Article

Discerning discretization for UUV DC Motor control

Jovan Menezes [1], Timothy Sands [2,*]

[1] Department of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853 USA
[2] Department of Mechanical Engineering (CVN), Columbia University, New York, NY 10027 USA
* Correspondence: dr.timsands@caa.columbia.edu

Abstract: Discretization is the process of converting a continuous function or model or equation into discrete steps. In this work, adaptive and learning methods are implemented to control DC motors that are used for actuating control surfaces of unmanned underwater vehicles. Adaptive control is a method in which the controller is designed to adapt the system with parameters which vary or are uncertain. Parameter estimation is the process of computing the parameters of a system using a model & measured data. Adaptive methods have been used in conjunction with different parameter estimation techniques. Deterministic artificial intelligence, a learning-based approach that uses the process dynamics defined by physics, is also applied to control the output of the DC motor to track a specified trajectory. This work goes further to evaluate the performance of the adaptive & learning techniques based on different discretization methods. The results are evaluated based on the absolute error mean between the output & the reference trajectory and the standard deviation of the error. The first order-hold method of discretization and surprisingly large sample time of seven tenths of a second yields over sixty percent improvement over the results presented in the prequel literature.

Keywords: Discretization; DC motors; deterministic artificial intelligence; adaptive control; learning control; proportional derivative; estimation; least squares; modeling

1. Introduction

Direct Current (DC) motors are a class of rotating electrical motors that convert DC electrical energy into mechanical energy, and such motors have ubiquitous applications including unmanned underwater vehicles. The operation of DC motors is amidst a revolutionary change with the adoption of sophisticated microcontrollers and control strategies. Control of DC motors is a well-studied topic in the literature, including using neural networks [1,2] including neural network-based auto-tuning of classical proportional, integral, derivative controllers [3], as well as recursive least squares [4]. Estimators and estimation techniques are deployed side-by-side with control strategies to determine the parameters and even the state of the system using a model.

Figure 1. Control surfaces and propellor on unmanned underwater vehicles [5] utilizing typically available DC motors [6,7] like those depicted in figure 2.

(a) (b) (c)

Figure 2. (**a**) Maxon high-torque DC brushless motors for unmanned underwater vehicles [6]. (**b**) Underwater thruster propeller motor [7]. (**c**) ECI-40 Maxon underwater drive motor and gear head.

1.1. Research lineage from the M.I.T. rule to regression

The method of least squares is one of the most foundational mathematical techniques used in modeling and estimation theory. The objective of the least squares method consists of adjusting the parameters of a model of the system to best fit a set of data. The least squares method forms the basis of estimation for several adaptive modeling methods presented in literature [8–13] along two lines of thought represented by Slotine [8–9] as modified by Fossen [10] and Åström [11–13] respectively. Each method involves formulation of canonical regression forms, while the Slotine/Fossen approach seeks to utilize the full regression form, and the Åström approach bifurcates the regression model into components exhibiting disparate characteristics that might or might not need to be adapted. The Slotine/Fossen approach was augmented [14] with physics-based methods of Lorenz [15] to formulate the burgeoning method referred to as deterministic artificial intelligence [16] which was proposed for applications to DC motors [17] and validated by Shah [18]. Shah's validation highlighted the criticality on motor performance of the discretization method and discretization time interval and recommended study of such. This manuscript is one such study as recommended by Shah.

A main theme of the research lineage is replacement of classical adaption methods (e.g., the co-called "M.I.T. rule") [19] with estimation methods based on least squares. Numerous variations of the least squares approach have been developed that have been used in designing different types of estimators. It is therefore worth getting an essence of these forms that are prevalent to the work presented in this manuscript.

1.2. Least squares variations

Recursive form (an adaptive algorithm that recursively estimates the parameters of a system using a model that is linear in those parameters) [13] and batch form (where all measurements are collected together and processed simultaneously) [9] are variations of the least squares approach that have also been applied as an adaptive method [14].

Another version of the method of least squares is the weighted least squares, also known as weighted linear regression, in which weights are assigned to the observations and these weights are proportional to the reciprocal of the error variance for that observation. Ideally, the weights in the weighted least squares analysis are nonrandom quantities that are proportional to the reciprocal of the variances of the measured data, but it might not always be clear as to how to choose the weights.

In cases where there is an ambiguity on choosing the weights for the weighted least squares approach, the extended least squares method may be adopted. The extended least squares approach provides a potential solution to the weighting problem experienced in the weighted least squares method by avoiding it. The extended least squares method is a maximum likelihood kind of statistical estimation method when the data is normally distributed. With the extended least squares, weights need not be chosen.

A common modeling assumption used in this work is the autoregressive moving average model. The autoregressive moving average method is a model in which the methods of autoregression analysis and moving average are both applied to time-series data. The autoregressive model specifies that the output variable depends linearly on its own previous values and on a stochastic term (an imperfectly predictable term); thus, the model is in the form of a stochastic difference equation. The moving-average model, also known as moving-average process, is a common approach for modeling univariate (an equation, function or polynomial involving only one variable) time series. The moving-average model specifies that the output variable is cross-correlated with a non-identical to itself random-variable. While implementing the autoregressive moving average approach to a model, it is assumed that the time series is stationary and when it fluctuates, it does so uniformly around a particular time.

1.3 Physics-based utilization of governing differential equations

For a system with unknown parameters, several methods of adaptation can be implemented to control the system and achieve a desired response. Often, a desired response is tracking of an input signal by an output signal [20]. Most natural processes do not cause a change in input to be matched by its resulting output, and certain adaptive methods can allow for control and tracking using physics-based methods. The main difference between nonlinear adaptive control [13] and the physics-based method [15] is the utilization of the complete mathematical expression of modeling by physics. The general spirit of both is embodied in the feedforward portion of the recently proposed deterministic artificial intelligence [17], referred to as assertion of self-awareness. Parameter adaption by classical methods (e.g., M.I.T. rule) is one option, while Smeresky and Rizzo recently proposed optimal learning [22] as another option utilizing batch least squares, where current research investigates the efficacies of variations of least squares. Last year, Zhai studied learning implementation by signal-encoded deep learning [23] and offered a direct comparison to deterministic algorithms in [24].

Learning methods like deterministic artificial intelligence involve using the process dynamics (dictated by mathematical models of physics) as self-awareness statements in the form of feedforward controls [17], and learning is driven by evaluation of command-input tracking performance metrics. It is noteworthy to indicate that self-awareness statements in deterministic artificial intelligence only function when supplied an analytic desired trajectory, where error calculation enables both adaption and optimal learning according to recently published results [18]. Latest literature shows that the adaptive approach achieves around 29% lower error than deterministic artificial intelligence in input tracking [18].

1.4. Discretization

The models developed for estimating, adaptation, and learning physical systems begin as continuous functions, since the modeling is strictly taken from the first principles of physics. However, when dealing with controllers and computers to implement the control strategies, it is essential to discretize the continuous system. Different discretization strategies are available to convert continuous systems into discrete systems, particularly in the fields of signal processing, control, and estimation. This manuscript presents a study of the effects of different discretization methods when converting the continuous model of the DC motor and the eventual efficacy applied to DC motor control. Arbitrary selection of different discretization methods and intervals led to the results depicted in figure 3.

The work includes a study of the effects of changing the sample time, a scalar value that represents the sampling period of the resulting discrete-time system. Discussions are offered on efficacy of DC motors' output tracking a desired trajectory (in this work a series of alternating step functions) with iterations of the discretization method and sample time. Finally, novel conclusions and recommendations are offered regarding what discretization method and sample time are best suited (including limiting cases) for the deterministic artificial intelligence method based on the mean and standard deviation of the error of the output from the desired trajectory.

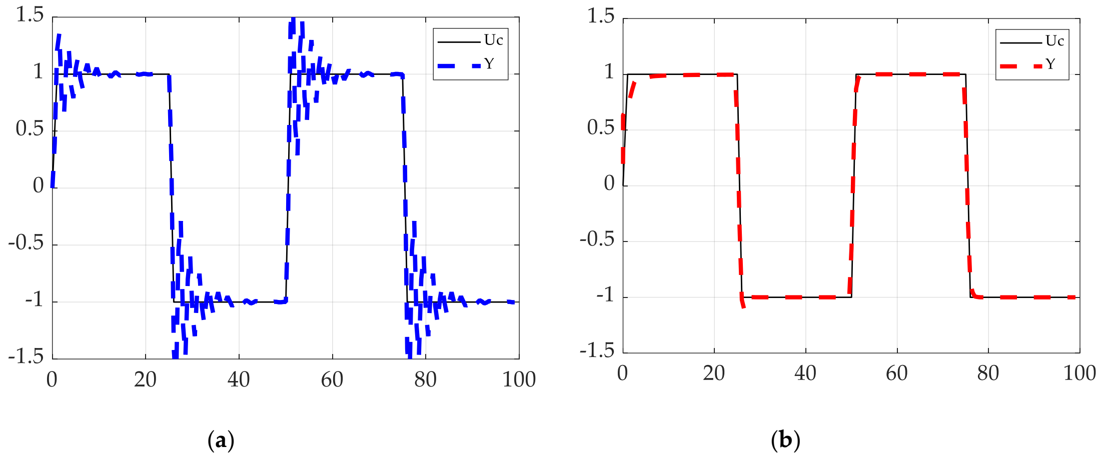


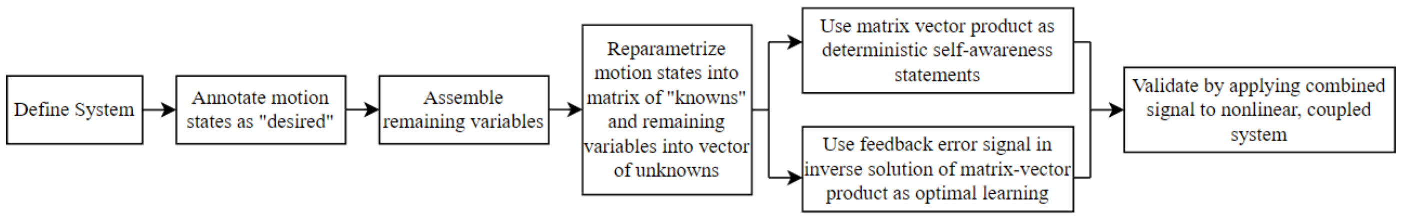
Figure 3. Recently proposed DC motor improvements in the literature, where the difference is the discretization method and sample time. [17] The black solid line describes the command signal. The blue dotted line in (**a**) represents the output signal generated by discrete deterministic artificial intelligence; The red dotted line in (b) describes the output signal generated by model-following method coupled with recursive least squares estimation.

Main conclusion of the prequel study. *The efficacy of the deterministic artificial intelligence approach is limited by sample time values and discretization method. Identification of the limiting discretization values and recommendations for discretization method was recommended.*

Main conclusion of this study. *The deterministic artificial intelligence approach is suitable only for a range of sample time values for each discretization method. To achieve high efficacy, these values and methods must be adhered to.*

2. Materials and Methods

To illustrate the main conclusions of this study, the Materials and Methods begin with overarching principles like methodological process flow (e.g., figure 4) and topology of eventual computer simulations (e.g., figure 5). Next, section 2.1 describes motor dynamics' modeling and control strategies. Next, discretization is discussed regarding the accompanying simulation. The simulation experiments are summarized in pithy tables of common figures of merit (e.g., means and standard deviations) with accompanying figures to provide qualitative depiction of the quantitative results compared. The comparison leads to recommendations for discretization method and time interval to achieve efficacy controlling DC motors.



Define System
Annotate motion states as "desired"
Assemble remaining variables
Reparametrize motion states into matrix of "knowns" and remaining variables into vector of unknowns
Use matrix vector product as deterministic self-awareness statements
Use feedback error signal in inverse solution of matrix-vector product as optimal learning
Validate by applying combined signal to nonlinear, coupled system

Figure 4: Topology of deterministic artificial intelligence whose depiction applied to unmanned underwater vehicle system whose actuators are powered by DC motors is in Figure 5.

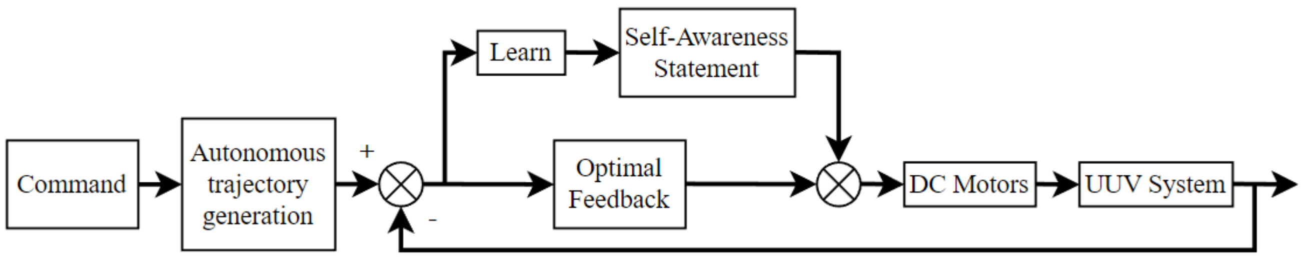


Figure 5: Topology of deterministic artificial intelligence applied to unmanned underwater vehicle system whose actuators are powered by DC motors.

2.1. Modeling DC motor dynamics and designing control strategies

The transfer function (a function in the frequency domain that relates the output of the system to its input), also known as the process equation or the process truth model, in the continuous time domain for natural dynamics of a DC motor is given by equation (1). Using analytical approaches presented in [15], the discrete transfer function for the DC motor is expressed using equation (2):

$$G(s) = \frac{B(s)}{A(s)} = \frac{1}{s(s+1)} \tag{1}$$

$$G(z) = \frac{B(z)}{A(z)} = \frac{b_0 z + b_1}{z^2 + a_1 z + a_2} = \frac{0.1065z + 0.0902}{z^2 - 1.9z + 0.88} \tag{2}$$

By altering the discretization method and the sample time, the values of b_0, b_1, a_1, and a_2 in equation (2) change, resulting in a different discrete transfer function each time. The transfer function in equation (2) has a zero at $z = -0.8469$ and poles at $z = 1.1, 0.80$. The transfer function has an unstable pole, and the approach with the adaptive control strategy will effectively relocate these poles to stable locations at $z = 0.2 \pm 0.2j$. The deterministic artificial intelligence modeling approach does not attempt to achieve pole relocation, but rather achieves an autonomously determined trajectory using proportional plus derivative (PD) feedback adaption of unknown parameters to follow the target path. The truth model also incorporates correlated Gaussian noise with distribution $N(0, 1/625)$ added as two delayed noise terms. Simulations are performed for both approaches using MATLAB®, and the code for each implementation is provided in Appendix A. Identical square wave inputs signals are fed to the adaptive as well as the learning control designs, while the latter method includes an autonomous path planning algorithm to create sinusoidal trajectories that commence at the starting discontinuity of the square wave and end at the peak of each square wave discontinuity. Using equation (2), the output equation for the DC motor system is obtained & given by equation (3):

$$y(t+2) = b_0 u(t+1) + b_1 u(t) - a_1 y(t+1) - a_2 y(t) \tag{3}$$

Through modeling with deterministic artificial intelligence, changes in input do not result in a drastically forced change in output to match the input signal value. Instead, with change in state of the input, a trajectory (sinusoidal in this work) is calculated for the output to follow in progression towards the target state so there is no undefined position (analytically) for any timestep. The dynamics of the process are asserted as the control mechanism in a feedforward fashion establishing self-awareness, and the control signal can be modulated through feedback parameters that are learned using a proportional-derivative feedback mechanism or 2-norm optimal methods. The process flow of deterministic artificial intelligence is illustrated in figure 4 while the topology including self-awareness statements are depicted in figure 5. Equation (4) demonstrates how the feedback parameters are calculated through batch least squares [22], with ϕ_d representing a matrix comprised of the desired trajectory states, $\hat{\theta}$ representing the learned parameters to adjust the control input u, and δu representing error in the control input.

$$u = \phi_d \hat{\theta} = \phi_d \left({\phi_d}^T \phi_d \right)^{-1} {\phi_d}^T \delta u \tag{4}$$

2.2. Discretization methods

The different discretization methods used in this study are: zero-order hold, first-order hold, impulse-invariant mapping, Tustin approximation, zero-pole matching equivalents, and least squares. The zero-order hold method provides an exact match between the continuous- & discrete-time systems in the time domain for staircase inputs. The zero-order hold discretization gives the discretized transfer function $H_d(z)$ of a continuous-time linear model $H(s)$. The zero-order hold method generates the continuous-time input signal $u(t)$ by holding each sample value $u(k)$ constant over one sample period, i.e., $u(t) = u(k)\ \forall\ kT_s \le t \le (k+1)T_s$. The signal $u(t)$ is the input to the continuous system $H(s)$. The output $y(k)$ results from sampling $y(t)$ every T_s seconds. The zero-order hold method is best suited when the exact discretization in the time domain for staircase inputs is required.

The first order hold method provides an exact match between the continuous- and discrete-time systems in the time domain for piecewise linear inputs. The method differs from the first order hold method by the underlying hold mechanism. To turn the input samples $u(k)$ into a continuous input $u(t)$, first order hold uses linear interpolation between samples given by equation (5):

$$u(t) = u(k) + (t - k * T_s)\big(u(k+1) - u(k)\big)/T_s\ \ \forall\ kT_s \le t \le (k+1)T_s \tag{5}$$

In general, the first order hold method is more accurate than the zero-order hold method, particularly for systems driven by smooth inputs. The first order hold method differs from standard causal first order hold and is more appropriately called the triangle approximation [25]. The first order hold method is also known as ramp-invariant approximation. The first order-hold discretization is best suited when exact discretization in the time domain is essential for piecewise linear inputs.

The impulse-invariant mapping produces a discrete-time model with the same impulse response as the continuous time system. The impulse-invariant mapping discretizes the system such that the impulse responses of the continuous & discretized systems match exactly. Therefore, it is evident that the impulse-invariant mapping approach is best used when exact discretization in the time domain is required for impulse train inputs.

The Tustin approximation, also known as the bilinear approximation, yields the best frequency-domain match between the continuous-time and discretized systems. The Tustin method relates the s-domain and z-domain transfer functions using the approximation given by equation (6):

$$z = e^{sT_s} \approx (1 + (0.5T_s)s)/(1 - (0.5T_s)s) \tag{6}$$

$$s' = (2z - 1)/(T_s z + 1) \tag{7}$$

In continuous to discrete conversions using the Tustin approach, the discretization $H_d(z)$ of a continuous transfer function $H(s)$ is done such that $H_d(z) = H(s')$ where s' is given by equation (7). When converting a state-space model using the Tustin method, the states are not preserved. The state transformation depends upon the state-space matrices and whether the system has time delays. The Tustin approximation is not defined for systems with poles at $z = -1$ and is ill-conditioned for systems with poles near $z = -1$. The Tustin approximation is used when good matching is required in the frequency domain between the continuous- and discrete-time models. The Tustin approach is also the best suited discretization method when the model of the system has important dynamics at certain frequency that need to be captured.

The zero-pole matching equivalents method of conversion applies only to single-input, single-output systems. The continuous and discretized systems have matching DC gains. The poles and zeros of the continuous and discretized system are related by the transformation: $z_i = e^{s_i T_s}$, where z_i is the ith pole or zero of the discrete-time system, s_i is the ith pole or zero of the continuous-time system, and T_s is the sample time of discretization. The zero-pole matching equivalents approach is preferred when the system at hand has a single-input, single-output model and good matching is desired in the frequency domain between the continuous- and discrete-time models.

Least squares method of discretization minimizes the error between the frequency responses of the continuous-time and discrete-time systems up to the Nyquist frequency using a vector-fitting optimization approach. The Nyquist frequency (or folding frequency) is a characteristic of a sampler which converts a continuous function into a discrete sequence. The value of the Nyquist frequency is one-half of the sampling rate. When the highest frequency (bandwidth) of a signal is less than the Nyquist frequency of the sampler, the resulting discrete-time sequence is said to be free of distortion. The least squares method is useful to capture fast system dynamics and large sample times are desired, for example, when computational resources are limited. The least square method is only suitable for changing continuous to discrete systems and for single-input, single-output systems. As with the Tustin approximation and zero-pole matching, the least squares method provides a good match between the frequency responses of the original continuous system & the converted discrete system. However, when using the least squares method with the same sample time as Tustin approximation or zero-pole matching, a smaller difference is obtained between the continuous and discrete frequency responses. Also, using the least squares method with a lower sample time will get the same results as with the Tustin approximation or zero-pole matching. Doing so is useful if computational resources are limited, since a slower sample time means that the processor must do less work.

3. Results

Simulations were performed for different time samples using each discretization method. Results are shown qualitatively in figure 6 with corresponding description in table 1. Each line plotted in figure 6 represents one of the four control strategies designed for the DC motor system. Quantitative results corresponding to the depicted qualitative results are provided in table 2.

Table 1. Discretization method and sample time used in each plot displayed in figure 6.

Plot	Description	Plot	Description	Plot	Description
(a)	ZOH & $T_s = 0.6\ s$	(b)	ZOH & $T_s = 0.5\ s$	(c)	ZOH & $T_s = 0.1\ s$
(d)	FOH & $T_s = 0.7\ s$	(e)	FOH & $T_s = 0.5\ s$	(f)	FOH & $T_s = 0.1\ s$
(g)	IIM & $T_s = 0.5\ s$	(h)	IIM & $T_s = 0.3\ s$	(i)	IIM & $T_s = 0.1\ s$
(j)	TA & $T_s = 0.8\ s$	(k)	TA & $T_s = 0.5\ s$	(l)	TA & $T_s = 0.1\ s$
(m)	ZPM & $T_s = 0.5\ s$	(n)	ZPM & $T_s = 0.3\ s$	(o)	ZPM & $T_s = 0.1\ s$
(p)	LS & $T_s = 0.6\ s$	(q)	LS & $T_s = 0.5\ s$	(r)	LS & $T_s = 0.1\ s$

[1] ZOH: Zero-order hold; FOH: First-order hold; IIM: Impulse-invariant mapping; TA: Tustin approximation; ZPM: Zero-pole matching equivalents; LS: Least squares.

For each of the discretization methods, sample time lower than 0.1 s results in significant error and deviation from the desired output for the deterministic artificial intelligence method. The plots (a), (d), (g), (j), (m), and (p) in figure 6 represent the largest permissible sample time value for the deterministic artificial intelligence approach for each discretization method. A sample time larger than the permissible value will lead to incorrect results. Based on the plots obtained by running the simulations using equation (1) and discretizing it using various methods & sample times, the mean of the absolute error between the output & the reference trajectory and the standard deviation of the error are tabulated in table 2. The lowest mean error for each method is highlighted as well.

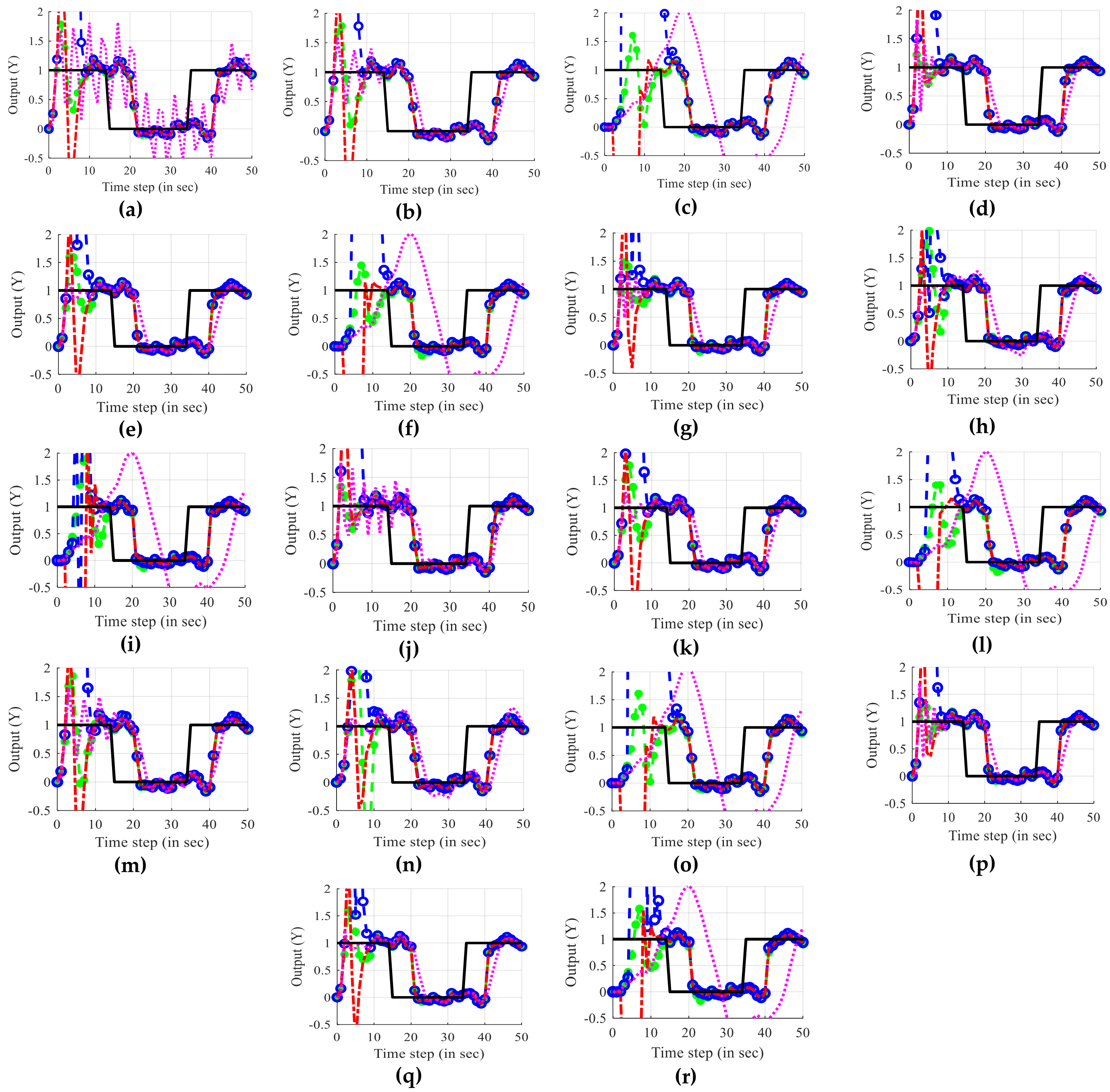


Figure 6. Output plot for each of the four control strategies with different discretization methods and sample time. Please refer to table 1 for a description of each plot.

4. Discussions

The prime contribution of this work is to demonstrate the effects of changing the discretization method and sample time on the control of DC motors used in unmanned underwater vehicles. This work directly uses the continuous transfer function and discretizes it to implement various adaptive control strategies as well as learning based deterministic AI. Latest literature states that using deterministic artificial intelligence yields an error of 0.224 [18]. However, from table 2 it is seen that using the appropriate sample time and discretization method for equation (1) will provide significantly lower errors and thus demonstrate the superiority of using deterministic artificial intelligence. Each discretization method can provide substantial lower error than stated in [18]. The lowest error of 0.0840 is obtained using the first order-hold method and sample time of 0.7s, a 62.5% reduction than stated in [18].

Table 2. Mean & standard deviation of the error for various discretization methods & sample time.

Sample Time (seconds)	ZOH	FOH	Impulse invariant mapping	Tustin approximation	Zero-pole matching equivalents	Least squares
0.8	--	--	--	0.1041/0.1646	--	--
0.7	--	**0.0840/0.1430**	--	**0.0894/0.1322**	--	--
0.6	0.1969/0.2683	0.0886/0.1297	--	0.0996/0.1403	--	**0.0912/0.1451**
0.5	**0.1137/0.1612**	0.1083/0.1513	**0.0967/0.1437**	0.1209/0.1658	**0.1219/0.1722**	0.0998/0.1416
0.4	0.1408/0.1897	0.1453/0.1953	0.1179/0.1628	0.1604/0.2141	0.1434/0.1932	0.1329/0.1802
0.3	0.2018/0.2647	0.2148/0.2771	0.1782/0.2331	0.2341/0.3012	0.2037/0.2669	0.1983/0.2571
0.2	0.3405/0.4225	0.3691/0.4512	0.3169/0.3903	0.3981/0.4849	0.3415/0.4238	0.3452/0.4232
0.1	0.8467/0.9812	0.8708/1.0032	0.8045/0.9288	0.9040/1.0405	0.8474/0.9822	0.8413/0.9698

[1] The smallest combination of error mean and standard deviation is highlighted for each method.

Recommended future research. *In this work, the sample time is changed in steps of 0.1 s. Reducing this further, a functional relationship could be generated between the sample time and the error statistics for the deterministic artificial intelligence approach. The advantages of doing this would be twofold. Firstly, it might provide even better results for the deterministic artificial intelligence approach. Secondly, comparing the error statistics of the approach with similar statistics from the other methods used in this work would provide quantitative results about the sample time at which the performance of the adaptive approach supersedes the performance of the learning-based approach.*

Funding: This research received no external funding.

Data Availability Statement: Data supporting reported results can be found by contacting the author at jcm483@cornell.edu.

Acknowledgments: The author acknowledges the contributions of Rohan Shah for providing the foundational code initiating this research.

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

Appendix A

This appendix provides the entire MATLAB® code that was implemented to get the results presented.

```
%% Investigating the effects of discretization for control of DC motors using deterministic artificial intelligence

clear; close all; clc;

rng(100);      %% Setting up the problem:

% Create square wave for reference input
maxtime=200;
Uc = zeros(1,201);
for i=1:length(Uc)
    if (mod(floor(i/20),2) == 0)
        Uc(i) = 1;
    else
        Uc(i) = 0;
    end
end
traj_Uc = zeros(1,length(Uc));
check = 1;
run_next = 0;
for i=1:length(Uc)-1
```

```
    if (check)
        traj_Uc(i) = Uc(i);
        diff = Uc(i+1)-Uc(i);
        lasti = i;
        lastval = Uc(i);
    end
    if (diff ~= 0)
        check = 0;
        if (run_next)
            traj_Uc(i) = lastval + diff/2*(1+(sin(0.2*pi*(i-lasti)-pi/2)));
        end
        run_next = 1;
        if (traj_Uc(i) == Uc(i) && (i ~= lasti))
            check = 1;
            run_next = 0;
        end
    end
end
Uc = Uc(1:200);

% Modeling the system
ts = 0.7;
Hc = tf(1,[1 1 0]);                                          % continuous transfer function
Hd = c2d(Hc,ts,'foh');                                       % discrete transfer function
% Hd=tf([0 0.1065 0.0902],poly([1.1 0.8]),0.5);       % analytic discrete transfer function
% Hd=tf([0 0.0902 0.06461],[1 -1.213 0.3679],0.5);     % offline discrete transfer function
B = [Hd.Numerator{1,1}(1,1),Hd.Numerator{1,1}(1,2),Hd.Numerator{1,1}(1,3)];
A = [Hd.Denominator{1,1}(1,1),Hd.Denominator{1,1}(1,2),Hd.Denominator{1,1}(1,3)];
nzeros=5;
factor = 25;
Noise=1/factor*randn(1,maxtime+nzeros);

%% Algorithm for Recursive Least Squares (RLS):
% Setup system parameters
a1=0; a2=0; b0=0.1; b1=0.2;
Am=poly([0.2+0.2j 0.2-0.2j]); Bm=[0 0.1065 0.0902];
am1=Am(2); am2=Am(3); a0=0;
Rmatrix=[];
lambda=1.0;
time=zeros(1,nzeros); Y_RLS=zeros(1,nzeros); Ym_RLS=zeros(1,nzeros);
U_RLS=ones(1,nzeros); Uc_RLS=[ones(1,nzeros),Uc];
P=[100 0 0 0;0 100 0 0;0 0 1 0;0 0 0 1]; THETA_hat_RLS(:,1)=[-a1 -a2 b0 b1]';
beta=[]; alpha = 0.5; gamma = 1.2;

%%%%%%%%%%%%%%%%%RECURSIVE LEAST SQUARES %%%%%%%%%%%%%%%%
for i=1:maxtime
    t=i+nzeros; time(t)=i; phi=[];
    Y_RLS(t)=[-A(2) -A(3) B(2) B(3)]*[Y_RLS(t-1) Y_RLS(t-2) U_RLS(t-1) U_RLS(t-2)]'+...
        Noise(t-1) + Noise(t-2);
    Ym_RLS(t)=[-Am(2) -Am(3) Bm(2) Bm(3)]*[Ym_RLS(t-1) Ym_RLS(t-2) Uc_RLS(t-1) Uc_RLS(t-2)]';
    BETA=(Am(1)+Am(2)+Am(3))/(b0+b1); beta=[beta BETA];

    % RLS implementation
    phi=[Y_RLS(t-1) Y_RLS(t-2) U_RLS(t-1) U_RLS(t-2)]';
    K=P*phi*1/(lambda+phi'*P*phi);
    P=P-P*phi*inv(1+phi'*P*phi)*phi'*P/lambda;                          % RLS-EF
    error(i)=Y_RLS(t)-phi'*THETA_hat_RLS(:,i);
    THETA_hat_RLS(:,i+1)=THETA_hat_RLS(:,i)+K*error(i);
    a1=-THETA_hat_RLS(1,i+1);a2=-THETA_hat_RLS(2,i+1);
    b0=THETA_hat_RLS(3,i+1);b1=THETA_hat_RLS(4,i+1);
    Af(:,i)=[1 a1 a2]'; Bf(:,i)=[b0 b1]';
    % Determine R,S, & T for CONTROLLER
    r1=(b1/b0)+(b1^2-am1*b0*b1+am2*b0^2)*(-b1+a0*b0)/(b0*(b1^2-a1*b0*b1+a2*b0^2));
    s0=b1*(a0*am1-a2-am1*a1+a1^2+am2-a1*a0)/(b1^2-a1*b0*b1+a2*b0^2)+b0*(am1*a2-a1*a2-a0*am2+a0*a2)/(b1^2-
a1*b0*b1+a2*b0^2);
    s1=b1*(a1*a2-am1*a2+a0*am2-a0*a2)/(b1^2-a1*b0*b1+a2*b0^2)+b0*(a2*am2-a2^2-a0*am2*a1+a0*a2*am1)/(b1^2-
a1*b0*b1+a2*b0^2);
    R=[1 r1];S=[s0 s1];T=BETA*[1 a0];
    Rmatrix=[Rmatrix r1];

    % Calculate control signal
    U_RLS(t)=[T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_RLS(t) Uc_RLS(t-1) U_RLS(t-1) Y_RLS(t) Y_RLS(t-1)]';
    U_RLS(t)=1.3*[T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_RLS(t) Uc_RLS(t-1) U_RLS(t-1) Y_RLS(t) Y_RLS(t-1)]'; % Arbitrarily increased to
duplicate text
end
%%%%%%%%%%%%%%%END OF RECURSIVE LEAST SQUARES %%%%%%%%%%%%%

%% Algorithm for Autoregressive moving average (ARMA):
% Setup system parameters
a1=0; a2=0; b0=0.01; b1=0.2;
Am=poly([0.2+0.2j 0.2-0.2j]); Bm=[0 0.1065 0.0902];
am1=Am(2); am2=Am(3); a0=0;
Rmatrix=[];
lambda=1;
time=zeros(1,nzeros); Y_ARMA=zeros(1,nzeros);
U_ARMA=ones(1,nzeros); Uc_ARMA=[ones(1,nzeros),Uc];
THETA_hat_ARMA=zeros(4,maxtime);
THETA_hat_ARMA(:,1)=[-a1 -a2 b0 b1]'; beta=[];
n=8;
P=10000*eye(n); P(1,1)=1000; P(2,2)=100; P(3,3)=100;
P(4,4)=10000; P(5,5)=1000; P(6,6)=100;
phi=[];

%%%%%%%%%%AUTOREGRESSIVE MOVING AVERAGE %%%%%%%%%%%%%%%%%
for i=1:maxtime
    t=i+nzeros; time(t)=i;
    Y_ARMA(t)=[-A(2) -A(3) B(2) B(3)]*[Y_ARMA(t-1) Y_ARMA(t-2) U_ARMA(t-1) U_ARMA(t-2)]'+...
        Noise(t-1) + Noise(t-2);                              % Create truth output
    BETA=(Am(1)+Am(2)+Am(3))/(b0+b1); beta=[beta BETA];
    phi=[phi; Y_ARMA(t-1) Y_ARMA(t-2) U_ARMA(t-1) U_ARMA(t-2)];
    if (i > 3)
        THETA_hat_ARMA(:,i+1) = inv(phi'*phi)*phi'*Y_ARMA(1+nzeros:t)';
    else
        THETA_hat_ARMA(:,i+1) = THETA_hat_ARMA(:,i);
    end
    a1=-THETA_hat_ARMA(1,i+1); a2=-THETA_hat_ARMA(2,i+1);
    b0=THETA_hat_ARMA(3,i+1); b1=THETA_hat_ARMA(4,i+1);      % Update A & B coefficients

    % Store final A and B for comparison with real A&B to generate epsilon errors
    Af(:,i)=[1 a1 a2]'; Bf(:,i)=[b0 b1]';

    % Determine R,S, & T for CONTROLLER
    r1=(b1/b0)+(b1^2-am1*b0*b1+am2*b0^2)*(-b1+a0*b0)/(b0*(b1^2-a1*b0*b1+a2*b0^2));
    s0=b1*(a0*am1-a2-am1*a1+a1^2+am2-a1*a0)/(b1^2-a1*b0*b1+a2*b0^2)+b0*(am1*a2-a1*a2-a0*am2+a0*a2)/(b1^2-
a1*b0*b1+a2*b0^2);
    s1=b1*(a1*a2-am1*a2+a0*am2-a0*a2)/(b1^2-a1*b0*b1+a2*b0^2)+b0*(a2*am2-a2^2-a0*am2*a1+a0*a2*am1)/(b1^2-
a1*b0*b1+a2*b0^2);
    R=[1 r1]; S=[s0 s1]; T=BETA*[1 a0];
    Rmatrix=[Rmatrix r1];

    % Calculate control signal
    U_ARMA(t)=[T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_ARMA(t) Uc_ARMA(t-1) U_ARMA(t-1) Y_ARMA(t) Y_ARMA(t-1)]';
    % Arbitrarily increased to duplicate text
    U_ARMA(t)=1.3*[T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_ARMA(t) Uc_ARMA(t-1) U_ARMA(t-1) Y_ARMA(t) Y_ARMA(t-1)]';
end
%%%%%%%%%%END OF AUTOREGRESSIVE MOVING AVERAGE %%%%%%%%%%%

%% Algorithm for Extended Least Squares (ELS):
% Setup system parameters
a1=0; a2=0; b0=0.01; b1=0.2;
Am=poly([0.2+0.2j 0.2-0.2j]); Bm=[0 0.1065 0.0902];
am1=Am(2); am2=Am(3); a0=0;
Rmatrix=[];
lambda=1;
time=zeros(1,nzeros); Y_ELS=zeros(1,nzeros); Ym_ELS=zeros(1,nzeros);
U_ELS=ones(1,nzeros); Uc_ELS=[ones(1,nzeros),Uc];
THETA_hat_ELS(:,1)=[-a1 -a2 b0 b1]';beta=[];    % initialize P(t), THETA_hat(t) & Beta
epsilon=[zeros(1,nzeros+maxtime)];
n=8;
P=10000*eye(n); P(1,1)=1000; P(2,2)=100; P(3,3)=100;
P(4,4)=10000; P(5,5)=1000; P(6,6)=100;
theta_hat_els=zeros(n,1);

%%%%%%%%%%%%%%%%%EXTENDED LEAST SQUARES %%%%%%%%%%%%%%%%
for i=1:maxtime
    phi=[]; t=i+nzeros; time(t)=i;
    Y_ELS(t)=[-A(2) -A(3) B(2) B(3)]*[Y_ELS(t-1) Y_ELS(t-2) U_ELS(t-1) U_ELS(t-2)]'+...
        Noise(t-1) + Noise(t-2);                                        % create truth output
    Ym_ELS(t)=[-Am(2) -Am(3) Bm(2) Bm(3)]*[Ym_ELS(t-1) Ym_ELS(t-2) Uc_ELS(t-1) Uc_ELS(t-2)]';
    BETA=(Am(1)+Am(2)+Am(3))/(b0+b1); beta=[beta BETA];
    k=i+nzeros;
    phi=[Y_ELS(t-1) Y_ELS(t-2) U_ELS(t-1) U_ELS(t-2) epsilon(t) epsilon(t-1) epsilon(t-2) epsilon(k-3)]';
    K=P*phi*1/(1+phi'*P*phi);
    P=P-P*phi*pinv(1+phi'*P*phi)*phi'*P;
    error(i)=Y_ELS(k)-phi'*theta_hat_els(:,i);
    theta_hat_els(:,i+1)=theta_hat_els(:,i)+K*error(i);
    epsilon(k)=Y_ELS(k)-phi'*theta_hat_els(:,i+1);            % Form Posterior Residual
    THETA_hat_ELS(:,i+1)=theta_hat_els(1:4,i+1);

    % Update A & B coefficients
    a1=-THETA_hat_ELS(1,i+1); a2=-THETA_hat_ELS(2,i+1);
    b0=THETA_hat_ELS(3,i+1); b1=THETA_hat_ELS(4,i+1);

    % Store final A and B for comparison with real A&B to generate epsilon errors
    Af(:,i)=[1 a1 a2]'; Bf(:,i)=[b0 b1]';
    r1=(b1/b0)+(b1^2-am1*b0*b1+am2*b0^2)*(-b1+a0*b0)/(b0*(b1^2-a1*b0*b1+a2*b0^2));
    s0=b1*(a0*am1-a2-am1*a1+a1^2+am2-a1*a0)/(b1^2-a1*b0*b1+a2*b0^2)+b0*(am1*a2-a1*a2-a0*am2+a0*a2)/(b1^2-
a1*b0*b1+a2*b0^2);
    s1=b1*(a1*a2-am1*a2+a0*am2-a0*a2)/(b1^2-a1*b0*b1+a2*b0^2)+b0*(a2*am2-a2^2-a0*am2*a1+a0*a2*am1)/(b1^2-
a1*b0*b1+a2*b0^2);
    R=[1 r1]; S=[s0 s1]; T=BETA*[1 a0];
    Rmatrix=[Rmatrix r1];

    % Calculate control signal
    U_ELS(t)=[T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_ELS(t) Uc_ELS(t-1) U_ELS(t-1) Y_ELS(t) Y_ELS(t-1)]';
    U_ELS(t)=1.3*[T(1) T(2) -R(2) -S(1) -S(2)]*[Uc_ELS(t) Uc_ELS(t-1) U_ELS(t-1) Y_ELS(t) Y_ELS(t-1)]'; % Arbitrarily increased to
duplicate text
end
%%%%%%%%%%%%%%%END OF EXTENDED LEAST SQUARES %%%%%%%%%%%%%%

%% Algorithm for Deterministic Artificial Intelligence (DAI):
% Create command signal
```

```
nzeros=5; time=zeros(1,nzeros);
Y_DAI=zeros(1,nzeros);                                    % initialize ouput vectors
U_DAI=ones(1,nzeros);
t=0:200;
hvy_m = [zeros(1,nzeros) traj_Uc];
eb = Y_DAI(1) - hvy_m(1);
err = 0; kp = 2.0; kd = 6.0;
phid = []; ustar = [];
hatvec = zeros(4,1);
Rmatrix=[];

%%%%%%%%%%%%%%%%%%%%DETERMINISTIC AI %%%%%%%%%%%%%%%%%%
% Loop through the output data Y(t)
for i=1:maxtime+1
    t=i+nzeros; time(t)=i;
    de = err-eb;
    u = kp*err + kd*de;
    U_DAI(t-1) = u;
    Y_DAI(t)=[-A(2) -A(3) B(2) B(3)]*[Y_DAI(t-1) Y_DAI(t-2) U_DAI(t-1) U_DAI(t-2)]'+...
        Noise(t-1) + Noise(t-2);
    phid = [phid; Y_DAI(t) -Y_DAI(t-1) Y_DAI(t-2) -U_DAI(t-2)];
    ustar = [ustar; u];
    newest = phid\ustar;
    hatvec(:,i) = newest;
    eb = err;
    err = hvy_m(t)-Y_DAI(t);
end
%%%%%%%%%%%%%%%%%END OF DETERMINISTIC AI %%%%%%%%%%%%%%%%%

THETA_hat_DAI = [hatvec(2,:)./hatvec(1,:); hatvec(3,:)./hatvec(1,:);...
    ones(1,201)./hatvec(1,:); hatvec(4,:)./hatvec(1,:)];

mean_abs_error_DAI = mean(abs(traj_Uc - Y_DAI(1,6:end)))
mean_abs_error_ELS = mean(abs(Uc - Y_ELS(1,6:end)))
mean_abs_error_ARMA = mean(abs(Uc - Y_ARMA(1,6:end)))
mean_abs_error_RLS = mean(abs(Uc - Y_RLS(1,6:end)))
std_error_DAI = std(traj_Uc - Y_DAI(1,6:end))
std_error_ELS = std(Uc - Y_ELS(1,6:end))
std_error_ARMA = std(Uc - Y_ARMA(1,6:end))
std_error_RLS = std(Uc - Y_RLS(1,6:end))

%% Plotting results:
figure(); hold on; grid on;
plot(time(1,1:205), Y_RLS,'g--*'); plot(time(1,1:205), Y_ARMA,'b--o');
plot(time(1,1:205), Y_ELS,'r-.'); plot(time, Y_DAI,'m:');
xlabel('Time step (in sec)'); ylabel('Output (Y)');
plot(time(1:200),Uc,'k-');
axis([0 50,-0.5 2.0]);
% legend('RLS','ARMA','ELS','DAI','Reference Input');
title("Discretization using First Order Hold and Sample Time "+ ts);
PrepFigPresentation(gcf);

function PrepFigPresentation(fignum)

% Prepares a figure for presentations
% Font size: 10
% Font Name: Times
% LineWidth: 2
%
figure(fignum);
fig_children=get(fignum,'children');                          % find all sub-plots
```

```
for i=1:length(fig_children)
    set(fig_children(i),'FontSize',10);
    set(fig_children(i),'FontName','times');
    fig_children_children=get(fig_children(i),'Children');
    set(fig_children_children,'LineWidth',2);
end
end
```

References

1. Liu, Z.; Zhuang, X.; Wang, S. Speed Control of a DC Motor using BP Neural Networks. In Proceedings of the 2003 IEEE Conference on Control Applications, Istanbul, Turkey, 25–25 June 2003; pp. 832–835.
2. Mishra, M. Speed Control of DC Motor Using Novel Neural Network Configuration. Bachelor's Thesis, National Institute of Technology, Rourkela, India, 2009.
3. Hernández-Alvarado, R.; García-Valdovinos, L.G.; Salgado-Jiménez, T.; Gómez-Espinosa, A.; Fonseca-Navarro, F. Neural Network-Based Self-Tuning PID Control for Underwater Vehicles. Sensors 2016, 16, 1429.
4. Rashwan, A. An Indirect Self-Tuning Speed Controller Design for DC Motor Using A RLS Principle. In Proceedings of the 21st International Middle East Power Systems Conference (MEPCON), Cairo, Egypt, 17–19 December 2019; pp. 633–638.
5. Keller, J. Navy Eyes Unmanned Underwater Vehicle (UUV) Weapons Payloads to Stop or Disable 160-Foot Ships at Sea. 24 May 2018. Available online: https://www.militaryaerospace.com/unmanned/article/16726886/navy-eyes-unmanned-underwater-vehicle-uuv-weapons-payloads-to-stop-or-disable-160foot-ships-at-sea (accessed on 19 December 2022).
6. Rees, C. Maxon Launches High Torque DC Brushless Motors. 5 May 2015. Available online: https://www.unmannedsystemstechnology.com/2015/05/maxon-launches-high-torque-dc-brushless-motors/ (accessed on 19 December 2022).
7. Underwater Thruster Propeller Motor for ROV AUV. Available online: https://www.alibaba.com/product-detail/underwater-thruster-propeller-motor-for-ROV_62275939884.html (accessed on 19 December 2022).
8. Slotine, J.; Benedetto, M. Hamiltonian adaptive control on spacecraft. *IEEE Trans. Autom. Control* **1990**, *35*, 848–852.
9. Slotine, J.; Weiping, L. *Applied Nonlinear Control*; Prentice Hall: Englewood Cliffs, NJ, USA, 1991.
10. Fossen, T. Comments on "Hamiltonian Adaptive Control of Spacecraft". *IEEE Trans. Autom. Control* **1993**, *38*, 671–672.
11. Åström, K.; Wittenmark, B. On the Control of Constant but Unknown Systems. In Proceedings of the 5th IFAC World Congress, Paris, France, 12–17 June 1972.
12. Åström, K.; Wittenmark, B. On self-tuning regulators. Automatica 1973, 9, 185–199.
13. Åström, K.; Wittenmark, B. Adaptive Control; Addison-Wesley: Boston, FL, USA, 1995; pp. 43, 48, 63–65, 331.
14. Sands, T.; Kim, J. Agrawal, B. Improved Hamiltonian Adaptive Control of spacecraft, 2009 IEEE Aerospace conference, Big Sky, Montana, USA. 07-14 March 2009.
15. Sheng, M.; Alvi, M.; Lorenz, R. GMR-based Integrated Current Sensing in SiC Power Modules with Phase Shift Error Reduction. *IEEE Journal of Emerging and Selected Topics in Power Electronics* **2022**, *10*(3), 3477 – 3487.
16. Sands, T. Development of Deterministic Artificial Intelligence for Unmanned Underwater Vehicles (UUV). *J. Mar. Sci. Eng.* **2020**, *8*(8), 578.
17. Sands, T. Control of DC Motors to Guide Unmanned Underwater Vehicles. Appl. Sci. 2021, 11, 2144.
18. Shah, R.; Sands, T. Comparing Methods of DC Motor Control for UUVs. Appl. Sci. 2021, 11, 4972.
19. Osburn J.; Whitaker H.; Kezer A. New developments in the design of model reference adaptive control systems *Inst. Aeronautical Sciences* **1961**, *61*(39).
20. Gauss, C. F. *Combinationes erroribus minimis obnoxiae. Parts 1, 2 and Suppl.* 1823, 1-108. Described in Sprott, D. Gauss's contribution to statistics, *Historia Mathematica* **1978**, *5*, 183-203. doi: 10.1016/0315-0860(78)90049-6.
21. Sands, T.; Kim, J.; Agrawal, B. Spacecraft fine tracking pointing using adaptive control. In Proceedings of the 58th International Astronautical Congress, Hyderabad, India, 24–28 September 2007; International Astronautical Federation: Paris, France, 2007.
22. Smeresky, B.; Rizzo, A.; Sands, T. Optimal Learning and Self-Awareness versus PDI. Algorithms 2020, 13, 23.
23. Zhai, H.; Sands, T. Controlling Chaos in Van Der Pol Dynamics Using Signal-Encoded Deep Learning. *Mathematics* **2022**, *10*, 453.
24. Zhai, H.; Sands, T. Comparison of Deep Learning and Deterministic Algorithms for Control Modeling. *Sensors* **2022**, *22*, 6362
25. Franklin, G.F., Powell, D.J., and Workman, M.L., Digital Control of Dynamic Systems (3rd Edition), Prentice Hall, 1997.