LABORATORY FOR INTELLIGENT SYSTEMS AND CONTROLS

An Adaptive Spiking Neural Controller for Flapping Insect-scale Robots

Taylor S. Clawson[1], Terrence C. Stewart[2], Chris Eliasmith[2], Silvia Ferrari[1]

[1] Department of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY
[2] Centre for Theoretical Neuroscience, University of Waterloo, Waterloo, Ontario, Canada

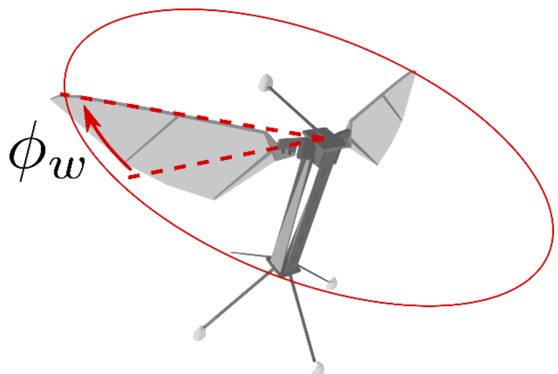
IEEE SSCI 2017

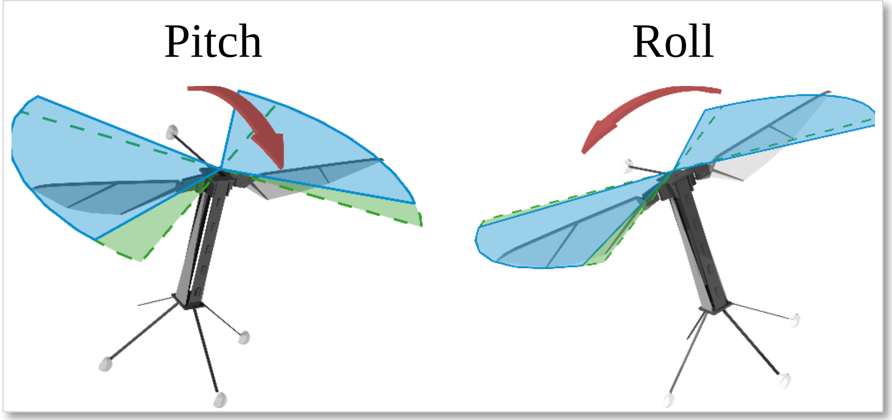
Nov 29, 2017

Introduction

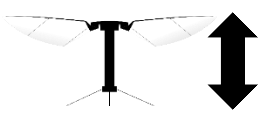
RoboBee Background

- Wing stroke angle ϕ_w controlled independently for each wing
- Thrust and body torques controlled by modulating stroke angle commands





100 mg



14 mm

120 Hz

Video Credit: [Ma K.Y., ’13]

Video of RoboBee test flight courtesy of the Harvard Microrobotics Lab

Introduction and Motivation

- Applications
  - Navigation in cluttered environments, requiring rapid precise feedback
  - Remote sensing using low power on-board sensors
- Research Goals
  - Adapt to unmodeled dynamics to control steady maneuvers
  - Integrate spiking controller with event-based sensors

- Previous work
  - Wind gust disturbance rejection [Chirarattananon, P. ’17]
    - Adaptive control to reduce error in constant wind gusts
  - Hovering control of simplified 2D model with SNN [Clawson, T. ’16]
    - Use Spiking Neural Network (SNN) to stabilize simplified 2D flight model
  - Other flapping-wing robots and theoretical developments
    - [De Croon, G. C. H. E. ’09], [Chang, S. ‘14], [Wu, J. H. ’12]

States and Control

$$\mathbf{x} = \begin{bmatrix} \Theta_r^T & \Theta_l^T & \dot{\Theta}_r^T & \dot{\Theta}_l^T & \Theta^T & \mathbf{r}^T & \dot{\Theta}^T & \dot{\mathbf{r}}^T \end{bmatrix}^T$$

$$\mathbf{u} = \begin{bmatrix} u_a & u_p & u_r \end{bmatrix}^T$$

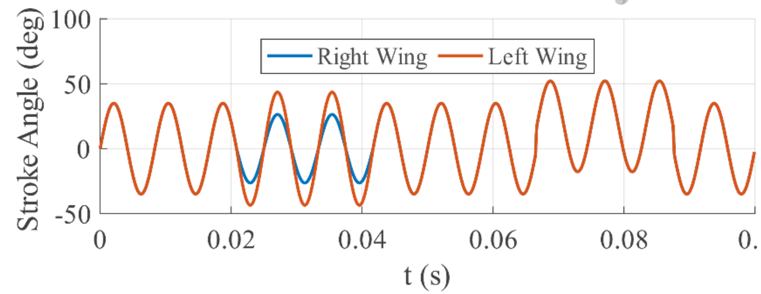
Stroke angle trajectory ϕ_w modeled as a function of input $\boldsymbol{u}$ following linear second-order system:

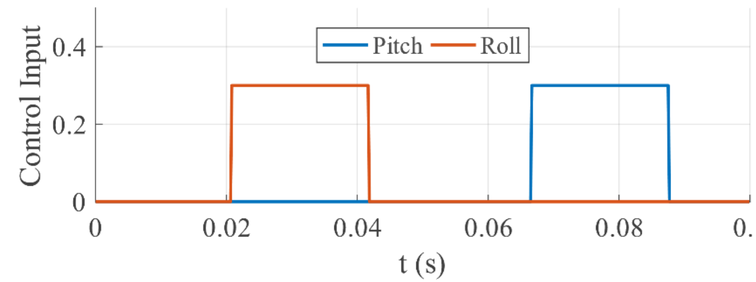
$$\ddot{\phi}_w(t) + 2\zeta\omega_n\dot{\phi}_w(t) + \omega_n^2\phi_w(t) = A_w \sin(\omega_f t) + b_w$$

For the right wing, for example,

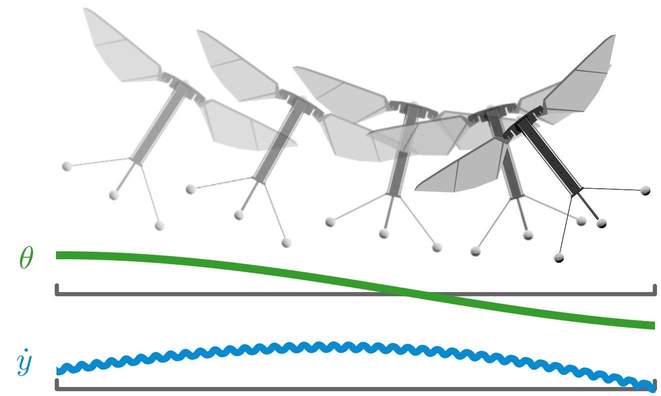
$$A_w = u_a - \frac{u_r}{2}, \quad b_w = -u_p$$

$\boldsymbol{x}$	State	ϕ_w	Wing stroke angle
$\boldsymbol{u}$	Control Input	ϕ_0	Nominal stroke amplitude
$\boldsymbol{\Theta}$	Body orientation	ϕ_p	Pitch input
$\boldsymbol{r}$	Body position	ϕ_r	Roll input
$\boldsymbol{\Theta}_r$	Right wing orientation	A_w	Wing stroke amplitude
ω_f	Flapping frequency	$\bar{\phi}_w$	Mean stroke angle



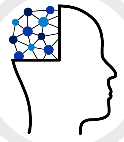


Control Challenges



- Hovering flight is unstable in both pitch ψ and roll θ
  - With non-zero velocity, drag acting on wings tilts the robot away from the current direction of travel [Wu, H. J. '12], [Ristroph, L. '13]
  - Tumbling occurs after approx 200-300 ms in open loop flight
- High state space dimensionality – 12 for body and additional 12 for wings
- Low power budget: < 5 mW for sensing and control

Adaptive Spiking Neural Network (SNN)



Leaky Integrate and Fire (LIF) Model

- Models the voltage *V(t)* across the membrane of a neuron as an RC circuit with resistance *R*, time constant τ_m, and input current *I(t)*

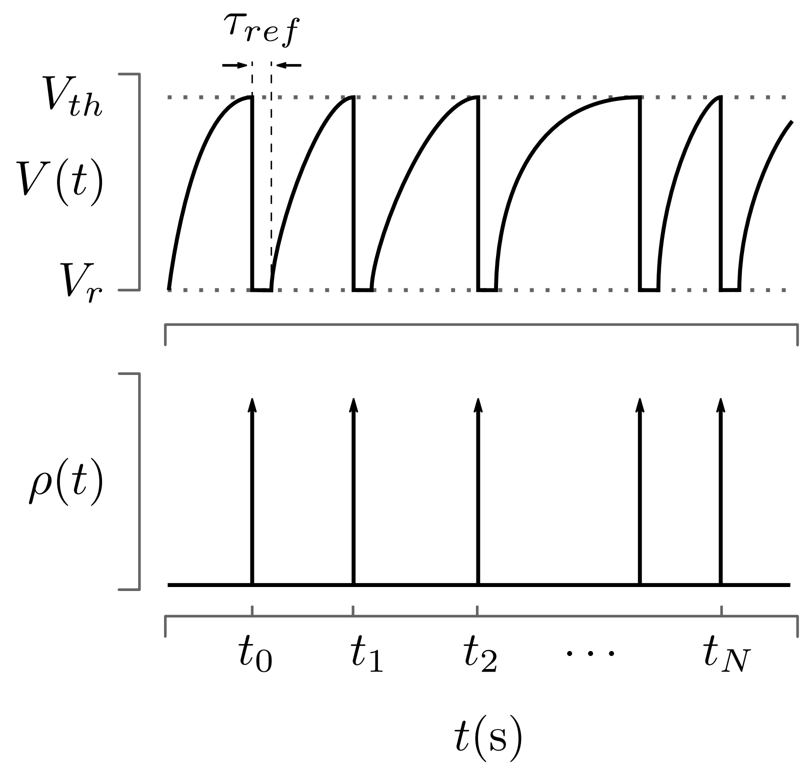
$$\tau_m \frac{dV}{dt} = -V(t) + RI(t)$$

- Model used to obtain spike times t_k when *V* reaches threshold V_{th}

$$t_k : \;\; V(t_k) = V_{th}$$

- After a spike, *V(t)* is reset to V_r for a refractory period τ_{ref}
- Output of the neuron is a spike train ρ, modeled as a series of Dirac Delta functions δ at the spike times

$$\rho(t) = \sum_k \delta(t - t_k)$$



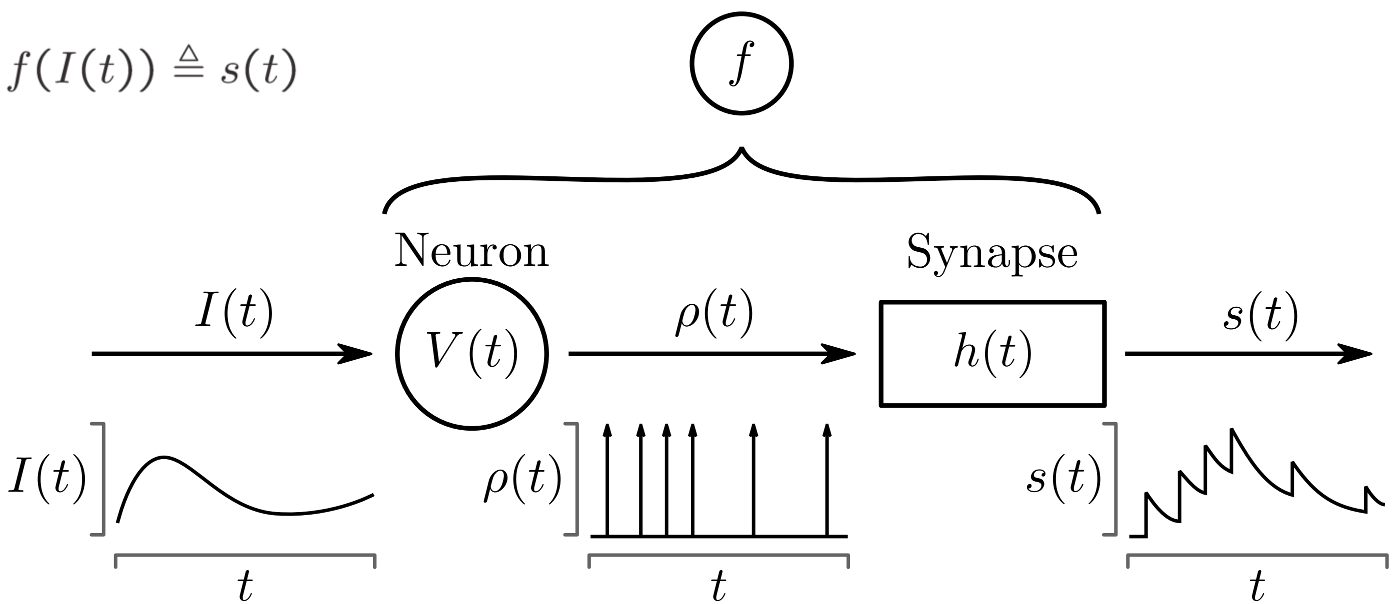
Neuron and Synapse

- The output from each neuron is filtered by a synapse with kernel *h(t)*, resulting in a postsynaptic current *s(t)*

$$h(t) = \frac{1}{\tau_s} e^{-t/\tau_s} \qquad s(t) = \int_0^t h(t-\tau)\rho(\tau)d\tau = \sum_{k=1}^{M} h(t - t_k)$$

- Together, the neuron and synaptic models form a nonlinear mapping *f* from the input current *I(t)* to the postsynaptic current *s(t)*

$$f(I(t)) \triangleq s(t)$$



Single Layer Feedforward SNN

- A single layer feedforward SNN forms the nonlinear mapping *F* between the vector of input current ***I(t)*** to the vector of postsynaptic currents ***s(t)***

$$\mathrm{s}(t) = F(\mathbf{I}(t))$$

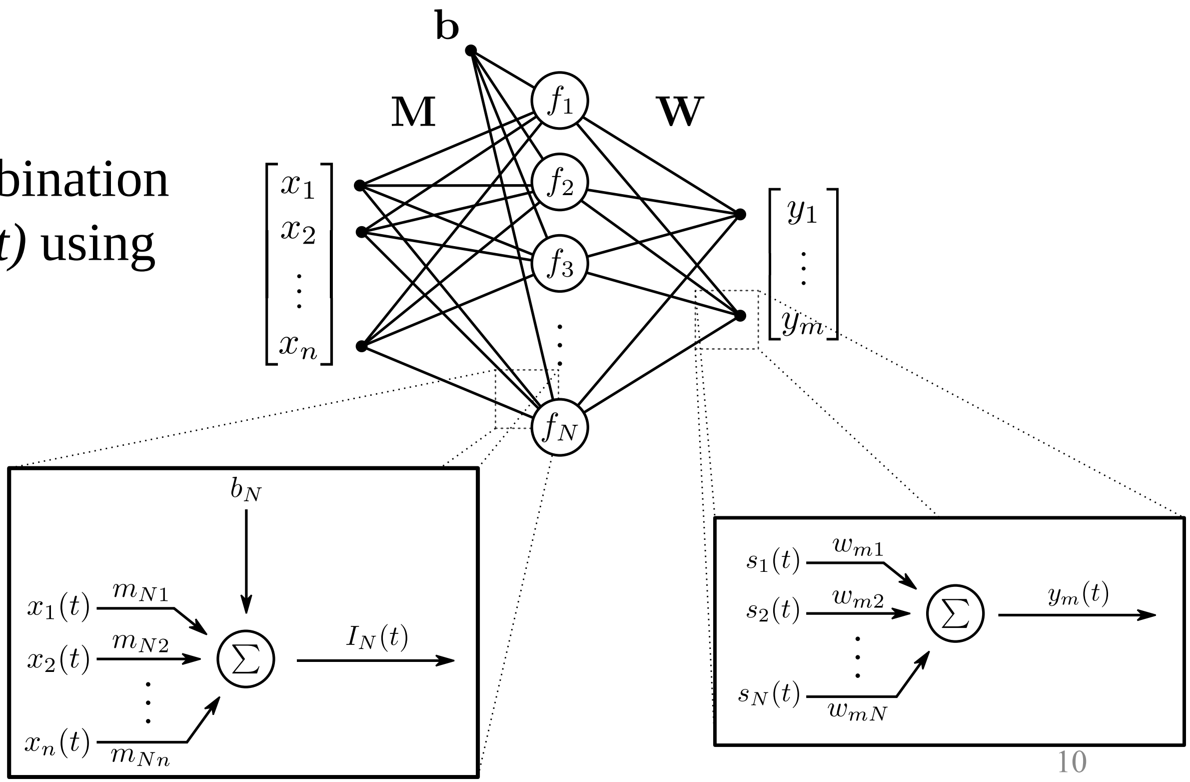
- Input current ***I(t)*** a function of input weights ***M***, input bias ***b***, and input ***x***

$$\mathbf{I}(t) = \mathbf{M}\mathrm{x}(t) + \mathrm{b}$$

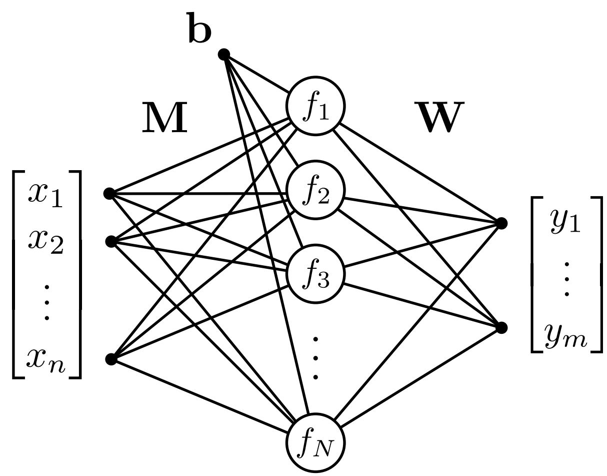
- Output ***y(t)*** is a linear combination of postsynaptic currents ***s(t)*** using output weights ***W***

$$\mathrm{y}(t) = \mathbf{W}\mathrm{s}(t)$$

- Linear combination effectively extracts information [Salinas, E. ’94], [Eliasmith, C. ’04]



Function Approximation



- The network output is

$$\mathbf{y}(t) = \mathbf{W}\mathbf{s}(t) = \mathbf{W}F(\mathbf{M}\mathbf{x}(t) + \mathbf{b})$$

- By tuning the output weights, the network can be trained to approximate a nonlinear function ***f(x)*** using least-squares optimization over a set of training data indexed by *j*

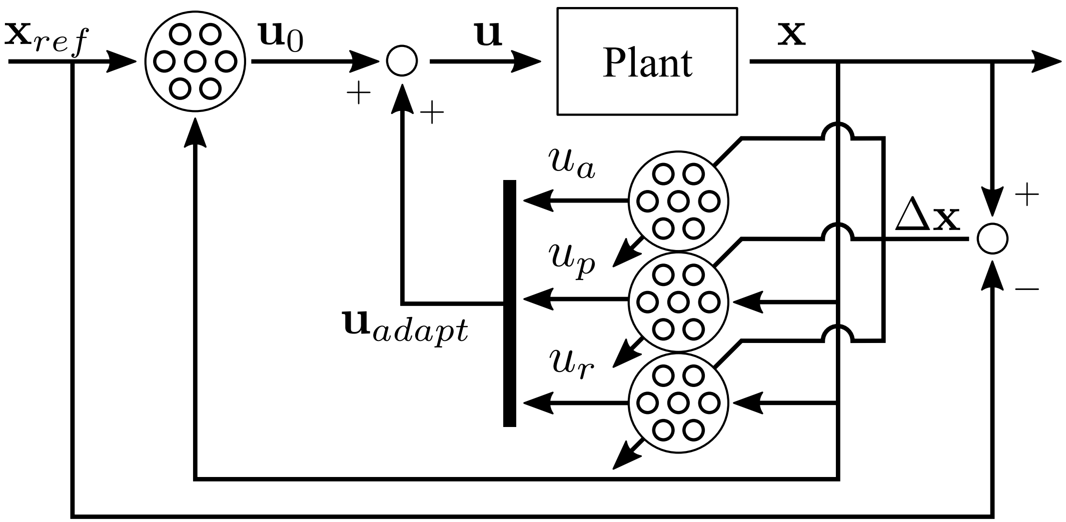
$$\mathbf{W}_f = \underset{\mathbf{W}}{\mathrm{argmin}} \sum_j \left\| \mathbf{f}(\mathbf{x}_j) - \mathbf{W}F(\mathbf{M}\mathbf{x}_j + \mathbf{b}) \right\|^2$$

- Using these weights, the output is

$$\mathbf{y}(t) = \mathbf{W}_f\mathbf{s}(t) \approx \mathbf{f}(\mathbf{x}(t))$$

Adaptive SNN Controller

Control Architecture



- Control signal ***u(t)*** provided entirely by feedforward networks of neurons

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

- Non-adaptive term $\boldsymbol{u}_0(t)$ trained offline to approximate a precomputed stabilizing control law
- Adaptive term $\boldsymbol{u}_{adapt}(t)$ adapts online to compensate for unmodeled dynamics

$\boldsymbol{x}_{ref}$	Reference state
$\boldsymbol{u}_0$	Non-adaptive control input
$\boldsymbol{u}_{adapt}$	Adaptive control input
$\boldsymbol{\Delta x}$	State error
u_a	Amplitude input
u_p	Pitch input
u_r	Roll input

Non-Adaptive Term

- Non-adaptive term $\boldsymbol{u}_0(t)$ is computed from a single-layer feedforward network of 500 neurons
- Approximates signal from a Proportional-Integral-Filter (PIF) Compensator, $\boldsymbol{u}_{PIF}(t)$, which guarantees stability of the linearized plant and follows

$$\dot{\mathbf{u}}_{PIF}(t) = -\mathbf{K}\boldsymbol{\chi}(t)$$

- Based on constant gain $\boldsymbol{K}$ and augmented state vector,

$$\boldsymbol{\chi}(t) = [\tilde{\mathbf{x}}^T(t)\ \tilde{\mathbf{u}}^T(t)\ \boldsymbol{\xi}^T(t)]$$

- Augmented state includes the state deviation, control deviation, and integral of the output deviation:

$$\tilde{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}^* \qquad \tilde{\mathbf{u}}(t) = \mathbf{u}(t) - \mathbf{u}^* \qquad \boldsymbol{\xi}(t) = \boldsymbol{\xi}(0) + \int_0^t \tilde{\mathbf{y}}(\tau)d\tau$$

- The PIF control law guarantees stability of the linearized plant
- SNN approximation of PIF is obtained using least-squares as shown before, so that

$$\mathbf{u}_0(t) \approx \mathbf{u}_{PIF}(t)$$

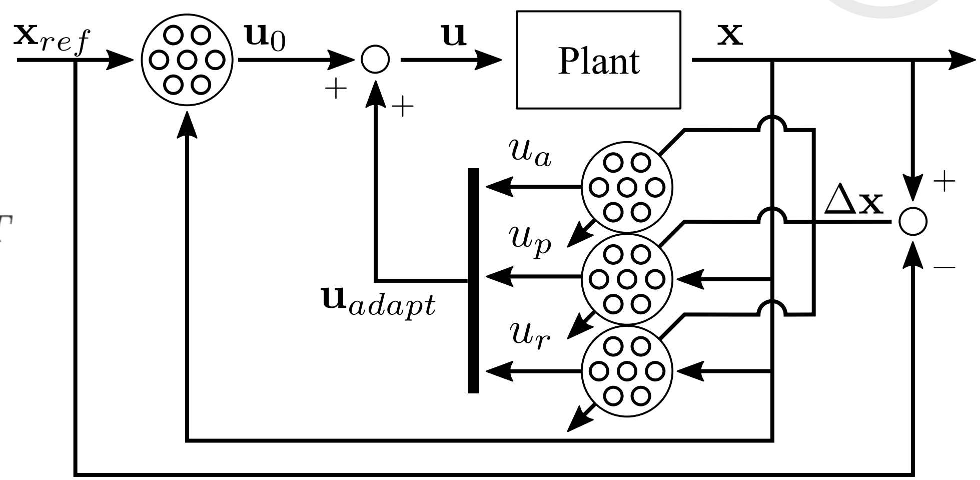
Adaptive Term

- Adaptive term $\boldsymbol{u}_{adapt}$ contains inputs for amplitude u_a, pitch u_p, roll u_r

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

- Output weights adjusted online to minimize an error E

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



- Each scalar in $\boldsymbol{u}_{adapt}$ computed from a single network of 100 neurons, e.g.

$$u_a = \mathbf{W}_a(t)\mathbf{s}_a(t)$$

- Where the connection weights are updated online according to

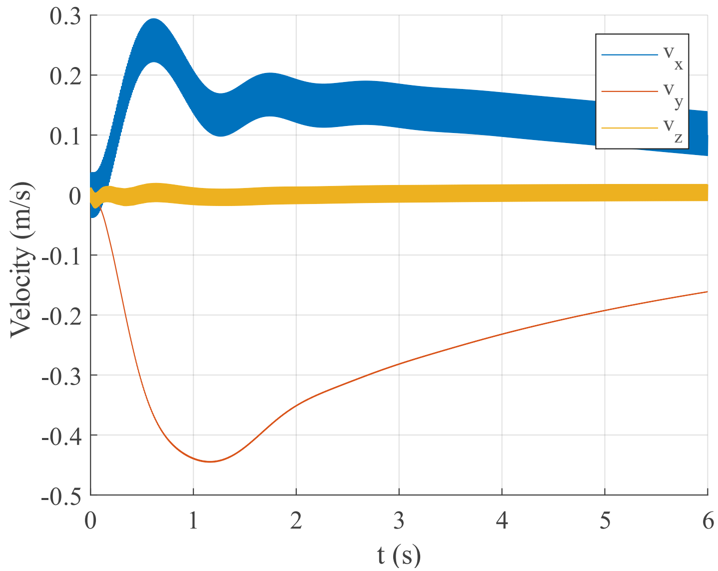
$$\dot{\mathbf{W}}_a(t) = \gamma\mathbf{s}_a(t)E(t)$$

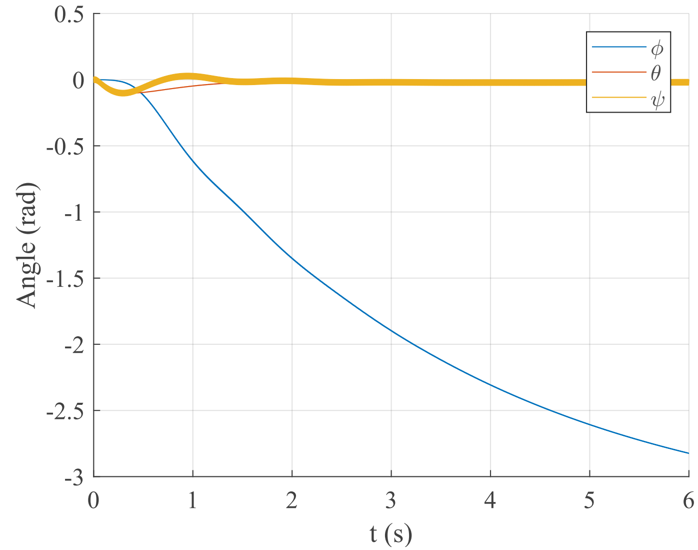
- Each adaptive network minimizes different state error

Control Input	Minimized State Error
Amplitude, u_a	Body z-velocity, w
Pitch, u_p	Body x-velocity, u
Roll, u_r	*Body y-velocity*, v

Results

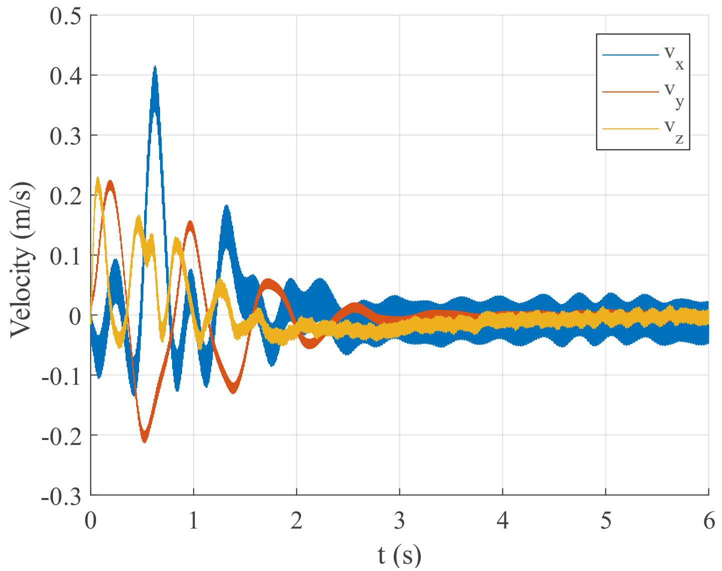
PIF Controlled Robot

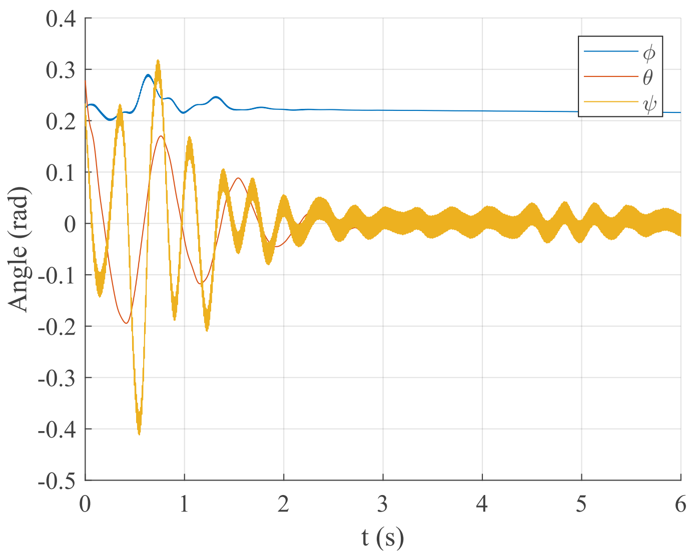




- PIF Compensator commanded to control hovering flight for 6 seconds with a simulated wing asymmetry
- Integral term acts slowly to stabilize the robot, causing significant positional drift over time

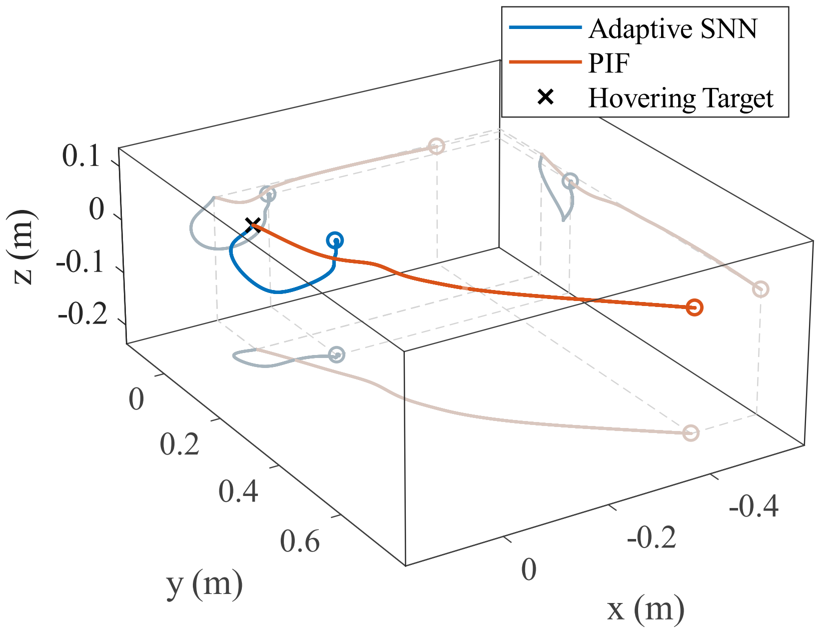
SNN Controlled Robot

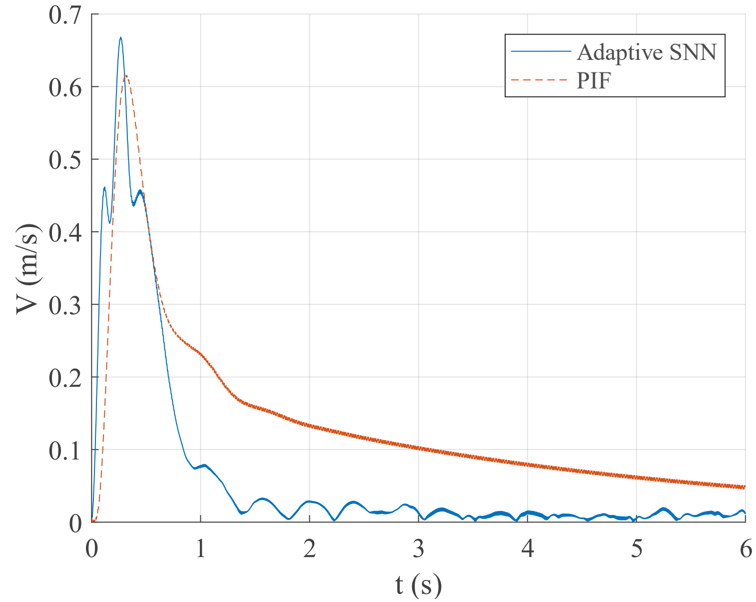




- Adaptive SNN commanded to control hovering flight for 6 seconds with a simulated wing asymmetry
- Adaptive input accounts for wing bias and stabilizes velocity, roll, and pitch near zero after ~3 seconds

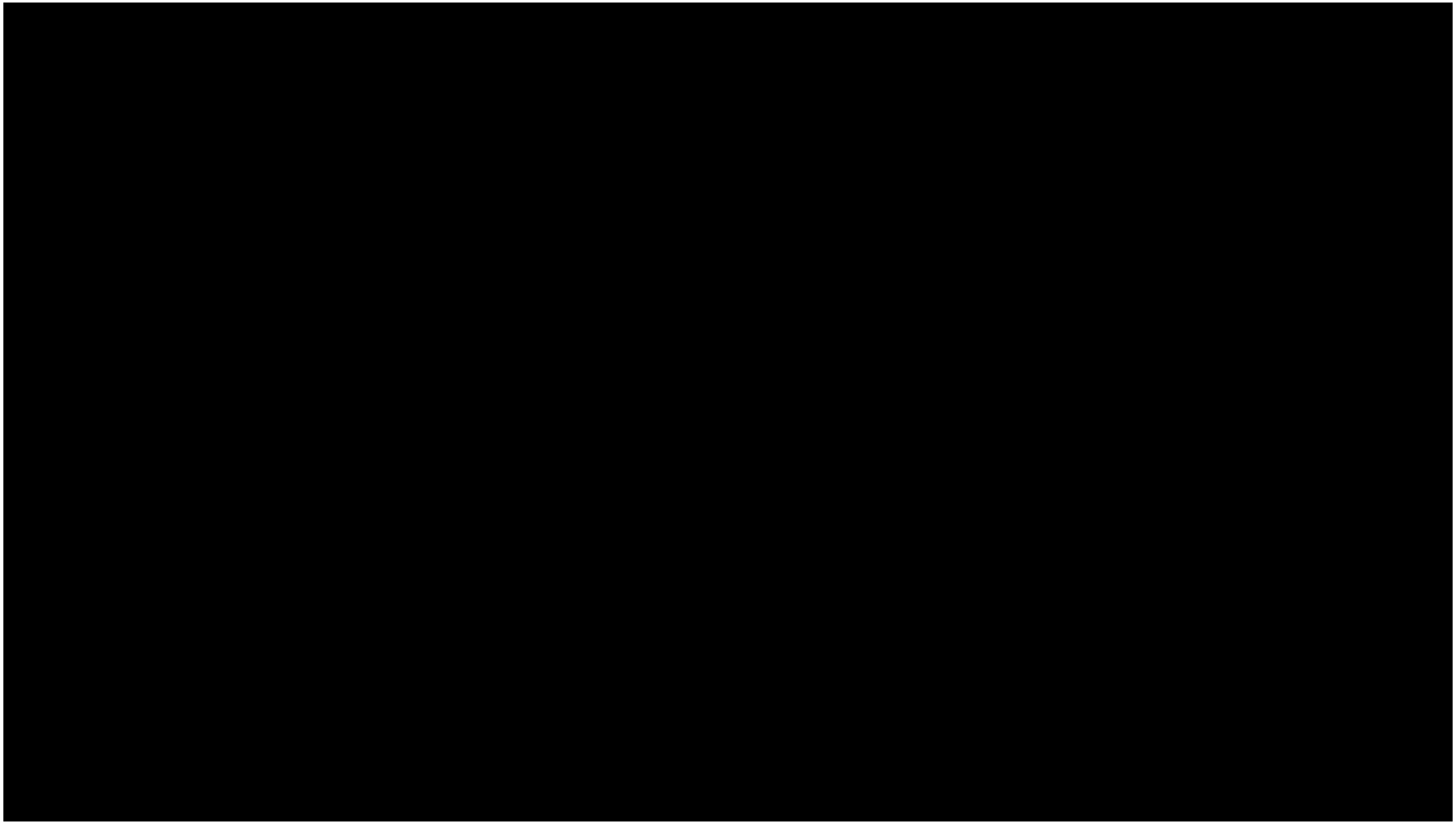
Trajectory Comparison





- Closed-loop response of the system in the presence of asymmetries in the wings
- Wing asymmetries result in static non-zero pitch and roll biases
- Adaptive SNN compensates more quickly and maintains hovering position much closer to hovering target
- PIF compensator drifts significantly from hovering target

SNN Controlled Robot



Conclusion

- Demonstrated that an adaptive SNN is a viable control method for stabilizing RoboBee flight
- Adaptive SNN quickly learns to compensate for parametric variations to stabilize hovering flight
- Future Work
  - Include critic network for ADP techniques
  - Control non-hovering maneuvers
  - Integrate with event-based sensors on the phyisical RoboBee

An Adaptive Spiking Neural Controller for Flapping Insect-scale Robots

Taylor S. Clawson[1], Terrence C. Stewart[2],

Chris Eliasmith[2], Silvia Ferrari[1]

[1] Department of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY

[2] Centre for Theoretical Neuroscience, University of Waterloo, Waterloo, Ontario, Canada

Further questions: Taylor Clawson

tsc83@cornell.edu

Related Work

Clawson, Taylor S., et al. "Spiking neural network (SNN) control of a flapping insect-scale robot." *Decision and Control (CDC), 2016 IEEE 55th Conference on*. IEEE, 2016. [PDF]

Clawson, Taylor S., et al. "A Blade Element Approach to Modeling Aerodynamic Flight of an Insect-scale Robot," *American Control Conference (ACC)*, Seattle, WA, May 2017. [PDF]