TA02 11:20

Proceedings of the 34th
Conference on Decision & Control
New Orleans, LA - December 1995

The Role of Time Constraints in the Design of Control for the Jogger's Problem*

Andrei M. Shkel and Vladimir J. Lumelsky
University of Wisconsin-Madison
Madison, Wisconsin 53706, USA
shkel@robios.me.wisc.edu and lumelsky@engr.wisc.edu

Abstract—**Control schemes in real-time sensor-based systems often operate under tight time constraints determined by the system sampling rate. One area where such constraints are especially severe is the sensor-based motion planning with dynamics in robotics. Though very important both theoretically and in practice, this problem has not been addressed so far. The typical sampling rates in such systems are 20 to 50 per second. This leaves only 20 to 50 msec for the whole cycle, including sensing, complex geometric (intelligence) analysis, calculations due to dynamics and control, and motion execution. As a first attempt to solve the problem, we show that the time constraints can be met by a combination of a simple model with a carefully chosen analytical solution of the dynamic equations. The resulting control scheme guarantees convergence and safety of motion, and blends well with kinematic path planning algorithms. Two such strategies are discussed, one using a simple heuristic and the other with local optimization.**

I. Introduction

Modern real-time control systems often need to combine gathering input information, its logical analysis, and calculation of the control actions within very short time intervals. This puts severe constraints on the choice of the corresponding calculation schemes.

A good example in point is the *Jogger's Problem* in robotics, which appears in the context of sensor-based motion planning [1]. Imagine a jogger in the city environment who takes his usual morning run in the neighborhood. This will involve a number of on-line control mechanisms. First, a global (convergence) mechanism is needed to assure that a roughly decided upon path will be executed, ending at the target location, in spite of the deviations and detours that the changes in the environment may require. Second, since an instantaneous stop is impossible due to inertia, in order to maintain a reasonable speed the jogger needs at any moment an "insurance" option of a safe stopping path. Third, when after turning the street corner the jogger suddenly sees a heavy large object right on the intended path, a quick local replanning mechanism is needed to take care of potential collision: his speed may temporarily decrease and the path will smoothly divert from the object. This path segment may also be locally optimized, to arrive at the street corner quicker or along a shorter path. All other options exhausted, the jogger may "brake" to a halt, and then start a detour path.

The jogger's path presents a fast sequence of steps, each of which involves sensing, logical (intelligent) decision-making, control calculations to account for body dynamics, and execution of motion. Some of those may be further intertwined within a single step. A typical sampling rate in the mobile robot's equivalent of our jogger is 20 to 50 control cycles per second. If it is, say, 40/sec, it amounts to 25 msec for the whole cycle, leaving 10-15 msec for the control proper. This includes dynamics analysis and calculation of controls, such as to guarantee the next step to be safe, reasonable (or optimal) from the standpoint of the local sensing information available, and satisfying the global path objectives. Clearly, many computation-intensive control techniques that would otherwise qualify for the task, will fail these stringent time constraints. One can argue that the only way to satisfy these constraints is to start with the simplest, and yet realistic, model and devise a very simple, and yet satisfactory, control scheme.

While accounting for body dynamics in sensor-based motion control is of serious practical importance, little attention has been paid to this connection in literature. Most of existing approaches deal solely with the system kinematics and geometry and leave out its dynamic properties. (Dynamics fares better for the model with complete information and off-line computation, see e.g., [2, 3, 4]).

A number of *kinematic* sensor-based strategies (that is, strategies that do not take into account system dynamics) originate in maze-searching algorithms [5, 6]; when

*This work was supported in part by the NOAA Sea Grant Program, US Dept of Commerce, Grant NA46RG048, and by DOE (Sandia Labs) Grant 18-4379C.

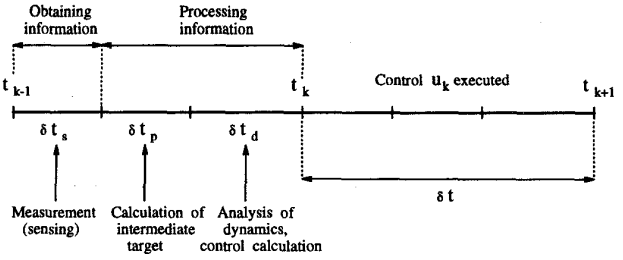

Fig. 1. Three operations – sensing, logical (intelligence) analysis, and control calculations – have to fit within a single control cycle (step of the robot) δt. Controls calculated by the end of the cycle period are applied during the next cycle.

applicable, they are usually fast, can be used for real time control, and guarantee convergence; the obstacles can be of arbitrary shapes.

To design a provably-correct *dynamic* algorithm for sensor-based motion planning, one needs a single control mechanism – separating it into stages is likely to destroy convergence. Convergence translates into two requirements – globally, a guarantee of finding a path to the target if one exists, and locally, an assurance of collision avoidance in view of the obstacles and the robot's inertia. The former can be borrowed from kinematic algorithms. The latter is the question of *controllability*: a controllable system can reach any free point if needed in spite of the dynamics and the presence of obstacles. This property requires an explicit consideration of dynamics in the control law.

In principle, any maze-searching algorithm can be utilized for the kinematic part, so long as it allows an extension to distant sensing. For the sake of specificity, here we make use of the VisBug algorithm [5]. Assume the robot is a point in the plane; let S and T be the starting and target positions, r_v – the radius of the robot's sensing range, i – the current step, M-line – the straight line (ST), see Figure 2. The VisBug procedure revolves around two steps: (1) Walk from S toward T along the M-line until, at some point C, detect an obstacle crossing the M-line, say at point H go to Step 2. (2) Using sensors, define the farthest visible intermediate target T_i on the obstacle boundary; make a step toward T_i; iterate Step 2 until detect M-line; go to Step 1.

Consider now the operations that need be done within one step, Figure 1. These are (a) sensing, (b) defining the current intermediate target, (c) control calculations due to body dynamics, and (d) execution of motion. Usually the control commands found within the current step are applied during the next step, simultaneously with the other three operations. For the corresponding times, this means $\delta t = \delta t_s + \delta t_p + \delta t_d$.

As said, the sequence (a) and (b), sensing and T_i calculation, may execute a number of times, before going to Step 2 of the algorithm. If needed, the number of such repetitions can be reduced to save time – which is equivalent to a loss in system efficiency due to inferior sensors. There is no such flexibility in the operation (c). Because of the local character of sensing, in addition to the usual computations due to dynamics the control scheme in (c) has to also include safety considerations, such as a provision for a safe stopping path, and a constraint on the maximum velocity the robot can safely maintain during its motion.

To speed up the control algorithms for real-time planning, we assume that (1) the robot is a mass point, and (2) control actions are constant within the step interval. Though simple, this model is already of practical value as long as the robot's dimensions can be ignored. This model allows us to find an analytical solution of the dynamic equations. Then, a control scheme that emphasizes a single step calculation is added, resulting in a significant speedup of the control calculations. On this latter mechanism, note that with a reasonably large radius of vision r_v, one can use a single sensing reading to calculate a relatively long – say a 10-step or 20-step long – piece of the robot's trajectory. This has its advantages – e.g., that piece of the path can be optimized. It is, however, computationally costly and largely wasteful since the sensing reading at the next step is likely to require changes in the path.

This leads to a class of fast control algorithms. The general strategy is as follows: at moment (step) i, the kinematic algorithm chosen identifies an intermediate target point, T_i, which lies on a convergent path and is far enough from the robot – normally at the boundary of the sensing range r_v. Because of the dynamics, a straight line step toward T_i may not be possible, thus requiring a smooth transition. Each step is planned as the first step of a trajectory which, given the current position, velocity, and control constraints, would bring the robot to a halt at T_i (even if the robot has no intention to stop at T_i).

We show two examples of such algorithms, one using a simple heuristics and the other relying on a local optimization scheme (many details and proofs, omitted here due to the lack of space, can be found in [1, 7]). In the first algorithm, called *Maximum Turn Strategy*, the objective is to align the direction of the robot's motion with the direction toward the intermediate target T_i as soon as possible. That is, if the angle between the current velocity vector and the direction toward T_i is larger than the maximum turn the robot can make in one step, the robot will maintain the maximum turning rate until the directions align (hence the name of the strategy).

In the second algorithm, called *Time-Optimal Strategy*,

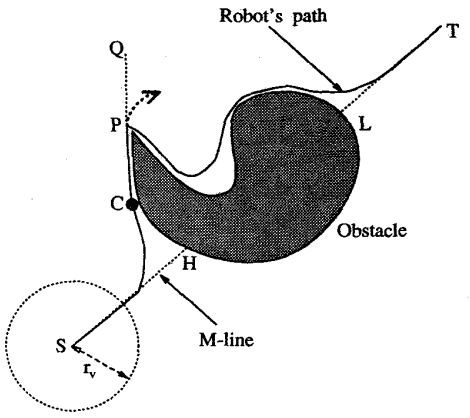

Fig. 2. Example of performance of a kinematic algorithm [5].

step planning is formulated as an optimization problem. At the current step, a *canonical solution* is found which, with no obstacles present, would bring the robot from its current position C_i to T_i with zero velocity and in minimum time. If the canonical path crosses an obstacle and thus not feasible, a *near-canonical* collision-free path segment is planned. It can be shown, first, that only a finite number of path options need be considered, and second, that the controllability is guaranteed – there always exists at least one path solution that is collision-free. Further, the two-dimensional control problem is divided into two one-dimensional control problems, each with a bang-bang control solution; this results in an extremely fast procedure, easily implementable in real time, for finding the (fairly complex) multi-step time-optimal path within the sensing range.

II. The Model

The environment (work space) is two-dimensional physical space $\mathcal{W} \subset \Re^2$; it may include a finite set of locally finite static obstacles $\mathcal{O} \in \mathcal{W}$. Each obstacle $\mathcal{O}_k \in \mathcal{O}$ is a simple closed curve of arbitrary shape and of finite length, such that a straight line will cross it in only a finite number of points. If obstacles touch each other, they are considered one obstacle.

The robot is a *point mass* of mass m. Its sensors allow it at its current location C_i to detect any obstacles and the distance to them within its *sensing range* – a disc of radius r_v ("radius of vision") centered at C_i. At moment t_i, the robot's input information includes its current velocity vector $\mathbf{V}_i$ and coordinates of C_i and target T.

Motion control is done via two components of the acceleration vector $\mathbf{u} = \frac{f}{m} = (p, q)$, where f is the force applied. Controls $\mathbf{u}$ come from a set $\mathbf{u}(\cdot) \in U$ of measurable, piecewise continuous bounded functions in $\Re^2$, $U = \{\mathbf{u}(\cdot) = (p(\cdot), q(\cdot))/p \in [-p_{max}, p_{max}],\ q \in [-q_{max}, q_{max}]\}$. By taking mass $m = 1$, we can refer to the components p and q as control forces. Force p controls the forward (or backward when braking) motion; its positive direction coincides with the robot's velocity vector $\mathbf{V}$. Force q, the steering control, is perpendicular to p forming a right pair of vectors, Figure 3. We assume no friction and no other external forces, except p and q; both can be incorporated if needed [8].

The task is to move from point S (start) in $\mathcal{W}$ to point T (target), Figure 2. The control of robot motion is done in *steps* $i = 0, 1, 2, ...$, each of time $\delta t = t_{i+1} - t_i = const$; p and q are constant within step i. The step length within time δt thus depends on the velocity $\mathbf{V}_i$; $C_0 = S$. We define three coordinate systems, Figure 3:

- fixed *world frame*, (x, y), with its origin at S;
- *primary path frame*, $(\mathbf{t}, \mathbf{n})$; the origin of this moving (inertial) frame is attached to the robot; axis $\mathbf{t}$ is aligned with the current velocity vector $\mathbf{V}$, axis $\mathbf{n}$ is normal to $\mathbf{t}$. Together with axis $\mathbf{b}$, which is a cross product $\mathbf{b} = \mathbf{t} \times \mathbf{n}$, the triple $(\mathbf{t}, \mathbf{n}, \mathbf{b})$ forms the *Frenet trihedron*, with the plane of $\mathbf{t}$ and $\mathbf{n}$ forming the *osculating plane* [9];
- *secondary path frame*, (ξ_i, η_i), is fixed during the time interval of step i. The frame's origin is at the intermediate target T_i, with axis ξ_i aligned with vector $\mathbf{V}_i$ at time t_i, and axis η_i normal to ξ_i.

III. Path Frame to World Frame Transformation

Consider the time sequence $\sigma_t = \{t_0, t_1, t_2, ...\}$ of the starting moments of steps; at moment t_i the robot is at the position C_i, with the velocity vector $\mathbf{V}_i$. Within the interval δt, based on the sensing data, intermediate target T_i (supplied by the kinematic planning algorithm), and vector $\mathbf{V}_i$, the control system will calculate controls p and q and apply them to execute step i, finishing at point C_{i+1} at moment t_{i+1}, with the velocity vector $\mathbf{V}_{i+1}$. Then the process repeats, resulting in a continuous path, $(x(t), y(t))$, as a function of time. The remainder of this section refers to the time interval $[t_i, t_{i+1})$ and its intermediate target T_i, and so index i can be dropped.

Denote $(x, y) \in \Re^2$ the robot's position in the world frame, and θ the (slope) angle between vector $\mathbf{V} = (V_x, V_y) = (\dot{x}, \dot{y})$ and $\mathbf{x}$-axis of the world frame, Figure 3. Taking mass $m = 1$, the equations of motion become

$$\ddot{x} = p\cos\theta - q\sin\theta$$
$$\ddot{y} = p\sin\theta + q\cos\theta$$

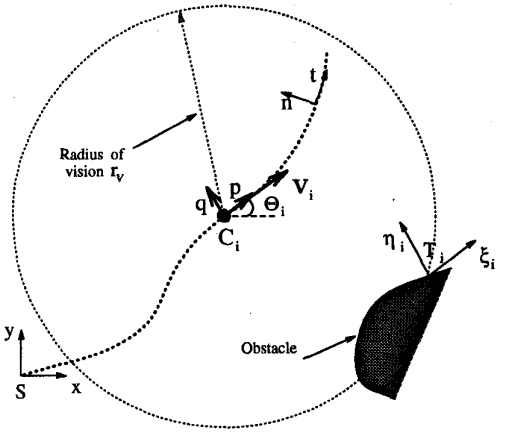

Fig. 3. The coordinate frame $(\mathbf{x}, \mathbf{y})$ is the *world frame* with its origin at S; $(\mathbf{t}, \mathbf{n})$ is the *primary path frame*, and (ξ_i, η_i) is the *secondary path frame* for the current robot's position C_i.

The angle θ between vector $\mathbf{V} = (V_x, V_y)$ and x-axis of the world frame is found as

$$\theta = \begin{cases} \arctan(\frac{V_y}{V_x}), & V_x \geq 0 \\ \arctan(\frac{V_y}{V_x}) + \pi, & V_x < 0 \end{cases}$$

The transformations between the world frame and secondary path frame, from (x, y) to (ξ, η) and from (ξ, η) to (x, y), are given by

$$\begin{pmatrix} \xi \\ \eta \end{pmatrix} = R \begin{pmatrix} x - x_T \\ y - y_T \end{pmatrix} \tag{1}$$

where

$$R = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix},$$

R is the rotation matrix between the frames (ξ, η) and $(\mathbf{x}, \mathbf{y})$, and (x_T, y_T) the coordinates of the (intermediate) target in the world frame $(\mathbf{x}, \mathbf{y})$.

To define the transformations between the world frame (x, y) and the primary path frame $(\mathbf{t}, \mathbf{n})$, write the velocity in the primary path frame as $\mathbf{V} = V\mathbf{t}$. Given that the control forces p and q act along the $\mathbf{t}$ and $\mathbf{n}$ directions, respectively, the equations of motion with respect to the primary path frame are:

$$\begin{aligned} \dot{V} &= p, \\ \dot{\theta} &= q/V \end{aligned}$$

Since p and q are constant over the time interval $t \in [t_i, t_{i+1})$, the solution for $V(t)$ and $\theta(t)$ within the interval becomes

$$\begin{aligned} V(t) &= pt + V_0, \\ \theta(t) &= \theta_0 + \frac{q \log(1 + tp/V_i)}{p} \end{aligned} \tag{2}$$

where θ_0 and V_0 are constants of integration and are equal to the values of $\theta(t_i)$ and $V(t_i)$, respectively. By parameterizing the path by the value and direction of the velocity vector, the path can be mapped into the world frame $(\mathbf{x}, \mathbf{y})$ using the vector integral equation

$$\mathbf{r}(t) = \int_{t_i}^{t_{i+1}} V \cdot \mathbf{t} \cdot dt \tag{3}$$

Here $\mathbf{r}(t) = (x(t), y(t))$, and $\mathbf{t}$ is a unit vector of direction $\mathbf{V}$, with the projections $\mathbf{t} = (\cos(\theta), \sin(\theta))$ onto the world frame $(\mathbf{x}, \mathbf{y})$. After integrating equation (3), obtain the set of solutions,

$$\begin{aligned} x(t) &= \frac{2p\cos\theta(t) + q\sin\theta(t)}{4p^2 + q^2} V^2(t) + A \\ y(t) &= -\frac{q\cos\theta(t) - 2p\sin\theta(t)}{4p^2 + q^2} V^2(t) + B \end{aligned} \tag{4}$$

where $V(t)$ and $\theta(t)$ are defined by controls p and q in 2), and terms A and B are

$$\begin{aligned} A &= x_0 - \frac{{V_0}^2 \, (2\,p\,\cos(\theta_0) + q\,\sin(\theta_0))}{4\,p^2 + q^2}, \\ B &= y_0 + \frac{{V_0}^2 \, (q\,\cos(\theta_0) - 2\,p\,\sin(\theta_0))}{4\,p^2 + q^2} \end{aligned}$$

Equations (4) describe a spiral curve. Note two special cases: when $p \neq 0$ and $q = 0$, (4) describe a straight line motion under the force along the vector of velocity; when $p = 0, q \neq 0$, the force acts perpendicular to the vector of velocity, and (4) produce a circle of radius $V_0^2/|q|$ centered at the point (A, B).

IV. Algorithmic issues

Safety considerations. Due to lack of information about the obstacles beyond distance r_v from the robot, for collision-free motion at least one "last resort" *stopping path* should be assured at any moment. Under continuous control and straight line motion, the following relationship ties the maximum velocity $\mathbf{V}$, mass m, radius r_v, and controls $\mathbf{u} = (p, q)$: the velocity must not exceed

$$V \leq \sqrt{2 p_{max} r_v} \tag{5}$$

However, after accounting for the discrete nature of our control, the maximum velocity can be shown to decrease to

$$V_{max} = \sqrt{p_{max}^2 \delta t^2 + 2 p_{max} r_v} - p_{max} \delta t$$

These equations show how changes in the system parameters – e.g. an increase in r_v due to better sensing, or an increase in p, q due to more powerful motors – will effect

the maximum velocity that the robot can afford without endangering safety.

Convergence. Because of the effect of dynamics, the convergence mechanism borrowed from a kinematic algorithm – say VisBug – needs some modification. The intermediate target points T_i produced by VisBug lie either on the boundaries of obstacles or on the M-line, and are visible from the corresponding robot's positions. However, the robot's inertia may cause it to move so that T_i will become invisible, either because it goes outside the sensing range r_v (as after point P, Figure 2), or due to occluding obstacles. The solution chosen is to keep the velocity high and, if point T_i does go out of sight, modify the motion locally until T_i is spotted again.

Step planning. The last step in the outlined methodology is to choose the step planning procedure proper. This part is what distinguishes between different control strategies. Two examples of such procedures are sketched below.

V. The Control Algorithms

The Maximum Turn Strategy. It includes three procedures (see details in [7]): (1) the *Main body* analyzes the path towards the intermediate target T_i; (2) *Define Next Step* chooses the forces p and q; (3) *Find Lost Target* deals with the case when T_i goes out of the robot's sight. Also used is a procedure called *Compute* T_i, from the VisBug algorithm [5], for computing the next intermediate target T_{i+1} and for analyzing the target reachability. Vector $\mathbf{V}_i$ is the current vector of velocity, T is the robot's target. For brevity, we sketch the *Main body* only.

Main Body: The procedure is executed at each step, and includes two steps:

- Step 1: Move in the direction specified by *Define Next Step*, while executing *Compute* T_i. If T_i is visible do: if $C_i = T$ the procedure stops; else if T is unreachable the procedure stops; else if $C_i = T_i$ go to Step 2. Otherwise, use *Find Lost Target* to make T_i visible. Iterate Step 1.
- Step 2: Make a step along vector $\mathbf{V}_i$ while executing *Compute* T_i: if $C_i = T$ the procedure stops; else if the target is unreachable the procedure stops; else if $C_i \neq T_i$ go to Step 1.

The Time-Optimal Algorithm. As mentioned above, at each step of this procedure a *canonical solution* is found which, with no obstacles present, would bring the robot from its current position C_i to the current intermediate target T_i with zero velocity and in minimum time. If the canonical path is not feasible because it crosses an obstacle, a *near-canonical* collision-free path segment is planned.

The algorithm is executed continuously, at each step of the path, and includes three procedures (see details in [1]. (1) The *Main Body* procedure monitors the general control of motion towards the intermediate target T_i. In turn, *Main Body* makes use of three procedures: (2) *Define Next Step* chooses controls (p, q) for the next step. (3) *Find Lost Target* deals with the special case when the intermediate target T_i goes out of the robot's sight. In turn, *Main Body* also uses the procedure *Compute* T_i [5], which computes the next intermediate target T_{i+1} and performs the test for target reachability. Initially, $i = 0, C_i = S$. We now sketch the *Main Body* procedure:

Procedure *Main Body:* At each step i:

- Step 1: If $C_i = T$, stop.
- Step 2: Find T_i from *Compute* T_i. If T is found unreachable, stop. If T_i is visible, find C_{i+1} from *Define Next Step*; make a step towards C_{i+1}; iterate; Else, use *Find Lost Target* to make T_i visible; iterate.

The convergence of the algorithm is guaranteed by the design of the canonical and near-canonical solutions, and also by the convergence properties of the kinematic algorithm [5]. Its computational complexity is quite low, thanks to the use of a moving reference frame and to limiting the explicit computation to a single step constant time calculation in each control cycle. That is, although the canonical solution represents the whole multi-step trajectory within the sensing range of radius r_v, its computation time is independent of that trajectory and of r_v.

VI. Examples

We show a set of simulated examples for the Time-Optimal strategy only, Figure 4; other examples can be found in [7, 1]. The generated paths are shown in thicker lines. For comparison, also shown in a thin line are the corresponding paths produced under the same conditions by the kinematic algorithm VisBug [5].

The examples demonstrate the effect of the radius of sensing r_v and the effect of additional obstacles. Note that in Figure 4b the path becomes tighter, shorter, though it takes longer compared to (a): measured in the number of steps, the path in (a) takes 242 steps, and in (b) 278 steps. One might say the robot becomes more cautious in (b). A similar pair of examples, shown in Figures 4c,d illustrates the effect of the radius of vision: in (c) and (d) r_v is twice that of (a) and (b). The times to execute the path are 214, and 244 steps, respectively, shorter than in the corresponding examples (a), (b). That is, better sensing (larger r_v) results in these examples in shorter time to complete the task; more crowded space resulted in longer time (though perhaps in shorter paths).

Note that from time to time the system found it necessary to make use of the stopping path – those points are easy to spot by the sharp turns in the path (those would be impossible with a nonzero velocity). For example, in Figure 4a the robot had to stop only twice, at points A,B; in the more crowded scene of Figure 4 it had to stop six times, at points A,B,C,D,E,F.

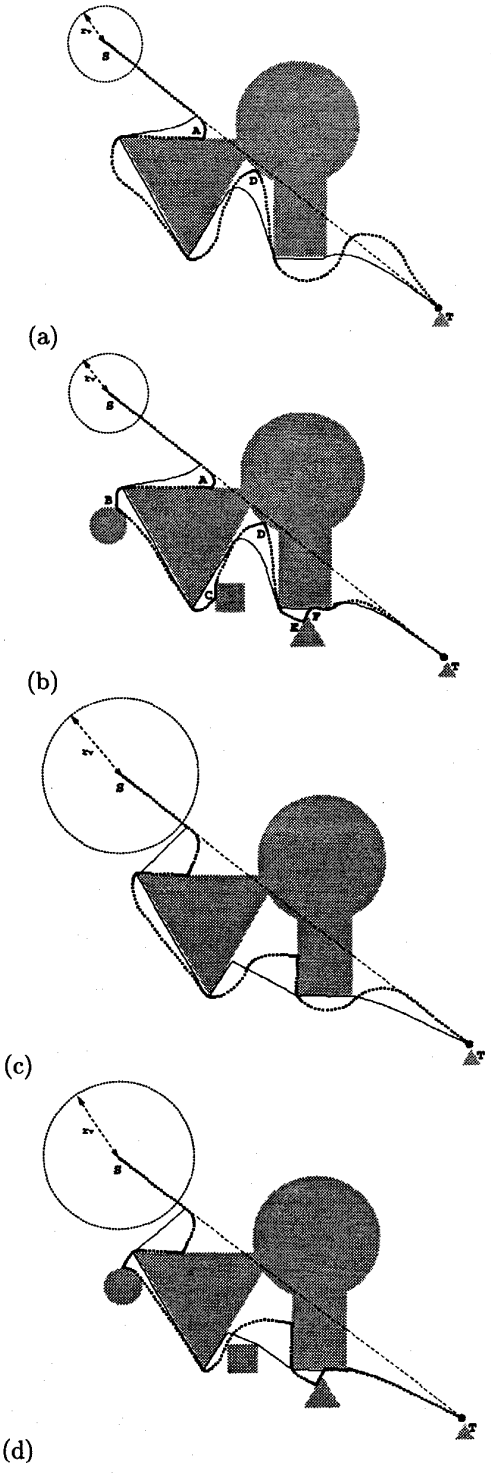

Fig. 4. Example of performance of the Time-Optimal Algorithm.

References

[1] A. Shkel and V. Lumelsky. The Jogger's Problem: Accounting for body dynamics in real-time motion planning. *IEEE/RSJ Intern. Workshop On Intelligent Robots and Systems*, August 1995. Pittsburgh, PA.

[2] B. Donald and P. Xavier. A provably good approximation algorithm for optimal-time trajectory planning. *Proc. IEEE Intern. Conf. on Robotics and Automation*, May 1989. Scottsdale, AZ.

[3] Z. Shiller and S. Dubowsky. On computing the global time optimal motions of robotic manipulators in the presence of obstacles. *IEEE Trans. on Robotics and Automation*, 7(6):785–797, 1991.

[4] J. Canny, A. Rege, and J. Reif. An exact algorithm for kinodynamic planning in the plane. *Proc. 6th Annual Symposium on Computational Geometry*, June 1990. Berkeley, CA.

[5] V. Lumelsky and T. Skewis. Incorporating range sensing in the robot navigation function. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(5):1058–1069, September 1990.

[6] A. Sankaranarayanan and M. Vidyasagar. Path planning for moving a point object amidst unknown obstacles in a plane: A new algorithm and a general theory for algorithm development. *Proc. 29th IEEE Intern. Conf. on Decision and Control*, 1990. Honolulu, HI.

[7] V. Lumelsky and A. Shkel. Incorporating body dynamics into the sensor-based motion planning paradigm. The maximum turn stratedy. *Proc. IEEE Intern. Conf. on Robotics and Automation*, May 1995. Nagoya, Japan.

[8] T. Fraichard and A.Scheuer. Car-like robots and moving obstacles. *Proc. IEEE Intern. Conf. on Robotics and Automation*, May 1994. San Diego,CA.

[9] G. Korn and T. Korn. *"Mathematical Handbook"*. McGraw-Hill, New York, 1968.