given the system

$$\begin{cases}\dot{x}_1(t)=x_2(t)\\ \dot{x}_2(t)=u(t)\end{cases}$$

find a control to force the system from any state to the origin(0,0) in the minimum time.

The optimal control exists and is unique. Only 4 configurations of the control satisfy the requirements:

{-1},{+1},{-1,+1},{+1,-1}

Krogh uses those results to give his control law:

$$\vec{u}^*(v_g,d_g)=\begin{cases}\alpha\vec{n}_g & \text{if condition I}\\ -\alpha\vec{n}_g & \text{if condition II}\end{cases}$$

(where n_g is the versor from current position to the goal)

According to [1] the optimum time is:

$$t^*(x_1,x_2)=\begin{cases}x_2+\sqrt{4x_1+2x_2^2} & \text{if } x_1>-\frac{1}{2}x_2|x_2|\\ -x_2+\sqrt{-4x_1+2x_2^2} & \text{if } x_1<-\frac{1}{2}x_2|x_2|\\ |x_2| & \text{if } x_1=-\frac{1}{2}x_2|x_2|\end{cases}$$

generalized in Krogh as:

$$T^*(v_g,d_g)=\begin{cases}\dfrac{\sqrt{2v_g^2+4\alpha d_g}-v_g}{\alpha}\\ \dfrac{\sqrt{2v_g^2-4\alpha d_g}+v_g}{\alpha}\end{cases}$$

To better assess the method we have also provided and intuitive formulation of the problem.

2.2.2 Intuitive formulation

We assume that the acceleration given is always the maximum; u* is:

$$\vec{u}^*(v_g,d_g)=\begin{cases}\alpha\vec{n}_g & \text{(centripetal acceleration)}\\ -\alpha\vec{n}_g & \text{(centrifugal acceleration)}\end{cases}$$

Here the acceleration is directed from the point to the goal: the control is a force field situated in the goal, with a direction centripetal if we want to reach the goal, centrifugal if we want to go away.

We define:

- d_g, *distance from current position to the goal;*

- d_f, *stop distance*, $d_f = \frac{v_g^2}{2\alpha}$, distance necessary to stop the robot moving at velocity v_g.

We set a reference system in the current robot position, with the x axis directed to the goal. Suppose to be at a distance d_g from the goal and to travel toward it with velocity v_g, so that in the reference $v_g>0$.

We distinguish 3 cases:

1) $d_f = d_g$

In this case it is possible to reach the goal and to stop using the centrifugal control, which will stop the robot progressively reducing its velocity.

2) $d_f < d_g$

In this case we have to accelerate the robot to augment d_f, until we reach the case 1. This is done using the centripetal control.

3) $d_f > d_g$

This is the most complicated, because in this case the robot velocity is so high that it is impossibile to have zero velocity in the goal. We start with a centripetal control, and the robot is accelerated toward the goal. When the goal has been surpassed, the centripetal field attracts the robot to the goal but the velocity sends it away. In this case the centripetal field stops the robot and moves it back to the goal. If the only law were the application of the centripetal law, the robot would bounce around the goal. But at a point we will have $d_f = d_g$, so from this point we change the field to centrifugal, and we are able to reduce the robot velocity to arrive in the goal with null velocity.

In a similar way we reason for $v_g < 0$.

Concluding, the bang-bang control is characterized by:

- the controlled variable can be interpreted as an acceleration field;
- the first field applied is the centripetal, then the centrifugal;
- the sign is changed in correspondance with the *inversion condition* $d_f = d_g$.

We are able now to compute the distance to stop and the time T*, as illustrated in Krogh.

Computing the distance to stop

At the final time $t = t_f$, we want $v_{final} = 0$; at the initial time $x_g = d_f$. From the first condition we have:

$$v(t) \Rightarrow v_g - \alpha t = 0 \Rightarrow t = \frac{v_g}{\alpha}$$

From the second condition:

$$x(t) \Rightarrow d_f = v_g t - \tfrac{1}{2}\alpha t^2$$

Substituting we get $d_f = \frac{v_g^2}{2\alpha}$.

*Computing T**

If $d_f>d_g$, T* is the sum of t_1 and t_2, used to travel d_1 e d_2 respectively. If v_p is the robot velocity in P, and v_g the positive initial velocity, for the first part we have

$$v_p = v_g + \alpha t \qquad\qquad d_1 = v_g t + \tfrac{1}{2}\alpha t^2$$

and for the second :

$$d_2 = v_p t_2 - \tfrac{1}{2}\alpha t_2^2$$

(where the acceleration has a different sign).

Since

$$t_2 = \frac{v_p}{\alpha} = \frac{v_g + \alpha t_1}{\alpha}$$

we get

$$d_g = d_1 + d_2 = 2v_g t_1 + \alpha t_1^2 + \frac{1}{2}\frac{v_g^2}{\alpha}$$

We resolve with respect to t_1 :

$$t_1 = -\frac{v_g}{\alpha} \mp \frac{\sqrt{2v_g^2 + 4\alpha d_g}}{2\alpha}$$

$$T^* = t_1 + t_2 = \frac{v_g}{\alpha} + 2t_1 = \frac{\sqrt{4\alpha d_g + 2v_g^2} - v_g}{\alpha}$$

existence condition: $d_g \geq -\frac{1}{2}\frac{v_g^2}{\alpha} \Leftrightarrow d_f \leq d_g$ hypothesis

2.3. CONTROL WITH OBSTACLES

We define:

d_g: distance between the current position and the goal

n_g: versor from the current position to the goal

o_g: versor perpendicular to n_g according to the right-hand rule

v_g: proiection of the velocity on n_g

v_o: projection of the velocity on o_g

The attractive vector is:

$$\vec{c}_g = w(v_g, d_g)\vec{n}_g + w(v_o, 0)\vec{o}_g$$

We make the hypothesis that without obstacles the control is the scalar product between maximum acceleration and the direction versor.

- we define an *acceleration versor* with direction and versus as the *acceleration vector*
- the control u is the scalar product between the maximum acceleration and this versor

In fact we compute the repulsive and attractive vectors using the module; after that the module is no more used to control the system.

In the control we combine the direction and versus of the control u (obtained from the acceleration) with the maximum available acceleration, to minimize the execution time. So the control is:

$$u^*(v_g, d_g) = \alpha\frac{\vec{c}}{|\vec{c}|}$$

If the attraction and repulsion versors are co-linear, with same module and opposite versus, the resulting acceleration vector is null and the system stops without reaching the goal.

To avoid this situation Krogh temporarly sets as the goal a subgoal determined by the edge of the obstacle lying between x and x_g which is closest to x_g. It means that:

- we consider the obstacle nearest to the current position,;

Transactions on Information and Communications Technologies vol 1, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

- we analyze the 2 'edges' of this obstacle, intersecting the perimeter of the obstacle with one of the tangents to the obstacle going through the current point.
- the subgoal is the edge nearer to the real goal.

This heuristics is modified by the *edge factor* , set by the user in RobotSIM, which guarantees a minimum distance from the obstacle.

The subgoal is defined as:

$$\bar{x}'_g = \bar{e}_g + \varepsilon \bar{s}_n$$

where x'_g is the subgoal, e_g is the edge point, and s_n is the direction normal to the obstacle and computed from the edge point. The control algorithm verifies at every iteration the obstacles; in presence of obstacles it sets a subgoal and produces the cinematic values.

3. ENVIRONMENT DESCRIPTION

RobotSIM is actually made of two applications:

- **RobotSIM.** This is the m⌣in application, containing the real simulation environment.
- **RobotSIM Viewer.** This application is used to allow the user to compare the graphical results of many simulations in a full windowed environment. This is because RobotSIM allows the execution and displaying of only one simulation at a time.

Both applications run under Microsoft Windows 3.x, taking great advantage of the graphical and interactive features of such a system. In the sequel, we will refer to main application RobotSIM, if not otherwise specified.

After launching RobotSIM, the following windows are displayed:

- **Main window.** This window carries the menu driving the whole application.
- **Graphic output window.** During the simulation execution, this window displays the collision-free path among obstacles. Moreover, it is used to interactively describe the workspace configuration.
- **Numerical output window.** During the simulation execution this window displays the current values of output variables. It is blank during the workspace definition phase.
- **Auxiliary window.** This window is used to display the workspace coordinates currently pointed by mouse.

The graphical output window plays a central role in the application, since the real-time trajectory tracing is the most impressive mean to display the simulation results. Its implementation is quite sophisticated, since it realizes the mapping from the Cartesian reference to the display reference, providing scaling and scrolling. Obviously, user interaction is not limited to windows handling: in the following we examine the various kinds of interaction the user is allowed to perform.

3.1 WORKSPACE DESCRIPTION

When RobotSIM is started it comes up with a default workspace description, containing only a start and a goal point, with no obstacle. Serious applications surely will involve the description of a more complex workspace including obstacles, and possibly with a different position of the start and goal point ("special points" in the sequel). RobotSIM offers two options to do this:

- **Mouse interaction.** The user can move the special points and the obstacles on the screen by clicking & dragging with the mouse. Moreover, obstacles' dimensions can be modified by clicking & dragging on obstacles' vertexes or sides, with real-time visual feedback. This is intended for gross positioning and dimensioning, due to the intrinsic low resolution of mouse and graphical device.

- **Form filling.** The user can fill in the right numeric values of coordinates in specifying position and/or obstacle dimensions. The object whose features are under update can be selected by mouse or by menu selection. This method is intended for fine positioning and dimensioning, possibly using data retrieved by on-the-field measurement.

3.2 SIMULATION PARAMETERS

Krogh method relies on the definition of values for some characteristic parameters that rule or tune the simulation behavior. Parameters' definition is performed through menu access: every parameter owns a dedicated dialog box accessible from the menu item "Parameters", where the real numeric value can be filled in both writing the number in the edit field or clicking on the scroll bar placed on the right. The first method allows precise input to be performed, while the second is better suitable for quick insertion of sample values. Moreover, another dialog box is provided, accessible through the "Tune all together" menu item, whose aim is to allow the global tuning of the simulation keeping all the parameters under control.

3.3 OUTPUT SELECTION

In addition to graphical and numerical output windows, RobotSIM provides a third mean to keep track of simulation outputs, namely the *log file*, which is constructed while the simulation is run and consists of an ASCII file where the values of the output variables at each iteration are recorded. The user is allowed to specify which variables are relevant outputs to keep track of, and the number of iterations between two similar outputs. This last feature allows to fit the three kinds of output according to the actual needs: for example, it may be useful to give a low frequency for the graphical output, since it is computationally expensive, while keeping a higher frequency for the log file, in order to get a full numerical image of the simulation execution. The selection of output variables to monitor affects the file log and the numerical output window.

3.4 STOPPING STRATEGIES

Krogh's method does not specify the strategy followed to stop the simulation once the goal has been reached. The strategy intrinsic to the method is checking when the time to be spent in reaching the obstacle is zero. Nevertheless, this is not really suitable, due to the time sampling performed to execute the simulation. We did not develop yet a method able to describe goal reaching assuming time discretization; however, we provide the user with two different stopping strategies that can be selected by the menu "Simulation|Stopping Options":

- **Time based.** The previously described control is performed, but a threshold is fixed, specifying the precision degree to be assumed while performing the check. This threshold varies from simulation to simulation and possibly has to be rearranged before every execution. Since this method is quite empirical, setting the right threshold is just a matter of intuition. Further study will involve the investigation of formal conditions applicable to this problem.
- **Position based.** Selecting this option, the control is made in terms of distance checking with respect to the goal point. No information about current velocity is investigated, so you can find that the simulation stops

reporting that the robot is traveling with a somewhat high velocity. The rationale about this is that it may be desirable to perform robot motion employing more than one control method, e.g. potential field method for gross motion and another method for fine motion. Thus, in RobotSIM, the user is allowed to see simulation results to the point were the latter control system will take the control.

3.5 FILES

RobotSIM handles the following four kinds of files:

- **World description files:** used to save workspace descriptions;
- **Parameters files:** allow the storing of lists of parameters values;
- **Log files:** used to keep track of numerical values of output variables;
- **Simulation files:** to store graphical output windows, to be read by the RobotSIM Viewer.

Note that the first three kind of files are ASCII files and a precise syntax is defined for their contents, allowing the user to edit them off-line using an editor.

4. IMPLEMENTATION ISSUES

RobotSIM has been developed using Borland Turbo Pascal for Windows 1.0. The source code is about 6,500 lines long, half of this dedicated to user interface implementation. A considerable effort has been spent to build reliable and efficient algorithms for the geometrical computations involved in simulation. In this section we describe the basic algorithms.

4.1 BASE SIMULATION ALGORITHM

The base simulation algorithm is built upon Krogh's work, with the addition of more controls and refinements to improve efficiency.

Let P be the current robot position, G the goal point, ObstacleSet the set containing the obstacles in the workspace; the following is the base algorithm employed in the program:

```
Initialization
LABEL StartAlgorithm
$t \rightarrow t+\Delta t$
$P \rightarrow P+\Delta P$
IF ObstacleSet$\neq\emptyset$ THEN
   Create the list Visible$_P$ containing obstacles visible from P
IF Visible$_P\neq\emptyset$ THEN
   IF there are obstacles between P and G THEN
      Take from Visible$_P$ the obstacle O nearest to P
      Computes the vertex of O which is nearest to G
      Computes the subgoal employing the edge factor
ELSE Subgoal$\equiv$G
Computes vectors: $n_g$ and $v_g$
$v \rightarrow v+\Delta v$
Computes: distance between P and Subgoal, attraction vector $c_g$,
global repulsion vector $c_o$,  acceleration vector
IF StopCondition=FALSE THEN GOTO StartAlgorithm
ELSE END.
```

The initialization sets the temporal reference, the initial values for position and velocity, and performs the first computation of n_g and o_g vectors. The repulsion vector is the sum of all the repulsion vectors generated by each obstacle. StopCondition contains a check on the simulation termination condition.

4.2 OBSTACLES REPRESENTATION

The algorithms performing geometrical computation on the workspace needs a uniform representation of obstacles. Note that obstacles are approximated by rectangles surrounding them, and that these rectangles can have any orientation; this is a major issue, since Krogh's simulation results involved only obstacles with sides parallel to reference axes. The minimal representation for obstacles needs five parameters; the ones we chosed are shown in Fig. 4.1:

- The crosspoint of diagonals. We will call this point obstacle center. Since the environment works in two dimensions, this point represents two parameters.
- Sides dimensions.
- The angle formed by the secondary axis with the x axis of absolute reference.

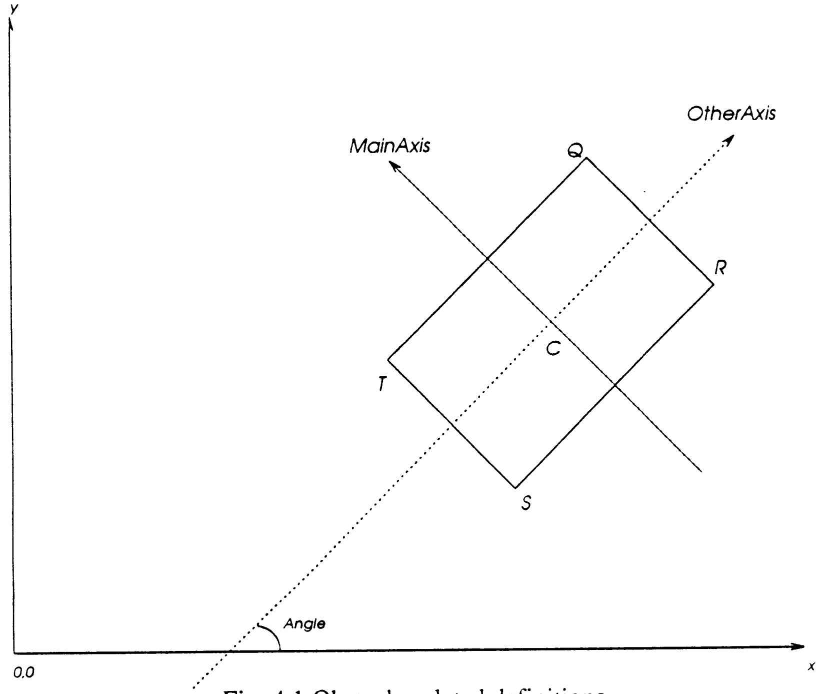

Fig. 4.1 Obstacle-related definitions.

A naming is determined for obstacle's vertexes, shown in the figure. The primary axis (MainAxis) is parallel to sides QR and TS, while secondary axis is parallel to QT and RS. This definition of vertexes is necessary to provide a unique reference for the algorithms operating on obstacles.

Note that if Angle=0, OtherAxis coincide with x axis and MainAxis with y axis. Theoretically, this representation is sufficient for any obstacle: nevertheless, due to efficiency reasons, every software object representing an obstacle carries also a four-vertex representation, which is not minimal since needs eight parameters, but is more suitable for drawing operations and transformations.

4.3 DETERMINING THE OBSTACLE NEAREST TO A POINT

The problem of determining which obstacle is nearest to a given point is central in the execution of the simulation algorithm, as emerging in the previous section. The check is performed determining the nearest point of each obstacle, then ordering consequently obstacles and determining the nearest. Thus, the problem is shifted to the search of the nearest point of an obstacle to a given point. We developed an algorithm to solve this problem, that uses multiple references, shown in Fig. 4.2:

- **xOy** is the absolute Cartesian reference, in which are defined obstacles' and special points' positions.
- **XCY** is the Cartesian reference integral with the obstacle, such that X≡OtherAxis and Y≡MainAxis.
- **X'CY'** is the reference with oblique coordinates, integral with the obstacle, within the algorithm will compute the nearest point.

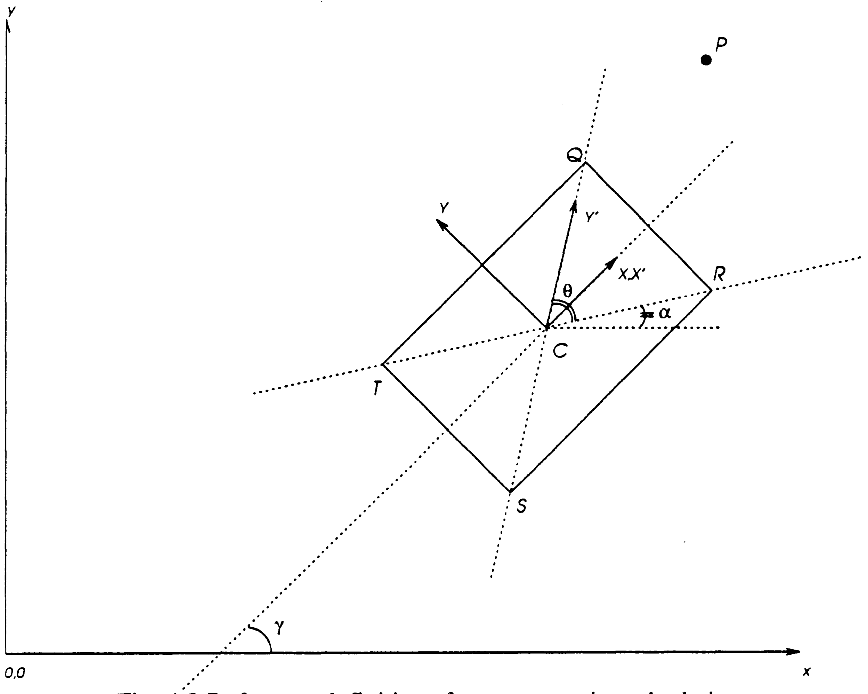

Fig. 4.2 Reference definitions for nearest point calculation.

The equation ruling trasformations is the following:

$$\begin{cases} X' = (x-a)\cos\alpha + (y-b)\sin\alpha - \cot\vartheta((y-b)\cos\alpha - (x-a)\sin\alpha \\ Y' = \dfrac{(y-b)\cos\alpha - (x-a)\sin\alpha}{\sin\vartheta} \end{cases}$$

The searched nearest point can be one of the obstacle's vertexes or can belong to one of obstacle's sides. The use of oblique coordinates allows the derivation of simpler mathematical conditions that help in determine which between these two cases is actually satisfied.

Tables 4.1 and 4.2 show the conditions we found: the first table allows to determine wheter the nearest point is also a vertex or not, while the second selects the obstacle's side where the nearest point must lie.

The algorithm can be stated as follows:

```
Trasformation of P from xOy to X'CY'
Computation of mathematical conditions in Tab. 4.1
```

```
IF P satisfies these conditions THEN Neareste{Q,R,S,T}
ELSE
   Computation of mathematical conditions in Tab. 4.2
   Store the side L=JK satisfying these conditions
   Computes N such that $N \perp L$ and $P \in N$
   $I \rightarrow L \cap N$
   IF $I \in L$ THEN $Nearest \equiv I$
   ELSE
      IF PJ<PK THEN $Nearest \rightarrow J$
      ELSE $Nearest \rightarrow K$
   END
END.
```

	X'>0	X'<0	Y'>0	Y'<0
X'=0	-	-	Q	S
Y'=0	R	T	-	-

Tab. 4.1 Conditions on vertexes.

	Y'>0	Y'<0
X'>0	QR	RS
X'<0	TQ	TS

Tab. 4.2 Conditions on sides.

4.4 DETERMINING VISIBLE OBSTACLES

As described before, at each iteration the simulation considers only in-sight obstacles: this is a major feature of the method, since it seems well suited for sensor integration, and does not require any a-priori knowledge. Visibility is determined by tracing lines connecting the point P with obstacles' vertexes and checking mathematical conditions on angles so determined within the Cartesian reference.

It can be demonstrated that if angles are normalized in the 0-360 interval and ordered increasingly, the angles corresponding to the "shadowing" vertexes are the second and the third of the list. Moreover, to an observer placed in P and watching toward the positive x, the second angle it is always placed at the left of x axis, and the third at the right.

We based our algorithm on the concept of cone of shadow. We can imagine to put a light source in P and determine which obstacles are shadowed and which lighted. The control is performed incrementally: first an object is considered, its cone of shadow is determined (current cone, denoted by cone lines SL and SR), then another obstacles is considered and its cone of shadow (new cone, denoted by cone lines NL and NR) is compared with the previous sample.

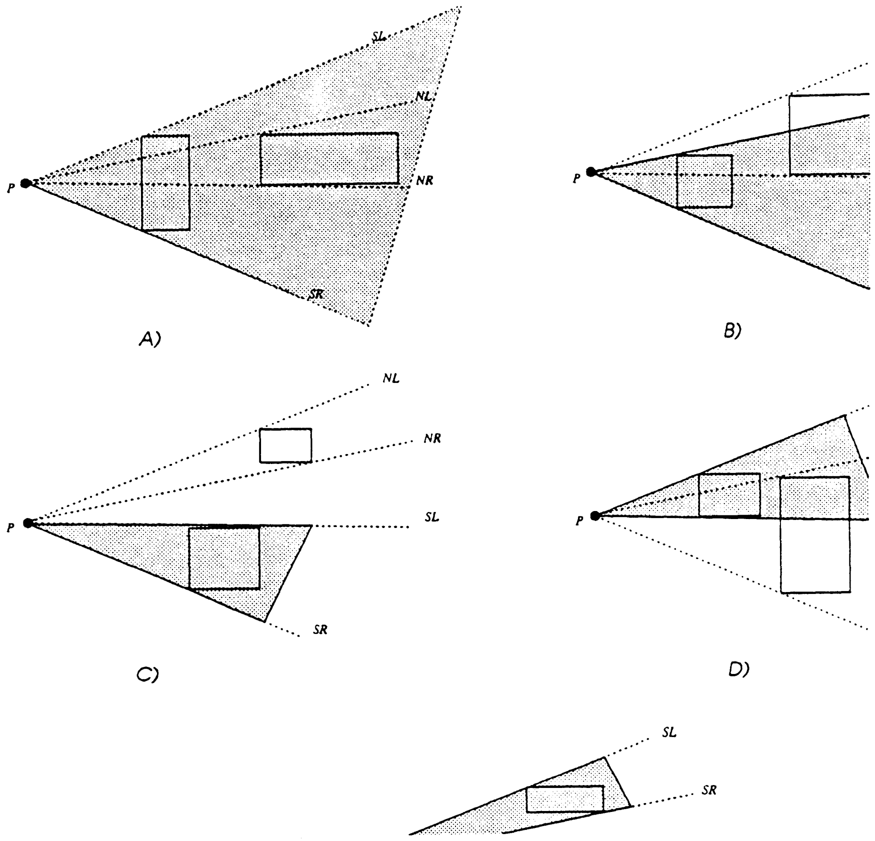

Fig. 4.3 Visibility conditions.

Some typical situations can occur, showed in Fig. 4.3:

- **Situation A).** The current cone contains completely the new cone. The new last obstacle is not visible.
- **Situation B) and D).** The current cone has intersections with the new cone, i.e. the new obstacle is partially lighted. Consequently, the current cone of shadow must be extended.
- **Situation C) and E).** The two cones have no intersection, i.e. both of obstacles create shadow.

This check on cone of shadow is repeated taking as current cone the cone of all the obstacles found creating shadow. Mathematical conditions are based on the relation "extem to", defined as follows:

If angles are normalized in 0-360 interval: line A is extern to line B to left if angle coefficient of line A within reference xPy is greater than the one of B; by converse, line A is extern to line B to right if angle coefficient of line A within reference xPy is lower than the one of B; note that this correspond to the intuitive notion of extern to, if we suppose to observe the scene in P watching towards positive x.

Four Boolean conditions can be defined:

- **Cond1.** True if SL is external to NL to left.

- **Cond2.** True if SR is external to NR to right.
- **Cond3.** True if SL is external to NR to left.
- **Cond4.** True if SR is external to NL to right.

Combining these conditions, we define the situations depicted in Fig. 4.3:

A $Cond1 \wedge Cond2$.; **B** $(\neg Cond1) \wedge Cond2 \wedge Cond3$.;

C $(\neg Cond1) \wedge Cond2 \wedge (\neg Cond3)$.; **D** $Cond1 \wedge (\neg Cond2) \wedge Cond3$.

E. $Cond1 \wedge (\neg Cond2) \wedge (\neg Cond3)$.

5. SIMULATION EXAMPLES

This section illustrates some simulation results. We have repeated the same examples reported by Krogh and found the same timing results, though we had some problem in defining coherent stopping conditions. We simple simulations to show the effects of parameters changes. If not otherwise specified, the parameter values used tare the following:

$\gamma = 0.005$; $\varepsilon = 10$; $\alpha = 1$; $t = 0.1$; $v_x, v_y = 0,0$

Fig. 5.1.a shows the simulation result for three obstacles placed between the starting point (in the lower left corner), and the goal point (in upper right corner), with the above parameters values. A slight overshoot is present, due to the intrinsic structure of repulsive and attractive field interactions. Figure 5.1.b shows the results with an increased edge factor ε=20. Since the edge factor is a sort of safety parameter, this trajectory follows a path less near to obstacles. The edge factor affects heavily trajectory shape: an edge factor ε=40 makes the free path as in Fig. 5.1.c, which is substantially different. Effects of attraction factor are illustrated in Fig. 5.1.d: here, we set γ=0.1 producing a smaller overshoot and a quicker traveling around obstacles.

Simulations in Fig. 5.2 are used to illustrate the effect of other parameters. Fig. 5.2.a shows the simulation with the standard parameter values: the overall time employed to move the robot from start to goal is about 40 seconds. Figure 5.2.b is the result of maximum acceleration allowed: with α=3 the trajectory is quicker, using about 30 seconds to perform traveling, but resulting in greater overshoot. Figure 5.2.c shows what happens with a non-zero starting velocity: here the y component of velocity has a module of 10. Note that the basic OAC problem does not consider the case of a robot starting with non-zero velocity: nevertheless, this case may be of practical interest when many controls must be integrated within the system. Moreover, this feature comes for free from the Krogh method formulation.

REFERENCES

[1] M. Athans and P. L. Falb, Optimal Control, McGraw Hill, 1966
[2] K. Fu, R.C. Gonzalez, C.S.G. Lee, Robotics: Control, Sensing, Vision and Intelligence, McGraw Hill, 1987.
[3] J. Latombe, Robot Motion Plannig, Kluwer Academic Publishers, 1991.
[4] O. Khatib, Real-Time Obstacle Avoidance For Robot Manipulator And Mobile Robots, International Journal of Robotics Research, Vol 5, N 1, 1986
[5] B. Krogh, A Generalized Potential Field Approach To Obstacle Avoidance Control, SME Conf. Proc. Robotics Research: The Next Five Years and Beyond 1984.

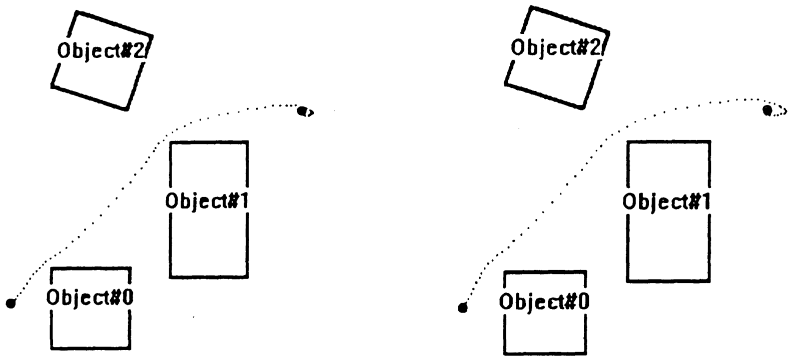

Fig. 5.1 (a,b) Simulation examples

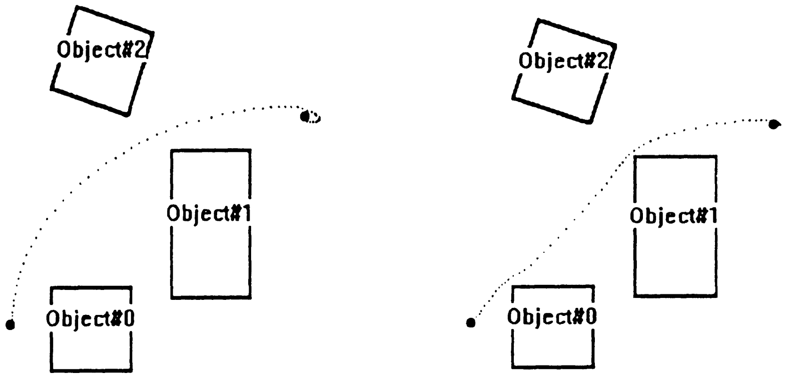

Fig. 5.1 (c,d) Simulation examples

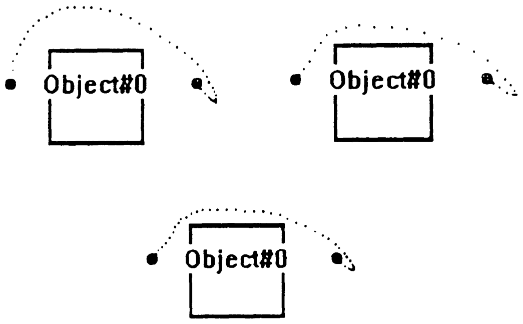

Fig. 5.2 (a,b,c). Simulation for velocity and acceleration parameters.