2020 59th IEEE Conference on Decision and Control (CDC)
Jeju Island, Republic of Korea, December 14-18, 2020

Learning Dynamic-Objective Policies from a Class of Optimal Trajectories

Christopher Iliffe Sprague* sprague@kth.se
Dario Izzo† dario.izzo@esa.int
Petter Ögren* petter@kth.se

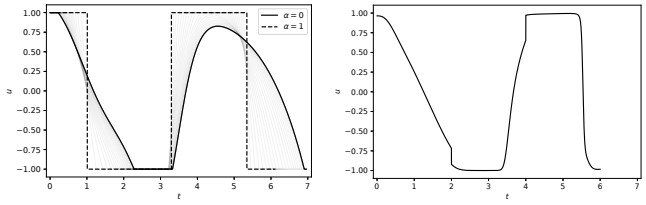
***Abstract*— Optimal state-feedback controllers, capable of changing between different objective functions, are advantageous to systems in which unexpected situations may arise. However, synthesising such controllers, even for a single objective, is a demanding process. In this paper, we present a novel and straightforward approach to synthesising these policies through a combination of trajectory optimisation, homotopy continuation, and imitation learning. We use numerical continuation to efficiently generate optimal demonstrations across several objectives and boundary conditions, and use these to train our policies. Additionally, we demonstrate the ability of our policies to effectively learn families of optimal state-feedback controllers, which can be used to change objective functions online. We illustrate this approach across two trajectory optimisation problems, an inverted pendulum swingup and a spacecraft orbit transfer, and show that the synthesised policies, when evaluated in simulation, produce trajectories that are near-optimal. These results indicate the benefit of trajectory optimisation and homotopy continuation to the synthesis of controllers in dynamic-objective contexts.**

***Index Terms*— Optimal Control, Deep Learning, Homotopy, Online Planning**

I. Introduction

An optimal controller that is capable of changing its objective function in the middle of a mission can be very useful in an autonomous system. Imagine a vehicle that started off on a time-optimal trajectory towards a given destination but suddenly finds itself having less energy than expected; perhaps due to a battery failure, a broken solar panel, or a leaking fuel tank. It can then instantly switch to an effort-optimal controller. Conversely, a vehicle that was saving energy for future transitions might suddenly be in a hurry to reach a destination and can then instantly switch to a controller that reaches the goal in the shortest possible time.

In general, applying an optimal control policy is desirable, but often infeasible for many autonomous systems. The reason for this is that, for many systems characterised by nonlinear dynamics, finding a closed-loop solution to the underlying optimal control is intractable. A common approach to circumvent this is to precompute an open-loop solution, through trajectory optimisation [1], and apply a simple state-feedback controller to track it. However, such an approach suffers when unforeseen events require a change of the objective function or lead the system's state out of the controller's region of attraction.



(a) (b)

Fig. 1: A depiction of the inverted pendulum's optimal control profiles of the (a) path-homotopy between two different objective functions, (b) and policy that switches objective functions parameterised by $\alpha = (0, 0.5, 1)$ at times $t = (0, 2, 4)$, respectively. Note how the policy in (b) thus is similar to (a)-solid in the beginning and (a)-dashed and end, corresponding to a 2-step switch from fuel to time optimal state feedback control learnt by a neural network.

An alternative approach is to apply model predictive control (MPC), where open-loop solutions are incrementally computed and executed on a finite-time horizon. Although this approach provides reactivity in response to unforeseen events, it might suffer in performance, as it does not consider the full-time horizon; and in computational complexity, as it requires repeatedly solving an optimisation problem.

In recent years, with the progression of machine learning (ML), data-driven approaches have become popular in the context of control. Across these approaches, control policies are represented by neural networks (NN) [2], whose feed-forward architectures express simple executable functions. When given appropriate data, these models can be trained to approximate optimal control policies, and pose a reasonable alternative to MPC in terms of both performance and computational complexity [3], especially on low-end hardware.

Whilst the problem of synthesising optimal control policies has been well addressed in ML literature, in both single and multi-objective contexts [4]–[6], relatively few have tackled the problem of adapting policies online to changing objectives [7], [8]. Such a case is particularly relevant to systems in which objective priorities can change unexpectedly, e.g. saving energy vs. saving time. To the best our knowledge, the use of trajectory optimisation and imitation learning has been unexplored in this regard.

In this work, we show that we can straightforwardly leverage trajectory optimisation and homotopy continuation to synthesise state-feedback controllers capable of adapting

* Robotics, Perception and Learning Lab., School of Electrical Engineering and Computer Science, Royal Institute of Technology (KTH), SE-100 44 Stockholm, Sweden
† Advanced Concepts Team, European Space Technology Center (ESTEC), Noordwijk, The Netherlands

978-1-7281-7447-1/20/$31.00 ©2020 IEEE

online to dynamic objectives. Our contributions are:

1) we develop an efficient data-generation pipeline to produce optimal demonstrations across multiple objectives and boundary conditions;
2) and we show that our trained policies can continuously represent homotopies between optimal policies.

The outline of the paper is as follows. In Section II we describe related work. Then, in Section III, we explain the homotopy continuation method we use for trajectory optimisation. In Section IV, we explain our method of dataset generation, our machine learning setup, and the results we achieved. Finally, in Section V, we reflect upon our results and discuss what they entail.

II. Related Work

Many problems in robotics are naturally formulated in terms of nonlinear optimal control problems [9]–[11], but they often lack a closed-form solution. One can usually, however, solve the trajectory optimisation problem (TOP) using numerical methods [1], to obtain open-loop solutions. But, as described before, naïvely executing or tracking these solutions might have drawbacks in terms of robustness.

MPC addresses this issue by iteratively computing and executing an open-loop solution on a finite-time horizon, allowing for online reactions in response to unforeseen events. With advances in MPC's efficiency [12] and increasing computational capabilities of many robotic platforms, this approach has become popular in a variety of applications [11], [13]–[17]. But, although it can be quite effective, the need to employ a finite-time horizon, regardless of computational capabilities, results in suboptimal trajectories, as they do not take the full-time horizon into account.

A set of approaches that mitigates these issues, by learning the optimal control with respect to the full- (or infinite-) time horizon, uses reinforcement learning (RL) [18], [19]. These approaches operate by constructing a policy through incremental interaction with the environment. Although such approaches have been quite successful in the continuous control domain [2], particularly for stochastic systems, they sometimes struggle to converge and are sensitive to the choice of rewards [20], [21].

In the case that expert demonstrations are either readily available or producible, it is sometimes more straightforward to rely on the alternative ML approach: imitation learning (IL). In this approach, expert demonstrations are utilised to synthesise optimal policies in a supervised-learning manner. This approach benefits from a greater sampling-efficiency and has been shown to rapidly accelerate policy-convergence in RL approaches [22], [23]. It is usually the case that demonstrations are not necessarily optimal, but if, however, the system at hand is well-suited to providing optimal demonstrations, e.g. through trajectory optimisation, this approach proves to be particularly useful in converging to an optimal policy in an uncomplicated way [5], [24]–[26].

A commonality among these approaches — IL and RL — is that their policies are represented by NNs and are traditionally orientated towards single-objective scenarios only. Some pertaining works have indeed addressed the multi-objective[1] case, but typically have only sought policies lying along the Pareto front [6], and only few have targeted the case where objective priorities may change over time [7], [8].

In the virtual-first work to address this case [7], a set of policies was learnt in an RL setting, to discretely represent a spectrum of different objectives, which could then be referred to online to adapt to changing objective priorities. However, as noted in [8], using a discrete set of policies does not scale to complex problems nor truly represent a continuous mapping between objectives. Recently, [8] showed that policies could be adapted online to dynamic preferences in an RL setting, using a Q-network, conditioned on the objective-priority. We diverge from these works in that we show how trajectory optimisation and homotopy can be straightforwardly leveraged to synthesise these conditional policies in an IL setting.

In [5], it was shown that Pontryagin's minimum principle (PMP)[2] [27] could be used to generate optimal demonstrations in order to synthesise optimal policies in an IL setting. It was shown, across a variety of two-dimensional dynamical systems, that the resulting policies, when executed in simulation, produced near-optimal trajectories, even when the systems were initialised in unseen regions of the state-space. In [24], we showed that this approach could be extended to three dimensions and used to encode manifold boundary conditions in the state-space, using transversality conditions of PMP. These results were built upon in [25], where an investigation of NN hyperparameters and an improved evaluation metric — *policy trajectory optimality* — were presented.

In these works, optimal policies were synthesised for a single objective. However, in both [5] and [26], homotopy continuation [28] was employed between different objectives to ease the burden of generating demonstrations under objectives which are particularly challenging for gradient-based optimisers. Such a case arises when the topology of the TOP becomes discontinuous due to the underlying optimal control being *bang-bang*. We build upon these works by taking advantage of the full homotopy path between objectives to straightforwardly synthesise conditional policies, which can react online to changing objective priorities, as we will describe in more detail in the following sections.

III. Trajectory optimisation

In this section, we outline our optimal-trajectory dataset generation process, used for learning in Section IV.

Throughout this paper, we consider nonlinear autonomous dynamical systems of the form $\dot{\boldsymbol{s}} = \boldsymbol{f}(\boldsymbol{s},\boldsymbol{u})$, with scalar objec-

[1]The single- and multi-objective modalities of trajectory optimisation are often referred to as *single-* and *multi-task* in ML parlance, respectively. Whilst, in ML, these terms often describe boundary conditions associated with different tasks, we maintain that an *objective* describes only the cost of a trajectory ($\mathscr{J}$).

[2]PMP is a manifestation of *indirect* trajectory optimisation; alternatively, *direct* methods could be used to generate optimal demonstrations in the case of a black-box dynamics model. For an overview of trajectory optimisation methods refer to [1].

tives of the form $\mathscr{J} = \int_{t_0}^{t_f} \mathscr{L}(\boldsymbol{u}(t))\,\mathrm{d}t$. With these, one can solve the TOP to determine an optimal trajectory $[\boldsymbol{s}(t), \boldsymbol{u}(t)]^\intercal$ — an open-loop solution — given an initial and desired state. In this work, we formulate this problem as a nonlinear programme and solve it using numerical optimisation [1], where we must determine the optimal *decision vector* $\boldsymbol{z}^\star$ to obtain the optimal trajectory. For brevity, we denote the TOPs in Sections III-B and III-C as P_0 and P_1, respectively.

A. Homotopy continuation

Under certain objectives, computing an optimal trajectory can be rather difficult; an example is when the optimal control profile $\boldsymbol{u}^\star(t)$ is discontinuous, e.g. in the bang-bang case. In order to decrease the burden of finding a solution, homotopy continuation, or regularisation techniques, can be employed to achieve convergence. This method is often used to solve singular-control problems [28].

To elaborate further: given two different objectives, one can define a linear homotopy between them

$$\mathscr{J} = (1-\alpha)\int_{t_0}^{t_f} \mathscr{L}(\boldsymbol{u}(t))\,\mathrm{d}t + \alpha \int_{t_0}^{t_f} \mathscr{L}'(\boldsymbol{u}(t))\,\mathrm{d}t, \qquad (1)$$

where the *homotopy parameter* $\alpha \in [0,1]$ characterises the objectives' priorities. Assuming $\mathscr{L}$ characterises the easier objective, we can solve the TOP first with $\alpha = 0$, then use its solution to resolve the same problem with a slightly increased value. With this process, we iteratively solve the TOP until $\alpha = 1$, at which point we will have converged to a solution under the difficult objective. Through this process, we obtain a convergent series of solutions — a *path-homotopy*. If there are several different objectives, one can extend this approach to multiple homotopies, as we do in Section III-C. This process is further described in Algorithm 1, where the boundary conditions remain constant, $\texttt{solve}(\boldsymbol{s}_0, \boldsymbol{z}, \alpha)$ minimises (1) from an initial state $\boldsymbol{s}_0$, and α^* is the last successful solution parameter.

Algorithm 1: Criteria homotopy

Function `homotopy_criteria` $(\boldsymbol{s}_0^\star,\ \boldsymbol{z}^\star,\ \alpha^\star)$ **:**
- $\boldsymbol{T}, \alpha \leftarrow \emptyset, \alpha^\star$
- **while** $\alpha < 1$ **do**
  - $\boldsymbol{z} = \texttt{solve}(\boldsymbol{s}_0^\star, \boldsymbol{z}^\star, \alpha)$
  - **if** `successful`$(\boldsymbol{z})$ **then**
    - $\boldsymbol{z}^\star, \alpha^\star \leftarrow \boldsymbol{z}, \alpha$
    - $\boldsymbol{T} \leftarrow \boldsymbol{T} \cup \{(\boldsymbol{s}_0^\star, \boldsymbol{z}^\star, \alpha^\star)\}$ `// save traj.`
    - $\alpha \leftarrow \alpha \in [\alpha^\star, 1]$ `// increase` α
  - **else**
    - $\alpha \leftarrow \alpha \in [\alpha^\star, \alpha]$ `// decrease` α
- **return** $\boldsymbol{T}$

Similarly, to obtain a database that encompasses a diverse set of states, we can also utilise continuation upon the initial state. To do this, we slightly perturb the initial state and resolve the TOP with the previous solution as the initial guess, again leading to a convergent series of solutions. We further describe this process in Algorithm 2, where the homotopy parameter remains constant, δ defines the perturbation size, and n is the desired number of solutions. Through these algorithms, we obtain a large set of TOP solutions $\boldsymbol{T}$, that encompass the entire range of homotopy parameters and a broad range of initial states.

Algorithm 2: State Homotopy

Function `homotopy_state` $(\boldsymbol{s}_0^\star,\ \boldsymbol{z}^\star,\ \alpha^\star,\ n)$ **:**
- $\boldsymbol{T} \leftarrow \emptyset$
- **while** $|\boldsymbol{T}| < n$ **do**
  - $\boldsymbol{s}_0 \leftarrow \texttt{perturb}(\boldsymbol{s}_0^\star,\ \delta)$
  - $\boldsymbol{z} \leftarrow \texttt{solve}(\boldsymbol{s}_0,\ \boldsymbol{z}^\star, \alpha^\star)$
  - **if** `successful`$(\boldsymbol{z})$ **then**
    - $\boldsymbol{s}_0^\star, \boldsymbol{z}^\star \leftarrow \boldsymbol{s}_0, \boldsymbol{z}$
    - $\boldsymbol{T} \leftarrow \boldsymbol{T} \cup \{(\boldsymbol{s}_0^\star, \boldsymbol{z}^\star, \alpha^\star)\}$
    - $\delta \leftarrow \texttt{increase}(\delta)$
  - **else**
    - $\delta \leftarrow \texttt{decrease}(\delta)$
- **return** $\boldsymbol{T}$

B. Inverted pendulum swing up

For our first test case, we consider the simple nondimensional inverted pendulum [29],

$$\dot{\boldsymbol{s}} = \begin{bmatrix} \dot{x} \\ \dot{v} \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v \\ u \\ \omega \\ \sin(\theta) - u\cos(\theta) \end{bmatrix} = \boldsymbol{f}(\boldsymbol{s}, u), \qquad (2)$$

where the variables are: cart position x, cart velocity v, pole angle θ, pole angular velocity ω, and control $u \in [-1, 1]$.

We begin by defining a homotopic objective

$$\mathscr{J} = (1-\alpha)\int_{t_0}^{t_f} u(t)^2\,\mathrm{d}t + \alpha\int_{t_0}^{t_f} \mathrm{d}t, \qquad (3)$$

defined from quadratically optimal ($\alpha = 0$) to time-optimal ($\alpha = 1$) control. Using PMP, we define the *Hamiltonian*

$$\begin{aligned}\mathscr{H} = \boldsymbol{\lambda}^\intercal \cdot \boldsymbol{f}(\boldsymbol{s}, u) + \mathscr{L} &= \lambda_x v + \lambda_v u + \lambda_\theta \omega \\ &+ \lambda_\omega\left(\sin(\theta) - u\cos(\theta)\right) + (1-\alpha)u^2 + \alpha, \end{aligned} \qquad (4)$$

where $\boldsymbol{\lambda}$ is the Lagrange multiplier vector — or *costate* — having the same dimensionality as the state. We then minimise $\mathscr{H}$ with respect to u to find the optimal control

$$u^\star = \underset{u}{\operatorname{argmin}}\,(\mathscr{H}) = \frac{\lambda_v - \lambda_\omega \cos(\theta)}{2(\alpha - 1)}, \qquad (5)$$

defined for $\alpha \in [0, 1)$. As we drive the homotopy parameter to unity, we find the optimal *switching function*

$$\sigma = \lambda_v - \lambda_\omega \cos(\theta), \qquad (6)$$

defining the bounded time-optimal control

$$\lim_{\alpha \to 1}\left(u^\star\right) = \begin{cases} -1 & \text{if } \sigma < 0 \\ 1 & \text{if } \sigma > 0 \end{cases}, \qquad (7)$$

where it should be noted that the edge case $\sigma = 0$ is only encountered instantaneously, if at all, so singular control

analysis is not necessary here. Lastly, we compute the *costate equations*

$$\dot{\boldsymbol{\lambda}} = -\nabla_{\boldsymbol{s}}\mathscr{H} = \begin{bmatrix} 0 \\ -\lambda_x \\ -\lambda_\omega \left(u\sin(\theta)+\cos(\theta)\right) \\ -\lambda_\theta \end{bmatrix}. \tag{8}$$

To encode the swingup task into the TOP, we enforce the equality constraint $\boldsymbol{s}(t_f) = \mathbf{0}$. With the shooting method [1], the decision vector becomes $\boldsymbol{z} = [T, \boldsymbol{\lambda}(t_0)]^\intercal$, where $T = t_f - t_0$ is the trajectory duration and $\boldsymbol{\lambda}(t_0)$ is the initial costate. Following Algorithm 1 and considering the nominal downright state $\boldsymbol{s}_0 = [0\ 0\ \pi\ 0]^\intercal$, we compute our initial path-homotopy between quadratically and time-optimal control, as shown in Figure 1a.

C. Spacecraft orbit transfer

As an increase in complexity of boundary conditions and dimensionality, we now consider the problem of optimising a spacecraft's trajectory from Earth to the orbit of Mars. Its heliocentric dynamics are given by

$$\dot{\boldsymbol{s}} = \begin{bmatrix} \dot{\boldsymbol{r}} \\ \dot{\boldsymbol{v}} \\ \dot{m} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \frac{Tu}{m}\hat{\boldsymbol{u}} - \frac{\mu}{r^3}\boldsymbol{r} \\ -\frac{Tu}{I_{sp}g_0} \end{bmatrix} = \boldsymbol{f}(\boldsymbol{s},\boldsymbol{u}), \tag{9}$$

where the state variables are: position $\boldsymbol{r} = [x,y,z]^\intercal$, velocity $\boldsymbol{v} = [v_x, v_y, v_z]^\intercal$, and mass m. The controls are the thrust's throttle $u \in [0,1]$ and direction $\hat{\boldsymbol{u}} = [\hat{u}_x, \hat{u}_y, \hat{u}_z]^\intercal$. The constant parameters governing the dynamics are: maximum thrust capability $T = 0.2$ $[N]$, specific impulse $I_{sp} = 2500$ $[s]$, gravitational acceleration at sea level $g_0 = 9.81$ $[m\cdot s^{-1}]$, and standard gravitational parameter of the sun $\mu = 1.3271 \times 10^{20}$ $[m^3\cdot s^{-2}]$.

We again construct a homotopic objective,

$$\mathscr{J} = (1-\beta)\int_{t_0}^{t_f} u(t)^2\,\mathrm{d}t + \beta\int_{t_0}^{t_f}\left(\alpha + u(t)(1-\alpha)\right)\mathrm{d}t, \tag{10}$$

where we have added a secondary homotopy parameter $\beta \in [0,1]$ as a means to feasibly arrive to a solution of the problem at $\alpha \in [0,1], \beta = 1$. The homotopies between optimal controls are defined as: quadratic to effort $\alpha = 0, \beta \in [0,1]$, quadratic to time $\alpha = 1, \beta \in [0,1]$, and effort to time $\alpha \in [0,1], \beta = 1$. Using PMP, we then define the Hamiltonian

$$\mathscr{H} = \boldsymbol{\lambda}_r\cdot\boldsymbol{v} + \boldsymbol{\lambda}_v\cdot\left(\frac{Tu}{m}\hat{\boldsymbol{u}} - \frac{\mu}{r^3}\boldsymbol{r}\right) + \lambda_m\left(-\frac{Tu}{I_{sp}g_0}\right) + \beta\left(\alpha + u(1-\alpha)\right) + u^2(1-\beta) \tag{11}$$

To find the optimal thrust direction $\hat{\boldsymbol{u}}^\star$, we isolate the portion of $\mathscr{H}$ that depends on it: $\frac{Tu}{m}\left(\hat{\boldsymbol{u}}\cdot\boldsymbol{\lambda}_v\right)$. Considering that T, m, and u are positive, $\hat{\boldsymbol{u}}^\star$ must be directed opposite of $\boldsymbol{\lambda}_v$ to be a minimiser:

$$\hat{\boldsymbol{u}}^\star = -\frac{\boldsymbol{\lambda}_v}{\lambda_v}. \tag{12}$$

Substituting $\hat{\boldsymbol{u}}^\star$ into $\mathscr{H}$ and minimising it with respect to u, we obtain

$$u^\star = \frac{1}{2(1-\beta)}\left(\frac{T\left(\boldsymbol{\lambda}_v\cdot\boldsymbol{\lambda}_v\right)}{\lambda_v m} + \frac{T\lambda_m}{I_{sp}g_0} + \beta(\alpha-1)\right), \tag{13}$$

defined for $\alpha \in [0,1], \beta \in [0,1)$. Driving β to unity, we find

$$\sigma = \frac{\boldsymbol{\lambda}_v\cdot\boldsymbol{\lambda}_v}{m\lambda_v} + \lambda_m + 1 - \alpha, \tag{14}$$

defining the bounded control

$$\lim_{\beta\to 1}\left(u^\star\right) = \begin{cases} 0 & \text{if } \sigma < 0 \\ 1 & \text{if } \sigma > 0 \end{cases}, \tag{15}$$

expressing the homotopy between effort- ($\alpha = 0$) and time-optimal control ($\alpha = 1$), which are discontinuous. Finally, we compute

$$\dot{\boldsymbol{\lambda}} = \begin{bmatrix} \frac{\mu}{r^3}\boldsymbol{\lambda}_v - \frac{3\mu}{r^5}\left(\boldsymbol{\lambda}_v\cdot\boldsymbol{r}\right) \\ -\boldsymbol{\lambda}_r \\ \frac{Tu}{m^2}\left(\boldsymbol{\lambda}_v\cdot\hat{\boldsymbol{u}}\right) \end{bmatrix}. \tag{16}$$

To finish defining the TOP, we enforce two *transversality conditions*. The first is to allow the optimiser to choose the best final mass $\lambda_m(t_f) = 0$, since it is affected by the control. The second is to allow the optimiser to choose the best terminal state along Mars's orbit

$$\frac{r^3\left(\boldsymbol{\lambda}_v\cdot\boldsymbol{v}\right) - \mu\left(\boldsymbol{\lambda}_r\cdot\boldsymbol{r}\right)}{\sqrt{\mu^2\left(\boldsymbol{r}\cdot\boldsymbol{r}\right) + \left(\boldsymbol{v}\cdot\boldsymbol{v}\right)\left(\boldsymbol{r}\cdot\boldsymbol{r}\right)^3}} = 0, \tag{17}$$

see [24] for details. The decision vector becomes $\boldsymbol{z} = [T, M(t_f), \boldsymbol{\lambda}(t_0)]^\intercal$, where $T = t_f - t_0$ is the time of flight, $M(t_f)$ is the mean anomaly of the final orbit, and again $\boldsymbol{\lambda}(t_0)$ is the initial costate. Following Algorithm 1 again and considering Earth's position and velocity as the initial state, and $m(t_0) = 1000$ $[kg]$, we compute each homotopy path, as shown in Figures 2a–2c, for an interplanetary transfer to somewhere along Mars's orbit[3].

IV. Learning

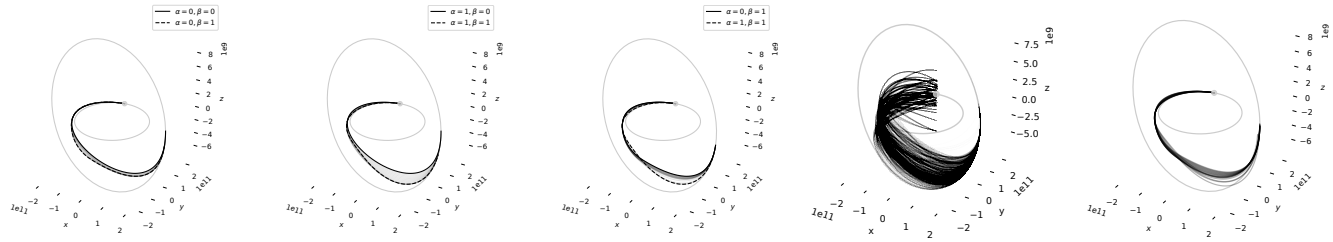
In this section we describe the generated datasets in detail, outline the implemented NN architectures, and evaluate the results of the learning process.

A. Datasets

To assemble the datasets, we first solve the TOPs from their nominal initial states, as described in Sections III-B and III-C. From these initial states, we employ Algorithm 2 to obtain a set of solutions $\boldsymbol{T}$ from various other initial states, under the nominal homotopy parameter configurations: $\alpha = 0$ and $\alpha = \beta = 0$ for P_0 and P_1, respectively. We then employ Algorithm 1 to obtain solutions across the homotopy parameters' ranges. Finally, we numerically integrate[4] these systems with the solutions' parameters to obtain datasets of the form

[3]We pick the starting time to be 0 $[MJD2000]$, using Modified Julian Date notation.

[4]We use an adaptive-stepsize Runge-Kutta method of order 8(5,3) [30].



(a) (b) (c) (d) (e)

Fig. 2: Positional depictions of the spacecraft orbit transfer's (P_1) path-homotopy between (a) quadratically–effort, (b) quadratically–time, and (c) effort–time optimal control, respectively; (d) dataset (D_1) of optimal trajectories; and (e) policy-controlled trajectories for $\alpha \in [0,1], \beta = 1$.

$D = \{(\boldsymbol{s}, \alpha, \boldsymbol{u})_i \forall i\}$ — D_0 with 945 trajectories ($i = 577,480$) from P_0, and D_1 with 5,525 trajectories ($i = 2,124,668$) from P_1 ($\beta = 1$).

B. Neural network architectures

We augment the state-feedback controller formalism to synthesise multilayer-perceptron (MLP) policies of the form $\pi(s, \alpha) \mapsto \boldsymbol{u}^\star$. We denote π's hidden architecture with $m \times n$, describing n hidden layers, each having m nodes. For every hidden layer we sequentially apply the standard NN operations: linear transformation, layer normalisation, and softplus — [25] indicated that softplus activations result in smoother and more meaningful control strategies. For brevity, we denote P_0's and P_1's instances of π as π_0 and π_1, respectively. We apply the hyperbolic tangent in the last layer and use its output to approximate the controls of P_0 and P_1: $\pi_0(\boldsymbol{s}, \alpha) \in [-1,1] = u$ and $\pi_1(\boldsymbol{s}, \alpha) \in [-1,1]^3 \mapsto [u \;\; \phi \;\; \theta]^\intercal \in [0,1] \times [0,\pi] \times [0,2\pi] \mapsto u \cdot [\sin(\theta)\cos(\phi), \sin(\theta)\sin(\phi), \cos(\theta)]^\intercal = u \cdot \hat{\boldsymbol{u}}$, respectively.

C. Evaluation

For π_0 and π_1, we consider the hidden architectures: 50×2, 50×4, 100×2, 100×4. We train these MLPs on their respective datasets — D_0 and D_1 — for 10,000 episodes with the Adam optimiser [31], with a learning rate of 10^{-3}, a decay rate of 10^{-5}, and a mean-squared error (MSE) loss function. At each training epoch, we randomly sample 20,000 datapoints, reserving 10% for validation. The training history and regression performances are shown in Figure 3c and Table I, respectively.

Since success in the regression task does not necessarily mean that the policy-controlled systems will behave as expected, we also evaluate the policy trajectory optimality metric, described in [25], indicating the optimality-difference between a policy-controlled and optimal trajectory from an initial state. From the systems' nominal initial states, we evaluate this metric across the hidden architectures and a range of homotopy parameters — shown in Table II.

Across all architectures, we find that π_0 and π_1 perform well in the regression task and result in trajectories that are near-optimal with respect to the true optimal, across a range of homotopy parameters. We observe that increasing node breadth and number of layers generally results in slightly better regression performance as well as trajectory optimality.

Nodes × Layers	ε_0	ε_0'	ε_1	ε_1'
50×2	0.043434	0.049832	0.014710	0.013711
50×4	0.034315	0.043264	0.014066	0.012860
100×2	0.038298	0.044142	0.018373	0.018608
100×4	0.029807	0.036758	0.012590	0.011232

TABLE I: The final *MSE* losses in training (ε) and validation (ε') of π_0 and π_1 in the regression of D_0 and D_1, respectively.

	$m \times n$			
α	50×2	50×4	100×2	100×4
0.0	8.366217	0.201963	6.911897	0.625839
0.1	2.310301	2.667611	2.722436	2.669494
0.2	1.270388	4.418089	1.152010	3.045441
0.3	3.473601	4.555533	3.798529	5.253961
0.4	4.832105	3.444665	4.429260	4.477721
0.5	3.260343	1.645225	2.438281	2.761684
0.6	1.209908	0.616588	0.729584	1.150946
0.7	0.669214	0.430044	0.279920	0.144534
0.8	0.497213	0.514005	0.356340	2.509390
0.9	0.904339	0.797179	0.485849	0.836367
1.0	2.309510	1.775131	1.027916	1.200968
Mean	2.645740	1.915094	2.212002	2.243304

(a) Inverted pendulum swingup

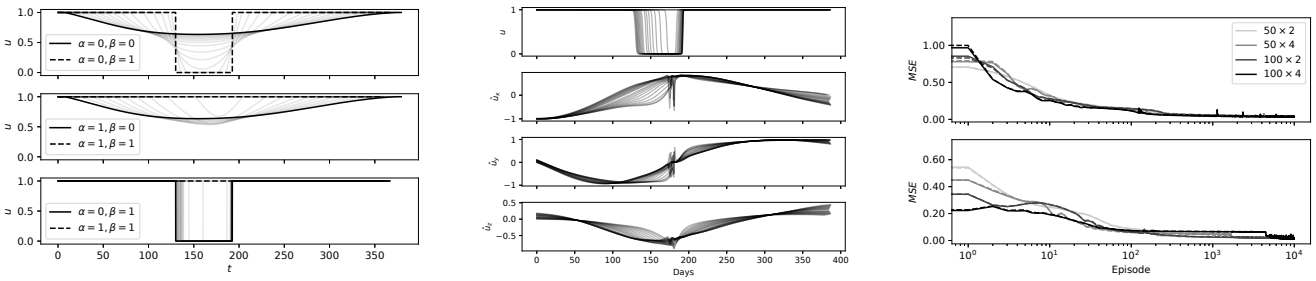
	$m \times n$			
α	50×2	50×4	100×2	100×4
0.0	5.097919	0.163055	0.264219	0.094422
0.1	3.551325	0.057574	0.109408	0.073708
0.2	2.160594	0.355176	0.468579	0.353486
0.3	0.954027	0.665326	0.754457	0.675646
0.4	0.117137	0.979295	0.975896	1.031487
0.5	0.519775	0.344638	0.242022	0.455965
0.6	0.832056	0.210354	0.353348	0.051248
0.7	0.940100	0.622716	0.765109	0.439597
0.8	1.231413	1.173717	1.259935	1.005639
0.9	0.799833	1.096328	1.055267	1.012649
1.0	1.103462	0.189937	0.768724	0.044418
Mean	1.573422	0.532556	0.637906	0.476206

(b) Spacecraft orbit transfer

TABLE II: Policy trajectory optimality gap (%) between the policy-controlled and optimal trajectories for different architectures ($m \times n$) and homotopy parameters (α).

V. Conclusions

We presented a novel approach to synthesising objective-conditioned near-optimal policies through trajectory optimisation, homotopy, and imitation learning. We further demonstrated that these state-feedback policies produce trajectories that are near-optimal, and can adapt their behaviours online



(a) (b) (c)

Fig. 3: (a) Control path-homotopies of P_1. (b) Control profiles of policy-controlled trajectories in P_1 across different homotopy parameters ($\alpha \in [0,1], \beta = 1$). (c) Training history of π_0 (top) and π_1 (bottom).

to changing objectives, due to unexpected events or user-defined preferences.

ACKNOWLEDGEMENT

This work was supported by Stiftelsen for Strategisk-Forskning (SSF) through the Swedish Maritime Robotics Centre (SMaRC) (IRC15-0046).

REFERENCES

[1] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*. Siam, 2010, vol. 19.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[3] B. M. Åkesson and H. T. Toivonen, "A neural network model predictive controller," *Journal of Process Control*, vol. 16, no. 9, pp. 937–946, 2006.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[5] C. Sánchez-Sánchez and D. Izzo, "Real-time optimal control via deep neural networks: study on landing problems," *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 5, pp. 1122–1135, 2018.

[6] C. Liu, X. Xu, and D. Hu, "Multiobjective reinforcement learning: A comprehensive overview," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 3, pp. 385–398, 2014.

[7] S. Natarajan and P. Tadepalli, "Dynamic preferences in multi-criteria reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 601–608.

[8] A. Abels, D. M. Roijers, T. Lenaerts, A. Nowé, and D. Steckelmacher, "Dynamic weights in multi-objective deep reinforcement learning," *arXiv preprint arXiv:1809.07803*, 2018.

[9] Y. Zeng and R. Zhang, "Energy-efficient uav communication with trajectory optimization," *IEEE Transactions on Wireless Communications*, vol. 16, no. 6, pp. 3747–3760, 2017.

[10] D. Izzo, D. Hennes, L. F. Simões, and M. Märtens, "Designing complex interplanetary trajectories for the global trajectory optimization competitions," in *Space Engineering*. Springer, 2016, pp. 151–176.

[11] C. Shen, Y. Shi, and B. Buckham, "Trajectory tracking control of an autonomous underwater vehicle using lyapunov-based model predictive control," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 7, pp. 5796–5805, 2018.

[12] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on control systems technology*, vol. 18, no. 2, pp. 267–278, 2010.

[13] R. R. Nair and L. Behera, "Robust adaptive gain higher order sliding mode observer based control-constrained nonlinear model predictive control for spacecraft formation flying," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 1, pp. 367–381, 2018.

[14] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6753–6760.

[15] M. Neunert, M. Stäuble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli, "Whole-body nonlinear model predictive control through contacts for quadrupeds," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1458–1465, 2018.

[16] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[17] M. N. Zeilinger, D. M. Raimondo, A. Domahidi, M. Morari, and C. N. Jones, "On real-time robust model predictive control," *Automatica*, vol. 50, no. 3, pp. 683–694, 2014.

[18] D. P. Bertsekas, *Reinforcement learning and optimal control*, 2019.

[19] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[20] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[21] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan, "Inverse reward design," in *Advances in neural information processing systems*, 2017, pp. 6765–6774.

[22] L. Sun, C. Peng, W. Zhan, and M. Tomizuka, "A fast integrated planning and control framework for autonomous driving via imitation learning," in *ASME 2018 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers, 2018, pp. V003T37A012–V003T37A012.

[23] C.-A. Cheng, X. Yan, N. Wagener, and B. Boots, "Fast policy learning through imitation and reinforcement," *arXiv preprint arXiv:1805.10413*, 2018.

[24] D. Izzo, C. Sprague, and D. Tailor, "Machine learning and evolutionary techniques in interplanetary trajectory design," *arXiv preprint arXiv:1802.00180*, 2018.

[25] D. Tailor and D. Izzo, "Learning the optimal state-feedback via supervised imitation learning," *arXiv preprint arXiv:1901.02369*, 2019.

[26] C. I. Sprague and P. Ögren, "Adding neural network controllers to behavior trees without destroying performance guarantees," *arXiv preprint arXiv:1809.10283*, 2018.

[27] L. S. Pontryagin, V. G. Boltyanshii, R. V. Gamkrelidze, and E. F. Mishenko, *The Mathematical Theory of Optimal Processes*. New York: John Wiley and Sons, 1962.

[28] F. Bonnans, P. Martinon, and E. Trélat, "Singular arcs in the generalized goddards problem," *Journal of optimization theory and applications*, vol. 139, no. 2, pp. 439–461, 2008.

[29] C. I. Sprague and P. Ögren, "Adding neural network controllers to behavior trees without destroying performance guarantees," *CoRR*, vol. abs/1809.10283, 2018. [Online]. Available: http://arxiv.org/abs/1809.10283

[30] J. R. Dormand and P. J. Prince, "A family of embedded runge-kutta formulae," *Journal of computational and applied mathematics*, vol. 6, no. 1, pp. 19–26, 1980.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.