Deterministic Kinodynamic Planning with Hardware Demonstrations

François Gaillard\textsuperscript{1,2} Michael Soulignac\textsuperscript{1} Cédric Dinont\textsuperscript{1} Philippe Mathieu\textsuperscript{2}

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

\textsuperscript{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).
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 constrained 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 sub-trajectories, 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 \( p_{k_0...k_n}(t) \) denote a subtrajectory in the exploration tree and \( P_{k_0...k_n}(t) \) the overall trajectory which contains the subtrajectories \( p_{k_0}(t), p_{k_0,k_1}(t), ..., p_{k_0...k_n}(t) \). Let \( T_{k_0...k_n} \) 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 \( \rho \) 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 semialgebraic shapes. Other types of constraints could be defined. From the end of selected subtrajectory \( p_{k_0...k_n}(t) \), the propagation process produces a set of new subtrajectories \( p_{k_0...k_n,K+1}(t) \) 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, denoted \( E \). We consider quadratic subtrajectories \( p_{k_0...k_n}(t) = (x_{k_0...k_n}(t) = \alpha_{0x} + \alpha_{1x}t + \alpha_{2x}t^2, y_{k_0...k_n}(t) = \alpha_{0y} + \alpha_{1y}t + \alpha_{2y}t^2) \) in the exploration tree, with \( t \in [0,T_{k_0...k_n}] \). The continuity through initial and speed with the previous subtrajectory \( p_{k_0...k_n}(t) \) sets the lesser degree parameters \( \alpha_{0x}, \alpha_{1x}, \alpha_{0y} \) and \( \alpha_{1y} \) of \( p_{k_0...k_n,k_{n+1}}(t) \). The remaining \( (\alpha_{2x}, \alpha_{2y}) \) parameters of \( p_{k_0...k_n,k_{n+1}}(t) \) (measured in \( m/s^2 \)) set the quadratic shape.
So, the parameter space \( E \) denotes all the allowable values of \((\alpha_x^2, \alpha_y^2)\) [16] which define subtrajectories satisfying the constraints.

### B. The kinodynamic constraints

We define a constraint \( c \) by its constraint function \( f_c(p_{t_0...t_n}(t), t) \), 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 \( c_{\text{Acceleration}}(t) \), within the domain \([A_{−}, A_{+}] \), with constraint function:
  \[ A(t) = \sqrt{(\dot{x}(t))^2 + (\dot{y}(t))^2}, \]
  defines an annulus of radii \( A_{−},A_{+} \) in the basis \((\dot{x}; \dot{y})\);

- the linear speed \( c_{\text{Speed}}(t) \), within the domain \([S_{−}, S_{+}] \), with constraint function:
  \[ S(t) = \sqrt{\dot{\alpha}(t))^2 + (\dot{\beta}(t))^2}, \]
  defines an annulus of radii \( S_{−},S_{+} \) in the basis \((\dot{\alpha}; \dot{\beta})\);

- a forbidden area in the real plane in the basis \((\dot{x}; \dot{y})\) 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 \( p_{t_0...t_n}(t) \), we define affine transformations matrices \( M_c(t) \) from the constraint \( c \) basis to the parameter space basis. Finally, a constraint on a quadratic \( p_{t_0...t_n}(t) \), projected in the parameter basis by \( M_c(t) \times G_c(t) \), is represented by a geometrical shape which restricts the parameter values \((\alpha_x^2, \alpha_y^2)\), hence the allowed shapes of \( p_{t_0...t_n}(t) \). 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 \( T_{t_0...t_n} \) the time horizon of the subtrajectory \( p_{t_0...t_n} \) to be created. \( E(T) \) denotes the parameter space for every time step, noted step, with \( t \in [0; T_{t_0...t_n}] \), such as: \( E(T) = \{f^T_{t = 0} M_c(t) \times G_c(t) \mid c \in C, t \text{ mod step} \equiv 0\} \). Every point \((\alpha_x^2, \alpha_y^2)\) chosen in \( E(T_{t_0...t_n}) \) defines a quadratic \( p_{t_0...t_n}(t) \) of duration \( T_{t_0...t_n} \) which satisfies the kinodynamic constraints at every \( t \in [0; \text{step}..., T_{t_0...t_n}] \). 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 choice is to keep the simpler subtrajectories and to lie on the selection process to achieve complex maneuvers.

### D. Example

Consider the following situation of a robot with:

- initial position \( p(0) = (0, 0)m \) and speed vector \( \dot{v}(0) = (0.1, 0.2)m/s \);
- a security distance \( r_{\text{robot}} = 0.5m \);
- the following constraints, considered every \( \text{step} = 0.1s \) 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 \( \text{Obs} = (2, 0) \).

Figure 3 shows the parameter space for each constraint. Figure 4 shows (a) their intersection and (b) how the choice of a point in \((\alpha_x^2, \alpha_y^2)\) sets the shape of a subtrajectory satisfying the constraints.

### 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 \( \rho_{\text{max}} \) to be minimized is the distance of the subtrajectories endpoint \( p_{t_0...t_n}(T_{t_0...t_n}) \) to Goal. We can analytically find the exact values needed to reach the Goal: \( \alpha_x^2 = (x_g - \alpha_0)^2 - \alpha^2T_{t_0...t_n}^2 \) and \( \alpha_y^2 = (y_g - \alpha_0^2 - \alpha^2T_{t_0...t_n}^2) / T_{t_0...t_n}^2 \). If this point belongs to \( E(T_{t_0...t_n}) \), then it is valid for all motion constraints and these values set the shape of a subtrajectory \( p_{t_0...t_n}(t) \) which reaches Goal at \( t = T_{t_0...t_n} \). 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 \( E(T_{k_0...k_n}) \) defines a subtrajectory on which \( p_{loc} \) 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 \( E(T) = \{ \bigcap_{t=0}^{T} M_c(t) \times G_c(t) \} = \{ \bigcap_{t=0}^{T'} M_c(t) \times G_c(t) \} \cap \{ \bigcap_{t=0}^{T} M_c(t) \times G_c(t) \} \subset E'(T') \) 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 spaces \( E'(T') \) when computing the parameter space \( E(T) \), as shown by Figure 5.

![Diagram](image)

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 \( g \). \( bias \) 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 \( p_{k_0...k_n} \) is selected, DKP creates new subtrajectories \( p_{k_0...k_n}(t) \) with various time durations, based on a step\_variety such that \( T_{\min} \leq T_{k_0...k_n,k_{n+1}} = k_{n+1} \times \text{step\_variety} \leq T_{\max} \). 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 \( P(t) = P_{k_0...k_n}(t) \) is built by retrieving the predecessors of the last subtrajectory \( p_{k_0...k_n}(t) \) which satisfies the stopping criterion. Consequently, the trajectory is made up of \( N \) subtrajectories. The total duration is \( T_{\text{end}} = \sum_{t=0}^{N} T_{k_0...k_n} \). Let \( t \) be in \([0; T_0] \cup \ldots \cup [T_{k_0...k_{n-1}}; T_{k_0...k_n}] \cup \ldots \cup [T_{k_0...k_{N-1}}; T_{k_0...k_N}] \). If \( t \) is in \([T_{k_0...k_{n-1}}; T_{k_0...k_n}] \), the value of the trajectory \( P(t) \) is the value of subtrajectory \( p_{k_0...k_n} \) such as \( P(t) = p_{k_0...k_n}(t - T_{k_0...k_{n-1}}) \).

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 \( P(t) = p_{k_0...k_n}(t) \) by applying the following commands: \( s(t) = \sqrt{x(t)^{\prime 2} + y(t)^{\prime 2}} \) and \( \omega(t) = \frac{y(t) \times (\dot{x}(t) \times (\ddot{x}(t) \times y(t)^{\prime 2})) - \dot{x}(t) \times (\ddot{x}(t) \times y(t)^{\prime 2}))}{x(t)^{\prime 2} + y(t)^{\prime 2}} \). 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 \( p_{k_0...k_{n+1}} \) cannot be pursued, i.e. its parameter space \( \bar{E} \) is empty, \( p_{k_0...k_{n+1}} \) is removed from the exploration tree. The previous subtrajectory \( p_{k_0...k_n} \) 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 \( p_{k_0...k_n} \). When the trajectory \( p_{k_0...k_n} \) backtracking cases reaches a \( \text{trigger} \) value, this subtrajectory is also removed, the previous subtrajectory \( p_{k_0...k_{n-1}} \) 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 odometry 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).

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 $r_{obstacle, p_{obstacle}} = T_{bias} \times S_+/size$ 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.
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 $p_{k_0...k_n}$ for the heuristic part $h = p_{loc} = d_{Goal}$ from $p$. We use the length of the subtrajectory $p_{k_0...k_n}$ as the distance evaluation function part, noted $d_{subtrajectory}$, between a subtrajectory $p_{k_0...k_n}$ and its previous subtrajectory $p_{k_0...k_{n-1}}$ 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 $r_{obstacle,p_{k_0...k_n}} = T_{k_0...k_{n+1}} \times \frac{S_3}{size}$ 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 branch 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.

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 $p_{k_0...k_n}$ for the heuristic part $h = p_{loc} = d_{Goal}$ from $p$. We use the length of the subtrajectory $p_{k_0...k_n}$ as the distance evaluation function part, noted $d_{subtrajectory}$, between a subtrajectory $p_{k_0...k_n}$ and its previous subtrajectory $p_{k_0...k_{n-1}}$ 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 $r_{obstacle,p_{k_0...k_n}} = T_{k_0...k_{n+1}} \times \frac{S_3}{size}$ 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 branch 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.

VI. 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.
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


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

In the table, the authors report on simulations of DKP. The optimal, greedy, and backtracked modes are evaluated across different numbers of obstacles. The table highlights the duration of these simulations, with the backtracking mode exhibiting a good balance between solution quality and computation time.

![Image of table showing simulation results](image)

**Table II**

Simulations on DKP: backtracking mode exhibits a good balance between the solution quality and the computation time.

![Graphs showing simulation results](image)