Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

A new Markov–Dubins hybrid solver with learned decision trees

Cristian Consonni [a], Martin Brugnara [b], Paolo Bevilacqua [b], Anna Tagliaferri [c,d], Marco Frego [c,*]

[a] *Eurecat, Centre Tecnològic de Catalunya, Unit of Big Data and Data Science, Barcelona, 08005, Spain*
[b] *University of Trento, Department of Information Engineering and Computer Science, Trento, 38123, Italy*
[c] *Free University of Bozen-Bolzano, Faculty of Engineering, Bozen-Bolzano, 39100, Italy*
[d] *Free University of Bozen-Bolzano, Faculty of Agricultural, Environmental and Food Sciences, Bozen-Bolzano, 39100, Italy*

ARTICLE INFO

Keywords:
Markov–Dubins
Machine learning
CatBoost
Shortest path
Non-holonomic planning
Path planning

ABSTRACT

In this paper, the applicability of machine learning models and techniques to the Markov–Dubins path planning problem have been explored. Machine learning techniques are already applied to several fields, which range from computer vision, to physics simulation, to item recommendation, to user profiling. This pervasiveness has led to marked improvements in the implementation and support for applying machine learning models, in particular for specialised use cases such as low-power devices, embedded hardware, and real-time applications. On the other hand, the Markov–Dubins path planning problem, which is central in robotic nonholonomic trajectory design, is already covered by established numerical and optimisation techniques. However, the benefits of applying machine learning approaches to this problem remain to be investigated. In particular, there is the need to research potential speed-ups or application domains that would be better solved by a machine learning approach compared to the traditional algorithmic approaches. In this study, we train a state-of-the-art machine learning model in a supervised setting on Markov–Dubins and use it in two different ways: to directly predict the solution, and to filter candidate solutions. Also, a comparison of the quality of these predictions with a state-of-the-art Markov–Dubins solver is made. The results obtained indicate that machine learning approaches are comparable to state-of-the-art solutions: our bare model, directly predicting the solution, appears to be 8.3 times faster than the current standard, sacrificing the accuracy, which amounts to a value close to 92%; the hybrid model that filters the solutions prior to finding the best candidate runs in times that are comparable to the classical solver (58 ms) and has over 98% accuracy. A further comparison with alternative solvers and techniques, such as Optimal Control, NonLinear Programming and Mixed Integer NonLinear Programming has been made, confirming the benefits of the machine learning approach over these, for which the computational times are in the range of seconds. This opens new avenues for interdisciplinary applications of machine learning to more general planning problems (e.g., the same problem in 3D), where the number of possible manoeuvres is large and the computation of each of them requires a considerable computational effort, which makes the brute force trial-and-error infeasible.

1. Introduction

The generation of paths with nonholonomic constraints is a central task in most applied sciences, such as mathematical modelling, robotics, additive manufacturing for biotechnology and industry, navigation and other related fields, (Laumond et al., 1994; Sharma et al., 2021). A feasible and classic approach for the design of trajectories (path and speed profile) is the employment of 2D Markov–Dubins paths (Markov, 1887; Dubins, 1957) or biarcs (Bolton, 1975; Sabin, 1976): both contain line segments and circle arcs as elements of the path. These elements are called curve primitives. Geometrically, they solve the C^1 Hermite interpolation problem, that is, they produce a sequence of lines and circles that connect an initial to a final posture with continuity up to the first derivative. By posture, it is intended a point with an orientation, which can be described by the triple $\boldsymbol{p} = (x, y, \vartheta)$. Biarcs solve the problem of joining $\boldsymbol{p}_i$ with $\boldsymbol{p}_f$ using two circle arcs (that may degenerate to a line segment), whereas a Markov–Dubins path is composed generally of a sequence of three elements (line segments and circle arcs) with a limited turning radius. A precise description of Markov–Dubins paths is presented in the next section.

Among their many good properties, lines and circles are particularly useful mainly for the parametrisation with arc length and because the offset of these curve primitives is the same primitive (e.g., the offset of a circle is a circle), also, their description is straightforward. These properties do not hold for more complex primitives, for instance,

* Corresponding author.
E-mail addresses: cristian.consonni@acm.org (C. Consonni), me@martinbrugnara.it (M. Brugnara), paolo.bevilacqua@unitn.it (P. Bevilacqua), atagliaferri@unibz.it (A. Tagliaferri), marco.frego@unibz.it (M. Frego).

https://doi.org/10.1016/j.engappai.2023.106166
Received 5 November 2021; Received in revised form 12 February 2023; Accepted 15 March 2023
Available online xxxx

polynomials or other transcendent curves that often cannot be expressed by elementary functions and do not allow easy integration in industrial tools, e.g., in Computer-Aided Design (CAD). Despite their simple formulation, lines and circle arcs are effectively implemented in a plethora of applications.

The first traditional field where Markov–Dubins curves are employed is in mobile robotics since a Markov–Dubins path is the shortest path that connects two postures $\boldsymbol{p}_i$, $\boldsymbol{p}_f$ enforcing the limited manoeuvrability of the vehicle (e.g., limited steering angle and no lateral movements), (Boissonnat et al., 1994). An interesting case study is the Markov–Dubins Travelling Salesman Problem (DTSP), where the aim is to find the shortest closed path that connects a list of points, each of which has to be visited exactly once, while also considering the restraints of a robot, here modelled as a Markov–Dubins vehicle. This problem has been traditionally solved in two steps: first, the mission planning stage determines the order in which the points need to be visited, then, the path planning step predicts the trajectories that connect each pair of points Ny et al. (2012).

In manufacturing technologies, and more specifically in the context of Computer Numerical Control (CNC) milling, machines and tools are controlled by a standard language, the so-called g-code. This is produced by CAD software from a model prototype, whereby lines and circles are implemented to be the set of instructions that the machines have to follow.

Similar calculations have to be performed in additive manufacturing, such as 3D printing and bioprinting: in these cases, a 3D solid object first needs to be converted into a sequence of machine movements, which is usually encoded in g-code (Gulyas et al., 2018). While posing biomaterials, cells and bioactive molecules, the nozzle follows the nominal trajectory layer by layer, hence the path is locally planar and parallel curves (offsets) are often required, thus the aforementioned property of expressing the offset with the same primitive of the nominal path is important.

In GPS navigation, a solution to optimal coverage path planning, for example, for agricultural machines in crop fields is based on Markov–Dubins curves, (Hameed, 2017).

More recently, a novel application of Markov–Dubins curves has been discovered: steerable needles have been developed to improve drug delivery and surgery in the medical field, to be able to target positions that are difficult to reach with traditional methods, (Webster III et al., 2006). Such needles can be approximated as having constant radius of curvature and controlled by applying forces at the base. In two dimensions, this is equivalent to using Markov–Dubins paths, (Duindam et al., 2010).

Although most of the relevant features, properties and algorithms about biarcs (Bolton, 1975; Sabin, 1976; Bertolazzi et al., 2020) have been solved, often in elegant closed form (Bertolazzi and Frego, 2019), thus making them particularly suitable for real-time applications (e.g., feeding tools with an update every 20 ms), (Piegl and Tiller, 2002b,a; Park, 2004), biarcs, do not consider the limitation of the curvature nor the length of the resulting path. Bounded curvature is important because it models a physical limitation of the tool: the turning radius cannot be arbitrarily small. This limitation models the fact that a physical vehicle cannot steer the wheels more than a fixed angle. These issues have been considered and solved with Markov–Dubins paths (Shkel and Lumelsky, 2001; Kaya, 2017, 2019; Bevilacqua et al., 2020; Frego et al., 2020; Saccon et al., 2021), which are introduced in the next section. On the counterpart, these improvements come at the price of losing the straightforward algorithms of biarcs because the problem shifts from having a closed-form solution to a nonlinear constrained minimisation. In practice, solutions to the Markov–Dubins problem are known via trial-and-error of feasible solutions (Shkel and Lumelsky, 2001) or by solving (small) instances of a Mixed Integer Nonlinear Programming (MINLP), or its relaxed version, Nonlinear Programming (NLP), (Bevilacqua et al., 2020).

1.1. Paper contributions

In this paper, the newest findings on Markov–Dubins curves are combined with a supervised learning approach, which has been developed in the context of categorisation problems of Machine Learning (ML). A new approach to solve the Markov–Dubins path is therefore proposed: the problem is tackled by exploiting Artificial Intelligence (AI) by means of ML.

The Markov–Dubins is formulated as a categorisation problem, where the inputs of the machine learning system are the input parameters, which serve as features, and the output is a categorisation that translates into one possible manoeuvre. The results can be readily used inside existing machinery, as a manoeuvre is still a sequence of line segments and circle arcs.

Furthermore, a state-of-the-art machine learning technique is employed, thereby leveraging the recent developments in the field; more precisely an algorithm called *CatBoost* is applied, which exploits gradient boosting to learn a set of decision trees over the input space. Then, it is shown how the trained model can be applied to build a software that solves the original Markov–Dubins problem. A graphical scheme of the overall procedure is sketched in Fig. 1.

The planar Markov–Dubins problem is at the same time difficult and easy: it is hard because a smart solution is not available and the best method is a brute force approach; it is easy because the size of the problem is small, thus the cases to be tested are limited. Nevertheless, especially in the last years, the research community tried to find alternative solution methods (Kaya, 2017, 2019; Bevilacqua et al., 2020; Frego et al., 2020; Faigl et al., 2020; Váňa and Faigl, 2020). The ML approach proposed in the present study is to understand how much AI can be useful in such a planning context.

We choose the planar Markov–Dubins problem, not because it is easy, but because it is hard, because that goal will serve to organise and measure what can be expected in more involved methods or in the 3D case.

This is motivated by the growing interest in extending the problem to 3D, in which case very little is known about the properties of the solution, but the brute force approach will become infeasible. Indeed, in 3D, the solution is supposed to be composed of 3 segments (as in the planar case) containing lines, circles and helices. This time, closed form solutions are not available, and a full numerical solution is sought. The problem becomes now the complexity, since every segment can be a line, left/right circle, up/down/left/right helix and each instance requires an optimisation step, which is costly. Since for the planar case complete knowledge is available, we selected the 2D problem in order to have an exact answer to be compared with the ML method. This would not have been possible in the 3D case, but our study will pave the way for a non brute force approach to it.

Another way to introduce the contributions of this paper is as follows: we learn a compressed representation of the function that solves the Dubins–Markov problem, i.e., which is the best manoeuvre for a given input represented by the triple $(\vartheta_i, \vartheta_f, \kappa)$ as in Fig. 1. Our research questions are the following: How good (lossy) is this representation? How does it change when using different machine learning models and tuning their parameters? How good is the decompression? How fast? Our work provides some answers to these questions.

The paper has the following structure: Section 2 reviews the state-of-the-art both on Markov–Dubins paths and supervised machine learning techniques. Section 3 provides the background necessary to formulate the problem. In Section 4 there is a detailed description of the machine learning technique and implementation of the prediction system using CatBoost. Results are presented and discussed in Section 5. Section 6 concludes the paper and offers new research directions.

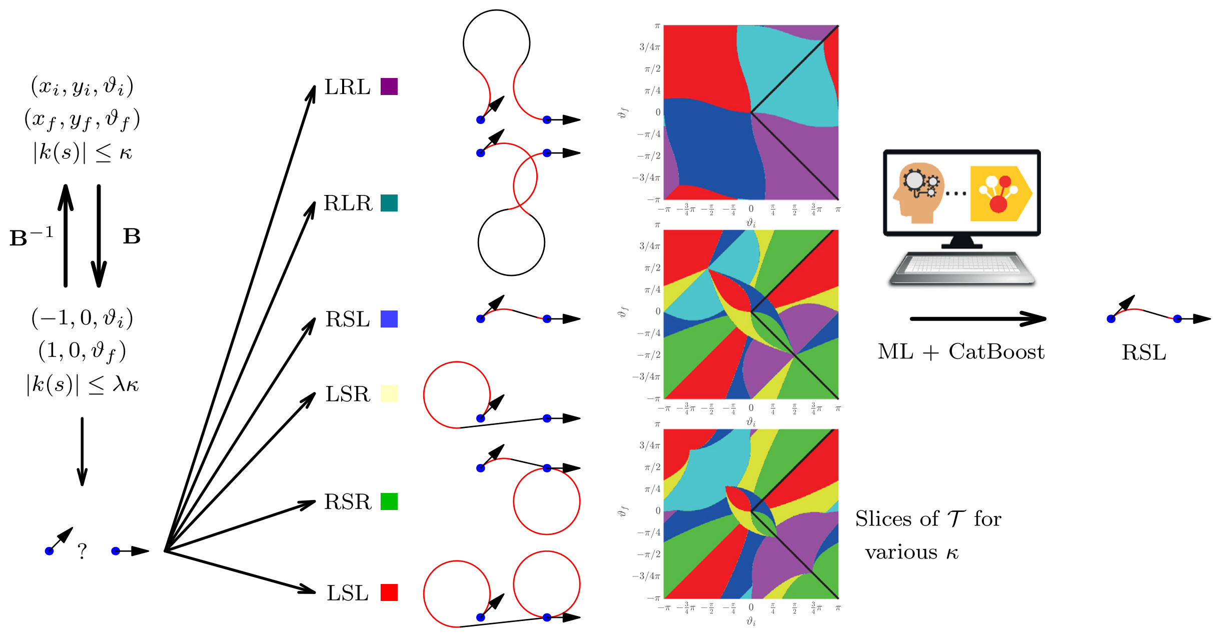


Fig. 1. Overview of the proposed solution method to the Markov–Dubins problem: starting with the 7 parameters of the general problem formulation, we reduce them to 3 thanks to the conformal map B, then, instead of testing by brute force trial-and-error the six possible manoeuvres, with a Machine Learning (ML) approach based on Catboost we learn the triangular prism $\mathcal{T}$. The coloured squares in the picture represent the solution of the Markov-Dubin problem for some slices of $\mathcal{T}$. We exploit the symmetries of our formulation to reduce the squares to the highlighted (solid black line) triangles, for several values of the maximum allowed curvature κ.

2. State-of-the-art

The first published work posing the problem of connecting two given postures with a path of minimum length, given a constraint on the minimum turning radius, dates back to 1889, when Markov proposed several variants of the problem related to the design of railways (Markov, 1887). After nearly 70 years, in 1957, the first, general solution to the problem was found and published by Dubins (1957). His work, based on geometric arguments, showed how the optimal solution can be constructed by concatenating at most three circle arcs and straight lines, as explained in Section 3. A different approach to the solution of the Markov–Dubins problem, based on optimal control theory and on the "minimum principle of Pontryagin", was proposed in the early 1990s by Boissonnat et al. (1994), Boissonnat and Bui (1994) and, independently, by Sussmann and Tang (1991); an optimal control solution using interval arithmetic and Chebyshev polynomials is presented in Razmjooy et al. (2016, 2019b,a). An extensive study on the nature of the solutions to the problem, and on the influence of the different parameters on the type and length of the optimal paths, was published in 2001 by Shkel and Lumelsky (2001). More recently, in 2017, Kaya conducted an in-depth analysis on the application of optimal control theory for the solution of the Markov–Dubins problem (Kaya, 2017). The author considered also some special conditions, yielding abnormal control solutions based on the concatenation of at most two circular arcs that were not taken into account by the previous literature. Moreover, the paper presented a numerical approach based on Nonlinear Programming to determine the optimal solution to the problem. A simpler modelling of the problem, based on the adoption of a single equation covering all the possible cases, was presented in 2020 by Bevilacqua et al. (2020). Since circle arcs and line segments are described using the same function, it is possible to smoothly blend from one type of curve to the other. Thus, the problem can be expressed via Mixed Integer Nonlinear Programming, or also via its relaxation to Nonlinear Programming. Moreover, the paper proposed a conformal bipolar transform of the problem to a standard form that better captures the inherent symmetries; this has been exploited to reduce the size of the space to be learned, as detailed in the next sections.

3. Background

3.1. The Markov–Dubins problem

The Markov–Dubins problem requires finding the shortest path that connects an initial posture $\boldsymbol{p}_i = (x_i, y_i, \vartheta_i)$ with a final posture $\boldsymbol{p}_f = (x_f, y_f, \vartheta_f)$ such that the curvature of the path, in absolute value, is not greater than a prescribed maximum value $\kappa > 0$, see Fig. 2. More formally, the Markov–Dubins path is the solution of a C^1 Hermite interpolation problem with a bound on the curvature. Let $\mathcal{D}(s) = (x(s), y(s))$ be a C^1 and piecewise C^2 parametric curve. Then $\mathcal{D}(s)$ solves the Markov–Dubins problem if $\mathcal{D}(s)$ has minimum length $L > 0$ and is subject to:

$$\begin{aligned} \mathcal{D}(0) &= (x_i, y_i), & \mathcal{D}(L) &= (x_f, y_f) \\ \mathcal{D}'(0) &= (\cos\vartheta_i, \sin\vartheta_i), & \mathcal{D}'(L) &= (\cos\vartheta_f, \sin\vartheta_f), \\ |k(s)| &\le \kappa, & \|\gamma'(s)\| &= 1, \quad s \in [0, L], \end{aligned} \tag{1}$$

where the (signed) curvature $k(s)$ can be written as

$$k(s) = \frac{x'(s)y''(s) - y'(s)x''(s)}{\sqrt{x'(s)^2 + y'(s)^2}}. \tag{2}$$

Stated like this, problem (1) is characterised by 7 parameters, namely the initial and final configurations (x_i, y_i, ϑ_i), (x_f, y_f, ϑ_f) and the curvature (2) is constrained such that the maximum absolute allowed value is κ. Without loss of generality, this number can be reduced to 3 if a certain transformation of coordinates is used. There are three maps proposed in literature to recast a general problem into a standard setting, the one in Boissonnat et al. (1994), the one in Shkel and Lumelsky (2001) and the recent one proposed in Bevilacqua et al. (2020). The present work is based on the latter, which is here briefly sketched. It is preferred over the others because it allows a topology with better properties such as two intuitive axes of symmetry that reduce the size of the parameter space.

Bipolar conformal transform. Let map $\boldsymbol{B} : \mathbb{R}^2 \mapsto \mathbb{R}^2$ be the composition of a translation, a rotation and a change of scale:

$$\boldsymbol{B}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \frac{1}{\lambda}\left(\begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix}\right), \tag{3}$$

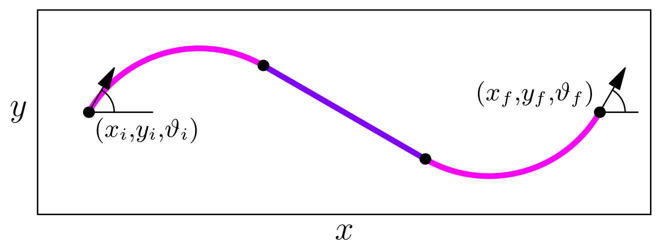


Fig. 2. The standard Markov–Dubins interpolation scheme between two postures (x_i, y_i, ϑ_i) and (x_f, y_f, ϑ_f). This solution is an example of Right-Segment-Left manoeuvre (RSL).

where the following auxiliary parameters are introduced:

$$\begin{aligned} \Delta x &= x_f - x_i, & \lambda &= \tfrac{1}{2}\sqrt{\Delta x^2 + \Delta y^2}, & \hat{\vartheta}_i &= \vartheta_i - \varphi, \\ \Delta y &= y_f - y_i & \varphi &= \mathrm{atan2}(\Delta y, \Delta x), & \hat{\vartheta}_f &= \vartheta_f - \varphi, \end{aligned} \tag{4}$$

and $\hat{\kappa} = \lambda\kappa$, so that φ is the unique angle in $[-\pi, \pi)$ that satisfies $\bar{x} = x_i \cos\varphi + y_i \sin\varphi + \lambda$, and $\bar{y} = -x_i \sin\varphi + y_i \cos\varphi$, with $\bar{x}$, $\bar{y}$ the translation along the axes. With the auxiliary parameters (4), the map $\boldsymbol{B}$ in (3) shifts the initial point (x_i, y_i) to $(-1, 0)$, rotates the axes so that the final point lies on the horizontal axis, applies a scaling so that the final point is mapped to $(1, 0)$. Being a conformal map, the angles ϑ_i, ϑ_f are preserved, but the curvature is scaled. In conclusion, with $\boldsymbol{B}$, the problem is reduced to canonical form and depends on ϑ_i, ϑ_f and κ only, instead of the 7 parameters $\boldsymbol{p}_i$, $\boldsymbol{p}_f$ and κ.

Characterisation of the solution. Dubins theorem (Dubins, 1957) states that $\mathcal{D}$ is composed of at most three pieces of curve: a combination of line segments and circle arcs, which are commonly identified with words, like RSL, where the letters stand for Right turn, (line) Segment and Left turn, respectively (see Fig. 2 for an example). Therefore, there are no more than 15 possible combinations, namely: LSL, LSR, RSL, RSR, RLR, LRL, LS, RS, SL, SR, S, LR, RL, L, R. In practice, only the first six words are considered, the others can be obtained by setting to zero some of their arcs. What makes it difficult to get the solution is the high nonlinearity of the problem: it is not known, given an input, how to determine which of the six manoeuvres is the optimal one.

The problem is therefore tackled with several techniques: via trial-and-error of the six candidate solutions computed with trigonometric formulas (Boissonnat et al., 1994; Shkel and Lumelsky, 2001), via optimal control (OCP) (Kaya, 2017, 2019), via NonLinear Programming (NLP) or Mixed Integer NonLinear Programming (MINLP) (Bevilacqua et al., 2020; Frego et al., 2020). Each technique has several pros and cons, for the purpose of the present work any of them is effective, indeed, the trial-and-error is selected because is the fastest one.

The parameter space for training. The bipolar conformal map $\boldsymbol{B}$ described above has the advantage of reducing the dimension of the parameter space from 7 to 3, moreover, this reduced space has two axes of symmetry that can be exploited to further reduce the size of the training set given to the Machine Learning algorithm. The parameters of the reduced space are the two angles ϑ_i, ϑ_f and the value of the maximum curvature $\kappa > 0$, hence the reduced space is $\wp = [-\pi, \pi) \times [-\pi, \pi) \times \mathbb{R}^+$. Geometrically, this is an infinite prism with square basis: any cutting plane of constant curvature produces a square $[-\pi, \pi) \times [-\pi, \pi)$ as the ones shown in Fig. 3.

The double symmetry of the solution space is apparent in Fig. 3, indeed the proof can be found in Bevilacqua et al. (2020). Keeping the curvature κ fixed, the axes of symmetry are the diagonals $\vartheta_f = \pm\vartheta_i$. The region $|\vartheta_f| \le \vartheta_i$, for $\vartheta_i \in [0, \pi)$ is informally identified as the triangular domain, to distinguish it from the complete square $[-\pi, \pi)\times[-\pi, \pi)$. This implies that the learning can be performed on the highlighted infinite triangular prism $\mathcal{T} = \{(\vartheta_i, \vartheta_f, \kappa) \in [-\pi, \pi) \times [-\pi, \pi) \times \mathbb{R}^+ \text{ s.t. } |\vartheta_f| < \vartheta_i\}$.

As a consequence, slicing this prism with planes of constant curvature requires sampling points of each triangle and not of the full square, thus needing only $1/4$ of the samples. Also, for high values of curvature, e.g., $\kappa > 6$, these regions do not change very much and only four (the circle-line-circle sequences) of the six candidate solutions are feasible, which can be intuitively explained considering that with enough curvature, the solution with three circles is not optimal, for more details see Boissonnat et al. (1994), Shkel and Lumelsky (2001), Kaya (2017, 2019). An important feature of our conformal map $\boldsymbol{B}$, compared to other approaches (e.g. Soueres and Laumond (1996)), is that the axes of symmetry are always the two diagonals of the square, corresponding to $\vartheta_f = \pm\vartheta_i$.

Finally, it is possible to formalise the parameter space for the training of the model. The set of input vectors $\boldsymbol{x} = (\vartheta_i, \vartheta_f, \kappa)$ is sampled from the infinite triangular prism $\mathcal{T}$ cut at different values of constant curvatures. The curvature κ is discretised in the interval $(0, 8]$. For each of the generated input vector $\boldsymbol{x}$, the optimal solution $y \in$ {LSL, LSR, RSL, RSR, RLR, LRL} labelled from 1 to 6 is computed, in order to obtain a supervised model.

3.2. Available solvers

3.2.1. The standard Markov–Dubins solver

There are different solvers available to solve the Markov–Dubins problem, among them, the one selected to produce the datasets and test the model is our implementation of the trial-and-error of the six candidate solutions with the formulas presented in Shkel and Lumelsky (2001). This is called the "standard" solver and its performance is discussed later in comparison with the novel ML solutions.

3.2.2. OCP-based solution

Another way of solving the problem is based on Optimal Control, where the total length of the path L is minimised, subject to a set of differential equations that model the system, boundary conditions and a bounded control (the curvature $k(s)$). The OCP, adapted to our notation from Kaya (2017), reads:

$$\begin{aligned} \min L &= \int_0^L 1 \,\mathrm{d}s \ \text{ subject to:} \\ x'(s) &= \cos(\theta(s)), \quad x(0) = x_i,\ x(L) = x_f, \\ y'(s) &= \sin(\theta(s)), \quad y(0) = y_i,\ y(L) = y_f, \\ \theta'(s) &= k(s), \quad \theta(0) = \vartheta_i,\ \theta(L) = \vartheta_f, \\ |k(s)| &\le \kappa \ \text{ for } s \in [0, L]. \end{aligned} \tag{5}$$

The OCP is stated in standard form and can be solved with off-the-shelf software.

3.2.3. OCP transcription to a NLP-based solution

The discretisation of a continuous OCP such as (5), leads to a NonLinear Programming (NLP). The formulation in Kaya (2017) glues together 5 arcs in the form LRSLR, which encompasses all possible line and circles combinations and solves for the optimality conditions. At an optimal solution, at least two of the five segments will be forced to have zero length: the ones that do not go to zero give the characterisation of the solution. The problem is formulated as:

$$\begin{aligned} &\min L = \textstyle\sum_{j=1}^{n} \sum_{i=1}^{5} s_{j,i} \quad \text{subject to} \\ x_j &= x_{j-1} + S_j(\theta_{j,0}, \dots, \theta_{j,5})/\kappa + s_{j,3} \cos\theta_{j,2}, \\ y_j &= y_{j-1} + C_j(\theta_{j,0}, \dots, \theta_{j,5})/\kappa + s_{j,3} \sin\theta_{j,2}, \\ S_j &= -\sin\theta_{j,0} + 2\sin\theta_{j,1} - 2\sin\theta_{j,2} + 2\sin\theta_{j,4} - \sin\theta_{j,5}, \\ C_j &= \cos\theta_{j,0} - 2\cos\theta_{j,1} + 2\cos\theta_{j,2} - 2\cos\theta_{j,4} + \cos\theta_{j,5}, \\ \theta_{1,0} &= \vartheta_0, \quad \sin\theta_{n,5} = \sin\vartheta_n, \quad \cos\theta_{n,5} = \cos\vartheta_n, \end{aligned}$$

with the length of the sub-arcs $s_{j,i} \ge 0$, angle continuity $\theta_{j+1,0} = \theta_{j,5}$, and the angles between the elementary manoeuvres satisfying

$$\theta_{j,1} = \theta_{j,0} + \kappa s_{j,1}, \quad \theta_{j,2} = \theta_{j,1} - \kappa s_{j,2}, \tag{6}$$

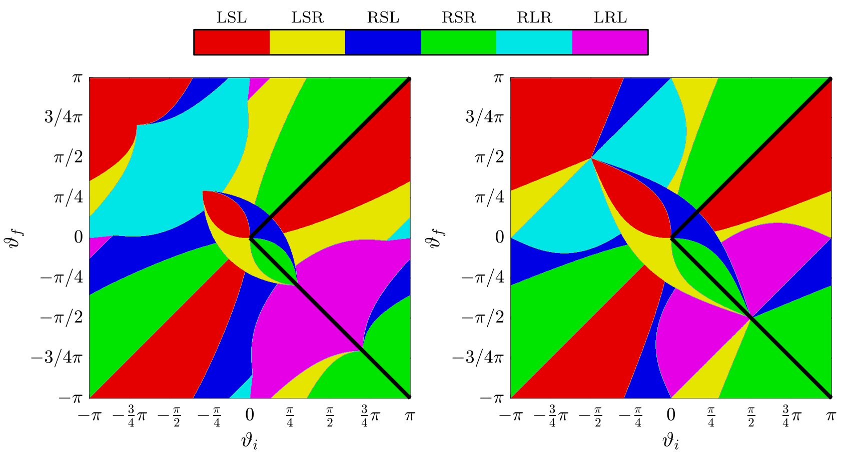


Fig. 3. Plot of the solution word of the Markov–Dubins problem for different values of κ, respectively $\kappa = 8/10$ and 1.

$$\theta_{j,4} = \theta_{j,2} + \kappa s_{j,4}, \quad \theta_{j,5} = \theta_{j,4} + \kappa s_{j,5}. \tag{7}$$

This NLP is also written in standard form, and can be solved with available software.

3.2.4. MINLP-based solution

A new method for solving the problem without requiring to know exactly which kind of curve appears in each of the three segments is to write a line and a circle with the same equation, and then dynamically change the defining parameters during the optimisation phase to obtain one curve or the other. The parametric equations of a line segment or of a circle arc, starting at the base point (x_0, y_0) with initial angle ϑ_0, with constant curvature κ (possibly zero) and length s, are given by

$$\begin{aligned} x(s) &= x_0 + s\,\text{sinc}\left(\frac{\kappa s}{2}\right)\cos\left(\vartheta_0 + \frac{\kappa s}{2}\right) \equiv x_0 + f(s,\kappa,\vartheta_0) \\ y(s) &= y_0 + s\,\text{sinc}\left(\frac{\kappa s}{2}\right)\sin\left(\vartheta_0 + \frac{\kappa s}{2}\right) \equiv y_0 + g(s,\kappa,\vartheta_0), \end{aligned} \tag{8}$$

where s is the arc length parameter (Bertolazzi and Frego, 2019). Adding three consecutive terms of the previous form (8) with opportune continuity conditions yields the system:

$$\begin{aligned} x_f &= f(s_0,\kappa\sigma_0,\vartheta_i) + f(s_1,\kappa\sigma_1,\vartheta_i + \kappa\sigma_0 s_0) \\ &\quad + f(s_2,\kappa\sigma_2,\vartheta_i + \kappa\sigma_0 s_0 + \kappa\sigma_1 s_1) + x_i, \\ y_f &= g(s_0,\kappa\sigma_0,\vartheta_i) + g(s_1,\kappa\sigma_1,\vartheta_i + \kappa\sigma_0 s_0) \\ &\quad + g(s_2,\kappa\sigma_2,\vartheta_i + \kappa\sigma_0 s_0 + \kappa\sigma_1 s_1) + y_i, \\ \vartheta_f &= \vartheta_i + \sigma_0\kappa s_0 + \sigma_1\kappa s_1 + \sigma_2\kappa s_2. \end{aligned} \tag{9}$$

The integer variables $\sigma_i \in \{-1,0,+1\}$ for $i = 0,1,2$ model a right turning circle, a line and a left turning circle, respectively. Therefore, the Mixed Integer NonLinear Programming becomes

$$\begin{aligned} &\text{minimise} \quad s_0 + s_1 + s_2 \quad \text{subject to (9)} \\ &\text{with} \quad \sigma_i \in \{-1,0,+1\} \subset \mathbb{Z},\ s_i \in \mathbb{R}^+,\ \vartheta_i \in \mathbb{R}, \end{aligned} \tag{10}$$

with assigned boundary conditions (for $L = s_0 + s_1 + s_2$):

$$\begin{aligned} x(0) &= x_i, \quad y(0) = y_i, \quad \vartheta(0) = \vartheta_i; \\ x(L) &= x_f, \quad y(L) = y_f, \quad \vartheta(L) = \vartheta_f. \end{aligned} \tag{11}$$

3.2.5. Relaxed MINLP-based solution

The MINLP (10) can be relaxed to the NLP given in (12), by allowing the σ_i variables to belong to the convexified domain $\sigma_i \in [-1,1]$. The optimisation problem becomes therefore:

$$\begin{aligned} &\text{minimise} \quad s_0 + s_1 + s_2 \quad \text{subject to (9)} \\ &\text{with} \quad \sigma_j \in [-1,1],\ s_i \in \mathbb{R}^+,\ i = 0,1,2, \end{aligned} \tag{12}$$

with the same boundary conditions (11).

Both problems (10) and (12) are stated with a standard formulation and can be solved with available optimisation software.

Remark 1. There are two main motivations for choosing the standard solver over the OCP, NLP, MINLP methods described above: firstly, the guarantee of convergence; secondly, the speed of the computation. The appeal of the other methods resides in the flexibility of extending the formulation of the problem with other characteristics. We point out that the standard solver is not suitable for the 3D case, where only optimisation-based methods are feasible, since no closed-form solution is available.

3.3. Machine learning

In a machine learning model, the objective consists in using data to model a problem of the form $y = f(\boldsymbol{x})$, where f is an unknown function that is approximated with $\hat{f}$. In other words, it is assumed that there exists an $\hat{f}$ that, for any arbitrary set of inputs $\boldsymbol{x}$, produces the desired answer $\hat{y}$. The input of the problem is generally multi-dimensional and is represented by an n-dimensional vector $\boldsymbol{x} \in \mathbb{R}^n$, while the output is 1-dimensional, $\hat{y} \in \mathbb{R}$.

The process in which the form and parameters of the function $\hat{f}$ are derived from the data is called the *training* of the model, while the comparison between possible values of the hyper-parameters (number of iterations, tree depth, learning rate, etc.) is called *validation*. Finally, the evaluation of the goodness of the inferred $\hat{f}$ with respect to the real f is usually called *testing*.

The characteristics and the form of the function to be learned, of the available input $\boldsymbol{x}$, and of the output y give rise to different kinds of machine learning problems. Following Goodfellow et al. (2016), Zhang et al. (2020), based on the availability of the target output, the machine learning problems are classified into: (i) *supervised learning*, when the output variables y, also known as *labels*, are known and can be used to define a *loss function*; (ii) in *unsupervised learning* the inputs are *unlabelled*.

Based on the form of the desired output, machine learning tasks can be described as *classification*: when the model needs to learn a mapping from a set of variables, called features, to a set of labels; *regression*: when the model predicts a continuous outcome variable; or *clustering* that seeks to identify a grouping of the examples given, such as the ones in the same group are more similar among each other than those belonging to other groups.

For the Markov–Dubins problem, the input variables are in principle 7, the initial and final postures $(\boldsymbol{p}_i, \boldsymbol{p}_f)$ and the maximum curvature κ, as shown in Section 3.1; it is possible to reduce them to 3, exploiting the double symmetry of the problem, taking advantage of the recent results presented in Bevilacqua et al. (2020).

The Markov–Dubins problem can be mapped to a multiclass classification problem in a supervised learning setting, where the input parameters can be used as features $\boldsymbol{x}$ to obtain the optimal manoeuvre $\hat{y}$.

3.3.1. *Supervised machine learning*

In the setting of supervised learning, a central concept is the *loss function*. Let $D = \{(\boldsymbol{x}^k, y^k)\}^{k=1\dots m}$ be a dataset of m samples, where $\boldsymbol{x}^k = (x_1^k, \dots, x_n^k)$ is a random vector of n features and each $y^k \in \mathbb{N}$ is a target. The samples in D can be assumed independent and identically distributed using the principle of maximum likelihood; it can be shown that the goal of a learning task is to train a function $\hat{f} : \mathbb{R}^n \mapsto \mathbb{N}$ which minimises the expected loss $\mathcal{L}(\hat{f}) := \mathbb{E}[L(\hat{y}, y)]$. L is a smooth loss function and $(\boldsymbol{x}, y)$ is a test example of the training set D:

$$L(\hat{y}, y) = \sum_{i=1}^{m} w_i \log\left(e^{a_{it_i}(\hat{y},y)} \Big/ \sum_{j=1}^{n} e^{a_{ij}(\hat{y},y)} \right) \Big/ \sum_{i=1}^{m} w_i. \tag{13}$$

A machine learning task aims at finding the set of parameters $\{w_i\}_1^m, a_{ij}$ such that the loss function (13) is minimised.[1]

3.3.2. *Decision trees*

Decision trees are classifiers that can work with both categorical and continuous input data. The basic idea consists in recursively dividing the feature space until the resulting regions all share a common output, y. The recursive *decision* process is thus modelled as a tree. Each node stands for a test on one of the input's features (attributes), and its children represent all the possible outcomes of the test. For categorical data, usually, each attribute value gets its nodes, whereas the domain of the continuous data is split by means of one or more thresholds. Each tree leaf represents a region where the expected answer for the learning task at hand is the same, from which the probability distribution over the output categories can be obtained.

3.3.3. *CatBoost*

We use a supervised categorical approach based on gradient boosting and decision trees using CatBoost, a state-of-the-art library for supervised categorical learning based on gradient boosting, which supports numerical and categorical features; see Prokhorenkova et al. (2019) and Hancock and Khoshgoftaar (2020) for a survey. A CatBoost model consists of a set of decision trees selected iteratively to minimise the collective expected loss. With reference to the loss function of Eq. (13), the model provides the values for a_{it_i} as `RawFormulaVal` outputs, thus it is possible to compute the probabilities for each class by computing the $\mathrm{softmax}$ function.[2]

4. A machine learning approach to the Markov–Dubins problem

The Markov–Dubins optimal path selection is a clear example of categorical inference problem. When it comes to ML, there are many methodologies and strategies that can be adopted to solve the problems. This Section is intended to describe the reasoning underlying the procedure that led to the results of Section 5.

Two approaches are proposed:

1. the "bare model", a model that directly outputs the best manoeuvre, whose results are discussed in Section 5.3.
2. the "hybrid model", a model that outputs a set of candidate manoeuvres, together with their probability of being the best manoeuvre. In this case, the most promising candidates can be then checked using a standard Markov–Dubins solver: the trial-and-error method is only carried out on selected cases with a probability above a certain threshold. The results obtained with this model are discussed in Section 5.4.

The aim of the first solution is to substitute the standard Markov–Dubins solver with a learned model, *i.e.*, given the inputs ϑ_i, ϑ_f and κ obtain the optimal manoeuvre. With this model, together with the prediction, a probability representing the confidence of the model about the output is obtained. In specific scenarios, when multiple manoeuvres are almost equally likely (such as along the region borders in Fig. 3), the model prediction heavily depends on the training dataset sampling resolution; in rare cases, a sub-optimal manoeuvre may be selected. To enhance the precision of the system without increasing the size of the learning dataset, another solution has been developed that exploits the information about the prediction's confidence, *i.e.*, a hybrid approach. In other words, the learned model is used to limit the number of candidates to evaluate with the standard Markov–Dubins solver and select the best.

Given the low dimensionality of the input and the fact that dense regions share the same solution, decision trees have been selected as models to train. CatBoost has thus been elected as a state-of-the-art ML library for categorical distribution learning with decision trees.

The training set is composed of 489 648 elements. For the validation and testing, two additional and independent datasets have been generated, by randomly generating samples within the triangular prism $\mathcal{T}$. These validation and test datasets contain 139 899 and 69 950 samples respectively, yielding the standard proportion of $7 : 2 : 1$ for training, validation and testing.

The model is characterised by several parameters[3]:

- ***Tree Depth***: the maximum depth for each decision tree;
- ***Learning Rate***: the percentage of the gradient that is back-propagated from one generation of the model to the next;
- ***Random Strength***: the amount of randomness to use for scoring splits when the tree structure is selected, this reduces the overfitting of the model;
- ***Number of Iterations***: the maximum number of trees that can be built when solving machine learning problems.

At the end of the training, the model gets pruned to the number of trees that show the best performance.[4] Models with less and shallower trees have a smaller memory footprint and are therefore faster to use. For this reason, being that the precision is equal, the simpler models have been selected.

Finally, the selected model has been exported in the binary dump format, in this way it has been possible to write a simple standalone program that loads the model and applies it to a set of inputs.

For the first solver, the model gets applied to return a prediction, which is the class with the maximum probability. For the second one, the model is used to obtain the probability for each manoeuvre, then those whose probability is higher than a pre-defined threshold are evaluated with the standard solver; finally, the actual best path is returned.

5. Results

5.1. *Hyper-parameters*

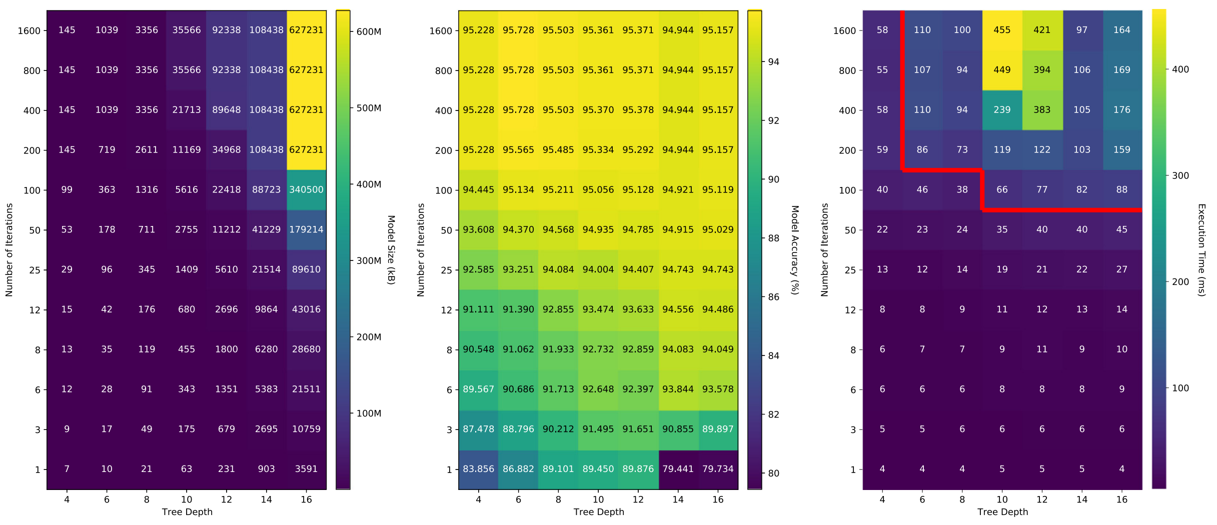
Several models were trained by varying the hyper-parameters over ranges reported in Table 1. A preliminary analysis showed that accuracy is inversely correlated with *Learning Rate* and directly correlated with *Number of Iterations* and *Tree Depth*, while *Random Strength* is uncorrelated. Moreover, the accuracy reaches a plateau for values of *Learning Rate* lower than 0.70. Thus the analysis focused on setting *Learning Rate* to 0.70 and *Random Strength* to 0.15.

[1] https://catboost.ai/docs/concepts/loss-functions-multiclassification.html

[2] https://gist.github.com/CristianCantoro/f850adf7999dc6224f189017b6b6e433

[3] For a more detailed definition, refer to CatBoost's documentation: https://catboost.ai/docs/concepts/parameter-tuning.html.

[4] Option `use_best_model=True`.



(a) Size (kB) (b) Accuracy (%) (c) Execution Time (ms)

Fig. 4. Heatmaps of the bare models versus the *Number of Iterations* and *Tree Depth*.
In (c) the solid red line corresponds to the execution time of the classical Markov–Dubins solver ($t = 58$ ms).

Table 1
Range of the hyper-parameters used in the training.

Tree Depth	4, 6, 8, 10, 12, 14, 16
No. of Iter.	1, 3, 6, 8, 12, 25, 50, 100, 200, 400, 800, 1600
Learn. Rate	0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90
Rnd. Strength	0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35

Table 2
Error on the optimal length L and timing with different OCP softwares.

OCP solver	Error	Time (ms)
PINS/XOptima	10^{-7}	357
Acado	10^{-2}	895
Gpops	10^{-7}	756
ICLOCS	10^{-5}	689

Table 3
Top-5 bare models by accuracy, ties are ordered by size.

#	No. of Iter.	Tree-Depth	Execution Time (ms)	Training Time (s)	Size (kB)	Accuracy (%)
1	400	6	117	47	1063	**95.728**
1	800	6	121	94	1063	95.728
1	1600	6	121	190	1063	95.728
4	200	6	97	24	736	95.565
5	1600	8	107	297	3437	95.502

5.2. Comparisons with other solution methods

In this section we compare and comment the pros and cons of the results obtained with other available techniques in literature.

5.2.1. Standard solver performance

As introduced in Section 3.2.1, the standard solver is a trial-and-error technique that enumerates all solutions and selects the best one. Therefore, it has 100% accuracy and is capable of solving the testset of the problem in $T_{\text{std}} = 58$ ms.

5.2.2. OCP solution performance

Solving a single OCP requires considerably more time (about half a second) than the use of the standard solver, which computes the whole dataset in 58 ms. We tested therefore the performance of the OCP on a single problem instance, with results averaged over multiple repetitions. Several softwares have been considered, the results are collected in Table 2. The results show that an OCP is by far slower than the standard solver, the accuracy is also not always very precise, compared to the exact solution. Another important comment is that the solution is highly dependant on the initial guess.

5.2.3. MINLP solution

A MINLP is also a hard problem to be solved in little time, we employed Knitro to solve a subset of the dataset. The computational time required to solve the subset of the dataset is large and an instance is on average comparable to the time required to solve an OCP, so less than one second. The MINLP was able to solve correctly about 85% of the cases, with good accuracy, i.e., an error smaller that E-06. These results are in agreement with Bevilacqua et al. (2020).

5.2.4. NLP solutions

We tested the solver described in Section 3.2.5 (NLP from Relaxed MINLP) and compared it with the results of Kaya (2017) (NLP from OCP discretisation) of Section 3.2.3. For the NLP obtained relaxing the MINLP (using Knitro), we obtained 100% accuracy with an error which was always smaller than E-13 (up to E-16 - machine precision), but computational times that were in the order of the second for each instance of the problem. The OCP discretisation was successfully used in Kaya (2017) using the solver Knitro and with an equivalent accuracy, the computational times were not reported, but can be expected to be in the same order of magnitude.

5.3. Bare-model solution

In this section we present the results obtained with the ML method proposed. Table 3 presents the top-5 ML-only models in terms of accuracy. All the experiments were executed on a machine equipped with an Intel® Core™ i7-4790@3.60 GHz CPU and 32 GB of DDR3@1600MHz RAM running the Linux kernel version 4.15.0-142-generic. The best models were obtained with *Tree Depth* equal to 6 and top off with *Number of Iterations* 400 at 95.728% of accuracy. The table also shows the time elapsed for training the models and the size in megabytes of the resulting trained models, when exported using the native CatBoost binary format `.cbm`. Both the training time and the size depend heavily on *Number of Iterations* and *Tree Depth*.

Fig. 4 shows the size 4(a) of the model, its accuracy 4(b) and execution time 4(c) for several different settings for *Number of Iterations*

Table 4
Accuracy on the test set for other methods in bare mode.

Family	Method	Accuracy (%)
Random Forest	Boosted Trees	91.76
	RUS boosted	84.94
	CatBoost	95.73
SVM	Linear	84.49
	Quadratic	83.89
	Cubic	78.51
	Gaussian	96.03
	Look-up Table	94.12

and *Tree Depth*. For the bare solver, the running time is $T_{\text{bare}} = 117$ ms, a size of 1 megabyte, and a training time of 47 s.

We conclude the section presenting the learning results of other classic ML methods, namely Random Forest and Support Vector Machine (SVM). The results are collected in Table 4, from which we can see that some of them perform worse than CatBoost, the Gaussian SVM is more or less equivalent to CatBoost, that is therefore kept as the baseline. As ground truth, we report also the accuracy of a look-up table, consisting of 489 648 samples (the same used as train set for the ML methods).

5.4. Hybrid solution

The results of the hybrid approach are herein discussed. This approach guarantees a higher accuracy for the predictions of the model, provides more flexibility and it is faster to compute than the standard Markov–Dubins solver. With this approach, it is possible to trade some accuracy of the bare model for a faster execution time.

When making a prediction, the model can return the probability associated with each class. In the first proposed solution (bare model), the class with the maximal probability is returned as the predicted class. In general, along the border between two classes (see Fig. 3), there can be multiple classes that receive a high probability. For this reason, a threshold on the predicted probability is set and the solution for all the candidate manoeuvres that are above the threshold is computed.

For each model trained in the previous approach, several values for *Threshold* have been chosen: $\{0.001, 0.01, 0.1\}$. The predicted classes that have a probability higher than the chosen thresholds are tested using a standard Markov–Dubins solver and the best manoeuvre among them is returned. This returns the correct solution if the optimal manoeuvre has not been incorrectly pruned by the model.

The top-5 models in terms of total accuracy that run faster than the standard Markov–Dubins solver are presented in Table 5.

The information provided in Table 5 can be read in several ways. For each model it is possible to consider the accuracy and speed of the bare model, *i.e.*, just selecting the manoeuvre with the highest probability or the hybrid model, which uses the stated threshold and computes the length of several manoeuvres to select the best one.

The best model (#1), when running bare achieves an 8.3-fold speedup compared with the standard Markov–Dubins solver, with an accuracy of 91.651%. Then, running this in hybrid mode with threshold, an accuracy of 98.556% is reached, with an execution time comparable with the classical solver $T_{\text{hybrid}} = 58$ ms.

Another point of view considers the error committed in terms of relative additional length to be travelled – *i.e.*, when the selected manoeuvre is sub-optimal – all the best models lie in the range of 58–77% on average, with the best model being #4, which is still 2.9 times faster than the standard Markov–Dubins solver.

Finally, the model with the lowest total execution time among the selected ones is #5, which runs in 53 ms, while maintaining a high accuracy both for the bare (92.855%) and hybrid (95.494%) models.

Table 6 reports the percentages of the tested manoeuvres for the given threshold, highlighting the proportion of when the model runs bare with respect to when it runs hybrid (when it tests two or more manoeuvres). In the majority of cases, every model returns the manoeuvre with the highest probability. The first model runs bare for 69% of the cases, thus returning the solution 8.3 times faster than the standard solver. In the remaining 31% of the cases, it works hybrid and has to test two or more candidate solutions with the standard solver. The worst case happens only in 12.3% of cases, when all 6 manoeuvres are tested, as with the standard solver.

Remark 2. The present study aims at understanding what can be the benefits of a ML approach in a path planning problem. Clearly, it is not possible to do better in terms of accuracy than the closed-form method represented by the standard solver, but we have proved that it is possible to reach the same time performance of the standard solver also with the learnt model, solving correctly more than 95% of the cases, a threshold that can be accepted in the application, for instance if the method is combined with a rapidly exploring algorithm like RRT*, which is a popular technique, or in case of high speed obstacle avoidance in real-time, for instance using a Dubins-tentacle method (like it has been proposed with clothoid curves in Alia et al. (2015)), where speed of computation is more important than length optimality (which is still at most 77% more than the optimum). The speed profile, taking into account the vehicle dynamics, can then be computed in semi-closed form, e.g., as in Frego et al. (2017), since Markov–Dubins paths can be considered particular cases of clothoids (Bertolazzi et al., 2018a,b). The ML method is also clearly better in terms of running time than any of the other optimisation-based methods.

Also, the availability of complete knowledge in the 2D case, allows us to precisely measure the results obtained with ML as well as with the other optimisation-based techniques.

What is more, in our perspective, is to understand what kind of help can this ML method give if we look at the more challenging problem of Markov–Dubins in 3D (e.g., shortest path for an airplane), where closed-form solutions are not known and a solver should rely on optimisation-based methods only. A ML-based solver in bare or hybrid mode would be very beneficial to reduce the number of trial-and-error manoeuvre testing to 1 or a bunch, respectively. Indeed, testing by brute force the many cases for a single 3D path, computing a NLP each time, takes some minutes: being able to select just a few manoeuvres to test would speed-up the process a lot, by inserting in the problem the characteristics of the candidate solution, thus reducing the number of the unknowns, in particular the integer variables of curvature and torsion.

5.5. Computational remarks

The CatBoost library offers a set of APIs for embedding models in other languages such as C, Python, and Rust, and comes with a huge overhead. To mitigate this behaviour, several strategies were tried, for example ensuring that the model was loaded ahead of time and that all support and intermediate data-structures were pre-built by running some warm-up evaluation tests.

Unfortunately, the issues seem to reside deep inside of the CatBoost library itself and in particular with how model evaluation APIs are being exposed. Indeed, the same behaviour has been observed across different languages.

For this reason, it was decided to use the batch API for model evaluation to minimise the effects of this intrinsic overhead by spreading it over all the test samples (69 950), being aware that this may not cover the same use cases as a classical Markov–Dubins solver.

An ad-hoc model evaluator can be developed for using the solution in production, which should overcome the aforementioned limitations faced with the CatBoost library. This solution remains an avenue to be explored in future work, however, it is expected that it will bring significant speedups.

Table 5
Top-5 hybrid models faster than the standard Markov–Dubins solver ordered by accuracy (higher is better, in bold the best model). The last column reports the relative additional length to be travelled (lower is better, in bold the best model).

#	No. of Iter.	Tree-Depth	Threshold	Bare Time (ms)	Hybrid Time (ms)	Training Time (s)	Size (kB)	Bare Acc. (%)	Hybrid Acc. (%)	Additional Len. (%)
1	3	12	0.01	**7**	58	3062	679	91.651	**98.556**	77
2	25	8	0.10	14	55	4613	345	94.084	96.417	65
3	12	12	0.10	13	58	12297	2696	93.633	96.096	64
4	25	10	0.10	20	57	8749	1409	94.004	95.920	**58**
5	12	8	0.10	10	**53**	2207	176	92.855	95.494	66

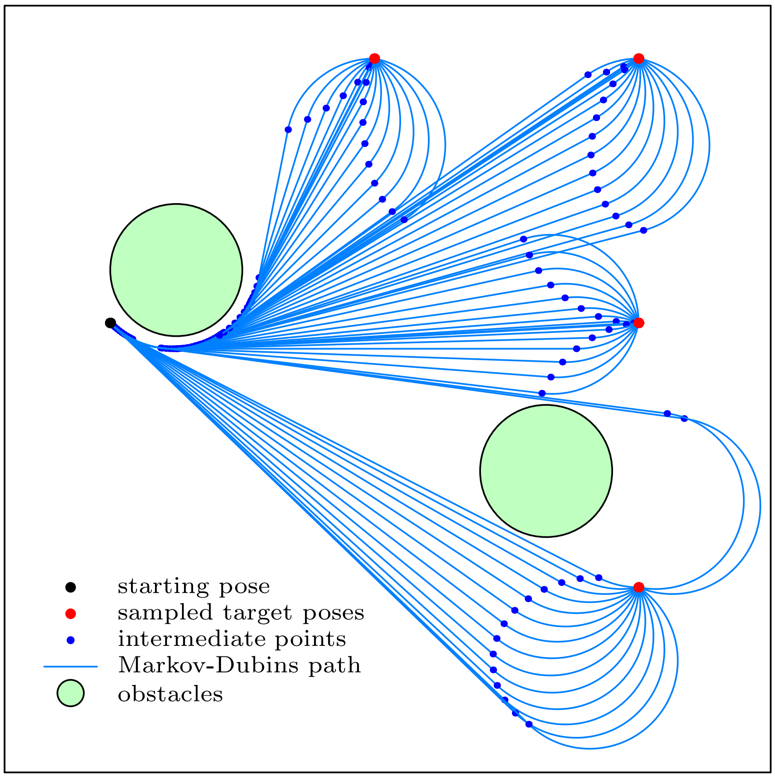


Fig. 5. Example of exploring tentacle method based on the Markov–Dubins planner. Starting from an initial pose, given by the current vehicle's position, many paths are computed simultaneously for multiple end poses, i.e. final points and orientations.

Table 6
Percentage of test points that required computing n manoeuvres.

#	Thr.	1	2	3	4	5	6
1	0.01	69.0	12.7	3.3	1.4	1.3	12.3
2	0.10	90.8	8.3	0.9	–	–	–
3	0.10	90.8	8.1	1.1	0.1	–	–
4	0.10	92.8	6.4	0.7	–	–	–
5	0.10	90.4	8.3	1.3	–	–	–

The task domain is restricted to variables in double precision; furthermore, the trained model can be exported in various formats, such as JSON. Then the model prediction can be evaluated by running the decision trees over the input features. This ad-hoc solution should also have a smaller footprint than the model provided by CatBoost, whose shared library weighs about 125 MB. The performance of such an evaluator is reasonably expected to be on par or better than the CatBoost's one, so this works' conclusions stand.

Experiment in Batch mode. A possible way of taking advantage of the vectorised instruction provided by CatBoost (as well as by many other ML tools) is in exploring task or reachability analysis. In such applications, a large number of paths needs to be generated at the same sampling time. For instance, we can combine the present approach with an exploration method based on the tentacle technique: given the current vehicle pose, many possible destination poses around it are sampled and the two posed connected with a Markov–Dubins path, see Fig. 5. The resulting paths are then pruned checking for possible collisions with obstacles, weighted with a score function like length, comfort, jerk, or other metrics and the best candidate is fed to the low level controller to be executed. Clearly, the finer the granularity of this exploration, the better the performance. The application scenarios, for a real car are usually of limited power embedded hardware and time limitations (100 ms), (Alia et al., 2015; Mouhagir et al., 2020). In this experiment we compared the times and number of generated paths with the standard solver and with our approach. For consistency, we kept the computational platform used for the previous examples, since we are interested more in the ratio of times than in their absolute value. We generated 1000 Markov–Dubins tentacles, like the ones depicted in Fig. 5. The computation with the standard solver took about 0.54 ms, whereas our method (with the most accurate hybrid model) took 0.09 ms, corresponding to a 6-fold gain. In a real planner (which is out of the scope of the paper), it would be then required to check if the paths are collision-free at least with static obstacles, find a suitable speed profile, merge path and speed profile to produce a trajectory and, finally, evaluate each feasible path with a score metric like length, time, energy, comfort, jerk, etc.

6. Conclusions and future research perspectives

The applicability of machine learning models and techniques to path planning and in particular to the Markov–Dubins problem has been explored in this paper. This fundamental planning problem is simultaneously easy enough to state and highly nonlinear to be challenging. Also, its complexity is sufficiently low to make a brute force solution search feasible, which allows for the creation of an exact supervised model for test and comparison.

A state-of-the-art machine learning model, based on the CatBoost library, was trained in a supervised setting on the Markov–Dubins problem and used in two different ways: to directly predict the best manoeuvre, and to filter candidate manoeuvres over a certain probability threshold before applying the standard solver. Aim of the study was to compare the quality and speed of these solutions with a state of the art solver, which is not based on machine learning.

Two solutions are therefore proposed: the first one focuses only on accuracy and, as best model, is selected the one with the highest predicted probability. This approach leads to selecting the correct manoeuvre in over 95% of the cases, albeit being slower than the standard Markov–Dubins solver that is not ML-based; a big advantage is however present if we compare our solution with solvers based on OCP, NLP and MINLP that have calculation times higher than one or two orders of magnitude.

The second is a hybrid approach that cares about accuracy and execution time using a threshold on the prediction probabilities; all the manoeuvres for which the model predicted a probability higher than a predefined threshold were tested with the standard Markov–Dubins solver. The best solution, when running in the fastest configuration, achieves an 8.3-fold speedup compared to the standard Markov–Dubins solver, with an accuracy of over 91% and a maximal suboptimal length of at most 77% of the optimal length. With the maximal accuracy configuration, it reaches an accuracy of over 98% with an execution time on par with the classical solver.

This exploratory work provides several insights into the application of machine learning techniques to the Markov–Dubins problem and it should represent a starting point for future research. For instance, the present work can be used in a future work to investigate the possibility of generating a large number of manoeuvres in a shorter time with respect to the standard brute force solution in the framework of sampling-based motion planners, also for the estimation of a heuristic cost to reach the goal posture.

Another consideration to make is that the Dubins car moves only forward; its extension, the Reeds-Shepp car, also moves backwards (Reeds and Shepp, 1990), generating up to a total of 48 possible manoeuvres made of lines and circles for the trial-and-error method. A further example is provided by the extension of the Markov–Dubins problem in 3D (Kladis et al., 2011; Marino et al., 2016). Already for these examples, the benefits of the proposed machine learning methods will be more significant.

CRediT authorship contribution statement

Cristian Consonni: Conceptualization, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Martin Brugnara:** Conceptualization, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Paolo Bevilacqua:** Conceptualization, Methodology, Software, Visualization, Writing – original draft; Writing – review & editing. **Anna Tagliaferri:** Conceptualization, Methodology, Software, Visualization, Writing – original draft, Writing - reviewing and editing. **Marco Frego:** Conceptualization, Methodology, Software, Visualization, Writing – original draft; Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

The participation of M. Frego in this research is supported by the project "Clothoids for Robotics" (CLO4ROB), funded by the Free University of Bozen-Bolzano, Faculty of Science and Technology (RTD call 2021). This work was supported by the Open Access Publishing Fund of the Free University of Bozen-Bolzano. All authors approved the version of the manuscript to be published.

References

Alia, C., Gilles, T., Reine, T., Ali, C., 2015. Local trajectory planning and tracking of autonomous vehicles, using clothoid tentacles method. In: 2015 IEEE Intelligent Vehicles Symposium. IV, pp. 674–679.
Bertolazzi, E., Bevilacqua, P., Biral, F., Fontanelli, D., Frego, M., Palopoli, L., 2018a. Efficient re-planning for robotic cars. In: 2018 European Control Conference. ECC, pp. 1068–1073.
Bertolazzi, E., Bevilacqua, P., Frego, M., 2018b. Clothoids: a C++ library with Matlab interface for the handling of clothoid curves. Rend. Semin. Mat. Univ. Politec. Torino 76 (2), 47–56.
Bertolazzi, E., Frego, M., 2019. A note on robust biarc computation. Comput.-Aided Des. Appl. 16 (5), 822–835.
Bertolazzi, E., Frego, M., Biral, F., 2020. Interpolating splines of biarcs from a sequence of planar points. Comput.-Aided Des. Appl. 18 (1), 66–85.
Bevilacqua, P., Frego, M., Fontanelli, D., Palopoli, L., 2020. A novel formalisation of the Markov-Dubins problem. In: Proceedings of the European Control Conference 2020. pp. 1987–1992.
Boissonnat, J.-D., Bui, X.-N., 1994. Accessibility Region for a Car that Only Moves Forwards Along Optimal Paths. Technical Report 2181, INRIA.
Boissonnat, J.-D., Cérézo, A., Leblond, J., 1994. Shortest paths of bounded curvature in the plane. J. Intell. Robot. Syst. 11 (1), 5–20.
Bolton, K.M., 1975. Biarc curves. Comput. Aided Des. 7 (2), 89–92.
Dubins, L., 1957. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. Am. J. Math. 79 (3), 497–516.
Duindam, V., Xu, J., Alterovitz, R., Sastry, S., Goldberg, K., 2010. Three-dimensional motion planning algorithms for steerable needles using inverse kinematics. Int. J. Robot. Res. 29 (7), 789–800.
Faigl, J., Vana, P., Drchal, J., 2020. Fast sequence rejection for multi-goal planning with dubins vehicle. pp. 6773–6780.
Frego, M., Bertolazzi, E., Biral, F., Fontanelli, D., Palopoli, L., 2017. Semi-analytical minimum time solutions with velocity constraints for trajectory following of vehicles. Automatica 86, 18–28.
Frego, M., Bevilacqua, P., Saccon, E., Palopoli, L., Fontanelli, D., 2020. An iterative dynamic programming approach to the multipoint markov-dubins problem. IEEE Robot. Autom. Lett. 5 (2), 2483–2490.
Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press, http://www.deeplearningbook.org.
Gulyas, M., Csiszer, M., Mehes, E., Czirok, A., 2018. Software tools for cell culture-related 3d printed structures. PLoS One 13 (9), 1–11.
Hameed, I.A., 2017. Coverage path planning software for autonomous robotic lawn mower using Dubins' curve. In: 2017 IEEE International Conference on Real-Time Computing and Robotics. RCAR, pp. 517–522.
Hancock, J.T., Khoshgoftaar, T.M., 2020. Catboost for big data: an interdisciplinary review. J. Big Data 7 (1), 1–45.
Kaya, Y., 2017. Markov–Dubins path via optimal control theory. Comput. Optim. Appl. 68 (3), 719–747.
Kaya, C.Y., 2019. Markov–Dubins interpolating curves. Comput. Optim. Appl. 73 (2), 647–677.
Kladis, G.P., Economou, J.T., Knowles, K., Lauber, J., Guerra, T.-M., 2011. Energy conservation based fuzzy tracking for unmanned aerial vehicle missions under a priori known wind information. Eng. Appl. Artif. Intell. 24 (2), 278–294.
Laumond, J.-P., Jacobs, P.E., Taïx, M., Murray, R.M., 1994. A motion planner for nonholonomic mobile robots. IEEE Trans. Robot. Autom. 10 (5), 577–593.
Marino, H., Salaris, P., Pallottino, L., 2016. Controllability analysis of a pair of 3d dubins vehicles in formation. Robot. Auton. Syst. 83, 94–105.
Markov, A.A., 1887. Some examples of the solution of a special kind of problem on greatest and least quantities. Soobshch. Karkovsk. Mat. Obshch. 1, 250–276.

Mouhagir, Hafida, Talj, Reine, Cherfaoui, Veronique, Aioun, Francois, Guillemard, Franck, 2020. Evidential-based approach for trajectory planning with tentacles, for autonomous vehicles. IEEE Trans. Intell. Transp. Syst. 21 (8), 3485–3496.

Ny, J., Feron, E., Frazzoli, E., 2012. On the dubins traveling salesman problem. IEEE Trans. Automat. Control 57 (1), 265–270.

Park, H., 2004. Optimal single biarc fitting and its applications. Comput.-Aided Des. Appl. 1 (4), 187–195.

Piegl, L.A., Tiller, W., 2002a. Biarc approximation of nurbs curves. Comput. Aided Des. 34 (11), 807–814.

Piegl, L.A., Tiller, W., 2002b. Data approximation using biarcs. Eng. Comput. 18 (1), 59–65.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A., 2019. Catboost: unbiased boosting with categorical features.

Razmjooy, N., Mousavifard, R., Abolghasemzadeh, M., Alizadeh, Y., 2019a. Optimal control of self-balancing robot in the presence of uncertainties based on interval analysis.

Razmjooy, N., Ramezani, M., Namadchian, A., 2016. A new lqr optimal control for a single-link flexible joint robot manipulator based on grey wolf optimizer. 10, p. 53.

Razmjooy, N., Ramezani, M., Vieira Estrela, V., 2019b. A solution for dubins path problem with uncertainties using world cup optimization and chebyshev polynomials. In: Iano, Yuzo, Arthur, Rangel, Saotome, Osamu, Estrela, Vânia Vieira, Loschi, Hermes José (Eds.), Proceedings of the 4th Brazilian Technology Symposium. BTSym'18, Springer International Publishing, Cham, pp. 45–54.

Reeds, J.A., Shepp, L.A., 1990. Optimal paths for a car that goes both forwards and backwards. Pacific J. Math. 145 (2), 367–393.

Sabin, M.A., 1976. The Use of Piecewise Forms for the Numerical Representation of Shape. Computer & Automation Institute, Hungarian Academy of Sciences.

Saccon, E., Bevilacqua, P., Fontanelli, D., Frego, M., Palopoli, L., Passerone, R., 2021. Robot motion planning: can GPUs be a game changer? In: 2021 IEEE 45th Annual Computers, Software, and Applications Conference. COMPSAC, pp. 21–30.

Sharma, O., Sahoo, N.C., Puhan, N.B., 2021. Recent advances in motion and behavior planning techniques for software architecture of autonomous vehicles: A state-of-the-art survey. Eng. Appl. Artif. Intell. 101, 104211.

Shkel, A.M., Lumelsky, V., 2001. Classification of the dubins set. Robot. Auton. Syst. 34 (4), 179–202.

Soueres, P., Laumond, J.-P., 1996. Shortest paths synthesis for a car-like robot. IEEE Trans. Automat. Control 41 (5), 672–688.

Sussmann, H.J., Tang, G., 1991. Shortest paths for the Reeds-Shepp car: a worked out example of the use of geometric techniques in nonlinear optimal control. Rutgers Cent. Syst. Control Tech. Rep. 10, 1–71.

Váňa, P., Faigl, J., 2020. Optimal solution of the generalized dubins interval problem: finding the shortest curvature-constrained path through a set of regions. Auton. Robots 44 (7), 1359–1376.

Webster III, R.J., Kim, J.S., Cowan, N.J., Chirikjian, G.S., Okamura, A.M., 2006. Nonholonomic modeling of needle steering. Int. J. Robot. Res. 25 (5–6), 509–525.

Zhang, J.M., Harman, M., Ma, L., Liu, Y., 2020. Machine learning testing: Survey, landscapes and horizons. IEEE Trans. Softw. Eng. 1.