An Adaptive Spiking Neural Controller for Flapping Insect-scale Robots

Taylor S. Clawson[1] *Student Member, IEEE,* Terrence C. Stewart[2], Chris Eliasmith[2], and Silvia Ferrari[1] *Senior Member, IEEE*

***Abstract*—Insect-scale flapping robots are challenging to stabilize due to their fast dynamics, unmodeled parameter variations, and the periodic nature of their control input. Effective controller designs must tolerate wing asymmetries that occur due to manufacturing errors and react quickly to stabilize the fast unstable modes of the system. Additionally, they should have minimal power requirements to fit within the tightly constrained power budget associated with insect-scale flying robots. Adaptive control methods are capable of learning online to account for uncertain physical parameters and other model uncertainties, and can thus improve system performance over time. In this work, a spiking neural network is used to stabilize hovering of an insect-scale robot in the presence of unknown parameter variations. The controller is shown to adapt rapidly during a simulated flight test and requires a total of only 800 neurons, allowing it to be implemented with minimal power requirements.**

I. INTRODUCTION

Insect-scale flapping robots have recently demonstrated the ability to hover and perform basic trajectory following and other maneuvers through the implementation of modular PID and adaptive controllers [1]–[3]. Due to their size and weight, these robots have a wide variety of potential applications, including search and rescue and remote monitoring in confined spaces or hazardous environments. Generating lift through flapping as opposed to more traditional fixed-wing or rotary designs is inherently more efficient at this scale [1]. Additionally, it provides a great deal of agility, as can be observed in many flying insects [4], [5].

Stabilizing and controlling these robots remains a challenging problem, however, due to a number of factors. A successful controller must be robust to uncertain physical parameters caused by manufacturing errors such as wing asymmetries, which result in a noticeable torque bias during flight that rapidly destabilizes the robot. Additionally, actuator dynamics vary between robots, leading to variations in flapping kinematics. Analytic models of aerodynamic forces are imperfect, and their accuracy can degrade during rapid maneuvers or in other poorly modeled flight regimes. The controller must be able to react quickly to stabilize the system's naturally fast, unstable dynamic modes. Additionally, the model and control law must account for the periodic flapping nature of the system. This periodicity means that the system has limit cycles instead of equilibrium points, and that linearizations thus result in time-varying instead of time-invariant systems. These systems are also severely underactuated, possessing between 10 and 12 degrees of freedom but only 3 or 4 control inputs. This limits the effectiveness of some common techniques such as feedback linearization.

Recent work has shown that spiking neural networks (SNNs) are capable of being used as adaptive controllers for a wide variety of systems. They have been used to stabilize longitudinal fixed-wing aircraft dynamics [6], perform trajectory following on robot arms [7], perform navigation and control for terrestrial insect-like robots [8], [9], and stabilize simplified longitudinal dynamics for insect-scale flapping robots [10]. Building on these previous efforts, the work presented here develops an adaptive SNN controller capable of stabilizing hovering flight of a simulated flapping insect-scale robot. The robot model used in this work computes aerodynamic forces on the wings during flight with blade-element theory and includes 6 degrees of freedom in the main robot body [11].

SNN-based controllers have several desirable properties for controlling insect-scale flapping robots. Hardware implementations of SNNs have shown to be very power efficient [12], an important consideration given the small power and weight budgets for robots at this scale. For instance, the power budget for the RoboBee is around 20mW [1], most of which is required to power the actuators. Additionally, event-based sensors are available which yield low-latency feedback on the order of 15μs [13]. The output from these sensors must be processed before being used with most existing control algorithms however, thus reducing the inherent benefits of low latency and low power consumption. Integrating the event-based output from these sensors with SNN-based control algorithms has the potential to provide low latency and low power feedback control if SNN control algorithms can be developed which demonstrate robust performance for complex unstable systems. To this end, this paper presents an adaptive SNN-based controller capable of stabilizing a realistic model of an insect-scale flapping robot while adapting for unknown parameter variations.

Due to a lack of sufficient rotational damping, open loop hovering flight is unstable for many insect-scale flapping robots, including the RoboBee shown in Fig. 1 [14], [15]. In systems such as the RoboBee, open loop flight tests often result in a crash in less than 0.3 seconds, leaving very little time for a poorly tuned controller to adapt to uncertain physical parameters. If the adaptive controller is not initially robust to uncertainties in physical parameters prior to any online learning, it will be unable to stabilize hovering flight long enough to learn the parameters of the robot. To achieve

[1]Taylor S. Clawson and Silvia Ferrari are with the Department of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14850 tsc83@cornell.edu, ferrari@cornell.edu

[2]Terrence C. Stewart and Chris Eliasmith are with the Centre for Theoretical Neuroscience, University of Waterloo, Waterloo, Ontario, Canada N2L3G1

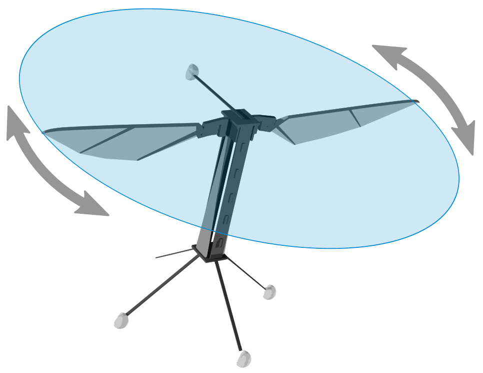


Fig. 1: The RoboBee, with the wing stroke plane shown in blue.

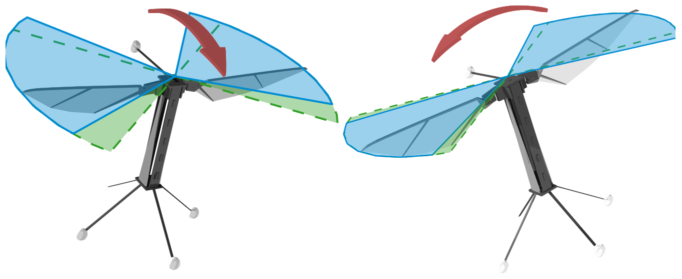


Fig. 2: Control torques in the direction indicated by the arrows are achieved by varying the wing stroke from the green region to the blue region

this, a portion of the SNN control signal is tuned offline to approximate a linear controller which was developed for the idealized robot model. A separate term in the control law is used to adapt online to any errors caused by parameter variations in the system.

Several distinct control methods have been implemented to stabilize hovering flight for flapping-wing robots previously. A hierarchical approach to designing flight controllers for insect-scale flapping robots proposed in [16]. The design calls for a stabilizing controller to compute desired forces and torques, which are passed on to a wing trajectory controller, responsible for generating periodic signals to be sent to the wings. This modular approach has been extended to several control methods for insect-scale flapping robots.

A modular PID approach was used to stabilize hovering and perform basic trajectory following for the RoboBee, shown in Fig. 1, in [1]. A similar approach in [17] used the same modular architecture, but included adaptive terms to estimate wing torque biases and improve tracking performance. This design was augmented with an iterative learning scheme to perch on vertical surfaces in [3]. The adaptive approach in [18] is an extension of [3], but also accounts for wind disturbance rejection based on feedback estimates.

The SNN controller proposed here is shown to be capable of stabilizing an insect-scale flapping robot in the presence of unknown parameter variations, achieving improved performance over the linear controller on which a portion of the SNN control law is based. This is demonstrated by controlling a simulation of the RoboBee which has been previously validated against experimental data [11].

II. DYNAMIC MODEL

The RoboBee is a 14mm tall flapping robot that has demonstrated hovering [1], basic lateral maneuvers, and perching [2]. Thrust is provided by two wings mounted to the top of the main body, as shown in Fig. 1. Each wing is independently actuated, and only the stroke angle of each wing can be directly controlled. The current design does not allow for any stroke-plane deviation, and the wing pitch cannot be controlled directly during flight.

Piezoelectric actuators drive transmissions that flap the wings to generate lift. Torque control is achieved by changing the wings' relative stroke amplitudes for roll and the mean stroke angle for pitch, as shown in Fig. 2. As in many biological insects, the robot is unstable in passive flight [14], [15]. The instability arises because of net drag forces acting high above the robots center of mass. This unstable mode arises in both the longitudinal and lateral directions, destabilizing the robot in both pitch ψ and roll θ. The dynamics are neutrally stable in yaw, so for simplicity, yaw control is not considered in this work.

The RoboBee model used for this paper combines rigid-body dynamics with blade element theory as described in [11]. The resulting model accounts for 6 degrees of freedom in the body of the robot, and 1 in each wing. Each wing's stroke angle ϕ_W is specified kinematically, while the wing pitch ψ_W is free to rotate. No stroke-plane deviation is allowed in the current design, so the position of the wings at any point in time can be specified using only ϕ_W and ψ_W.

The state vector $\mathbf{x}$ is broken up into two halves, the configuration $\mathbf{q}$ and the time derivative of the configuration $\dot{\mathbf{q}}$, where $\mathbf{q} = [\phi_R \;\; \phi_L \;\; \psi_R \;\; \psi_L \;\; \phi \;\; \theta \;\; \psi \;\; x \;\; y \;\; z]^T$. The stroke angles of the right and left wings are ϕ_R and ϕ_L, the pitch angles of the right and left wings are ψ_R and ψ_L, the body Euler angles are yaw ϕ, roll θ, and pitch ψ, and the position of the body in the fixed frame is given by $\mathbf{r} = x\hat{\imath} + y\hat{\jmath} + z\hat{\mathbf{k}}$. The complete state is written as

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{q}(t) \\ \dot{\mathbf{q}}(t) \end{bmatrix} \in \mathbb{R}^n \tag{1}$$

The control vector $\mathbf{u} = [u_a \;\; u_p \;\; u_r] \in \mathbb{R}^m$ contains the stroke amplitude input u_a, the pitch bias u_p, and the roll bias u_r. These parameters are used to compute the stroke amplitude and mean stroke angle for each wing. The equations of motion are linear in $\ddot{\mathbf{q}}$, and can be expressed in terms of the inertia matrix $\mathbf{M}(\mathbf{q})$, a vector of nonlinear forces $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, and the input matrix $\mathbf{B}(t)$,

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{B}(t)\mathbf{u} \tag{2}$$

Solving (2) for $\ddot{\mathbf{q}}$, the expression obtained for the state derivative $\dot{\mathbf{x}}$ is

$$\begin{bmatrix}\dot{\mathbf{q}}\\ \ddot{\mathbf{q}}\end{bmatrix} = \begin{bmatrix}\mathbf{0} & \mathbf{I}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{x}(t) - \begin{bmatrix}\mathbf{0}\\ \mathbf{M}^{-1}\mathbf{C}\end{bmatrix} + \begin{bmatrix}\mathbf{0}\\ \mathbf{M}^{-1}\mathbf{B}(t)\end{bmatrix}\mathbf{u}(t) \quad (3)$$

where the dependence on the configuration $\mathbf{q}$ and its derivative $\dot{\mathbf{q}}$ are suppressed for brevity. As can be seen from (3), the dynamics can be written in the control affine form

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x}, t)\mathbf{u}(t) \quad (4)$$

If the equations of motion given in (4) can be linearized about hovering without significant loss of accuracy, then control design is greatly simplified. Linearization about a single point in state space will yield a poor model for control design however, as the wing states vary greatly during flight. On the other hand, linearizing about the periodic hovering trajectory should provide a good approximation for relatively small deviations from hovering. This will yield a linear time varying solution in terms of deviation from the hovering trajectory. The steady hovering trajectory is characterized by a time-varying state denoted by $\mathbf{x}^*(t)$ and a constant control input $\mathbf{u}^*$. The values for this trajectory can be computed by solving (4) for the initial conditions and constant control inputs that produce a periodic trajectory so that $\mathbf{x}^*(t) = \mathbf{x}^*(t+T)$, where the period $T = 120$Hz is equal to the flapping frequency.

After defining the state deviation $\tilde{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}^*(t)$ and the control deviation $\tilde{\mathbf{u}}(t) = \mathbf{u}(t) - \mathbf{u}^*$, the linearized equations of motion are given by

$$\dot{\tilde{\mathbf{x}}} = \mathbf{A}(t)\tilde{\mathbf{x}} + \mathbf{B}(t)\tilde{\mathbf{u}} \quad (5)$$

where, if $\dot{\mathbf{x}} = \mathbf{h}(\mathbf{x}, \mathbf{u}, t)$,

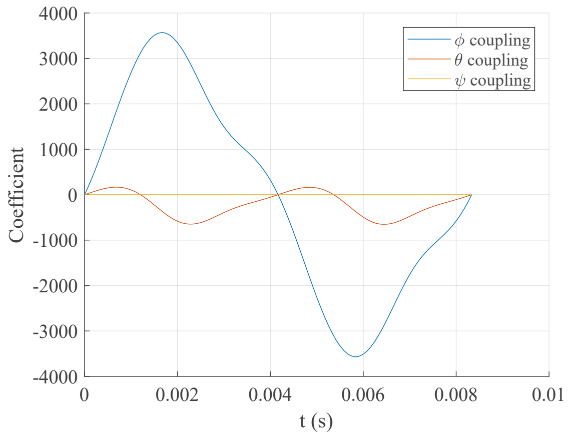
$$\mathbf{A}(t) = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{(\mathbf{x}^*(t),\mathbf{u}^*)}, \quad \mathbf{B}(t) = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{u}}\right|_{(\mathbf{x}^*(t),\mathbf{u}^*)} \quad (6)$$

Also, due to the periodic nature of flapping flight, the model is periodic with period T, such that

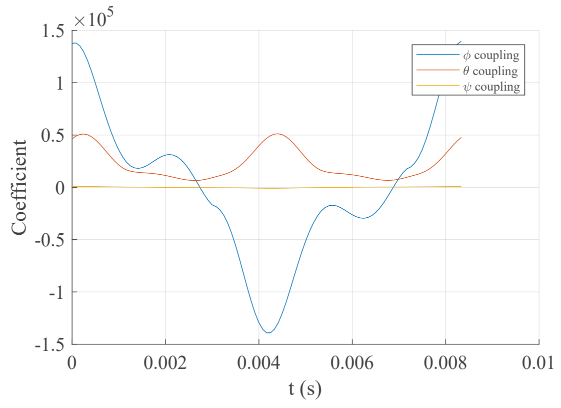
$$\mathbf{A}(t) = \mathbf{A}(t+T), \quad \mathbf{B}(t) = \mathbf{B}(t+T)$$

The linear time-varying model shown in (5) is far more suitable for controller design than the full non-linear time-varying equations of motion. The equations in this form do not accurately represent the system responsiveness to all control inputs however, even near the equilibrium trajectory. This is caused by the inclusion of the actuator dynamics in the full equations of motion (4). Linearizing the equations of motion directly as shown in (5) captures the response of the system to an instantaneous control input, which includes the transient wing response instead of the steady state response. The transient response is, in some cases, significantly different than the steady state response and thus causes the linearized control input matrix $\mathbf{B}(t)$ to inaccurately represent the system response to control inputs.

The most noticeable case of the transient wing response differing from the steady state response is after a step change to the roll input u_r. Once the wing stroke amplitudes and pitch angles reach a steady state, a positive roll bias will increase the flapping amplitude of the left wing relative to the right wing, which causes a greater net lift force on the left wing than the right wing, as shown in Fig. 2b. The result is a torque that, in the steady state, causes primarily rolling.



(a) Using the transient wing response



(b) Using the steady state wing response

Fig. 3: Coefficients of the linearized input matrix $\mathbf{B}(t)$ which couple the roll input u_r to body Euler angles

However, instantaneous response to a step change in roll input u_r is, briefly, quite different from the steady state response. As one wing accelerates and the other decelerates, a yaw torque is generated and the body very slightly rolls opposite the desired direction. Flapping one wing faster than the other is known to theoretically cause yaw torque [1], [19]. The impact on the control input matrix $\mathbf{B}(t)$ is significant. The instantaneous response to a roll control input is captured by the linearization, which then indicates that a roll control works in the opposite of the desired direction and causes significant yawing. Figure 3a plots the coefficients of the $\mathbf{B}(t)$ matrix that couple the roll input u_r to the body Euler angle accelerations when the $\mathbf{B}(t)$ matrix is calculated using the linearization shown in (5).

Since the desired effect of the linearization is to capture the response of the body due to the steady state motion of the actuators, the steady state wing motion must be computed for any control input used when calculating the numerical

Jacobian shown in (6). The state vector must first be divided into the wing states $\mathbf{x}_w$ and the body states $\mathbf{x}_b$. The steady state wing motion can then be computed by integrating the equations of motion for the wing states, $\dot{\mathbf{x}}_w = \mathbf{f}_w(\mathbf{x}, \mathbf{u}, t)$, while assuming the body remains in the equilibrium trajectory defined by $\mathbf{x}_b^*$. Applying this to the linearization results in a much improved representation of the LTV system. The coefficients of the $\mathbf{B}(t)$ matrix computed using the steady state wing trajectories are shown in Fig. 3b. These now correctly indicate that a positive roll control input will result in a positive roll torque. The yawing effect remains, but if the control is held constant throughout the period, yaw torque will cancel.

III. SNN CONTROL DESIGN

The control signal can be provided entirely by populations of spiking neurons using an adaptation of the approach presented in [7]. Each individual population of neurons is structured as a single-layer feed forward network. A total of four separate neuron populations are used to compute the complete control signal, which is the summation of two terms. One term approximates the signal from a linear controller that is designed to stabilize the ideal model [11], and the other is an adaptive term to account for parameter variations in the robot. A total of 800 neurons are used across all four populations.

The linear controller used here is a proportional-integral-filter (PIF) compensator [20], which uses an augmented state vector $\boldsymbol{\chi}(t) = [\tilde{\mathbf{x}}^T(t)\ \tilde{\mathbf{u}}^T(t)\ \boldsymbol{\xi}^T(t)] \in \mathbb{R}^q$ and a constant gain matrix $\mathbf{K} \in \mathbb{R}^{m\times q}$. Using $(\cdot)^*$ to denote the desired set point condition, the augmented state vector contains the state error $\tilde{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}^* \in \mathbb{R}^n$, the control error $\tilde{\mathbf{u}}(t) = \mathbf{u}(t) - \mathbf{u}^* \in \mathbb{R}^m$, and the integral of the output error $\boldsymbol{\xi}(t) = \boldsymbol{\xi}(0) + \int_0^t \tilde{\mathbf{y}}(\tau)d\tau \in \mathbb{R}^r$. The PIF control law is

$$\mathbf{u}_{PIF}(t) = \int_0^t -\mathbf{K}\boldsymbol{\chi}(\tau)d\tau \tag{7}$$

For simplicity, the control input error and integral of the output error are both assumed to be zero for the SNN approximation. The most significant terms in the PIF control signal are dependent on the state error $\tilde{\mathbf{x}}(t)$ only.

To formulate the PIF approximation term $\mathbf{u}_0$ in terms of spiking neurons, the Neural Engineering Framework (NEF) [21], [22] is used. The NEF provides a framework for representing algorithms in terms of spiking neuron models, and includes methods for both encoding time-varying vectors in populations of neurons and for decoding the output of spiking neurons into a continuous signal.

The state $\mathbf{x}(t)$ is encoded in a population of neurons as a sequence of spikes from each neuron. The nonlinear spiking neuron model of the ith neuron within the population is $G_i : \mathcal{J} \times \mathcal{T} \to \mathbb{R}$, which maps the input current $J_i \in \mathcal{J} \subset \mathbb{R}$ and time $t \in \mathcal{T} \subset \mathbb{R}_0^+$ to the spike train $r_i \in \mathbb{R}$,

$$r_i = G_i(J_i(\mathbf{x}), t) \tag{8}$$

The total current J_i into neuron i is written in terms of a gain term $\alpha_i \in \mathbb{R}$, the neuron's preferred direction vector $\mathbf{e}_i \in \mathbb{R}^n$, and a fixed background current $J_i^{bias} \in \mathbb{R}$,

$$J_i(\mathbf{x}) = \alpha_i \mathbf{e}_i^T \mathbf{x}(t) + J_i^{bias} \tag{9}$$

The nonlinear spiking neuron model G_i used in this work is the leaky integrate-and-fire model [23]. This models the voltage V_i across the membrane of a neuron as an RC circuit governed by the first-order ODE

$$\tau_{RC}\frac{dV_i}{dt} = -V_i(t) + R_i J_i(\mathbf{x}) \tag{10}$$

where τ_{RC} is the decay time constant, and R_i is the resistance in the circuit and accounts for the passive "leak" of the current, and the initial condition is $V_i(0) = V_0$. The spike train r_i of the neuron can be written as a summation of Dirac delta functions indexed by k. A spike time t_{ik} is defined when the voltage reaches a threshold V_{th}, following which the membrane potential V_i is set to the reset value $V_r < V_{th}$. Expressed formally,

$$t_{ik} = \{\tau :\ V_i(\tau) = V_{th}\} \tag{11}$$

$$\lim_{t\to t_{ik}^+} V_i(t) = V_r \tag{12}$$

Following a spike, neuronal activity is held at the reset value V_r for a refractory period τ_{ref}, after which the membrane potential $V_i(t)$ again follows (10).

The spiking activity contained in r_i can be expressed using the spike times t_{ik} as,

$$r_i = \sum_k \delta_i(t - t_{ik}) \tag{13}$$

where $\delta_i(t)$ denotes the Dirac delta function. To decode a continuous estimate of the state $\mathbf{x}(t)$ from the spiking activity r_i, the filtered postsynaptic activity is used. In this model, synapses act as linear filters $h_i(t)$ on the spike trains. This can be explicitly stated using the postsynaptic time constant τ_{PSC} as $h_i(t) = (1/\tau_{PSC})e^{-t/\tau_{PSC}}$. The activity a_i of neuron i is thus the summation of impulse responses from linear filters, which can also be written as the convolution of the synaptic filters $h_i(t)$ with the spike train,

$$\begin{aligned} a_i(\mathbf{x}, t) &= h_i(t) * G_i(J_i(\mathbf{x}), t) \\ &= \sum_k h_i(t - t_{ik}) \end{aligned} \tag{14}$$

where "$*$" denotes the convolution operator. The activity of the population of neurons can be used to represent the PIF control signal through the use of carefully chosen linear decoders $\mathbf{d}_i \in \mathbb{R}^m$,

$$\mathbf{u}_0(t) = \sum_i a_i(\mathbf{x}, t)\mathbf{d}_i \tag{15}$$

The linear decoders which give the optimal representation of the PIF control signal are found using standard least squares optimization over a set $\mathcal{B}$ of sampled data,

$$\mathbf{d}_i = \underset{\mathbf{m}_i}{\operatorname{argmin}} \int_{\mathbf{x}\in\mathcal{B}} \|\ \mathbf{g}(\mathbf{x}) - \sum_i \bar{a}_i(\mathbf{x})\mathbf{m}_i\ \|^2\ d\mathbf{x} \tag{16}$$

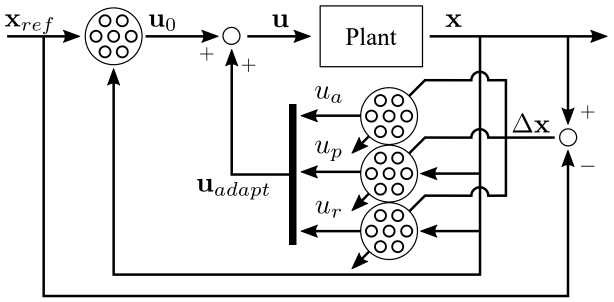


Fig. 4: The SNN control architecture, where clusters of circles represent neuron populations.

where $\mathbf{g}(\mathbf{x})$ is the output of the PIF control law for a given point in state space and $\bar{a}_i(\mathbf{x})$ is the time-averaged steady-state neuron activity for the same point. The elements of $\mathcal{B}$ are sampled from a subset of the state space near the hovering condition, where the neuron population is expected to be capable of approximating the PIF control law.

The control input $\mathbf{u}_0(t)$ closely approximates the signal from the PIF compensator, which is designed to control an ideal model of the RoboBee [11]. In reality however, the RoboBee is subject to manufacturing defects which are not present in the ideal model. These defects include variations in the wing hinges and asymmetries between the actuators that result in anomalous torque generation during flight. An adaptive control input $\mathbf{u}_{adapt}(t)$ accounts for this and other unmodeled dynamics present in the system. The computation of $\mathbf{u}_{adapt}(t)$ follows a direct adaptive method similar to that outlined in [7], [24], [25].

The adaptive term $\mathbf{u}_{adapt}(t)$ contains three scalar quantities: an amplitude input $u_a(t)$, a pitch input $u_p(t)$, and a roll input $u_r(t)$,

$$\mathbf{u}_{adapt}(t) = \begin{bmatrix} u_a(t) & u_p(t) & u_r(t) \end{bmatrix}^T \tag{17}$$

Each of these scalars are computed from a separate population of neurons, as shown in Fig. 4. Each population is trained online to zero out an error signal that depends on the error $\Delta\mathbf{x}(t) = \mathbf{x}(t) - \mathbf{x}_{ref}(t)$ between the state $\mathbf{x}(t)$ and the reference state $\mathbf{x}_{ref}(t)$, a constant gain term $\alpha \in \mathbb{R}_0^+$, and the constant vector $\mathbf{\Lambda} \in \mathbb{R}^n$,

$$E(t) = \mathbf{\Lambda}^T(\Delta\mathbf{x}(t) + \alpha\Delta\dot{\mathbf{x}}(t)) \tag{18}$$

In general, variations in actuator performance between robots result in unknown dynamics that change as a function of the state. However, it is assumed that the actuator performance dominates any variations in the dynamics in the body z direction, so the adaptive amplitude $u_a(t)$ is not computed as a function of the state. The adaptive amplitude input $u_a(t)$ is computed by decoding the neuron activity through linear decoders $d_{a,i} \in \mathbb{R}$,

$$u_a(t) = \sum_i a_{a,i}(t) d_{a,i}(t) \tag{19}$$

where the activity of the neurons in this population is

$$a_{a,i}(t) = h_i(t) * G_i(J_i^{bias}, t) \tag{20}$$

This can be written more compactly by defining the activity for the amplitude population as $\mathbf{a}_a = [a_{a,1},\ a_{a,2},\ \dots,\ a_{a,N}]^T$ and the corresponding population decoders $\mathbf{d}_a = [d_{a,1},\ d_{a,2},\ \dots,\ d_{a,N}]^T$. The amplitude control signal (19) can now be rewritten as

$$u_a(t) = \mathbf{a}_a^T(t)\mathbf{d}_a(t) \tag{21}$$

The linear decoders $\mathbf{d}_a(t)$ for the amplitude population are continuously updated based on an adaptation rate $\gamma \in \mathbb{R}_0^+$, the population activity, and the error signal (18),

$$\dot{\mathbf{d}}_a(t) = \gamma\mathbf{a}_a(t)E(t) \tag{22}$$

For the amplitude input $u_a(t)$, the vector $\mathbf{\Lambda}$ is chosen so that the error function is only a function of the body z velocity error $\Delta v_z(t)$ and its derivative.

The adaptive pitch input $u_p(t)$ and adaptive roll input $u_r(t)$ are computed similarly to the adaptive amplitude input (21). Pitch and roll torques experienced during flight are in general a function of the state, because drag on the wings varies as a function of the robot's angular and linear velocity. This is accounted for in both $u_p(t)$ and $u_r(t)$ by incorporating the state in the neural activity, as shown in (14),

$$u_p(t) = \mathbf{a}_p^T(\mathbf{x}, t)\mathbf{d}_p(t) \tag{23}$$

$$\dot{\mathbf{d}}_p(t) = \gamma\mathbf{a}_p(\mathbf{x}, t)E(t) \tag{24}$$

The pitch input is computed to reduce the error in body x velocity $v_x(t)$ by choosing $\mathbf{\Lambda}$ in the error function (18) so that the error depends only on $v_x(t)$ and its derivative.

The roll input $u_r(t)$ is computed identically to the pitch input (23), except that $\mathbf{\Lambda}$ in the error function is chosen so that the error depends only on the body y velocity $v_y(t)$ and its derivative,

$$u_r(t) = \mathbf{a}_r^T(\mathbf{x}, t)\mathbf{d}_r(t) \tag{25}$$

$$\dot{\mathbf{d}}_r(t) = -\gamma\mathbf{a}_r(\mathbf{x}, t)E(t) \tag{26}$$

The sign difference between the right hand side of (24) and (26) results from differences in sign convention between the two control inputs.

In summary, the adaptive amplitude input (21) is designed to zero out any error in body z velocity, the pitch input (23) is designed to zero out error in the body x velocity, and the roll input (25) will zero out the error in body y velocity. The complete control signal sent to the plant is the sum of (15) and (17),

$$\mathbf{u}(t) = \mathbf{u}_0(t) + \mathbf{u}_{adapt}(t) \tag{27}$$

The first term $\mathbf{u}_0(t)$ is trained offline to control the ideal model, and the adaptive term $\mathbf{u}_{adapt}(t)$ is trained online to minimize the steady-state error $\Delta\mathbf{x}(t)$ caused by parameter variations and constant disturbances.

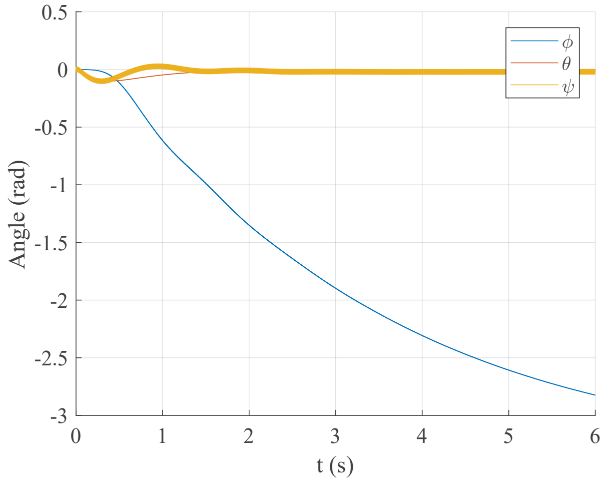


Fig. 5: For comparison with the SNN controller, the PIF compensator quickly stabilizes roll θ and pitch ψ, but the yaw angle ϕ is not stabilized.

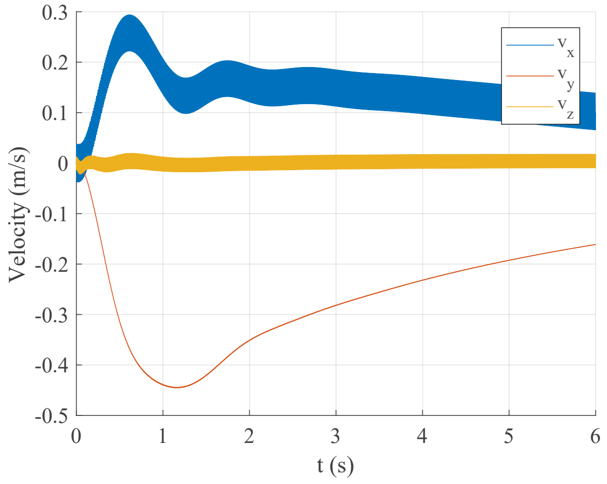


Fig. 6: For comparison with the SNN controller, the PIF compensator successfully stabilizes v_z, but only slowly stabilizes v_x and v_y, resulting in significant drift from the initial position.

IV. RESULTS

Closed loop flight test simulations were conducted using both the SNN controller (27) and the PIF compensator (7) to control the model from [11]. Compared to previous stroke-averaged models, this model more accurately represents the coupling between various degrees of freedom in the robot by using blade element theory to compute instantaneous aerodynamic forces during flapping. The model was compared with experimental data in [11] to validate its effectiveness at representing the flight dynamics of the physical robot. To verify the ability of each controller to compensate for parameter variations in the model, wing asymmetry was introduced in both simulations by applying a static amplitude bias between the right and left wings of the model for roll and a static mean stroke angle offset for pitch. In both cases, the controllers were commanded to control hovering flight.

The attitude and body velocity during the flight simulation using the PIF compensator are shown in Fig. 5 and Fig. 6. Although the integral term in the PIF compensator does work to eliminate steady-state error in v_x and v_y as shown in Fig. 6, it does so slowly, and at the end of the 6 second test flight the robot maintains a significant non-zero velocity. Due to the positional drift, the robot continues yawing throughout the simulation as shown in Fig. 5. However, the PIF compensator is able to quickly stabilize pitch and roll to near zero.

The same initial conditions and wing biases were used to simulate another closed loop flight, this time using the adaptive SNN (27) to control the simulated robot. The parameters for the adaptive input $\mathbf{u}_{adapt}(t)$ were tuned using a grid parameter search. The search varied the parameters of the decoder update laws to find the values that lead to the smallest total deviation from the starting hover position in meters. The attitude and body velocity from a trial with simulated wing bias and random initial conditions are shown in Fig. 7 and Fig. 8, respectively. During this simulated flight test, the RoboBee remained within 10cm of its starting position.

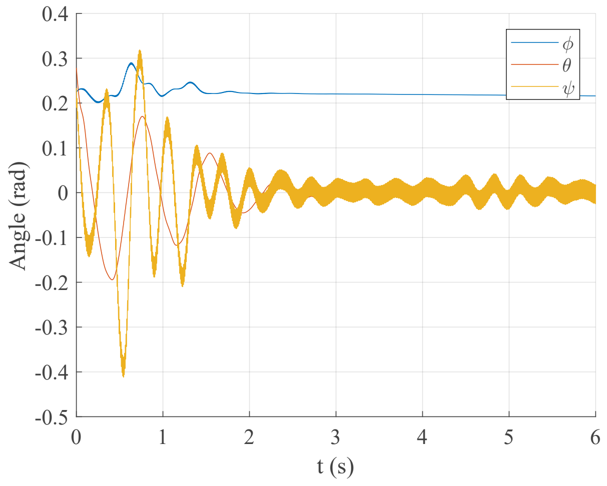


Fig. 7: During a simulated hovering flight, the adaptive SNN successfully stabilizes roll θ and pitch ψ about zero despite parameter variations in the model, while the yaw angle ϕ stabilizes at a constant non-zero value.

The body attitude and velocity are both stabilized by the adaptive SNN controller. The yaw angle ϕ settles at a non-zero constant in Fig. 7 and the pitch angle ψ and roll angle θ both stabilize near zero. The yaw angle was not included in the set point and thus was not expected to reach zero, since the current model lacks direct yaw control authority. All components of the body velocity stabilize near zero as shown in Fig. 8 after approximately 3 seconds. Comparing the RoboBee attitude and velocity when controlled by the PIF compensator (Fig. 5 and Fig. 6) with the attitude and velocity when controlled by the adaptive SNN (Fig. 7 and Fig. 8) demonstrates the ability

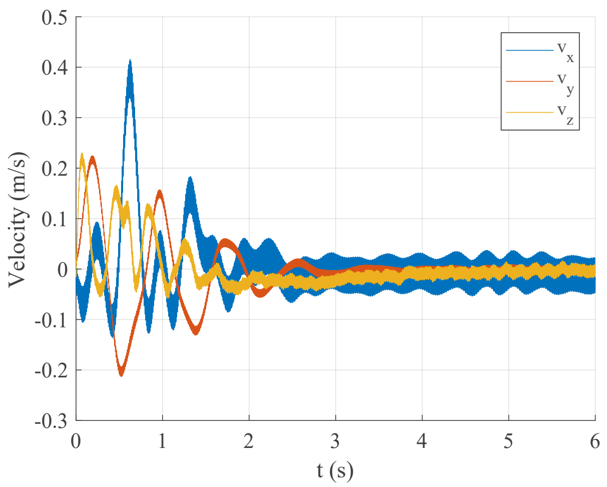


Fig. 8: During a simulated hovering flight, the adaptive SNN successfully stabilizes all components of the body velocity near zero despite parameter variations.

of the adaptive SNN to account for the unmodeled torque disturbances caused by the simulated wing bias.

V. CONCLUSIONS

Control using SNNs has several potential benefits, including the potential for low latency control loops and low power consumption. This is especially true when considering integrating SNN controllers with event-based sensors. This paper presents an SNN-based control algorithm that is shown to be capable of stabilizing an accurate model of an unstable insect-scale flapping robot despite unknown parameter variations. One term in the control law approximates the control signal from a linear controller designed for the idealized model, while another term adapts online to account for parameter variations that are unknown *a-priori*. The controller is able to adapt to the parameter variations within 3 seconds and successfully stabilizes hovering flight. This demonstrates the ability of SNN controllers to stabilize inherently unstable systems that require rapid, accurate feedback control.

ACKNOWLEDGMENT

This research was supported by ONR grant N000141712614, the Canada Research Chairs program, NSERC Discovery grant 261453, ONR grant N0001415l2827, and AFOSR grant FA9550-17-1-0026. This collaborative project started at the 2017 Telluride Neuromorphic Engineering Workshop.

REFERENCES

[1] K. Y. Ma, P. Chirarattananon, S. B. Fuller, and R. J. Wood, "Controlled flight of a biologically inspired, insect-scale robot," *Science*, vol. 340, no. 6132, pp. 603–607, 2013. [Online]. Available: http://science.sciencemag.org/content/340/6132/603

[2] M. Graule, P. Chirarattananon, S. Fuller, N. Jafferis, K. Ma, M. Spenko, R. Kornbluh, and R. Wood, "Perching and takeoff of a robotic insect on overhangs using switchable electrostatic adhesion," *Science*, vol. 352, no. 6288, pp. 978–982, 2016.

[3] P. Chirarattananon, K. Y. Ma, and R. J. Wood, "Fly on the wall," in *Biomedical Robotics and Biomechatronics (2014 5th IEEE RAS & EMBS International Conference on.* IEEE, 2014, pp. 1001–1008.

[4] M. H. Dickinson, F.-O. Lehmann, and S. P. Sane, "Wing rotation and the aerodynamic basis of insect flight," *Science*, vol. 284, no. 5422, pp. 1954–1960, 1999.

[5] S. N. Fry, R. Sayaman, and M. H. Dickinson, "The aerodynamics of free-flight maneuvers in drosophila," *Science*, vol. 300, no. 5618, pp. 495–498, 2003.

[6] G. Foderaro, C. Henriquez, and S. Ferrari, "Indirect training of a spiking neural network for flight control via spike-timing-dependent synaptic plasticity," in *Decision and Control (CDC), 2010 49th IEEE Conference on.* IEEE, 2010, pp. 911–917.

[7] T. DeWolf, T. C. Stewart, J.-J. Slotine, and C. Eliasmith, "A spiking neural model of adaptive arm control," in *Proc. R. Soc. B*, vol. 283, no. 1843. The Royal Society, 2016, p. 20162134.

[8] P. Mazumder, D. Hu, I. Ebong, X. Zhang, Z. Xu, and S. Ferrari, "Digital implementation of a virtual insect trained by spike-timing dependent plasticity," *INTEGRATION, the VLSI journal*, vol. 54, pp. 109–117, 2016.

[9] X. Zhang, Z. Xu, C. Henriquez, and S. Ferrari, "Spike-based indirect training of a spiking neural network-controlled virtual insect," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on.* IEEE, 2013, pp. 6798–6805.

[10] T. S. Clawson, S. Ferrari, S. B. Fuller, and R. J. Wood, "Spiking neural network (snn) control of a flapping insect-scale robot," in *Decision and Control (CDC), 2016 IEEE 55th Conference on.* IEEE, 2016, pp. 3381–3388.

[11] T. S. Clawson, S. B. Fuller, R. J. Wood, and S. Ferrari, "A blade element approach to modeling aerodynamic flight of an insect-scale robot," in *American Control Conference (ACC), 2017.* IEEE, 2017, pp. 2843–2849.

[12] K. Boahen, "A neuromorph's prospectus," *Computing in Science & Engineering*, vol. 19, no. 2, pp. 14–28, 2017.

[13] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128$\times$128 120 db 15μs latency asynchronous temporal contrast vision sensor," *IEEE journal of solid-state circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[14] Z. J. Wang, "Insect flight: From newton's law to neurons," *Annual Review of Condensed Matter Physics*, vol. 7, pp. 281–300, 2016.

[15] S. B. Fuller, M. Karpelson, A. Censi, K. Y. Ma, and R. J. Wood, "Controlling free flight of a robotic fly using an onboard vision sensor inspired by insect ocelli," *Journal of The Royal Society Interface*, vol. 11, no. 97, p. 20140281, 2014.

[16] X. Deng, L. Schenato, and S. S. Sastry, "Flapping flight for biomimetic robotic insects: Part ii-flight control design," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 789–803, 2006.

[17] P. Chirarattananon, K. Y. Ma, and R. J. Wood, "Adaptive control of a millimeter-scale flapping-wing robot," *Bioinspiration & Biomimetics*, vol. 9, no. 2, p. 025004, 2014.

[18] P. Chirarattananon, Y. Chen, E. F. Helbling, K. Y. Ma, R. Cheng, and R. J. Wood, "Dynamics and flight control of a flapping-wing robotic insect in the presence of wind gusts," *Interface Focus*, vol. 7, no. 1, p. 20160080, 2017.

[19] M. W. Oppenheimer, D. B. Doman, and D. O. Sigthorsson, "Dynamics and control of a biomimetic vehicle using biased wingbeat forcing functions," *Journal of guidance, control, and dynamics*, vol. 34, no. 1, pp. 204–217, 2011.

[20] R. F. Stengel, *Optimal control and estimation.* Courier Corporation, 2012.

[21] C. Eliasmith and C. H. Anderson, *Neural engineering: Computation, representation, and dynamics in neurobiological systems.* MIT press, 2004.

[22] T. C. Stewart and C. Eliasmith, "Large-scale synthesis of functional spiking neural circuits," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 881–898, 2014.

[23] W. Gerstner and W. Kistler, "Spiking neuron models cambridge university press," 2002.

[24] R. M. Sanner and J.-J. Slotine, "Gaussian networks for direct adaptive control," *IEEE Transactions on neural networks*, vol. 3, no. 6, pp. 837–863, 1992.

[25] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *The international journal of robotics research*, vol. 6, no. 3, pp. 49–59, 1987.