Path planning using Constrained Differential Dynamic Programming

Gael Colas
Department of Aeronautics
and Astronautics
Stanford University
Stanford, USA
Email: colasg@stanford.edu

Ianis Bougdal-Lambert
Department of Aeronautics
and Astronautics
Stanford University
Stanford, USA
Email: ianisbl@stanford.edu

***Abstract*—Off-line path planning is a mature field, with numerous efficient and robust algorithms that allow to compute optimal paths that can avoid potential obstacles in a static environment. A*, to name one, is a very efficient algorithm that finds the optimal path by discretizing the sate space and searching among all possible paths. However, better methods exist to solve problems of motion planning in higher dimensions or when the dynamics and the objective cost function are more complex. In this paper, we investigate an extension of Differential Dynamic Programming to motion planning problems with nonlinear constraints, and analyze its efficiency by comparing it to A*. We show that the new method studied here succeeds in finding a locally optimum trajectory that minimizes the objective function while avoiding the obstacles. We note however that for the simple models that we consider, the method is much slower than A* and suffers from many drawbacks which make it more suited for problems of greater complexity.**

I. Introduction

Path planning is a corner stone of robotic autonomy. Given an environment that potentially contains obstacles, an autonomous robot has to find an optimal collision-free path from a starting position to a goal position. Optimality can obviously have different meanings depending on the type of application and refers to a set of metrics defined by the problem to solve. It is usually natural to consider a path optimal if it is of minimal length, but other additional costs can also be considered.

We focus here on the specific case of off-line path planning, where the environment is deterministic, perfectly known in advance, and static. Furthermore, we assume that an initial feasible but sub-optimal path is known. To optimize that trajectory in the face of potential obstacles, we study the use of a new method of Differential Dynamic Programming. Differential Dynamic Programming (DDP) is widely used in nonlinear optimal control problems and specifically for trajectory optimization. It has the advantage of efficiently handling nonlinear cost functions but does not handle state or control constraints, which makes it unusable even for simple path planning when obstacles are present.

Here we implement an improvement to DDP, developed in [1]. In their article, the authors modify the DDP algorithm to handle any kind of nonlinear inequality constraints on state and control. The resulting Constrained Differential Dynamic Programming (CDDP) algorithm is tested on a simple model of a point mass evolving in a constrained environment, and its efficiency is tested against the classic A* algorithm [2].

II. Problem Statement and Model

In this paper, we implement CDDP first on a single-objective problem, then on a multi-objective problem.

The single-objective problem can be formulated as follows: given a 2D continuous map of size $[25 \times 25]$ containing two circular obstacles, find the shortest path from $[1, 1]$ to $[25, 25]$. For computational reasons, we fix the number of time steps to $N = 50$. We also provide a sub-optimal trajectory that avoids the two obstacles (see Fig.1).

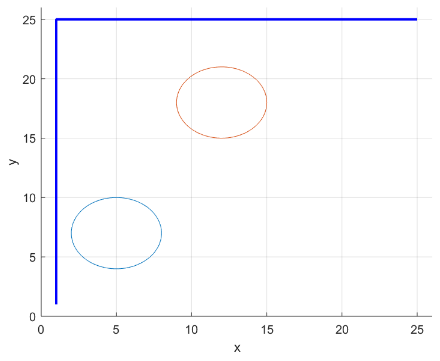


Fig. 1: 2D map with obstacles and initial, sub-optimal trajectory in blue

The multi-objective problem can be formulated as follows: we transform the previous 2D into a 3D map by considering a variable cost on the map. That cost can be seen as an elevation gain, that would naturally incur a greater cost for the robot. Therefore, the objective is now to find a path as short as possible that also minimizes the cumulative cost along it. A coefficient α is introduced to balance the importance of the two objectives. We once again provide an initial feasible but sub-optimal trajectory.

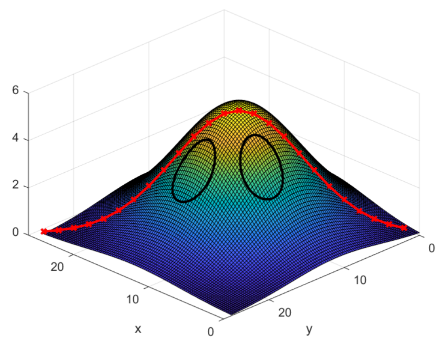


Fig. 2: 3D level map with obstacles and initial, sub-optimal trajectory in red

In both cases, the state is defined as the current position

$$X_k = [x_k, y_k]^T$$

and the controls are taken to be the velocity inputs at every time step

$$U_k = [\dot{x_k}, \dot{y_k}]^T.$$

Naturally, the discrete dynamics of our system are then

$$X_{k+1} = f(X_k, U_k) = X_k + \Delta t.U_k$$

where Δt is taken to be 1.

The length of the path being equal to the integral of the norm of the velocity along the path, we define our running cost in the simple case as the squared norm of the velocity inputs

$$l(X_k, U_k) = ||U_k||^2 = U_k^T\, R\, U_k, \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

In the second case, we add a cost directly proportional to the elevation at the given point:

$$l(X_k, U_k) = U_k^T\, R\, U_k + \alpha\, \text{elev}(X_k)$$

The total cost of the trajectory is then

$$J(X, U) = \sum_{k=0}^{N-1} l(X_k, U_k) + l_f(X_N)$$

where

$$l_f(X_n) = (X_N - X_{\text{goal}})^T\, Q_f\, (X_N - X_{\text{goal}})$$

$$Q_f = \begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}$$

is a final cost that greatly penalizes the distance of the final point to the goal point.

The obstacles are represented through inequality constraints on the state. Namely, for every circular obstacle i of center $X_{c,i} = [x_{c,i},\, y_{c,i}]^T$ and radius r_i, the corresponding inequality constraint writes

$$g_i(X_k, U_k) = r_i^2 - (X_k - X_{c,i})^T\, I\, (X_k - X_{c,i}) \leq 0$$

III. Technical Approach

In this section, we provide a quick overview of the principles of Differential Dynamic Programming and its extension, Constrained Differential Dynamic Programming, as explained in [1]. More precisely, we focus on their application to optimal control.

A. *Differential Dynamic Programming*

The key idea behind DDP is to solve the optimal control problem by using Dynamic Programming. The optimal value function at the time step k and state X is defined as

$$V_k(x) = \min_U \sum_{j=k}^{N-1} l(X_j, U_j) + l_f(X_N).$$

Using Bellman's Optimality Principle, we have

$$V_k(x) = \min_u l(x, u) + V_{k+1}(f(x, u))$$

with the boundary condition

$$V_N(x) = l_f(x)$$

DDP solves the optimal control problem through an iterative method. Each iteration is the succession of a backward pass and a forward pass. At the end of each iteration, the trajectory is updated.

Backward pass

In the backward pass, the value function V is approximated as a quadratic function along the current trajectory, starting from the end of the trajectory. We define an action-value function Q that evaluates at each step the cost of taking a little different action from a little different state than the one on the current trajectory, and to follow the optimal policy from there.

$$Q_k(\delta x, \delta u) = l(x_k+\delta x, u_k+\delta u) + V_{k+1}\big(f(x_k+\delta x, u_k+\delta u)\big)$$

The action-value is approximated by a quadratic function $\tilde{Q}$ using Taylor Expansion around $(\delta x, \delta u) = (0, 0)$. We then use CVX to minimize that quadratic approximation with respect to δu:

$$V_k(x) = \min_{\delta u} \tilde{Q}_k(\delta x, \delta u). \tag{1}$$

This minimization results in a linear feedback $\delta u = h(\delta x)$.

Forward pass

During the forward pass, we use that approximated value function to produce a new optimal trajectory that minimizes the approximated cost. More precisely, we use the linear feedback found above to update the nominal trajectory

$$U_k^{new} = U_k + h(X_k^{new} - X_k)$$

and

$$X_{k+1}^{new} = f(X_k^{new}, U_k^{new})$$

B. Constrained Differential Dynamic Programming

In their paper [1], the authors extend the DDP method to handle any type of nonlinear constraints on state and controls of the form

$$g_{i,k}(x,u) \leq 0$$

where the index i denotes the index of the constraint and k is the k^{th} time step. The main idea here is to modify the approximation of the value function in the backward pass to take into account constraints that are already active.

Backward pass

In CDDP, the backward pass is globally similar to DDP. The main difference is that instead of solving the unconstrained convex optimization problem (1), we solve the following constrained optimization problem:

$$\min_{\delta u} \tilde{Q}_k(\delta x, \delta u)$$

$$\text{subject to} \quad g_k(x_k + \delta x, u_k + \delta u) \leq 0$$

However this does not result in an invertible feedback between δu and δx anymore. To counter that, we only consider the set of active constraints and linearize them with respect to δx and δu, which allows us to transform the inequality constraints into an equality constraint

$$C_k \delta u = D_k \delta x$$

Where C_k and $-D_k$ are the gradients of the active constraints functions with respect to u and k respectively. We can then express δu as a linear function of δx and find the new approximated value function at time step k.

Forward Pass

The approximated value function from the backward pass takes into account estimated active constraints but cannot guarantee in general that the resulting optimized trajectory given by the linear feedback will actually be feasible. We thus need to solve again a QP problem with all of the inequality constraints considered.

Regularization

In [1], the authors note that some time steps might result in infeasible trajectories. Therefore, it becomes necessary to add regularization parameters. Two parameters define how close to the nominal trajectory the updated trajectory should stay, and regular the control trajectory should be. After a successful forward pass, regularization parameters are decreased to allow for the following iteration to search for an optimized trajectory further away from the current one. On the opposite, regularization parameters are increased if the forward pass has failed because of an infeasible QP that violated the constraints.

IV. Results

We implemented the Constrained DDP algorithm described in [1] on our two problems of interest. We used MATLAB and CVX to solve the QP problems. All tests were run on a laptop computer with a 2.7 GHz Intel Core i5 processor and 8GB of RAM. The C-DDP method was compared to A*, for which the map was discretized into a grid.

Single Objective Problem

C-DDP was used to find the shortest path between $[1,1]$ and $[25,25]$, given the initial sub-optimal path shown in blue in Fig.1. The evolution of the trajectory after each iteration is shown below in Fig.3.

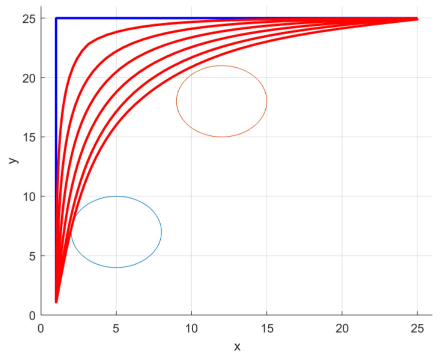
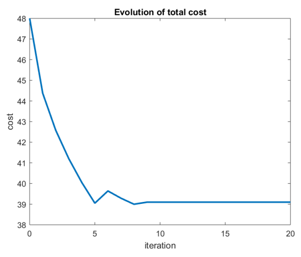
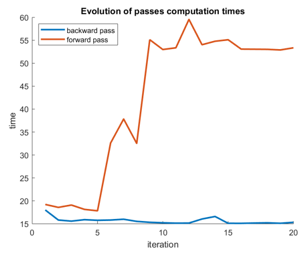


Fig. 3: Evolution of the nominal trajectory

We see that the nominal trajectory gradually evolves to minimize the total length and passes as close as possible to the obstacles.



(a) Total cost



(b) Computation time of backward and forward passes

Fig. 4: Evolution of the total cost and of the computation times with iterations

We see in Fig.4(a) above that the cost drops with the number of iterations before reaching a minimum after 5 iterations or so. After the 5^{th} iteration, the forward pass fails and the interval of confidence around the nominal trajectory keeps being reduced but the algorithm cannot find a better trajectory that satisfies the constraints.

This can also be seen on the evolution of the computation times of the backward and forward passes in Fig.4(b) above. We note that the computation time of the forward pass dramatically increases after the 5^{th} iteration, because CVX struggles to try to find a feasible solution to the QP's. We also note that each iteration is rather slow, with an average of 40s while the cost keeps decreasing, and up to a minute when we get closer to the obstacles. Stopping the algorithm once the cost stops decreasing would result in an overall computation time of about 4 minutes.

Multi Objective Problem

Below are the same plots for the multi-objective version of our problem. We observe very similar patterns, with the path gradually converging down the slope to minimize the elevation, and stopping when it reaches the obstacles. Here again the computation time of the forward passes dramatically increases once a feasible trajectory is not found. Stopping after the third iteration would in that case yield a total computation time of 2.5 min.

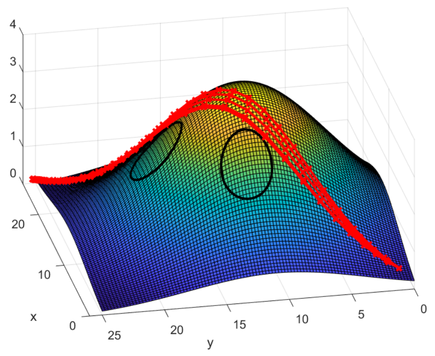
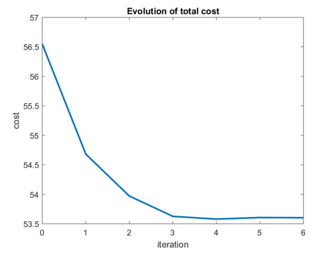
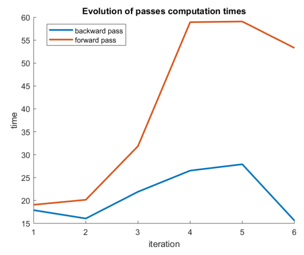


Fig. 5: Evolution of the nominal trajectory.



(a) Total cost



(b) Computation time of backward and forward passes

Fig. 6: Evolution of the total cost and of the computation times with iterations

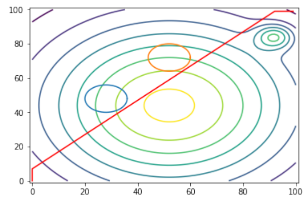
V. Analysis and Conclusion

As we can see in the results above, Constrained DDP suffers from several drawbacks when it comes to solving an optimal path problem.

First, the method requires the initialization of a feasible sub-optimal trajectory. This could be done quickly by using RRT for instance, which would yield a sub-optimal trajectory, but it would still be a handicap compared to methods that do not require initialization. It also requires a lot of parameter tuning in order not to get stuck in sub-optimal trajectories when adapting the confidence interval around the nominal trajectory.

Second, C-DDP is only able to find a local optimum and is thus sensible to the initial path. Once it has reached one of the obstacles, the algorithm stalls and is unable to "jump" over them, even though a trajectory with lower cost could be found beyond the obstacles. This is clear in our second test where going further down the slope on the other side of the obstacles would result in a much lower cost due to elevation, with a path of relatively similar length.

Finally, the main drawback of this method is that it is excessively computationally expensive compared to standard methods for path planning like A*. Below is a similar 3D problem solved using A* for various weights given to the elevation cost with respect to the length of the path. We see that in every case, A* finds the globally optimal path, "jumping" beyond the obstacles when that can result in a smaller cost. More importantly, A* finds the optimal solution in only 15s on average, which makes it far more efficient than CDDP for that particular problem. Finally, it doesn't require any initialization and there is no parameter to tune.


(a) $\alpha = 1.12$

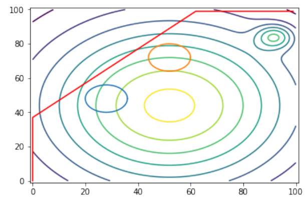

(b) $\alpha = 1.15$

Fig. 7: Optimal path found by A* for a running cost of $||u|| + \alpha.elev(x)$

As a conclusion, we can argue that Constrained DDP is a very heavy tool not well suited for solving simple path finding problems like ours. It is computationally expensive and requires some initial guess as well as extensive tuning. For regular path planning where the running cost can be defined as a function of the state only, A* appears to be a more robust and much faster algorithm, that is sure to converge to a global optimum.

However, constrained-DDP is an interesting improvement to standard DDP, as it succeeds in incorporating nonlinear constraints. It would potentially be a lot more interesting to solve off-line, motion planning problems in higher dimensions and/or with more complex dynamics, as well as costs that are direct functions of the controls. In all of those cases, combinatorial algorithms like A* would fail and the C-DDP approach would be more interesting.

References

[1] Z. Xie, C. K. Liu and K. Hauser, "Differential dynamic programming with nonlinear constraints," 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 695-702.

[2] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968.