Deterministic Kinodynamic Planning

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

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

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

Abstract

This paper proposes a Deterministic Kinodynamic Planning approach (DKP for short) offering the degree of generality of randomized techniques for two wheeled robots, such as RRT, but also guarantees on the computation time. This approach combines a global path planner based on an A^*-like algorithm, and a local motion planner based on spline optimization techniques. Implementation details and comparative examples with RRT are provided.

Introduction

Efficiently computing a trajectory that takes into account both kinematic and dynamic constraints is still an open and challenging problem, known as kinodynamic planning. Indeed, in the general case, it is believed to be at least as hard as the generalized mover's problem, which has been proven to be PSPACE-hard (Reif 1987). For this reason, kinodynamic planning is commonly separated into two subproblems: path planning and motion planning. Based on this decomposition, two kind of approaches can be found in literature: decoupled approaches and hybrid approaches.

Decoupled approaches compute a path taking into account only a part of the problem constraints (classically obstacles) and then smooth this path, handling remaining constraints. Variants of Elastic Bands (S. Quinlan 1993) have been widely used in the 90's because they provide bounds on the computation time, allowing online planning. Efficiency of decoupled approaches is explained by the fact that they are generally customized for specific kinodynamic problems. Therefore, they fall short of being able to solve many complicated, high degrees of freedom problems. Moreover, they suffer from incompleteness issues: since the initial path is not guaranteed to be feasible by the robot, the path smoothing phase can fail to respect all kinodynamic constraints. Thus, decoupled approaches may fail to find a solution even if one exists.

Hybrid approaches incorporate a local motion planner within a global path planner. The use of randomized techniques makes them well scalable for problems with high degrees of freedom and/or complicated system dynamics. They are thus usable in a wide range of applications, including robotics, virtual prototyping, and computer graphics. Variants of RRT (LaValle and Kuffner 2001) are still widely used because they quickly explore the state space of the robot, thus reducing the computation time. However, the computation time remains hardly quantifiable in the general case, and may considerably vary between several planning requests on the same problem. Thus, performances of randomized hybrid approaches are by nature unpredictable, in terms of computation time, but also in terms of solution quality.

In this paper, we present a new hybrid approach called *DKP* (for *Deterministic Kinodynamic Planning*). Contrary to RRT which uses random processes, DKP is a deterministic approach. DKP uses the flatness properties of the cinematic model of a non holonomic two wheeled robot to express polynomial path which are usable as commands for the robot. The flatness properties were firstly developed by (Fliess, Lvine, and Rouchon 1995) and used in motion planning works on car with trailers (Lamiraux and Laumond 1998). Contrary to RRT which locally integrates the cinematic model of the robot, our local planner manipulates a local path limited by its time horizon. DKP combines the degree of generality of RRT for two wheeled robots and guarantees the computation time. Implementation details and comparative examples with RRT are provided.

Problem statement

The problem statement is illustrated by figure 1. A trajectory $P(t)$ is seen as a spline with polynomial pieces $p_k(t)$ from a *Start* state to a *Goal* state. Our problem is to incrementally build this spline in a 2-D space. This problem is still challenging (Suryawan, De Don'a, and Seron 2010). This build is guided by an optimization criterion which is application dependant. In this paper, we choose to minimize the distance to the *Goal* state. The resulting trajectory should be executable by a robot, meaning that the trajectory should respect kinodynamic constraints, noted C, on linear speed $V(t)$, linear acceleration $A(t)$ and obstacle avoidance. The

This work is supported by the Catholic University of Lille, as part of a project in the HDC pole.

obstacle avoidance constraint implies that the robot should bypass obstacles with respect to a minimal security distance.

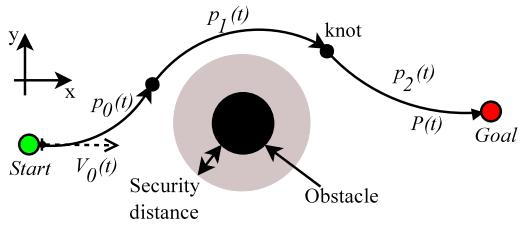

Figure 1: Problem statement: building a trajectory $P(t)$ from *Start* to *Goal* with polynomial pieces $p_k(t)$ guided by an optimization criterion under constraints

The main difficulties of this problem are:

- which kind of polynomial pieces to use, i.e. how to choose their degree?
- how to choose the number of knots in the spline, i.e. the number of polynomial pieces?
- how to define the knots position?
- how to express the kinodynamic constraints in this problem?

An algorithm that solves this problem should be able to explore a large diversity of polynomial pieces in order to find the best solution. With a classical local motion planner which takes into account the kinodynamic constraints, we would need to often call the local planner to create this diversity. Such a solution would be very expensive in computation time. Moreover, to use the motion planner in an online context, its computation time should not exceed the duration of the generated trajectory. Finally, such an algorithm should prove its completeness: in a finite computation time, we should be sure to find a solution, if one exists, using the generated polynomial pieces $p_k(t)$.

Deterministic Kinodynamic Planning (DKP)

Principle

Our solution, named DKP, is a deterministic approach to build spline trajectories from the *Start* state until it reaches a *Goal* state. DKP uses an A*-like algorithm as a global planner. Guided by the distance to the *Goal* as minimization criterion, DKP incrementally builds an approximation of the optimal trajectory $P(t)$ using polynomial pieces $p_k(t)$. These polynomial pieces are generated by our local motion planner which efficiently solves local optimization problems under kinodynamic constraints C. They locally tend to reach the *Goal*. The constraints are kinodynamic constraints over linear speed $V(t)$, linear acceleration $A(t)$ and obstacle avoidance.

The local motion planner The efficiency of DKP lies on our local motion planner, which generates the polynomial pieces $p_k(t)$. The polynomial pieces have a duration of T_k. Two parameters in each $p_k(t)$ are set in order to tune its shape. The main idea of the local planner is to establish which are all the shapes respecting the kinodynamic constraints, *i.e.* all the allowable $p_k(t)$ parameters values. First, this planner creates, for every time t in $[0, T_k]$, a geometrical interpretation of the local problem which contains kinodynamic constraints and the *Start* state. Then, with this geometrical interpretation called constrained parameters space, we can express all the allowable polynomial pieces for this local problem. They respect the kinodynamic constraints for every time in $[0, T_k]$. Second, we find the locally optimal solution on the the constrained parameters space for a specified *Goal*.

Diversity for DKP Using this geometrical interpretation is very efficient to create a large diversity of polynomial pieces. To create the constrained parameters space, the local motion planner evaluates the constraints for every time t in $[0, T_k]$. Because of that, we wan generate a set of constrained parameters space for the time horizon between 0 and $T_k = T_{max}$, where T_{max} is the maximum duration of the polynomial pieces. We can find locally optimal solutions on it. Consequently, the polynomial pieces $p_k(t)$ generated by the local planner for a problem have a large diversity of time horizons. With the geometrical interpretation of constraints, we can identify sub-problems when the constrained parameters space is split into two or more distinct subspaces. When the local motion planner identifies these sub-spaces, it computes the locally optimal solutions in these sub-spaces. For example, in the case of obstacle avoidance, the sub-spaces may express the avoidance in two different directions.

Then, this diversity is used by a global motion planner which is effective to find globally optimal solutions over all these polynomial pieces. The resulting trajectory is built with polynomial pieces that respect the kinodynamic constraints of the robot. Contrary to decoupled approaches, this trajectory is directly usable for the control and the path tracking of an nonholonomic robot with two independent wheels (Defoort et al. 2008).

The resulting trajectories DKP can be tuned in order to find a good solution with less computation time or a near-optimal solution (with the generated polynomial pieces) but with a greater computation time, as shown in figure 2. In this example, trajectories are defined with degree two polynomial pieces under motion constraints of linear speed, linear acceleration and obstacle avoidance. These pieces have a duration of 1 or 2 seconds. The obstacle avoidance constraint takes into account the radius of the obstacle covering disk (in dark) and the security distance for the robot (the light grey part). All the polynomial pieces created by the local motion planner appear on figure 2.

DKP creates trajectories which tend to the direct line from the *Start* to the *Goal* with a length of 17.23m and a maximum speed of 1 m/s. The figure 2(a) shows the result of the search of a near-optimal solution. The trajectory is made up of 16 pieces, with a length of 21.53m and a duration of 23s. The computation time is 7 times higher than the trajectory duration on a recent computer. The figure 2(b) illustrates the result of a greedy search of a good solution. The resulting trajectory is made up of 18 pieces, with a length of 22.06m and a duration of 26s. The computation time is half a time lesser than the trajectory duration. In a real time execution, the greedy mode should be chosen in order to find solutions almost as good than the optimal one. Contrary to RRT (LaValle and Kuffner 2001), DKP is a deterministic approach, meaning that two separated executions of the algorithm on the same problem will provide exactly the same solutions.

Figure 2 also shows the ability of DKP algorithm to handle complex environments. We do not need to increase the degree of the polynomial pieces in the local motion planner and the inherent time to compute them: degree two polynomials are enough to create complex trajectories in DKP.

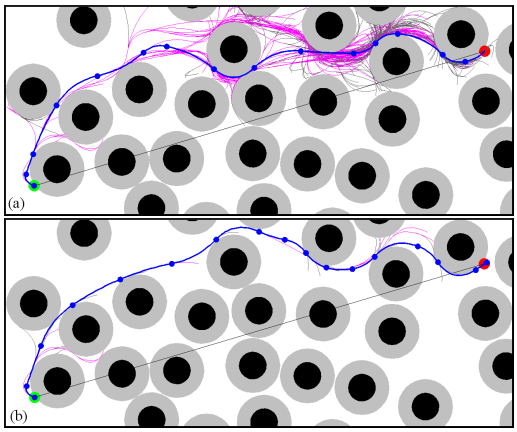

Figure 2: DKP final trajectories in bold with (a) optimal search and (b) greedy search

The global motion planner

DKP is an A*-like algorithm. We use polynomial pieces as nodes of an exploration tree in which DKP incrementally builds the solution. The neighbourhood of a polynomial piece is determined by the local motion planner. The result of DKP is a spline trajectory $P(t)$ composed of polynomial pieces.

DKP, which implementation is given in algorithm 1, slightly differs from the traditional A*. A node contains a polynomial piece $p_k(t)$ and its duration T_k. The neighbourhood of a node is built by the local motion planner from a polynomial piece $p_k(t)$. The neighbourhood of piece $p_k(t)$ is generated by the local motion planner using the ending state $p_k(T_k)$ as a *Start* state

Algorithm 1 DKP(*Start*, *Goal*)

```
1:  open_set.add(Start) closed_set ← ∅
2:  Start.g_score ← 0
3:  Start.h_score ← distance(Start, Goal)
4:  Start.f_score ← Start.h_score + Start.g_score
5:  while open_set ≠ ∅ do
6:     piece ← remove the piece from open_set with the
       lowest f_score
7:     if stop_test(piece) then
8:        return success
9:     end if
10:    closed_set.add(piece)
11:    T_k ← piece.total_time end ← piece(T_k)
12:    neighbours ← local_planner(end, Goal, {C}, T_k)
13:    filtered_neighbours ← filter(neighbours)
14:    for neighbour ∈ filtered_neighbours do
15:       if neighbour ∈ closed_set then
16:          break the loop
17:       end if
18:       evaluate_node(neighbour, piece, Goal, open_set)
19:    end for
20: end while
21: return failure
```

to the next polynomial pieces $p_{k+1}(t)$. New nodes are added to the problem each time a neighbourhood is created.

Contrary to usual A* path planners, the polynomial pieces used in DKP lies on a continuous space. It means that two nodes rarely coincide. Consequently, the number of nodes to evaluate could grow very fastly, affecting the efficiency of the search. To prevent this problem, polynomial pieces are registered with discretization criteria upon polynomial piece final point, speed vector direction, speed vector norm and length of the piece. Two polynomial pieces coincide if they share the same discretization.

Then, newly polynomial pieces from a neighbourhood are filtered (*filter*() called in the algorithm 1). The resulting *filtered_neighbours* only contains pieces which do not coincide with already registered polynomial pieces. Thus, we can control the number of polynomial pieces created by DKP and, consequently, the computation time of the algorithm can be bounded.

The final trajectory $P(t)$ is built by retrieving the predecessors of the last polynomial piece $p_N(t)$ which satisfies the stopping criterion. Consequently, the trajectory is made up of N polynomial pieces. The total duration of $P(t)$ is found by summing the respecting durations of the N polynomial pieces such as $T_{end} = \sum_0^N T_k$. The evaluation of $P(t)$ is the sum of the successive evaluations of each polynomial pieces during their relative durations. Let $t = t' + \sum_0^{k-1} T_k$ and $t \in [0; T_0] \cap ... \cap [T_{k-1}; T_k] \cap ... \cap [T_{N-1}; T_{end}]$. If t is in $[T_{k-1}; T_k]$, the value of the trajectory $P(t)$ is the value of polynomial piece k such as $P(t) = p_k(t - T_{k-1})$.

Algorithm 2 *evaluate_node*(*neighbour*, *piece*, *Goal*, *open_set*)

```
1: new_score ← piece.g_score + distance(piece, neighbour)
2: if neighbour ∉ openset then
3:    open_set.add(neighbour)
4:    best_score ← true
5: else if new_score < neighbour.g_score then
6:    best_score ← true
7: else
8:    best_score ← false
9: end if
10: if best_score then
11:    neighbour.predecessor ← piece
12:    neighbour.g_score ← new_score
13:    neighbour.h_score ←distance(neighbour, Goal)
14:    neighbour.f_score ←neighbour.h_score + neighbour.g_score
15: end if
```

The local motion planner for DKP

DKP could use any local motion planner. The neighbourhood of a node is built from the registered polynomial piece $p_k(t)$. Generating only one new polynomial piece $p_{k+1}(t)$ as neighboor of polynomial piece $p_k(t)$ would lead the algorithm to fail by producing trajectories which do not respect motion constraints. This kind of problem is illustrated by figure 3 where the lack of diversity in polynomial pieces should lead to unsolvable situations. Diversity in the polynomial pieces is the key to face heavily constrained environment, as shown by figure 4.

We need to expect which are the trajectories that respect the motion constraints but producing a large diversity of constraints-respecting polynomial pieces would be very expensive. Our local motion planner creates a constrained parameters space. It is a geometrical approach to the local motion planning problems in which we express all the polynomial pieces which respect constraints. The cost of generating this constrained parameters space is separated from the cost of finding the optimal solution. The overall computation cost of generating allowable polynomial pieces is about the same than finding one solution. Consequently, creating a large neighbourhood of polynomial pieces $p_k(t)$ is not expensive. This makes DKP efficient to explore all these polynomial pieces and create near-optimal trajectories $P(t)$ with them.

The local motion planner

We introduce in this section our local motion planner under kinodynamic constraints. This local motion planner creates a constrained parameters space to represent all the allowable polynomial pieces. Then, the local motion planner finds optimal solutions over this constrained parameters space.

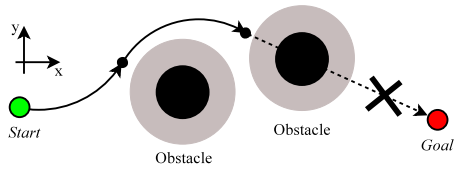

Figure 3: Illustration of a naive planning: with one polynomial piece generated at a time, the algorithm leads to an unsolvable situation

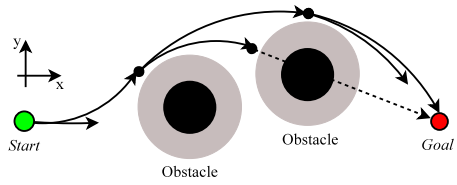

Figure 4: Illustration of a planning with diversity on polynomial pieces: the unfeasible situation is proposed but the algorithm can generate an alternative trajectory

The polynomial pieces equations

In DKP, we consider polynomial pieces of degree 2 to build the trajectory $P(t)$. The polynomial piece $p_k(t)$ equations are two polynomials in 2D space such as:

$$p_k(t) = \begin{cases} x(t) = c_0 + c_1 t + c_2 t^2 \\ y(t) = d_0 + d_1 t + d_2 t^2 \end{cases}$$

The speed vector $Sv_k(t)$ is found by derivation of position $P(t)$. The acceleration vector $Av_k(t)$ is found by derivation of speed vector $Sv(t)$. A polynomial piece $p_k(t)$ is defined with $t \in [0, T_k]$. T_k is the time horizon of the polynomial piece $p_k(t)$.

A polynomial piece $p_k(t)$ is the successor of a previous polynomial piece $p_{k-1}(t)$. The continuity of the polynomial pieces is respected through position and speed. Thus, the lesser degree parameters of $p_k(t)$ are set:

- the initial position:
  $p_{k-1}(T_{k-1}) = p_k(0) = (x(0), y(0)) = (c_0, d_0)$;
- the initial speed vector:
  $Sv_{k-1}(T_{k-1}) = Sv_k(0) = (\dot{x}(0), \dot{y}(0)) = (c_1, d_1)$;

The remaining (c_2, d_2) couple parameters the polynomial piece shape. This polynomial piece $p_k(t)$ should respect the motion constraints: we want to establish the constrained parameters space as a reflect of the parameters values for which the polynomial piece $p_k(t)$ respects all the constraints of the motion problem.

Constraints

The local motion planner optimizes polynomial pieces under kinodynamic constraints and obstacle constraints. We define a constraint $C(t)$ by:

- $f_C(p_k(t), t)$ the constraint function applied on the polynomial piece $p_k(t)$ at time step t;
- the domain $[D_-, D_+]$ within which the polynomial piece $p_k(t)$ is constrained by $f_C(p_k(t), t)$ with:
  $D_- \leq f_C(f(t), t) \leq D_+$.

We only consider in this paper the following **motion constraints** applied to the polynomial piece $p_k(t)$:

- the linear speed $C_{Speed}(t)$, within the domain $[S_-, S_+]$, with constraint function: $S(t) = \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2}$.
- the linear acceleration $C_{Acceleration}(t)$, within the domain $[A_-, A_+]$, with constraint function: $A(t) = \sqrt{\ddot{x}(t)^2 + \ddot{y}(t)^2}$.
- obstacle avoidance of an obstacle $Obs = \{x_{obs}, y_{obs}\}$, giving a distance constraint $C_{Obstacle}(Obs, t)$, within the domain $[D_{obs-}, D_{obs+}]$, with constraint function: $d_{obs} = \sqrt{(x_{obs} - x(t))^2 + (y_{obs} - y(t))^2}$.

Let $MC = \{C_{Speed}(t), C_{Acceleration}(t), C_{Obstacle}(Obs, t)\}$ be a set of the motion constraints. The constraint for obstacle avoidance can be extended to mobile obstacle avoidance by using their predicted trajectory $P_{Mobs} = (x_{Mobs}(t), y_{Mobs}(t))$ in the constraint function, instead of static position (x_{obs}, y_{obs}).

We can set bounds on the angular speed ϕ_d in the clockwise direction with equation: $\phi_d(t) = (\dot{y}(t) \times \ddot{x}(t) - \dot{x}(t) \times \ddot{y}(t))/v_1(t)^2$. Using interval arithmetics (Moore 1966), if $\phi_d(t)$ is within the domain $[\phi_{d-}, \phi_{d+}]$, then these bounds depend on the limits on linear speed and linear acceleration with the following relation: $\phi_{d+} = -\phi_{d-} = (2S_+A_+)/S_-^2$. If $S_- \neq 0$, we enforce a bound on the angular speed.

Constrained parameters space

We remark that the motion constraints equations applied to polynomial piece $p_k(t)$ correspond to the expression of an ellipsoid in a 2-D space. The equation respects the following relationship: $\sqrt{D_-}^2 \leq (f(t) - xc)^2 + (g(t) - yc)^2 \leq \sqrt{D_+}^2$, where:

- $f(t)$ is the function of free parameter c_2 in $x(t)$;
- $g(t)$ is the function of free parameter d_2 in $y(t)$;

A polynomial piece $p_k(t)$ is valid for the constraint C at time step t if the free parameters (c_2, d_2) couple from the polynomial piece is an inner point in the ellipsoid from constraint C. Let $I_{C,p_k}(t)$ denote the set of valid parameters values for a constraint $C(t)$ at time *step* t for the *Start* state of a polynomial piece $p_k(t)$. The constrained parameters space for the motion constraints MC, noted E_{p_k}, for a polynomial piece of duration T_k is: $E_{p_k}(T_k) = \{\bigcap I_{C,p_k}(t) \mid t \in [0, T_k], C \in MC\}$. Other kinds of constraints could be expressed if we are able to translate their equations applied to polynomial piece $p_k(t)$ to a geometrical shape in the constrained parameters space.

Example

We set the following situation, illustrated by the figure 5:

- the polynomial piece $p_k(t)$ is computed for every time *step* $t \in [0, T_k = 10]$;

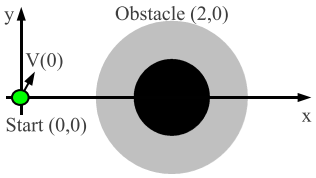

Figure 5: Representation of the initial situation: the robot is located at timestep $t = 0$ in $P(0) = (0, 0)$, with a speed vector $(0.1, 0.2)$. An obstacle is set in $Obs = (2, 0)$

- the motion constraints are considered every *step* $= 0.1$ seconds;
- we use the following constraints in the set of motion constraints MC:

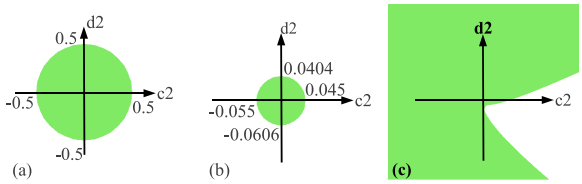

Figure 6: The constrained parameters space for (a) the linear acceleration constraint, (b) the linear speed constraint and (c) the distance constraint from obstacle Obs

The three sub-figures from figure 6 represent the constrained parameters spaces for each constraint from MC and for every $t \in [0, 10]$ seconds, created thanks to the algorithm 3. In the colored areas, the parameters (c_2, d_2) set a polynomial piece which verifies the specified motion constraint for every $t \in [0, 10]$s:

- the linear acceleration $C_{Acceleration}(t)$ is bounded between 0 and $1m/s^2$ with equation: $0 \leq 2{c_2}^2 + 2{d_2}^2 \leq 1$ (figure 6(a));
- the linear speed $C_{Speed}(t)$ is bounded between 0 and $1m/s$ with equation: $0 \leq (c_1 + 2c_2t)^2 + (d_1 + 2d_2t)^2 \leq 1$ (figure 6(b));
- we set a fixed obstacle $Obs = (2, 0)$, giving a distance constraint $C_{Obstacle,Obs}(t)$ bounded between 0.6 and ∞ meters (we set 1000 meters as infinity in this situation) with equation: $0.6 \leq (2 - (0 + 0.1t + c_2t^2))^2 + (0 - (0 + 0.2t + d_2t^2))^2 \leq 1000$ (figure 6(c))

The valid parameters space $E_{p_k}(T_k)$ is shown in figure 7(a): this is the intersection of the three previous surfaces, with a center in $(-0.0555, 0.0606)$ and included in a rectangle of width 0.1004 and height 0.101. Figure 7(b) illustrates the impact of the choice of a point (c_2, d_2) on the shape of a polynomial piece: choosing (c_2, d_2) outside the area means that it exists at least one instant $t \in [0, 10]$ for which a constraint is not respected by the polynomial piece $p_k(t)$, for example the obstacle avoidance as in the figure.

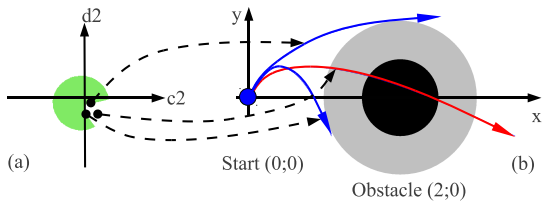

Figure 7: Figure (a) represents all valid parameters (c_2, d_2) values eligible for all constraints. Figure (b) shows the impact of (c_2, d_2) on the shape of the polynomial piece. Choosing parameters (c_2, d_2) outside the constrained parameters space impacts the respect of the constraints, here the obstacle avoidance

In this local motion planner, adding further constraints, for example obstacles, is like creating a new area representing its constrained parameters space and, then, to intersect it with the previously computed valid constrained parameters space E.

Finding solutions in the constrained parameters space

The constrained parameters space $E_{p_k}(T_k)$ is a geometrical representation of allowable parameters for polynomial piece $p_k(t)$. Choosing a point (c_2, d_2) in E is equivalent to choosing a polynomial piece which respects all motion constraints. This is the starting point to create a large diversity of allowable and optimal polynomial pieces used in DKP.

A local optimization problem Let (x_g, y_g) denote the *Goal* position. With the valid constrained parameters space $E_{p_k}(T_k)$, we can express a quadratic optimization problem with boundary limited parameters. The function to be minimized is the distance between the end of trajectory $p_k(t)$ of time horizon T_k and the *Goal*: $distance(P(T_k), Goal)$. The boundary on the parameters are set by the constrained parameters space. This also means that the polynomial piece set by these parameters respects the kinodynamic constraints. Because our problem contains two distincts parameters, c_2 and d_2, this boundary can be seen as the border of a rectangle, denoted *rec*.

Algorithm 3 $space_{constraints}(T_k)$

Require: $MC \leftarrow \{C_{Acceleration}, C_{Speed}, C_{Obstacle,Obs}\}$
Require: T_k the horizon time of the polynomial piece $p_k(t)$
Require: *step* the step to discretize the time
1: $E \leftarrow E(step)$
2: $nb_{step} \leftarrow T_k/step$
3: **for** $i = 1$ **TO** nb_{step} **do**
4: $t \leftarrow step \times i$
5: **for** $C \in MC$ **do**
6: $E'_{p_k}(t) \leftarrow I_{C,p_k}(t)$
7: $E(T_k) \leftarrow E' \cap E$
8: **end for**
9: **end for**

Direct solution It is easy to analytically find the exact values of parameters c_2 and d_2 needed to reach the *Goal*:

$$\begin{cases} c_2 = (x_g - c_0 - c_1 T_k)/{T_k}^2 \\ d_2 = (y_g - d_0 - d_1 T_k)/{T_k}^2 \end{cases}$$

If the point (c_2, d_2) belongs to $E_{p_k}(T_k)$, the polynomial piece set by these parameters is valid for all motion constraints and the local problem is solved. Otherwise, the optimal solution for $E_{p_k}(T_k)$ is on the boundary of this space.

Approximating the valid constrained parameters space To work on the boundary of $E_{p_k}(T_k)$, we propose a tiling with rectangles with the use of the QuadTree algorithm (Finkel and Bentley 1974). We get three kinds of tilings where the rectangles *rec* have:

- an empty intersection with E: $rec \cap E = \emptyset$;
- a full intersection with E: $rec \cap E = rec$;
- an incomplete intersection with E: $rec \cap E \neq \emptyset \neq rec$;

This algorithm only iterates the tiling over uncertain intersections, in order to get a better tiling over the border of E. Let Rec_{full} denote the set of rectangles with a full intersection with E.

Finding near-optimal solution Using the paving Rec_{full}, we can define a set of optimization problems where $p_k(t)$ is the polynomial piece to be optimized over time T_k. c_2 and d_2 are the parameters of the function to be minimized and are bounded by *rec* to make them respect the motion constraints. This problem is effectively solved by BLMVM (Benson and More 2001).

Back to the example The *Goal* is set in $(4, 0)$.
The time horizon is $T_k = 10$s. The local planner finds the following result:

$$\begin{cases} x(t) = 0.1t + 0.02726t^2 \\ y(t) = 0.2t - 0.00753t^2 \end{cases}$$

Using interval arithmetics (Moore 1966), we can project E on the (x, y) space. Thus, we can see on figure 8 the solution and all end points that are reachable for these motion constraints.

A large set of optimal solutions without additional cost The local motion planner complexity essentially lies on the constrained parameters space E. We can take advantage of this cost to find additional solutions with no real additional cost in two ways. First, E is sequentially built by intersecting the constrained parameters space every $step_{space}$ seconds from $step_{space}$ to T_k, the duration of the polynomial piece $p_k(t)$. We can find solutions on E for the intermediate $step_{space}$ time steps. We define T_{max} as the maximum duration of the polynomial pieces. Second, E may be formed with two or more distinct areas. We see these areas as

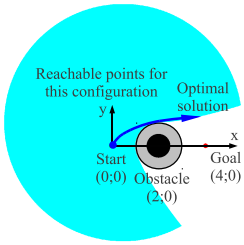

Figure 8: Solution drawn in bold, with equation $\{x(t) = 0.1t + 0.02726t^2; y(t) = 0.2t - 0.00753t^2\}$. The light grey area represents the set of reachable points for this problem, as a projection of E on 2-D $\{x, y\}$ space

Algorithm 4 *local_planner*$(start, Goal, \{C\}, T, depth)$

```
lets solutions be the set of polynomial pieces
for step ∈ T do
    E ← space_constraints(step)
    sub_spaces ← split(E)
    optimal ← compute_optimal_config({C}, start)_
    for sub_space ∈ sub_spaces do
        if optimal ∈ sub_space then
            solutions ← create_polynomial(optimal)
        else
            E_rec ← QuadTree(sub_space, depth)
            near_optimal ← find_best_optimize(E_rec)
            solutions ← create_polynomial(near_optimal)
        end if
    end for
end for
return solutions
```

distinct sub-spaces on which we can find optimal solutions.

The algorithm 4 shows the final implementation of our local motion planner. This planner is able to create a large diversity of polynomial pieces by varying time limit and by finding locally optimal solutions.

DKP with the example. The result of the application of DKP to the local motion planner example is illustrated by the figure 9. The neighbourhood of a polynomial piece $p_k(t)$ is the set of polynomial pieces $p_{k+1}(t)$ found every 0.5 from $T_k = 0.5$s to $T_k = T_{max} = 10$s. This first resulting trajectory is the concatenation of four polynomial pieces:

$$\begin{cases} x(t) = 0.1t + 0.1141t^2 \\ y(t) = 0.2t + 0.0250t^2 \end{cases} \quad for\ t \in [0; 3.5]s;$$

$$\begin{cases} x(t) = 1.7481 + 0.8989t + 0.0916t^2 \\ y(t) = 1.006 + 0.3750t - 0.4857t^2 \end{cases} \quad for\ t \in [3.5; 4]s;$$

$$\begin{cases} x(t) = 2.2205 + 0.9906t - 0.1657t^2 \\ y(t) = 1.0723 - 0.1107t - 0.4485t^2 \end{cases} \quad for\ t \in [4; 4.5]s;$$

$$\begin{cases} x(t) = 2.6744 + 0.8248t - 0.0810t^2 \\ y(t) = 0.9048 - 0.5592t + 0.0534t^2 \end{cases} \quad for\ t \in [4.5; 6.5]s;$$

This trajectory $P(t)$ has a total duration of 6.5 seconds and a length of 4.61 meters.

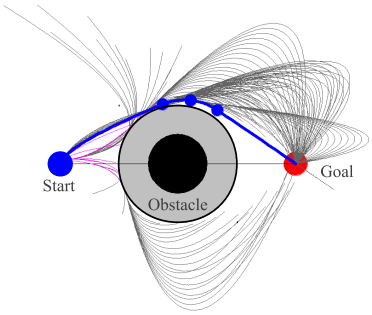

Figure 9: Representation of polynomial pieces evaluated by DKP in *open_set* (dark grey) and *closed_set* (light grey); in bold the first trajectory which satisfies the stopping criterion and reaches the *Goal*

Comparative examples

Randomized Kinodynamic Planning

RRT is an hybrid approach which randomly grows an exploration tree in the state space from the *Start* state, until it becomes close enough to the *Goal* state. As explained before, hybrid approaches incorporate a local motion planner within a global path planner. In the case of RRT, the global planner is used to select the branch of the tree to grow, and the local planner actually grows the selected branch with respect to kinodynamic constraints.

An example is provided in figure 10. The global planner randomly picks a state S_{rand} in the state space, and the nearest state S_{near} in the existing tree is selected for expansion. Next, from S_{near}, the local planner integrates all possible controls for the robot during a fixed interval ΔT, yielding to valid branches candidates. Among these candidates, the nearest (collision-free) branch to S_{rand} is selected, and the corresponding leaf is inserted in the tree, as a successor of S_{near}.

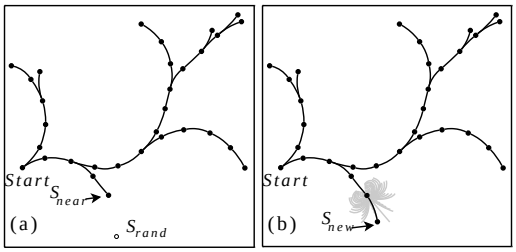

Figure 10: State space exploration using RRT. (a) The global planner selects a node S_{near} for expansion; (b) The local planner generates a valid successor S_{new}

Examples

As RRT, DKP creates a tree with evaluted paths in order to explore the environment. The aim of this section is to apply RRT and DKP on same environments and compare their results, in terms of computation time

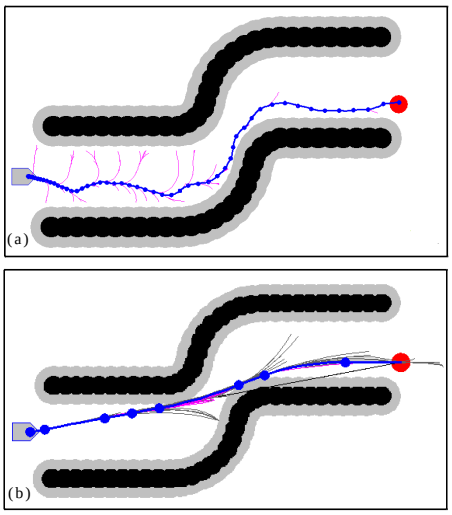

Figure 11: Candidate and final trajectories found by (a) RRT and (b) DKP. Final trajectories are illustrated in bold: (a) length: 9.04m, duration: 12.6s, found in 67s. (b) length: 8.19m, duration: 3.2s, found in 27.6s

and solution quality (length, duration). Unfortunately, DKP is by nature heuristically guided, and not RRT. In this context, comparing the search effort would be meaningless. To make RRT and DKP search homogeneous (both guided toward the *Goal*), we ran RRT with an improved strategy called "Goal bias" (Ferguson and Stentz 2006): with probability $1-p$, a random sample is used to guide the exploration tree, and with probability p, the *Goal* state is used. As the value of p is higher, RRT behaves greedily. We opted for a common value found in literature $p = 0.2$.

Solutions qualified as "final trajectories" correspond to the first solution found by each algorithm. Computation times have been obtained on a 1.83 GHz dualcore PC with 1 GB of RAM. We give two examples with a corridor environment (figure 11) and a cluttered environment (figure 12).

Conclusion and future works

DKP effectively solves the problem of planning a trajectory seen as a spline under kinodynamic constraints. Our algorithm mixes a global planner based on A* guided by an optimization criterion and our local motion planner which solves local motion problems under kinodynamic constraints. First, the local motion planner creates a geometrical interpretation of the motion constraints. The resulting constrained parameters space contains all the allowable parameters of the potential solutions. Second, we can identify sub-problems when the constrained parameters space is formed with distinct areas. Then, the effectiveness of our solution comes from the ability of the local motion planner to create a large diversity of allowable polynomial pieces by varying their time horizons. DKP uses this diversity

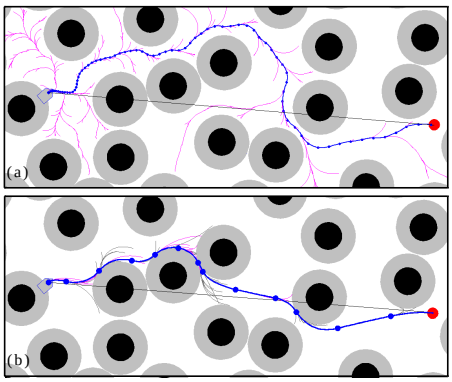

Figure 12: Candidate and final trajectories found by (a) RRT and (b) DKP. Final trajectories are illustrated in bold: (a) length: 19.44m, duration: 18.51s, found in 168s. (b) length: 16.41m, duration: 18s, found in 13.7s

to deal with complex environments. The future works will expand the expression of obstacles. We will focus on the links between DKP and its local motion planner to improve the quality of solutions. With more complex obstacles than covering circles, local minima will appear in the search and disturb the environment exploration.

References

Benson, S. J., and More, J. J. 2001. A limited memory variable metric method in subspaces and bound constrained optimization problems. Technical report, in Subspaces and Bound Constrained Optimization Problems.

Defoort, M.; Palos, J.; Kokosy, A.; Floquet, T.; and Perruquetti, W. 2008. Performance-based reactive navigation for non-holonomic mobile robots. *Robotica* 27(02):281–290.

Ferguson, D., and Stentz, A. 2006. Anytime rrts. In *IROS, Beijing, China*, 5369–5375.

Finkel, R. A., and Bentley, J. L. 1974. Quad trees a data structure for retrieval on composite keys. *Acta Informatica* 4:1–9. 10.1007/BF00288933.

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

Lamiraux, F., and Laumond, J.-P. 1998. A practical approach to feedback control for a mobile robot with trailer. In *ICRA*, 3291–3296.

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

Moore, R. E. 1966. *Interval Analysis*. Prentice-Hall.

Reif, J. 1987. Complexity of the generalized mover's problem. In Schwartz, J.; Sharir, M.; and Hopcroft, J., eds., *Planning, Geometry, and Complexity of Robot Motion*. Ablex Publishing Corporation; Norwood, NJ. 267–281.

S. Quinlan, O. K. 1993. Elastic bands: connecting path planning and control. In *ICRA, Atlanta, USA*, 802–807.

Suryawan, F.; De Don'a, J.; and Seron, M. 2010. On splines and polynomial tools for constrained motion planning. Mediterranean Conference on Control and Automation.