Neural network approximator for input allocation*

*Final project for course *Design Methods for Unmanned Vehicles*, prof.s D. Brunelli, D. Fontanelli, G. Moretti, L. Zaccarian.

Matteo Dalle Vedove
Department of Industrial Engineering
University of Trento, Italy
matteo.dallevedove@studenti.unitn.it

***Abstract*—With this project I investigated the viability of using deep neural-networks to approximate a optimal allocation law. Tests have been carried out on the ROtor graSPing Omnidirectional (ROSPO) platform [1] that provides a simple-enough, yet feature-full, testbed for propeller-driven vehicles. Experiments demonstrates the difficulty in accomplishing such task due to the high dimensionality of the function that needs to be approximated, showing that this approach is not satisfactory.**

I. Introduction

When working with unmanned aerial vehicles, we usually deal with over-actuated systems. In fact even the simplest quadcopter can use different actuator configurations to achieve the same applied wrench to the center of mass.

More complex drones, such the one presented in [2], might use even more rotors that can be tilded by means of another servo motor, increasing the degree of overactuation in the system.

Whenever dealing with redundant actuation, key is to define an optimal way to distribute the load between the different components in order to accomplish the goal while minimizing a cost index that usually is the required actuation tower, in order to improve battery lifetime.

The goal of this project is to evaluate the performance that can be achieved when using neural networks for approximating the optimal control allocation of an over-actuated system.

To tackle this problem I used the ROtor graSPing Omnidirectional (ROSPO) platform [1], Fig. 1, a system developed at LAAS-CNRS to evaluate allocation scheme in propeller-driven vehicles. Such system has in fact the advantage of having a planar kinematics that simplifies the analytical burden of the system's dynamics, but presents all issues that must be addressed when working with drones.

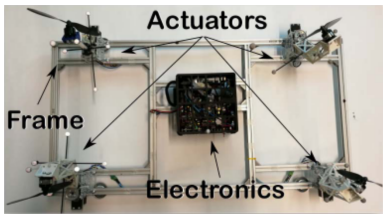

Fig. 1. photo of the ROSPO prototype, courtesy of [1].

The proposed idea is to use artificial neural networks trained offline on a dataset obtained by solving non-linear allocation problems; Khan et al. [3] reported promising results while solving allocation problem on an aerial vehicle, showing best regulation performance and lower evaluation time than common QP-based online solution.

This document will continue as follows: in Sec. II the ROSPO platform is described in more detail and its model is derived, Sec. III describes the control scheme that I used while Sec. IV reports some implementation details. Later Sec. V reports some simulation results and finally conclusions are summarized in Sec. VI.

II. System description and modelling

As described in [1], the ROSPO platform mainly consists in a rigid platform on top of which an arbitrary amount of actuator modules can be mounted.
Each of such module is a turret that is able to rotate on it's center with a stepper motor and on top of which a BLDC motor is used to drive propellers that will provide the thrust to move the system.
The moving frame is supported by means of omnidirectional passive spherical wheels.

A. Motor modelling

In contrast to what has been done in the reference paper, this simulation considers a more complex model of the BLDC motor that moves the propeller.

The proposed model is based on the equivalent single coil electrical circuit for the motor that's ruled by the following differential equation:

$$L\frac{di}{dt} = v - R\,i - e \tag{1}$$

where L, R are respectively the inductance and the resistance of the circuit, i is the current flowing, v is the voltage commanded to the circuit and e is the back-electromotive voltage due to the rotation of the motor itself. Called such speed Ω, it holds

$$\Omega = k_v\, e$$

By balancing the electrical power dissipated by the BLDC motor with the resulting mechanical power, the output torque T_m of the motor is

$$T_m = \frac{i\,e}{J_m}$$

TABLE I
COEFFICIENTS USED IN THE MOTOR'S MODEL.

constant	value
k_v	$750V/rpm$
L	$27.2mH$
R	0.7Ω
k_l	$6.4 \cdot 10^{-4}$
k_r	$1.2 \cdot 10^{-5}$
J_m	$2.8 \cdot 10^{-4} kg \cdot m^2$

From the body mechanics, the differential equation ruling the angular speed of the rotor is

$$J_m \frac{d\Omega}{dt} = T_m - T_r \tag{2}$$

where T_r is the (positive) resisting torque and J_m is the motor's inertia (that comprises also the inertia of the propeller); given that the resisting torque is given just by aerodynamic actions, it can be approximated as $T_r = k_r \Omega^2$.

Combining (1) and (2) and removing the different algebraic constraints, leads to the following system of ODEs ruling the behavior of the BLDC motor:

$$\begin{cases} L\frac{di}{dt} & = v - R\,i - \frac{\Omega}{k_v} \\ J_m \frac{d\Omega}{dt} & = \frac{i}{k_v} - k_r \Omega^2 \end{cases} \tag{3}$$

where Ω, i are the states of the system and v is the input. Finally the force of the propeller F_p is given by the state simply as

$$F_p = k_l \Omega^2 \tag{4}$$

where k_l is the lift coefficient.

Numerically, all constants have been obtained, whenever possible, directly from datasheet of propellers and motors present in the laboratories; when data were not available, they have been derived considering simple mechanics. Table I report all such values.

B. Friction force

The reference model embeds also the effects of friction forces on the spherical wheels; the continuous function used is based on [4] and is of the form

$$\boldsymbol{F}_{f,\boldsymbol{p}}(\boldsymbol{v}) = \begin{cases} \mu(|\boldsymbol{v}|)\frac{\boldsymbol{v}}{|\boldsymbol{v}|} & \boldsymbol{v} \neq 0 \\ 0 & v = 0 \end{cases} \tag{5}$$

where $\boldsymbol{v} \in \mathbb{R}^2$ is the velocity of the point $\boldsymbol{p}$ in the plane and μ is the continuous function defined as

$$\mu(s) = \gamma_1\big(\tanh(\gamma_2 s) - \tanh(\gamma_3 s)\big) + \gamma_4 \tanh(\gamma_5 s) + \gamma_6 s$$

In the provided model $\gamma_1, \gamma_2, \gamma_3, \gamma_6$ are set to zero to neglect both Stribeck effect and viscous dissipation. Since (5) is continuous, it's time derivative later used in the control law can be computed as

$$\dot{\boldsymbol{F}}_{r,\boldsymbol{p}}(\boldsymbol{v}) = \left(\left(\mathbf{I} - \frac{\boldsymbol{v}\boldsymbol{v}^\top}{\boldsymbol{v}^\top\boldsymbol{v}}\right)\mu\big(|\boldsymbol{v}|\big) + \frac{\boldsymbol{v}\boldsymbol{v}^\top}{|\boldsymbol{v}|}\mu'\big(|\boldsymbol{v}|\big)\right)\frac{\dot{\boldsymbol{v}}}{|\boldsymbol{v}|} \tag{6}$$

with

$$\mu'(s) = \gamma_4\big(1 - \tanh^2(\gamma_5 s)\big)\gamma_5 \tag{7}$$

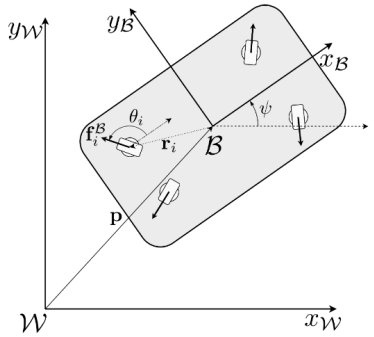

Fig. 2. schematic illustration of the ROSPO system; image courtesy of [1].

C. ROSPO modelling

The ROSPO system is described as a single rigid body, i.e. the changes in mass and inertias due to the turrets rotations have been disregarded. With this approximation in place the system's dynamic is described as

$$\begin{cases} m\ddot{\boldsymbol{p}}_{com} & = \mathbf{R}(\psi)\sum_i F_{p,i}\hat{\boldsymbol{u}}(\phi_i) + \sum_i \boldsymbol{F}_{r,\boldsymbol{p}_i} \\ J\ddot{\psi} & = \sum_i \boldsymbol{r}_i \times F_{p,i}\hat{\boldsymbol{u}}(\phi_i) + \sum_i \boldsymbol{r}_{\boldsymbol{p}_i} \times \boldsymbol{F}_{r,\boldsymbol{p}_i} \\ \dot{\phi}_i & = \delta_i \end{cases} \tag{8}$$

where $\hat{\boldsymbol{u}}(\phi_i) = \big(\cos(\phi_i), \sin(\phi_i)\big)^\top$ represents the direction where the force $F_{p,i}$ of the i-th propeller is applied, with ϕ_i the relative orientation of the turret w.r.t the body's local frame that's described by an orientation ψ; $\boldsymbol{r}_\circ$ are instead the vectors describing the distance of the point where the force is applied w.r.t. the center of mass of the ROSPO; finally m, J are the mass and the inertia of the system.
The rotation matrix

$$\mathbf{R}(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix}$$

is used to convert the force computed in the local frame of the body into the inertial ground reference frame.
The dynamics of the turret's attitude ϕ_i is linear w.r.t. the corresponding input δ_i since such element is moved by a stepper motor.

After reducing the system to a first order ODE the states $\boldsymbol{x}$ and input $\boldsymbol{u}$ are:

$$\boldsymbol{x} = \begin{pmatrix} x_{com} \\ y_{com} \\ \psi \\ v_{x,com} \\ v_{y,com} \\ \omega \\ \Omega_1 \\ i_1 \\ \phi_1 \\ \vdots \\ i_{N_t} \\ \phi_{N_t} \end{pmatrix} \in \mathbb{R}^{6+3N_t} \qquad \boldsymbol{u} = \begin{pmatrix} v_1 \\ \delta_1 \\ \vdots \\ v_{N_t} \\ \delta_{N_t} \end{pmatrix} \in \mathbb{R}^{2N_t} \tag{9}$$

where N_t is the number of turrets installed on the ROSPO platform that for the following simulation is chosen to 4.

III. Control

The control scheme developed is similar to the one proposed in [1] that exploits an hierarchical control architecture with an high level controller responsible to convert the system's state and the respective desired motion into a virtual wrench that's desired on the center of mass (com), and an allocator that dispatches the loads into the different turrets to achieve the goal.

A. High level controller

The high level controller used in this project is the one derived in [1] and is developed so that the controller sees an equivalent first-order linear dynamics.

The commanded virtual input $\boldsymbol{u}_{v,c}$, i.e. the desired wrench on the com of the ROSPO, is computed by the law

$$\boldsymbol{u}_{v,c} = \mathbf{B}^{-1}\big(\dot{\boldsymbol{u}}_v^\star - \mathbf{A}\boldsymbol{u}_v^\star\big) - \mathbf{K}\tilde{\boldsymbol{u}}_v \tag{10}$$

with $\mathbf{A} = -\gamma_P \mathbf{I}_3$, $\mathbf{B} = \gamma_P \mathbf{I}_3$ and $\mathbf{K} \in \mathbb{R}^{3\times 3}$ a gain matrix such that $\mathbf{A} - \mathbf{BK}$ is Hurwitz. In (10) the feedforward signal and it's time derivative are

$$\boldsymbol{u}_v^\star = \begin{pmatrix} m\mathbf{R}^\top(\psi)\left(-k_p\tilde{\boldsymbol{p}} - k_d\dot{\tilde{\boldsymbol{p}}} + \ddot{\boldsymbol{p}}_{ref} - \frac{\boldsymbol{F}_f(\boldsymbol{x})}{m}\right) \\ J\left(-k_{p,\psi}\tilde{\psi} - k_{d,\psi}\dot{\tilde{\psi}} + \ddot{\psi}_{ref} - \frac{T_{f,\psi}(\boldsymbol{x})}{J}\right) \end{pmatrix}$$

$$\dot{\boldsymbol{u}}_v^\star = \begin{pmatrix} m\mathbf{R}^\top(\psi)\big(\mathbf{S}(\psi)\boldsymbol{v} + \boldsymbol{v}_d\big) \\ J\left(-k_{p,\psi}\dot{\tilde{\psi}} - k_{d,\psi}\ddot{\tilde{\psi}} + \dddot{\psi}_{ref} - \frac{\dot{T}_{f,\psi}(\boldsymbol{x})}{J}\right) \end{pmatrix}$$

with $k_\circ$ all positive gains, $\boldsymbol{F}_f, T_{f,\psi}$ respectively the force and the torque due to the friction (as function of the current state of the ROSPO), and $\tilde{\boldsymbol{p}} = \boldsymbol{p} - \boldsymbol{p}_{ref}$ the position error w.r.t. the reference trajectory described by $\boldsymbol{p}_{ref}$. Finally

$$\mathbf{S}(\psi) = \begin{bmatrix} 0 & \dot{\psi} \\ -\dot{\psi} & 0 \end{bmatrix}$$

$$\boldsymbol{v} = -k_p\tilde{\boldsymbol{p}} - k_d\dot{\tilde{\boldsymbol{p}}} + \ddot{\boldsymbol{p}}_{ref} - \frac{\boldsymbol{F}_f}{m}$$

$$\boldsymbol{v}_d = -k_p\dot{\tilde{\boldsymbol{p}}} - k_d\ddot{\tilde{\boldsymbol{p}}} + \dddot{\boldsymbol{p}}_{ref} - \frac{\dot{\boldsymbol{F}}_f}{m}$$

B. Allocator

Most of the effort for the project has been put into the development of the allocator whose goal is to convert the commanded virtual input $\boldsymbol{u}_{v,c}$ into a set of proper inputs $\boldsymbol{u}$ for the system.

In the reference paper, such task is accomplished exploiting the system's model by using an input that guarantees an optimal regulation as well as asymptotic optimality condition; due to the nature of such problem, no saturation or boundaries on the inputs can be ensured, but they are regarded as soft constraints by properly choosing the cost function.

The idea that I propose is instead to use deep neural networks to approximate the optimal input allocation; based on the system's model, a dataset of optimal allocations can be computed offline and used to train the neural network that later can be used only in prediction while running on the effective system.
With this approach computationally expensive algorithms can be used offline to accomplish the allocation while at runtime a deterministic computation of the input is guaranteed by the evaluation of the neural network.

The developed allocation method relies on 2 stages that exploits different neural networks: the first one performs the proper allocation by computing the force (with its attitude) that each turret should have at the next controller update (based on the current state and the commanded virtual input) and a lower-level controller that converts the desired force into a voltage to be applied at the BLDC motors. For sake of simplicity the latter one is described first.

1) BLDC voltage identification: To compute the voltage v that allows to reach a desired force F_{des}, model (3) is used.

Since v doesn't appear explicitly in (4), the force F generated by the model can't be instaneously changed by choosing the voltage; for this reason a discrete approach has been used: given the initial state $\boldsymbol{x}_0 \in \mathbb{R}^2$ of the motor, the ODE is numerically integrated using Runge Kutta methods to determine the force F^+ at the next cycle of the controller. With this idea the optimal control $v^\star$ is given by

$$v^\star(\boldsymbol{x}_0) = \arg\min_v \left\| F^+(v, \boldsymbol{x}_0) - F_{des} \right\|^2 \tag{11}$$

To improve the convergence of numerical methods, the explicit definition of the jacobian of the cost w.r.t. the decision variable v has been provided; rewriting in fact the cost as

$$c = \big(F^+ - F_{des}\big)^\top \big(F^+ - F_{des}\big)$$

it turns out that

$$\frac{dc}{dv} = 2\big(F^+ - F_{des}\big)\frac{\partial F}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{dv}$$

where $\frac{d\boldsymbol{x}}{dv}$ can be obtain by computing the sensitivities with the Runge Kutta methods.

The dataset for computing the input voltage to the motor has been generated from solving (11) by uniformly sampling $\boldsymbol{x}_0 \in \mathcal{X} \subset \mathbb{R}^2$ on a set of bounded values for the state and constraining $v \in [0, v_{max}]$.
To increase the quality of the dataset, data have been generated also considering $\boldsymbol{x}_0$ as a small deviation from a steady-state configuration.

2) Allocation: The dataset for the allocator neural-network has been developed similarly to the previous case, i.e. by solving constrained least-square problems on state uniformly sampled from a defined subspace.

For this problem let's consider the state $\boldsymbol{x} \in \mathbb{R}^{2N_t}$ the one defined as

$$\boldsymbol{x} = \Big(F_1, \phi_1, \ldots, F_{N_t}, \phi_{N_t}\Big)^\top$$

while $\boldsymbol{u} \in \mathbb{R}^{2N_t}$ is simply the ratio of change in time of the respective variables, i.e.

$$\boldsymbol{u} = \Big(\delta F_1, \delta\phi_1, \ldots, \delta F_{N_t}, \delta\phi_{N_t}\Big)^\top$$

Defined with $\boldsymbol{F}_{com}(\boldsymbol{x}) : \mathbb{R}^{2N_t} \to \mathbb{R}^{31}$ the function that given the current state provides the wrench applied to the com, then the minimization problem to solve is of the form

$$\boldsymbol{u}^\star(\boldsymbol{x}_0) = \arg\min_{\boldsymbol{u}} \left\| \boldsymbol{F}_{com}(\boldsymbol{x}_0 + T_s\boldsymbol{u}) - \boldsymbol{F}_{des} \right\|^2_{\mathbf{W}_1} + k\|\boldsymbol{u}\|^2_{\mathbf{W}_2} \tag{12}$$

The reported cost function have 2 main contributions: the first one is used to regulate the input toward the desired wrench, while the second one minimizes the actuation effort. In both cases the least-square costs are weighted by matrices $\mathbf{W}_\circ$ in order to improve the control selection.

Also in this case a subset for both $\boldsymbol{F}_{des}$ and $\boldsymbol{x}_0$ is defined; the dataset is built by randomly sampling from those spaces and solving problem (12) considering boundaries on $\boldsymbol{u}$.

IV. Implementation details

Implementation-wise, the project has been developed using different tools.

Main symbolic computation has been carried out using the software *Maple* [5]. With the help of integrated C code generation of the software, an additional library has been developed to convert all symbolic function into *C++* function for faster evaluation, using the *Eigen* [6] linear algebra library as backend; furthermore the utility create wrapper code for all functions for the python language using *pybind11* [7], with a script that automatically creates necessary dynamic libraries.

Numerical evaluations have been carried out in python; optimization problems have been solved using utilities from the *scipy* [8] library, while *tensorflow* [9] has been used to train deep neural-network models.

Deep-learning models tested have been fully-connected neural networks with different amount of hidden layers and corresponding sizes; the chosen activation function for each perceptron is the rectified linear unit (*ReLU*), since it's less prone to the vanishing gradient problem [10].

All the source code is freely available at https://github.com/matteodv99tn/UAV-project.

V. Results

With the code developed, several tests have been performed to assess the viability of the proposed solution. Even if the underlying idea of fitting functions using neural networks seems interesting, evidence shows non-promising results.

The main issue can be related to the dataset generation and model training. Considering a case with 4 turrets, the goal is to fit a function $\boldsymbol{f} : \mathcal{X} \subset \mathbb{R}^{11} \to \mathbb{R}^3$; to have a good and reliable approximation, the whole space $\mathcal{X}$ should be *sufficiently visited* from a numerical standpoint.

Working from my laptop, even tough minimization problems can be solved as fast as $1ms$, generating $1\,000\,000$ data points requires almost 30 minutes; furthermore with such generated data, each deep-learning model requires about 1 hour to train over 100 epochs.

[1]In this case we implicitly assume that the force is a 3-dimensional vector with the 3rd component being the applied torque.

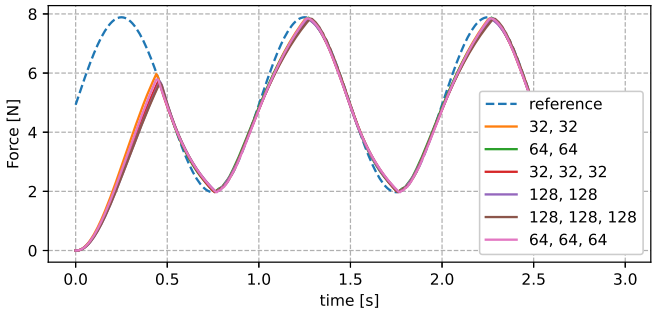

Fig. 3. response of different controllers to a sinusoidal input. For each model the number of perceptron in each hidden layer is reported, e.g "32, 32" means *2 hidden layers with 32 perceptrons each.*

With such amount of data, still the process of training the neural network fails to achieve satisfactory results, probably due to the lack of data that enables a proper function fitting.

The potentiality of the tool is well expressed when controlling the voltage to be applied to the BLDC motor; such problem requires the fitting of a function $f : \mathcal{X} \subset \mathbb{R}^3 \to \mathbb{R}$ with a domain of smaller size for which the computation of a sufficiently large data-set is viable.

As exposed in Fig. 3, models with different configurations of hidden layers are able to reach and correctly track the reference trajectory asked the system.

Using one of the above models and solving the allocation problem (12) at each control step shows the desirable behavior in a trajectory-tracking scenario. Considering the 8-shaped trajectory described by

$$\boldsymbol{p} = \begin{pmatrix} \rho_x \cos(c_1 t) \\ \rho_y \sin(c_2 t) \\ 0 \end{pmatrix}$$

then Fig. 4 shows the path followed by the ROSPO considering simulation parameters in Table II and having chosen the feedback matrix $\mathbf{K}$ for which the eigenvalues of the close-loop matrix $\mathbf{A} - \mathbf{KB}$ are $(-3, -3, -3)$. Figure 5 reports a more detailed look at the time behavior of the different x, y and ψ component of the state. Finally, Fig. 6 shows how the allocator responds to the commanded virtual input.

At the beginning, due to a standstill initial condition, the controller isn't capable of following the requested trajectory, and goes into severe errors while following the path; toward the end instead the system converges to the desired trajectory as expected.

VI. Conclusions

As discussed in the previous section, it can be concluded that using deep neural networks to approximate the optimal allocation law is not feasible, at least with the developed procedure.

The problem of dimensionality will be even higher in case of

TABLE II
COEFFICIENTS USED IN THE ROSPO SIMULATION.

parameter	value
ρ_x	$1m$
ρ_y	$0.6m$
c_1	0.4
c_2	0.8
γ_P	2.5
k_p	1.2
k_d	1.4
$k_{p,\psi}$	3.2
$k_{d,\psi}$	6.35
γ_4	0.44
γ_5	1
m	$6kg$
J	$6.1kg \cdot m^2$

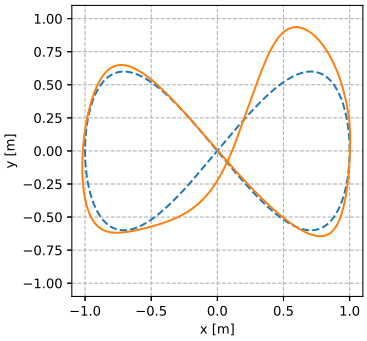

Fig. 4. path followed by the ROSPO while prompted to follow the 8-shaped trajectory.

aerial UAVs, with 3-dimensional space of motion and a higher amount of actuators.

Leaning toward data-driven control methods, interesting could be to develop a reinforcement learning allocator: such technique is based on collected experience by the system itself: this might reduce the exploration of the state space mainly to relevant subspaces, and the learning process is performed online with incremental steps.

As final notes, two points are still open for evaluation:

- comparing the allocation results with the one provided in the reference paper; in this case the model of the BLDC motor has been developed more in detail, and a review of the presented theory in [1] is required; alternatively one could also use a lower-level controller to compute the voltage given the speed reference as in that paper;
- developing a more complex model that takes into account for uncertainties: both high level controller and allocator use at each timestep the values computed by the simulator, so the system is fully observable and behaves deterministically. Interesting should be an approach where the dataset is built from the deterministic model, but testing is carried out by faking measurement comparable with a real world scenario and using state estimators to reconstruct the actual state of the system.

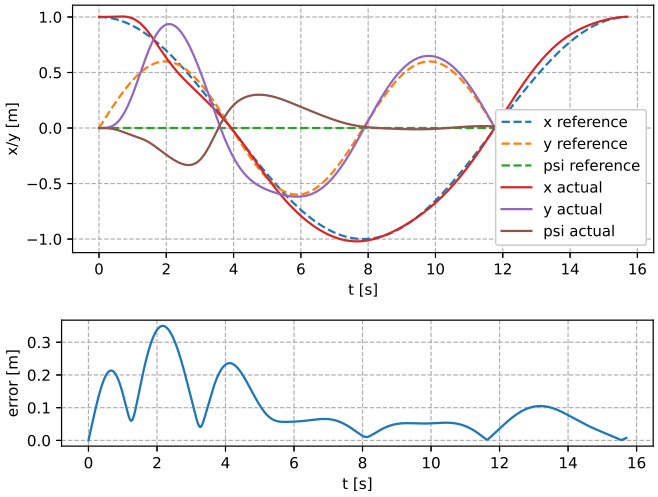

Fig. 5. desired and actual position and orientation of the ROSPO during the simulation.

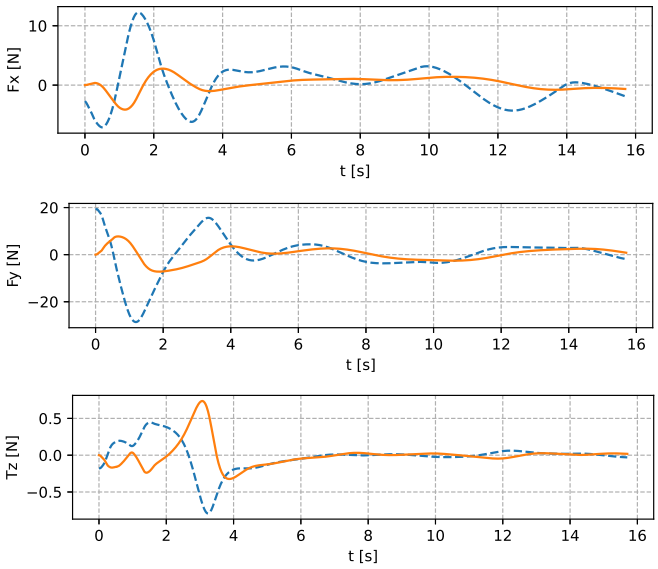

Fig. 6. commanded virtual input and actual force exherted by the system as function of time in the cardinal directions and resulting torque.

REFERENCES

[1] Michele Furci et al. “Input Allocation for the Propeller-Based Overactuated Platform ROSPO”. In: *IEEE Transactions on Control Systems Technology* 28.6 (2020), pp. 2720–2727. DOI: 10.1109/TCST.2019.2944341.

[2] Sujit Rajappa et al. “Modeling, control and design optimization for a fully-actuated hexarotor aerial vehicle with tilted propellers”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 4006–4013. DOI: 10.1109/ICRA.2015.7139759.

[3] Hafiz Zeeshan Iqbal Khan et al. *Nonlinear Control Allocation: A Learning Based Approach*. 2022. arXiv: 2201.06180 [eess.SY].

[4] C. Makkar et al. "A new continuously differentiable friction model for control systems design". In: *Proceedings, 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics.* 2005, pp. 600–605. DOI: 10.1109/AIM.2005.1511048.

[5] MapleSoft. *Maple 2023*. Waterloo, Ontario: a division of Waterloo Maple Inc., 2023.

[6] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[7] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. *pybind11 — Seamless operability between C++11 and Python*. https://github.com/pybind/pybind11. 2016.

[8] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[9] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[10] Tomasz Szandała. *Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks*. Wroclaw University of Science and Technology, 2020.