energies

MDPI

Article

Model Reference Tracking Control Solutions for a Visual Servo System Based on a Virtual State from Unknown Dynamics

Timotei Lala, Darius-Pavel Chirla and Mircea-Bogdan Radac *

Department of Automation and Applied Informatics, Politehnica University of Timisoara, 300223 Timisoara, Romania; timotei.lala@student.upt.ro (T.L.); darius.chirla@student.upt.ro (D.-P.C.)
* Correspondence: mircea.radac@upt.ro

Abstract: This paper focuses on validating a model-free Value Iteration Reinforcement Learning (MFVI-RL) control solution on a visual servo tracking system in a comprehensive manner starting from theoretical convergence analysis to detailed hardware and software implementation. Learning is based on a virtual state representation reconstructed from input-output (I/O) system samples under nonlinear observability and unknown dynamics assumptions, while the goal is to ensure linear output reference model (ORM) tracking. Secondary, a competitive model-free Virtual State-Feedback Reference Tuning (VSFRT) is learned from the same I/O data using the same virtual state representation, demonstrating the framework's learning capability. A model-based two degrees-of-freedom (2DOF) output feedback controller serving as a comparisons baseline is designed and tuned using an identified system model. With similar complexity and linear controller structure, MFVI-RL is shown to be superior, confirming that the model-based design issue of poor identified system model and control performance degradation can be solved in a direct data-driven style. Apart from establishing a formal connection between output feedback control, state feedback control and also between classical control and artificial intelligence methods, the results also point out several practical trade-offs, such as I/O data exploration quality and control performance leverage with data volume, control goal and controller complexity.

Keywords: reinforcement learning and approximate dynamic programming; virtual state feedback reference tuning; model reference control; unknown dynamics; input-output observable system; visual servo; image processing

Citation: Lala, T.; Chirla, D.-P.; Radac, M.-B. Model Reference Tracking Control Solutions for a Visual Servo System Based on a Virtual State from Unknown Dynamics. *Energies* **2022**, *15*, 267. https://doi.org/10.3390/en15010267

Academic Editor: Eduard Petlenkov

Received: 7 November 2021
Accepted: 28 December 2021
Published: 31 December 2021

1. Introduction

The reinforcement learning (RL) paradigm has seen significant recent development in terms of both artificial intelligence (AI) and in control systems as the two main paths leading the research, with some of the most popularized achievements belonging to the former area. However, recent progress has seen hybridization of ideas from the two fields. The promise of data-driven RL is its ability to learn complex decision policies (controllers) under uncertain, unknown (model-free), high-dimensional environments. While the AI approach largely deals with scalability issues, learning efficiency in terms of data volume and speed, and complex environments, some other features are also well-established, such as, e.g., using neural networks (NNs) (plain feedforward, convolutional or LSTM-like recurrent ones) as function approximators, both for the value function and for the policy function, which is the golden standard. The AI approach presents many contributions, such as parallel exploration using multiple environments to enrich exploration diversity, or sparse or dense reward-shaping in various environments under hierarchical learning problem formulation [1,2], and value function overestimation biasing the learning in random environments, which is dealt with by multiple value function approximators, different update frequencies of the critic and policy networks, target policy smoothing, etc., in DDPG, TD3 and SAC versions [3–5], to name a few.

But it should be of no surprise that most of these RL achievements in AI have been tested under simulation environments, like video games (DQN Atari for example) and other simulated mechatronics, and not in real world test rigs. In the context of real-world control problems other issues prevail, such as: stability and learning convergence meet the learning efficiency in terms of data volume, and mechanical constraints and real-time implementation requirements being leveraged by efficient function approximators. This is the reason why classical control theory has addressed RL control from somewhat different perspectives than AI. In addition, real-world control problems have often and traditionally been considered as low-dimensional compared to AI problems.

Some of the earliest works in RL, also known as approximate (adaptive) dynamic programming, have been established by seminal works of [6–8]. RL for control uses two main algorithms, namely Policy Iteration (PoIt) and Value Iteration (VI), and their in-between, Generalized PoIt [7]. In the context of these two popular RL algorithms, the implementation flavors have witnessed offline and online variants, batch- and sample-by-sample adaptive-wise, model-based and model-free, and their combinations. Out of the two algorithms, VI-RL has been arguably considered as the most attractive, since it spans the "unknown dynamics" control case, and its initial controller and value function approximators bear random initialization, which is natural when unknown system dynamics are assumed from start. A significant effort was dedicated to ensuring learning convergence and stability, more generally under generic function approximators, such as NNs.

Much of the learnability in RL relies on the Markovian assumption about the controlled system, together with its full state availability. Although different state representations have been proposed in AI, involving virtual states (called state aliases) built from present and past system's input-output (I/O) data samples, it has not benefited from a well-established theoretical foundation [9]. However, for RL in control systems, the system observability and controllability are well-founded concepts from a historical perspective [10,11].

Among the vast practical control problems approached by RL under classical control analysis approach [12–17] and under AI-based analysis approach [18–21], the output reference model (ORM) tracking has attracted a lot of research attention [12,22–24]. This control problem tries learning a state-feedback controller to make the controlled system output behave similarly to the ORM's output, under the same reference input excitation signal. The practical implications of ORM tracking are even more appealing when linear ORMs are used, as indirect feedback linearization is ensured, i.e., a nonlinear system coupled with a nonlinear controller to make the closed-loop control system (CLS) match a linear ORM. Such control problem can be posed as an optimal control problem and has been solved, e.g., by VI-RL and Virtual State-Feedback Reference Tuning (VSFRT) control learning approaches [24]. This linear ORM tracking has proven itself as the building block for more advanced hierarchical control architectures [25,26] aimed at generalizing learned optimal tracking behavior to new unseen trajectories, therefore endowing control systems with extrapolation reasoning similar to intelligent beings. Such goals have been tackled by primitive-based learning frameworks [25–30].

Learning LORM tracking control from system I/O data samples with unknown dynamics based on a virtual state representation is a data-driven research approach motivated by the need to solve the control performance degradation occurring due to the mismatch between the true, complex system and its identified, simplified and uncertain model. This is considered a nuisance with the model-based control paradigm. In this context, this work's contribution is threefold:

(a) a novel convergence analysis of the model-free VI-RL (MFVI-RL) algorithm is proposed. The iterative convergence of the cost function towards the optimal cost and the convergence of the iterative controller towards the optimal controller are proven, as updated by VI steps;
(b) an original validation on a visual servo tracking system comprising of an inexpensive and modern hardware processing platform (Arduino) combined with the software

platform MATLAB. The visual tracking control validation uses the new state representation comprising of present and past I/O data samples;

(c) the proposed MFVI-RL (implemented with NN approximator for the Q-value function and with linear controller parameterization) is compared with:
   (1) a model-based control strategy aiming at the same objective, namely a two-degree-of-freedom (2DOF) controller for ORM tracking; and
   (2) with a competitive VSFRT linearly parameterized controller also sharing the same learning control objective and the same underlying virtual state representation.

The learning process for MFVI-RL and VSFRT utilizes a form of pseudo-state-feedback control where, in fact, I/O data is used only for reconstructing the virtual state.

The paper is organized as next presented. Section 2 deals with VI-RL convergence under general assumptions. Section 3 integrates several subsections, such as: the visual servo system hardware and software description, followed by the LORM tracking problem formulation, subsequently solved by 2DOF control, by VI-RL algorithm and by VSFRT algorithm. All three are presented in detail. Performance comparisons reveal the learned control performance and offer the achieved results' interpretation.

2. The VI-RL Convergence Analysis for ORM Tracking Systems

The VI-RL convergence analysis is first performed in terms of the original "V" cost function, for ORM tracking systems for which particular penalty functions are sought. The result is then straightforwardly extended to the case of extended "Q" cost function, which characterizes the subsequent MFVI-RL implementation where the controlled system is assumed with unknown dynamics. Furthermore, the derivations are based on a state-space model, however this will be relaxed later on since a new state-space representation is reconstructed from I/O system data samples, using the virtual state transformation concept. Which implies that from any I/O nonlinear unknown dynamics system model which is assumed to be observable, we could arrive at a state-space model, with unknown dynamics a but measurable state. This will be clearly illustrated in the validation case study. For now, let the nonlinear unknown state-space system be

$$\begin{cases} \boldsymbol{s}_{k+1} = \boldsymbol{f}(\boldsymbol{s}_k, \boldsymbol{u}_k), \\ \boldsymbol{y}_k = \boldsymbol{h}(\boldsymbol{s}_k), \end{cases} \tag{1}$$

defined in discrete-time, comprising of a transition function equation and an output equation, respectively. The system state is $\boldsymbol{s}_k = [s_{k,1} \dots s_{k,n}]^T \in \Omega_S \subset \Re^n$, the system's input is $\boldsymbol{u}_k = [u_{k,1}, \dots, u_{k,m_u}]^T \in \Omega_U \subset \Re^{m_u}$, and the measured (and controlled) output is denoted as $\boldsymbol{y}_k = [y_{k,1}, \dots, y_{k,p}]^T \in \Omega_Y \subset \Re^p$. The functions $\boldsymbol{f}, \boldsymbol{h}$ are assumed as unknown on their definition domains and also continuously differentiable.

The cost function to be minimized is the summed penalties over an infinite horizon, which reduces to the control problem

$$\begin{gathered} C^* = \underset{C}{argmin} V(\boldsymbol{s}_k) = \sum_{i=k}^{\infty} \mho(\boldsymbol{s}_i, \boldsymbol{u}_i), \\ \text{s.t. } \boldsymbol{u}_i = C(\boldsymbol{s}_i), \end{gathered} \tag{2}$$

where $\mho(\boldsymbol{s}_i, \boldsymbol{u}_i) : \Omega_S \to \Re^n$ is the penalty function which is dependent only on $\boldsymbol{s}_i$, and has the property that $\mho(\boldsymbol{s}_i, \boldsymbol{u}_i) \geq 0$, $\forall \boldsymbol{s}_i \in \Omega_S$, $\forall \boldsymbol{u}_i \in \Omega_U$. The penalty function has the value $\mho(\boldsymbol{s}_i, \boldsymbol{u}_i) = 0$ only when $\boldsymbol{s}_k = \boldsymbol{s}_\mathbf{G}$, with $\boldsymbol{s}_\mathbf{G}$ some goal (or target) state.

Observation 1. In the case of linear quadratic control, the penalty function is commonly $\mho(\boldsymbol{s}_i, \boldsymbol{u}_i) = \boldsymbol{s}_i^T \mathbf{Q} \boldsymbol{s}_i + \boldsymbol{u}_i^T \mathbf{R} \boldsymbol{u}_i$ with respect to regulation of the system state towards zero. Here, $\mathbf{Q} \geq 0$ and $\mathbf{R} > 0$ are positive definite penalty matrices on the state energy and on the control input energy, respectively. It is known that the missing input energy penalization (for $\mathbf{R} = 0$) renders the linear optimal control unfeasible. In the case of ORM tracking

systems where the time-varying goal state is the result of a reference model dynamic system, it is possible to define a penalty function independent on the control input energy, e.g., as in $\mho(\boldsymbol{s}_i, \boldsymbol{u}_i) = (\boldsymbol{s}_i - \boldsymbol{s}_\mathbf{G})^T \mathbf{Q}(\boldsymbol{s}_i - \boldsymbol{s}_\mathbf{G})$. This would eventually lead to a feasible optimal control solution. We preserve notation $\mho(\boldsymbol{s}_i, \boldsymbol{u}_i)$ even if there is no explicit dependence on $\boldsymbol{u}_i$ since this will prove to be useful when acknowledging the role of the control input $\boldsymbol{u}_i$ in transiting from one state $\boldsymbol{s}_k$ to another one $\boldsymbol{s}_{k+1}$.

Let $V^*(\boldsymbol{s}_k)$ and $C_j^*(\boldsymbol{s}_k)$ be the optimal cost and the optimal controller. In the following, $\underset{\boldsymbol{u}}{min}\{.\}$ and $\underset{C}{min}\{.\}$ will imply the same operation under the constraint that $\boldsymbol{u} = C(\boldsymbol{s})$. Starting with an initial controller C_0 and V-function $V_0(\boldsymbol{s}_k) = 0$, for all possible combinations of $(\boldsymbol{s}_k, \boldsymbol{u}_k)$, $C_0(\boldsymbol{s}_k)$ is solved as follows:

$$C_0(\boldsymbol{s}_k) = arg\ \underset{\boldsymbol{u}}{min}\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + V_0(\boldsymbol{s}_{k+1})\} = arg\ \underset{\boldsymbol{u}}{min}\{\mho(\boldsymbol{s}_k, \boldsymbol{u})\}. \tag{3}$$

Having the policy determined, the c.f. update renders

$$V_1(\boldsymbol{s}_k) = \mho(\boldsymbol{s}_k, C_0(\boldsymbol{s}_k)) + V_0(\boldsymbol{s}_{k+1}). \tag{4}$$

Then, for the iterative VI algorithm the following steps are alternated at each j-th iteration:

S1. Update the controller

$$C_j(\boldsymbol{s}_k) = arg\ \underset{\boldsymbol{u}}{min}\big\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + V_j(\boldsymbol{s}_{k+1})\big\}. \tag{5}$$

S2. Update the V-function

$$V_{j+1}(\boldsymbol{s}_k) = \mho\big(\boldsymbol{s}_k, C_j(\boldsymbol{s}_k)\big) + V_j(\boldsymbol{s}_{k+1})\ . \tag{6}$$

Assumption 1. *We assume that the system is fully state controllable. This means that any state $\boldsymbol{s}_k \subset \Omega_S$ can be accessed from any other initial state in a finite number of time steps.*

Assumption 2. *We assume all the states from the process have been uniformly visited, and each possible action was tried for each possible state. This is equivalent to a complete fully state exploration, that reveals all the process dynamics.*

Definition 1. *The transition distance $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = \varpi$* is defined as the number ϖ of minimum state transitions from $\boldsymbol{s}_k$ to $\boldsymbol{s}_G$ using a succession of commands $\boldsymbol{u}_k, \boldsymbol{u}_{k+1}, \boldsymbol{u}_{k+2}, \ldots, \boldsymbol{u}_{k+\varpi-1}$.

Definition 2. *We define a controller $C_j(\boldsymbol{s}_k)$* to be admissible with respect to (2), if it stabilizes the system (1) from state $\boldsymbol{s}_k \in \Omega_S$ to the goal state $\boldsymbol{s}_G$, with $V(\boldsymbol{s}_k)$ *finite.*

Lemma 1. *The command selection operation performed at each j^{th} VI-RL iteration consists of selecting the minimum value of the series rendered by the next $j-1$ successive state transitions penalties*

$$\mho(\boldsymbol{s}_k, \boldsymbol{u}) + \mho\big(\boldsymbol{s}_{k+1}, C_{j-1}(\boldsymbol{s}_{k+1})\big) + \mho\big(\boldsymbol{s}_{k+1}, C_{j-1}(\boldsymbol{s}_{k+1})\big) + \ldots + \mho\Big(\boldsymbol{s}_{k+j}, C_1\Big(\boldsymbol{s}_{k+j}\Big)\Big). \tag{7}$$

Proof. Having $V_0(\boldsymbol{s}_k) = 0, \forall \boldsymbol{s}_k \in \Omega_S$ and the controller update $C_0(\boldsymbol{s}_k) = arg\ \underset{\boldsymbol{u}}{min}\{\mho(\boldsymbol{s}_k, \boldsymbol{u})\}$, the c.f. and controller update for j = 1 are

$$V_1(\boldsymbol{s}_k) = \mho(\boldsymbol{s}_k, C_0(\boldsymbol{s}_k)) + V_0(\boldsymbol{s}_{k+1}) = \mho(\boldsymbol{s}_k, C_0(\boldsymbol{s}_k))\ . \tag{8}$$

$$C_1(\boldsymbol{s}_k) = arg\ \underset{\boldsymbol{u}}{min}\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + V_1(\boldsymbol{s}_{k+1})\} = arg\ \underset{\boldsymbol{u}}{min}\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + \mho(\boldsymbol{s}_{k+1}, C_0(\boldsymbol{s}_{k+1}))\}\ . \tag{9}$$

This step consists of finding the command $\boldsymbol{u}$ that transitions from the current state $\boldsymbol{s}_k$ to the next state $\boldsymbol{s}_{k+1}$ ($\boldsymbol{s}_k \xrightarrow{\boldsymbol{u}} \boldsymbol{s}_{k+1}$), rendering the minimal value for $\mho(\boldsymbol{s}_k, \boldsymbol{u}) + \mho(\boldsymbol{s}_{k+1}, C_0(\boldsymbol{s}_{k+1}))$.

For iteration $j = 2$, the c.f. and the controller update are

$$V_2(\boldsymbol{s}_k) = \mho(\boldsymbol{s}_k, C_1(\boldsymbol{s}_k)) + V_1(\boldsymbol{s}_{k+1}) = \mho(\boldsymbol{s}_k, C_1(\boldsymbol{s}_k)) + \mho(\boldsymbol{s}_{k+1}, C_0(\boldsymbol{s}_{k+1})), \tag{10}$$

$$C_2(\boldsymbol{s}_k) = arg\ \min_{\boldsymbol{u}}\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + V_2(\boldsymbol{s}_{k+1})\} = arg\ \min_{\boldsymbol{u}}\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + \mho(\boldsymbol{s}_{k+1}, C_1(\boldsymbol{s}_{k+1})) + \mho(\boldsymbol{s}_{k+2}, C_0(\boldsymbol{s}_{k+2}))\}. \tag{11}$$

Since $V_2(\boldsymbol{s}_{k+1})$ can be written as a sum of penalties, this operation can be expressed as the selection of the two-state transitions with minimum cost. This step finds the optimal command in the first step of the *two-stage* state transitions series, depicted as

$$\boldsymbol{s}_k \xrightarrow{\boldsymbol{u}^*} \boldsymbol{s}_{k+1} \overset{C_1(\boldsymbol{s}_{k+1})}{\rightarrow} \boldsymbol{s}_{k+2}. \tag{12}$$

Then, the c.f. update at iteration $j = 3$ is

$$V_3(\boldsymbol{s}_k) = \mho(\boldsymbol{s}_k, C_2(\boldsymbol{s}_k)) + V_2(\boldsymbol{s}_{k+1}) = \mho(\boldsymbol{s}_k, C_2(\boldsymbol{s}_k)) + \mho(\boldsymbol{s}_{k+1}, C_1(\boldsymbol{s}_{k+1})) + \mho(\boldsymbol{s}_{k+2}, C_0(\boldsymbol{s}_{k+2})). \tag{13}$$

Following the reasoning, for any given iteration j, the cost function update is

$$\begin{aligned} V_j(\boldsymbol{s}_k) \quad &= \mho\big(\boldsymbol{s}_k, C_{j-1}(\boldsymbol{s}_k)\big) + V_{j-1}(\boldsymbol{s}_{k+1}) \\ &= \mho\big(\boldsymbol{s}_k, C_{j-1}(\boldsymbol{s}_k)\big) + V_{j-1}\big(f\big(\boldsymbol{s}_k, C_{j-1}(\boldsymbol{s}_k)\big)\big) \\ &= \mho\big(\boldsymbol{s}_k, C_{j-1}(\boldsymbol{s}_k)\big) + \mho\big(\boldsymbol{s}_{k+1}, C_{j-2}(\boldsymbol{s}_{k+1})\big) + V_{j-2}(\boldsymbol{s}_{k+2}) = \\ &= \mho\big(\boldsymbol{s}_k, C_{j-1}(\boldsymbol{s}_k)\big) + \sum_{n=1}^{j-1} \mho\big(\boldsymbol{s}_{k+n}, C_{j-n-1}(\boldsymbol{s}_{k+n})\big). \end{aligned} \tag{14}$$

The controller improvement step is then

$$\begin{gathered} C_j(\boldsymbol{s}_k) = arg\ \min_{\boldsymbol{u}}\big\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + V_j(\boldsymbol{s}_{k+1})\big\} = \\ arg\ \min_{\boldsymbol{u}}\Big\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + \mho\big(\boldsymbol{s}_k, C_{j-1}(\boldsymbol{s}_k)\big) + \textstyle\sum_{n=1}^{j-1} \mho\big(\boldsymbol{s}_{k+n}, C_{j-n-1}(\boldsymbol{s}_{k+n})\big)\Big\}. \end{gathered} \tag{15}$$

This step consists of finding the optimal command $\boldsymbol{u}^* = C_j(\boldsymbol{s}_k)$ in the first step of the j-stage state transitions series

$$\boldsymbol{s}_k \xrightarrow{\boldsymbol{u}^*} \boldsymbol{s}_{k+1} \overset{C_{j-1}(\boldsymbol{s}_{k+1})}{\rightarrow} \boldsymbol{s}_{k+2} \overset{C_{j-2}(\boldsymbol{s}_{k+2})}{\rightarrow} \boldsymbol{s}_{k+3} \ldots \boldsymbol{s}_{k+j-1} \overset{C_1(\boldsymbol{s}_{k+j})}{\rightarrow} \boldsymbol{s}_{k+j}, \tag{16}$$

which concludes the proof. □

Lemma 2. *For each iteration $j \in [0, \infty)$, the cost function of the goal state $\boldsymbol{s}_G$ is $V_j(\boldsymbol{s}_G) = 0$, and the controller $C_j(\boldsymbol{s}_G)$ maintains the system in state $\boldsymbol{s}_G$. Consequently, the c.f. of any state $\boldsymbol{s}_k \neq \boldsymbol{s}_G$ is $V_j(\boldsymbol{s}_k) > 0$.*

Proof. Since the initial cost function for any state $\boldsymbol{s}_k$ is $V_0(\boldsymbol{s}_k) = 0$, the controller update for any state $\boldsymbol{s}_k$ at $j = 0$ is

$$C_0(\boldsymbol{s}_G) = arg\ \min_{\boldsymbol{u}}\{\mho(\boldsymbol{s}_G, \boldsymbol{u}) + V_0(\boldsymbol{s}_{k+1})\} = arg\ \min_{\boldsymbol{u}}\{\mho(\boldsymbol{s}_G, \boldsymbol{u})\}\ . \tag{17}$$

The VI-update for the state $\boldsymbol{s}_G$ at iteration $j = 1$ is

$$V_1(\boldsymbol{s}_G) = \mho(\boldsymbol{s}_G, \boldsymbol{C}_0(\boldsymbol{s}_G)) + V_0(\boldsymbol{s}_{k+1}) = \mho(\boldsymbol{s}_G, \boldsymbol{C}_0(\boldsymbol{s}_G)) + 0 = 0, \tag{18}$$

and for any state $\boldsymbol{s}_k \neq \boldsymbol{s}_G$ is

$$V_1(\boldsymbol{s}_k) = \mho(\boldsymbol{s}_k, \boldsymbol{C}_0(\boldsymbol{s}_k)) + V_0(\boldsymbol{s}_{k+1}) = \mho(\boldsymbol{s}_k, \boldsymbol{C}_0(\boldsymbol{s}_k)) > 0 \tag{19}$$

The controller update

$$C_1(\boldsymbol{s}_k) = arg\ \underset{\boldsymbol{u}}{min}\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + V_1(\boldsymbol{s}_{k+1})\}, \tag{20}$$

selects for the state $\boldsymbol{s}_G$ the command $\boldsymbol{u}$ that takes the system to the state $\boldsymbol{s}_{k+1} = \boldsymbol{s}_G = f(\boldsymbol{s}_k, \boldsymbol{u})$.

Assuming by induction that $V_{j-1}(\boldsymbol{s}_G) = 0$, the controller updates as

$$C_{j-1}(\boldsymbol{s}_G) = arg\ \underset{\boldsymbol{u}}{min}\{\mho(\boldsymbol{s}_G, \boldsymbol{u}) + V_{j-1}(\boldsymbol{s}_{k+1})\}. \tag{21}$$

Since $\mho(\boldsymbol{s}_G, \boldsymbol{u}) = 0$ for any $\boldsymbol{u} \in \Omega_U$, and $V_{j-1}(\boldsymbol{s}_G) = 0$, the VI update will be $C_{j-1}(\boldsymbol{s}_G) = arg\ \underset{\boldsymbol{u}}{min}\{V_{j-1}(\boldsymbol{s}_G) = 0\}$, meaning that the $arg\ \underset{\boldsymbol{u}}{min}$ operation will select for state $\boldsymbol{s}_G$ the command $\boldsymbol{u}$ that maintains the system to the state $\boldsymbol{s}_{k+1} = \boldsymbol{s}_G = f(\boldsymbol{s}_G, \boldsymbol{u})$.

The c.f. update renders

$$V_j(\boldsymbol{s}_G) = \mho\big(\boldsymbol{s}_G, C_{j-1}(\boldsymbol{s}_G)\big) + V_{j-1}(\boldsymbol{s}_{k+1}) = \mho\big(\boldsymbol{s}_G, C_{j-1}(\boldsymbol{s}_G)\big) + 0 = 0, \tag{22}$$

proving thus that $V_j(\boldsymbol{s}_G) = 0, \forall j \in [0, \infty)$.

Also, accordingly to Lemma 1, the cost function $V_j(\boldsymbol{s}_k)$ of any $\boldsymbol{s}_k \neq \boldsymbol{s}_G$ can be written as

$$\begin{aligned} V_j(\boldsymbol{s}_k) &= \mho\big(\boldsymbol{s}_k, C_{j-1}(\boldsymbol{s}_k)\big) + \sum_{n=1}^{j-1} \mho\big(\boldsymbol{s}_{k+n}, C_{j-n-1}(\boldsymbol{s}_{k+n})\big) \\ &= \mho\big(\boldsymbol{s}_k, C_{j-1}(\boldsymbol{s}_k)\big) \\ &+ \sum_{n=1}^{j-2} \mho\big(\boldsymbol{s}_{k+n}, C_{j-n-1}(\boldsymbol{s}_{k+n})\big) + \mho\Big(\boldsymbol{s}_{k+j-1}, C_0\Big(\boldsymbol{s}_{k+j-1}\Big)\Big). \end{aligned} \tag{23}$$

Since $\mho\Big(\boldsymbol{s}_{k+j-1}, C_0\Big(\boldsymbol{s}_{k+j-1}\Big)\Big) > 0$, this proves that $V_j(\boldsymbol{s}_k) > 0$ □

Lemma 3. *All c.f. $V_j(\boldsymbol{s}_k)$ that have its series of penalties ending in the goal state $\boldsymbol{s}_G$ are finite, as $j \to \infty$.*

Proof. A necessary condition for convergent series is that the sequence of its elements must converge to zero. Having the sequence of the penalties $[\mho(\boldsymbol{s}_k, \boldsymbol{u}), \mho\big(\boldsymbol{s}_{k+1}, C_{j-1}(\boldsymbol{s}_{k+1})\big), \mho\big(\boldsymbol{s}_{k+1}, C_{j-1}(\boldsymbol{s}_{k+1})\big), \ldots, \mho\big(\boldsymbol{s}_{k+n}, C_{j-n}(\boldsymbol{s}_{k+n})\big)]$, it is required that $\mho\big(\boldsymbol{s}_{k+n}, C_{j-n}(\boldsymbol{s}_{k+n})\big) \to 0$ as $j \to \infty$. Since only the state $\boldsymbol{s}_G$ has penalty $\mho(\boldsymbol{s}_G, \boldsymbol{u}) = 0$, it follows that the sequence must converge to $\mho\big(\boldsymbol{s}_{k+n}, C_{j-n}(\boldsymbol{s}_{k+n})\big) \to \mho(\boldsymbol{s}_G, \boldsymbol{u}) = 0$. With this result, it follows that the finite c.f. can be written as

$$\mho(\boldsymbol{s}_k, \boldsymbol{u}_k) + \textstyle\sum_{n=1}^{p} \mho\big(\boldsymbol{s}_{k+n}, C_{j-n-1}(\boldsymbol{s}_{k+n})\big) + \underbrace{\textstyle\sum_{m=p+1}^{j-1} \mho\big(\boldsymbol{s}_G, C_{j-m-1}(\boldsymbol{s}_G)\big)}_{=0}. \tag{24}$$

As $j \to \infty$, the c.f. is

$$\begin{aligned} &\mho(\boldsymbol{s}_k, \boldsymbol{u}_k) + \sum_{n=1}^{p} \mho\big(\boldsymbol{s}_{k+n}, C_{j-n-1}(\boldsymbol{s}_{k+n})\big) + \underbrace{\sum_{m=p+1}^{\infty} \mho\big(\boldsymbol{s}_G, C_{j-m-1}(\boldsymbol{s}_G)\big)}_{=0} \\ &= \mho(\boldsymbol{s}_k, \boldsymbol{u}_k) + \textstyle\sum_{n=1}^{p} \mho\big(\boldsymbol{s}_{k+n}, C_{j-n}(\boldsymbol{s}_{k+n})\big). \end{aligned} \tag{25}$$

This proves that all c.f. that have its penalties sequence ending on state $\boldsymbol{s}_G$ are finite and remain constant as $j \to \infty$. □

Theorem 1. *Let $V_j(\boldsymbol{s}_k)$ be the c.f. of a state $\boldsymbol{s}_k \in \Omega_S$ with $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = \varpi$ defined on the set $\Omega_S \subset \Re^n$. Then, $V_j(\boldsymbol{s}_k)$ converges on Ω_S to a limit $V(\boldsymbol{s}_k)$ if, whenever $j \geq \varpi + 1$, it follows that $\big|V_j(\boldsymbol{s}_k) - V(\boldsymbol{s}_k)\big| = 0$. Moreover, as $j \geq \varpi + 1$, $V_j(\boldsymbol{s}_k) \to V^*(\boldsymbol{s}_k)$ and $C^j(\boldsymbol{s}_k) \to C_j^*(\boldsymbol{s}_k)$.*

Proof. Let $V_0(\boldsymbol{s}_k) = 0, \forall \boldsymbol{s}_k \in \Omega_S$. We first show that $V_j(\boldsymbol{s}_k)$ is convergent for $\boldsymbol{s}_k \in \Omega_S$ with $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = 1$.

Having the controller update for $j = 0$

$$C_0(\boldsymbol{s}_k) = arg\ \underset{\boldsymbol{u}}{min}\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + V_0(\boldsymbol{s}_{k+1})\} = arg\ \underset{\boldsymbol{u}}{min}\{\mho(\boldsymbol{s}_k, \boldsymbol{u})\}, \tag{26}$$

the c.f. and controller update for $j = 1$ are

$$\begin{gathered} V_1(\boldsymbol{s}_k) = \mho(\boldsymbol{s}_k, C_0(\boldsymbol{s}_k)) + V_0(\boldsymbol{s}_{k+1}) = \mho(\boldsymbol{s}_k, C_0(\boldsymbol{s}_k)), \\ C_1(\boldsymbol{s}_k) = arg\ \underset{\boldsymbol{u}}{min}\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + V_1(\boldsymbol{s}_{k+1})\} = arg\ \underset{\boldsymbol{u}}{min}\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + \mho(\boldsymbol{s}_{k+1}, C_0(\boldsymbol{s}_{k+1}))\}, \end{gathered} \tag{27}$$

For all the states $\boldsymbol{s}_k \in \Omega_S$ with $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = 1$, the optimal command $\boldsymbol{u}$ that transitions to $\boldsymbol{s}_{k+1} = \boldsymbol{s}_G$ will be selected. So, the c.f. of all states $\boldsymbol{s}_k$ with $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = 1$ at iteration $j = 2$, can be written as

$$V_2(\boldsymbol{s}_k) = \mho(\boldsymbol{s}_k, C_1(\boldsymbol{s}_k)) + V_1(\boldsymbol{s}_G) = \mho(\boldsymbol{s}_k, C_1(\boldsymbol{s}_k)). \tag{28}$$

At any at iteration j, the c.f. and controller update are

$$V_j(\boldsymbol{s}_k) = \mho\big(\boldsymbol{s}_k, C_{j-1}(\boldsymbol{s}_k)\big) + V_{j-1}(\boldsymbol{s}_{k+1}) = \big(\boldsymbol{s}_k, C_{j-1}(\boldsymbol{s}_k)\big) + \textstyle\sum_{n=1}^{j-1} \mho\big(\boldsymbol{s}_{k+n}, C_{j-n}(\boldsymbol{s}_{k+n})\big), \tag{29}$$

$$C_j(\boldsymbol{s}_k) = arg\ \underset{\boldsymbol{u}}{min}\big\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + V_j(\boldsymbol{s}_{k+1})\big\} = arg\ \underset{\boldsymbol{u}}{min}\Big\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + \textstyle\sum_{n=1}^{j-1} \mho\big(\boldsymbol{s}_{k+n}, C_{j-n}(\boldsymbol{s}_{k+n})\big)\Big\}. \tag{30}$$

Since according to Lemma 1 $V_{j-1}(\boldsymbol{s}_G) = 0$, the selected command $\boldsymbol{u}$ after the controller update (5) will be the one that transitions the system to $\boldsymbol{s}_{k+1} = \boldsymbol{s}_G$.

Assuming that $V_{j-1}(\boldsymbol{s}_G) = 0$, after the controller update (5), the command $\boldsymbol{u} = C_j(\boldsymbol{s}_k)$ that transitions to $\boldsymbol{s}_{k+1} = \boldsymbol{s}_G$ will be selected. The c.f. relative to iteration $j + 1$ is going to be

$$V_{j+1}(\boldsymbol{s}_k) = \mho\big(\boldsymbol{s}_k, C_j(\boldsymbol{s}_k)\big) + V_j(\boldsymbol{s}_{k+1} = \boldsymbol{s}_G) = \mho\big(\boldsymbol{s}_k, C_j(\boldsymbol{s}_k)\big) + 0 = \mho\big(\boldsymbol{s}_k, C_j(\boldsymbol{s}_k)\big). \tag{31}$$

It can be seen that for all states with $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = 1$, for any $j \in [2, \infty)$, $V(\boldsymbol{s}_k) = V_j(\boldsymbol{s}_k) = \mho\big(\boldsymbol{s}_k, C_j(\boldsymbol{s}_k)\big)$, meaning that $\big|V_j(\boldsymbol{s}_k) - V(\boldsymbol{s}_k)\big| = 0$. This follows since at each iteration j, the improved controller $\boldsymbol{u} = C_j(\boldsymbol{s}_k)$ transitions $\boldsymbol{s}_k$ to $\boldsymbol{s}_G$.

Assuming that $V_{j-1}(\boldsymbol{s}_{k+1})$ is convergent for all states $\boldsymbol{s}_{k+1} \in \Omega_S$ with $d(\boldsymbol{s}_{k+1}, \boldsymbol{s}_G) = \varpi$, it is next proven that $V_j(\boldsymbol{s}_k)$ is convergent for all states $\boldsymbol{s}_k \in \Omega_S$ with $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = \varpi + 1$.

Having the controller update as (5) at iteration $j = \varpi$, it is assumed for states $\boldsymbol{s}_k \in \Omega_S$ with $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = \varpi + 1$ that there exists a command $\boldsymbol{u}$ that transitions the system to a state $\boldsymbol{s}_{k+1} \in \Omega_S$ with $d(\boldsymbol{s}_{k+1}, \boldsymbol{s}_G) = \varpi$ that has the c.f. finite. According to Lemma 2, the *argmin* operator has to choose between different series of j elements, where the one associated with state $\boldsymbol{s}_{k+1}$ that has the transition distance $d(\boldsymbol{s}_{k+1}, \boldsymbol{s}_G) = \varpi$, will therefore be selected instead of any other state $\boldsymbol{s}_{k+1}$ with $d(\boldsymbol{s}_{k+1}, \boldsymbol{s}_G) \geq \varpi + 1$, since it represents the minimum penalty series. As $j \to \infty$, the *argmin* operator will also select the command $\boldsymbol{u}$ that transitions to the state that has $d(\boldsymbol{s}_{k+1}, \boldsymbol{s}_G) = \varpi$, since according to *Lemma* 2, its c.f. will remain constant and therefore finite.

Showing that $V_j(\boldsymbol{s}_k) = V(\boldsymbol{s}_k)$ from $j \geq \varpi + 1$, for a state $\boldsymbol{s}_k \in \Omega_S$ with $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = \varpi + 1$, proves that the sequence of $\varpi + 1$ optimal commands that bring the system to the goal state $\boldsymbol{s}_G$ will be selected as $j \to \infty$. Updating $C_j(\boldsymbol{s}_k)$ using (5), will find the controller that bring the system to the goal state using optimal commands, meaning that the found controller is the optimal controller $C_j^*(\boldsymbol{s}_k)$. Since the optimal controller can be derived only from the optimal c.f., this means that $V_j(\boldsymbol{s}_k) = V(\boldsymbol{s}_k) = V^*(\boldsymbol{s}_k)$, from $j \geq \varpi + 1$. This completes the proof. □

The VI convergence proved in Theorem 1 is based on the true V-function parameterization, and hence can be used only for systems for which the c.f. is known and for table-based VI implementations with finite states and actions. For continuous-time nonlinear processes as in (1), a broad range of function approximators need to be used for the V-function

$V_j(\boldsymbol{s}_k)$ and for the controller $C_j(\boldsymbol{s}_k)$. Employing these approximations in the VI operations, some approximation error is added at each iteration, enforcing thus the V-function and the controller to be suboptimal. It is next proved that the VI learning scheme also converges under approximation errors.

Having the c.f. approximation error function $\delta_j(\boldsymbol{s}_k) : \Omega_S \to \Re$ and starting with an initial V-function $\widetilde{V}_0$, the controller and c.f update are represented as

SA1. Controller update

$$\widetilde{C}_j(\boldsymbol{s}_k) = arg\ \min_{\boldsymbol{u}}\Big\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + \widetilde{V}_j(\boldsymbol{s}_{k+1})\Big\}, \tag{32}$$

SA2. V-function update

$$\widetilde{V}_{j+1}(\boldsymbol{s}_k) = \mho\Big(\boldsymbol{s}_k, \widetilde{C}_j(\boldsymbol{s}_k)\Big) + \widetilde{V}_j(\boldsymbol{s}_{k+1}) = \mho\Big(\boldsymbol{s}_k, \widetilde{C}_j(\boldsymbol{s}_k)\Big) + V_j(\boldsymbol{s}_{k+1}) + \delta_j(\boldsymbol{s}_{k+1}). \tag{33}$$

Equations (32) and (33) show that the approximate V-function and controller follow the real ones with a small approximation deviation $\delta_j(\boldsymbol{s}_{k+1})$. As long as the true V-function parameterization is unknown, the residuals $\left|\delta_j(\boldsymbol{s}_{k+1})\right| > 0$, making the approximated VI (AVI) updates (32), (33) differ from (3), (4). Even if the obtained controller is suboptimal, it must be admissible to make the system reach the goal state $\boldsymbol{s}_G$. To prove the existence of an approximation error $\delta_j(\boldsymbol{s}_{k+1})$ upper bound at each iteration j such that the controller $\widetilde{C}_j(\boldsymbol{s}_k)$ will be admissible for each state $\boldsymbol{s}_k$, with the c.f. constant and finite in the original VI-update, the following definition must be given.

Definition 3. *We define Λ_j as the set that contains all the states $\boldsymbol{s}_k$ for which the controller $\widetilde{C}_j(\boldsymbol{s}_k)$ is admissible. Additionally, the complement set $\overline{\Lambda}_j$ represents the set of all states $\boldsymbol{s}_k$ for which the controller $\widetilde{C}_j(\boldsymbol{s}_k)$ is not admissible.*

Theorem 2. *For the c.f. $\widetilde{V}_{j+1}(\boldsymbol{s}_k)$ at iteration j, $\forall \boldsymbol{s}_k \in \Omega_S$ that under any command $\boldsymbol{u}_k \in \Omega_U$ will transition the system to a state $\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1} \in \Lambda_j$, the c.f. approximation error of the state $\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1} \in \Lambda_j$, denoted as $\delta_j\big(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1} \in \Lambda_j\big)$, respects the following condition*

$$\mho\Big(\boldsymbol{s}_k, \widetilde{C}_j(\boldsymbol{s}_k)\Big) + V_j(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1} \in \Lambda_j) + \delta_j(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1} \in \Lambda_j) < \mho\Big(\boldsymbol{s}_k, \widetilde{C}_j(\boldsymbol{s}_k)\Big) + V_j(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1} \in \overline{\Lambda}_j) + \delta_j(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1} \in \overline{\Lambda}_j), \tag{34}$$

in order to make the controller $\widetilde{C}_j(x_k)$ admissible. Additionally, as $j \to \infty$, $\Lambda_j \to \Omega_S$.

Proof. Let $\widetilde{V}_0(\boldsymbol{s}_k) = 0$, $\delta_0\ (\boldsymbol{s}_k\ ) = 0$, $\Lambda_0 = \varnothing$, $\forall \boldsymbol{s}_k \in \Omega_X$. The controller update is

$$\widetilde{C}_0(\boldsymbol{s}_k) = \underset{\boldsymbol{u}}{argmin}\Big\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + \widetilde{V}_0(\boldsymbol{s}_{k+1})\Big\} = \underset{\boldsymbol{u}}{argmin}\{\mho(\boldsymbol{s}_k, \boldsymbol{u})\}. \tag{35}$$

According to Lemma 1, for the state $\boldsymbol{s}_k = \boldsymbol{s}_G$ with $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = 0$, the optimal command maintains the system to the same state $\boldsymbol{s}_{k+1} = \boldsymbol{s}_G$. Since $\delta_0(\boldsymbol{s}_G) = 0$, this command will be selected by the minimization operation, making thus $\Lambda_1 = \Lambda_0 \cup \{\boldsymbol{s}_k | d(\boldsymbol{s}_k, \boldsymbol{s}_G) = 0\}$. The c.f. and the controller update at $j = 1$ are

$$\widetilde{V}_1(\boldsymbol{s}_k) = \mho\Big(\boldsymbol{s}_k, \widetilde{C}_0(\boldsymbol{s}_k)\Big) + \widetilde{V}_0(\boldsymbol{s}_{k+1}) + \delta_1(\boldsymbol{s}_k) = \mho\Big(\boldsymbol{s}_k, \widetilde{C}_0(\boldsymbol{s}_k)\Big) + \delta_1(\boldsymbol{s}_k), \tag{36}$$

$$\widetilde{C}_1(\boldsymbol{s}_k) = arg\ \min_{\boldsymbol{u}}\Big\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + \widetilde{V}_1(\boldsymbol{s}_{k+1})\Big\} = arg\ \min_{\boldsymbol{u}}\Big\{\mho(\boldsymbol{s}_k, \boldsymbol{u}) + \mho\Big(\boldsymbol{s}_{k+1}, \widetilde{C}_0(\boldsymbol{s}_{k+1})\Big) + \delta_1(\boldsymbol{s}_{k+1})\Big\}. \tag{37}$$

According to Theorem 1, for $\boldsymbol{s}_k$ with $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = 1$, the optimal command transitions the system to the goal state $\boldsymbol{s}_{k+1} = \boldsymbol{s}_G$. Also, for the state $\boldsymbol{s}_k = \boldsymbol{s}_G$, the command that maintains the system to the goal state $\boldsymbol{s}_{k+1} = \boldsymbol{s}_G$ needs to be selected. If the c.f. approximation error $\delta_1(\boldsymbol{s}_G)$ has its value, such that the cost of transitioning to $\boldsymbol{s}_G$ from any state $\boldsymbol{s}_k$ with $d(\boldsymbol{s}_k, \boldsymbol{s}_G) = 1$ will be lower than any cost associated with $\forall \boldsymbol{s}_{k+1}$ with $d(\boldsymbol{s}_{k+1}, \boldsymbol{s}_G) > 0$, namely

$$\min_{\boldsymbol{u}}\Big\{\mho(\boldsymbol{s}_k,\boldsymbol{u})+\mho\Big(\boldsymbol{s}_G,\widetilde{C}_0(\boldsymbol{s}_G)\Big)+\delta_1(\boldsymbol{s}_G)\Big\}<\min_{\boldsymbol{u}}\Big\{\mho(\boldsymbol{s}_k,\boldsymbol{u})+\mho\Big(\boldsymbol{s}_{k+1},\widetilde{C}_0(\boldsymbol{s}_{k+1})\Big)+\delta_1(\boldsymbol{s}_{k+1})\Big\}, \tag{38}$$

then the $arg\ \min_{\boldsymbol{u}}$ operator will not be motivated to choose the command that transits the system to another state rather than the goal state $\boldsymbol{s}_G$. By selecting the command $\boldsymbol{u}$ for the state $\boldsymbol{s}_k$, $\Lambda_2=\Lambda_1\cup\{\boldsymbol{s}_k|d(\boldsymbol{s}_k,\boldsymbol{s}_G)=1\}$. Having done so, the c.f. and controller update for $j=2$ are

$$\widetilde{V}_2(\boldsymbol{s}_k)=\mho\Big(\boldsymbol{s}_k,\widetilde{C}_1(\boldsymbol{s}_k)\Big)+\widetilde{V}_1(\boldsymbol{s}_{k+1})+\delta_2(\boldsymbol{s}_{k+1})=\mho\Big(\boldsymbol{s}_k,\widetilde{C}_1(\boldsymbol{s}_k)\Big)+V_1(\boldsymbol{s}_{k+1})+\delta_1(\boldsymbol{s}_{k+1})+\delta_2(\boldsymbol{s}_{k+1}), \tag{39}$$

$$\widetilde{C}_2(\boldsymbol{s}_k)=arg\ \min_{\boldsymbol{u}}\Big\{\mho(\boldsymbol{s}_k,\boldsymbol{u})+\widetilde{V}_2(\boldsymbol{s}_{k+1})+\delta_2(\boldsymbol{s}_{k+1})\Big\}. \tag{40}$$

For all states $\boldsymbol{s}_k$ with $d(\boldsymbol{s}_k,\boldsymbol{s}_G)=2$, the optimal command selected at this iteration by controller update (32) is the one that transitions the system to a state $\boldsymbol{s}_{k+1}$ with $d(\boldsymbol{s}_{k+1},\boldsymbol{s}_G)=1$. For all states $\boldsymbol{s}_k$ with $d(\boldsymbol{s}_k,\boldsymbol{s}_G)=1$, an admissible command can transition the system to the goal state $\boldsymbol{s}_{k+1}=\boldsymbol{s}_G$ or to any $\boldsymbol{s}_{k+1}|d(\boldsymbol{s}_{k+1},\boldsymbol{s}_G)=1$. Then, the approximation error $\delta_2(\boldsymbol{s}_{k+1})$ for $d(\boldsymbol{s}_{k+1},\boldsymbol{s}_G)\le 1$ must have the value such that

$$\min_{\boldsymbol{u}}\Big\{\mho(\boldsymbol{s}_k,\boldsymbol{u})+\widetilde{V}_2(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1}\in\Lambda_2)+\delta_2(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1}\in\Lambda_2)\Big\}<\min_{\boldsymbol{u}}\Big\{\mho(\boldsymbol{s}_k,\boldsymbol{u})+\widetilde{V}_2(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1}\in\overline{\Lambda}_2)+\delta_2(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1}\in\overline{\Lambda}_2)\Big\}, \tag{41}$$

making thus $\Lambda_3=\Lambda_2\cup\{\boldsymbol{s}_k|d(\boldsymbol{s}_k,\boldsymbol{s}_G)=2\}$.

At any iteration j, we assume that for some states $\boldsymbol{s}_k$ with $(\boldsymbol{s}_k,\boldsymbol{s}_G)=j$ there exists a command that transitions to $\boldsymbol{s}_{k+1}\in\Lambda_j$, $d(\boldsymbol{s}_{k+1},\boldsymbol{s}_G)=j-1$. The selected command $\boldsymbol{u}$ by controller update (35) for state $\boldsymbol{s}_k$ needs to transition the system to any state $\boldsymbol{s}_{k+1}$ with $d(\boldsymbol{s}_{k+1},\boldsymbol{s}_G)=j-1$, in order to make $\boldsymbol{s}_k\in\Lambda_{j+1}$. Also, for any state $\boldsymbol{s}_k$ with $d(\boldsymbol{s}_k,\boldsymbol{s}_G)\le j-1$ the selected command $\boldsymbol{u}$ can transition to any state $\boldsymbol{s}_{k+1}\in\Lambda_j$, to make $\boldsymbol{s}_k\in\Lambda_{j+1}$. The approximation error $\delta_j(\boldsymbol{s}_{k+1})$ for $d(\boldsymbol{s}_{k+1},\boldsymbol{s}_G)\le j-1$ must then hold the following inequality

$$\min_{\boldsymbol{u}}\Big\{\mho(\boldsymbol{s}_k,\boldsymbol{u})+\widetilde{V}_j(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1}\in\Lambda_j)+\delta_j(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1}\in\Lambda_j)\Big\}<\min_{\boldsymbol{u}}\Big\{\mho(\boldsymbol{s}_k,\boldsymbol{u})+\widetilde{V}_j(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1}\in\overline{\Lambda}_j)+\delta_j(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1}\in\overline{\Lambda}_j)\Big\}, \tag{42}$$

making $\Lambda_{j+1}=\Lambda_j\cup\{\boldsymbol{s}_k|d(\boldsymbol{s}_k,\boldsymbol{s}_G)=j\}$. □

Theorem 2 shows that by bounding the approximation error $\delta_j\big(\boldsymbol{s}_{k+1}\big|\boldsymbol{s}_{k+1}\in\Lambda_j\big)$ such that the selected command will transition to a state $\boldsymbol{s}_{k+1}\in\Lambda_j$, although the obtained controller is suboptimal, it will be restricted to be admissible, thus stabilizing the system asymptotically. This does not guarantee that $\widetilde{V}_j(\boldsymbol{s}_k)$ and $\widetilde{C}_j(\boldsymbol{s}_k)$ will remain constant for all iterations j, but it will ensure that their distance to the original V-function $V_j(\boldsymbol{s}_k)$ and the original controller $C_j(\boldsymbol{s}_k)$, will be dependent on the performance of the selected function approximators.

For the model-free implementation of the VI-RL algorithm, subsequently called MFVI-RL, the V-function is extended to the so-called Q-function that has both the state and the command at time k as its arguments. The Q-function update for an arbitrary command $\boldsymbol{u}_k$ in state $\boldsymbol{s}_k$ is

$$Q_j(\boldsymbol{s}_k,\boldsymbol{u}_k)=\min_{C(\cdot)}\big\{\mho(\boldsymbol{s}_k,\boldsymbol{u}_k))+Q_{j-1}(\boldsymbol{s}_{k+1},\ C(\boldsymbol{s}_{k+1}))\big\}=\min_{C(\cdot)}\big\{\mho(\boldsymbol{s}_k,\boldsymbol{u}_k))+V_{j-1}(\boldsymbol{s}_{k+1})\big\}. \tag{43}$$

The control update step at current iteration is represented by

$$C_j(\boldsymbol{s}_k)=arg\ \min_{\boldsymbol{u}}\big\{Q_j(\boldsymbol{s}_k,\boldsymbol{u})\big\}. \tag{44}$$

It results from *Theorem 1* that $\lim_{j\to\infty}V_j(\boldsymbol{s}_k)=V^*(\boldsymbol{s}_k)$ and that the right-hand side of Equation (29) embeds in fact the VI-RL update of the V-function, via the operation $\min_{C(\cdot)}\big\{V_{j-1}(\boldsymbol{s}_{k+1}\ )\big\}$. This means that $Q_j(\boldsymbol{s}_k,\boldsymbol{u}_k)$ and $V_j(\boldsymbol{s}_k)$ are paired. It follows that $\lim_{j\to\infty}Q_j(\boldsymbol{s}_k,\boldsymbol{u}_k)=Q^*(\boldsymbol{s}_k,\boldsymbol{u}_k)=\min_{C(\cdot)}\{\mho(\boldsymbol{s}_k,\boldsymbol{u}_k))+V^*(\boldsymbol{s}_{k+1}\ )\}$. This result also implies that

$\lim_{j\to\infty} C_j(\boldsymbol{s}_k) = C^*(\boldsymbol{s}_k)$. Using function approximators for both the Q-function and the controller, namely $\widetilde{Q}_j(\boldsymbol{s}_k, \boldsymbol{u}_k)$ and $\widetilde{C}_j(\boldsymbol{s}_k)$, the c.f. approximation error $\delta_j\left(\boldsymbol{s}_{k+1}\middle|\boldsymbol{s}_{k+1} \in \Lambda_{j-1}\right)$, must respect the condition $\min_{C(\cdot)}\left\{\upsilon(\boldsymbol{s}_k, \boldsymbol{u}_k) + V_j\left(\boldsymbol{s}_{k+1}\middle|\boldsymbol{s}_{k+1} \in \Lambda_{j-1}\right) + \delta_j\left(\boldsymbol{s}_{k+1}\middle|\boldsymbol{s}_{k+1} \in \Lambda_{j-1}\right)\right\}$ $< \min_{C(\cdot)}\left\{\upsilon(\boldsymbol{s}_k, \boldsymbol{u}_k) + V_j\left(\boldsymbol{s}_{k+1}|\boldsymbol{s}_{k+1} \in \overline{\Lambda}_{j-1}\right) + \delta_j\left(\boldsymbol{s}_{k+1}\middle|\boldsymbol{s}_{k+1} \in \overline{\Lambda}_{j-1}\right)\right\}$, in order to make the controller admissible for states $\boldsymbol{s}_k$ that under any command $\boldsymbol{u}_k \in \Omega_U$ will transition the system to a state $\boldsymbol{s}_{k+1}\middle|\boldsymbol{s}_{k+1} \in \Lambda_{j-1}$.

The MFVI-RL using Q-functions is shown convergent to the optimal control which must also be stabilizing in order to render a finite cost function. At this point, the problem formulation is sufficiently general, it will be made particular for the output reference model tracking in the case study.

The next section presents in detail the case study of a visual servo system built for output reference model tracking purpose. Tracking control is learned with MFVI-RL but also with competitive methods used for comparisons: a model-based 2DOF controller and a model-free VSFRT controller.

3. Visual Output Reference Model Tracking Control Design

3.1. The Visual Tracking Servo System Description

Building a hardware system from scratch assumes a well-structured plan, in which the desired behavior must be established from the system. Several components were used in the project, which will be presented below [31]: Arduino Nano, webcam, double H-bridge L298n, a micro actuated DC motor (ADCM) with reduction, two side wheels, an independent wheel with a supporting role, acrylic chassis, breadboard, power cord, webcam connection cable to PC, Arduino connection cable to the PC, and a power adapter.

The resulting hardware assembly has a chassis represented by an acrylic plate, having three wheels, two of which are the side ones, the left one being connected to the ADCM so that the assembly can perform rotational (yaw) movements, in order to track a fixed or mobile visual target seen through the webcam [31]. The third wheel is placed in the rear of the chassis; its free motion has solely the role of facilitating the whole assembly yaw motion. On top of the acrylic board and in front of the assembly, the webcam captures images to send back to the PC. The Arduino board was placed in the breadboard, from here making the connection with the ADCM being used, through the H-bridge. The assembly can be observed in Figure 1.

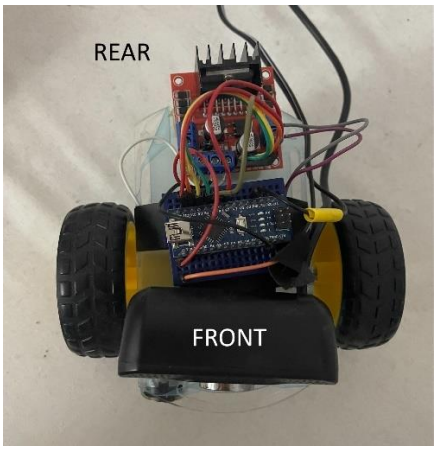

Figure 1. Hardware connection between the engine, motor driver and Arduino Board [31].

Sending commands from the MATLAB environment running on the PC to the ADCM, is handled via Arduino using an USB cable. Arduino sends the commands to the motor driver, which powers the ADCM. The L298n motor driver operates in this project with a single motor. It is powered by a 7.5 V input voltage from a power adapter connected to the

outlet that has an output voltage of 7.5 V. Part of this voltage is used to power the ADCM. The ADCM selection was made by connecting the D5 pin to one of the two "Enable" ports of the L298n. Thus, the rotating direction is controlled by the input pins I1 and I2 of the driver, receiving logical voltage level from pins D11 and D12 of Arduino. Depending on the state of these two pins (set or reset), the ADCM spin direction is set. For pin D11 having logical value 1 (5 V) and pin D12 having the value 0 (0 V), the motor will spin clockwise and when D11 has the value 0 and D12 the value 1, the ADCM will spin in counterclockwise direction. In each specific direction, the ADCM's angular speed is set by a PWM signal generated with Arduino, whose duty cycle value is set in MATLAB and transmitted from the PC. The ADCM is a carbon-brushed motor from Pololu, 6 V, reduction ration 1000:1, with 33 revolutions per minute at no load, drawing a nominal current of 0.1 amps. Due to the high reduction factor, it is ideal for fine low angular speed control.

However, the micro reduction ADCM used is not built for precise position control (as demanded by a video servo tracking system), while a dedicated one is costlier. The ADCM has a variable dead-zone (DZ) around about 0.156 V (per each rotational direction) which is time-varying due to the multiple magnetic and thermal effects that occur after longer time usage. A logical solution is to compensate the DZ, but due to the mentioned phenomena, the DZ thresholds are impossible to compensate exactly. However, an approximate compensation is actually performed, allowing acceptable positioning visual tracking accuracy [31].

In practice, there are certain video tracking platforms involving advanced image processing and features detections techniques, combined with advanced localization techniques of complex motion dynamics, such as unmanned aerial vehicles (UAVs), as done, e.g., in [32]. In work [32] the focus is to ensure tracking reliability under complex practical conditions, including huge scale variation, occlusion and long-term task. Other visual tracking solutions were proposed in [33–35]. Such a tracking approach relies on low level controllers which could be designed based on the proposed learning strategies carried out in this work. The main advantage of our platform is its cost efficiency and flexibility, allowing for affordable educational and research activities.

Further on, at the software level, color-based identification of the moving target object (in this project, a blue ball) was performed in MATLAB. The implementation of the object identification process involved several steps. A general position control goal is to track the desired object (blue ball) to keep its position in the image center.

After setting up the desired image capture settings such as resolution (640×360) and the RGB (red, green and blue) image color type, to be able to process it, the next step is to extract the RGB components. Three 2D matrices are extracted using the commands (in MATLAB code style): *r = img (:,:, 1); g = img (:,:, 2); b = img (:,:, 3)* where *img* is the captured image object. To emphasize the tracked object whose blue color is known inside the image, a single 2D matrix is obtained having each pixel intensity level denoted "*blueComponent*" calculated with (matrix element-wise operation) [31]

$$blueComponent = b - \frac{r}{2} - \frac{g}{2}. \tag{45}$$

The next step is to set some threshold values in order to identify the regions inside the image that are considered "sufficiently blue", to include the object to be identified. This is ensured by the *bwareafilt* function which extracts objects from an image depending on their size or their area. For the detection of unique objects, an analysis of the connected components is performed, in which case each blue region is assigned a unique ID. We ensure that the largest blue object inside the scene is the ball. The image center coordinates being easily calculated as

$$x_c = \frac{n_x}{2}, \;\; y_c = \frac{n_y}{2}. \tag{46}$$

where n_x, n_y are the image width and height, with respect to the top-left corner. Identifying the center of gravity (CoG) of the identified ball object is the next action. The MATLAB

function *logical* first converts numeric values into logical values for the final 2D image. The calculation of the center of gravity of the target object was performed by the *regionprops* function which measures the properties of a specific region of an image. The function returns the measurements for the property set of each connected component (object) in the binary image. The function can be used for both contiguous or adjacent regions and for discontinuous regions. Regarding these types, we can say about the adjacent regions that they are also called objects, connected components or blobs. The object's CoG specific coordinates (o_x,o_y), locates its position within the image. The system's controlled output is defined as $y = y_c - o_y$. In sample-based measurement in discrete-time, it will denote y_k.

To complete the system overview in the proposed design, we denote the PC command sent from the MATLAB environment to the ADCM as the duty cycle factor $u_k \in [-1; 1]$. Regardless of the DZ size, this u_k is sent to the ADCM by adding an offset value for each direction of rotation corresponding to the DZ threshold values. The position reference input r_k sets the desired distance of the object center to the image center. In most applications, $r_k = 0$ means that the assembly (the camera in particular) should rotate in order to keep the tracked object in the center. Normally, both y_k, r_k are measured in pixels, leading to a magnitude of hundreds of pixels, depending on the image size. A normalized value of y_k is being used by dividing its pixel value with 1000. Therefore, normalized reference inputs r_k are used as well. The system is capable of consistently processing the frames in native resolution and all the subsequent controller calculations, for a sample period of $T_s = 0.125\ s$. This sample period was established after many statistical evaluations, by timing all the image transfer, visual processing tasks and control law evaluations [31]. Timer objects were used for software implementation convenience, to properly run the recurrent control tasks.

3.2. The Visual Servo Tracking in Model Reference Control Setting

The controlled system is formally described by the discrete-time I/O equation (with k the sample index)

$$y_k = f\Big(y_{k-1}, \ldots, y_{k-ny}, u_{k-1}, \ldots, u_{k-nu}\Big), \tag{47}$$

with system control input $u_k \in \Omega_u \subset \Re$ inside the known domain Ω_u, system output $y_k \in \Omega_Y \subset \Re$ in known domain Ω_Y, unknown integer orders $ny, nu \in \mathbb{Z}_+^*$, and unknown dynamic nonlinear continuous differentiable map $f : \Omega_Y \times \ldots \times \Omega_Y \times \Omega_u \times \ldots \times \Omega_u \to \Omega_Y$. System (47) is assumed to be observable and controllable; these are common data-driven assumptions extracted from working interaction with the system, from technical specifications or from literature survey. Lemma 1 in [36] allows for a virtual state space construct equivalent model of (47) defined as

$$x_{k+1} = \boldsymbol{F}(x_k, u_k), y_k = [1, 0, \ldots, 0]x_k = x_{k,1}, \tag{48}$$

where $\boldsymbol{x}_k = [\boldsymbol{Y}_{\overline{k,k-\tau}}^T, \boldsymbol{U}_{\overline{k-1,k-\tau}}^T]^T \triangleq [x_{k,1}, x_{k,2}, \ldots x_{k,2\tau+1}]^T \in \Omega_X \subset \Re^{2\tau+1}$, more specifically with $\boldsymbol{Y}_{\overline{k,k-\tau}} = [y_k \ \ldots \ y_{k-\tau}]^T = [x_{k,1} \ \ldots \ x_{k,\tau+1}]^T$, $\boldsymbol{U}_{\overline{k-1,k-\tau}} = [u_{k-1} \ \ldots \ u_{k-\tau}]^T = [x_{k,\tau+2} \ \ldots \ x_{k,2\tau+1}]^T$, where $\tau \in \mathbb{Z}_+^*$ has the role of a nonlinear observability index similar to the one in linear systems theory. Here, τ is defined as the minimal value for which the true state of (48) is recoverable from I/O data samples u_k, y_k. When the order of (48) is unknown, so is τ. In practice, one starts with an initial value and searches for the one which achieves the best control when using the virtual state $\boldsymbol{x}_k$ for feedback. Some remarkable features are synthesized in the following observations.

Observation 2. Systems (47) and (48) have the same input and output, hence from I/O perspective their control is the same. The means to achieve control may be different, however the intention is to attempt state feedback control on the virtual system (48) and feed it as input to the true system (47). System (48) is fully state-observable, with measurable states and with partially known dynamics $\boldsymbol{F}(.) : \Omega_X \times \Omega_u \to \Omega_X$ where domain Ω_X is completely known since it is built on known domains Ω_u, Ω_Y.

Observation 3. Time delays in (47) can also lead to a fully observable description (48) by proper definition of additional states. If present, the (assumed constant) time delays should be known and easily measured from the historical I/O response data.

The visual tracking problem in model reference control intends to find a control law which makes the output y_k in (47) match (or track) the output of a linear output reference model (LORM) best described in its discrete-time state space form as

$$\begin{cases} \boldsymbol{x}_{k+1}^m = \boldsymbol{G}\boldsymbol{x}_k^m + \boldsymbol{H}r_k, \\ y_k^m = \boldsymbol{L}\boldsymbol{x}_k^m, \end{cases} \tag{49}$$

where $\boldsymbol{x}_k^m = \left[x_{k,1}^m, \ldots, x_{k,n_x}^m\right] \in \Omega_{X^m} \subset \Re^{n_x}$ is the LORM state, $r_k \in \Omega_R \subset \Re$ is the reference input in known domain Ω_R and $y_k^m \in \Omega_{Y^m} \subset \Re$ is the LORM output within a known domain. Note that (49) is a state space realization of a more compact linear pulse transfer function (PTF) $M(Q)$ which relates $y_k^m = M(Q)r_k$, among all infinite realizations' triplet $(\boldsymbol{G}, \boldsymbol{H}, \boldsymbol{L})$. Here, Q is the unit time step advance operator ($Qy_k^m = y_{k+1}^m$). Selection of $M(Q)$ is not arbitrary. Under classical model reference control guidelines, it should account for true system dynamics (47) and its bandwidth, and it should include the true system's time delay and nonminimum-phase dynamic if it is the case.

The ORM control problem is formally expressed as in

$$\begin{aligned} \boldsymbol{u}_k^* = arg\ \min_{\boldsymbol{u}_k} V_{LRMO}^{\infty}(\boldsymbol{u}_k), \qquad & V_{LORM}^{\infty}(\boldsymbol{u}_k) = \sum_{k=0}^{\infty} \left[y_k(\boldsymbol{u}_k) - y_k^m\right]^2, \\ & s.t. (48), (49), \end{aligned} \tag{50}$$

where the explicit dependence of y_k on $\boldsymbol{u}_k$ has been captured. The intention is to find an explicit closed-loop control law $\boldsymbol{u}_k = \boldsymbol{C}(\boldsymbol{s}_k, r_k)$ with $\boldsymbol{s}_k$ some regressor state dependent on the control design approach. For example, if $\boldsymbol{s}_k = \boldsymbol{x}_k$, the control renders $\boldsymbol{u}_k = C(y_k \ldots y_{k-\tau}, \boldsymbol{u}_{k-1} \ldots \boldsymbol{u}_{k-\tau}, r_k)$ which is a recurrence equation accommodating many control architectures (PI/PID, etc.). However, other selections of $\boldsymbol{s}_k$ are possible, as illustrated in the following sections. Essentially, when driven by the same r_k, we should ideally get $y_k = y_k^m$, i.e., the system's output (also the closed-loop system (CLS) output) equals the ORM output. The class of ORM tracking control problems belong to the "dense rewards" settings, as opposed to the "sparse rewards" case which is found in many AI-related environments, whereas in classical control, the problem is known as a model reference control.

Differently from the model reference adaptive control (MRAC) but nevertheless sharing the same overall goal of output reference tracking, the control solution to (50) should be a stationary, non-adaptive controller. Formulation of the penalty function may impact the resulted control performance significantly. For example, non-quadratic Lyapunov functions serving the role of penalty functions in adaptive control have been proposed in MRAC [37,38]. Based on *Observation 1*, herein we employ quadratic penalties since we preserve interpretation parallelism with the linear quadratic controller case. A quadratic penalty combined with a linear system renders an infinite-horizon cost that is quadratic convex with respect to the state, which facilitates the existence and uniqueness of a global minimum, but also its analytic derivation. Such desirable properties are expected also with smooth nonlinear systems, to some extent at least.

3.3. Closed-Loop Input-Output Data Collection

The simplest standardized controller, namely the proportional (P)-type one will be used to stabilize the positioning process and to further collect I/O data for finally learning more advanced ORM tracking control. Its transfer function is $H_R(\boldsymbol{s}) = k_p$, where k_p is the transfer coefficient. As shown in [31], this type of controller is recommended in the case of simple driven processes with a single large time constant, with or without an integral

component. In this case, the servo positioning has an integral component. The P controller's law in discrete-time (with transfer function $H_R(Q) = k_p$)

$$\boldsymbol{u}_k = k_p(\rho_k - y_k), \tag{51}$$

is used with $k_p = 0.19$ established following experimental tests to give satisfactory tracking performance. Here, ρ_k is the reference input to the control system. The tracking scenario implies a sequence of successive piecewise constant reference input values ρ_k (each value is kept constant for 10 s) with the amplitude uniformly randomly distributed in $[-0.19; 0.21]$. Additive noise on $\boldsymbol{u}_k$ is applied for enhancing exploration (also understood as persistent excitation) with the following settings: the command $\boldsymbol{u}_k$ receives an additive noise with uniform random amplitude in $[-0.1; 0.1]$, every three samples.

The ORM continuous-time transfer function is selected as

$$m_{ref} = \frac{\Omega_n^2}{\boldsymbol{s}^2 + 2\xi\Omega_n\boldsymbol{s} + \Omega_n^2}, \tag{52}$$

where $\Omega_n = 1$ sets the response speed and $\xi = 0.7$ sets the magnitude overshoot. This transfer function was discretized by using the zero-order hold method with the sampling period $T_s = 0.125\boldsymbol{s}$, to render:

$$M(Q) = \frac{0.007367Q + 0.006949}{Q^2 - 1.825Q + 0.8395}. \tag{53}$$

For $M(Q)$, a state-space controllable canonical form transform is

$$\begin{cases} \boldsymbol{x}_{k+1}^m = \begin{bmatrix} 1.8251 & -0.8395 \\ 1 & 0 \end{bmatrix} \boldsymbol{x}_k^m + \begin{bmatrix} 1 \\ 0 \end{bmatrix} r_k, \\ y_k^m = \begin{bmatrix} 0.0074 & 0.0069 \end{bmatrix} \boldsymbol{x}_k^m, \end{cases} \tag{54}$$

which will be used to generate the input-state-output data for the ORM. Meanwhile, a different reference input r_k which excites the ORM was also selected as a sequence of successive piecewise constant values where each value is kept constant for 15 s and the amplitude is uniformly randomly distributed in $[-0.19; 0.21]$. The switching periods of the reference inputs ρ_k, r_k was not a negligible problem, their values trade-off exploration and lengthy collection time [31]. The I/O data samples contained within the trajectory $\{\boldsymbol{u}_k,\ y_k,\ r_k, \boldsymbol{x}_k^m\}$ will be subjected to various control design approaches. A number of 2400 samples for 300 s of collection were obtained, as in Figure 2.

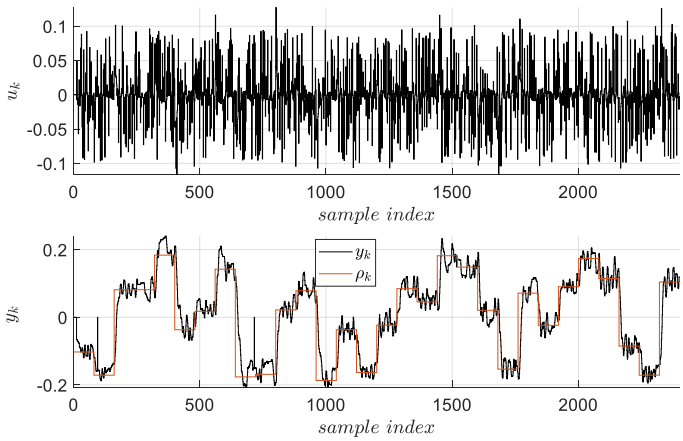

Figure 2. I/O data collected from the closed-loop visual servo system.

3.4. The Visual Servo System Identification

In order to apply a model-based control design dedicated to ORM tracking, a pulse transfer function of the visual servo system must be identified. Since the system has an integral component, the identification is indirectly performed, by firstly identifying the closed-loop pulse transfer function $\rho_k \rightarrow y_k$, based on closed-loop data from Figure 2 [31]. A set of validation data for the same dynamics is generated, as in Figure 3.

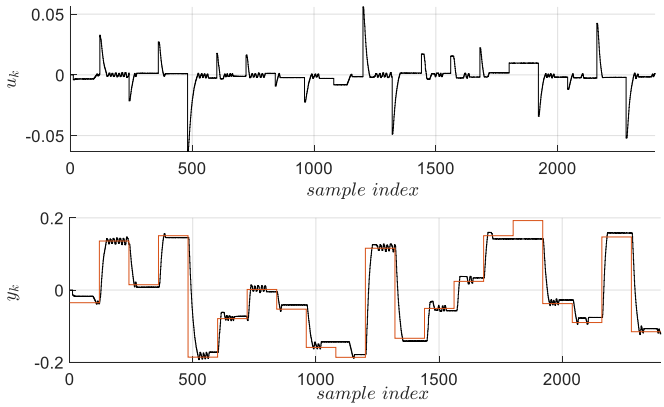

Figure 3. Data used for validating the identified closed-loop visual servo system model (ρ_k-red, y_k-black).

A second-order output-error (OE) recurrent model for the closed-loop visual servo system transfer function is selected, being identified as (and whose step response is shown in Figure 4) [31]

$$M_{CL}(Q) = \frac{B(Q)}{F(Q)} = \frac{-0.08293Q^{-1} + 0.1134Q^{-2}}{1 - 1.73Q^{-1} + 0.7617Q^{-2}}. \tag{55}$$

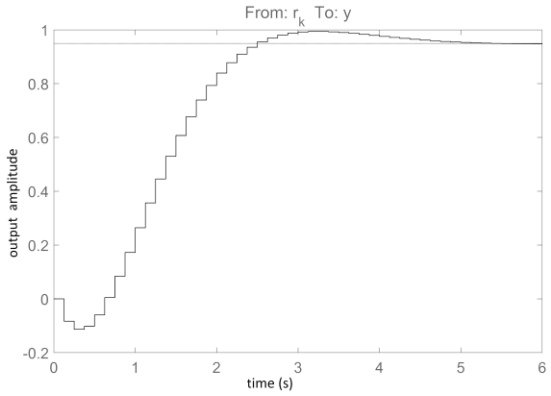

Figure 4. Step response of the OE identified model [31].

The closed-loop transfer function allows for identifying the I/O visual servo system model as [31]

$$H_{PC}(Q) = \frac{M_{CL}(Q)}{(1 - M_{CL}(Q))\, H_R(Q)} = \frac{-0.4365Q^{-1} + 0.5966Q^{-2}}{1 - 1.647Q^{-1} + 0.6483Q^{-2}}, \tag{56}$$

whose step response is shown in the Figure 5. Note that the identified $M_{CL}(Q)$ has a non-minimum-phase character which is not observed in the true system, this non-minimum-phase is a result of the imperfect DZ compensation. Additionally, the model $H_{PC}(Q)$ appears as a first-order lag with a large time constant, although its true dynamics must contain an integrator. However, the incipient step response phase (first 20 s) overlaps with

that of an integral-type system. Finally, the indirectly identified $H_{PC}(Q)$ can be used for model-based control design.

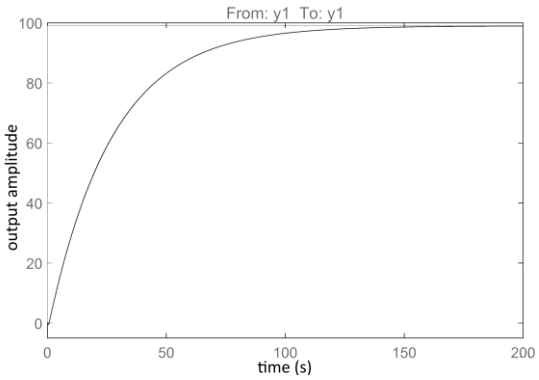

Figure 5. The step response of $H_{PC}(Q)$ [31].

3.5. The 2DOF Model-Based Control Design

The 2DOF (also known as RST) control design procedure is next described, for the case with the same imposed dynamics in terms of setpoint tracking and disturbance rejection. Assume that $H_{PC}(Q)$ can be written in the form

$$H_{PC}(Q) = \frac{K(Q+\beta)}{Q^2+\alpha_1 Q+\alpha_0} Q^{-d} = \frac{B(Q)}{A(Q)} Q^{-d}, \tag{57}$$

which is validated for the identified model since $\partial(B(Q)) = 1, \partial(A(Q)) = 2$, $d = 0$ (the operator $\partial(.)$ measures the polynomial degree) in our case. With the ORM transfer function (53), in the case of the 2DOF controller without stable zero compensation ($H_{PC}(Q)$ has a non-minimum-phase zero) and with integral component in the controller to ensure zero steady-state tracking error and steady-state disturbance rejection simultaneously, we first define the polynomials as factors for $B(Q) = B^C(Q)B^{NC}(Q)$:

$$B^C(Q) = 1, B^{NC}(Q) = K(Q+\beta). \tag{58}$$

The denominator $A_m(Q) = Q^2 + p_1 Q + p_0$ from the imposed $M(Q)$ will actually set the step-response character of the closed-loop system. An artificial, maximal degree polynomial $B_m(Q)$ is constructed as $B_m(Q) = \frac{K(Q+\beta)(1+p_1+p_0)}{K(1+\beta)} = \frac{(Q+\beta)(1+p_1+p_0)}{1+\beta}$ such that it will simultaneously fulfil a set of conditions set out in [39], namely $\partial(B_m(Q)) \leq \partial(B(Q)) + d$ and $B_m(1) = A_m(1)$. After factoring $B_m(Q) = B^{NC}(Q)\overline{B}_m(Q)$, the polynomial $\overline{B}_m(Q)$ results $\overline{B}_m(Q) = (1 + p_1 + p_0)/[K(1+\beta)]$.

The polynomials $R(Q)$, $s(Q)$ and $T(Q)$ are imperative in the 2DOF controller design. Firstly, it must be ensured that $\partial R(Q) = \partial s(Q) = \partial T(Q) = n$ [39] where $\partial A(Q) = \partial A_m(Q) = n = 2$ in our case. Then, $R(Q) = B^+(Q)\,\overline{R}(Q)$ must be monic (leading coefficient is 1) and since $R(Q)$ must have an integral component, based on $B^C(Q) = 1$, we must further explicit $\overline{R}(Q) = (Q-1)\,\overline{\overline{R}}(Q)$. Due to the degree condition $n = 2$, we must write $\overline{\overline{R}}(Q) = Q + r_0$. Then, $s(Q) = s_2Q^2 + s_1Q + s_0$, fulfils the degree condition, while the observer polynomial $A_o(Q) = Q^2$ is selected to ensure minimal degree fulfilment of the polynomial equation [39]

$$A_o(Q)A_m(Q) = A(Q)(Q-1)\overline{\overline{R}}(Q) + B^{NC}(Q)s(Q), \tag{59}$$

equivalent to

$$\begin{gathered} Q^2(Q^2+p_1Q+p_0) = (Q^2+\alpha_1Q+\alpha_0)(Q-1)(Q+r_0) + K(Q+\beta)(s_2Q^2+s_1Q+s_0) \equiv \\ Q^4+p_1Q^3+p_0Q^2 = Q^4+(Ks_2+r_0+\alpha_1-1)Q^3+[K(s_0+\beta s_1)+\alpha_0(r_0-1)-\alpha_1 r_0]Q+K\beta s_0-r_0\alpha_0. \end{gathered} \tag{60}$$

The above polynomial equation was solved as a compatible determined system of linear equations in the unknown coefficients of $\overline{R}(Q), \boldsymbol{s}(Q)$:

$$\left\{\begin{array}{c} K\boldsymbol{s}_2 + r_0 + \alpha_1 - 1 = p_1, \\ K(\boldsymbol{s}_1 + \beta\boldsymbol{s}_2) - r_0 + \alpha_0 + \alpha_1(r_0 - 1) = p_0, \\ K(\boldsymbol{s}_0 + \beta\boldsymbol{s}_1) + \alpha_0(r_0 - 1) - \alpha_1 r_0 = 0, \\ K\beta\boldsymbol{s}_0 - r_0\alpha_0 = 0 \end{array}\right. \equiv \left(\begin{array}{cccc} K & 0 & 0 & 1 \\ \beta K & K & 0 & \alpha_1 - 1 \\ 0 & \beta K & K & \alpha_0 - \alpha_1 \\ 0 & 0 & \beta K & -\alpha_0 \end{array}\right)\left(\begin{array}{c} \boldsymbol{s}_2 \\ \boldsymbol{s}_1 \\ \boldsymbol{s}_0 \\ r_0 \end{array}\right) = \left(\begin{array}{c} p_1 - \alpha_{1+1} \\ p_0 + \alpha_1 - \alpha_0 \\ \alpha_0 \\ 0 \end{array}\right). \tag{61}$$

The solution to (61) will produce $R(Q)$ of the form $R(Q) = Q^2 + r_1 Q + r_0$. Finally, $T(Q) = \overline{B}_m(Q)A_o(Q) = t_2 Q^2$ renders the transfer function description of the 2DOF controller as [31]

$$\boldsymbol{u}_k = \frac{T(Q)}{R(Q)}\rho_k - \frac{\boldsymbol{s}(Q)}{R(Q)}y_k = \frac{t_2 Q^2}{Q^2 + r_1 Q + r_0}\rho_k - \frac{\boldsymbol{s}_2 Q^2 + \boldsymbol{s}_1 Q + \boldsymbol{s}_0}{Q^2 + r_1 Q + r_0}y_k, \tag{62}$$

which in its recurrent filter form writes as

$$\boldsymbol{u}_k = -r_1\boldsymbol{u}_{k-1} - r_0\boldsymbol{u}_{k-2} + t_2\rho_k - \boldsymbol{s}_2 y_k - \boldsymbol{s}_1 y_{k-1} - \boldsymbol{s}_0 y_{k-2}. \tag{63}$$

Using this scheme, we validated the 2DOF controller with the identified system model $H_{PC}(Q)$ in closed-loop with unit reference input step response, in Figure 6.

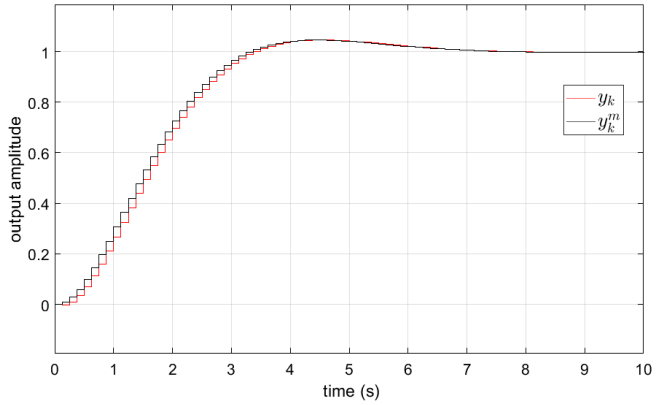

Figure 6. Unit reference input step response of the closed-loop with 2DOF controller, vs. the ORM unit step response.

The experimental validation of the designed 2DOF controller used with the true visual servo system is depicted in Figure 7 below, on a test scenario.

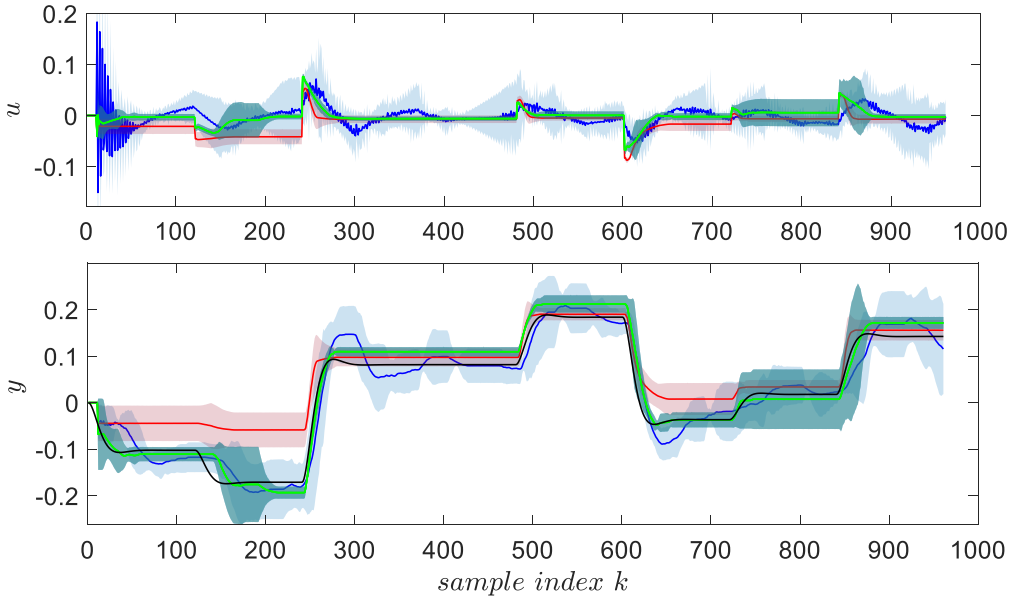

Figure 7. The ORM tracking control experiment with 2DOF (blue), VSFRT (red) and MFVI-RL (green) averaged control performance, with 95% confidence regions. y_k^m is in black line in bottom subplot.

3.6. The MFVI-RL Control for ORM Tracking

MFVI-RL requires some Markovian assumptions about the controlled system, hence the controlled extended system will be

$$\boldsymbol{s}_{k+1} = \boldsymbol{E}(\boldsymbol{s}_k, u_k) \Leftrightarrow \begin{bmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{x}_{k+1}^m \\ r_{k+1} \end{bmatrix} = \boldsymbol{E}\left(\begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{x}_k^m \\ r_k \end{bmatrix}, u_k \right) = \begin{bmatrix} \boldsymbol{F}(\boldsymbol{x}_k, u_k) \\ \boldsymbol{G}\boldsymbol{x}_k^m + \boldsymbol{H}r_k \\ t(r_k) \end{bmatrix}, \tag{64}$$

where $\boldsymbol{E}(.)$ is some generic nonlinear map which is partially unknown, ($\boldsymbol{F}(.)$ is partially unknown) over the extended state, and $\boldsymbol{s}_k = \left[\boldsymbol{x}_k^T, \left(\boldsymbol{x}_k^m\right)^T, r_k\right]^T$. The domain Ω_S of $\boldsymbol{s}_k$ is known, as the domains Ω_X, Ω_{X^m}, Ω_R are known. In addition, $r_{k+1} = \approx(r_k)$ is the reference input generative model which is user-defined. Many types of practical reference inputs comply with this generative model equation (e.g., piece-wise constant model $r_{k+1} = r_k$).

MFVI-RL is able to learn the ORM tracking control solution for the system (64) without complete knowledge of $\boldsymbol{E}(.)$. The extended states-space system representation (64) can perfectly replace the dynamics (47) in the problem definition (50). Note that y_k, y_k^m in (50) depends on components of $\boldsymbol{s}_k$ (y_k is the first component from the vector $\boldsymbol{x}_k$ while $y_k^m = \boldsymbol{L}\boldsymbol{x}_k^m$). In its off-policy offline version (more popularly known as Q-learning), MFVI-RL starts with a dataset of transition samples (known as tuples or as experiences) of the form $D\boldsymbol{s} = \left\{ \left(\boldsymbol{s}_k^{[i]}, u_k^{[i]}, \boldsymbol{s}_{k+1}^{[i]} \right) \right\}$ for a number of $i = \overline{1, B}$ elements.

MFVI-RL relies on the estimate of a "Q-function" which is an extension of the original cost function $V(\boldsymbol{s}_k) = \sum_{i=k}^{\infty} \gamma^{i-k} \pi(\boldsymbol{s}_i, u_i)$. We notice that $V(\boldsymbol{s}_0) = V_{LORM}^{\infty}$ whenever the discount factor $\gamma = 1$ and the penalty term is defined, like in $\pi(\boldsymbol{s}_i, u_i) = \left(y_i - y_i^m\right)^2$ which explicitly captures the ORM tracking goal. Using the Bellman equation to which we associate the controller $\boldsymbol{u}_k = \boldsymbol{C}(\boldsymbol{s}_k)$, the Q-function is defined as $Q^{\boldsymbol{C}}(\boldsymbol{s}_k, u_k) = \pi(\boldsymbol{s}_k, u_k) + \sum_{i=k+1}^{\infty} \gamma^{i-k-1} \pi(\boldsymbol{s}_i, \boldsymbol{C}(\boldsymbol{s}_i)) = \pi(\boldsymbol{s}_k, u_k) + \gamma V^{\boldsymbol{C}}(\boldsymbol{s}_{k+1}) = \pi(\boldsymbol{s}_k, u_k) + \gamma Q^{\boldsymbol{C}}(\boldsymbol{s}_{k+1}, \ \boldsymbol{C}(\boldsymbol{s}_{k+1}))$.

To cope with infinitely dense state and control input domains, it is common to parameterize both the Q-function and the controller with (deep) neural networks (NNs). In the following, the MFVI-RL steps with a generic NN approximator for the Q-function (which is expressed as $Q(\boldsymbol{s}_k, u_k, \boldsymbol{\theta})$ with $\boldsymbol{\theta}$ being the NN weights) and a *linear* controller defined by $u_k = \boldsymbol{K}^T \boldsymbol{s}_k = K_0 y_k + \ldots + K_\tau y_{k-\tau} + K_{\tau+1} u_{k-1} + \ldots + K_{2\tau+1} u_{k-\tau} + K_{2\tau+2} x_{k,1}^m + \ldots + K_{2\tau+n_x+1} x_{k,n_x}^m + K_{2\tau+n_x+2} r_k$ where $\boldsymbol{K} \in \Re^{2\tau+n_x+2}$, are presented in Algorithm 1.

Algorithm 1. The MFVI-RL Algorithm 1 for ORM tracking

1. Choose a reference model with dynamic $M(Q)$, under the rules and in terms of specifications which are representative for model reference control. Build a state space realization (49) from $M(Q)$.
2. Run data collection experiments on system (47) to obtain a dataset of I/O samples $\{u_k,\ y_k\}$. In the same time-frame or in a simulated process afterwards, generate $\left\{r_k, \boldsymbol{x}_k^m, y_k^m\right\}$ using a prescribed generative reference input model $r_{k+1} = \approx(r_k)$ and the state-space (49). Make sure the domains Ω_Y and Ω_{Y^m} are correlated in amplitude. The collection of both $\{u_k,\ y_k\}$ and $\left\{r_k, \boldsymbol{x}_k^m, y_k^m\right\}$ must be sufficiently exploratory i.e., the variables should cover all combinations of their values inside their respective domains. This is usually achieved for long enough trajectories. However, good coverage of the state-action space in a shorter time is achievable in various ways: straightforward procedures include adding noise and avoiding already visited combinations by recording the tracks.
3. Select the observability index τ. Data-driven construct the virtual trajectory $\{u_k,\ y_k, \boldsymbol{x}_k\}$ which defines the virtual state-space system (49). Couple this trajectory with $\left\{r_k, \boldsymbol{x}_k^m, y_k^m\right\}$ to construct transition samples $\left(\boldsymbol{s}_k^{[i]}, u_k^{[i]}, \boldsymbol{s}_{k+1}^{[i]}\right), i = \overline{1, B}$ from (64).
4. Select an initial parameterization for the NN modeling the Q-function, let this be denoted as $Q\left(\boldsymbol{s}_k, u_k, \theta^0\right)$ where θ^j lumps all the NN tunable weights at iteration j. Select $\boldsymbol{K}^0$ as the initial controller parameter, it need not be stabilizing for MFVI-RL algorithm (e.g., random initial value is feasible).
5. At current iteration j, prepare the Q-function NN input patterns as $in\boldsymbol{s} = \left\{\left[\boldsymbol{s}_k^{[i]^T},\ u_k^{[i]}\right]^T\right\}$ and the target output patterns as $out\boldsymbol{s} = \left\{\pi\left(\boldsymbol{s}_k^{[i]}, u_k^{[i]}\right) + Q\left(\boldsymbol{s}_{k+1}^{[i]},\ \boldsymbol{K}^{j-1^T}\boldsymbol{s}_{k+1}^{[i]^T},\ \theta^{j-1}\right)\right\}$, for all $i = \overline{1, B}$. Train the NN using the I/O patterns, based on architecture-dependent training settings. It is equivalent to solving $\theta^j = arg\ \underset{\theta}{min} \frac{1}{B}\sum_{i=1}^{B}\left(\ Q\left(\boldsymbol{s}_k^{[i]},\ \boldsymbol{u}_k^{[i]},\ \theta\right)\ -\ \pi\left(\boldsymbol{s}_k^{[i]},\ u_k^{[i]}\right)\ -\ Q\left(\boldsymbol{s}_{k+1}^{[i]},\ \boldsymbol{K}^{j-1^T}\boldsymbol{s}_{k+1}^{[i]},\ \theta^{j-1}\right)\right)^2$ which is the well-known MSSE cost.
6. For each $\boldsymbol{s}_k^{[i]}, i = \overline{1, B}$ in the dataset, find $u_k^{[i]^*} = arg\ \underset{u}{min} Q\left(\boldsymbol{s}_k^{[i]}, u, \theta^j\right)$. For a bounded infinite domain e.g., $\Omega_u = [-1; 1]$, this minimization will be approximately done by enumerating a finite set of equidistant values for u e.g., $\{-1, -0.9, \ldots, 0.9, 1\}$. Although apparently exhaustive and not as accurate as a gradient based search scheme, this minimization can be efficiently implemented even for multivariable systems where the dimension of Ω_u is acceptable (up to three). Build the overdetermined system of linear equations

$$\begin{bmatrix} \left(\boldsymbol{s}_k^{[1]}\right)^T \\ \ldots \\ \left(\boldsymbol{s}_k^{[B]}\right)^T \end{bmatrix} \boldsymbol{K}^j = \begin{bmatrix} u_k^{[1]^*} \\ \ldots \\ u_k^{[B]^*} \end{bmatrix} \Leftrightarrow \boldsymbol{\Xi_1 K}^j = \boldsymbol{\Xi_2}, \tag{65}$$

   and solve it by ordinary linear least-squares as $\boldsymbol{K}^j = pinv(\boldsymbol{\Xi_1})\boldsymbol{\Xi_2}$ where $pinv(\boldsymbol{\Xi_1}) = \left(\boldsymbol{\Xi_1}^T\boldsymbol{\Xi_1}\right)^{-1}\boldsymbol{\Xi_1}^T$. Another possible gradient-based search solution for finding $\boldsymbol{K}^j$ is to train the cascaded network $Q\left(\boldsymbol{s}_k^{[i]}, \boldsymbol{K}^T\boldsymbol{s}_k^{[i]}, \theta^j\right)$ w.r.t. $\boldsymbol{K}$, over all inputs $\boldsymbol{s}_k^{[i]}, i = \overline{1, B}$, with the target values set to zero.
7. If the number of maximal MFVI-RL algorithm iterations was reached, or there is no improvement in the control parameter ($\|\boldsymbol{K}^j - \boldsymbol{K}^{j-1}\| < threshold$), exit the algorithm; otherwise increment j and go to step 5.

For the MFVI-RL practical implementation of learning visual servo tracking, the Q-function is a deep neural network of feedforward type, with two hidden layers having 100 and 50 rectified linear unit (ReLU) activation function, respectively. A single neuron in the output with linear activation predicts the Q-value of a state-action input. Training settings include: 90%–10% training-validation data splitting with random shuffling; early stopping after 10 consecutive increases of the MSSE cost on the validation data; and maximum 500 epochs with a scaled conjugated gradient descent backprop algorithm. The entire

training data is used as one batch in each epoch, therefore the "replay buffer" consists of all the transition samples in the experience replay mechanism. This can be considered a deep reinforcement learning approach belonging to the supervised machine learning field.

Starting with the I/O data collection from the closed-loop system under the P-type controller and with the ORM input-state-output data collected offline, the trajectory $\{u_k,\ y_k,\ r_k, \boldsymbol{x}_k^m\}$ has 2400 samples out of which only 2369 obey the piecewise-constant generative reference input model $r_{k+1} = r_k$. A total number of 100 iterations lead to the MFVI-RL controller matrix $\boldsymbol{K} = [-0.6337, 0.3008, -0.1373, -0.0838, -0.0527, -0.0007, 0.0041, 0.3094]^T$ for the controller modeled as $u_k = \boldsymbol{K}^T \boldsymbol{s}_k$, shown performing in Figure 7. We stress that the linear controller structure complies very well with the 2DOF controller structure (and with the subsequent VSFRT controller structure), in an attempt to investigate and check the best achievable framework-dependent control performance.

3.7. The VSFRT Learning Control Design for ORM Tracking

The VSFRT feedback control concept is briefly detailed in this section. Since it relies on state information for feedback, it uses again a state-space model, this time it is the model (48), together with a different rationale for computing offline the virtual reference input. It generalizes over the classical nonlinear virtual reference feedback tuning (VRFT) concept [40–47] in the sense that the control law is not explicitly depending on the output feedback error as in $u_k = \boldsymbol{C}(e_k = r_k - y_k)$ but rather in the form $u_k = \boldsymbol{C}(\boldsymbol{x}_k,\ r_k)$, or, more generally, depending on the lumped regressor $\boldsymbol{s}_k = \left[x_k^T,\ r_k\right]^T$ hence resulting in the law $u_k = \boldsymbol{C}(\boldsymbol{s}_k)$. Starting from a dataset of I/O samples $\{u_k,\ y_k\}$ collected from the system (47), VSFRT computes the virtual trajectory $\{u_k,\ y_k, \boldsymbol{x}_k\}$ of (48) first. The virtual reference is afterwards computed as $\widetilde{r}_k = M^{-1}(Q)y_k$. From this, the virtual lumped regressor builds the dataset of states as $\left\{\boldsymbol{s}_k = \left[x_k^T,\ \widetilde{r}_k\right]^T\right\}, k = \overline{1, B}$. The VSRFT principle solves (50) indirectly by solving a controller identification problem posed as [48]

$$\vartheta^* = arg\ \min_{\vartheta} V_{VR}^N(\vartheta), \quad V_{VR}^N(\vartheta) = \frac{1}{B}\sum_{k=1}^{B}(u_k - \boldsymbol{C}(\boldsymbol{s}_k, \vartheta))^2, \tag{66}$$

which specifies that the controller fed with $\boldsymbol{s}_k$ and parameterized by ϑ should output u_k obtained in the first place in the collection phase. A remarkable observation is that $\boldsymbol{s}_k$ in VSFRT does not contain the ORM state $\boldsymbol{x}_k^m$ as in the MFVI-RL case, since it is viewed as an unnecessary redundancy [24]. Hence, the ORM state-space model is useless, except for its I/O model $M(Q)$. VSFRT does not require the Markovian assumption about the system state-space model, hence it does not require a reference input generative model because the VRFT principle is different about getting r_k. The VSFRT steps are next described in Algorithm 2 for the case of a linear control $u_k = \boldsymbol{K}^T \boldsymbol{s}_k = K_0 y_k + \ldots + K_\tau y_{k-\tau} + K_{\tau+1} u_{k-1} + \ldots + K_{2\tau+1} u_{k-\tau} + K_{2\tau+2} r_k$ where $\boldsymbol{K} \in \Re^{2\tau+2}$ (notice the resemblance with MFVI-RL control, except for the ORM states). In the linear controller case, $\vartheta = \boldsymbol{K}$.

Commonly, nonlinear controllers modeled as NNs have also been employed in many settings [24,48–54]. Most of these deal with tracking applications extensively [55–62]. In the case of mildly nonlinear systems, linear state-feedback controllers, such as the proposed one, have proven functional. In fact, using the input patterns $\{\boldsymbol{s}_k\}$ and the output patterns $\{u_k\}$, one can train any approximator for the controller using appropriate methods by minimizing the cost function (66), which is the popular MSSE cost. Several intercoupled aspects of how can VSFRT serve for transfer learning to the MFVI-RL controller have been thoroughly investigated in [24], where other aspects, such as using principal component analysis and autoencoders for dimensionality reduction of the state, have been tested as well. Depending on the VSFRT NN controller complexity, it can be framed within the deep learning type of applications.

Algorithm 2. The VSFRT Algorithm 2 for ORM tracking

1. Choose a reference model dynamic $M(Q)$ under the rules and in terms of specifications which are representative for model reference control.
2. Run data collection experiments on system (47) to obtain a dataset of I/O samples $\{u_k,\ y_k\}$. Here, u_k should be persistently exciting to make y_k capture all of the system's dynamics. This is a sort of exploration condition, as in MFVI-RL. The excitation condition is usually ensured by (additive) exploratory noise, whether in an open- or in closed-loop.
3. Select the observability index τ. Data-driven build the virtual trajectory $\{u_k,\ y_k, x_k\}$ which defines the virtual state-space system (48).
4. Compute the virtual reference input as $\widetilde{r}_k = M^{-1}(Q)y_k$. Here, a series of aspects are of interest. $M(Q)$ commonly has a low-pass behavior, making $M^{-1}(Q)$ as a high-pass filter. If y_k is noisy, low-pass prefiltering is strongly advised, which could be done, for instance, as a zero-phase filtering. This filtering is not the same as the classical VRFT pre-filter $L(Q)$ as it is not applied on the entire regressor vector $\boldsymbol{s}_k$ but only on y_k. Further, even if $M^{-1}(Q)$ is non-causal, it can be implemented offline.
5. Build the regressor vector $\left\{\boldsymbol{s}_k = \left[\boldsymbol{x}_k^T,\ \widetilde{r}_k\right]^T\right\}, k = \overline{1,B}$.
6. Form the overdetermined system of linear equations

$$\begin{bmatrix} \boldsymbol{s}_1^T \\ \dots \\ \boldsymbol{s}_B^T \end{bmatrix} K = \begin{bmatrix} u_1 \\ \dots \\ u_B \end{bmatrix} \Leftrightarrow \boldsymbol{\Xi_1 K} = \boldsymbol{\Xi_2}, \tag{67}$$

and solve it by ordinary linear least-squares as $\boldsymbol{K} = pinv(\boldsymbol{\Xi_1})\boldsymbol{\Xi_2}$ where $pinv(\boldsymbol{\Xi_1}) = \left(\boldsymbol{\Xi_1}^T\boldsymbol{\Xi_1}\right)^{-1}\boldsymbol{\Xi_1}^T$. This, in fact, solves the controller identification (66) over the controller parameter $\boldsymbol{K}$.

For our practical implementation on the visual servo tracking, the same dataset of I/O data was used for VSFRT control design; namely, there were 2400 samples of trajectory $\{u_k,\ y_k, \boldsymbol{x}_k\}$. There is no need for a specific generative reference input model, hence all samples were used. Before computing the virtual reference $\widetilde{r}_k = M^{-1}(Q)y_k$, the output was prefiltered through the low-pass filter $QM(Q)$. The reason is that there exists some high-frequency content in y_k due to the imperfect DZ compensation in the collection experiment with the P-type controller. Different from the classical VRFT setting, the role of the prefilter here is not to make the controller identification cost function match the ORM tracking cost function, but to merely minimize the DZ's negative influence. Solving (66) with (67) renders $\boldsymbol{K} = [-0.3469, -0.1222,\ 0.1001,\ 0.0488,\ 0.0060,\ 0.3565]^T$ and the resulted VSFRT controller is $u_k = \boldsymbol{K}^T\boldsymbol{s}_k$, performing in Figure 7.

3.7.1. Testing the Controllers and Measuring the Resulted Tracking Control Performance

For a fair evaluation of the ORM tracking control, the index

$$fit_{\boldsymbol{score}}[\%] = 100\left(1 - \sum_{i=1}^{N}\left(y^k - y_k^m\right)^2 / \left(\sum_{i=1}^{N}\left(y^k - \overline{y}_k\right)^2\right)\right), \tag{68}$$

was measured, where the mean value is defined $\overline{y}_k = 1/N\sum_{i=1}^{N} y_k$ for $N = 950$ samples. This fit score metric is more practical and correlates well with the cost V_{LORM}^{∞} in (50). It also represents the percentage expression of the R^2 Pearson coefficient of determination used in statistics. The ORM tracking performance for the visual servo system is shown in Figure 7, in terms of mean response and 95% confidence intervals obtained after five runs with each of the 2DOF, MFVI-RL and VSFRT controllers. The actual values of $fit_{\boldsymbol{score}}$ are rendered in Table 1 below.

Table 1. 2DOF, VSFRT and MFVI-RL tracking performance score when learning uses closed-loop IO data, measured on fit_{score} [%] [31].

Trial	2DOF	VSFRT	MFVI-RL
1	75.4897%	18.8667%	80.8252%
2	62.1113%	25.3567%	81.9418%
3	73.7258%	49.2332%	78.1441%
4	70.1448%	39.4324%	72.6856%
5	69.4888%	54.4153%	73.2465%
Average	70.1921%	37.4609%	**77.3686**%
Std	5.1563%	15.1837%	**4.2540**%

From a testing viewpoint, we do not move the blue ball target manually since this behavior is hardly reproducible. Instead, we change the reference setpoint distance of the image center with respect to the detected blue ball as a sequence of piecewise constant steps. This piecewise constant variation of the reference is then filtered through the linear ORM. This ORM's output should be tracked accordingly with all tested controllers (An illustration of the test scenario with the MFVI-RL controller is to be found in the movie that can be accessed and viewed at the following address https://drive.google.com/file/d/1VI8mbi7GOyaOE4ceIxJyaEPvRVBqTQPP/view?usp=sharing (accessed on 7 November 2021)).

A linear controller parameterization was used in this case study, to verify and discriminate the control capability of MFVI-RL and VSFRT, compared to the model-based 2DOF controller. As seen from the results recorded in Table 1, the MFVI-RL controller achieves the best statistical result, even better than the 2DOF model-based controller, which supports the conclusion that model-free design can in fact surpass the model-based design which relies on approximate identified system models. The inferiority of VSFRT with respect to both the baseline 2DOF controller and MFVI-RL controller, observed in Table 1, indicates that the I/O data exploration issues is the main responsible factor. This conclusion is supported by the fact that VSFRT has been shown to be superior to MVFI-RL when applied to other systems, such as aerodynamic or robotic ones [24]. In fact, VSFRT was able to learn better control with fewer transition samples and with less exploration. This observation confirms that the exploration issue in data-driven control is still an open problem and efforts are needed to define exploration quality indexes. These indexes must be accounted for and balance the overall responses to the big questions of data-driven control, such as what is the tradeoff between the controller complexity and parameterization vs. the control goal vs. the optimal data volume and exploration quality?

On the application level, all three compared controllers are limited in the achievable linear ORM tracking performance. The reason lies with the visual servo system actuation quality from the ADCM: it has a time-varying dead zone whose thresholds in each rotation direction change with longer runs, owing to electromagnetic effects. A perfect compensation is therefore impossible to achieve without an adaptive mechanism. The used ADCM is certainly not suitable for quality positioning systems. Under these adversarial settings, the learned control for the visual servo system ORM tracking is deemed satisfactory and it achieved the desired quality level. With better actuators, the tracking performance is certainly expected to increase. In conclusion, such video servo systems are widely usable in real world applications, such as surveillance cameras, motion tracking, etc.

4. Conclusions

Learning controllers from I/O data based on the new virtual state representation was validated in a data-driven style using MFVI-RL and VSFRT frameworks. This validation is important since, on one hand, it makes a strong connection between modern control concepts and classical control concepts such as ORM tracking, two-degrees-of-freedom control and classical control performance indices. On the other hand, it uses machine learning methods and tools to achieve the control goal, therefore bridging the gap between

control systems and artificial intelligence. The generalizability of these techniques is straightforward extendable to more, nonlinear, multivariable systems. Validation on more systems like the visual servo will confirm the viability of the proposed frameworks and open the road to more systematic investigations. It will also lead to automatic control design tools with powerful generalization capability, belonging to the next generation of intelligent and autonomous control systems.

Author Contributions: Conceptualization, M.-B.R.; methodology, M.-B.R.; software, M.-B.R. and D.-P.C.; validation D.-P.C. and M.-B.R.; formal analysis, T.L. and M.-B.R.; investigation, T.L., D.-P.C. and M.-B.R.; writing—original draft preparation, T.L., D.-P.C. and M.-B.R.; writing—review and editing, T.L., D.-P.C. and M.-B.R.; visualization, D.-P.C. and M.-B.R.; supervision, M.-B.R.; project administration, M.-B.R. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a grant of the Ministry of Research, Innovation and Digitization, CNCS/CCCDI—UEFISCDI, project number PN-III-P1-1.1-TE-2019-1089, within PNCDI III.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Le, T.P.; Vien, N.A.; Chung, T. A Deep Hierarchical Reinforcement Learning Algorithm in Partially Observable Markov Decision Processes. *IEEE Access* **2018**, *6*, 49089–49102. [CrossRef]
2. Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; Zaremba, W. Hindsight experience replay. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 5049–5059.
3. Kong, W.; Zhou, D.; Yang, Z.; Zhao, Y.; Zhang, K. UAV Autonomous Aerial Combat Maneuver Strategy Generation with Observation Error Based on State-Adversarial Deep Deterministic Policy Gradient and Inverse Reinforcement Learning. *Electronics* **2020**, *9*, 1121. [CrossRef]
4. Fujimoto, S.; Van Hoof, H.; Meger, D. Addressing Function Approximation Error in Actor-Critic Methods. In Proceedings of the 35th International Conference on Machine Learning, ICML, Stockholm, Sweden, 10–15 July 2018; Volume 4, pp. 2587–2601.
5. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proceedings of the 35th International Conference on Machine Learning, ICML, Stockholm, Sweden, 10–15 July 2018; Volume 5, pp. 2976–2989.
6. Werbos, P.J. A Menu of Designs for Reinforcement Learning Over Time. In *Neural Networks for Control*; Miller, W.T., Sutton, R.S., Werbos, P.J., Eds.; MIT Press: Cambridge, MA, USA, 1990; pp. 67–95.
7. Lewis, F.L.; Vrabie, D. Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE Circuits Syst. Mag.* **2009**, *9*, 32–50. [CrossRef]
8. Wang, F.Y.; Zhang, H.; Liu, D. Adaptive Dynamic Programming: An Introduction. *IEEE Comput. Intell. Mag.* **2009**, *4*, 39–47. [CrossRef]
9. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
10. Lewis, F.L.; Vamvoudakis, K.G. Reinforcement Learning for Partially Observable Dynamic Processes: Adaptive Dynamic Programming Using Measured Output Data. *IEEE Trans. Syst. Man, Cybern. Part B Cybern.* **2010**, *41*, 14–25. [CrossRef]
11. Wang, Z.; Liu, D. Data-Based Controllability and Observability Analysis of Linear Discrete-Time Systems. *IEEE Trans. Neural Netw.* **2011**, *22*, 2388–2392. [CrossRef]
12. Yu, W.; Yu, H.; Du, J.; Zhang, M.; Liu, J. DeepGTT: A general trajectory tracking deep learning algorithm based on dynamic law learning. *IET Radar Sonar Navig.* **2021**, *15*, 1125–1150. [CrossRef]
13. Wang, W.; Chen, X.; Fu, H.; Wu, M. Data-driven adaptive dynamic programming for partially observable nonzero-sum games via Q-learning method. *Int. J. Syst. Sci.* **2019**, *50*, 1338–1352. [CrossRef]
14. Perrusquia, A.; Yu, W. Neural H_2 Control Using Continuous-Time Reinforcement Learning. *IEEE Trans. Cybern.* **2020**, 1–10. [CrossRef]
15. Sardarmehni, T.; Heydari, A. Sub-optimal switching in anti-lock brake systems using approximate dynamic programming. *IET Control. Theory Appl.* **2019**, *13*, 1413–1424. [CrossRef]
16. Tang, H.; Wang, A.; Xue, F.; Yang, J.; Cao, Y. A Novel Hierarchical Soft Actor-Critic Algorithm for Multi-Logistics Robots Task Allocation. *IEEE Access* **2021**, *9*, 42568–42582. [CrossRef]
17. Liu, Y.; Zhang, H.; Yu, R.; Xing, Z. H∞ Tracking Control of Discrete-Time System with Delays via Data-Based Adaptive Dynamic Programming. *IEEE Trans. Syst. Man, Cybern. Syst.* **2020**, *50*, 4078–4085. [CrossRef]

18. Buşoniu, L.; de Bruin, T.; Tolić, D.; Kober, J.; Palunko, I. Reinforcement learning for control: Performance, stability, and deep approximators. *Annu. Rev. Control.* **2018**, *46*, 8–28. [CrossRef]
19. de Bruin, T.; Kober, J.; Tuyls, K.; Babuska, R. Integrating State Representation Learning into Deep Reinforcement Learning. *IEEE Robot. Autom. Lett.* **2018**, *3*, 1394–1401. [CrossRef]
20. Ni, Z.; He, H.; Zhong, X. Experimental Studies on Data-Driven Heuristic Dynamic Programming for POMDP. In *Frontiers of Intelligent Control and Information Processing*; Liu, D., Alippi, C., Zhao, D., Zhang, H., Eds.; World Scientific: Singapore, 2014; Chapter 3; pp. 83–106. [CrossRef]
21. Ruelens, F.; Claessens, B.J.; Vandael, S.; De Schutter, B.; Babuska, R.; Belmans, R. Residential Demand Response of Thermostatically Controlled Loads Using Batch Reinforcement Learning. *IEEE Trans. Smart Grid* **2017**, *8*, 2149–2159. [CrossRef]
22. Fu, H.; Chen, X.; Wang, W.; Wu, M. MRAC for unknown discrete-time nonlinear systems based on supervised neural dynamic programming. *Neurocomputing* **2020**, *384*, 130–141. [CrossRef]
23. Xue, W.; Lian, B.; Fan, J.; Kolaric, P.; Chai, T.; Lewis, F.L. Inverse Reinforcement Q-Learning Through Expert Imitation for Discrete-Time Systems. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, 1–14. [CrossRef]
24. Radac, M.B.; Borlea, A.I. Virtual State Feedback Reference Tuning and Value Iteration Reinforcement Learning for Unknown Observable Systems Control. *Energies* **2021**, *14*, 1006. [CrossRef]
25. Radac, M.B.; Precup, R.E. Three-level hierarchical model-free learning approach to trajectory tracking control. *Eng. Appl. Artif. Intell.* **2016**, *55*, 103–118. [CrossRef]
26. Wu, B.; Gupta, J.K.; Kochenderfer, M. Model primitives for hierarchical lifelong reinforcement learning. *Auton. Agents Multi-Agent Syst.* **2020**, *34*, 28. [CrossRef]
27. Li, J.; Li, Z.; Li, X.; Feng, Y.; Hu, Y.; Xu, B. Skill Learning Strategy Based on Dynamic Motion Primitives for Human-Robot Cooperative Manipulation. *IEEE Trans. Cogn. Dev. Syst.* **2021**, *13*, 105–117. [CrossRef]
28. Kim, Y.-L.; Ahn, K.-H.; Song, J.-B. Reinforcement learning based on movement primitives for contact tasks. *Robot. Comput. Integr. Manuf.* **2020**, *62*, 101863. [CrossRef]
29. Camci, E.; Kayacan, E. Learning motion primitives for planning swift maneuvers of quadrotor. *Auton. Robot.* **2019**, *43*, 1733–1745. [CrossRef]
30. Yang, C.; Chen, C.; He, W.; Cui, R.; Li, Z. Robot Learning System Based on Adaptive Neural Control and Dynamic Movement Primitives. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 777–787. [CrossRef] [PubMed]
31. Chirla, D.P. Video Tracking Control System for a Moving Target. Bachelor's Thesis, Politehnica University of Timisoara, Timisoara, Romania, 2021.
32. Jiang, M.; Li, R.; Liu, Q.; Shi, Y.; Tlelo-Cuautle, E. High speed long-term visual object tracking algorithm for real robot systems. *Neurocomputing* **2021**, *434*, 268–284. [CrossRef]
33. Tsai, C.-Y.; Song, K.-T. Visual Tracking Control of a Wheeled Mobile Robot with System Model and Velocity Quantization Robustness. *IEEE Trans. Control. Syst. Technol.* **2009**, *17*, 520–527. [CrossRef]
34. Tsai, C.-Y.; Song, K.-T.; Dutoit, X.; Van Brussel, H.; Nuttin, M. Robust visual tracking control system of a mobile robot based on a dual-Jacobian visual interaction model. *Robot. Auton. Syst.* **2009**, *57*, 652–664. [CrossRef]
35. Hua, C.; Wang, Y.; Guan, X. Visual tracking control for an uncalibrated robot system with unknown camera parameters. *Robot. Comput. Integr. Manuf.* **2014**, *30*, 19–24. [CrossRef]
36. Radac, M.-B.; Lala, T. Robust Control of Unknown Observable Nonlinear Systems Solved as a Zero-Sum Game. *IEEE Access* **2020**, *8*, 214153–214165. [CrossRef]
37. Tao, G. Model reference adaptive control with $L^{1+\alpha}$ tracking. *Int. J. Control.* **1996**, *64*, 859–870. [CrossRef]
38. Hosseinzadeh, M.; Yazdanpanah, M.J. Performance enhanced model reference adaptive control through switching non-quadratic Lyapunov functions. *Syst. Control. Lett.* **2015**, *76*, 47–55. [CrossRef]
39. Preitl, S.; Precup, R.E.; Preitl, Z. *Structuri și Algoritmi Pentru Conducerea Automată a Proceselor: Volumul 2*; Editura Orizonturi Universitare: Timisoara, Romania, 2009.
40. Campi, M.C.; Lecchini, A.; Savaresi, S.M. Virtual reference feedback tuning: A direct method for the design of feedback controllers. *Automatica* **2002**, *38*, 1337–1346. [CrossRef]
41. Formentin, S.; Savaresi, S.M.; Del Re, L. Non-iterative direct data-driven controller tuning for multivariable systems: Theory and application. *IET Control. Theory Appl.* **2012**, *6*, 1250–1257. [CrossRef]
42. Campestrini, L.; Eckhard, D.; Gevers, M.; Bazanella, A.S. Virtual Reference Feedback Tuning for non-minimum phase plants. *Automatica* **2011**, *47*, 1778–1784. [CrossRef]
43. Eckhard, D.; Campestrini, L.; Boeira, E.C. Virtual disturbance feedback tuning. *IFAC J. Syst. Control.* **2018**, *3*, 23–29. [CrossRef]
44. Yan, P.; Liu, D.; Wang, D.; Ma, H. Data-driven controller design for general MIMO nonlinear systems via virtual reference feedback tuning and neural networks. *Neurocomputing* **2016**, *171*, 815–825. [CrossRef]
45. Campi, M.C.; Savaresi, S.M. Direct Nonlinear Control Design: The Virtual Reference Feedback Tuning (VRFT) Approach. *IEEE Trans. Autom. Control.* **2006**, *51*, 14–27. [CrossRef]
46. Jianhong, W.; Ramirez-Mendoza, R.A. Finite sample properties of virtual reference feedback tuning with two degrees of freedom controllers. *ISA Trans.* **2020**, *99*, 37–49. [CrossRef]
47. Chiluka, S.K.; Ambati, S.R.; Seepana, M.M.; Gara, U.B.B. A novel robust Virtual Reference Feedback Tuning approach for minimum and non-minimum phase systems. *ISA Trans.* **2021**, *115*, 163–191. [CrossRef]

48. Radac, M.-B.; Lala, T. Hierarchical Cognitive Control for Unknown Dynamic Systems Tracking. *Mathematics* **2021**, *9*, 2752. [CrossRef]
49. Vodovozov, V.; Aksjonov, A.; Petlenkov, E.; Raud, Z. Neural Network-Based Model Reference Control of Braking Electric Vehicles. *Energies* **2021**, *14*, 2373. [CrossRef]
50. Alimohamadi, H.; Alagoz, B.B.; Tepljakov, A.; Vassiljeva, K.; Petlenkov, E. A NARX Model Reference Adaptive Control Scheme: Improved Disturbance Rejection Fractional-Order PID Control of an Experimental Magnetic Levitation System. *Algorithms* **2020**, *13*, 201. [CrossRef]
51. Cao, S.; Sun, L.; Jiang, J.; Zuo, Z. Reinforcement Learning-Based Fixed-Time Trajectory Tracking Control for Uncertain Robotic Manipulators with Input Saturation. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, 1–12. [CrossRef] [PubMed]
52. Dong, F.; Jin, D.; Zhao, X.; Han, J.; Lu, W. A non-cooperative game approach to the robust control design for a class of fuzzy dynamical systems. *ISA Trans.* **2021**. [CrossRef] [PubMed]
53. Chai, Y.; Luo, J.; Ma, W. Data-driven game-based control of microsatellites for attitude takeover of target spacecraft with disturbance. *ISA Trans.* **2021**, *119*, 93–105. [CrossRef]
54. Dogru, O.; Velswamy, K.; Huang, B. Actor–Critic Reinforcement Learning and Application in Developing Computer-Vision-Based Interface Tracking. *Engineering* **2021**, *7*, 1248–1261. [CrossRef]
55. Li, H.; Wang, Y.; Pang, M. Disturbance compensation based model-free adaptive tracking control for nonlinear systems with unknown disturbance. *Asian J. Control.* **2021**, *23*, 708–717. [CrossRef]
56. Lee, W.; Jeoung, H.; Park, D.; Kim, T.; Lee, H.; Kim, N. A Real-Time Intelligent Energy Management Strategy for Hybrid Electric Vehicles Using Reinforcement Learning. *IEEE Access* **2021**, *9*, 72759–72768. [CrossRef]
57. Moreno-Valenzuela, J.; Montoya-Cháirez, J.; Santibáñez, V. Robust trajectory tracking control of an underactuated control moment gyroscope via neural network–based feedback linearization. *Neurocomputing* **2020**, *403*, 314–324. [CrossRef]
58. Fei, Y.; Shi, P.; Lim, C.-C. Robust and Collision-Free Formation Control of Multiagent Systems with Limited Information. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, 1–10. [CrossRef] [PubMed]
59. Meng, X.; Yu, H.; Xu, T.; Wu, H. Disturbance Observer and L_2-Gain-Based State Error Feedback Linearization Control for the Quadruple-Tank Liquid-Level System. *Energies* **2020**, *13*, 5500. [CrossRef]
60. Mohammadzadeh, A.; Vafaie, R.H. A deep learned fuzzy control for inertial sensing: Micro electro mechanical systems. *Appl. Soft Comput.* **2021**, *109*, 107597. [CrossRef]
61. Zhao, H.; Peng, L.; Yu, H. Model-free adaptive consensus tracking control for unknown nonlinear multi-agent systems with sensor saturation. *Int. J. Robust Nonlinear Control.* **2021**, *31*, 6473–6491. [CrossRef]
62. Zhao, J.; Na, J.; Gao, G. Robust tracking control of uncertain nonlinear systems with adaptive dynamic programming. *Neurocomputing* **2021**, *471*, 21–30. [CrossRef]