Scientific Research Publishing

Intelligent Control and Automation, 2017, 8, 175-197
http://www.scirp.org/journal/ica
ISSN Online: 2153-0661
ISSN Print: 2153-0653

Hamiltonian Servo: Control and Estimation of a Large Team of Autonomous Robotic Vehicles

Vladimir Ivancevic[1], Peyam Pourbeik[2]

[1]Joint and Operations Analysis Division, Defence Science & Technology Group, Canberra, Australia
[2]CEWD, Assured Communications, Survivable Networks, Defence Science & Technology Group, Adelaide, Australia
Email: Vladimir.Ivancevic@dst.defence.gov.au, Peyam.Pourbeik@dst.defence.gov.au

How to cite this paper: Ivancevic, V. and Pourbeik, P. (2017) Hamiltonian Servo: Control and Estimation of a Large Team of Autonomous Robotic Vehicles. *Intelligent Control and Automation*, **8**, 175-197.
https://doi.org/10.4236/ica.2017.84014

Received: November 7, 2017
Accepted: November 27, 2017
Published: November 30, 2017

Abstract

This paper proposes a novel *Hamiltonian servo system*, a combined modeling framework for control and estimation of a large team/fleet of autonomous robotic vehicles. The Hamiltonian servo framework represents high-dimensional, nonlinear and non-Gaussian generalization of the classical *Kalman servo system*. After defining the Kalman servo as a motivation, we define the affine Hamiltonian neural network for adaptive nonlinear control of a team of UGVs in continuous time. We then define a high-dimensional Bayesian particle filter for estimation of a team of UGVs in discrete time. Finally, we formulate a hybrid Hamiltonian servo system by combining the continuous-time control and the discrete-time estimation into a coherent framework that works like a predictor-corrector system.

Keywords

Team of UGVs, Kalman Servo, Hamiltonian Control, Bayesian Estimation

1. Introduction

Today's military forces face an increasingly complex operating environment, whether dealing with humanitarian operations or in the theater of war. In many situations one has to deal with a wide variety of issues from logistics to lack of communications infrastructure [1]. In many situations one may be even denied satellite access and hence unable to utilize services such as GPS for Geo-location and the ability reconnect back to HQ [2].

Given the increasing success of *artificial intelligence* (AI) techniques in recent times and the advancements made in *unmanned vehicle* (UV) design, autonomous systems in particular teams or swarms of autonomous vehicles have become a potential attractive solution to solving the complexity issues facing the military today. Teams or swarms of intelligent autonomous vehicles would also introduce a level of survivability to the systems which would increase its utility in such complex and hostile environments [3].

A problem which arises in utilizing large teams or swarms of autonomous vehicles is one of control. Recently, the US *Defence Advanced Research Projects Agency* (DARPA) has sought to address some of the control issues associated with large swarm through its *Offensive Swarm-Enabled Tactics* (OFFSET) [4]. Therefore, there is a clear need for both local and global control of large teams or swarms of UV. This would allow for autonomous or semi-autonomous operation of large teams of UV which would be easily manageable.

This paper proposes a Hamiltonian servo system which is a combined modeling framework for the control and estimation of such autonomous teams of UV's. This paper will demonstrate the ability of this framework to handle the high-dimensional, nonlinear and non-Gaussian nature of the problem at hand. The authors will also show this to be a generalization of the classic Kalman Servo System. Finally results of a number of simulations will be presented to demonstrate the validity of the framework through the authors implementation using Mathematica & C++ code (which is provided in the **Appendix**).

Slightly more specifically, we recall here that traditional robotics from the 1970s-80s were mainly focused on building *humanoid robots,* manipulators and leg locomotion robots. Its modeling framework was linear dynamics and linear control, that is, linearized mechanics of multi-body systems (derived using Newton-Euler, Lagrangian, Gibs-Appel or kinetostatics equations of motion) and controlled by Kalman's linear quadratic controllers (for a comprehensive review, see [5]-[11]). The pinnacle of this approach to robotics in the last decade has been the famous Honda robot ASIMO (see [12]), with a related Hamiltonian biomechanical simulator [13].

In contrast, contemporary robotics research has mainly been focused on *field robotics*, based on estimation rather than control, resulting in a variety of *self-localization and mapping* (SLAM) algorithms (see [14] [15] [16] [17] [18] and the OpenSLAM web-site [19]).

Instead of following one of these two sides of robotics, in the present paper we will try to combine them. From a robotics point-of-view, control and estimation are two complementary sides, or necessary components, of efficient manipulation of autonomous robotic vehicles. In this paper we attempt to develop a computational framework (designed in Wolfram Mathematica® and implemented in C++) that unifies both of these components. From a mathematical point-of-view, in this paper we attempt to provide nonlinear, non-Gaussian and high-dimensional generalization of the classical *Kalman servo* system, outlined in the next section.

2. Motivation: Kalman Servo

In 1960s, Rudolf Kalman developed concept of a *controller-estimator servo*

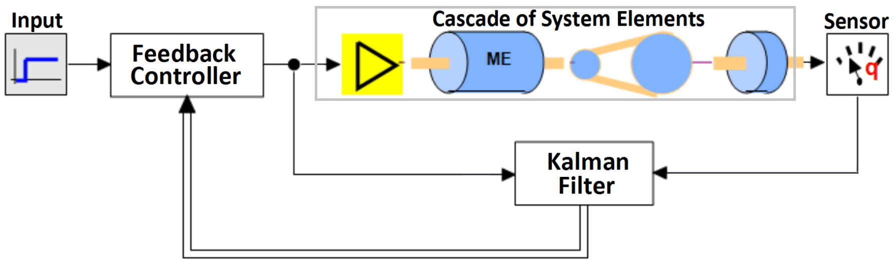

Figure 1. Kalman's fundamental controller-estimator servo system, including state-space dynamics, Kalman filter and feedback controller.

system (see Figure 1; for original Kalman's work see [20] [21] [22]), which consists of the following three main components, written in classical boldface matrix notation:

System state dynamics: linear, low-dimensional, multi-input multi-output process cascade (as implemented in Matlab® Control and Signal Processing toolboxes):

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \quad \boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u}, \quad \left[\boldsymbol{x}(0) = \boldsymbol{x}_0\right], \tag{1}$$

where we have three vector spaces: *state space* $\mathbb{X} \subset \mathbb{R}^n$, *input space* $\mathbb{U} \subset \mathbb{R}^m$ and *output space* $\mathbb{Y} \subset \mathbb{R}^k$, such that $\boldsymbol{x}(t) \in \mathbb{X}$ is an *n*-vector of *state variables,* $\boldsymbol{u}(t) \in \mathbb{U}$ is an *m*-vector of *inputs* and $\boldsymbol{y}(t) \in \mathbb{Y}$ is a *k*-vector of *outputs;* $\boldsymbol{x}_0$ is the initial state vector. The matrix variables are the following maps: i) $n \times n$ dimensional *state dynamics* $\boldsymbol{A} = \boldsymbol{A}(t) : \mathbb{X} \to \mathbb{X}$; ii) $n \times m$ dimensional *input map* $\boldsymbol{B} = \boldsymbol{B}(t) : \mathbb{U} \to \mathbb{X}$; iii) $k \times n$ dimensional *output map* $\boldsymbol{C} = \boldsymbol{C}(t) : \mathbb{X} \to \mathbb{Y}$; and (iv) $k \times m$ dimensional *input-output transform* $\boldsymbol{D} = \boldsymbol{D}(t) : \mathbb{U} \to \mathbb{Y}$. The first equation in (1) is called the state or dynamics equation and the second equation in (1) is called the output or measurement equation.

Kalman feedback controller: optimal LQR-controller:[1]

$$\boldsymbol{y} = \boldsymbol{K}\left[\boldsymbol{x}_{ref}(t) - \boldsymbol{x}(t)\right], \tag{2}$$

with the *controller gain matrix*: $\boldsymbol{K} = \boldsymbol{K}(t) : \mathbb{X} \to \mathbb{Y} = \boldsymbol{R}_c^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{P}_c$ determined by the matrix $\boldsymbol{P}_c$ governed by the *controller Riccati* ordinary differential equation (ODE):

$$\dot{\boldsymbol{P}}_c = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{P}_c + \boldsymbol{P}_c\boldsymbol{A} + \boldsymbol{Q}_c - \boldsymbol{P}_c\boldsymbol{B}\boldsymbol{R}_c^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{P}_c, \quad \left[\boldsymbol{P}_c(0) = \boldsymbol{P}_0^c\right],$$

where $\boldsymbol{P}_0^c$ is the initial controller matrix. $\boldsymbol{Q}_c : \mathbb{X} \to \mathbb{X}$ and $\boldsymbol{R}_c : \mathbb{U} \to \mathbb{U}$ are matrices from the quadratic cost function:

$$J = \int_{t_0}^{t_1} \left(\boldsymbol{x}^{\mathrm{T}}Q\boldsymbol{x} + \boldsymbol{u}^{\mathrm{T}}R\boldsymbol{u}\right)\mathrm{d}t. \tag{3}$$

Kalman filter: optimal LQG-estimator[2] with the (additive, zero-mean, delta-correlated) Gaussian white noise $\boldsymbol{\eta}(t)$:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{L}\left[\boldsymbol{y} - (\boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u})\right] + \boldsymbol{\eta}, \quad \left[\boldsymbol{x}(0) = \boldsymbol{x}_0\right], \tag{4}$$

[1]The acronym LQR means *linear quadratic regulator.*
[2]The acronym LQG means *linear quadratic Gaussian.*

with the *filter gain matrix*: $\boldsymbol{L}=\boldsymbol{L}(t):\mathbb{X}\to\mathbb{Y}=\boldsymbol{R}_f^{-1}\boldsymbol{P}_f\boldsymbol{C}^{\mathrm{T}}$ determined by the matrix $\boldsymbol{P}_f$ governed by the *filter Riccati* ODE:

$$\dot{\boldsymbol{P}}_f=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{P}_f+\boldsymbol{P}_f\boldsymbol{A}+\boldsymbol{Q}_f-\boldsymbol{P}_f\boldsymbol{C}^{\mathrm{T}}\boldsymbol{R}_f^{-1}\boldsymbol{C}\boldsymbol{P}_f,\ \left[\boldsymbol{P}_f(0)=\boldsymbol{P}_0^f\right],$$

where $\boldsymbol{P}_0^f$ is the initial filter matrix. $\boldsymbol{Q}_f:\mathbb{X}\to\mathbb{X}$ and $\boldsymbol{R}_f:\mathbb{U}\to\mathbb{U}$ are covariance matrices from the noisy η-generalization of the quadratic cost function (3).

Defined in this way, Kalman's controller-estimator servo system has a two-cycle action, similar to the predictor-corrector algorithm: in the first time-step the servo controls the plant (e.g., a UGV), in the second step it estimates the plants (x,y) coordinates, in the next step it corrects the control, then estimates again, and so on.

In the remainder of this paper we will develop a nonlinear, non-Gaussian and high-dimensional generalization of the Kalman servo.

3. Control of an Autonomous Team of UGVs: Affine Hamiltonian Neural Network

Now we switch from matrix to ($\alpha,\beta=1,\cdots,n$)-index notation,[3] to label the position of $n\times n$ individual UGVs within the swarm's global plane coordinates, longitude and latitude, respectively. The first step in the nonlinear generalization of the Kalman servo is replacing the linear, low-dimensional, state dynamics (1) and control (2)-(3) with nonlinear, adaptive and high-dimensional dynamics and control system, as follows.

In the recent paper [23], we have proposed a dynamics and control model for a joint autonomous land-air operation, consisting of a swarm of *unmanned ground vehicles* (UGVs) and swarm of *unmanned aerial vehicles* (UAVs). In the present paper we are focusing on a swarm of UGVs only, from both control and estimation perspectives. It can be modeled as an affine Hamiltonian control system (see [24]) with many degrees-of-freedom (DOFs), reshaped in the form of a (q,p)-pair of attractor matrix equations with nearest-neighbor couplings defined by the matrix of affine Hamiltonians:

$$H_{\alpha\beta}=\frac{1}{2}\left(q_{\alpha\beta}q_{\alpha-1\beta-1}+p_{\alpha\beta}p_{\alpha-1\beta-1}\right).$$

The following pair of attractor matrix equations defines the activation dynamics of a bidirectional recurrent neural network:

$$\dot{q}_{\alpha\beta}=\varphi\left(A^q-q_{\alpha\beta}-\omega_{\alpha\beta}^q q_{\alpha\beta}^2 p_{\alpha\beta}\right)-\sum_{j,k}\partial_{p_{\alpha\beta}}H_{\alpha\beta}u_{\alpha\beta}^q, \tag{5}$$

$$\dot{p}_{\alpha\beta}=\varphi\left(A^p-p_{\alpha\beta}-\omega_{\alpha\beta}^p p_{\alpha\beta}^2 q_{\alpha\beta}\right)+\sum_{j,k}\partial_{q_{\alpha\beta}}H_{\alpha\beta}u_{\alpha\beta}^p, \tag{6}$$

where superscripts (q,p) denote the corresponding Hamiltonian equations and the partial derivative is written as: $\partial_u H\equiv\partial H/\partial u$. In Equations ((5), (6)) the

[3]For simplicity, we are dealing with a square $(n\times n)$-matrix of UGV configurations. The whole approach works also for a rectangular $(n\times m)$-matrix, in which $\alpha=1,\cdots,n,\beta=1,\cdots,m$.

matrices $q_{\alpha\beta} = q_{\alpha\beta}(t)$ and $p_{\alpha\beta} = p_{\alpha\beta}(t)$ define time evolution of the swarm's coordinates and momenta, respectively, with initial conditions: $q_{\alpha\beta}(0)$ and $p_{\alpha\beta}(0)$. The field attractor lines A^q and A^p (with the field strength φ) are defined in the simulated urban environment using the A-*star* algorithm that finds the shortest Manhattan distance from point A to point B on the environmental digital map (*i.e.*, street directory). The adaptive synaptic weights matrices: $\omega^q_{\alpha\beta} = \omega^q_{\alpha\beta}(t)$ and $\omega^p_{\alpha\beta} = \omega^p_{\alpha\beta}(t)$ are trained by Hebbian learning (11) defined below. Input matrices $u^q_{\alpha\beta} = u^q_{\alpha\beta}(t)$ and $u^p_{\alpha\beta} = u^p_{\alpha\beta}(t)$ are Lie-derivative based controllers (explained below).

Equations ((5), (6)) are briefly derived as follows (for technical details, see [23] and the references therein). Given the swarm configuration manifold M (as a product of Euclidean groups of motion for each robotic vehicle), the affine Hamiltonian function $H_a(q,p,u): T_*M \to \mathbb{R}$ is defined on its cotangent bundle T_*M by (see [24]):

$$H_a(q,p,u) = H_0(q,p) - \sum_{j<n} H_j(q,p)\,u_j(t,q,p), \tag{7}$$

where $H_0(q,p): T_*M \to \mathbb{R}$ is the physical Hamiltonian function (kinetic plus potential energy of the swarm) and control inputs $u_j(t,q,p)$ are defined using recurrent Lie derivatives:[4]

$$u_j = \frac{\dot{o}_R^{(r)j} - L_f^{(r)} H_j + \sum_{s=1}^{r} c_{s-1}\left(o_R^{(s-1)j} - L_f^{(s-1)} H_j\right)}{L_g L_f^{(r-1)} H_j}. \tag{8}$$

Using (7) and (8), the affine Hamiltonian control system can be formulated (for $i = 1,\cdots,n; j < i$) as:

$$\dot{q}_i = V_i + \partial_{p_i} H_0 - \sum_{j<i} \partial_{p_i} H_j u_j + \partial_{p_i} D, \tag{9}$$

$$\dot{p}_i = F_i - \partial_{q_i} H_0 + \sum_{j<i} \partial_{q_i} H_j u_j + \partial_{q_i} D. \tag{10}$$

where D is Rayleigh's dissipative function and $V_i = V_i(t,q,p)$ and $F_i = F_i(t,q,p)$ are velocity and force controllers, respectively.

The affine Hamiltonian control system (9)-(10) can be reshaped into a matrix form suitable for a UGV-swarm or a planar formation of a UAV-swarm: $q_i \to q_{\alpha\beta}$, $p_i \to p_{\alpha\beta}$, $u_i \to u_{\alpha\beta}$, by evaluating dissipation terms into cubic terms in the brackets, and replacing velocity and force controllers with attractors A^q and A^p, respectively (with strength φ). If we add synaptic weights $\omega_{\alpha\beta}$ we come to our recurrent neural network Equations ((5), (6)).

Next, to make Equations ((5), (6)) adaptive, we use the abstract Hebbian learning scheme:

$$\text{New Value} = \text{Old Value} + \text{Innovation}$$

which in our settings formalizes as (see [25]):

[4] c_{s-1} are the coefficients of the linear ODE of order r for the error function $e(t) = q_j(t) - q_j^{\text{ref}}(t)$:

$$e^{(r)} + c_{r-1} e^{(r-1)} + \cdots + c_1 e^{(1)} + c_0 e = 0.$$

$$\dot{\omega}^q_{\alpha\beta} = -\omega^q_{\alpha\beta} + \Phi^q_{\alpha\beta}\left(q_{\alpha\beta}, p_{\alpha\beta}\right), \tag{11}$$
$$\dot{\omega}^p_{\alpha\beta} = -\omega^p_{\alpha\beta} + \Phi^p_{\alpha\beta}\left(q_{\alpha\beta}, p_{\alpha\beta}\right),$$

where the innovations are given in the matrix signal form (generalized from [26]):

$$\Phi^q_{\alpha\beta} = S^q_{\alpha\beta}\left(q_{\alpha\beta}\right)S^q_{\alpha\beta}\left(p_{\alpha\beta}\right) + \dot{S}^q_{\alpha\beta}\left(q_{\alpha\beta}\right)\dot{S}^q_{\alpha\beta}\left(p_{\alpha\beta}\right), \tag{12}$$
$$\Phi^p_{\alpha\beta} = S^p_{\alpha\beta}\left(q_{\alpha\beta}\right)S^p_{\alpha\beta}\left(p_{\alpha\beta}\right) + \dot{S}^p_{\alpha\beta}\left(q_{\alpha\beta}\right)\dot{S}^p_{\alpha\beta}\left(p_{\alpha\beta}\right),$$

with sigmoid activation functions $S(\cdot) = \tanh(\cdot)$ and signal velocities: $\dot{S}(\cdot) = 1 - \tanh(\cdot)$.

In this way defined affine Hamiltonian control system (5)-(6), which is also a recurrent neural network with Hebbian leaning (11)-(12), represents our nonlinear, adaptive and high-dimensional generalization of Kalman's linear state dynamics (1) and control (2)-(3).

Using Wolfram Mathematica® code given below, the Hamiltonian neural net was simulated for 1 sec, with the matrix dimensions $n = 10$ (*i.e.*, with 100 neurons in both matrices q_{ij} and p_{ij}, which are longitudes and latitudes of a large fleet of 100 UGVs, see **Figures 2-4**). The Figures illustrate both stability and convergence of the recurrent Hamiltonian neural net.

C++ code for the the Hamiltonian neural net is given in **Appendix 1.1**.

Mathematica® code

Dimensions:

$$n = 10; \mathrm{Tfin} = 1; \varphi = 30;$$

Coordinates and momenta:

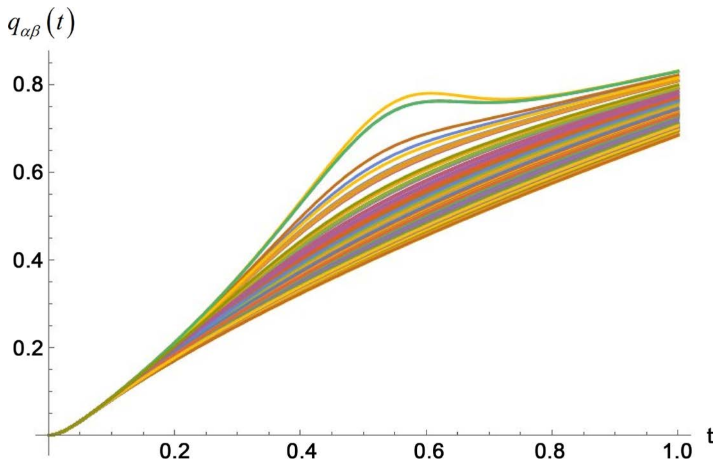

Figure 2. Adaptive control for a large fleet (a 10×10 matrix) of 100 UGVs: time evolution of coordinates $q_{\alpha\beta}(t)$, where $\alpha, \beta = 1, \cdots, 10$. As expected, we can see the exponential-type growth of coordinates $q_{\alpha\beta}(t)$ together with their spreading from zero initial conditions. This plot is the simulation output of the recurrent activation dynamics given by Equations ((5), (6)) with Hebbian learning dynamics given by Equations ((11), (12)), starting from zero initial coordinates and momenta and random $(-1,1)$ initial weights.

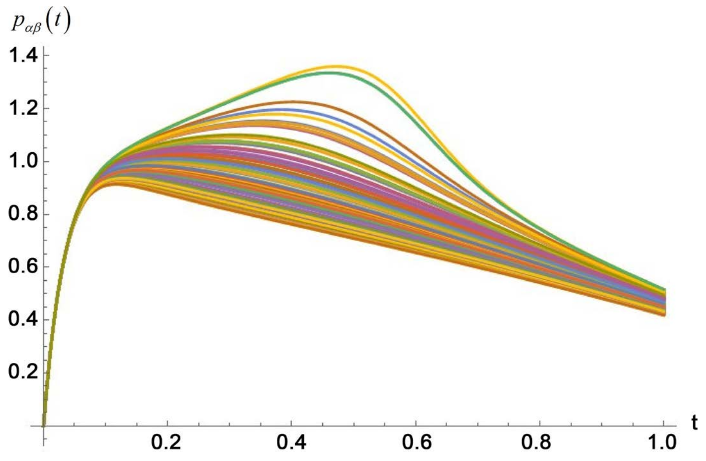

Figure 3. Adaptive control for a large fleet (a 10×10 matrix) of 100 UGVs: time evolution of momenta $p_{\alpha\beta}(t)$. As expected, we can see the exponential-type decay of momenta $p_{\alpha\beta}(t)$ together with their spreading, after the initial convergent growth from zero initial conditions. This plot is the simulation output from the same recurrent neural network as in **Figure 2**.

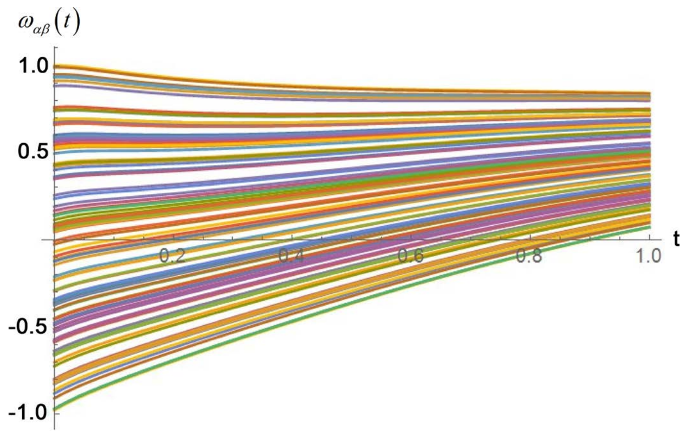

Figure 4. Adaptive control for a large fleet (a 10×10 matrix) of 100 UGVs: time evolution of synaptic weights $\omega_{\alpha\beta}(t)$. As expected, we can see the converging behavior of the weights $\omega_{\alpha\beta}(t)$, starting from their initial random distribution. Simulating the same recurrent neural network from **Figure 2**.

$$\text{Table}\left[q_{\alpha,\beta}[t],\{\alpha,n\},\{\beta,n\}\right];$$
$$\text{Table}\left[p_{\alpha,\beta}[t],\{\alpha,n\},\{\beta,n\}\right];$$

Hyperbolic tangent activation functions:

$$\text{Table}\left[\text{Sq}_{\alpha,\beta}=\text{Tanh}\left[q_{\alpha,\beta}[t]\right],\{\alpha,n\},\{\beta,n\}\right];$$
$$\text{Table}\left[\text{Sp}_{\alpha,\beta}=\text{Tanh}\left[p_{\alpha,\beta}[t]\right],\{\alpha,n\},\{\beta,n\}\right];$$

Derivatives of activation functions:

$$\text{Table}\left[\text{Sdq}_{\alpha,\beta}=1-\text{Sq}^2_{\alpha,\beta},\{\alpha,n\},\{\beta,n\}\right];$$
$$\text{Table}\left[\text{Sdp}_{\alpha,\beta}=1-\text{Sp}^2_{\alpha,\beta},\{\alpha,n\},\{\beta,n\}\right];$$

Control inputs:

$$\text{Table}\left[a_{\alpha,\beta} = \text{RandomReal}\left[\{-1,1\}\right],\{\alpha,n\},\{\beta,n\}\right];$$

$$\text{Table}\left[b_{\alpha,\beta} = \text{RandomReal}\left[\{-1,1\}\right],\{\alpha,n\},\{\beta,n\}\right];$$

$$\text{Table}\left[c_{\alpha,\beta} = \text{RandomReal}\left[\{-1,1\}\right],\{\alpha,n\},\{\beta,n\}\right];$$

$$\text{Table}\left[u_{\alpha,\beta}[t] = a_{\alpha,\beta}\text{Sin}\left[b_{\alpha,\beta} + c_{\alpha,\beta}t\right],\{\alpha,n\},\{\beta,n\}\right];$$

Affine Hamiltonians (nearest-neighbor coupling):

$$\text{Table}\left[H_{\alpha,\beta} = \frac{1}{2}\left(q_{\alpha,\beta}[t]q_{\alpha-1,\beta-1}[t] + p_{\alpha,\beta}[t]p_{\alpha-1,\beta-1}[t]\right),\{\alpha,2,n\},\{\beta,2,n\}\right];$$

Attractor lines (figure-8 attractor):

$$\text{Aq} = \text{Sin}[t];\text{Ap} = \text{Cos}[t];$$

Equations of motion:

$$\text{Eqns} = \text{Flatten}\Big[\text{Join}\Big[$$

$$\text{Table}\Big[\Big\{q'_{\alpha,\beta}[t] == \varphi\left(\text{Aq} - q_{\alpha,\beta}[t] - \omega_{\alpha,\beta}[t]q_{\alpha,\beta}[t]^2\, p_{\alpha,\beta}[t]\right)$$

$$-\sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}\partial_{p_{\alpha,\beta}[t]}H_{\alpha,\beta}u_{\alpha,\beta}[t], q_{\alpha,\beta}[0] == 0\Big\},\{\alpha,n\},\{\beta,n\}\Big],$$

$$\text{Table}\Big[\Big\{p'_{\alpha,\beta}[t] == \varphi\left(\text{Ap} - p_{\alpha,\beta}[t] - \omega_{\alpha,\beta}[t]p_{\alpha,\beta}[t]^2\, q_{\alpha,\beta}[t]\right)$$

$$+\sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}\partial_{q_{\alpha,\beta}[t]}H_{\alpha,\beta}u_{\alpha,\beta}[t], p_{\alpha,\beta}[0] == 0\Big\},\{\alpha,n\},\{\beta,n\}\Big],$$

$$\text{Table}\Big[\Big\{\omega'_{\alpha,\beta}[t] == -\omega_{\alpha,\beta}[t] + \text{Sq}_{\alpha,\beta}\text{Sp}_{\alpha,\beta} + \text{Sdq}_{\alpha,\beta}\text{Sdp}_{\alpha,\beta},$$

$$\omega_{\alpha,\beta}[0] == \text{RandomReal}\left[\{-1,1\}\right]\Big\},\{\alpha,n\},\{\beta,n\}\Big]\Big]\Big];$$

Numerical solution:

$$\text{sol} = \text{NDSolve}\Big[\text{Eqns},\text{Flatten}\Big[\text{Join}\Big[\text{Table}\left[q_{\alpha,\beta},\{\alpha,n\},\{\beta,n\}\right],$$

$$\text{Table}\left[p_{\alpha,\beta},\{\alpha,n\},\{\beta,n\}\right],\text{Table}\left[\omega_{\alpha,\beta},\{\alpha,n\},\{\beta,n\}\right]\Big]\Big],\{t,0,\text{Tfin}\}\Big];$$

Plots of coordinates:

$$\text{Plot}\Big[\text{Evaluate}\Big[\text{Table}\left[q_{\alpha,\beta}[t]/.\text{sol},\{\alpha,n\},\{\beta,n\}\right],\{t,0,\text{Tfin}\}\Big],$$

$$\text{PlotRange} \to \text{All},\text{PlotStyle} \to \text{AbsoluteThickness}[1.5],\text{Frame} \to \text{True}\Big]$$

Plots of momenta:

$$\text{Plot}\Big[\text{Evaluate}\Big[\text{Table}\left[p_{\alpha,\beta}[t]/.\text{sol},\{\alpha,n\},\{\beta,n\}\right],\{t,0,\text{Tfin}\}\Big],$$

$$\text{PlotRange} \to \text{All},\text{PlotStyle} \to \text{AbsoluteThickness}[1.5],\text{Frame} \to \text{True}\Big]$$

Plots of weights:

$$\text{Plot}\Big[\text{Evaluate}\Big[\text{Table}\left[\omega_{\alpha,\beta}[t]/.\text{sol},\{\alpha,n\},\{\beta,n\}\right],\{t,0,\text{Tfin}\}\Big],$$

$$\text{PlotRange} \to \text{All},\text{PlotStyle} \to \text{AbsoluteThickness}[1.5],\text{Frame} \to \text{True}\Big]$$

4. Estimation of an Autonomous Team of UGVs: High-Dimensional Bayesian Particle Filter

The second step in the nonlinear generalization of the Kalman servo is to replace the linear/Gaussian Kalman filter (4) with a general nonlinear/non-Gaussian Monte-Carlo filter, as follows.

4.1. Recursive Bayesian Filter

Recall that the celebrated *Bayes rule* gives the relation between the three basic *conditional probabilities*: i) a Prior $P(A)$ (*i.e.*, an initial degree-of-belief in event *A*, that is, Initial Odds), ii) Likelihood $P(B|A)$ (*i.e.*, a degree-of-belief in event *B*, given *A*, that is, a New Evidence); and Posterior: $P(A|B)$ (*i.e.*, a degree-of-belief in *A*, given *B*, that is, New Odds). Provided $P(B)\neq 0$, Bayes rule reads:

$$\overset{\text{Posterior}}{P(A|B)}=\frac{\overset{\text{Prior}}{P(A)}\times\overset{\text{Likelihood}}{P(B|A)}}{P(B)}. \tag{13}$$

In statistical settings, Bayes rule (13) can be rewritten as:

$$\overset{\text{Posterior}}{p(H|D)}=\frac{\overset{\text{Prior}}{p(H)}\times\overset{\text{Likelihood}}{p(D|H)}}{p(D)}, \tag{14}$$

where $p(H)$ is the *prior* probability density function (PDF) that the hypothesis *H* is true, $p(D|H)$ is the *likelihood* PDF for the data *D* given a hypothesis *H*, $p(H|D)$ is the *posterior* PDF that the hypothesis *H* is true given the data *D*, and the normalization constant: $p(D)=\sum_{H_i}p(D|H)p(H)$ is the PDF for the data *D* averaged over all possible hypotheses H_i.

When Bayes rule (13)-(14) is applied iteratively/recursively over signal distributions evolving in discrete time steps, in such a way that the Old Posterior becomes the New Prior and New Evidence is added, it becomes the *recursive Bayesian filter*, the formal origin of all Kalman and particle filters. The Bayesian filter can be applied to estimate the hidden state x_t of a nonlinear dynamical system evolving in discrete time steps (e.g., a numerical solution of a system of ODEs, or discrete-time sampling measurements) in a recursive manner by processing a sequence of observations $Y_T=\{y_t\}_{t=1}^T$ dependent on the state x_t within a dynamic noise (either Gaussian, or non-Gaussian) η_t measured as μ_t. The state dynamics and measurement, or the so-called *state-space model* (SSM) are usually given either by Kalman's state Equation (1), or by its nonlinear generalization. Instead of the Kalman filter Equation (4) combined with the cost function (3), the Bayesian filter includes (see, e.g. [27] [28]):

Time-update equation:

$$\overset{\text{Predictor}}{p(x_t|Y_{t-1})}=\int_{\mathbb{R}^n}\overset{\text{Prior}}{p(x_t|x_{t-1})}\,\overset{\text{Old Posterior}}{p(x_{t-1}|Y_{t-1})}\mathrm{d}x_{t-1}, \tag{15}$$

and

Measurement-update equation:

$$\overset{\text{New Posterior}}{p(x_t \mid Y_t)} = Z_t^{(-1)} \overset{\text{Predictor}}{p(x_t \mid Y_{t-1})} \overset{\text{Likelihood}}{l(y_t \mid x_t)}, \tag{16}$$

where Z_t is the normalizing constant, or partition function, defined by:

$$Z_t = \int_{\mathbb{R}^n} p(x_t \mid Y_{t-1}) l(y_t \mid x_t) \mathrm{d}x_t.$$

In the Bayesian filter (15)-(16), given the sequence of observations Y_t, the posterior fully defines the available information about the state of the system and the noisy environment—the filter propagates the posterior (embodying time and measurement updates for each iteration) across the nonlinear state-space model; therefore, it is the *maximum a posteriori* estimator of the system state. In a special case of a linear system and Gaussian environment, the filter (15)-(16) has optimal closed-form solution, which is the Kalman filter (4). However, in the general case of a nonlinear dynamics and/or the non-Gaussian environment, it is no longer feasible to search for closed-form solutions for the integrals in (15)-(16), so we are left with suboptimal, approximate, Monte Carlo solutions, called particle filters.

4.2. Sequential Monte Carlo Particle Filter

Now we consider a general, discrete-time, nonlinear, probabilistic SSM, defined by a *Markov process* $\{x_t\}_{t\geq 1} \subseteq \mathbb{R}^{n_x}$, which is observed indirectly via a *measurement process* $\{y_t\}_{t\geq 1} \subseteq \mathbb{R}^{n_y}$. This SSM consists of two probability density functions: dynamics $f(\cdot)$ and measurement $h(\cdot)$. Formally, we have a system of nonlinear difference equations, given both in Bayesian probabilistic formulation (left-hand side) and nonlinear state-space formulation (right-hand side):

$$\text{Dynamics}: \; x_{t+1} \mid x_t \sim f(x_{t+1} \mid x_t) \Leftrightarrow x_{t+1} = f(x_t) + \epsilon_t, \tag{17}$$

$$\text{Measurement}: y_t \mid x_t \sim h(y_t \mid x_t) \Leftrightarrow y_t = h(x_t) + \nu_t, \tag{18}$$

$$\text{with initial state}: \; x_1 \sim \mu(x_1),$$

where f and h are the state and output vector-functions, ϵ_t and ν_t are non-Gaussian process and measurement noises and μ is a given non-Gaussian distribution. The *state filtering problem* means to recover information about the current state x_t in (17) based on the available measurements $y_{1:t}$ in (18) (see, e.g. [29]; for a recent review, see [30] and the references therein).

The principle solution to the nonlinear/non-Gaussian state filtering problem is provided by the following recursive Bayesian measurement and time update equations:

$$\overset{\text{New Filter}}{p(x_t \mid y_{1:t})} = \frac{\overset{\text{Measure}}{h(y_t \mid x_t)} \overset{\text{Predictor}}{p(x_t \mid y_{1:t-1})}}{p(y_t \mid y_{1:t-1})}, \tag{19}$$

where

$$\overset{\text{Predictor}}{p(x_t \mid y_{1:t-1})} = \int \overset{\text{Dynamics}}{f(x_t \mid x_{t-1})} \overset{\text{Old Filter}}{p(x_{t-1} \mid y_{1:t-1})} \mathrm{d}x_{t-1}. \tag{20}$$

In a particular case of linear/Gaussian SSMs (17)-(18), the state filtering problem (19)-(20) has an optimal closed-form solution for the *filter PDF* $p(x_t \mid y_{1:t})$, given by the Kalman filter (4). However, in the general case of nonlinear/non-Gaussian SSMs—such an optimal closed-form solution does not exist.

The general state filtering problem (19)-(20) associated with any nonlinear/non-Gaussian SSMs (17)-(18) can be only numerically approximated using the *sequential Monte Carlo* (SMC) methods. The SMC-approximation to the filter PDF, denoted by $\hat{p}(x_t \mid y_{1:t})$, is an empirical distribution represented as a weighted sum of Dirac-delta functions:

$$\hat{p}(x_t \mid y_{1:t}) = \sum_{i=1}^{N} w_t^i \delta_{x_t^i}(x_t), \tag{21}$$

where the samples $\{x_t^i\}_{i=1}^N$ are called *particles* (point-masses 'spread out' in the state space); each particle x_t^i represents one possible system state and the corresponding weight w_t^i assigns its probability. In this way defined *particle filter* (PF) plays the role of the Kalman filter for nonlinear/non-Gaussian SSMs. PF approximates the filter PDF $p(x_t \mid y_{1:t})$ using the SMC delta-approximation (21).

Briefly, a PF can be interpreted as a sequential application of the SMC technique called *importance sampling* (IS, see e.g. [31]). Starting from the initial approximation: $\hat{p}(x_1 \mid y_1) \propto h(y_1 \mid x_1)\mu(x_1)$, at each time step the IS is used to approximate the filter PDF $p(x_t \mid y_{1:t})$ by using the recursive Bayesian Equations ((19), (20)) together with the already generated IS approximation of $p(x_{t-1} \mid y_{1:t-1})$. The particles $\{x_1^i\}_{i=1}^N$ are sampled independently from some proposal distribution $r(x_1)$. To account for the discrepancy between the proposal distribution and the target distribution, the particles are assigned importance weights, given by the ratio between the target and the proposal: $w_1^i \propto h(y_1 \mid x_1^i)\mu(x_1^i)/r(x_1^i)$, where the weights are normalized to sum to one (for a recent technical review, see e.g., [30]).

The IS proceeds in an inductive fashion:

$$\hat{p}(x_{t-1} \mid y_{1:t-1}) = \sum_{i=1}^{N} w_{t-1}^i \delta_{x_{t-1}^i}(x_{t-1}),$$

which, inserted as an old/previous filter into the time-update Equation (20), gives a *mixture distribution,* approximating $p(x_t \mid y_{1:t-1})$ as:

$$\hat{p}(x_t \mid y_{1:t-1}) = \int \overset{\text{Dynamics}}{f(x_t \mid x_{t-1})} \sum_{i=1}^{N} w_{t-1}^i \delta_{x_{t-1}^i}(x_{t-1}) \mathrm{d}x_{t-1} = \sum_{i=1}^{N} w_{t-1}^i f(x_t \mid x_{t-1}^i). \tag{22}$$

Subsequent insertion of (22) as a predictor into the measurement-update Equation (19), gives the following approximation of the filter PDF:

$$\overset{\text{Filter}}{p(x_t \mid y_{1:t})} \approx \frac{\overset{\text{Measure}}{h(y_t \mid x_t)}}{p(y_t \mid y_{1:t-1})} \sum_{i=1}^{N} \overset{\text{Weighted Dynamics}}{w_{t-1}^i f(x_t \mid x_{t-1}^i)},$$

which needs to be further approximated using the IS. The *proposal density* is pragmatically chosen as: $r(x_t \mid y_{1:t}) = \sum_{j=1}^{N} w_{t-1}^j f(x_t \mid x_{t-1}^j)$. The N ancestor indices $\{a_t^i\}_{i=1}^N$ are *resampled* into a new set of particles $\{a_{t-1}^i\}_{i=1}^N$ which is subsequently used to propagate the particles to time *t*.

The final step is to assign the importance weights to the new particles as:

$$\bar{w}_t^i = \frac{h(y_t \mid x_t^i) \sum_{j=1}^{N} w_{t-1}^j f(x_t^i \mid x_{t-1}^j)}{\sum_{j=1}^{N} w_{t-1}^j f(x_t \mid x_{t-1}^j)}.$$

By evaluating $\bar{w}_t^i$ for $i = 1, \cdots, N$ and normalizing the weights, we obtain a new set of weighted particles $\{x_t^i, w_t^i\}_{i=1}^N$, constituting an empirical approximation of the filter PDF $p(x_t \mid y_{1:t})$. Since these weighted particles $\{x_t^i, w_t^i\}_{i=1}^N$ can be used to iteratively approximate the filter PDF at all future times, this completes the PF-algorithm with an overall computational complexity of $O(N)$ (as pioneered by [31]; for more technical details, see e.g., [30] and the references therein).

We remark that the pinnacle of the PF-theory is the so-called Rao-Blackwellized (or, marginalized) PF, which is a special kind of factored PF, where the state-space dynamics (17) is split into a linear/Gaussian part and a nonlinear/non-Gaussian part, so that each particle has the optimal linear-Gaussian Kalman filter associated to it (see [32]-[40]). However, in our case of highly nonlinear and adaptive dynamics, the use of Rao-Blackwellization does not give any advantage, while its mathematics is significantly heavier than in an ordinary PF.

5. Hamiltonian Servo Framework for General Control and Estimation

The Hamiltonian servo system is our *hybrid*[5] global control-estimation framework for a large team/fleet of UGVs (e.g., $10 \times 10 = 100$), consisting of the following four components:

Adaptive Control, which is nonlinear and high-dimensional, defined by:

$$\dot{q}_{\alpha\beta} = \varphi\left(A^q - q_{\alpha\beta} - \omega_{\alpha\beta}^q q_{\alpha\beta}^2 p_{\alpha\beta}\right) - \sum_{j,k} \partial_{p_{\alpha\beta}} H_{\alpha\beta} u_{\alpha\beta}^q, \tag{23}$$

$$\dot{p}_{\alpha\beta} = \varphi\left(A^p - p_{\alpha\beta} - \omega_{\alpha\beta}^p p_{\alpha\beta}^2 q_{\alpha\beta}\right) + \sum_{j,k} \partial_{q_{\alpha\beta}} H_{\alpha\beta} u_{\alpha\beta}^p,$$

$$\dot{\omega}_{\alpha\beta}^q = -\omega_{\alpha\beta}^q + \Phi_{\alpha\beta}^q\left(q_{\alpha\beta}, p_{\alpha\beta}\right), \quad \dot{\omega}_{\alpha\beta}^p = -\omega_{\alpha\beta}^p + \Phi_{\alpha\beta}^p\left(q_{\alpha\beta}, p_{\alpha\beta}\right), \tag{24}$$

$$\Phi_{\alpha\beta}^q = S_{\alpha\beta}^q\left(q_{\alpha\beta}\right) S_{\alpha\beta}^q\left(p_{\alpha\beta}\right) + \dot{S}_{\alpha\beta}^q\left(q_{\alpha\beta}\right) \dot{S}_{\alpha\beta}^q\left(p_{\alpha\beta}\right), \ (\alpha, \beta = 1, \cdots, n)$$

$$\Phi_{\alpha\beta}^p = S_{\alpha\beta}^p\left(q_{\alpha\beta}\right) S_{\alpha\beta}^p\left(p_{\alpha\beta}\right) + \dot{S}_{\alpha\beta}^p\left(q_{\alpha\beta}\right) \dot{S}_{\alpha\beta}^p\left(p_{\alpha\beta}\right).$$

Measurement process, given in discrete-time, probabilistic and state-space Hamiltonian (q, p)-form:

$$p_t^{\alpha\beta} \mid q_t^{\alpha\beta} \sim h\left(p_t^{\alpha\beta} \mid q_t^{\alpha\beta}\right) \Leftrightarrow p_t^{\alpha\beta} = h\left(q_t^{\alpha\beta}\right) + \nu_t, \ \text{with: } q_1^{\alpha\beta} \sim \mu\left(q_1^{\alpha\beta}\right), \tag{25}$$

[5]Half continuous-time, half discrete-time.

where v_t is a non-Gaussian measurement noise. Note that here we use the *inverse dynamics* (given coordinates, calculate velocities, accelerations and forces)—as a bridge between robot's self-localization and control: measuring coordinates $q_t^{\alpha\beta}$ at discrete time steps t and from them calculating momenta $p_t^{\alpha\beta}$ and control forces $\dot{p}_t^{\alpha\beta}$;

Bayesian recursions, given in discrete-time Hamiltonian (q, p)-form:

$$\overset{\text{New Filter}}{p\left(q_t^{\alpha\beta} \mid p_{1:t}^{\alpha\beta}\right)} = \frac{\overset{\text{Measure}}{h\left(p_t^{\alpha\beta} \mid q_t^{\alpha\beta}\right)}\overset{\text{Predictor}}{p\left(q_t^{\alpha\beta} \mid p_{1:t-1}^{\alpha\beta}\right)}}{p\left(p_t^{\alpha\beta} \mid p_{1:t-1}^{\alpha\beta}\right)},$$

where

$$\overset{\text{Predictor}}{p\left(q_t^{\alpha\beta} \mid p_{1:t-1}^{\alpha\beta}\right)} = \int \overset{\text{Dynamics}}{f\left(q_t^{\alpha\beta} \mid q_{t-1}^{\alpha\beta}\right)}\overset{\text{Old Filter}}{p\left(q_{t-1}^{\alpha\beta} \mid p_{1:t-1}^{\alpha\beta}\right)}\mathrm{d}x_{t-1}; \tag{26}$$

and

$$\overset{\text{Filter PDF}}{p\left(q_t^{\alpha\beta} \mid p_{1:t}^{\alpha\beta}\right)} \approx \frac{\overset{\text{Measure}}{h\left(p_t^{\alpha\beta} \mid q_t^{\alpha\beta}\right)}}{p\left(p_t^{\alpha\beta} \mid p_{1:t-1}^{\alpha\beta}\right)} \overset{\text{Weighted Dynamics}}{\sum_{(i)=1}^{N} w_{t-1}^{(i)} f\left(q_t^{\alpha\beta} \mid q_{t-1}^{\alpha\beta(i)}\right)}, \tag{27}$$

with IS weights:

$$w_t^{(i)} = \frac{h\left(p_t^{\alpha\beta} \mid q_t^{\alpha\beta(i)}\right)\sum_{n=1}^{N} w_{t-1}^{(n)} f\left(q_t^{\alpha\beta(i)} \mid q_{t-1}^{\alpha\beta(n)}\right)}{\sum_{n=1}^{N} w_{t-1}^{(n)} f\left(q_t^{\alpha\beta} \mid q_{t-1}^{\alpha\beta(n)}\right)},$$

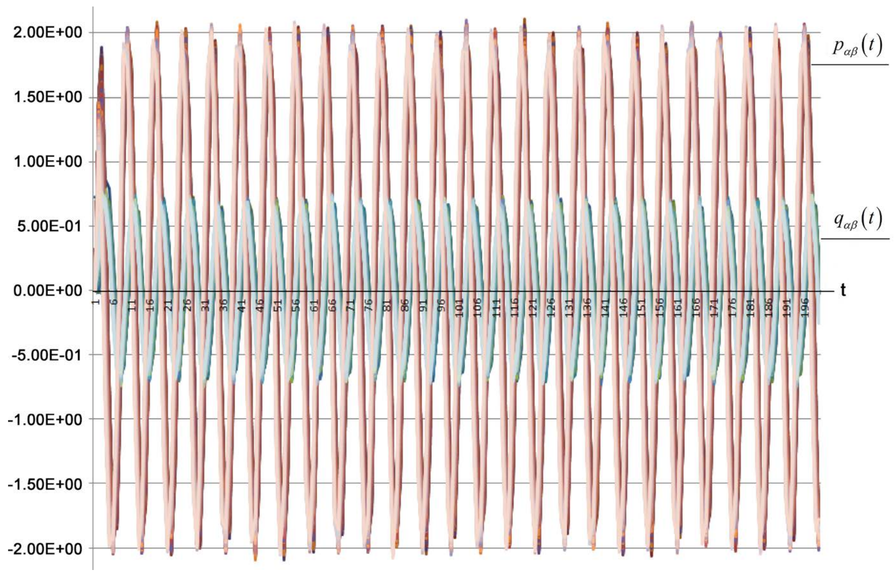

Figure 5. Sample output of the Hamiltonian servo framework applied to a large fleet (a 9×9 matrix) of 81 UGVs. As expected, the plot shows almost periodic time evolution of coordinates $q_{\alpha\beta}(t)$ and momenta $p_{\alpha\beta}(t)$. Slight variation of both coordinates $q_{\alpha\beta}(t)$ and momenta $p_{\alpha\beta}(t)$ from the intended regular harmonic motion is due to adaptive control with initial random distribution of synaptic weights $\omega_{\alpha\beta}(t)$. This plot is the simulation output of the Hamiltonian neural network (23)-(24) and recurrent estimation (larger amplitudes, governed by the Bayesian particle filter (25)-(27)).

where bracketed superscripts (i,n) label IS weights.

This Hamiltonian servo framework has been implemented in C++ language (using Visual Studio 2015; see **Appendix 1.2**). A sample simulation output of the servo system is given in **Figure 5**. The Figure demonstrates the basic stability and convergence of the servo system.

6. Conclusion

In this paper we have outlined a novel control and estimation framework called the Hamiltonian Servo system. The framework was shown to be a generalization of the Kalman Servo System. Using an example of a large team/fleet of 81 (9×9) unmanned ground vehicles (UGVs), it was shown that the framework described provided the control and estimation as required. This approach encompassed a three tired system of adaptive control, measurement process and Bayesian recursive filtering which was demonstrated to enable control and estimation of multidimensional, non-linear and non-Gaussian systems.

Acknowledgements

The authors are grateful to Dr. Martin Oxenham, JOAD, DST Group, Australia for his constructive comments which have improved the quality of this paper.

References

[1] Hui, K.-P. and Pourbeik, P. (2011) OPAL—A Survivability-Oriented Approach to Management of Tactical Military Networks. *IEEE MILCOM*, 7-10 November 2011.

[2] Nicholas, P., *et al.* (2016) Measuring the Operational Impact of Military SATCOM Degradation. *Winter Simulation Conference* (*WSC*), 11-14 December 2016.

[3] Jormakha, J., *et al.* (2011) UAV-Based Sensor Networks for Future Force Warriors. *International Journal of Advances in Telecommunications*, **4**, 58-71.

[4] Offensive Swarm-Enabled Tactics (OFFSET). https://www.grants.gov/web/grants/view-opportunity.html?oppId=291825

[5] Vukobratovic, M. and Potkonjak, V. (1982) Scientific Fundamentals of Robotics, Vol. 1, Dynamics of Manipulation Robots: Theory and Application. Springer, Berlin.

[6] Vukobratovic, M. and Stokic, D. (1982) Scientific Fundamentals of Robotics, Vol. 2, Control of Manipulation Robots: Theory and Application. Springer, Berlin.

[7] Vukobratovic, M. and Kircanski, M. (1985) Scientific Fundamentals of Robotics, Vol. 3, Kinematics and Trajectories Synthesis of Manipulation Robots. Springer, Berlin.

[8] Vukobratovic, M. and Kircanski, N. (1985) Scientific Fundamentals of Robotics, Vol. 4, Real-Time Dynamics of Manipulation Robots. Springer, Berlin.

[9] Vukobratovic, M., Stokic, D. and Kircanski, N. (1985) Scientific Fundamentals of Robotics, Vol. 5, Non-Adaptive and Adaptive Control of Manipulation Robots. Springer, Berlin.

[10] Vukobratovic, M. and Potkonjak, V. (1985) Scientific Fundamentals of Robotics, Vol. 6, Applied Dynamics and CAD of Manipulation Robots. Springer, Berlin.

[11] Ivancevic, V. and Ivancevic, T. (2006) Human-Like Biomechanics. Springer, Dor-

drecht.

[12] ASIMO (2016) The Honda Humanoid Robot. http://world.honda.com/ASIMO/

[13] Ivancevic, V.G. (2010) Nonlinear Complexity of Human Biodynamics Engine, Nonlin. *Dynamics*, **61**, 123-139.

[14] Durrant-Whyte, H. and Bailey, T. (2006) Simultaneous Localization and Mapping (SLAM): Part I. *IEEE Robotics & Automation Magazine*, **13**, 99-110. https://doi.org/10.1109/MRA.2006.1638022

[15] Wikipedia (2016) Simultaneous Localization and Mapping.

[16] Montemerlo, M., Thrun, S., Koller, D. and Wegbreit, B. (2002) FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem.

[17] Montemerlo, M., Thrun, S., Koller, D. and Wegbreit, B. (2003) FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping That Provably Converges. *Proceedings of IJCAI*, 1151-1156.

[18] Thrun, S., Montemerlo, M., Koller, D., Wegbreit, B., Nieto, J. and Nebot, E. (2004) FastSLAM: An Efficient Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association. *Journal of Machine Learning Research*, 1-48.

[19] OpenSLAM (2016). https://www.openslam.org/

[20] Kalman, R.E. (1960) A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, **82**, 34-45. https://doi.org/10.1115/1.3662552

[21] Kalman, R.E. and Bucy, R.S. (1961) New Results in Linear Filtering and Prediction Theory. *Journal of Basic Engineering*, **96**, 95-108. https://doi.org/10.1115/1.3658902

[22] Kalman, R.E., Falb, P. and Arbib, M.A. (1969) Topics in Mathematical System Theory. McGraw Hill, New York.

[23] Ivancevic, V. and Yue, Y. (2016) Hamiltonian Dynamics and Control of a Joint Autonomous Land-Air Operation. *Nonlinear Dynamics*, **84**, 1853-1865. https://doi.org/10.1007/s11071-016-2610-y

[24] Ivancevic, V. and Ivancevic, T. (2006) Geometrical Dynamics of Complex Systems. Springer, Dordrecht. https://doi.org/10.1007/1-4020-4545-X

[25] Ivancevic, V. and Ivancevic, T. (2007) Neuro-Fuzzy Associative Machinery for Comprehensive Brain and Cognition Modelling. Springer, Berlin. https://doi.org/10.1007/978-3-540-48396-0

[26] Kosko, B. (1992) Neural Networks and Fuzzy Systems, A Dynamical Systems Approach to Machine Intelligence. Prentice-Hall, New York.

[27] Arulampalam, S., Maskell, S., Gordon, N. and Clapp, T. (2002) A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Transactions on Signal Processing*, **50**, 174-188. https://doi.org/10.1109/78.978374

[28] Ristic, B., Arulampalam, S. and Gordon, N. (2004) Beyond the Kalman Filter. Artech House.

[29] Schön, T.B., Gustafsson, F. and Nordlund, P.-J. (2005) Marginalized Particle Filters for Mixed Linear/Nonlinear State-Space Models. *IEEE Transactions on Signal Processing*, **53**, 2279-2289. https://doi.org/10.1109/TSP.2005.849151

[30] Schön, T.B., Lindsten, F., Dahlin, J., *et al.* (2015) Sequential Monte Carlo Methods for System Identification. *IFAC-PapersOnLine*, **48**, 775-786.

[31] Gordon, N.J., Salmond, D.J. and Smith, A.F.M. (1993) Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation. *I Radar and Signal Processing*, **140**, 107-113.

[32] Casella, G. and Robert, C.P. (1996) Rao-Blackwellisation of Sampling Schemes. *Biometrika*, **83**, 81-94. https://doi.org/10.1093/biomet/83.1.81

[33] Chen, R. and Liu, J.S. (2000) Mixture Kalman Filters. *Journal of the Royal Statistical Society*, **62**, 493-508. https://doi.org/10.1111/1467-9868.00246

[34] Doucet, A., Godsill, S.J. and Andrieu, C. (2000) On Sequential Monte Carlo Sampling Methods for Bayesian Filtering. *Statistics and Computing*, **10**, 197-208. https://doi.org/10.1023/A:1008935410038

[35] Doucet, A., Gordon, N. and Krishnamurthy, V. (2001) Particle Filters for State Estimation of Jump Markov Linear Systems. *IEEE Transactions on Signal Processing*, **49**, 613-624. https://doi.org/10.1109/78.905890

[36] Andrieu, C. and Doucet, A. (2002) Particle Filtering for Partially Observed Gaussian State Space Models. *Journal of the Royal Statistical Society*, **64**, 827-836. https://doi.org/10.1111/1467-9868.00363

[37] Schön, T., Gustafsson, F. and Nordlund, P.-J. (2003) Marginalized Particle Filters for Nonlinear State-Space Models. Tec. Report LiTH-ISY-R-2548, Linköping Univ.

[38] Schön, T., Gustafsson, F. and Nordlund, P.-J. (2005) Marginalized Particle Filters for Mixed Linear/Nonlinear State-Space Models. *IEEE Transactions on Signal Processing*, **53**, 2279-2289. https://doi.org/10.1109/TSP.2005.849151

[39] Karlsson, R., Schön, T. and Gustafsson, F. (2005) Complexity Analysis of the Marginalized Particle Filter. *IEEE Transactions on Signal Processing*, **53**, 4408-4411. https://doi.org/10.1109/TSP.2005.857061

[40] Schön, T., Karlsson, R. and Gustafsson, F. (2006) The Marginalized Particle Filter in Practice. *Aerospace Conference*, 4-11 March 2006. https://doi.org/10.1109/AERO.2006.1655922

[41] Wikipedia (2016) Armadillo (C++ Library).

Appendix: C++ Code

1.1. Affine Hamiltonian Neural Network

```
// Affine Hamiltonian neural network
// Using Runge-Kutta-4 fixed ODE integrator
// Simulating UGV-swarm

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

// attractor: (Aq(t),Ap(t))
#define Aq(t) ( sin(t) )
#define Ap(t) ( 3.*cos(t) )
#define M 2
#define N 10

double k1[M], f1[M], k2[M], f2[M], k3[M], f3[M], k4[M];

double h = 0.001;           // Time step
double t = 0.;              // Initial time
double Tfin = 30.;          // Final time
double fld = 50.;
double eta = 0.5;
double q[N][N], p[N][N], w[N][N], qc[2], wt[3];
double ranf();              // calling random generator

// Affine Hamiltonian equations
void Eqs(double wt[], double t, double y[], double dy[]) {
dy[0] = fld*(Aq(t) - y[0] - wt[0] * y[0] * y[1] * y[0] - y[0] * wt[1]);
dy[1] = fld*(Ap(t) - y[1] - wt[0] * y[1] * y[0] * y[1] - y[1] * wt[2]);
}

// RK4 integrator
void RK4(double *wt, double t, double *x) {
double h2;  int i;
Eqs(wt, t, x, k1);  h2 = h / 2.;
for (i = 0; i < M; i++)  f1[i] = x[i] + h2*k1[i];
Eqs(wt, t + h2, f1, k2);
for (i = 0; i < M; i++)  f2[i] = x[i] + h2*k2[i];
Eqs(wt, t + h2, f2, k3);
for (i = 0; i < M; i++)  f3[i] = x[i] + h2*k3[i];
Eqs(wt, t + h, f3, k4);
for (i = 0; i < M; i++)
x[i] += h*(k1[i] + 2.0*k2[i] + 2.0*k3[i] + k4[i]) / 6.;
}

//initializing q[i][j] and p[i][j]
void Initialise() {
```

```
for (int i = 0; i < N; i++)
for (int j = 0; j < N; j++) {
q[i][j] = 0.3*(ranf() - 0.5);
p[i][j] = 0.3*(ranf() - 0.5);
w[i][j] = 0.3*(ranf() - 0.5);
}
 q[-1][-1] = 0.3*(ranf() - 0.5);   // bounds for q[i][j] and p[i][j]
p[N][N] = 0.3*(ranf() - 0.5);
}

// better quality random generator
double ranf() {
const int ia = 16807, ic = 2147483647, iq = 127773, ir = 2836;
int il, ih, it;  double rc;  int iseed = 0;
ih = iseed / iq;  il = iseed%iq;  it = ia*il - ir*ih;
if (it > 0) { iseed = it; }
else { iseed = ic + it; }
rc = ic;  return iseed / rc;
}

// main code
int main() {
int ticks;  double ms;
clock_t start, stop;
 FILE* pOut;                            // file pointer to output file
FILE* wOut;
 fopen_s(&pOut, "swarmRK4.csv", "w");   // opening files
 start = clock() * CLK_TCK;             // clock setup
Initialise();
 while (t <= Tfin)                      // main time loop
for (int i = 0; i < N; i++)
for (int j = 0; j < N; j++) {
// representing q[i][j] and p[i][j] using array qc[]
qc[0] = q[i][j];  qc[1] = p[i][j];

// representing other constant terms using array wt[]
wt[0] = w[i][j]; wt[1] = p[i + 1][j + 1]; wt[2] = q[i - 1][j - 1];
RK4(wt, t, qc);
                t += h;        // updating time

//Hebb's rule: covariance hypothesis
    w[i][j] =-w[i][j]+ eta*(q[i][j] -
        (q[i][j] + p[i][j]) / M)*(qc[0] - (qc[0] + qc[1]) / M);

// updating q[i][j] and p[i][j]:
    q[i][j] = qc[0];  p[i][j] = qc[1];

// printouts to files
   fprintf(pOut, ",%.5f,%.5f", qc[0], qc[1]);
   fprintf(wOut, ",%.5f", w[i][j]);
   } fprintf(pOut, "\n"); fprintf(wOut, "\n");
}
```

```
 stop = clock() * CLK_TCK;            // timing
 ticks = stop - start;
 ms = (float)ticks / 1000.0f;
 printf("finished... (ms = %f)\n", ms);
 getch();  return 0;
}
```

1.2. High-Dimensional Bayesian Particle Filter

The following C++ code uses the Armadillo matrix library [41].

```
// Integration of the swarm control with the particle filter

#include <iostream>
#include <armadillo>        // using Armadillo (64-bit)

using namespace std;
using namespace arma;

// Define a float matrix type.
typedef Mat<float> matf;
float dt = 0.01, t = 0.;
int N = 9;           //Number of vehicles
int b = 2*N*N;       //Number of equations

struct m_struct
{
 int nx;             // State dimension
 int ny;             // Measurement dimension
 matf x0;            // Initial state
 matf P0;            // Covariance for the initial state
 matf Q;             // Process noise covariance
 matf Q_sqrt;
 matf R;             // Measurement noise covariance
 matf R_inverse;
 matf R_sqrt;
 matf P0_sqrt;
};

struct z_struct
{
 matf xTrue;
 matf y;
};

struct gpf_struct
{
 matf xf;
 int info;
};

matf g;
```

```
// Inline function declaration for m.f
matf m_f(matf* x)
{
 g = zeros<matf>(x->n_rows, x->n_cols);
 for (int j = 0; j < b; j += 2)
 {
  g.row(j) = x->row(j) + 50 * (sin(t) - x->row(j))*dt;
  g.row(j + 1) = x->row(j + 1) + 50 * (3 * cos(t) - x->row(j + 1))*dt;
  t += dt;
 }
 return g;
}

// Inline function declaration for m.f for the particle filter one step forward
matf m_fTime(matf* x, matf t)
{
 g = zeros<matf>(x->n_rows, x->n_cols);
 for (int j = 0; j < b; j += 2)
 {
  g.row(j) = x->row(j) + 50 * (sin(t.row(j)) - x->row(j))*dt;
  g.row(j + 1) = x->row(j + 1) + 50 * (3 * cos(t.row(j + 1)) - x->row(j + 1))*dt;
 }
 return g;
}

matf h;

// Measurement equation
matf m_h(matf* x)
{
 h = zeros<matf>(x->n_rows, x->n_cols);
 for (int j = 0; j < b; j += 2)
 {
  h.row(j) = -0.5*x->row(j) + 0.1*x->row(j + 1);
  h.row(j + 1) = +0.1*x->row(j) + 0.5*x->row(j + 1);
 }
 return h;
}

// Particle filter function declaration
matf pf(const matf* y, const m_struct* m, int Npf);

// MC function declaration
void mc(matf& q, matf& i);

int main(int argc, char** argv)
{
 wall_clock timer;
 m_struct m;

 m.x0 = zeros<matf>(b, 1);        // Initial State
 m.nx = b;              // State dimension
```

```
m.ny = b;             // Measurement dimension

m.P0 = (1e-6)*eye<matf>(b, b);
m.P0(0, 0) = 1;

m.Q = 0.001*eye<matf>(b, b);
m.R = 0.01*eye<matf>(b, b);
m.R_inverse = inv<matf>(m.R);
m.R_sqrt = sqrt(m.R);
m.Q_sqrt = sqrt(m.Q);
m.P0_sqrt = sqrt(m.P0);

///////////////////////////////
//    Simulate the model     //
//////////////////////////////

int Tfinal = 200;
const int MC = 1;
z_struct z[MC];

timer.tic();

for (int j = 0; j < MC; ++j)
{
 matf x = zeros<matf>(m.nx, Tfinal + 1);
 matf y = zeros<matf>(m.ny, Tfinal + 1);

 x.col(0) = m.x0 + m.P0_sqrt*randn<matf>(m.nx, 1);

 for (int i = 0; i < Tfinal; i++)
 {
  matf tmp = x.col(i);
  x.col(i) = m_f(&tmp) + m.Q_sqrt*randn<matf>(m.nx, 1);
  y.col(i) = m_h(&tmp) + m.R_sqrt*randn<matf>(m.ny, 1);
  x.col(i + 1) = x.col(i);
  y.col(i + 1) = y.col(i);
 }
 z[j].xTrue = x.cols(0, Tfinal - 1);     // store the true states
 z[j].y = y.cols(0, Tfinal - 1);         // store the measurements
}


/////////////////////////////////////////////
//    Compute the estimates using the PF  //
////////////////////////////////////////////

int Npf = 200;
gpf_struct gPF[MC];
for (int i = 0; i < MC; ++i)
{
 cout << "Monte Carlo iteration:" << i << endl;
 gPF[i].xf = pf(&z[i].y, &m, Npf);         // Run the particle filter
 }

 cout << endl << "Total Code Execution Time is:
\qquad \qquad \qquad " << timer.toc() << "  seconds" << endl;

 //Save csv file for plotting
 z[0].xTrue.save("z2.csv", csv_ascii);
 gPF[0].xf.save("gPF.csv", csv_ascii);


 // Compute RMSE values
 matf RMSE_PF = zeros<matf>(m.nx, 1);
 double eterm;
 matf estErrorPF = zeros<matf>(m.nx, Tfinal);

 for (int i = 0; i < MC; i++)
 {
  estErrorPF = gPF[i].xf - z[i].xTrue;
  RMSE_PF = RMSE_PF + sum<matf>(pow(estErrorPF, 2), 1);
 }
 eterm = MC*Tfinal;
 eterm = 1 / eterm;
 RMSE_PF = sqrt(eterm*RMSE_PF);

 cout << endl << "RMSE for particle filter:" << endl;
 RMSE_PF.print();

 cin.get();
 return 0;
}


matf pf(const matf* y, const m_struct* m, int Npf)
{
 int Tfinal = y->n_cols;
 matf q, index = zeros<matf>(1, Tfinal);
 matf e, yNow;

 matf xf = zeros<matf>(m->nx, Tfinal);
 matf x = repmat<matf>(m->x0, 1, Npf) + sqrt(m->P0)*randn<matf>(m->nx, Npf);
 matf tm = zeros<matf>(b, Npf);
 float t = 0.;

 for (int u = 0; u < Npf; u++)            // generating time matrix TM
  for (int v = 0; v < b; v += 2)
  {
   tm(v, u) = t;
   tm(v + 1, u) = t;
   t += dt;
  }

 for (int j = 0; j < Tfinal; j++)
 {
  yNow = y->col(j);
  e = repmat<matf>(yNow, 1, Npf);
  e -= m_h(&x);
// Compute the importance weights
  q = exp(-0.12*(sum<matf>(e % (m->R_inverse*e))));

  if (accu<matf>(q)>1e-12)         // divergence check
  {
   q = q / accu(q);
   xf.col(j) = sum<matf>(repmat<matf>(q, m->nx, 1) % x, 1);
   mc(q, index);           // Resample using MC
   for (int i = 0; i < Npf; ++i)
   {
    x.col(i) = x.col(index(i));
   }
  }

  else
  {
   xf = zeros<matf>(m->nx, Tfinal);
   cout << "Weights close to zero at j= " << j << endl;
   break;
  }

  x = m_fTime(&x, tm);
// Predict the particles one step forward using normal random generates
  x = x + m->Q_sqrt*randn<matf>(m->nx, Npf);
 }
 return xf;
}


void mc(matf& q, matf& i)
{
 int M = q.n_cols;
 i = zeros<matf>(1, M);
 matf u, qc;
 qc = cumsum<matf>(q, 1);

// Random generates from uniform distribution (randu)
 u = cumsum<matf>(randu<matf>(1, M), 1) / M;
 int k = 1;

 for (int j = 0; j < M; ++j)
 {
  while (qc(k) < u(j))
  {
   ++k;
  }
  i(j) = k;
 } }
```