2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Complexity Reduction of the Three-Point Dubins Problem (3PDP) via Symmetry Exploitation for Machine Learning Purposes

Marco Frego *Member IEEE*, Enrico Saccon *Student Member IEEE*,
Davide De Martini, Luigi Palopoli *Senior Member IEEE*

***Abstract*— This work proposes a machine learning approach for the Three-Point Dubins Problem (3PDP) based on classification and regression. The 3PDP is a path planning problem with Dubins curves through 3 waypoints. The goal is to find the heading at the intermediate point and the form of the two Dubins paths joining the three points. Classification is used to select the correct path type (out of 18) to avoid the trial-and-error enumeration of all cases; regression is employed to have a good initial guess for finding the heading angle. Our results are used to improve and speed-up existing methods in terms of efficiency and accuracy.**

I. Introduction

A fundamental concept in motion planning, particularly in robotics, autonomous vehicles, and aerospace navigation is represented by the family of Dubins related problems [10]. They address the challenge of determining the shortest path between points in the plane or in space, while considering curvature constraints on the vehicle's motion. The problem assumes that the vehicle moves forward at a constant speed and has a minimum turning radius, or, equivalently, a maximum curvature bound, which models non-holonomic features of the system, e.g., a wheeled robot or a fixed-wing aircraft.

The importance of the Dubins problem in motion planning derives from the characteristic of generating feasible and optimal paths for curvature constrained systems, unlike traditional shortest-path algorithms, such as Dijkstra's or A* on a grid, which, in their basic definition, do not account for the vehicle's kinematic limitations. This ensures that the resulting trajectory is both G^1 and physically realizable. We use such construction as a baseline and further smooth it [1], [5]) with clothoids to obtain G^2 continuity and use it as a reference for trajectory tracking purpose, in particular, with the Internal Model Principle [14]. This step is crucial for real-world applications, where abrupt turns or unrealistic manoeuvres can lead to inefficiencies, uncomfortable sensations and other undesired situations.

By using Dubins' theory, autonomous vehicles can efficiently compute paths that avoid obstacles and respect road constraints [11], [17]. Similarly, in aerospace applications, Dubins' paths help design flight trajectories for drones and UAVs that must follow aerodynamic constraints [24], [13]. In automated warehouses [5], and in marine navigation [16], vehicles often operate with similar kinematic constraints, requiring smooth and feasible paths to navigate efficiently.

Beyond practical applications, Dubins' problem has deep theoretical significance in control theory and computational geometry. It serves as a foundation for studying non-holonomic motion planning, influencing algorithms that extend beyond simple forward-motion constraints. A notable extension is the Dubins Traveling Salesman Problem (DTSP) [22], [15], which generalizes the original problem to multiple waypoints. The DTSP seeks the shortest possible route that visits a series of target locations while adhering to Dubins' motion constraints, making it highly relevant for applications in aerial surveillance, robotic inspection, and autonomous delivery systems, [7].

Variants of Dubins' paths, such as Reeds-Shepp curves, which allow for both forward and reverse motion, further expand the applicability of Dubins' theory in robotics and automation, [19]. Furthermore, Dubins' paths have recently been investigated by means of Machine and Deep Learning [4]: neural networks can be used to predict Dubins paths without using the traditional approach of enumerating the possible candidate solutions. These techniques enhance Dubins-based motion planning by improving real-time decision-making and optimizing computational efficiency.

The present paper contributes in this direction by proposing a Machine Learning approach for selecting the optimal Dubins path through a sequence of three points, referred to as the Three-Point Dubins Problem (3PDP). Although highly specialised, this task is essential for efficient path editing within the solution of a DTSP and for the planning of ground, aerial, and underwater vehicles [16]. Moreover, it can enhance the performance of algorithms addressing the Curvature-Constrained Shortest-Path Problem (CCSPP), including approaches based on convex optimisation [8], [2], nonlinear programming [9], and dynamic programming [6].

The work exploits interesting symmetries in the 3PDP problem to reduce its dimensionality and construct a more compact dataset. Three models were then trained, a K-nearest neighbor model, a classification neural network and a regression neural network. We present their structure, the training and the results that we obtained.

M. Frego is with Faculty of Engineering, Free University of Bozen-Bolzano, Bolzano, Italy. marco.frego@unibz.it.

E. Saccon, D. De Martini and L. Palopoli are with Department of Information Engineering and Computer Science, University of Trento, Trento, Italy. {name.surname}@unitn.it.

This work was funded by the Italian Ministry of University and Research (MUR) under the PRIN project - DD n. 1401 of 18.09.2024, project "Clothoids in 3D for engineering and biomechanical applications" (Clothoid3d)_2022E954LL, CUP I53C24002460006.

II. Problem Formulation and Related Work

A. Problem Formulation

Given a vehicle constrained to move on a plane along paths with curvature bounded in modulus by a value $\kappa > 0$, the Three-Point Dubins Problem (3PDP), [8], [21], [3], [18], [16], seeks the shortest path, that:

- Starts at an initial configuration (P_i, ϑ_i),
- Passes through an intermediate waypoint P_m,
- Ends at a final configuration (P_f, ϑ_f).

Each configuration consists of a position $P = (x, y)$ in the plane and an orientation $\vartheta \in [0, 2\pi)$. The resulting path is proved to be composed of arcs of circles with a minimum turning radius $\rho = 1/\kappa$ and straight-line segments, as defined by Dubins curves, and actually is the juxtaposition of two classic Dubins paths, smoothly joined. Formally, the objective is to determine the shortest path $\gamma : [0, L] \to \mathbb{R}^2$ that satisfies the following constraints:

$$\gamma(0) = P_i, \qquad \gamma(s_m) = P_m, \qquad \gamma(L) = P_f,$$
$$\dot{\gamma}(0) = \vartheta_i, \qquad \dot{\gamma}(L) = \vartheta_f, \qquad |k(s)| \leq \kappa,$$

where s_m is the arc length corresponding to the middle waypoint, $k(s)$ is the curvature function and L is the path length (to be minimised). Due to Bellman's Dynamic Programming Principle (DPP), the solution is the (optimal) juxtaposition of two Dubins path, the first from (P_i, ϑ_i) to the middle point (P_m, ϑ_m), where ϑ_m is the angle at P_m, and the second from that middle point and orientation to the final configuration (P_f, ϑ_f). Therefore, the solution to the 3PDP is one dimensional and accounts for finding the optimal transit angle ϑ_m at P_m using interpolating Dubins paths. This suggests that there can be a maximum of $6 \cdot 6 = 36$ possible Dubins path combinations yielding a valid solution. However, it has been proved in [3], that only 18 of them are optimal, which are listed in Table I. The letter C identifies a circle, whereas S stands for straight line; also, R and L are a right and left turn, respectively.

TABLE I
TYPES OF THE 18 ADMISSIBLE PATHS, ACCORDING TO [3].

CCCCC:	RLRLR, LRLRL
CCCSC:	RLRSR, RLRSL, LRLSL, LRLSR
CSCCC:	RSRLR, LSRLR, RSLRL, LSLRL
CSCSC:	RSRSR, LSRSR, RSRSL, LSRSL, LSLSL, RSLSL, LSLSR, RSLSR

B. Related Work

According to the previous discussion, it is straightforward to solve the 3PDP by discretising the range $[0, 2\pi)$ of possible values for ϑ_m and test by enumeration the resulting path lengths. The shortest candidate is elected as solution of the problem. This technique takes the name of Discretisation Based Method (DBM) and when $[0, 2\pi)$ is discretised with 360 samples, it is used as reference benchmark for comparing the performance of other methods, [21], [3], [18], [16]. Despite DBM being easy to implement and effective, it is not very efficient when called a large number of times inside other planners, like the DTSP.

Therefore, the 3PDP has received a significant attention in the last few years. Research on the 3PDP has progressed along several directions, with various methodologies proposed to solve it efficiently. The first alternative solutions to DBM were based on Inversive Geometry (IG), which was exploited to develop an iterative method [21]. Therein, the problem is reduced to a nonlinear system of 3 equations in 3 unknowns, yielding a computational improvement over the DBM in terms of both computational time and accuracy of the resulting angle ϑ_m. A limitation of this method is that it considers 8 of the 18 cases (i.e., the cases with CCC subpaths are excluded), possibly leading to non-optimal solutions.

A different method, [3], which is based on the observation that the angle ϑ_m satisfies a certain trigonometric relation, allows for producing a single polynomial equation in the variate $\tan(\vartheta_m/2)$ for each of the 18 possible cases. The main advantage of this approach is that it is recast to a single equation, that can be solved using standard polynomial root finding algorithms. The drawback is that all 18 polynomials need to be constructed (starting from the initial and final condition of each instance) and solved: the degree of the polynomials varies from 4 to 20, for a total of at most 180 possible roots. The construction of such high degree polynomials implies instabilities of the roots, due to the inexactness of the coefficients and to the large number of monomial which constitute them. Each solution needs to be used to evaluate the corresponding path length, and the one yielding the shortest path is elected as the global optimal solution. This method is called Polynomial Based Method (PBM) and has a good advantage over DBM and IG, for both accuracy and computational times.

An alternative approach based on analytic geometry, called Geometry Based Method (GBM), uses geometric tools to compute the solution [16]. It is based on the observation that the ellipse having foci the centers of the initial and final circles of the optimal manoeuvre and which is tangent to the circle centered at P_m solves the problem: the optimal angle ϑ_m is equal to the angle of the common tangent line between such an ellipse and the circle centered at P_m. Unfortunately, the results of [16] cover only the paths that do not contain CCC subpaths, like the IG method.

Finally, a non-smooth numerical optimization approach has been formulated [18]. It addresses the challenge of the discontinuity of the length function in the problem, providing an analytical expressions for the derivatives of the Dubins path length and leading to improved accuracy and speed.

C. Impact and Applications

The 3PDP is a core component in solving the DTSP, which involves finding the shortest path for a Dubins vehicle to visit a set of points and return to its starting point. The 3PDP provides insight into solving the multipoint Markov-Dubins problem [9], [6], that is, the interpolation of a given ordered sequence of n points with a Dubins path satisfying the curvature-constrained shortest path problem (CCSPP), which

is a standalone planning problem or can be considered as an intermediate step of the DTSP. In multiagent robotics, the 3PDP is related to the multiple traveling salesperson problem (MTSP), where multiple vehicles visit a set of points.

The recent active research on the 3PDP is motivated by the need for fast and efficient path planning in various applications, such as in Unmanned Aerial Vehicles (UAVs), marine vehicles, and autonomous robots.

A fundamental problem of the PBM and GBM is that they have to enumerate the 18 cases, in order to fully exploit their effectiveness. Each case is solved by means of a particular subproblem. Having a predictor of which cases to consider and of an initial guess of the angle ϑ_m would greatly impact their performance. The identification of the case allows to directly apply the corresponding subproblems; a good initial guess for the angle ϑ_m allows to warm-start a Newton-like method for the IG or for the PBM (if tackled as a general nonlinear function and not using QR-based eigenvalue methods for finding all the roots simultaneously). Indeed, for the PBM, besides solving an up to degree 20 polynomial, each solution must be tested to find the one yielding the shortest path. Having a good initial guess would help to reduce the number of roots to find or give the opportunity to use a Newton-like method and to converge directly at the correct root, given that the initial guess is inside the basin of attraction of the root (e.g., see Figure 3).

The initial guess is also useful to improve discretisation based methods (e.g. DBM) to explore angles close to the solution, avoiding unnecessary sampling and search on intervals far from it. A direct benefit of it, is either an increased accuracy in the precision or a reduced number of iterations.

Discussion on these points is presented in the Results Section V on a concrete study case.

III. Reduction of the Complexity

The problem, in its general formulation, does not lend itself naturally to machine learning for classifying the path type or for regression on the optimal angle, because of the large number of degrees of freedom (the problem is defined by 9 real variables) and because of the possibly large variation of the values of the input variables, that can span from $-\infty$ to $+\infty$. Indeed, six of the nine variables correspond to the coordinates of the points P_i, P_m and P_f, which can take arbitrary real values, 2 more variables are the initial ϑ_i and final ϑ_f angles, which are bounded, and finally the curvature, which takes values in $(0, \infty)$. Overall, there are $6 + 2 + 1$ variables, of which only two are defined on a compact set, and therefore can be efficiently sampled, making it difficult to construct a dataset that represents the variability of the other variables without making it prohibitively large.

In this section we show how to reduce the input variables from 9 to 5 and how we can formulate the problem in such a way that 4 of these variables are angles, which can therefore be discretised into a compact set, the fifth being the curvature, which still spans the semi-infinite set $(0, \infty)$. Next, we describe the invertible transform that allows us to reduce the complexity of the problem and lends itself to the construction of a representative dataset of virtually all possible cases. The idea is to describe the points P_i, P_m and P_f on a common scaled circle, thus replacing the need of using two Cartesian coordinates for each of them with an angle, which is a quantity defined in a bounded range.

Remark 1. *We have to distinguish two configurations of the points: when P_i, P_m and P_f are collinear, and when not. The case of collinear points is theoretically possible but not very probable, however it is discussed separately [20]. Therefore, herein, we focus on the case when P_i, P_m and P_f form a non degenerate triangle. Thus, it is possible to find the circle passing through them, as shown in the next proposition.*

Proposition 2. *Let P_i, P_m and P_f be three non collinear points, then there exist a circle of center $P_c = (x_c, y_c)$ and radius $r > 0$ such that the three points belong to that circle.*

Proof. The proof is constructive and is included for making the paper self-contained. The circle parameters are found by solving the following interpolation problem that yields the center of the circle and its radius. Let $\boldsymbol{A}$ be the matrix

$$\boldsymbol{A} = \begin{bmatrix} x_i^2 + y_i^2 & x_i & y_i & 1 \\ x_m^2 + y_m^2 & x_m & y_m & 1 \\ x_f^2 + y_f^2 & x_f & y_f & 1 \end{bmatrix}$$

and $\boldsymbol{A}_j$ the matrix obtained from $\boldsymbol{A}$ removing column j. Then, the center $P_c = (x_c, y_c)$ of the interpolating circle is

$$x_c = \frac{1}{2}\frac{\det \boldsymbol{A}_2}{\det \boldsymbol{A}_1}, \qquad y_c = -\frac{1}{2}\frac{\det \boldsymbol{A}_3}{\det \boldsymbol{A}_1},$$

and

$$r = \sqrt{x_c^2 + y_c^2 + \frac{\det \boldsymbol{A}_4}{\det \boldsymbol{A}_1}},$$

is its radius. □

Having found the circle, we can eliminate 4 parameters of the problem by shifting the three points by P_c, by rotating by an angle φ so that P_i lies on the positive $x-$axis and by scaling by a factor $1/r$ so that P_i is mapped to the point $(1, 0)$, see Figure 1. This invertible transform T is given by:

$$T\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \frac{1}{r} R(\varphi) \begin{bmatrix} x - x_c \\ y - y_c \end{bmatrix},$$

where,

$$\varphi = -\operatorname{atan2}(y_i - y_c,\ x_i - x_c),$$
$$R(\varphi) = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}.$$

This transform maps (x, y) positions of the three points on the unitary circle, with $T(P_i) = (1, 0)$. Therefore, we can simply use an angle to identify P_m and P_f, respectively α_m and α_f. The advantage of this description is that the input variables reduce to the two angles α_m, α_f, the two shifted

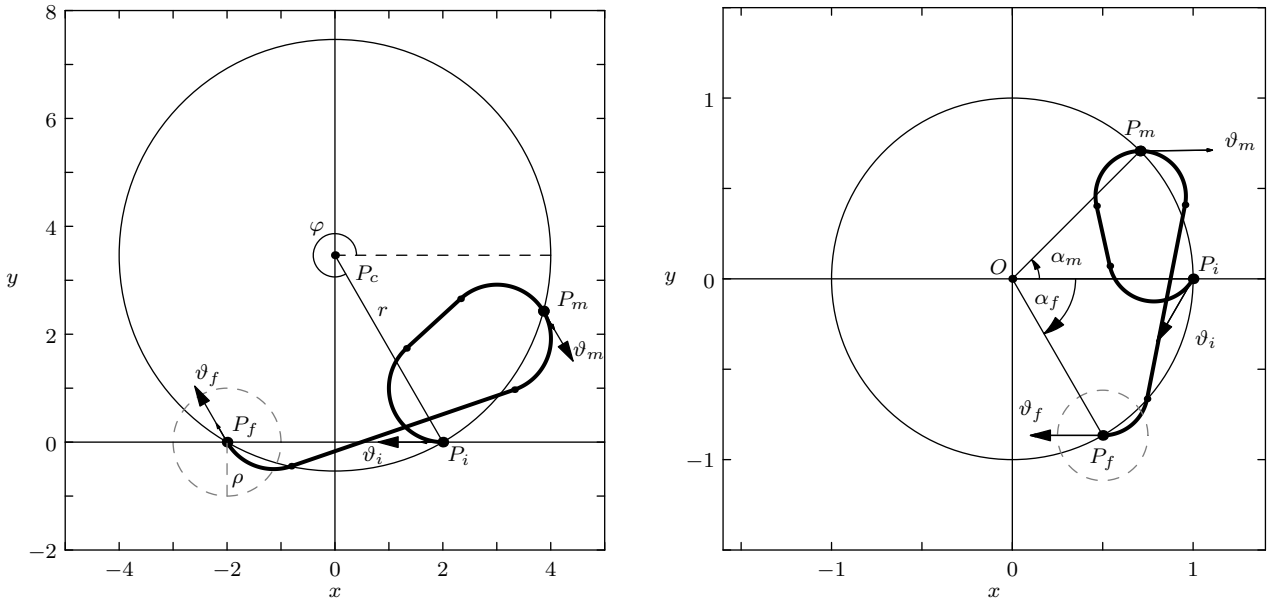

Fig. 1. Left: Points and angles in the original reference system, the dashed circle has radius $1/\kappa^{\text{orig}}$. Right: Transformed points and angles, here P_m and P_f are identified with the angles α_m and α_f; the dashed circle has radius $1/\kappa = 1/(r\kappa^{\text{orig}})$. The thick path represents the optimal 3PDP.

angles ϑ_i and ϑ_f, with the new value of the scaled maximum curvature of $\kappa = r \cdot \tilde{\kappa}$:

$$\alpha_m = \operatorname{atan2}(y_m - y_c,\ x_m - x_c) + \varphi, \qquad \tilde{\vartheta}_i = \vartheta_i + \varphi,$$
$$\alpha_f = \operatorname{atan2}(y_f - y_c,\ x_f - x_c) + \varphi, \qquad \tilde{\vartheta}_f = \vartheta_f + \varphi.$$

With an abuse of notation, we label with the same variables the original and the transformed ones, and we will be working in the transformed space, that we call *canonical*. In ambiguous situations, the tilde "$\sim$" identifies variables in the original space and the hat "$\wedge$" an estimation of a variable.

A. The Dataset

In the canonical space, it is possible to map the 9 input variables defining the problem in the *original* space into 5 of the *canonical*, with the advantage of designing a discretisation that captures the system and does not suffer from points that can be arbitrarily far. Let n_ϑ be an integer that represents the discretisation of the round angle for the angles ϑ_i and ϑ_f; n_α be an integer that represents the discretisation of the round angle for the angles α_m and α_f, and n_κ be the integer for the partition of the interval $[\kappa_{\min}, \kappa_{\max}]$. For the creation of the training dataset, we chose to set $n_\kappa = 41$ with $\kappa_{\min} = 0.2$ and $\kappa_{\max} = 8$, and $n_\alpha = n_\vartheta = 24$, as a trade-off between size of the dataset and coverage. With this choice, the dataset contains $N = 12,142,080$ entries, since some entries are removed for practical reasons, e.g., when P_i or P_f coincide with P_m. The solution for each entry was computed using the MPDP library [6].

The dataset is defined by the input data in the canonical space characterised by the 5 variables ϑ_i, α_m, α_f, ϑ_f and κ: $P_i = (1, 0)$ with angle ϑ_i, $P_m = (\cos\alpha_m,\ \sin\alpha_m)$, $P_f = (\cos\alpha_f,\ \sin\alpha_f)$ with angle ϑ_f. The solution of the problem can be seen in two ways, one is the angle ϑ_m at P_m, the other one is the type of the manoeuvre, which is one of the 18 sequences appearing in Table I. The first can be directly used as the initial guess of a sampling based algorithm such as MPDP [6], whereas the second allows us to directly compute the best path using, e.g., method [3].

IV. Training Phase

The training phase employed the specialized dataset described in Section III-A to train three models: a KNN classifier, a neural network for classification, and a neural network for regression. The dataset was first randomly shuffled to increase stochasticity, and then split into 70% training, 15% validation, and 15% testing. While this split was used for the neural networks, the KNN model does not require a validation set, so that portion was discarded.

A. KNN

The KNN algorithm was set up using the Annoy library, which is a state-of-the-art library for precise modeling of the nearest neighbour algorithm. The Annoy algorithm exploits a forest of random trees to better divide the values and take decisions at inference time. The number of trees in the forest is an hyperparameter, of which a few alternatives were tested, but the best results were obtained by setting the value to 11.

The training phase for such a model is quick when compared to the neural network models, taking only approximately 1 minute when run on a MacBook Pro with an M1 Pro processor. The Annoy library does not allow for embedding the labels directly into the model, whereas it extracts the closest indexes w.r.t. the rest of the points in the training set. This means that, when saving the model for inference purpose, one also has to save the vector of all the training labels. This can quickly become a large file to process, hence we ordered the training data by class and

TABLE II

THE AVERAGE INFERENCE TIMES AND ACCURACY TO GET N LABELS.

# Labels N		1	2	3	4	5
NN Class.	t [μs]	56.07	55.69	54.83	54.94	66.66
	Acc [%]	96.73	99.63	99.92	99.99	100.0
KNN	t [μs]	101.6	6086	26271	83092	151288
	Acc [%]	86.82	97.22	99.39	99.87	99.95

saved the interval of the indexes. This, coupled with a binary search, improves performance for both the reading of the file and the evaluation of the input query.

B. Neural Network Models

For the neural network models, the angles in the dataset had to be split in the corresponding sine and cosine components. Indeed, training the network on the angles without this pre-processing step lead to worse results, with the regression model predicting results with a mean absolute error of almost π. The models were trained over 150 epochs with early stopping after 10 epochs. The batch size was set to 16 and we used the Adam optimizer with both learning rate and weight decay set to 10^{-5}. These hyper-parameters, as well as the number of nodes per layer, were set after different trials to find a trade-off between the size and the inference time of the network, and its accuracy. In this work we focused on using small networks in order to get a faster inference time. Using more complex models such as transformers may lead to better embeddings, but a slower inference time.

Regression Model. The neural network has an initial layer with 9 inputs (the number of features: $\sin$ / $\cos$ of the four angles and κ) and 1024 outputs. These are connected to 3 hidden linear layers with input and output sizes of (1024, 2048), (2048, 2048), (2048, 1024), respectively. We have placed a ReLu activation function after the input layer and before the output layer. We have also placed a LeakyReLu before and after the second hidden layer since it showed to perform better for regression tasks [23], [12]. After the first ReLu layer and we set up a dropout stage (with probability 0.1) to avoid overfitting and to obtain generalization in the training of the network. Eventually, the network outputs two values, the predicted sine and the cosine of $\hat{\vartheta}_m$, which can then be used to find the estimated angle $\hat{\vartheta}_m$. We used the mean squared error loss function to train the model.

Classification Model. The neural network is composed of 5 total layers. The input layer takes 9 inputs and has 2048 connections in output. Three hidden linear layers separate the input and the output, with sizes of (2048,4096), (4096,4096), (4096,2048), respectively. After each linear layer, a ReLU activation is applied to improve the model's ability to capture non-linear patterns. Similarly to the regression model, we set up dropout layers (with probability 0.2) to improve learning abilities between the input and the first hidden layer and between the last hidden layer and the output layer. The output layer outputs 18 classes/manoeuvres.

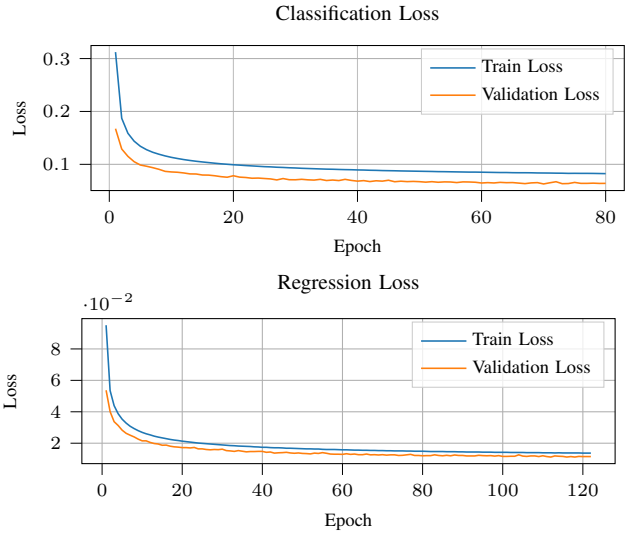

Fig. 2. These graphs show the train and the validation loss for the training of the classification (left) and the regression (right) models.

C. Training and Inference

The previous models were trained using Python3 and then exported in C++, which allows for an easier handling of the data at training time and then improving performance at inference time. The KNN was trained using the Python3 wrappers for the Annoy library. The model was then exported in the format required by the library and an inference program was coded in C++, to boost performance. The neural networks were modeled and trained using PyTorch, taking approximately 76986 and 26724 seconds for classification and regression, respectively. We used a Nvidia A100 with 80GB of VRAM. The models were then exported using the Onnx library, which allows for fast inference executions on multiple platforms. The inference was carried out on a Macbook Pro with M1 Pro. The loss for the classification and the reference training can be seen in Figure 2.

One important remark must be made for the KNN algorithm, because it does not allow for the retrieval of k nearest labels, but only of the nearest indexes. This implies that if we want to check multiple labels for a given input to increase the accuracy of the model at inference time, we need to sample more than k neighbours. We do this by sampling a large number of neighbours (e.g., 10^3) and with the corresponding distances. We then sort them based on the distance and save the first k unique labels.

V. RESULTS

A. Prediction Results

The evaluation phase on the models showed that the regression model is able to achieve a mean angular error of 0.277 ($\sim 3°$) on the test set with an average prediction of $7\mu s$ for the models exported in C++, supporting the hypothesis that the model could provide a valuable initial guess for iterative algorithms, subsequently reducing the number of required refinements to improve the approximation of the solution. The classification results are reported in II alongside with the KNN results (average timings for inference in C++).

The first striking difference is the computation time: the neural network takes only between 50μs and 70μs on average to extract N labels, independently of the value of N. On the contrary, the time to extract N labels from the KNN model changes as the number of neighbours to be extracted increases, taking almost 100ms when having to guess 5 labels. As explained in Section IV-C, this depends on the fact that KNN retrieves neighbours IDs and not their labels, meaning that to get $N > 1$ different labels, one has to query the model for many neighbours. Even when asking for $N = 5$ labels, the performance of the model drops significantly. Moreover, while the neural network is able to reach a 100% accuracy using up to 5 labels, the KNN cannot reach the same performance with a few labels only. The combination of the small time required to predict the class of the three-point path and the high-level accuracy of the model, makes classification a powerful tool to significantly reduce the computational cost of finding the optimal path.

B. A Case Study

Problem instance. Consider the following instance of the problem formulated in the original space, see Figure 1 (left):

$$P_i = (2,0)\,, \quad P_m = \left(\sqrt{2}(1+\sqrt{3}), (2-\sqrt{2})\sqrt{3}+\sqrt{2}\right),$$
$$P_f = (-2,0)\,, \quad \vartheta_i = -\pi, \quad \kappa = 1 = \frac{1}{\rho}, \quad \vartheta_f = -\frac{4\pi}{3}.$$

1) Reference Solution (DBM).: A solution of the problem with the reference method DBM [8], [21], [3], [16], [18], says that the type is RSRSR with angle is $\vartheta_m^{\text{DBM}} = 5.28835$.

2) Solution With the Proposed ML Approach.: The transform to map the original data to the canonical space (see Figure 1 - right) gives $(x_c, y_c) = (0, 2\sqrt{3})$, $r = 4$ and $\varphi = \pi/3$. The data of the problem transforms to

$$P_m = (\sqrt{2}/2, \sqrt{2}/2), \qquad P_f = (1/2, -\sqrt{3}/2),$$

in the canonical space:

$$\alpha_m = \frac{\pi}{4}, \quad \alpha_f = -\frac{\pi}{3}, \quad \kappa = 4, \quad \vartheta_i = -\frac{2\pi}{3}, \quad \vartheta_f = -\pi.$$

The ML prediction of the model in the canonical space says that the type is RSRSR and that the angle estimation is $\hat{\vartheta}_m = -0.0525$. We discuss now how to use this information to improve the performance of other methods.

3) Improvements for the PBM.: The PBM [3] requires to build a specific polynomial for each of the 18 cases, then, each real root produces a candidate optimal angle. Each of the candidates must be compared and the one yielding the shortest path is elected as optimal solution. Using the prediction, it is possible to build and solve just one of the 18 polynomials, the one, $p(z)$, related to the RSRSR case, where $z = \tan(\vartheta_m/2)$ (we do not report $p(z)$ due to space reasons, because of long coefficients). In this case, $p(z)$ is of sixth degree, with 4 real roots, $z_1 \approx -0.569$, $z_2 \approx 0.099$, $z_3 \approx 1.682$, $z_4 \approx -2.321$; and 2 complex roots $z_5, z_6 \in \mathbb{C}$, which are spurious solutions resulting from the squaring process required to build $p(z)$ and can be neglected (see the Newton fractal in Figure 3). Using the regressed variable $\hat{\vartheta}_m$, recast to the original space as $\hat{z} = \tan((\hat{\vartheta}_m + \varphi)/2) = 0.543$, it is possible to conclude that the correct root is z_2 without the need of computing the length of the paths related to z_1, z_3, z_4. The optimal angle associated to root z_2 is $\vartheta_m^{\text{PBM}} = 5.2480313$, way more accurate than the result ϑ_m^{DBM}.

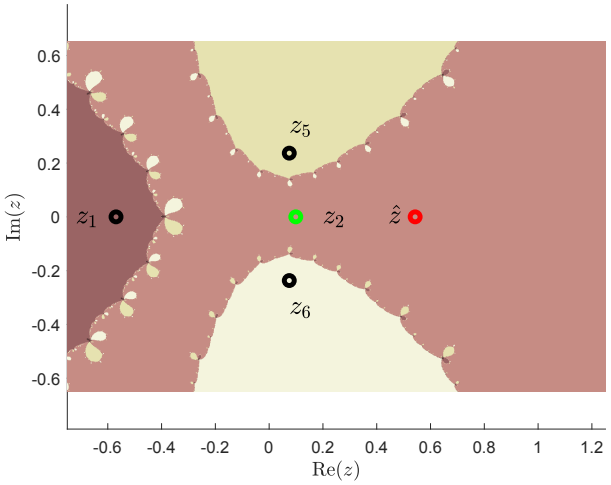

Fig. 3. Newton fractal of the basins of attraction to some of the roots of $p(z)$, marked with the black dots. In green the correct root z_2, in red the predicted root $\hat{z}$, which is closest to z_2 and well inside its attraction basin.

Another option of using our result, in a more conservative way, is to compute all the roots of $p(z)$ but then test the length of the path resulting from the closest roots to $\hat{z}$ only. In our example, just z_1, z_2 and z_3. When $p(z)$ is of higher degree, this choice still yields an improvement.

4) Improvements of Root-Finding Methods.: A problem of the methods based on root-finding (e.g., [21], or [3], if seen as a nonlinear function) is the initial guess to start a Newton-like method. These methods try to solve the nonlinear system of (up to) 5 equations that [3] converts to a univariate polynomial of possibly high degree (4 to 20). The prediction of an approximate solution $\hat{\vartheta}_m$ helps to start the solver close to the correct (global) optimal solution, see Figure 3, where $\hat{z}$ is inside the basin of attraction of the correct root z_2. The advantage is that the nonlinear system is numerically better conditioned than the univariate polynomial, because it has less spurious roots and more stable coefficients.

VI. Conclusions and Future Work

We applied ML techniques to improve the identification process of the correct global solution (among many local) of the Three-point Dubins problem (3PDP). The classification helps in reducing the need to test all 18 cases, the regression gives a good initial guess for identifying the correct root of the nonlinear system or of the univariate polynomial.

Future works would be to exploit Physics-informed Neural Networks to improve the performance of the prediction in terms of smaller models and higher accuracy.

REFERENCES

[1] Efstathios Bakolas and Panagiotis Tsiotras. On the generation of nearly optimal, planar paths of bounded curvature and bounded curvature gradient. In *2009 American Control Conference*, pages 385–390, 2009.

[2] Enrico Bertolazzi, Paolo Bevilacqua, and Marco Frego. Efficient intersection between splines of clothoids. *Mathematics and Computers in Simulation*, 2019.

[3] Zheng Chen and Tal Shima. Shortest Dubins paths through three points. *Automatica*, 105:368 – 375, 2019.

[4] C. Consonni, M. Brugnara, P. Bevilacqua, A. Tagliaferri, and M. Frego. A new Markov–Dubins hybrid solver with learned decision trees. *Engineering Applications of Artificial Intelligence*, 122:106166, 2023.

[5] Marco Frego, Paolo Bevilacqua, Stefano Divan, Fabiano Zenatti, Luigi Palopoli, Francesco Biral, and Daniele Fontanelli. Minimum time—minimum jerk optimal traffic management for AGVs. *IEEE Robotics and Automation Letters*, 5(4):5307–5314, 2020.

[6] Marco Frego, Paolo Bevilacqua, Enrico Saccon, Luigi Palopoli, and Daniele Fontanelli. An Iterative Dynamic Programming Approach to the Multipoint Markov-Dubins Problem. *IEEE Robotics and Automation Letters*, 5(2):2483–2490, 4 2020.

[7] W. Ghadiry, J. Habibi, A. G. Aghdam, and Y. M. Zhang. Enhancements to patrolling operations based on dubins' traveling salesman problem. In *World Automation Congress (WAC)*, pages 1–6, 2016.

[8] Xavier Goaoc, Hyo-Sil Kim, and Sylvain Lazard. Bounded-curvature shortest paths through a sequence of points using convex optimization. *SIAM Journal on Computing*, 42(2):662–684, 2013.

[9] C. Yalçın Kaya. Markov–Dubins interpolating curves. *Computational Optimization and Applications*, 73(2):647–677, Jun 2019.

[10] S. Lavalle. *Planning Algorithms*. Cambridge University Press, 2006.

[11] Xinfang Li, Yangwang Fang, and Wenxing Fu. Obstacle avoidance algorithm for multi-uav flocking based on artificial potential field and dubins path planning. In *2019 IEEE International Conference on Unmanned Systems (ICUS)*, pages 593–598, 2019.

[12] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[13] Hamal Marino, Paolo Salaris, and Lucia Pallottino. Controllability analysis of a pair of 3D Dubins vehicles in formation. *Robotics and Autonomous Systems*, 83:94–105, 2016.

[14] Nicola Mimmo and Marco Frego. An internal model control system for clothoid tracking. In *2025 IEEE 64th Conference on Decision and Control (CDC)*, pages 6382–6387, 2025.

[15] J. Ny, E. Feron, and E. Frazzoli. On the Dubins Traveling Salesman Problem. *IEEE Trans. on Automatic Control*, 57(1):265–270, 2012.

[16] G. Parlangeli, D. De Palma, and R. Attanasi. A novel approach for 3PDP and real-time via point path planning of Dubins' vehicles in marine applications. *Control Engineering Practice*, 144:105814, 2024.

[17] Patrick Pastorelli, Simone Dagnino, Enrico Saccon, Marco Frego, and Luigi Palopoli. Fast Shortest Path Polyline Smoothing With G^1 Continuity and Bounded Curvature. *IEEE Robotics and Automation Letters*, 10(4):3182–3189, 2025.

[18] Mattia Piazza, Enrico Bertolazzi, and Marco Frego. A Non-Smooth Numerical Optimization Approach to the Three-Point Dubins Problem (3PDP). *Algorithms*, 17(8), 2024.

[19] J. Reeds and L. Shepp. Optimal paths for a car that goes forwards and backwards. *Pac. J. Math.*, 145(2):67–93, 1990.

[20] Enrico Saccon and Marco Frego. A Machine Learning Approach for the Three-Point Dubins Problem (3PDP). *Symmetry*, 17(12), 2025.

[21] Armin Sadeghi and S. L. Smith. On efficient computation of shortest Dubins paths through three consecutive points. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 6010–6015, 2016.

[22] Ketan Savla, Emilio Frazzoli, and Francesco Bullo. Traveling salesperson problems for the Dubins vehicle. *IEEE Transactions on Automatic Control*, 53(6):1378–1391, 2008.

[23] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[24] Chaojie Yang, Lifen Liu, and Jiang Wu. Path planning algorithm for small uav based on dubins path. In *2016 IEEE International Conference on Aircraft Utility Systems (AUS)*, pages 1144–1148, 2016.