Deterministic Kinodynamic Planning with Hardware Demonstrations

François Gaillard[1,2] Michaël Soulignac[1] Cédric Dinont[1] Philippe Mathieu[2]

[1] ISEN Lille, CS Dept.
41, Boulevard Vauban
59046 Lille Cedex, France
{firstname.lastname}@isen.fr

[2] LIFL
University of Lille 1
59655 Villeneuve d'Ascq Cedex, France
UMR USTL/CNRS 8022
{firstname.lastname}@lifl.fr

***Abstract*— DKP (Deterministic Kinodynamic Planning) is a bottom-up trajectory planner for robots with flatness properties. DKP builds an exploration tree of which the branches are spline trajectories. DKP employs an A^*-like algorithm to select which branch of the tree to grow. The selected trajectories are then grown in a propagation process which respects the kinematic constraints, such as linear/angular speed limits or obstacle avoidance. In addition, DKP produces trajectories that are immediately executable by the robot. Various experiments are provided to show the ability of DKP to effectively handle complex environments with one or more robots.**

I. Introduction

Computing a trajectory that takes into account both kinematic and dynamic constraints is known as kinodynamic planning [5], and is proven to be PSPACE-hard [22].

Decoupled approaches separate kinodynamic planning in two successive problems; first, compute a path taking into account a part of the problem constraints (classically obstacles) and, then, smooth this path with the remaining constraints to make the solution admissible by the robot. The efficiency of decoupled approaches, such as variants of Elastic Bands [23], is explained by the fact that they are generally customized for specific kinodynamic problems [15]. They also provide bounds on the computation time, allowing on-line planning, which explains their wide usage. However, decoupled approaches present difficulties to solve complicated problems, with many degrees of freedom. Moreover, they suffer from incompleteness issues: since the initial path is not guaranteed to be feasible by the robot, the path smoothing phase may fail to satisfy all kinodynamic constraints or fail to find a solution even if one exists.

To solve these difficulties [18], we can distinguish two categories of **hybrid approaches** which incorporate a local motion planner (selection process) within a global path planner (propagation process) to ensure the respect of constraints. The first category contains heavily customized approaches: both local and global motion planner are then designed to generate complex local maneuvers [11] and/or improve the quality of the global path to be tracked. They successfully deal with very specific problems, such as [6] which integrates perception sensors in the local planner; or the multi resolution approaches like AD* [18].

This work is supported by the Lille Catholic University, as part of a project in the Handicap, Dependence and Citizenship pole.

The second category contains approaches such as RRT [17], EST [12], DSLx [20] or PDST-EXPLORE [14]. They use randomized techniques to integrate controls and explore the local reachable space. This propagation process makes them well scalable for robots with high degrees of freedom and/or complicated system dynamics and avoids the drawbacks of the previous techniques. Nevertheless, their results are by nature unpredictable, in terms of computation time and solution quality. Recent works focuses then on the solution optimality (RRT* [13]) and the use of sampled-based approach in both discrete and continuous hybrid state spaces [1].

Our hybrid approach **DKP**, first introduced in [9], proposes an intermediate solution between these two categories. DKP builds in a 2-D space an exploration tree of which the branches are spline trajectories guided by an optimization criterion. DKP is designed for robots which accept spline trajectories as solution when their model satisfies flatness properties [8] with two degrees of freedom. Contrary to [19], DKP does not rely on a non linear quadratic solver which hides some random processes in order to explore the state space. In DKP, robots are specified with kinematic constraints, such as linear/angular speed limits or obstacle avoidance, represented in a geometrical approach. The propagation process then defines a locally, continuous and complete reachable space, called *parameter space*, which models all the possible subtrajectories while satisfying all constraints of the problem. We might also design specific constraints for a particular robot which restricts this *parameter space*. With a deterministic process, DKP then produces various locally optimal subtrajectories with respect of kinematic constraints and diverse time durations. This propagation process provides an expansive

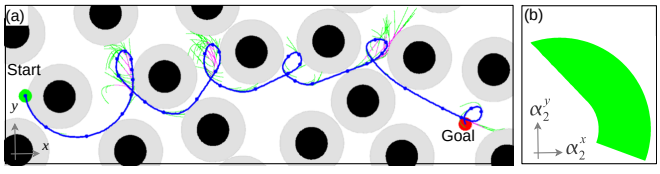

Fig. 1. (a) Trajectory computed by DKP (in blue) for a robot which can only turn left. (b) *parameter space* associated to the goal point (see Section III-A for details).

Approach	**Selection process**	**Propagation process**	**Diversity source**
planner from [11]	none	degree 5 polynomials generated by non-linear programming	lookup table
AD*	inflated A* searches	combination of actions	planner from [11]
RRT	random configuration	random integration	commands
EST	random milestones	random integration	commands
DSLx	random leads	random integration	commands
PDST	deterministic branch	random integration	commands + time
RRT*	random configuration + cost-based "rewire"	random integration	commands
DKP	A* search	quadratics generated by deterministic optimization over *parameter space*	trajectory time duration

TABLE I

PROPERTIES OF SOME HYBRID APPROACHES

exploration of the local search space and prevents DKP from using random processes to avoid from local minima.

Unlike [11] or random approaches, even if subtrajectories are simple, DKP takes advantage of this simplicity to control the behavior of the DKP local planner by separating the reachable space computation from the solution search. Moreover, this simplicity, coupled to an efficient global planner, is enough to produce complex maneuvers, as illustrated in Figure 1.

DKP combines some of the advantages of existing hybrid approaches, such as the exploration tree of random approaches, with guarantees on the feasibility of the trajectory, the quality of the solution, the reproducibility of the results and the control of the computation time. Moreover, the parameter space is a powerful way to design specific constraints for our robots and clearly identify the reachable space. A summary of the differences with previous techniques is provided in Table I.

This paper proposes an enhanced and more efficient definition of DKP, which is not constrainted to disk shaped constraint. We also introduce a backtracking mode which exploits the parameter space properties to enhance the exploration process. This paper provides then several hardware experiments to highlight the description of different two-wheeled robots in three situations. Our geometrical approach to constraints makes these various descriptions easily solvable in DKP. In each described case, DKP precisely renders the abilities of the robots. A simulation study provides key results about overall performances.

II. DETERMINISTIC KINODYNAMIC PLANNING

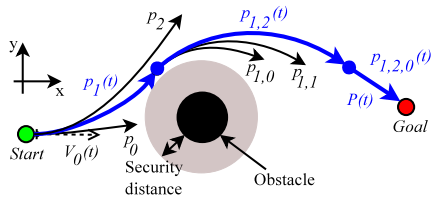

Fig. 2. DKP builds spline trajectories from *Start* to *Goal* with subtrajectories $p_{k_0 \ldots k_n}(t)$ in an exploration tree guided by an optimization criterion subject to kinematic constraints. The security distance corresponds to robot clearance.

To find trajectories from a *Start* state to a *Goal* state, DKP builds an exploration tree of which the branches are subtrajectories, lying into the reachable regions of a 2-D continuous space. The exploration tree expands in a fully known environment (static and mobile obstacles). Let $p_{k_0 \ldots k_n}(t)$ denote a subtrajectory in the exploration tree and $P_{k_0 \ldots k_n}(t)$ the overall trajectory which contains the subtrajectories $p_{k_0}(t), p_{k_0 k_1}(t), ..., p_{k_0 \ldots k_n}(t)$. Let $T_{k_0 \ldots k_n}$ be the time horizon of subtrajectories (we should select it as a subdivision of an estimated time needed to reach the *Goal* state). To choose which branch to grow, the selection process is guided by an application-dependent optimization criterion ρ in an A* manner. The selected subtrajectories are then pursued by the propagation process which builds subtrajectories subject to continuity and kinematic constraints. In DKP, we deal with constraints by setting bounds on linear/angular speed and linear acceleration. We describe static/mobile obstacle avoidance by using their semialgebraical shape. We are able to enforce the robot motion in a moving or static area using semialgebraical shapes. Other types of constraints could be defined. From the end of selected subtrajectory $p_{k_0 \ldots k_n}(t)$, the propagation process produces a set of new subtrajectories $p_{k_0 \ldots k_n K_{n+1}}(t)$ with various time durations and adds them to the exploration tree. Contrary to [19], we do not impose the number of knots in our spline trajectories in the problem definition (how could we choose this number without solving the problem itself ?). The overall process is illustrated by the Figure 2.

III. PROPAGATION PROCESS

A. The parameter space

The DKP propagation process creates new subtrajectories from a selected branch, taking into account the kinodynamic constraints. DKP first finds all the admissible subtrajectories, represented by a *parameter space* denoted E. We consider quadratic subtrajectories $p_{k_0 \ldots k_n}(t) = (x_{k_0 \ldots k_n}(t) = \alpha_0^x + \alpha_1^x t + \alpha_2^x t^2, y_{k_0 \ldots k_n}(t) = \alpha_0^y + \alpha_1^y t + \alpha_2^y t^2)$ in the exploration tree, with $t \in [0, T_{k_0 \ldots k_n}]$. The continuity through initial position and speed with the previous subtrajectory $p_{k_0 \ldots k_n}(t)$ sets the lesser degree parameters $\alpha_0^x, \alpha_1^x, \alpha_0^y$ and α_1^y of $p_{k_0 \ldots k_n k_{n+1}}(t)$. The remaining (α_2^x, α_2^y) parameters of $p_{k_0 \ldots k_n k_{n+1}}(t)$ (mesured in m/s^2) set the quadratic shape.

So, the *parameter space* E denotes all the allowable values of (α_2^x, α_2^y) [16] which define subtrajectories satisfying the constraints.

B. The kinodynamic constraints

We define a constraint c by its constraint function $f_c(p_{k_0 \dots k_n}(t), t)$, bounded in $[D_-, D_+]$ $(D_-, D_+ \in \mathbb{R})$, applied on a quadratic at time step t. The kinodynamic constraints are implemented in DKP by their geometrical representation (using semialgebraic shapes), denoted $G_c(t)$, in their respective basis:

- the linear acceleration $c_{Acceleration}(t)$, within the domain $[A_-, A_+]$, with constraint function: $A(t) = \sqrt{\ddot{x}(t)^2 + \ddot{y}(t)^2}$, defines an annulus of radii A_-,A_+ in the basis $(\ddot{x}; \ddot{y})$;
- the linear speed $c_{Speed}(t)$, within the domain $[S_-, S_+]$, with constraint function: $S(t) = \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2}$, defines an annulus of radii S_-,S_+ in the basis $(\dot{x}; \dot{y})$;
- a forbidden area in the real plane in the basis $(x; y)$ applies readily to static obstacles. Because our constraints are time-defined, a mobile obstacle avoidance only differs by the need to translate its shape along its trajectory. Shapes are grown to model the robot clearance.
- forcing the motion to lie in an allowed area of the environment almost shares the definition of the obstacle avoidance.

As we want to work on the allowable shapes of $p_{k_0 \dots k_n k_{n+1}}(t)$, we define affine transformations matrices $M_c(t)$ from the constraint c basis to the *parameter space* basis. Finally, a constraint on a quadratic $p_{k_0 \dots k_n}(t)$, projected in the parameter basis by $M_c(t) \times G_c(t)$, is represented by a geometrical shape which restricts the parameter values (α_2^x, α_2^y), hence the allowed shapes of $p_{k_0 \dots k_n}(t)$. Other constraints on quadratic could be defined whenever we can describe its shape in one of the previous bases. An example of a constraint which forbids the robot to turn in one direction is illustrated by the Figure 1(a).

C. Parameter space building process

The propagation process defines a locally reachable space, called *parameter space*, which models all the possible subtrajectories satisfying all constraints of the problem. Let C be a set of such constraints as described in the previous section and $T_{k_0 \dots k_n}$ the time horizon of the subtrajectory $p_{k_0 \dots k_n}$ to be created. $E(T)$ denotes the *parameter space* for every time step, noted $step$, with $t \in [0, T_{k_0 \dots k_n}]$, such as: $E(T) = \{\bigcap_{t=0}^{T} M_c(t) \times G_c(t) \mid c \in C, t \bmod step \equiv 0\}$. Every point (α_2^x, α_2^y) chosen in $E(T_{k_0 \dots k_n})$ defines a quadratic $p_{k_0 \dots k_n}(t)$ of duration $T_{k_0 \dots k_n}$ which satisfies the kinodynamic constraints at every $t \in \{0, step, ..., T_{k_0 \dots k_n}\}$. Our current approach is limited for robots with two degrees of freedom and flat outputs, so the parameter space is a 2-D space. We should extends this problem to cubics (or higher degrees of freedom) instead of quadratics but the parameters space building complexity would dramatically grow: our design choise is to keep the simpler subtrajectories and to lie on the selection process to achieve complex maneuvers.

D. Example

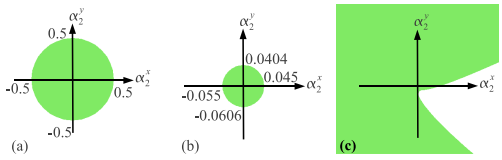

Fig. 3. The resulting *parameter spaces* which model all valid parameter values (α_2^x, α_2^y) (in m/s^2) for (a) a linear acceleration constraint, (b) a linear speed constraint and (c) an obstacle avoidance constraint, after considering them for every $step = 0.1s$ up to time horizon $T = 10s$.

Consider the following situation of a robot with:

- initial position $p(0) = (0, 0)m$ and speed vector $Sv(0) = (0.1, 0.2)m/s$;
- a security distance $r_{robot} = 0.5m$;
- the following constraints, considered every $step = 0.1s$ up to time horizon $T = 10s$:
  - linear acceleration bounded between 0 and $1m/s^2$;
  - linear speed bounded between 0 and $1m/s$;
  - disk shaped static obstacle $Obs = (2, 0)$.

Figure 3 shows the *parameter space* for each constraint. Figure 4 shows (a) their intersection and (b) how the choice of a point in (α_2^x, α_2^y) sets the shape of a subtrajectory satisfying the constraints.

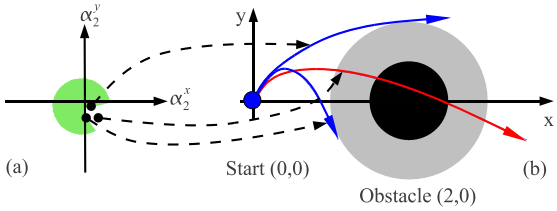

Fig. 4. The green area in (a) represents the intersection of the previous constraint representations: it models all valid parameter values (α_2^x, α_2^y) eligible for each constraint. Figure (b) shows the impact of (α_2^x, α_2^y) on the shape of the quadatric subtrajectory. Choosing parameters (α_2^x, α_2^y) outside the constrained *parameter space* leads to a constraint violation, here obstacle avoidance.

E. Solutions over parameter space

The propagation process exploits the *parameter space* to identify the locally optimal subtrajectory. Let the $Goal$ be in (x_g, y_g). The function ρ_{loc} to be minimized is the distance of the subtrajectories endpoint $p_{k_0 \dots k_n}(T_{k_0 \dots k_n})$ to $Goal$. We can analytically find the exact values needed to reach the $Goal$: $\alpha_2^x = (x_g - \alpha_0^x - \alpha_1^x T_{k_0 \dots k_n})/{T_{k_0 \dots k_n}}^2$ and $\alpha_2^y = (y_g - \alpha_0^y - \alpha_1^y T_{k_0 \dots k_n})/{T_{k_0 \dots k_n}}^2$. If this point belongs to $E(T_{k_0 \dots k_n})$, then it is valid for all motion constraints and these values set the shape of a subtrajectory $p_{k_0 \dots k_n}(t)$ which reaches $Goal$ at $t = T_{k_0 \dots k_n}$. Otherwise, the best solution for E is on the boundary of this space. We use QuadTree [7] to get a precise tiling with rectangles over the border of E. We then use a rough optimization which consists

in choosing the best point among remarkable points in the considered rectangle (corners and center). Each of the points from $E(T_{k_0 \dots k_n})$ defines a subtrajectory on which ρ_{loc} is applied. The best solution is found from all the rectangles of the tiling: this subtrajectory has the closest end point to the goal.

F. Diverse time durations for exploration

We remark that $E(T) = \{\bigcap_{t=0}^{T} M_c(t) \times G_c(t)\} = \{\bigcap_{t=0}^{T'} M_c(t) \times G_c(t)\} \cap \{\bigcap_{t=T'}^{T} M_c(t) \times G_c(t)\} \subset E'(T')$ with $T' < T$. This means that the *parameter space* E computed for a duration T involves the computation of the *parameter space* E' with lower time durations T'. As a consequence, for given step, we can create more subtrajectories with lower time horizons T' using the intermediate *parameter space*s $E'(T')$ when computing the *parameter space* $E(T)$, as shown by Figure 5.

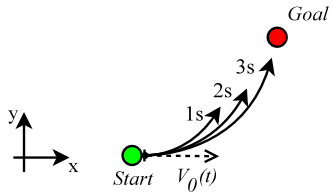

Fig. 5. Deterministic propagation of subtrajectories with different time durations and respect of kinodynamic constraints.

IV. Selection process

A. Exploration tree in a continuous space

DKP explores the environment with an exploration tree guided by an optimization criterion in an A* manner. As the A* algorithm, the scores that DKP propagates to quadratic subtrajectories are the result of an evaluation function $\rho = g + bias \times h$. g is the cumulated real cost and h is the heuristic part. More commonly, we use length of the root to the current branch as real cost g and euclidian distance from the end of the evaluated branch to the goal as the heuristic part h. $bias$ modifies the heuristic and the behavior of the selection process.

Let T_{min} and T_{max} be the minimum and maximum allowed time horizons of subtrajectories. When a subtrajectory $p_{k_0 \dots k_n}$ is selected, DKP creates new subtrajectories $p_{k_0 \dots K_n}(t)$ with various time durations, based on a $step_{variety}$ such that $T_{min} \leq T_{k_0 \dots k_n k_{n+1}} = k_{n+1} \times step_{variety} \leq T_{max}$. In contrast to [2] which needs to invalidate some generated controls, the *parameter space* guarantees that all subtrajectories generated by our propagation process satisfy all the constraints, including obstacle avoidance.

Unlike usual A* path planners, the subtrajectories used in DKP lies on a continuous space and two subtrajectories rarely coincide. Subtrajectories are filtered with discretization criteria upon subtrajectory endpoint, speed vector direction, speed vector norm and subtrajectory length. We are thus able to control the number of subtrajectories created by DKP and, consequently, the computation time of the algorithm can be bounded.

B. A spline as trajectory solution

The final trajectory $P(t) = P_{k_0 \dots k_N}(t)$ is built by retrieving the predecessors of the last subtrajectory $p_{k_0 \dots k_N}(t)$ which satisfies the stopping criterion. Consequently, the trajectory is made up of N subtrajectories. The total duration is $T_{end} = \sum_{i=0}^{N} T_{k_0 \dots k_i}$. Let t be in $[0; T_0] \cup \dots \cup [T_{k_0 \dots k_{n-1}}; T_{k_0 \dots k_n}] \cup \dots \cup [T_{k_0 \dots k_{N-1}}; T_{k_0 \dots k_N}]$. If t is in $[T_{k_0 \dots k_{n-1}}; T_{k_0 \dots k_n}]$, the value of the trajectory $P(t)$ is the value of subtrajectory $p_{k_0 \dots k_n}$ such as $P(t) = p_{k_0 \dots k_n}(t - T_{k_0 \dots k_{n-1}})$.

The DKP global planner can build complex maneuvers even with quadratics selected in the *parameter space* from an exploration tree built in an A* manner. The solution is admissible for robots with flatness properties in the condition they do not suffer from the acceleration discontinuity between branches.

C. Robot control

The robot controls are obtained from the trajectory $P(t) = P_{k_0 \dots k_N}(t)$ by applying the following commands: $s(t) = \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2}$ and $\omega(t) = \frac{\ddot{y}(t) \times \dot{x}(t) - \ddot{x}(t) \times \dot{y}(t)}{\dot{x}(t)^2 + \dot{y}(t)^2)^{3/2}}$. We then use a saturated controller [3] which solves the tracking problem in the presence of input saturations and unknown disturbances.

D. DKP exploration modes

We can naturally use DKP exploration in the following two modes: optimal mode with $bias = 1$ or greedy mode with $bias >> 1$. Optimal and greedy modes are respectively used in the illustrative examples of Sections V-A and V-B.

As a third mode, the DKP selection process may be enhanced by a backtracking process that adds virtual obstacles in order to handle local minima. When a subtrajectory $p_{k_0 \dots k_{n+1}}$ cannot be pursued, *i.e.* its *parameter space* E is empty, $p_{k_0 \dots k_{n+1}}$ is removed from the exploration tree. The previous subtrajectory $p_{k_0 \dots k_n}$ is selected and this adaptation process locally modifies the environment representation by adding some virtual obstacles in order to change the reachable parts. A new propagation process is then done on the subtrajectory $p_{k_0 \dots k_n}$. When the trajectory $p_{k_0 \dots k_n}$ backtracking cases reaches a $trigger$ value, this subtrajectory is also removed, the previous subtrajectory $p_{k_0 \dots k_{n-1}}$ is selected, a bigger virtual obstacle is added, new propagation is done with it and so on. With this adaptation process, DKP will try to pursue a branch by locally modifying its best solutions. This backtracking mode is used in the illustrative example of Section V-C.

DKP could be used in real world applications with limited information and dynamic environment: in such cases, we are aware that DKP is hard to tune and that DKP should be customized with better selection processes, for example D^* [24] or AD^* [18], and better guidance. Our future works will focus on an intelligent online planning with a common adaptation of the exploration process (the backtracking mode being an example) and the iterative planning.

V. Illustrative examples

The following examples demonstrate the ability of DKP to take into account various kinodynamic constraints, static and moving obstacles. They also illustrate that trajectories provided by DKP are directly executable on real robots, without any smoothing phase. Our solution is deployed on Mindstorm robots using our platform APM(Robot) [4].

APM(Robot) provides generic communication processes between modules and/or robots for implementation, testing and deployment of our experiments, such as Robot Operating System [21] or Player/Stage [10].

The robots localize themselves by integrating odometer data, and continuously send localization messages to a computer. The computer generates commands using the trajectory tracking algorithm described in [3], and sends them back to the robots.

Figures - depict top views of the solutions planned by DKP and their real executions, illustrating the state of the robots every 100ms. In the planned subtrajectories, real obstacles are drawn in black and grown obstacles (tacking into account the robots clearance) in light gray. The branches of the exploration tree are shown in green ($open$ nodes in the A^* sense) and magenta ($closed$ nodes).

A. Medium and low-acceleration robots

This example illustrates the benefits of DKP over classical grid-based approaches, such as A* and variants (D*, E*), providing trajectories which are not necessarily executable by the robot.

1) Description: DKP has been run in optimal mode ($bias = 1$) with the following parameters: maximal time horizon $T_{max} = 6s$, time step for constraint evaluation $step = 0.5s$; robots characteristics: velocity bounds: $S_- = 9cm/s$, $S_+ = 18cm/s$, acceleration bounds: $A_- = 0cm/s^2$, $A_+ = 3cm/s^2$ for the low-acceleration robot (Fig. 6a) and $A_+ = 7cm/s^2$ for the medium-acceleration robot (Fig. 6c).

2) Results: The trajectory provided by DKP is similar to the one provided by a grid-based approach (here A*) for a robot with medium acceleration capabilities, but the two trajectories can significantly differ for low acceleration capabilities. Trajectories provided by DKP seems longer, but the corresponding overcosts are necessary to respect the robot's kinodynamic constraints.

Counter-intuitively, using such grid-based trajectories as leads (like in DSLx) does not necessarily speed up the search, and worse, could lead to incompleteness issues (since the dotted line trajectory is not guaranteed to be executable by the robot, like in Figure 6c).

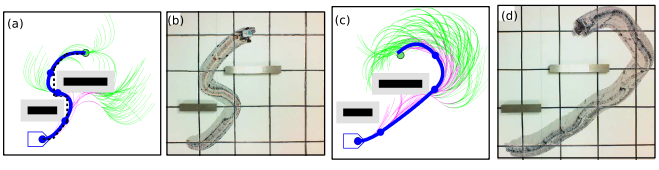

Fig. 6. Trajectories for a medium-acceleration robot ($7cm/s^2$): (a) trajectory planned by DKP; the dotted line represents the trajectory planned by the A^* algorithm on a 25x25 grid, (b) executed trajectory; trajectories for a low-acceleration robot ($3cm/s^2$): (c) trajectory planned by DKP; (d) executed trajectory.

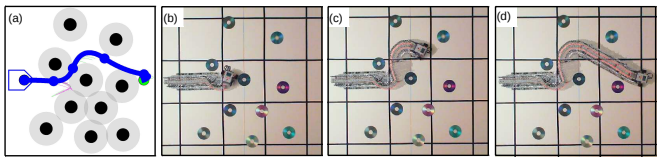

Fig. 7. (a) Planned trajectory; from (b) to (d) executed trajectory

B. Cluttered environment

This example illustrates the ability of DKP to find solutions very quickly in complex environments, using the greedy mode.

1) Description: In this example, the environment contains randomly placed obstacles, creating numerous dead ends where random approaches could classically get stuck for a long time. DKP has been run in greedy mode ($bias = 10$) with the following parameters: $T_{max} = 6s$, $step = 0.5s$; $S_- = 0cm/s$, $S_+ = 10cm/s$, $A_- = 0cm/s^2$, $A_+ = 10cm/s^2$.

2) Results: DKP founds a solution with a very limited exploration tree (compared to those of Fig. 6, which are much more expanded). Even if the search is here strongly biased towards the goal, the time diversity on subtrajectories duration allows DKP to escape from local minima.

C. Overtaking robot

This example illustrates the ability of DKP to handle moving obstacles. Here, only 0-order data on obstacles (i.e. their position) is taken into account during the propagation process. Similarly to [25], higher order data could be integrated in DKP to anticipate the future states of the obstacle.

1) Description: Two robots R_1 (bottom) and R_2 (top) perform straight line moves at constant velocities (respectively $3cm/s$ and $4cm/s$). A third robot R_3 uses DKP to plan the time-minimal trajectory avoiding R_1 and R_2. Since this paper does not focus on trajectory estimation, obstacle trajectories are here known in advance and provided to DKP. DKP has been run in backtracking mode ($bias = 10$) with the following parameters: $T_{min} = T_{max} = 1s$, $step = 0s$; $S_- = 0cm/s$, $S_+ = 20cm/s$, $A_- = 0cm/s^2$, $A_+ = 10cm/s^2$. Additional parameters, specific to the backtracking mode, are: the minimal virtual obstacle radius $r_{vobstacle,p_{k_0 \ldots k_n}} = T_{k_0 \ldots k_{n+1}} \times S_+/size$ with $size = 10$, trigger value to backtrack $trigger = 4$.

2) Results: In this example, DKP computed a trajectory overtaking R_1 while avoiding R_2. Other parameters (a lower acceleration or faster obstacles) lead to a totally different solution (results not shown), consisting, for instance, in following R_1 until the goal is reached.

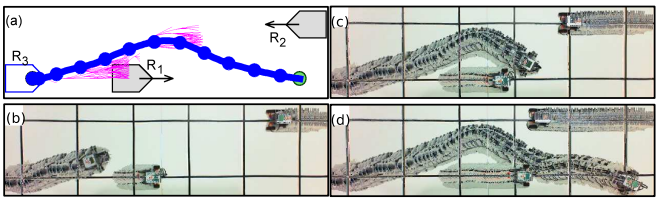

Fig. 8. (a) Planned trajectory; from (b) to (d) executed trajectory

VI. Simulation study

A. Simulations preparation

We produce 100 series of cluttered environments of size $24m \times 24m$ with $N_{obs} \in [0, 10, ..., 100]$ non-overlapping static obstacles with disk shape of radius $1m$, as illustrated in Introduction (Fig. 1). In each of them, we set a random $Start$ state (position and speed vector) and a $Goal$ state in free areas in a square of size $20m \times 20m$ in the center of the environment. The distance between random $Start$ and $Goal$ is between 5 and 10 meters. In addition to the static obstacle avoidance constraints, we set the following kinematic constraints: a linear acceleration between 0 and $1m/s^2$ and a linear speed between 0 and $1m/s$. We use Euclidean distance from the subtrajectory endpoint $p_{k_0 \dots k_n}$ for the heuristic part $h = \rho_{\text{loc}} = d_{Goal}$ from ρ. We use the length of the subtrajectory $p_{k_0 \dots k_n}$ as the distance evaluation function part, noted $d_{subtrajectory}$, between a subtrajectory $p_{k_0 \dots k_n}$ and its previous subtrajectory $p_{k_0 \dots k_{n-1}}$ such as $g = d_{subtrajectory}$. On each environment and with the defined constraints, we use DKP in its three modes described in the first 3 columns of table II. We produce a total number of 1100 simulations for each mode with an increasing number of obstacles. For the backtracking mode, the minimal virtual obstacle radius is set to $r_{vobstacle, p_{k_0 \dots k_n}} = T_{k_0 \dots k_{n+1}} \times S_+ / size$ with $size = 10$. The trigger value to backtrack is set to $trigger = 4$.

To bound the experiment time, the maximum number of allowed propagation processes is set to 500 iterations. To compare the solution, we present the average minimum duration to cover direct line from the $Start$ to the $Goal$ at the maximum allowed speed, when a solution is found by DKP.

B. Results and computation times

Simulations have been run on a 2.53 GHz 64-bits dual-core PC with 4 GB of RAM (DKP implementation is single-threaded). Table II sums up the overall averages and standard deviations for the results of simulations. They show that DKP needs a lot of time diversity to deal with complex environments and obstacle avoidance. With subtrajectories of 0.5, 1, 1.5 and 2, seconds, both the greedy and optimal searches only fail in the typical situation in which the random initial speed vector directly faces an obstacle. In this case, as expected from the A*-based selection process, the optimal mode produces the best solutions but a lot of computation time is used to grow the exploration tree and some situations are not solved within the 500 allowed iterations of the propagation processes. The greedy mode considerably reduces both this computation time and the number of simulations. In this mode, the number of simulations with no solutions (simulations which need more than 500 iterations to get a solution and simulations for which DKP stops on a failure) does not increase. Nevertheless, solution quality is diminished. Backtrack mode exhibits a good balance between the speed of the greedy mode and the quality of solutions in optimal mode. With small subtrajectories of 0.5 seconds, the backtracking mode gets good solutions with a quality close to optimal mode while approaching the greedy mode computation time. As explained by [18], with backtracking mode, we avoid the mismatch between the powerful local planning and the approximate global planning. Even if backtracking mode requires a lot of time to get unstuck from local minima, the solution quality remains close to the optimal mode. One effective optimization should be to register the *parameter space*, and reuse it when backtracked with virtual obstacles. Nevertheless, such a solution would require a lot of memory. This mode would effectively scales in larger environment with the same step size because only one main branche is pursued: it only requires more iterations to provide a solution. The overall complexity then relies on the number of obstacles in the environment but a dynamic window approach should be used in this case.

Figure 9 focuses on the computation times for every set of obstacles in each mode. The evolution of computation time is the same for each mode. When the environment is nearly filled by non-overlapping obstacles, the free space between obstacles forms corridors which naturally guide the search. Therefore, the computation time begins to decrease. As expected, the performance of the backtracking mode are between the optimal and greedy mode.

VII. Conclusion and Future works

DKP is a hybrid path planner for robots which are able to follow spline trajectories. DKP is an intermediate approach between heavily customized approaches which are efficient for specific problems and the more general approaches which use random processes to deal with complex environments and models with many degrees of freedom. Based on a selection/propagation architecture, DKP creates an exploration tree with subtrajectories. The solution, a spline trajectory, can characterize complex maneuvers, even with quadratic subtrajectories, thanks to the efficient relationship between our selection process and our propagation process. In particular, this simplifying choice allows us to exploit an exact representation of all admissible subtrajectories which satisfy all constraints of our problem: the so-called *parameter space*. Our propagation process first sets up those parameter spaces and then identifies the suboptimal subtrajectories with diverse time durations. With these subtrajectories, DKP balances well the complexity of the propagation process and the selection process. Moreover, the *parameter space* construction, separated from the solution search, enables us to build specific adaptation processes, such as the backtracking process.

mode	bias	subtrajectories duration	Computation time	Solution duration	Line solution duration	simulations number	DKP fails	Not finished search
optimal	1	0.5;1;1.5;2	1.54 ± 0.7	$\mathbf{8.79 \pm 1.62}$	7.46 ± 1.39	1100	57	59
greedy	10	0.5;1;1.5;2	$\mathbf{0.4 \pm 0.07}$	12.88 ± 2.87	7.58 ± 1.42	1100	57	0
backtracked	1	0.5	$\mathbf{0.5 \pm 0.19}$	$\mathbf{9.97 \pm 1.98}$	7.59 ± 1.42	1100	61	0

TABLE II

SIMULATIONS ON DKP: BACKTRACKING MODE EXHIBITS A GOOD BALANCE BETWEEN THE SOLUTION QUALITY AND THE COMPUTATION TIME

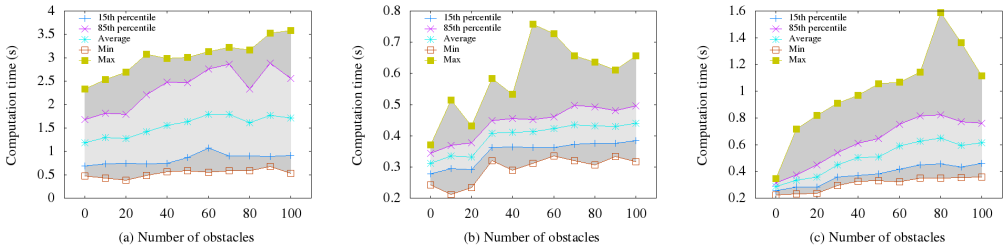

Fig. 9. average, minima, maxima, 15^{th} and 85^{th} percentiles of computation times of 1100 DKP simulations in (a) optimal mode, (b) greedy mode and (c) backtracking mode (100 simulations for [0,10,...,100] obstacles).

In this paper, hardware experiments highlight the strong points of DKP by describing three situations applied to different two-wheeled robots. We illustrate our geometrical approach for constraints by simply providing bounds on the robot's speed and acceleration, a description of the shapes and trajectory for static/mobile obstacle avoidance. This makes DKP easily applicable to various problems for which can set specific restrictions over the *parameter space*. In each described case, we can precisely describe the abilities of the robots. Furthermore, the solutions are directly usable for the control of our robots.

Future work will focus on the adaptation process. We should take advantage of our description of constraints to provide a common reasoning for the exploration process, the analysis of the results and the replanning process.

REFERENCES

[1] M.S. Branicky, M.M. Curtiss, J. Levine, and S. Morgan. Sampling-based planning, control and verification of hybrid systems. *IEE Proceedings Control Theory and Applications*, 153(5):575, 2006.

[2] M.S. Branicky, R.A. Knepper, and J.J. Kuffner. Path and trajectory diversity: Theory and algorithms. In *ICRA'08*, pages 1359–1364.

[3] M. Defoort, J. Palos, A. Kokosy, T. Floquet, W. Perruquetti, and D. Boulinguez. Experimental motion planning and control for an autonomous nonholonomic mobile robot. In *ICRA'07*, pages 2221–2226.

[4] C. Dinont, F. Gaillard, and M. Soulignac. APM(robot): Towards a platform for meta-reasoning in robotic applications. In *Proceedings 5th National Conference on Control Architecture of Robots*.

[5] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, 1993.

[6] C. Urmson et al. A robust approach to high-speed navigation for unrehearsed desert terrain. *Journal of Field Robotics*, pages 467–508, 2006.

[7] R. A. Finkel and J. L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[8] M. Fliess, J. Lévine, and P. Rouchon. Flatness and defect of nonlinear systems: Introductory theory and examples. *International Journal of Control*, 61:1327–1361, 1995.

The authors gratefully thank Gabriel Chênevert and Benjamin Parent for their carreful reading of this paper.

[9] F. Gaillard, C. Dinont, M. Soulignac, and P. Mathieu. Deterministic kinodynamic planning. In *Proceedings of the Eleventh AI*IA Symposium on Artificial Intelligence*, pages 54–61.

[10] B. Gerkey, R.T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *ICRA'03*, pages 317–323.

[11] T. Howard and A. Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *IJRR*, 26(1):141–166, February 2007.

[12] D. Hsu, R. Kindel, J.C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233, 2002.

[13] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *Proceedings of Robotics: Science and Systems, Zaragoza, Spain*, 2010.

[14] A.M. Ladd and L.E. Kavraki. Fast tree-based exploration of state space for robots with dynamics. *Algorithmic Foundations of Robotics VI*, pages 297–312, 2005.

[15] F. Lamiraux, E. Ferré, and E. Vallée. Kinodynamic motion planning: connecting exploration trees using trajectory optimization methods. In *ICRA'04. Proceedings*, volume 4, pages 3987–3992.

[16] S. M. LaValle. *Planning Algorithms*. Cambridge University Press.

[17] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. *IJRR*, 20:278–400, 2001.

[18] M. Likhachev and D. Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *IJRR*, 28(8):933, 2009.

[19] M.B. Milam, K. Mushambi, and R.M. Murray. A new computational approach to real-time trajectory generation for constrained mechanical systems. In *CDC'00*, pages 845–851.

[20] E. Plaku, L.E. Kavraki, and M.Y. Vardi. Discrete search leading continuous exploration for kinodynamic motion planning. In *Robotics: Science and Systems*, pages 326–333, 2007.

[21] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[22] J.H. Reif. Complexity of the generalized mover's problem. In J.T. Schwartz, M. Sharir, and J. Hopcroft, editors, *Planning, Geometry, and Complexity of Robot Motion*, pages 267–281. 1987.

[23] O. Khatib S. Quinlan. Elastic bands: connecting path planning and control. In *ICRA'93*, pages 802–807.

[24] A. Stentz. The focussed D* algorithm for real-time replanning. In *ICRA'05*, pages 1652–1659.

[25] D. Wilkie, J. van den Berg, and D. Manocha. Generalized velocity obstacles. In *IROS'09*, pages 5573–5578.