# Machine Learning in Control Systems: An Overview of the State of the Art

Signe Moe(✉), Anne Marthine Rustad, and Kristian G. Hanssen

Department of Mathematics and Cybernetics, SINTEF Digital, Oslo, Norway
{Signe.Moe,AnneMarthine.Rustad,Kristian.Gaustad.Hanssen}@sintef.no

**Abstract.** Control systems are in general based on the same structure, building blocks and physics-based models of the dynamic system regardless of application, and can be mathematically analyzed w.r.t. stability, robustness and so on given certain assumptions. Machine learning methods (ML), on the other hand, are highly flexible and adaptable methods but are not subject to physic-based models and therefore lack mathematical analysis. This paper presents state of the art results using ML in the control system. Furthermore, a case study is presented where a neural network is trained to mimic a feedback linearizing speed controller for an autonomous ship. The neural network outperforms the traditional controller in case of modeling errors and measurement noise.

**Keywords:** Control systems · Machine learning · Hybrid analytics

## 1 Introduction

The use of Machine Learning (ML) is increasing rapidly within numerous applications. These highly flexible, adaptable methods are in several settings, such as natural language processing [13] and medical diagnosis [19], showing better performance, higher robustness and adaptability beyond that of traditional approaches. However, these methods are lacking in terms of stability proofs and mathematical analysis and explanation.

On the other hand, traditional control systems are mostly based on the same blocks, regardless of the application. These blocks are illustrated in Fig. 1 and are described in detail in Sect. 2. In general, such control loops are based on system and parameter identification, and are designed to give the controlled system desired properties such as stability and robustness given reasonable assumptions. However, identifying and modeling a system is challenging and in some cases infeasible due to unobservability and highly non-linear effects. This paper presents an overview of state of the art of machine learning in the control system, where one or more of the traditional control blocks have been replaced or combined with a machine learning-approach. This results in a *hybrid analytics control system*, which may benefit from the strengths of both approaches.


M. Bramer and M. Petridis (Eds.): SGAI-AI 2018, LNAI 11311, pp. 250–265, 2018.
https://doi.org/10.1007/978-3-030-04191-5_23

This paper is organized as follows. Section 2 presents the traditional control loop and the different system blocks, their main purpose and the most commonly used techniques and considerations. State of the art results of ML in the control system are described in Sect. 3. Furthermore, a case study is presented in Sect. 4, where a traditional, model-based speed controller for an autonomous surface vessel is replaced with a neural network controller. Finally, conclusions are given in Sect. 5.

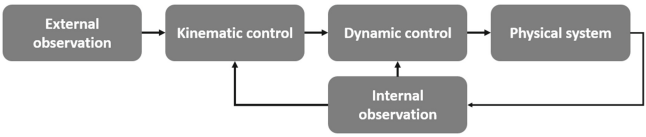



**Fig. 1.** The traditional control loop consists of multiple components for observation and control.

## 2 The Control Loop

This section briefly describes the overall function and interaction of the blocks in a traditional control loop (Fig. 1). Furthermore, the most commonly used approaches and considerations for the different control blocks are briefly described. Control loops are used to regulate a number of states such as motion, temperature and voltage, and can be applied to various physical systems, e.g. robotic manipulators, autonomous vehicles and process systems. The dynamics of the system is modeled using the laws of physics, and these models are commonly used for simulations and in the design of the different control components, which are described below.

In a control loop, the **internal observation** block provides information on the internal states of the system, either from direct sensor measurements or from estimations based on sensor data combined with the system model [10]. Furthermore, the control system may require information about its surroundings to properly be able to perform its tasks. This occurs in the **external observation** block. The relevant states of the system and the external observations are used by the **kinematic** and **dynamic controllers**. The kinematic controller calculates the desired behavior of the system, i.e. what the states of the system should be to achieve one or more tasks [3]. These reference states are given as input to the dynamic controller in addition to the actual states, and the objective of the dynamic controller is to use the control inputs to drive the difference between these to zero, i.e. to bring the system to the desired state [17].

An illustrative example is an adaptive cruise control of a car. In this case, the internal observation block estimates the velocity of the car by measuring the rotations per minute of the wheels and calculating the resulting velocity. The external observation block uses sensors such as radars or cameras to detect other vehicles or obstacles ahead. The original reference velocity is constant and

manually set by the driver. However, based on the external observations, this velocity may be altered by the kinematic controller should it for instance be necessary to slow down to avoid collisions with slower-moving vehicles. Finally, the dynamic controller ensures that the engine speeds up if the car is moving slower than the reference velocity and vice versa.

### 2.1 Modeling of Physical Systems

Physics-based models of dynamic systems are often given as differential equations describing the relationship between the system states, their derivatives w.r.t. time and the control inputs. A number of approaches to analyze the behavior of the system exist, e.g. Lyapunov's methods and passivity-based theorems [17]. Hence, they are often used in combination with the model when designing of the control system to ensure that the estimated states converge to the actual states, or that the actual states converge to the reference states.

For a robotic manipulator, the state is given by the joint angles $\boldsymbol{q}$. The position and orientation of the end effector is denoted $\boldsymbol{\eta}$. The system model is then given as

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}, \tag{1}$$

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}, \tag{2}$$

where $\boldsymbol{J}$ is the configuration-dependent Jacobian matrix, $\boldsymbol{M}$ and $\boldsymbol{C}$ are the mass/inertial and Coriolis matrices, $\boldsymbol{g}$ is a vector describing the gravitational forces and $\boldsymbol{\tau}$ are the forces produced by the joint actuators [31]. This model assumes frictionless joints.

Although such mathematical models are useful tools based on a solid foundation of physics and reasoning, there are also challenges related to achieving a sufficiently accurate model. They are often based on assumptions that may not always be satisfied, e.g. frictionless joints or negligible forces at low velocities [34]. Furthermore, accurate modeling of non-linear effects such as damping and friction and parameter estimation is highly challenging [7]. Finally, such models, often referred to as *process plants*, may be too complex to be utilized in the control system, and are often replaced by simplified models, called *control plants* describing only the most essential physical effects of the system [6,16].

### 2.2 Kinematic Control and Decision Making

A system may have one or multiple tasks in a prioritized order that may be in conflict, or several modes of operation. The kinematic controller considers the desired overall behavior of the system and calculates the desired states to achieve this based on the current states of system and external observations.

A common type of kinematic controller is based on **inverse kinematics**. For instance, a robot manipulator is controlled through actuators at the joints $\boldsymbol{q}$. A given joint configuration corresponds to an exact end effector position and orientation $\boldsymbol{\eta}$, but a given position/orientation may be achieved by multiple joint

configurations. Based on specified tasks such as desired end effector configuration $\boldsymbol{\eta}_{\text{des}}$ and the current configuration $\boldsymbol{q}$, the kinematic controller calculates the corresponding desired joint angles $\boldsymbol{q}_{\text{des}}$ [4]. However, the stability analysis is based on the assumption that the reference $\boldsymbol{q}_{\text{des}}$ is tracked perfectly [3]. Furthermore, it is unsuitable for tasks requiring force control, such as manipulation tasks [18].

Unlike the example above, many systems are **underactuated**, i.e. do not have a control input related to every state. One example of this is surface vessels, which typically only have two control inputs (a rudder and a thruster) in spite of having 6 states (described in Sect. 4). Thus, the kinematic controller must calculate the reference for the states that are controlled such that the desired behavior is achieved [11].

Furthermore, a system may have multiple modes of operations. For complex systems, it is highly challenging to make a rule-based **decision maker** to handle all eventualities. A common approach is to use fuzzy logic [2], which ensures a smooth transition between different modes of operation.

### 2.3 Dynamic Control

The dynamic controller receives the desired and actual states $\boldsymbol{q}_{\text{des}}$ and $\boldsymbol{q}$. The necessary control forces $\boldsymbol{\tau}$ to drive the states to their references, i.e. the error $\boldsymbol{e} \triangleq \boldsymbol{q}_{\text{des}} - \boldsymbol{q}$ to zero, are then calculated.

A widely used dynamic controller is the proportional integral derivative **(PID)** controller (3). This is based on a linear combination of the error, its derivative and its integral, and gain matrices $\boldsymbol{K}_{\text{p}}$, $\boldsymbol{K}_{\text{i}}$ used for controller tuning. Thus, it is not affected by modeling errors and is applicable to almost any system. However, optimal control or stability can not be guaranteed, and badly tuned PIDs may even result in unstable systems. Furthermore, it performs poorly on systems with strong non-linear effects, and it is susceptible to measurement noise in sensors [1].

$$\boldsymbol{\tau}(t) = \boldsymbol{K}_{\text{p}}\boldsymbol{e}(t) + \boldsymbol{K}_{\text{i}} \int_0^t \boldsymbol{e}(s)ds + \boldsymbol{K}_{\text{d}}\dot{\boldsymbol{e}}(t). \tag{3}$$

The **feedback linearizing** controller is model-based and aims to cancel out non-linear effects of the system to overcome one of the weaknesses of the PID-controller. Thus, it is possible to guarantee stability using such controllers, but they are susceptible to modeling errors and measurement noise. For instance, for the dynamics of a manipulator given by (2), a feedback linearizing PD-controller is given in (4) (the argument $t$ is omitted for readability). The resulting closed loop system has a uniformly globally exponentially stable equilibrium point at $\boldsymbol{e} = \boldsymbol{0}$, i.e. the error will converge to zero in a finite amount of time [17]. Furthermore, it is clear that the effects related to gravity, centripetal and Coriolis forces have been compensated for by the control system.

$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})(\ddot{\boldsymbol{q}}_{\text{des}} + \boldsymbol{K}_{\text{p}}\boldsymbol{e} + \boldsymbol{K}_{\text{d}}\dot{\boldsymbol{e}}) + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}). \tag{4}$$

A model predictive controller (**MPC**) is implemented by solving an optimization problem at every control step. The system behavior is predicted by the use of a model, and a trajectory of the system states over the prediction horizon is obtained. This allows formulating constraints on both decision variables and the predicted states. Only the first control step is used; at the next control step new state estimates are obtained, and the optimization problem is solved over again. The success of MPC has been attributed to the ability to handle multivariable systems with both input and output constraints in a consistent manner [25]. However, the controller can be computationally heavy and its performance depends heavily on the accuracy of the process model [28].

### 2.4 Internal and External Observation

A variety of **sensors** are available to provide relevant information about internal states, such as encoders, Global Positioning System (GPS), Inertial Measurement Units (IMUs) and thermometers. However, physical sensors have several shortcomings. First, certain sensors are expensive and may constitute a large part of the total cost of a control system in addition to inducing errors such as stochastic noise and biases. Furthermore, certain states and signals are challenging to measure directly [10] and must be estimated using an observer. For instance, the **Kalman filter** is a widely used observer to estimate missing states from indirect and noisy measurements and the system model. Although the traditional Kalman filter is only applicable to linear systems, the **extended Kalman filter** is based on linear approximations of non-linear systems [14]. Note that this is computationally heavy for complex systems.

In certain situations, it is necessary to gather and process information about the environment to assist the kinematic or dynamic controller. Examples of such **external observations** are environmental disturbances such as wind and waves and obstacle detection. Commonly used sensors for the latter include radars, lasers and cameras, where the sensor data must be analyzed through computer vision algorithms to determine whether a given area is free of obstacles [29]. Traditionally, computer vision methods have been based on camera imaging geometry, feature detection, stereo vision, and so on [32]. However, the use of machine learning is highly increasing in the field of image analysis. The following section will provide several examples of this.

## 3 ML in Control Systems

This section will present examples of machine learning approaches being used in one or more blocks of the control loop in Fig. 1.

### 3.1 Modeling of Physical Systems

As mentioned in Sect. 2, several components in the control loop are often based on differential equations describing the behavior of the system, which are are highly challenging to solve explicitly. However, ML may provide additional tools for analyzing such equations. In [20–22], neural networks in various forms are used to approximate solutions of ordinary and partial differential equations (PDEs). In [30], an algorithm for approximating the solution of a PDE using a deep neural network is presented. The network is trained assuming no knowledge of the solution and a theorem is presented stating that the neural network will converge to the solution of the PDE as the size of the network increases.

ML may also be applied to learn a dynamic model on the same form as (2) without solving the equation. In [33], a neural network is used to identify parts of a model for an underwater vehicle as illustrated in Fig. 2. In most practical cases, all parameters apart from the hydrodynamic damping parameters can be calculated with sufficient accuracy with ordinary methods. Hence, the term $\hat{\Phi}$ is estimated using a neural network, whereas the known parameters are represented by the known model $\Phi_M$. After training, the neural network is used in a feedback linearizing controller to compensate for the damping.

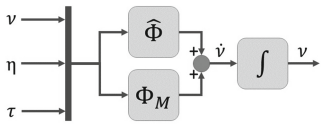



**Fig. 2.** Illustration reproduced from [33]. The nonlinear dynamic equations with neural network $\hat{\Phi}$ to model unknown parameters in parallel with known parameters and model structure $\Phi_M$ is used to estimate the vehicle accelerations $\dot{\boldsymbol{\nu}}$ given the position/orientation $\boldsymbol{\eta}$, velocities $\boldsymbol{\nu}$ and control inputs $\boldsymbol{\tau}$.

In [8], a neural network is used to approximate the dynamics of the leader and follower, and this approximation is utilized in a feedback control law to achieve formation control. This control law also requires information about the internal states of the agents, and to achieve this another neural network is applied as an observer to estimate linear and angular velocities of the robots, which are assumed to have limited communication between them. Thus, machine learning techniques are applied both in the dynamic controller and internal observer of the control loop.

### 3.2 Kinematic Control/Decision Making

ML is often applied in the kinematic control component in situations requiring online analysis of the environment. In such cases, the input to the algorithms are often camera images or other sensor data from the external observation component. In [15], a quadrotor is trained to autonomously follow a forest trail to assist in search and rescue missions. To achieve this, the robot has a front view camera. The images are given as input to the kinematic controller, where a deep convolutional neural network (CNN) outputs a calculated probability of the trail being left, straight or right relative to the camera reference frame. These probabilities are then recalculated into desired orientation and forward velocity.[1]

[1] A narrated video summary is available at http://bit.ly/perceivingtrails/.

Similarly, in [27] a deep convolutional neural network is used in the kinematic controller for an underwater snake robot to ensure proper docking. The snake head is equipped with a single camera which provides images as input to the kinematic controller. During training, the position $(x, y)$ and orientation $\psi$ of the robot are measured through a camera positioning system and used with a traditional kinematic controller to calculate the desired heading $\psi_{\text{des}}$ required to follow a path to the docking station. The path is designed so that the docking station is visible from the camera on the path. The error $\tilde{\psi} = \psi_{\text{des}} - \psi$ is the input to the dynamic controller. However, in uncontrolled environments such as subsea, it is not straightforward to obtain the position and orientation of the robot. Therefore, a neural network is trained to estimate $\tilde{\psi}$ given only camera input. Hence, the network is trained to mimic the behavior of the traditional kinematic controller without being dependent on state measurements that may be unavailable outside of the controlled environment (see Fig. 3).

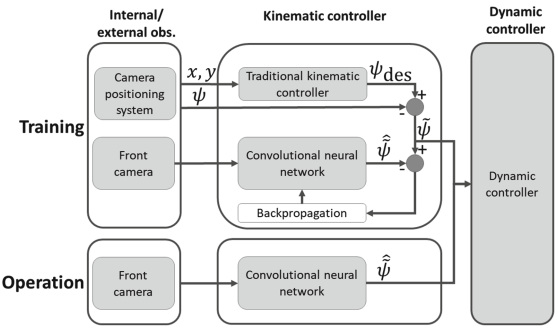



**Fig. 3.** Illustration reproduced from [27]. A CNN is trained to provide the same heading error as a traditional kinematic controller without depending on state measurements that are not available in uncontrolled environments.

### 3.3 Dynamic Control

Machine learning may also be applied as dynamic controllers. In fact, *Machine Learning Control* (MLC) is the concept of using machine learning algorithms to learn an effective control law $\boldsymbol{b} = \boldsymbol{K}(\boldsymbol{s})$ that maps the system input (sensors $\boldsymbol{s}$) to the system output (actuators $\boldsymbol{b}$) motivated by problems involving complex control tasks where it may be difficult or impossible to model the system and develop a useful control law. Instead, data is utilized to learn effective controllers [9].

A similar approach as in [27] is applied in [36] for the dynamic control component. Here, a deep neural network is trained to mimic the behavior of an MPC for a quadcoptor that should avoid columns in its path. The MPC requires estimation or measurements of position. The neural network, however, is not dependent on position but is based only on observations that are directly available from the

vehicle's onboard sensors. Since the MPC is only used at training time, one can perform training in a controlled environment where the full state is known at training time (e.g. using motion capture), but unavailable at test time. During test time, the neural network performs very well both in the face of modeling errors and never before seen scenarios such as multiple obstacles[2].

In [35], a bipedal robot is considered. The dynamics are modeled similar to (2), but also include static and dynamic friction and other disturbance torques and faults that occur in robot manipulator. These are considered unknown, hence they cannot be directly included in a feedback linearizing controller. Therefore, the dynamic controller of the bipedal robot is

$$\boldsymbol{\tau}_{\text{total}} = \boldsymbol{\tau} + \boldsymbol{\tau}_{\text{c}}(\hat{y}_{\text{rnn}}), \tag{5}$$

where $\boldsymbol{\tau}$ is given by (4) and $\boldsymbol{\tau}_{\text{c}}$ is a term to compensate for the error and disturbance terms, and in turn to improve system response speed and reduce the steady state error. This term is based on the output of a recurrent neural network (RNN) $\hat{y}_{\text{rnn}}$, and simulations confirm that the tracking error $\boldsymbol{e}$ is decreased significantly when the RNN is activated compared to only using the feedback linearizing controller.

### 3.4 Internal and External Observation

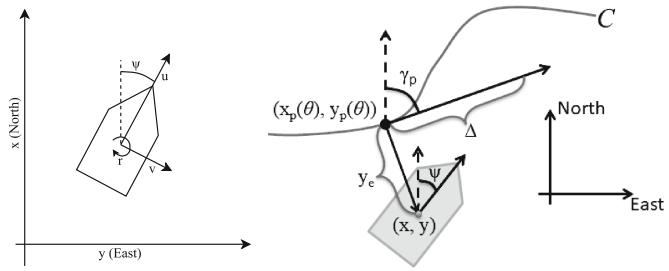
In [26], a Radial Basis Function neural network (RBF NN) is trained to predict air data for a small unmanned aerial vehicle (UAV). Traditional air databooms are sensor-based and are therefore physically impractical for small UAVs. The RBF NN has a 5-dimensional input consisting of pressure measurements from sensors on the leading edge of the wing and outputs estimates of the freestream static pressure, freestream airspeed and UAV angle of attack, which is important information for the remainder of the control system. Data is gathered in a wind tunnel where the network output variables are known and the network is then trained in a supervised manner.

## 4 Case Study: Speed Controller

This section presents a case study of ML-in-the-loop, where a neural network replaces the dynamic speed controller of an autonomous unmanned surface vessel (USV). The neural network is trained to mimic the behavior of a feedback linearizing controller. The entire control system including the original speed controller is presented below, followed by a description of the training and evaluation of the controller.

The states of the USV are given by $\boldsymbol{\eta} \triangleq [x, y, \psi]^T$ and $\boldsymbol{\nu} \triangleq [u, v, r]^T$ and are illustrated in Fig. 4(a). The control inputs $T$ and $\delta$ are the thruster force and rudder angle, respectively. In this use case, we assume that measurements of the full state are available. Furthermore, we assume that there are no external elements and disturbances to consider.

[2] A video summary is available at http://rll.berkeley.edu/icra2016mpcgps/.




(a) (b)

**Fig. 4.** (a) The states of the USV: position $(x, y)$ and orientation $\psi$ of the USV with respect to the North-East-Down (NED) frame and surge, sway and yaw rate $u$, $v$ and $r$ w.r.t. the body-fixed frame. (b) Desired path $C$, path-tangential reference frame with orientation $\gamma_{\mathrm{p}}(\theta)$ and cross-track error $y_{\mathrm{e}}$ illustrated.

### 4.1 Model of Physical System

The simulation model in component form is given in [5,11] as

$$\begin{aligned}
\dot{x} &= \cos(\psi)u - \sin(\psi)v, \\
\dot{y} &= \sin(\psi)u + \cos(\psi)v, \\
\dot{\psi} &= r, \\
\dot{u} &= F_u(v, r) - \frac{d_{11}}{m_{11}}u + \frac{b_{11}}{m_{11}}T, \\
\dot{v} &= X(u)r + Y(u)v, \\
\dot{r} &= F_r(u, v, r) + \frac{m_{22}b_{32} - m_{23}b_{22}}{m_{22}m_{33} - m_{23}^2}\delta.
\end{aligned} \tag{6}$$

This model assumes non-zero velocity. The expressions for $F_u(v, r)$, $X(u)$, $Y(u)$ and $F_r(u, v, r)$ are given in [23]. The control inputs consist of the thruster force $T$ and rudder angle $\delta$ of the USV, and the model constants $m_{ij}$, $d_{ij}$ and $b_{ij}$ are mass/inertia, hydrodynamic damping and actuator configuration parameters that depend on the physical properties of the ship. The numeric values for these parameters are given in [12].

### 4.2 Kinematic Control

For this case study, the control system should make the vessel follow a given smooth path $C$ and maintain a desired constant surge velocity $u_{\mathrm{des}}$. The path variables are illustrated in Fig. 4(b). The path $C$ is parametrized by $\theta \geq 0$ with respect to the NED-frame as $(x_{\mathrm{p}}(\theta), y_{\mathrm{p}}(\theta))$. The cross-track error $y_{\mathrm{e}}$ is computed as the orthogonal distance between the vessel position $(x, y)$ and the path-tangential reference frame defined by the point $(x_{\mathrm{p}}, y_{\mathrm{p}})$. Note that the

position of the path-tangential reference frame is always such that the along-track error $x_e \equiv 0$. The orientation of the path-tangential frame is defined as

$$\gamma_p(\theta) = \text{atan}\left(\frac{y_p'(\theta)}{x_p'(\theta)}\right). \tag{7}$$

It is clear from Fig. 4(b) that $y_e = 0$ implies that the vessel is on the desired path. Hence, we define the control objectives as

$$\begin{aligned} \lim_{t\to\infty} u(t) &= u_{\text{des}}, \\ \lim_{t\to\infty} y_e(t) &= 0. \end{aligned} \tag{8}$$

The kinematic controller in this case should provide the desired states to ensure that the control objectives are satisfied. In case of the desired velocity, this is trivially equal to the constant, desired velocity $u_{\text{des}}$. It is proven in [24] that if the reference heading

$$\psi_{\text{des}} = \gamma_p(\theta) - \underbrace{\text{atan}\left(\frac{v}{u_{\text{des}}}\right)}_{\triangleq \beta_{\text{des}}} - \text{atan}\left(\frac{y_e}{\Delta}\right) \tag{9}$$

is tracked, the USV will converge to and follow the path. Here, the constant $\Delta > 0$ is the lookahead-distance, and the term $\beta_{\text{des}}$ is a side-slip term to compensate for the sideways motion that occurs when a ship turns. In this case study, $\Delta = 75$ m.

### 4.3 Dynamic Control

Feedback linearizing controllers are highly suitable to ensure tracking of the desired surge velocity $u_{\text{des}}$ and heading $\psi_{\text{des}}$ for the system (6) due to the many non-linear components of the system. The controllers are given as

$$T = \frac{m_{11}}{b_{11}}\left(-F_u(v,r) + \frac{d_{11}}{m_{11}}u_{\text{des}} + \dot{u}_{\text{des}} - k_u(u - u_{\text{des}})\right), \tag{10}$$

$$\delta = \frac{m_{22}m_{33} - m_{23}^2}{m_{22}b_{32} - m_{23}b_{22}}\Big(-F_r(u,v,r) + \ddot{\psi}_{\text{des}} - k_\psi(\psi - \psi_{\text{des}}) - k_r(\dot{\psi} - \dot{\psi}_{\text{des}})\Big), \tag{11}$$

where the gains $k_u$, $k_\psi$ and $k_r$ are strictly positive constant controller gains. It is proven in [23] that the controllers (10) and (11) ensure that the references $u_{\text{des}}$ and $\psi_{\text{des}}$ are tracked. In this case study, the controller gains are chosen as $k_u = 0.1\ \text{s}^{-1}$, $k_\psi = 0.04\ \text{s}^{-2}$ and $k_r = 0.9\ \text{s}^{-1}$.

### 4.4 Training of the Neural Network Speed Controller

The model and control system presented in Sects. 4.1, 4.2 and 4.3 was implemented in Matlab. The neural network surge controller is trained using supervised learning to mimic the behavior of the feedback linearizing controller (10). Therefore, the network input $\boldsymbol{X} \in \mathbb{R}^5$ is composed of the USV velocity $u$, $v$ and $r$ and the desired surge velocity and acceleration $u_{\text{des}}$ and $\dot{u}_{\text{des}}$, and the output is the estimated thruster force $\hat{T}$. Note that in this particular case study the reference velocity is always constant, and therefore $\dot{u}_{\text{des}} = 0\ m/s^2$ for all data points except the moment the velocity reference changes from one to another constant value. Future work includes training for time-varying surge velocity references.

Simulations with a perfectly known model and no measurement noise are run to collect training data. To ensure tracking of the reference surge velocity for all types of paths, simulations are performed on a large number of sine wave paths and a straight line path, given as

$$x_{\text{p}}(\theta) = \theta, \quad y_{\text{p}}(\theta) = A \sin\left(\frac{2\pi}{L}\theta\right), \tag{12}$$

where $A$ and $L$ are the amplitude and period of the path in meters (Fig. 5). In the data gathering, simulations have been run for every combination of $A \in [0,\ 100]$ and $L \in [500,\ 1500]$ in 5 and 100 m intervals, respectively. Each path is simulated for 800 s and data is saved with a frequency of 20 Hz. The reference velocity $u_{\text{des}}$ is chosen randomly from a predefined set of values every 200 s. This interval is given as $u_{\text{des}} \in [0.5,\ 5.75]$ m/s, where $u_{\text{des}}$ increases in 0.25 m/s increments. The simulation data is then used to train the neural network using TensorFlow. Several possible architectures were evaluated. The chosen network has three hidden layers of 100 nodes each with ReLu activation functions, which are suitable as they do not squeeze the input and they are less likely to have vanishing gradient problems than activation functions like sigmoid and hyperbolic tangent functions. The Adam optimizer was used to train the network to minimize the cost-function

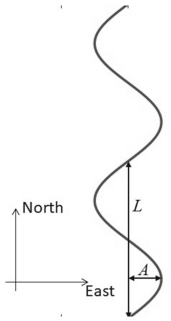



**Fig. 5.** Sine wave path used to gather training data for multiple combinations of $A$ and $L$.

$$\text{Cost} = \frac{1}{m}\sum_{i=1}^{m}(\hat{T} - T)^2, \tag{13}$$

where $m$ is the number of data points, $T$ is the thruster force calculated by the feedback linearizing controller (10), and $\hat{T}$ is the output of the neural network. Since training data is saved with high frequency it was not necessary to reuse any points in the training. We therefore use a batch size of $m = 1000$ data points and run training until all data from simulations have been used once.

### 4.5 Evaluation of the Neural Network Speed Controller

To evaluate the performance of the neural network speed controller, simulations are performed with a new path and reference velocities that the network has not been specifically trained for and compared to the performance of the feedback linearizing controller (10). This is described in Case I below. In addition, robustness is tested by introducing measurement noise (Case II) and modeling errors (Case III). In all test cases, simulations are run for 500 s where the reference velocity is increased from $u_{\text{des}} = 1.82$ m/s to $u_{\text{des}} = 3.64$ m/s after 250 s. The new path is illustrated in Fig. 6 and is defined as

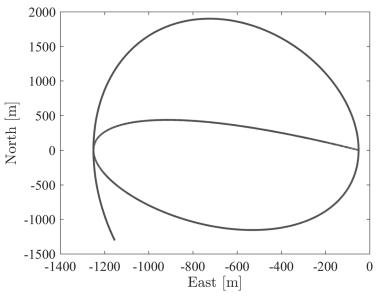



**Fig. 6.** Curved path used in simulations to evaluate the neural network.

$$x_{\text{p}}(\theta) = 1.2\theta \sin(0.005\theta), \quad y_{\text{p}}(\theta) = 600 \cos(0.005\theta) - 650. \tag{14}$$

**Case I - Perfect model and no measurement noise**: In this case the feedback linearizing controller (10) perfectly cancels out the nonlinear terms, stabilizes the system and guarantees that the surge velocity converging to and tracking the reference velocity. This is confirmed by Fig. 7(a)–(b), where the thruster force calculated by the controller results in the surge velocity $u$ converging to $u_{\text{des}}$. Similarly, the neural network controller has learned to mimic the behavior of the feedback linearizing controller and also results in tracking of the reference velocity $u_{\text{des}}$ in spite of simulating a new path and setting $u_{\text{des}}$ to values the network was not explicitly trained on. The main difference occurs early in the simulation, where the neural network controller imposes less thrust force than the feedback linearizing controller, resulting in a slightly slower convergence to the first reference value. This occurs in all test cases.

**Case II - Perfect model and measurement noise**: In this case, white noise with a zero mean and a standard deviation of 0.4 m/s, 0.04 m/s and 3.0 rad/s is added on the velocity measurements $u$, $v$ and $r$, respectively. This corresponds to around 10% of the actual maximum value during simulations. In this case, the feedback linearizing controller is unable to ensure perfect tracking of the reference velocity. As seen in Fig. 7(c)–(d), the resulting surge velocity $u$ converges to a near-stationary error of approximately 0.2 m/s. However, the neural network controller ensures nearly perfect tracking of $u_{\text{des}}$ in spite of the measurement noise.

**Case III - Modeling errors and no measurement noise**: In this case, the numeric values in the feedback linearizing controller and neural network remain unchanged, but some of the numeric values in the simulation model are altered. In particular, the parameters $m_{11}$, $d_{11}$ and $b_{11}$ are set to 85%, 110% and 108% of their original value, respectively. Thus, we evaluate the effects of

notable modeling errors of magnitudes over and under the actual value. These values mainly influence the surge dynamics, so these modeling errors do not notably affect the heading controller (11). This is desirable to test the difference in performance of the two speed controllers only. Similar to Case II, the feedback linearizing controller results in a stationary deviation (Fig. 7(e)–(f)), whereas the neural network controller ensures perfect tracking of the reference velocity in spite of the modeling errors.

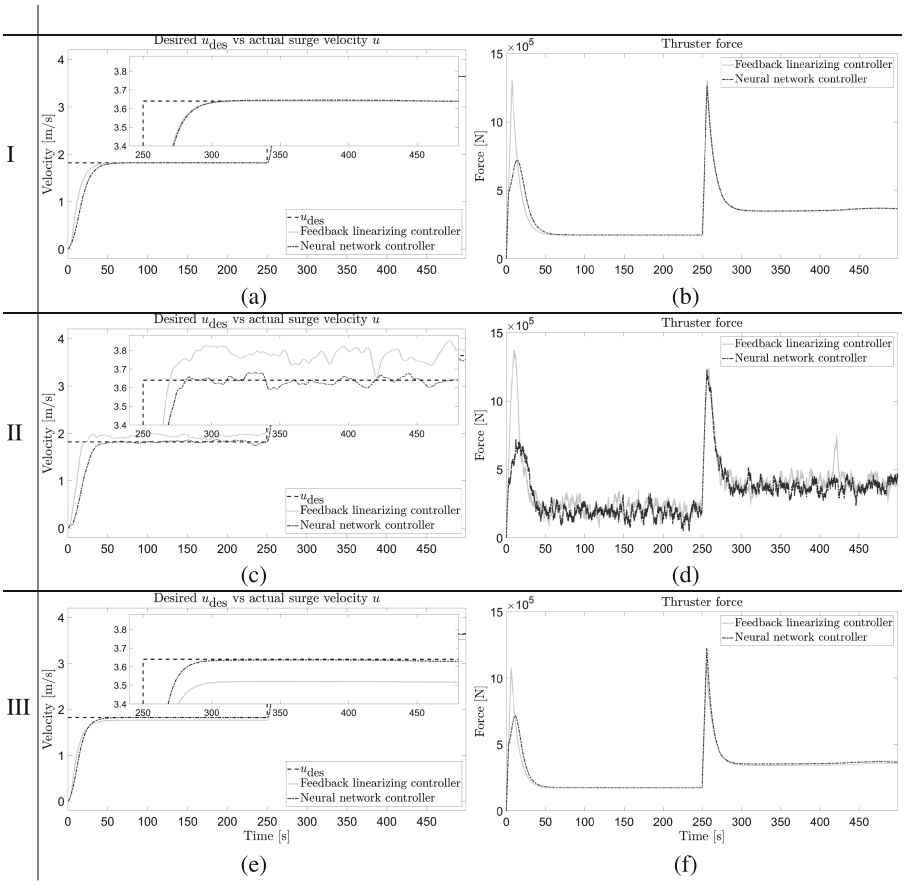



**Fig. 7.** Simulation results for evaluation and comparison of the feedback linearizing and neural network surge controller for three test cases. Confirming the theoretical results, the feedback linearizing controller performs very well in Case I, but results in a stationary deviation in case of measurement noise and modeling errors (Case II and III, respectively). The neural network is trained to mimic the behavior of the feedback linearizing controller on simulation data without modeling errors or measurement noise, but outperforms the original controller and achieves tracking also for Case II and III.

## 5 Conclusions

This paper presents the building blocks of a general control system along with some of the most used traditional methods. In general, traditional control methods are based on simple principles which are not sufficiently accurate for complex systems, or on a physics-based dynamic model of the system, which is time-consuming, challenging to derive and subject to assumptions and simplifications.

ML is in general highly adaptable and flexible, and may be introduced in a control system to a large degree (e.g. replacing and merging several blocks) or as a small addition to the existing system (e.g. accounting for unmodeled effects) to increase robustness, accuracy, and to remove simplifying assumptions. Finally, ML controllers may be trained to mimic the behavior of more complex, traditional controllers using fewer and/or cheaper sensors, thereby lowering the cost of the control system and increasing the applicability.

A use case is presented where a neural network is trained to mimic the behavior of a feedback linearizing controller to control the surge velocity of an autonomous ship. Training data is collected in simulations with a perfectly known model and no measurement noise. During the testing phase, the performance of the two controllers is compared also when measurement noise and modeling errors are present. In spite of being trained to mimic the feedback linearizing controller under ideal circumstances, the neural network controller performs better in the more challenging test cases and is thereby more robust and accurate than the original controller.

**Acknowledgments.** This project has been supported through the basic funding from the Norwegian Research Council.

## References

1. Aastrøm, K.J., Murray, R.M.: Feedback Systems: An Introduction for Scientists and Engineers. Princeton University Press, Princeton (2008)
2. Abdullah, L.: Fuzzy multi criteria decision making and its applications: a brief review of category. Procedia Soc. Behav. Sci. **97**, 131–136 (2013)
3. Antonelli, G.: Stability analysis for prioritized closed-loop inverse kinematic algorithms for redundant robotic systems. IEEE Trans. Robot. **25**(5), 985–994 (2009)
4. Antonelli, G., Arrichiello, F., et al.: The null-space-based behavioral control for autonomous robotic systems. Intell. Serv. Robot. **1**(1), 27–39 (2008)
5. Caharija, W., Candeloro, M., et al.: Relative velocity control and integral LOS for path following of underactuated surface vessels. In: Proceedings of the 9th IFAC Conference on Manoeuvring and Control of Marine Craft, pp. 380–385 (2012)
6. Candeloro, M., Sørensen, A.J., et al.: Observers for dynamic positioning of ROVs with experimental results. IFAC Proc. Vol. **45**(27), 85–90 (2012)
7. Chin, C., Lau, M.: Modeling and testing of hydrodynamic damping model for a complex-shaped remotely-operated vehicle for control. J. Mar. Sci. Appl. **11**(2), 150–163 (2012)
8. Dierks, T., Jagannathan, S.: Neural network output feedback control of robot formations. IEEE Trans. Syst. Man Cybern. **40**(2), 383–399 (2010)

9. Duriez, T., Brunton, S.L., Noack, B.R.: Machine learning control (MLC). Machine Learning Control – Taming Nonlinear Dynamics and Turbulence. FMIA, vol. 116, pp. 11–48. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-40624-4_2
10. Ellis, G.: Observers in Control Systems: A Practical Guide. Academic Press, Cambridge (2002)
11. Fossen, T.I.: Handbook of Marine Craft Hydrodynamics and Motion Control. Wiley, Hoboken (2011)
12. Fredriksen, E., Pettersen, K.Y.: Global kappa-exponential way-point manoeuvering of ships. In: Proceedings of the 43rd IEEE Conference on Decision and Control, pp. 5360–5367 (2004)
13. Goodfellow, I., Bengio, Y., et al.: Deep Learning. MIT Press, Cambridge (2016)
14. Grewal, M.S., Andrews, A.P.: Kalman Filtering : Theory and Practice. Wiley, Hoboken (2001)
15. Guisti, A., Guzzi, J., et al.: A machine learning approach to visual perception of forest trails for mobile robots. Robot. Autom. Lett. **1**(2), 661–667 (2016)
16. Kelasidi, E., Pettersen, K.Y., et al.: A control-oriented model of underwater snake robots. In: Proceedings of the 2014 IEEE International Conference on Robotics and Biomimetics, pp. 753–760 (2014)
17. Khalil, H.K.: Nonlinear systems. Prentice Hall PTR, Upper Saddle River (2002)
18. Khatib, O.: A unified approach for motion and force control of robot manipulators: the operational space formulation. IEEE J. Robot. Autom. **3**(1), 43–53 (1987)
19. Kononenko, I.: Machine learning for medical diagnosis: history, state of the art and perspective. Artif. Intell. Med. **23**(1), 89–109 (2001)
20. Lagaris, I.E., Likas, A., et al.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Netw. **9**(5), 987–1000 (1998)
21. Lagaris, I.E., Likas, A.C., et al.: Neural-network methods for boundary value problems with irregular boundaries. IEEE Trans. Neural Netw. **11**(5), 1041–1049 (2000)
22. Malek, A., Shekari Beidokhti, R.: Numerical solution for high order differential equations using a hybrid neural network – optimization method. Appl. Math. Comput. **183**(1), 260–271 (2006)
23. Moe, S., Pettersen, K.Y.: Set-based line-of-sight (LOS) path following with collision avoidance for underactuated unmanned surface vessel. In: Proceedings of the 1st Conference on Control Technology and Applications (2016)
24. Moe, S., Pettersen, K.Y., et al.: Line-of-sight curved path following for underactuated USVs and AUVs in the horizontal plane under the influence of ocean currents. In: Proceedings of the 1st IEEE Conference on Control Technology and Applications (2016)
25. Qin, J., Badgwell, T.: A survey of industrial model predictive control technology. Control Eng. Pract. **11**, 733–764 (2003)
26. Samy, I., Postlethwaite, I., et al.: Neural-network-based flush air data sensing system demonstrated on a mini air vehicle. J. Aircr. **47**(1), 18–31 (2010)
27. Sans-Muntadas, A., Pettersen, K.Y., et al.: Learning an AUV docking maneuver with a convolutional neural network. In: Proceedings of the IEEE Oceans (2017)
28. Seborg, D.E., Edgar, T.F., et al.: Process dynamics and control. AIChE J. **54**(11), 3026–3026 (2008)
29. Singh, S., Keller, P.: Obstacle detection for high speed autonomous navigation. In: Proceedings of the 1991 IEEE International Conference on Robotics and Automation, pp. 2798–2805 (1991)
30. Sirignano, J., Spiliopoulos, K.: DGM: a deep learning algorithm for solving partial differential equations (2017). http://arxiv.org/abs/1708.07469

31. Spong, M.W., Hutchinson, S.: Robot Modeling and Control. Wiley, Hoboken (2005)
32. Szeliski, R.: Computer Vision: Algorithms and Applications. Springer, London (2011). https://doi.org/10.1007/978-1-84882-935-0
33. van de Ven, P.W.J., Johansen, T.A., et al.: Neural network augmented identification of underwater vehicle models. Control Eng. Pract. **15**(6), 715–725 (2007)
34. Vidoni, R., Carabin, G., Gasparetto, A., Mazzetto, F.: Stability analysis of an articulated agri-robot under different central joint conditions. Robot 2015: Second Iberian Robotics Conference. AISC, vol. 417, pp. 335–346. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27146-0_26
35. Wu, Y., Song, Q., et al.: Robust recurrent neural network control of biped robot. J. Intell. Robot. Syst. **49**(2), 151–169 (2007)
36. Zhang, T., Kahn, G., et al.: Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search. In: Proceedings of the 2016 International Conference on Robotics and Automation, pp. 528–535 (2016)