2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Convolutional Vision Transformer as a Path Following Controller for Omnidirectional Robots

Sandesh Hiremath[1], Cheng-Yi Huang[2], Argtim Tika[3] and Naim Bajcinca[4]

***Abstract*— A novel deep neural network (DNN) based controller for omnidirectional robots is proposed. The controller decomposes the prescribed reference path, corresponding to a fixed prediction horizon, into multiple paths of shorter horizons. This implicitly enforces a Hankel structure in the input and consequently also on the output. Taking advantage of this, a convolutional vision transformer model is used to realize the controller which is then trained to predict state and controls over multiple prediction horizons. Model training is performed in a self-supervised manner using a synthetic dataset. The proposed controller is shown to be more efficient than a model designed for a single prediction horizon. In comparison to a model predictive controller, the proposed approach exhibits competitive performance in path following tasks and is three times faster on average for the same prediction length.**

I. INTRODUCTION

Path following control is a common technique used in autonomous systems to guide a system along a predefined path. In this work, we propose a deep neural network (DNN) based path following feedback controller to enable a mobile robot to follow the given planar geometric path at the maximum possible speed. Model predictive control (MPC) is a widely used optimal control method for the design of path following controllers. MPC has been applied to a variety of applications, including mobile robots [1], autonomous driving cars [2], and underactuated vehicles [3]. MPC is often preferred over other algorithms such as, backstepping [4] and feedback linearization [5], because it can handle complex constraints and is more robust to disturbances. Recent research has focused on developing MPC-based path following controllers that can achieve minimum-time tracking. One such approach is model predictive contouring control (MPCC) [6], which minimizes the distance to the reference path while maximizing the progress along it. Recently, an automated-tuning, maximum velocity MPC was proposed in [7]. The main challenges in developing MPC-based path following controllers are the complexity of the optimization problem and the real-time implementation requirements [8].

Profound achievements of deep neural networks (DNNs) in the field of computer vision have exerted a considerable influence on diverse scientific domains ([9], [10], [11]). This widespread success can be largely attributed to their exceptional capacity for approximating intricate non-linear dynamics. Evidently, the adoption of DNNs for controller design has recently gained significant traction. Noteworthy among these applications is the work of [12], where recurrent neural networks (RNNs) was used with online training for trajectory tracking tasks. A novel paradigm proposed in [13] integrates neural-network models with a model predictive control (MPC) framework, harnessing the capabilities of NNs to capture highly non-linear dynamics. Similarly, [8] incorporate RNNs into the MPC scheme to approximate locally feasible solutions in dynamic environments, thereby enhancing real-time navigation performance. In contrast to these methodologies, [14] and [15] consider stochastic dynamical systems, and propose a RNN-based solution for stochastic optimal control problems. The latter additionally addresses dynamic state constraints using barrier functions.

A common aspect among the above methods is their focus on control generation for a single prediction horizon. For long input sequences, traditional RNN models begin to deteriorate. Recent developments in deep learning has led to the development of transformer models [16] that are adept at analyzing extended sequential data and thus enable predictions spanning longer horizons. Organizing sequential data into Hankel matrix structures facilitates the detection of patterns across multiple prediction horizons, thereby opening the opportunity for leveraging vision-based transformers. Convolution vision transformer (CvT) models, inspired by the success of convolutional neural networks (CNNs) in image-related tasks [17], [18], combine CNN features such as local receptive fields, shared weights, and spatial subsampling with attention-based information retrieval mechanisms for superior spatio-temporal input processing [19].

Motivated by these advantages, we propose using a CvT to handle sequential data. The usage of CvT is further made possible because of restructuring the reference path (sequence) into an image having Hankel block structure. A related approach, albeit in a different context involving data-driven predictive control, was employed in [20] to predict the outcomes of an unidentifiable system using a GRU-based RNN model. By adapting these principles, we present a novel controller model for the problem of path following. This model performance is similar to an MPC in terms of accuracy and robot behavior while offering faster and consistent execution times.

The paper is organized as follows. The path following problem and a corresponding MPC is presented in Section II. Next we describe the procedure for implementing a DNN based predictive controller in Section III. The developed model is implemented on an actual omnidirectional robot and the experimental results are discussed in Section IV. The paper concludes with a brief summary in Section V.

[1]Sandesh Hiremath, [2]Cheng-Yi Huang, [3]Argtim Tika and [4]Naim Bajcinca are with the Department of Mechanical and Process Engineering, Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau (RPTU), Germany. {sandesh.hiremath, chengyi.huang, argtim.tika, naim.bajcinca}@mv.rptu.de

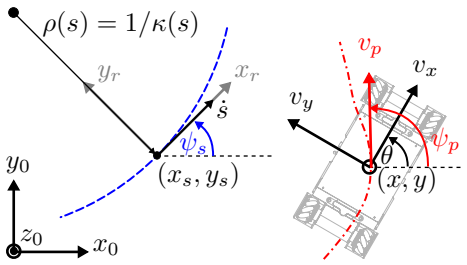


Fig. 1: Path following problem: (x_r, y_r) represents the two dimensional Frenet-Serret frame attached to a reference point, and (x_m, y_m) the reference frame attached to the mobile robot.

II. Path Following MPC Problem

A. Robot kinematic model

For a given parameterized reference path $\mathrm{p}_s(s) \in \mathbb{R}^2$

$$\mathrm{p}_s(s) = \begin{bmatrix} x_s(s) & y_s(s) \end{bmatrix}^\top : \mathcal{S} \to \mathbb{R}^2 , \tag{1}$$

with the path parameter $s \in \mathcal{S} \subseteq \mathbb{R}$, the goal of a path following feedback controller is to make the robot converge to the reference path and follow it as accurate as possible. Considering an omnidirectional mobile robot, the authors have presented in [7] a predictive-based path following controller using a kinematic prediction model with the tracking error dynamics defined in the Frenet-Serret frame. Referring to Fig. 1, let $\mathrm{v} = [v_x, v_y]^\top$ denote the robot velocity vector relative to the local body-attached robot frame and $\nu = [\nu_x, \nu_y]^\top$ the robot velocity relative to the inertial frame of reference $(x_0 y_0 z_0)$. With the state vector $\mathrm{x} = [x, y, \theta]^\top$, containing the robot position $\mathrm{p}_p := [x, y]^\top$, orientation θ, and the orientation ψ_p of the velocity vector with respect to the inertial frame, the nonlinear kinematic prediction model $\dot{\mathrm{x}}(t) = \mathrm{f}(\mathrm{x}, \mathrm{u})$ is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \nu_x \\ \nu_y \\ \omega \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{2}$$
$$v_p = \sqrt{v_x^2 + v_y^2}, \quad \psi_p = \arctan(\frac{v_y}{v_x}) + \theta.$$

Here, the input vector is given by $\mathrm{u} = [\mathrm{u}_v^\top, \mathrm{u}_\omega]^\top$, with $\mathrm{u}_v = [v_x, v_y]^\top$ and $\mathrm{u}_\omega = \omega$. Due to the omnidirectional kinematics, which imposes no additional velocity constraints on the robot's chassis, the robot is able to move in any direction in a two-dimensional plane. Consequently, the robot orientation θ can be entirely decoupled from the orientation ψ_p of the reference velocity vector. The reference path as function of the path parameter s is represented by the vector $\mathrm{x}_{\mathrm{ref}}(s) = [x_s, y_s, \theta_d]^\top$.

Since the path parameter is time dependant, there exist different approaches to chose its time evolution. Following the idea of maximum-speed reference path tracking, presented in [7], the dynamics of the path parameter is chosen as

$$\dot{s}(t) = f_s(\mathrm{x}, \psi_p, v_p) = \frac{v_p \cos(e_\psi(s))}{1 - e_y(s)\kappa(s)} . \tag{3}$$

Here, $\kappa_s(s) = \|\mathrm{p}_s(s)''\|_2$ denotes the curvature of the path, $e_\psi(s) = (\psi_p - \psi_s(s))$ represents the alignment error between the robot and the reference velocity vector, and

$$e_y(s) = -(x - x_s(s))\sin(\psi_s(s)) + (y - y_s(s))\cos(\psi_s(s))$$

denotes the lateral error in the Frenet-Serret frame. Note that ψ_s is given as

$$\psi_s(s) = \arctan\left(\frac{y_s'(s)}{x_s'(s)}\right),\ x_s'(s) = \frac{\partial x_s(s)}{\partial s}, y_s'(s) = \frac{\partial y_s(s)}{\partial s}.$$

The initial value for the path parameter can be found by computing the minimum distance between the actual robot position p_p and the parameterized reference curve $\mathrm{p}_s(s)$, which requires solving the following optimization problem

$$s_0^* = \arg\min_{s \in \mathcal{S}} \|\mathrm{p}_p - \mathrm{p}_s(s)\|_2 . \tag{4}$$

B. Model Predictive Control

The introduced MPC-based path following controller minimizes the error to a given reference path while aiming to maximize the covered path distance, leading to a maximum speed path following control strategy. Let $\mathrm{u}_N = [\mathrm{u}^\top(0), \cdots, \mathrm{u}^\top(N-1)]^\top$, $\mathrm{x}_N = [\mathrm{x}^\top(1), \cdots, \mathrm{x}^\top(N)]^\top$ and $\mathrm{y}_N = [\mathrm{x}_{\mathrm{ref}}^\top(0), \cdots, \mathrm{x}_{\mathrm{ref}}^\top(N-1)]^\top$ denote the input and state vector along the prediction horizon N, respectively. The objective function subject to minimization is then given as

$$J(\mathrm{u}_N; \mathrm{x}_0(t), \mathrm{y}_N(t), s_0) = \sum_{i=1}^{N} \|\mathrm{x}(i) - \mathrm{x}_{\mathrm{ref}}(i)\|_{\mathrm{Q}_x}^2 - \sum_{i=0}^{N-1} q_v v_p(i)$$
$$+ \sum_{i=0}^{N-2} \|\Delta \mathrm{u}_\omega(i)\|_{\mathrm{q}_{du}}^2 + \sum_{i=0}^{N-1} \left(\|\mathrm{u}_\omega(i)\|_{\mathrm{q}_u}^2 + \|\Delta \mathrm{x}(i)\|_{\mathrm{Q}_{dx}}^2 \right) \tag{5a}$$

where $\Delta \mathrm{w}(i) = \mathrm{w}(i+1) - \mathrm{w}(i)$, for $\mathrm{w} \in \{\mathrm{x}, \mathrm{u}_\omega\}$. The objective function J is a function of the decision variables u_N, state feedback $\mathrm{x}_0(t)$ obtained at time t, the initial path parameter s_0, and some fixed positive definite matrices $\mathrm{Q}_{(\cdot)}$. In particular, we have that $\mathrm{Q}_x \in \mathbb{R}^{3\times3}$, $\mathrm{Q}_{dx} \in \mathbb{R}^{3\times3}$, and $\mathrm{q}_u, \mathrm{q}_{du} \in \mathbb{R}$ represent the weighting matrices and coefficients for the state and change in state, input and change in input, respectively. The coefficient $q_v > 0$ represents the weight for the robot velocity. Accordingly, the MPC-based path following controller reads

$$\mathrm{u}_N^* = \underset{\mathrm{u}_N}{\operatorname{argmin}}\ J(\mathrm{u}_N; \mathrm{x}_0(t), \mathrm{y}_N(t), s_0) \tag{6}$$
$$\text{s.t.} \quad \mathrm{x}(i+1) = \mathrm{x}(i) + T_s\, \mathrm{f}(\mathrm{x}(i), \mathrm{u}(i)),\ \mathrm{x}(0) = \mathrm{x}_0(t) \tag{6a}$$
$$s(i+1) = s(i) + T_s\, f_s(\mathrm{x}(i), \psi_p(i), \mathrm{u}(i)) \tag{6b}$$
$$s(0) = s_0(t), \quad s(i) \in \mathcal{S}$$
$$\underline{\mathrm{u}} \le \mathrm{u}(i) \le \bar{\mathrm{u}}\ ,\ \underline{\mathrm{x}} \le \mathrm{x}(i) \le \bar{\mathrm{x}} \tag{6c}$$

with the iteration index $i \in \{0, \ldots, N-1\}$, and the initial conditions x_0, s_0. The optimization is performed for the input variables u with the upper and lower bounds as expressed by the inequality constraints (6c). The equality constraints (6a) and (6b) represent the discretized dynamic equations (2) and (3) by applying the forward Euler integration method. The robot state is also bounded to some upper $\bar{\mathrm{x}}$ and lower $\underline{\mathrm{x}}$.

III. DNN based Implementation of MPC

In this section, we introduce a novel learning based control algorithm for the problem of path following for an omnidirectional robot based on the above MPC formulation (6). Given $\mathbf{x} := (\mathrm{x}, \mathrm{y}_N)$, where x is the current state of the vehicle, y_N is the reference trajectory, the solution u_N^* to the OCP (6) can be written as a mapping $\mathbf{x} \mapsto \mathcal{M}(\mathbf{x}) =: \mathrm{u}_N^*$.

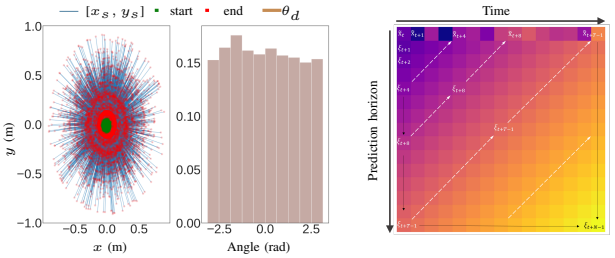
With this perspective, the objective is to implement a model that is able to learn the control synthesis fuction $\mathcal{M}(\mathbf{x})$. Thus, the idea is to develop a statistical estimator $\mathfrak{M}$ that approximates the true function $\mathcal{M}$. Due to good approximation power [21], [22], [23], high computational efficiency and flexibility of neural networks (NN) we shall implement $\mathfrak{M}$ using a suitable NN model. The so obtained network is given a name called *DonkeyNet* which is inspired from the name of the omnidirectional robot *DonkeyMotion*.

A. Network design

Although the goal of the network is to statistically approximate the MPC controller, i.e., to generate the optimal controls $\mathrm{u}_N^*(t)$ for the given input data comprising of the robot state $\mathrm{x}(t)$ and reference path $\mathrm{y}_N(t) := [\mathrm{y}_t, \cdots, \mathrm{y}_{t+N-1}] = [\mathrm{x}_{\text{ref}}(t), \cdots, \mathrm{x}_{\text{ref}}(t+N)]$ for the current time t, we propose a novel approach to feed the network T number of consecutive reference paths. For a given $\mathrm{y}_N(t)$, with fixed N, instead of naively reshaping it in row-major or column-major matrix, we propose to arrange it in the form of a Hankel (block) matrix structure with T columns and $N-T+1$ rows. We represent this rearrangement via the mapping $\mathrm{y}_N(t) \mapsto \mathcal{H}(\mathrm{y}_N(t))$ given as

$$\mathcal{H}(\mathrm{y}_N(t)) = \begin{bmatrix} \mathrm{y}_t & \mathrm{y}_{t+1} & \cdots & \mathrm{y}_{t+T-1} \\ \mathrm{y}_{t+1} & \mathrm{y}_{t+2} & \cdots & \mathrm{y}_{t+T} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{y}_{t+N-T} & \mathrm{y}_{t+N-T+1} & \cdots & \mathrm{y}_{t+N-1} \end{bmatrix}.$$

For the sake of implementational convenience, going forward we fix $T := \frac{N+1}{2}$ to get that the number of columns T is equal to the number of rows $N-T+1$. Thus, $\mathcal{H}(\mathrm{y}_N(T))$ is a block-square matrix. Here every j-th column vector represents the reference path $\mathrm{y}_T(t+j)$ for the control prediction at $t+j$ time step. Since the robot dynamics is continuous we require that $\mathrm{p}_p(t+1) \approx \mathrm{p}_p(t)$, i.e., $\mathrm{y}_T(t+1) \approx \mathrm{y}_T(t)$. In particular, we have that $\mathrm{y}_T(t+1)[0:T-2] = \mathrm{y}_T(t)[1:T-1]$. Thus, $\mathrm{y}_T(t+k)$ can be obtained by shifting $\mathrm{y}_T(t)$ k steps in the direction of robot motion. Accordingly, for a given reference path $\mathrm{y}_N(t)$, localized with respect to the robot current state $\mathrm{x}(t)$, we take $(\mathrm{x}(t), Y_N(t))$ with $Y_N(t) := \mathcal{H}(\mathrm{y}_N(t))$ as the input to the network. The novel input structure facilitates us to design a network that is capable of solving T number of optimal control problems at once. This is to say that the network $\mathfrak{M}$ takes $\mathrm{x}(t)$ and $Y_N(t)$ as input and produces $\hat{U}_N(t) \in \mathbb{R}^{T\times T\times 3}$ as the control input for the robot. Furthermore, $\hat{U}_N(t)$ preserves the Hankel structure of $Y_N(t)$. As a consequence $\hat{U}_N(t)$ can be decomposed as a sequence of T controls $\hat{\mathrm{u}}_T(r)$, for $r \in \{t, \cdots, t+T-1\}$, that corresponds (ideally also equal) to the optimal control $\mathrm{u}_T^*(r)$ obtained at time point r for the prediction horizon $[r, r+T-1]$. Succinctly, $U_N(t)$ is given as $U_N(t) = [\hat{\mathrm{u}}_T(t), \cdots, \hat{\mathrm{u}}_T(t+T-1)] \approx [\mathrm{u}_T^*(t), \cdots, \mathrm{u}_T^*(t+T-1)]$. Based on this philosophy, the network is designed to have image as both input and output. Due to the sequential dependency along the prediction horizon and the Hankel structure, we propose to use a network that enables to capture both sequential and spatial features in the input. To this end, as motivated in Section I, we shall use a CvT network for synthesizing the predictive controls, by making use of the architecture proposed in [19]. Although, originally proposed for classification task, we use CvT for the generation problem by sequentially composing the MLP head with regression layers followed by Tanh activation layer. The latter is specifically required to enforce the box constraints on the predicted states and controls (6c).



(a) Dataset samples: reference paths p_s (left) and desired orientation θ_d (right).

(b) Hankel structure of the input to the DNN.

Fig. 2: Dataset $\mathbb{X}$ (left) and sample input data (right).

B. Dataset generation

Keeping in mind the omnidirectional capability of the robot, the dataset is obtained by randomly generating different reference paths and target orientations for the robot. The generated dataset, $\mathbb{X} = \{\xi^k\}_{k=1}^M$, is as show in Fig. 2a. The samples ξ are obtained via the following systematic steps: 1) Generate a set $\mathcal{Y}$ of different parametric curves $y_s^j = [\mathrm{p}_s^j(s), \psi_s^j(s), \theta_d^j, s^j, \kappa_s^j(s)]^\top$ with $\mathcal{S} \ni s \mapsto \mathrm{y}^j(s) \in \mathbb{R}^6$ for each $j \in \mathbb{N}$. As a result $\mathcal{Y} = \{\mathrm{y}^j(\cdot)\}_{j=1}^L$ represents the set of all trajectories that are feasible for the robot. As per the physical conditions of the lab area, we have considered $L = 100$ and the resulting length of each curve is in the range of 1m to 20m. Lastly, since the robot orientation is independent of the path characteristics, we generate desired orientations θ_d by randomly sampling from $\mathcal{U}([-\pi, \pi])$. 2) Now let $j \sim \mathcal{U}\{1, \cdots, L\}$, $s_l \sim \mathcal{U}(\mathcal{S})$, $n \sim \mathcal{U}([0,1])$ and generate a sequence $s_n^l = (s_l, s_l+\tau, \cdots, s_l+i\tau, \cdots, s_l+N\tau)$ where $\tau := n/N$. From the sequence s_N^l we obtain a sample reference path $\tilde{\xi}^{j,l,n} = \mathrm{y}^j(s_n^l)$ of length N. Since we uniformly sample over all j, s_l and n, the permutation of indices j, l, n are re-enumerated and denoted as k. Thus, $\tilde{\xi}^{j,l,n}$ is relabeled as $\tilde{\xi}^k$ to represent the k-th sample of the dataset. 3) For each $\tilde{\xi}^k$ we reset the $\theta_d(s_n^l) = \theta_d(s_l+n)$, i.e., θ_d remains constant along the given trajectory by setting to the end value. 4) Taking N to be an odd number, we reshape the sample $\tilde{\xi}^k \in \mathbb{R}^{N\times 6}$ into a 3D tensor $\hat{\xi}^k \in \mathbb{R}^{T\times T\times 6}$ via the Hankel mapping introduced above, i.e., $\hat{\xi}^k = \mathcal{H}(\tilde{\xi}^k)$. 5) to incorporate the current state x^k of the robot corresponding to each k-th sample, we obtain it as a random perturbation of the start value of the reference. Consequently, $\mathbb{R}^6 \ni \tilde{\mathrm{x}}^k = \hat{\xi}_1^k + z_1^k$, where $z_1^k \sim \mathcal{N}(0,1)$. As mentioned above, since we intend to perform T batch predictions, we take $\mathrm{x}^k = [\tilde{\mathrm{x}}_1^k, \cdots, \tilde{\mathrm{x}}_T^k]^\top = [\hat{\xi}_1^k + z_1^k, \cdots, \hat{\xi}_T^k + z_T^k]^\top \in \mathbb{R}^{T\times 6}$. 6) Lastly we obtain ξ^k by $\xi^k = \tilde{\xi}^k$ and $\xi^k[0,:] = \mathrm{x}^k$, which can be visualized as shown in Fig. 2b. The steps from

2) to 6) of obtaining ξ^k from y_N^k and x^k is symbolically represented via the mapping $(\mathrm{x}^k, \mathrm{y}_N^k) \mapsto \mathfrak{D}(\mathrm{x}^k, \mathrm{y}_N^k) =: \xi^k$.

C. Training the DonkeyNet $\mathfrak{M}$

Let $\mathfrak{M}(\cdot;\vartheta)$ denote a parameterized estimator of the controller $\mathcal{M}$. Given a set $\mathbb{X}$ of M independent samples ξ^k consisting of the robot state and reference paths, the task is to estimate $\hat{\vartheta} \approx \vartheta^*$ that produces an optimal control $\hat{U}_N^k := \mathfrak{M}(\xi^k;\hat{\vartheta})$ on average for all $\xi^k \in \mathbb{X}$. The synthesized control $\hat{U}_N^k$ is such that for every input $\xi^k = \mathfrak{D}(\mathrm{x}^k(t), \mathrm{y}_N^k(t))$, it propagates the initial state x_0^k to follow the prescribed reference trajectory $\mathrm{y}_N^k(t)$. More precisely, the parametric estimate $\hat{U}_N = \mathfrak{M}(\hat{\xi};\hat{\vartheta})$ of the optimal control $\mathcal{H}(\mathrm{u}_N^*)$ should be such that $\hat{X}_N(t) = G(\mathrm{x}(t), \hat{U}_N)$, on average, is equal to $Y_N(t) = \mathcal{H}(\mathrm{y}_N(t))$. Here, G denotes the solution operator (e.g., discrete Euler/RK4 integrator) for the robot kinematic model (2). Based on the aforementioned context, $\mathfrak{M}$ shall be designed based on the method of maximum log-likelihood estimation. To this end, the loss (log-likelihood) function is given as:

$$\mathfrak{L}(\vartheta;\hat{X},\hat{U}) = \frac{1}{m}\sum_{k=1}^{M} \gamma_1 \mathbb{L}_x^k + \gamma_2 \mathbb{L}_u^k + \gamma_4 \mathbb{L}_H^k \tag{7}$$

$$\mathbb{L}_x^k := \sum_{i=1}^{T-1} \lambda_x^i (\|\hat{X}_i^k(t) - Y_i^k(t)\|_{R_x}^2 + \|\hat{X}_i^k(t) - \hat{X}_{i-1}^k(t)\|_{R_x}^2),$$

$$\mathbb{L}_u^k := \sum_{i=1}^{T-1} \lambda_u^i \|\hat{U}_i^k(t)\|_{R_u}^2 + \lambda_{du}^i \|\hat{U}_i^k(t) - \hat{U}_{i-1}^k(t)\|_{R_u}^2,$$

$$\mathbb{L}_H^k := \sum_{j=0}^{T-1} \lambda_{H_x}^i \sum_{i=1}^{T-1} \|\hat{X}_{i+1}^k(t+j) - \hat{X}_i^k(t+j+1)\|_{R_x}^2 + \lambda_{H_u}^i \|\hat{U}_{i+1}^k(t+j) - \hat{U}_i^k(t+j+1)\|_{R_u}^2.$$

Altogether, the learning problem of the control synthesis function $\mathcal{M}$ can be written as $\hat{\vartheta} = \underset{\vartheta}{\operatorname{argmin}}\, \mathfrak{L}(\vartheta;\mathbb{X})$. In order to solve the statistical optimization problem, we use the popular 1st order stochastic optimization schemes such as SGD, Adam, AdaMax [24]. Since these schemes involve gradient computation, they are efficiently achieved by back-propagating the loss (7). Based on this the parameters ϑ are updated suitably along the decreasing direction of the expected loss. Altogether, we obtain an offline training procedure of the DNN model $\mathfrak{M}$. The exact sequence of steps performed for calculating the loss components are described in Algorithm 1. In there $\hat{X}^j \in \mathbb{R}^{(T+1)\times T\times 5\times m}$, $\hat{Y}^j \in \mathbb{R}^{T\times T\times 6\times m}$ and $\hat{U}^j \in \mathbb{R}^{T\times T\times 3\times m}$. For $Z \in \mathbb{R}^{A\times B\times r\times m}$, the index notation $Z_{a:b}(p:q)$ yields $z \in \mathbb{R}^{(b-a)\times(q-p)\times r\times m}$ for $b > a, q > p$, where z is a sub-matrix of Z from row-index a up till b and column-index p up till q. With this the core steps of the training are the following lines:

line 3: the network $\mathfrak{M}$ takes a batch of inputs $\hat{Y}^j$, comprising of robot state x_0 and reference path y_N stacked as a multi-channel image (cf. Fig. 2b), and generates a batch of controls represented as $\hat{U}^j$.

line 4: Read the robot state $\hat{X}_0^j(t)$ stored in the first index of the input $\hat{Y}^j$.

line 5: Compute the robot state along the first horizon using only the controls from the first column of $\hat{U}^j$, i.e., $\hat{U}_{0:T}^j(t)$. Thus predicted robot state is stored in $\hat{X}_{0:T+1}^j(t)$, i.e., in the first column of $\hat{X}^j$.

line 6: Due to Hankel structure, the prediction of state in the first horizon should serve as initial state for the predictions along the subsequent horizons. Thus, we now use $\hat{X}_{1:T}^j(t)$ as the initial states of the robot in the (near) future and predict the robot states along the future ($[t+1$ to $t+T-1]$) prediction horizons. For this we now use rest of the controls $\hat{U}_{0:T}^j(t+1:t+T)$, i.e, the controls from the 2nd column on wards of $\hat{U}^j$. Altogether, the predicted states are now stored in $\hat{X}_{0:T+1}^j(t+1:t+T)$ which yields a block matrix of robot states across $T-1$ consecutive prediction horizons. Altogether, the computed $\hat{X}_{0:T+1}^j(t:t+T)$, $\hat{U}_{0:T}^j(t:t+T)$ along with reference path $\hat{Y}_{0:T}^j(t:t+T)$ are then used for loss computation as per (7). The corresponding vector, $\mathfrak{C}$, of hyperparameters is given by

$$\mathfrak{C} := [R_x, R_u, \lambda_x, \lambda_u, \gamma_1, \gamma_2, \gamma_3, \gamma_5, T_s, T] \tag{8}$$

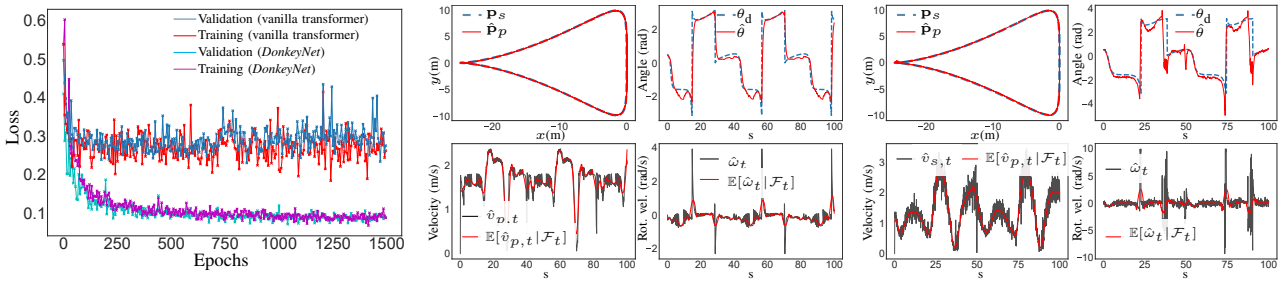
$$R_x = \mathrm{diag}[200, 200, 0.1, 100, 0.1], \quad \lambda_{H_x}^i = \frac{1}{3} e^{iT_s\lambda_x},$$

$$R_u = \mathrm{diag}[100, 100, 5], \quad \lambda_{H_u}^i = \frac{1}{6} e^{iT_s\lambda_u},\ \lambda_u = .01,$$

$$\lambda_x^i = e^{iT_s\lambda_x},\ \lambda_u^i = e^{iT_s\lambda_u},\ \lambda_{du}^i = \lambda_u^i,\ \lambda_x = -.01,$$

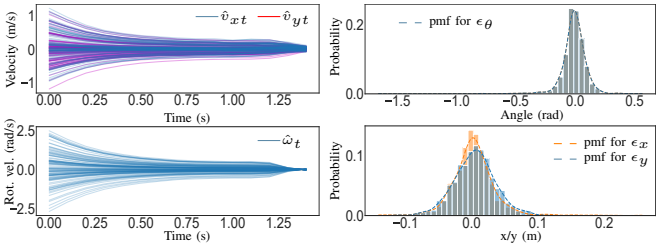
$$\gamma_1 = 4\cdot 10^{-3},\ \gamma_2 = 10^{-4},\ \gamma_3 = 10^{-5}, \gamma_4 = 5\cdot 10^{-4},$$

The dataset $\mathbb{X}$, consisting of $M = 10K$ samples, was randomly partitioned, in the ratio 80:10:10, into training, testing and validation sets. Each partitioned data-subset is then used for training, testing and validation tasks, respectively. The training task was performed on a Tesla V100-DGXS-32GB GPU, for $E = 1500$ epochs. Looking at the loss curves in Fig. 3a, we see that the train and validation losses follow a similar decay profile, thus indicating good training with negligible over-fitting. Also in comparison to the vanilla transformer, the CvT based model has achieved better loss reduction. Next we evaluated the network output on the test set and the results are as show in Fig. 4a - Fig. 5. From Fig. 4a we see that $\hat{v}_x, \hat{v}_y$ and $\hat{\omega}$ have nice decay property, thus indicating stable control behavior. From Fig. 4b we see that the error distribution of the robot position and orientation for the terminal reference value has mean almost zero and standard deviation less than 0.15. Also, from Fig. 6a and Fig. 6b we see that both position and orientation tracking are qualitatively good. Next we tested the network in a closed loop simulation, for which we choose a closed path with singularity at $(0,0)$ as reference. We tested both the models and the results are as shown in Fig. 3b and Fig. 3c. For both models we see that position tracking is qualitatively good and quite similar, but the CvT based model has a much better orientation tracking with smooth orientation profile. Based on the plots for the control values, (lower plots of Fig. 3b and Fig. 3c) we see that the CvT based model not only generates smoother controls but the $\hat{v}_p$ is on average higher, thus able to cover more distance in fixed time. Furthermore, the predicted states and controls approximates the Hankel structure fairly well (as seen in from Fig. 5), thus predicted controls, for N



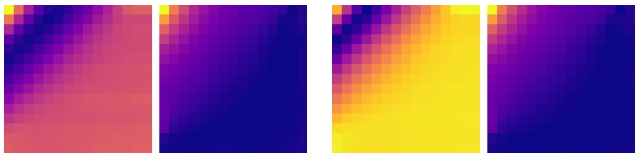
(a) Training and validation losses (b) $\mathfrak{M}$ based on convolutional transformer model (c) $\mathfrak{M}$ based on vanilla transformer model

Fig. 3: Training losses and closed loop simulation.



(a) Predicted control values $(\hat{v}_x, \hat{v}_y)$ and $\hat{\omega}$ (b) Path and orientation error distribution of states and orientation

Fig. 4: Generated controls (left) and error distribution.



(a) Sample 1 $(\hat{X}^1, \hat{U}^1)$ (b) Sample 2 $(\hat{X}^2, \hat{U}^2)$

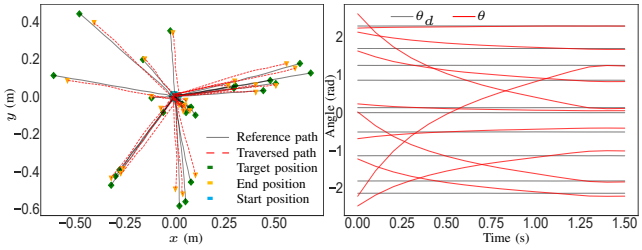
Fig. 5: Sample prediction of $\mathfrak{M}$, state and control vectors, across multiple horizons which are depicted as images.

long horizon, is almost surely optimal for $T < N$ number of real time steps. Architecture-wise, the both models have approximately 19M trainable parameters. However, the CvT based model has 82M computation operations while the other has only 19M. Thus, the latter model is on average 8 ms faster. Since both models have the computation time in the range $10-30$ ms range, this difference is quite negligible compared to the improvement in the performance. We also note that a GRU based RNN based model was also experimented with and it showed below par performance and was thus abandoned. Based on this offline analysis we proceed with integrating the *DonkeyNet* on the real robot and compare its performance with a high-fidelity MPC.

IV. Implementation and Results

A. Implementation details

Both MPC and *DonkeyNet* have been implemented and validated on an omnidirectional robot. The reference paths are generated using basis splines (B-splines) [25] in combination with a probabilistic roadmap (PRM) [26] based global planner. Initially, the PRM searches for collision-free connections between sampling points within a given static map of our laboratory. Subsequently, these identified sampling points are used as control points to generate the B-spline paths. The system architecture is realized by using Robot Operating System [27], as illustrated in Fig. 8. The implementation of



(a) Predicted sample paths (b) Predicted robot orientations

Fig. 6: Predicted paths (a) and orientation (b) of the robot.

Algorithm 1: *DonkeyNet* $\mathfrak{M}$

Given: $T_s = 0.1$, $N = 29$, $T = 15$, $\mathfrak{C}$: Model and cost parameters, m: batch size, M: total training samples, E: number of epochs.

Data: Dataset $\mathbb{X} = \{\xi^k\}_{k=1}^{M}$, consisting of M samples with $\xi^k = \mathfrak{D}(\mathrm{x}^k, \mathrm{y}_N^k)$.

for $j = 1, \dots E$ **do**
- $\hat{Y}^j_{0:T}(t : t+T) = [\xi^{k_1}, \cdot, \xi^{k_m}]$, $\xi^{k_l} \in \{\mathbb{X}\}$;
- $\hat{U}^j_{0:T}(t : t+T) = \mathfrak{M}(\hat{Y}^j_{0:T}(t : t+T); \hat{\vartheta}^j)$;
- $\hat{X}^j_0(t) = \hat{Y}^j_0(t)$;
- $\hat{X}^j_{0:T+1}(t) = G(\hat{X}^j_0(t), \hat{U}^j_{0:T}(t))$;
- $\hat{X}^j_{0:T+1}(t+1 : t+T) = G(\hat{X}^j_{1:T}(t), \hat{U}^j_{0:T}(t : t+T))$;
- **if** *Training* **then**
  - compute $\mathfrak{L}(\vartheta; \hat{X}, \hat{U})$ as per (7)
  - $\mathfrak{M}(\cdot; \hat{\vartheta}^j) \longleftarrow$ backpropagte($\mathfrak{L}$)

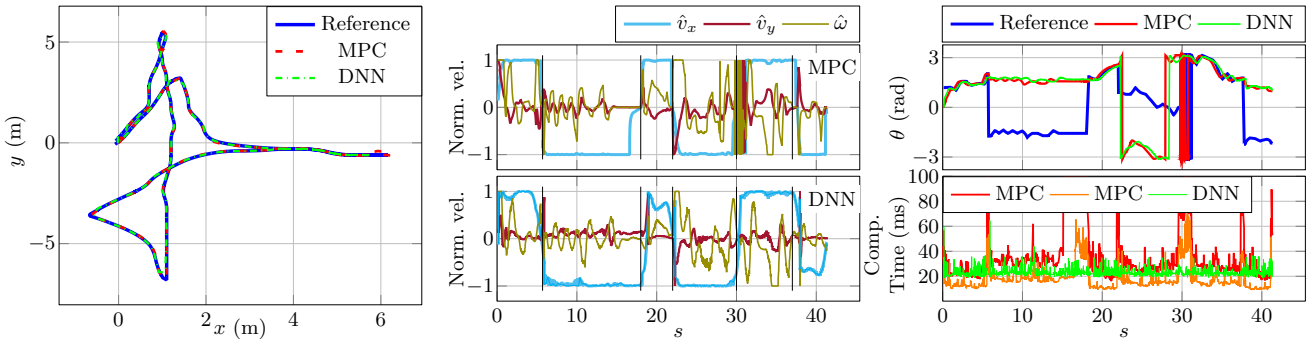
end

return $\hat{X}_{0:T}(t : t+T), \hat{U}_{0:T}(t : t+T)$.

MPC builds upon the work presented in [7] using the CasADi framework [28] with MA57 solver [29]. The *DonkeyNet* is implemented in PyTorch [30]. They are deployed on a CarPC equipped with Intel Core i7-8700T Processor 2.40 GHz clock rate and Nvidia GeForce GTX 1650 graphic card running on Ubuntu 18.04.6 LTS. Furthermore, to facilitate continuous navigation in the real environment, a state machine with three states: idle, execution, and switch, is implemented to guide the transition to a new reference path.

B. Experimental Results

The MPC and *DonkeyNet* ($\mathfrak{M}$) approaches are experimentally analyzed on identical reference path and the desired orientation θ_d is set to the tangential direction ψ_s of the reference path. To draw a clear comparison between the two,



I) Reference and driven robot path. II) Control inputs. III) Robot orientation and computation time.

Fig. 7: Comparison of MPC and *DonkeyNet* ($\mathfrak{M}$) performance for indoor path following task. The path consists of several singular points denoted by black vertical lines in II). Intuitively, these points represents a switch in the path direction. In II), we see the control inputs generated by both controllers MPC (top) and $\mathfrak{M}$ bottom. The θ profiles for MPC and $\mathfrak{M}$ along with the reference θ_d are depicted in the upper plot of III). The computation times for the both are indicated in the lower plot of III). For MPC with $N = 15$ (red) results in 27.28 ms on average and 71.16 ms with $N = 29$ (orange). The *DonkeyNet*, which is trained for $N = 29$, takes 24.04 ms on average.

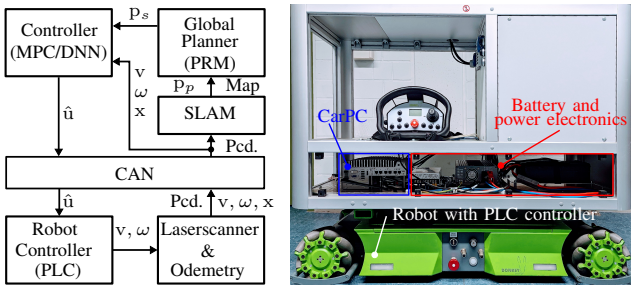


Fig. 8: Mobile robot platform with omnidirectional wheels.

data are plotted with respect to the arc length of the reference path. In Fig. 7 I), both controllers demonstrate satisfactory performance in tracking target paths, with the maximum velocity being nearly maintained throughout the path. It's noteworthy that, near the position $(x, y) = (6, -1)$, there's a noticeable divergence from the reference for the MPC solution. This observable lateral error is attributed to the jump in ψ_s from π to $-\pi$, as shown in Fig. 7 III) at $s = 30$. This causes the MPC solver to struggle to find a feasible solution which leads to exceptionally high computation time during this situation. On the contrary, the DNN is able to perform consistently even at these singular points which can be mainly attributed to the CvT based model, robust training, Hankel structure of the input/output and also due to the unwrapped ψ_s as the reference. Regarding the generated controls (see Fig. 7 II)), since the reference orientation θ_d is set to ψ_s, the determined optimal speed is primarily allocated to v_x, while v_y is kept low. It can also be observed that the synthesized controls from MPC are smooth, whereas the result from $\mathfrak{M}$ is relatively noisy. Nevertheless, the effect of this on the robot behavior is practically negligible which can again be attributed to the Hankel structure of the controls. Fig. 7 III) depicts the computation time for both controllers which are relatively competitive. For the MPC, throughout most of the reference path, the computation is around $10-30$ ms when $N = 15$ and $20-60$ ms when $N = 29$. However, near singular points of the path, there is significant surge in computation time. This is expected, since the nonlinear-solver struggles to find the new local minimizer which can potentially be far from the previous solution. This may further lead to cascading effect due to the increased sensitivity of the solver to the starting seed value. In contrast, $\mathfrak{M}$ is not just faster than MPC on average but is $\sim$3 times faster on average when comparing the two for the same prediction length $N = 29$. Furthermore, $\mathfrak{M}$ also ensures consistent computation times across the entire path, falling within a narrow range of $20-40$ ms. Thus, $\mathfrak{M}$ is able to overcome the challenges posed by the nonlinear model complexity inherent in the traditional OCPs, and thereby effectively reducing the demand for online optimization computations.

V. CONCLUSIONS

In this work, we have presented a novel DNN based controller for the path following problem of an omnidirectional mobile robot. The novel design of input and output format in the form of Hankel structure enabled the use of vision based transformer model as a controller. In comparison to the use of vanilla transformers, the use of CvT not only facilitated in achieving lower trainable-parameters to input-size ratio as well as computation-time to compute-operations ratio but also improved the quality of controls in terms of accuracy and smoothness. Implementation-wise our work also demonstrates that both approaches (MPC and *DonkeyNet*) can be seamlessly integrated with the global planner, allowing the robot to adeptly follow generated B-spline paths. While MPC excels in synthesizing smooth controls it is sensitive to model complexity and path discontinuities. On the other hand, *DonkeyNet* exhibits competitive accuracy with consistently faster computation times, thus robust to path discontinuities and nonlinear model complexities. Some ideas for improving the current model goes in the direction of variable horizon inputs and smoothening of the output. One idea for the latter would be to incorporate previous states as part of the input to obtain averaging effect on the synthesized controls.

VI. ACKNOWLEDGMENT

The authors would like to thank *imetron GmbH* for providing the omnidirectional mobile robot *DONKEYMotion* for testing and evaluation purposes.

REFERENCES

[1] K. Kanjanawanishkul and A. Zell, "Path following for an omnidirectional mobile robot based on model predictive control," in *2009 IEEE International Conference on Robotics and Automation*, pp. 3341–3346, 2009.

[2] T. Weiskircher, Q. Wang, and B. Ayalew, "Predictive guidance and control framework for (semi-)autonomous vehicles in public traffic," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 6, pp. 2034–2046, 2017.

[3] A. Alessandretti, A. P. Aguiar, and C. N. Jones, "Trajectory-tracking and path-following controllers for constrained underactuated vehicles using model predictive control," in *2013 European Control Conference (ECC)*, pp. 1371–1376, 2013.

[4] G. Raffo, M. Ortega, and F. Rubio, "Backstepping/nonlinear $\mathcal{H}_\infty$ control for path tracking of a quadrotor unmanned aerial vehicle," pp. 3356 – 3361, 07 2008.

[5] A. Banaszuk and J. Hauser, "Feedback linearization of transverse dynamics for periodic orbits," *Systems & Control Letters*, vol. 26, no. 2, pp. 95–105, 1995.

[6] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, "Model predictive contouring control for collision avoidance in unstructured dynamic environments," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 07 2019.

[7] A. Tika, S. Hiremath, and N. Bajcinca, "Predictive path following control for mobile robots with automatic parameter tuning," in *2023 European Control Conference (ECC)*, pp. 1–8, 2023.

[8] M. Mohaghegh, S.-A. Saeedinia, and Z. Roozbehi, "Optimal predictive neuro-navigator design for mobile robot navigation with moving obstacles," *Frontiers in Robotics and AI*, vol. 10, 2023.

[9] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," *SN Computer Science*, vol. 2, no. 3, p. 160, 2021.

[10] J. Schmidt, M. R. G. Marques, S. Botti, and M. A. L. Marques, "Recent advances and applications of machine learning in solid-state materials science," *npj Computational Materials*, vol. 5, no. 1, p. 83, 2019.

[11] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Science Advances*, vol. 3, no. 4, p. e1602614, 2017.

[12] J. Velagić, N. Osmic, and B. Lacevic, "Neural network controller for mobile robot motion control," *World Academy of Science, Engineering and Technology, International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering*, vol. 2, pp. 2471–2476, 2008.

[13] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, "Real-time neural-mpc: Deep learning model predictive control for quadrotors and agile robotic platforms," *IEEE Robotics and Automation Letters*, 2023.

[14] Exarchos, Ioannis and Theodorou, Evangelos A., "Stochastic optimal control via forward and backward stochastic differential equations and importance sampling," *Automatica*, vol. 87, pp. 159–165, 2018.

[15] S. A. Hiremath and N. Bajcinca, "Dnn based learning algorithm for state constrained stochastic control of a 2d cartpole system," in *2022 European Control Conference (ECC)*, pp. 1132–1139, 2022.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[18] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[19] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang, "Cvt: Introducing convolutions to vision transformers," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, (Los Alamitos, CA, USA), pp. 22–31, IEEE Computer Society, oct 2021.

[20] S. A. Hiremath, V. K. Mishra, and N. Bajcinca, "Learning based stochastic data-driven predictive control," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 1684–1961, 2022.

[21] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[22] H. T. Siegelmann and E. D. Sontag, "On the Computational Power of Neural Nets," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, (New York, NY, USA), pp. 440–449, Association for Computing Machinery, 1992.

[23] A. M. Schäfer and H. G. Zimmermann, "Recurrent Neural Networks Are Universal Approximators," in *Artificial Neural Networks – ICANN 2006* (S. D. Kollias, A. Stafylopatis, W. Duch, and E. Oja, eds.), (Berlin, Heidelberg), pp. 632–640, Springer Berlin Heidelberg, 2006.

[24] E. Hassan, M. Y. Shams, N. A. Hikal, and S. Elmougy, "The effect of choosing optimizer algorithms to improve computer vision tasks: a comparative study," *Multimedia Tools and Applications*, vol. 82, pp. 16591 – 16633, 2022.

[25] I. J. Schoenberg, *Contributions to the Problem of Approximation of Equidistant Data by Analytic Functions*, pp. 3–57. Boston, MA: Birkhäuser Boston, 1988.

[26] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[27] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system."

[28] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[29] I. S. Duff, "Ma57—a code for the solution of sparse symmetric definite and indefinite systems," *ACM Trans. Math. Softw.*, vol. 30, pp. 118–144, 2004.

[30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.