ECE 780 To 8 Project
Bug Algorithms using Neural Networks

Andreas Stöckel

Computational Neuroscience Research Group
Centre for Theoretical Neuroscience

UNIVERSITY OF WATERLOO

July 24th, 2017

Motivation

- In class, we mostly talked about path planning using *global* information.
- Often, robotic and biological agents alike are restricted to *local*, *sensor-based* information.

Source: Wikimedia Commons

Motivation

- In class, we mostly talked about path planning using *global* information.
- Often, robotic and biological agents alike are restricted to *local*, *sensor-based* information.
- *Bug algorithms* (Lumelsky and Stepanov 1987) are a simplistic class of local planing algorithms/policies.

Source: Wikimedia Commons

Motivation

- In class, we mostly talked about path planning using *global* information.
- Often, robotic and biological agents alike are restricted to *local*, *sensor-based* information.
- *Bug algorithms* (Lumelsky and Stepanov 1987) are a simplistic class of local planing algorithms/policies.

⇝ Can these algorithms be implemented in a biological, neural substrate?

Source: Wikimedia Commons

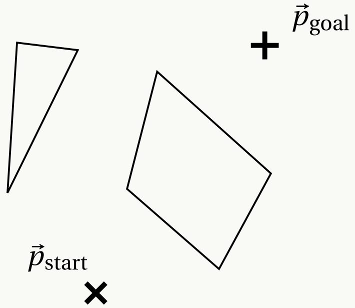
PART I

Bug Algorithms

Bug Algorithm Assumptions

Environment

- Start location $\vec{p}_{\text{start}}$, goal location $\vec{p}_{\text{goal}}$
- Polygonal obstacles $O_1, \ldots, O_n$



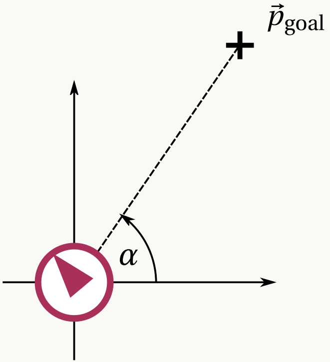
Bug Algorithm Assumptions

Environment

- Start location $\vec{p}_{\text{start}}$, goal location $\vec{p}_{\text{goal}}$
- Polygonal obstacles $O_1, \ldots, O_n$

Robot

- Knows direction towards goal (absolute and relative)



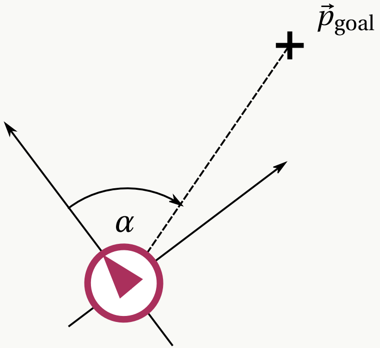
Bug Algorithm Assumptions

Environment

- Start location $\vec{p}_{\text{start}}$, goal location $\vec{p}_{\text{goal}}$
- Polygonal obstacles $O_1, \ldots, O_n$

Robot

- Knows direction towards goal (absolute and relative)



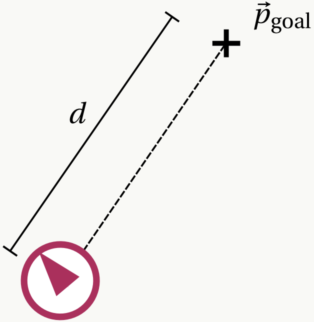
Bug Algorithm Assumptions

Environment

- Start location $\vec{p}_{\text{start}}$, goal location $\vec{p}_{\text{goal}}$
- Polygonal obstacles $O_1, \ldots, O_n$

Robot

- Knows direction towards goal (absolute and relative)
- Knows the straight line distance to the goal



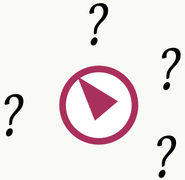
Bug Algorithm Assumptions

Environment

- Start location $\vec{p}_{\text{start}}$, goal location $\vec{p}_{\text{goal}}$
- Polygonal obstacles $O_1, \ldots, O_n$

Robot

- Knows direction towards goal (absolute and relative)
- Knows the straight line distance to the goal
- Has no knowledge of obstacles (no map)



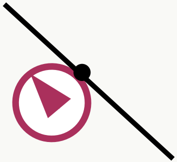
Bug Algorithm Assumptions

Environment

- Start location $\vec{p}_{\text{start}}$, goal location $\vec{p}_{\text{goal}}$
- Polygonal obstacles $O_1, \ldots, O_n$

Robot

- Knows direction towards goal (absolute and relative)
- Knows the straight line distance to the goal
- Has no knowledge of obstacles (no map)
- Has a contact sensor



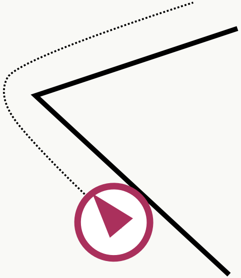
Bug Algorithm Assumptions

Environment

- Start location $\vec{p}_{\text{start}}$, goal location $\vec{p}_{\text{goal}}$
- Polygonal obstacles $O_1, \ldots, O_n$

Robot

- Knows direction towards goal (absolute and relative)
- Knows the straight line distance to the goal
- Has no knowledge of obstacles (no map)
- Has a contact sensor
- Moves in straight lines or along obstacle boundaries



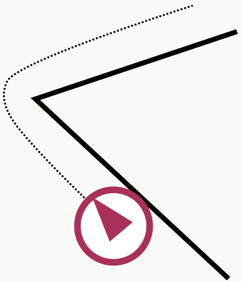
Bug Algorithm Assumptions

Environment

- Start location $\vec{p}_{\text{start}}$, goal location $\vec{p}_{\text{goal}}$
- Polygonal obstacles $O_1, \ldots, O_n$

Robot

- Knows direction towards goal (absolute and relative)
- Knows the straight line distance to the goal
- Has no knowledge of obstacles (no map)
- Has a contact sensor
- Moves in straight lines or along obstacle boundaries
- Has memory to store distances and angles



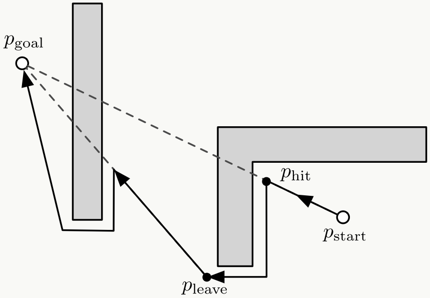
media/video/demo_bug_direct_success.mp4

media/video/demo_bug_direct_fail.mp4

media/video/demo_bug_0_success.mp4

The Bug 0 algorithm

```
while not at goal do
    move towards the goal
    if hit an obstacle then
        while not able to move towards the goal do
            follow obstacle boundary CCW
        end while
    end if
end while
```



Source: Lectures on Robotic Planning and Kinematics, Bullo and Smith, 2016

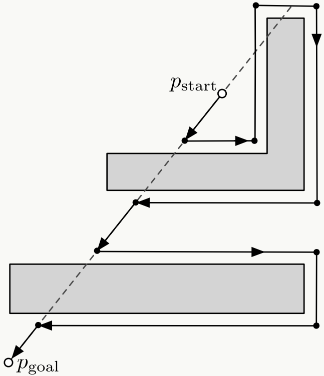
- ⊕ Algorithm solely depends on goal direction
- ⊖ The algorithm may fail...

media/video/demo_bug_0_fail.mp4

The Bug 2 algorithm

$\alpha \leftarrow$ goal direction
while not at goal **do**
    move towards the goal
    if hit an obstacle **then**
        $d_{\text{hit}} \leftarrow$ distance to goal
        while distance to goal $\geq d_{\text{hit}}$ **and**
                goal direction $\neq \alpha$ **do**
            follow obstacle boundary
        end while
    end if
end while

- ⊕ If $\vec{p}_{\text{goal}}$ is reachable from $\vec{p}_{\text{start}}$, the algorithm will find a path
- ⊖ Not always better than Bug 1

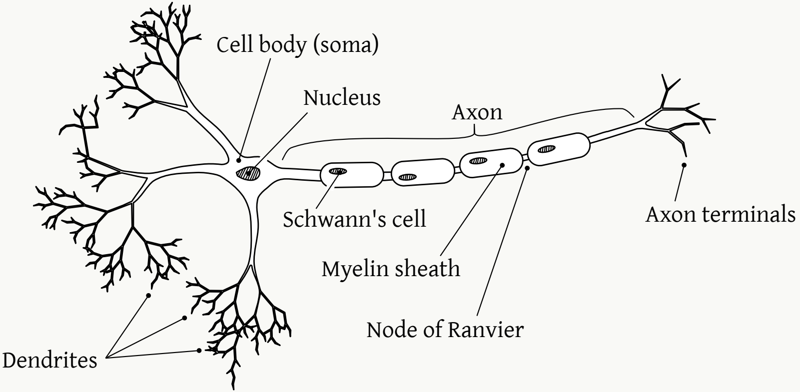


Source: Lectures on Robotic Planning and Kinematics, Bullo and Smith, 2016 `media/video/demo_bug_2.mp4`

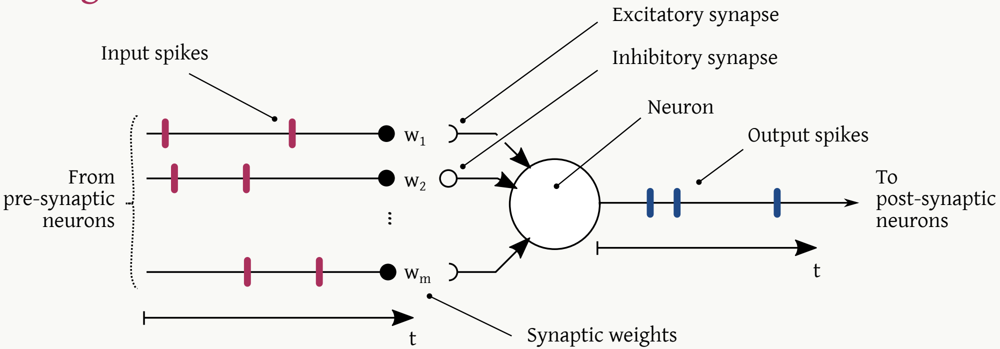
PART II

Spiking Neural Networks and the Neural Engineering Framework

Textbook Biological Model Neuron



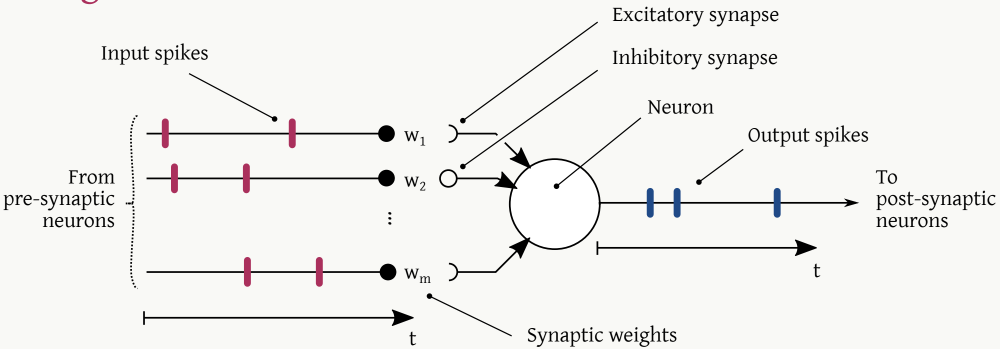
Leaky Integrate and Fire Neuron



- Input spikes induce current J in cell body
- Cell body modelled as dynamical system in continuous time

$$\tau \dot{u}(t) = J - u(t) \qquad\qquad u(t) \leftarrow 0 \text{ if } u(t) = 1$$

Leaky Integrate and Fire Neuron



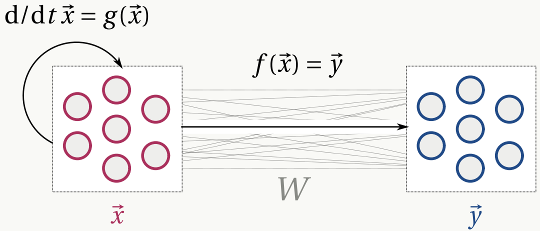
- Input spikes induce current J in cell body
- Cell body modelled as dynamical system in continuous time

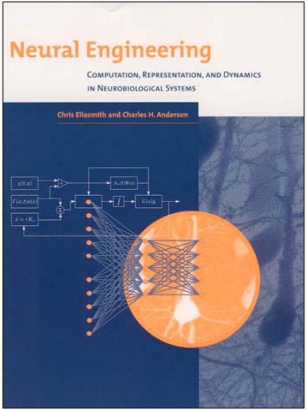
$$\tau \dot{u}(t) = J - u(t) \qquad\qquad u(t) \leftarrow 0 \text{ if } u(t) = 1$$

⚠ Not your usual artificial neural network! Intrinsic dynamics! Spike noise!

The Neural Engineering Framework (NEF)

- Systematic way of building spiking neural networks
- *Principle 1:*
  Populations of spiking neurons represent $\vec{x}$
- *Principle 2:*
  Connections between populations compute $f(\vec{x}) = \vec{y}$
- *Principle 3:*
  Self-connections implement dynamics $\mathrm{d}/\mathrm{d}t\,\vec{x} = g(\vec{x})$

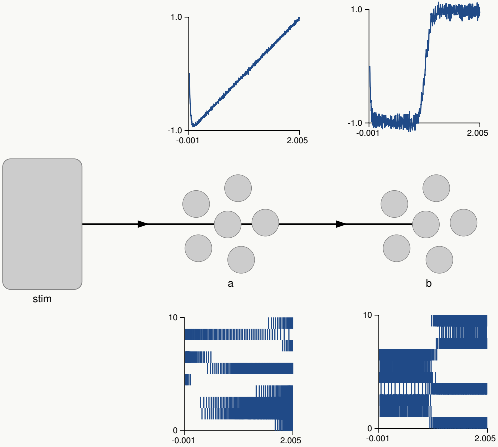




Chris Eliasmith and Charles H. Anderson, Neural Engineering, 2003

Nengo

- Nengo: Python software implementation of the NEF, available on GitHub
  (Bekolay et al., Nengo: A Python tool for building large-scale functional brain models, Frontiers in Neuroinformatics, 2014)




```python
import nengo
import numpy as np

model = nengo.Network()
with model:
    stim = nengo.Node(
        lambda t: t - 1)

    a = nengo.Ensemble(
        n_neurons=400, dimensions=1)
    b = nengo.Ensemble(
        n_neurons=400, dimensions=1)

    nengo.Connection(stim, a)
    nengo.Connection(a, b,
        function=np.sign)
```

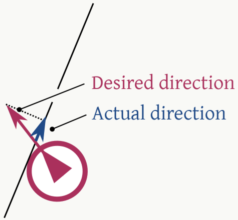
PART III

Implementation and Results

Simulator Environment

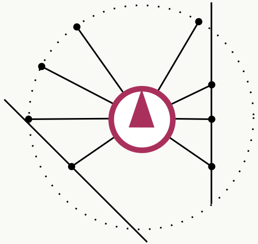
Body & Environment

- Discrete simulation $\Delta t = 10\,\mathrm{ms}$
- Disk-shaped robot
- Robot slides along obstacles



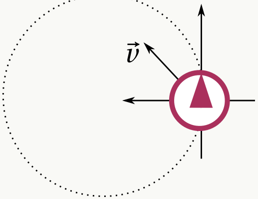
Sensors

- Distance d, contact sensor
- Absolute and relative orientation (unit vectors $\vec{\alpha}_\mathrm{a}, \vec{\alpha}_\mathrm{r}$)
- Radar for obstacle boundary following



Motor System

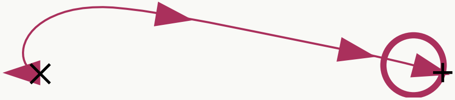
- Non-holonomic 2-DOF drive
- Relative control vector $\vec{v} = (v_\mathrm{x}, v_\mathrm{y})$



Basic Behaviours: Move Towards Goal & Follow Obstacle

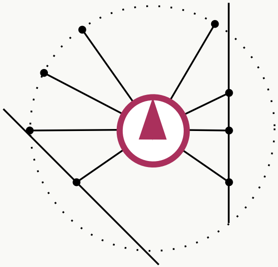
Move Towards Goal

- Connect relative direction vector $\vec{\alpha}_r$ to the motor control output $\vec{v}$



Follow Obstacle

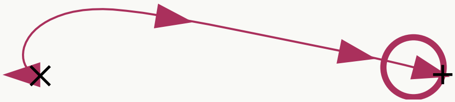
- Rotate radar vectors by $\approx 45°$
- Weighted sum of radar vectors; smaller distance, more weight



Basic Behaviours: Move Towards Goal & Follow Obstacle

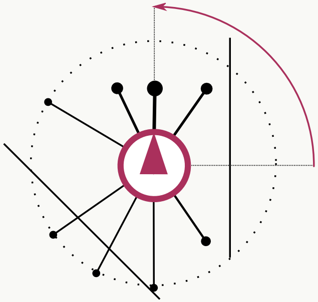
Move Towards Goal

- Connect relative direction vector $\vec{\alpha}_r$ to the motor control output $\vec{v}$



Follow Obstacle

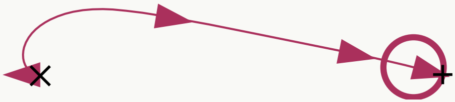
- Rotate radar vectors by $\approx 45°$
- Weighted sum of radar vectors; smaller distance, more weight



Basic Behaviours: Move Towards Goal & Follow Obstacle

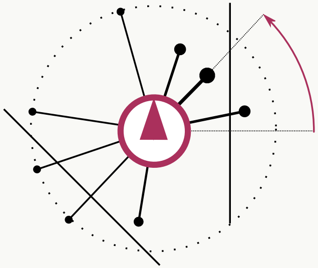
Move Towards Goal

- Connect relative direction vector $\vec{\alpha}_r$ to the motor control output $\vec{v}$



Follow Obstacle

- Rotate radar vectors by $\approx 45°$
- Weighted sum of radar vectors; smaller distance, more weight

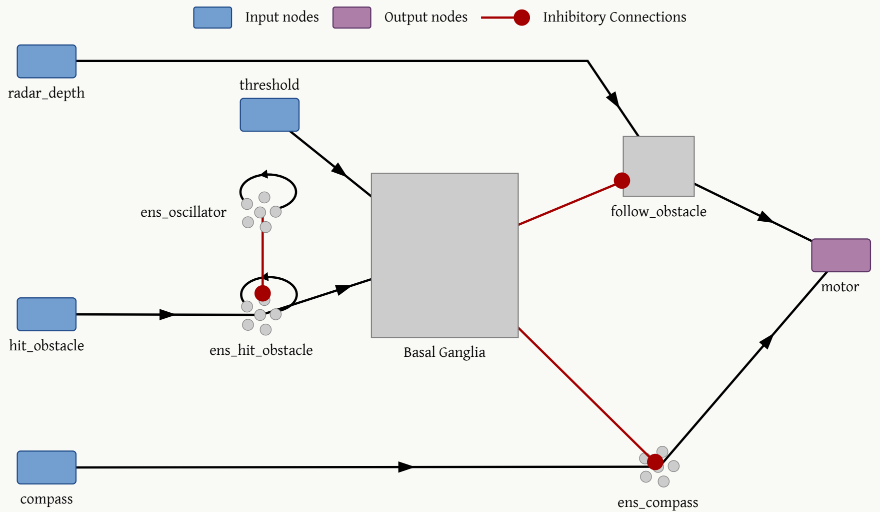


media/video/demo_bug_follow.mp4

Bug 0: Reference Implementation

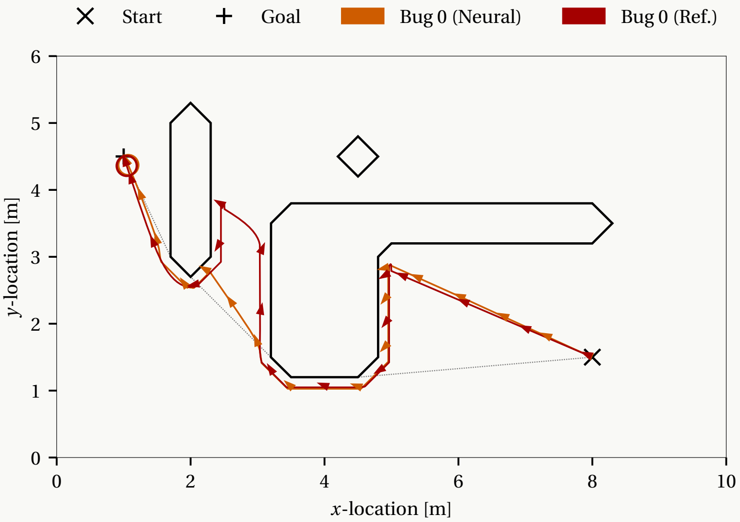
```python
def behave(self, sensors, motor):
    # State transition
    if self.t_follow > 1.0:
        self.follow_obstacle *= 0.5
    elif sensors.hit_obstacle():
        self.follow_obstacle += 1.0 * self.dt_coarse
    else:
        self.follow_obstacle -= self.follow_obstacle * self.dt_coarse

    # Behaviour implementation
    if self.follow_obstacle > 0.25:
        common.follow_obstacle(sensors, motor, self.radius)
        self.t_follow += self.dt_coarse
    else:
        self.t_follow = 0
        common.move_towards_goal(sensors, motor)
```

Bug 0: Neural Network

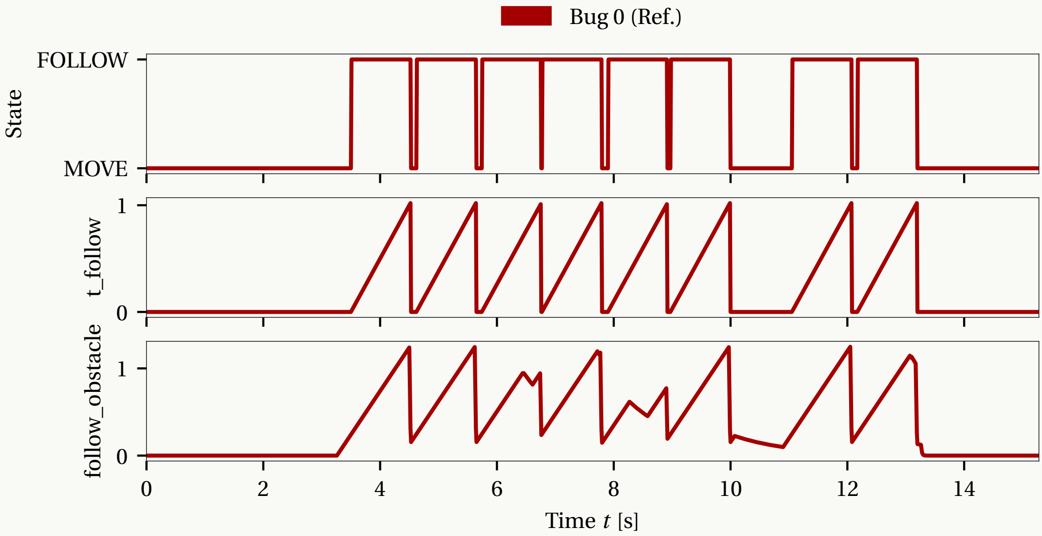


media/video/bug_0_comparison.mp4

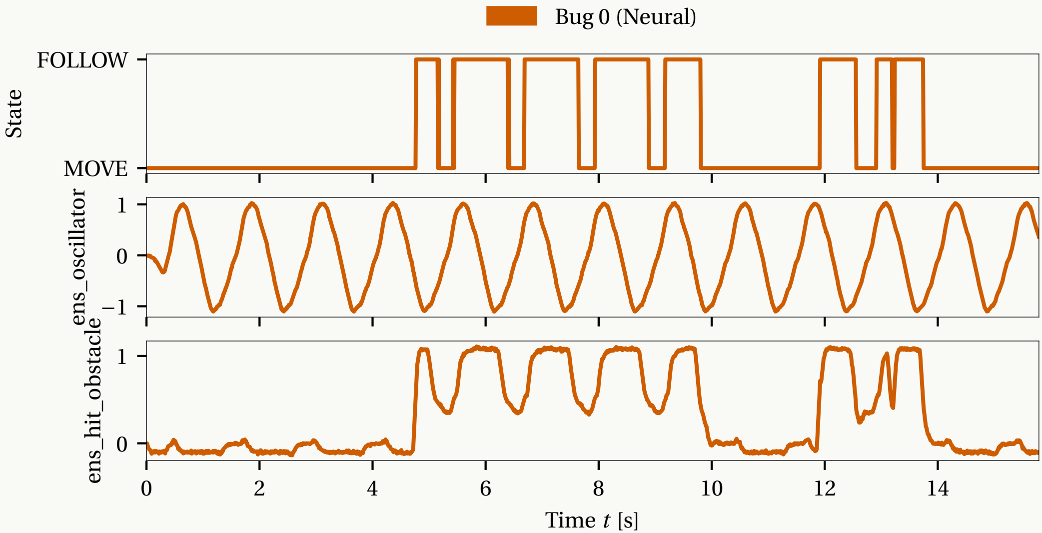
Bug 0: State over Time



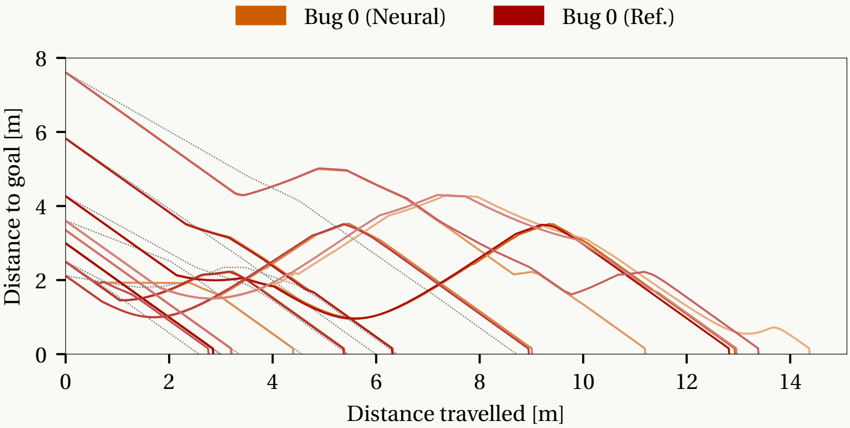
Bug 0: State over Time



Bug 0: State over Time



Bug 0: Distance Plot



Bug 2: Thinking About States

$\alpha \leftarrow$ goal direction
while not at goal **do**
  move towards the goal
  if hit an obstacle **then**
    $d_{\text{hit}} \leftarrow$ distance to goal
    while distance to goal $\geq d_{\text{hit}}$ **and**
        goal direction $\neq \alpha$ **do**
      follow obstacle boundary
    end while
  end if
end while

- *Simulator/Neural network:*
  Executes algorithm "tick-wise"

⇝ Translate to state machine

Bug 2: Thinking About States

$\alpha \leftarrow$ goal direction
while not at goal **do**
    move towards the goal
    if hit an obstacle **then**
        $d_{\text{hit}} \leftarrow$ distance to goal
        while distance to goal $\geq d_{\text{hit}}$ **and**
                goal direction $\neq \alpha$ **do**
            follow obstacle boundary
        end while
    end if
end while

States:

1. `MEM_DIR`: Memorize direction
2. `MOVE`: Move towards goal
3. `MEM_DIST`: Memorize distance
4. `FOLLOW`: Follow obstacle outline

- *Simulator/Neural network:*
  Executes algorithm “tick-wise”

$\rightsquigarrow$ Translate to state machine

Bug 2: Thinking About States

```
α ← goal direction
while not at goal do
    move towards the goal
    if hit an obstacle then
        d_hit ← distance to goal
        while distance to goal ≥ d_hit and
                goal direction ≠ α do
            follow obstacle boundary
        end while
    end if
end while
```

States:

1. `MEM_DIR`: Memorize direction
2. `MOVE`: Move towards goal
3. `MEM_DIST`: Memorize distance
4. `FOLLOW`: Follow obstacle outline

State Transition Table:

$$\epsilon \to 1, \quad 1 \to 2, \quad 2 \to 3, \quad 4 \to 2$$

- *Simulator/Neural network:*
  Executes algorithm "tick-wise"

⇝ Translate to state machine

Bug 2: Thinking About States

$\alpha \leftarrow$ goal direction
while not at goal **do**
    move towards the goal
    if hit an obstacle **then**
        $d_{\text{hit}} \leftarrow$ distance to goal
        while distance to goal $\geq d_{\text{hit}}$ **and**
                goal direction $\neq \alpha$ **do**
            follow obstacle boundary
        end while
    end if
end while

States:

1. MEM_DIR: Memorize direction
2. MOVE: Move towards goal
3. MEM_DIST: Memorize distance
4. FOLLOW: Follow obstacle outline

State Transition Table:

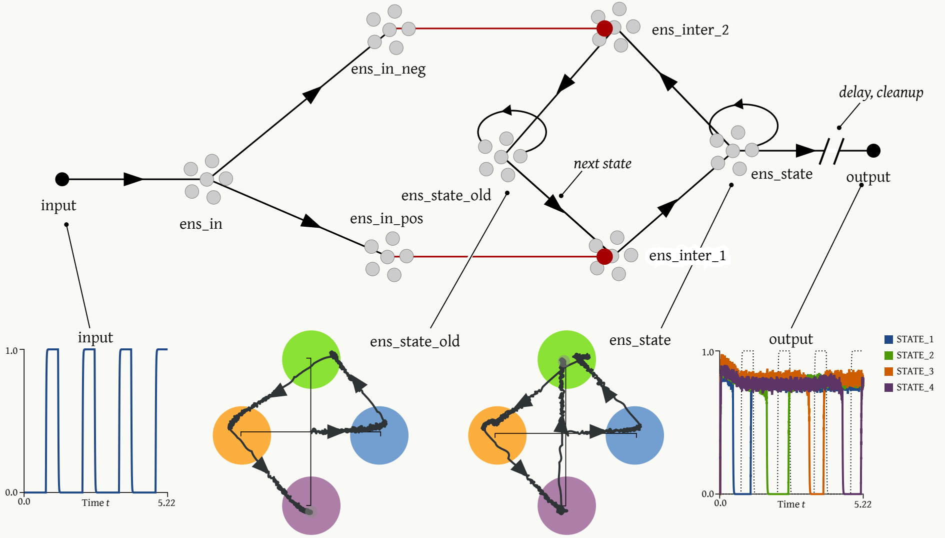
$$\epsilon \to 1, \quad 1 \to 2, \quad 2 \to 3, \quad 4 \to 2$$

- *Simulator/Neural network:* Executes algorithm "tick-wise"

⇝ Translate to state machine

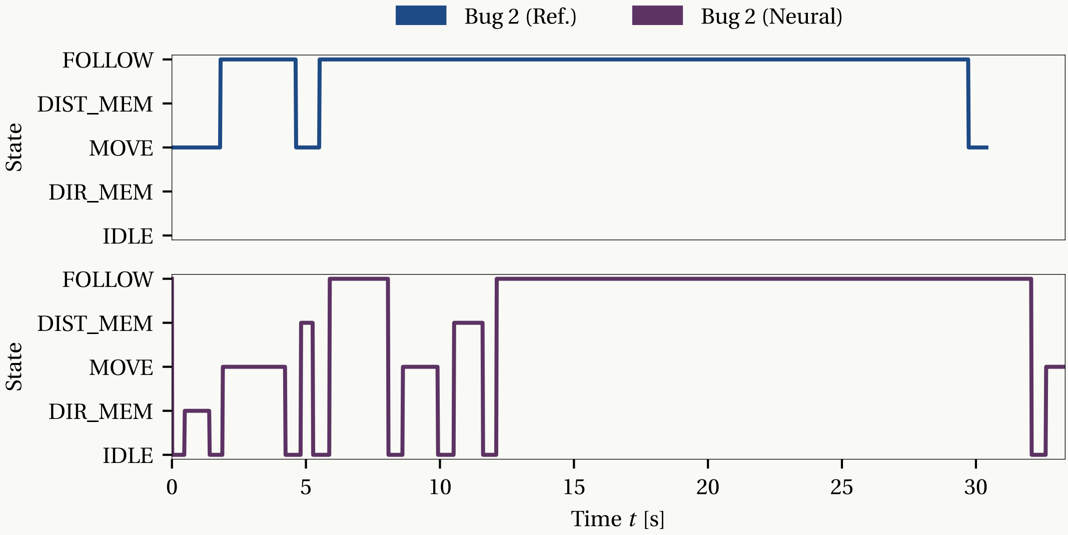
⇝ How to implement this as a spiking neural network?

Bug 2: State Transition Network

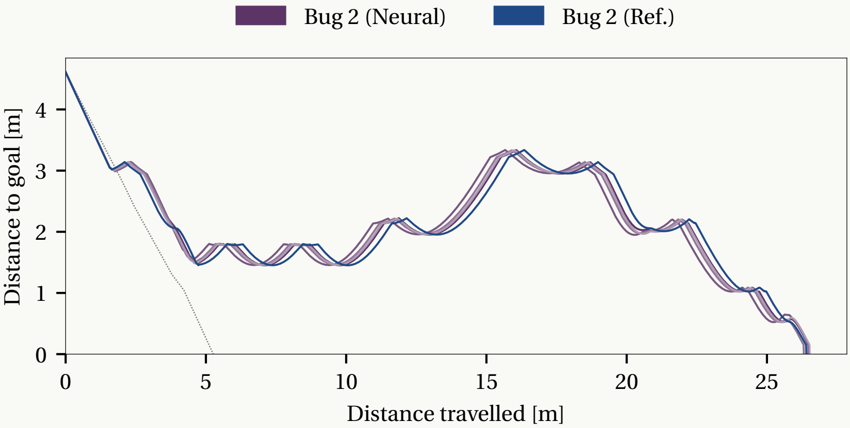


media/video/bug_2_comparison.mp4

Bug 2: State over Time



Bug 2: Distance Plot



Conclusion

Biological Plausibility

- Given the sensors, neurobiological implementation plausible, only few thousand neurons required
- Many of the sensors map to biological systems:
  - radar $\leftrightarrow$ structure from motion
  - contact $\leftrightarrow$ tactile sensors
  - distance and direction $\leftrightarrow$ olfactory system in insects

⚠ Yet, the given implementation must not be seen as a biological model!

Conclusion

Biological Plausibility

- Given the sensors, neurobiological implementation plausible, only few thousand neurons required
- Many of the sensors map to biological systems:
  - radar $\leftrightarrow$ structure from motion
  - contact $\leftrightarrow$ tactile sensors
  - distance and direction $\leftrightarrow$ olfactory system in insects

⚠ Yet, the given implementation must not be seen as a biological model!

Lessons

- Implementing the simulator: doing CG right is hard...
- Implementation of *Bug 0* as neural network quite trivial and works well.
- Implementation of *Bug 2* suffers from noise/chaotic behaviour.

Conclusion

Biological Plausibility

- Given the sensors, neurobiological implementation plausible, only few thousand neurons required
- Many of the sensors map to biological systems:
  - radar $\leftrightarrow$ structure from motion
  - contact $\leftrightarrow$ tactile sensors
  - distance and direction $\leftrightarrow$ olfactory system in insects

⚠ Yet, the given implementation must not be seen as a biological model!

Lessons

- Implementing the simulator: doing CG right is hard...
- Implementation of *Bug 0* as neural network quite trivial and works well.
- Implementation of *Bug 2* suffers from noise/chaotic behaviour.

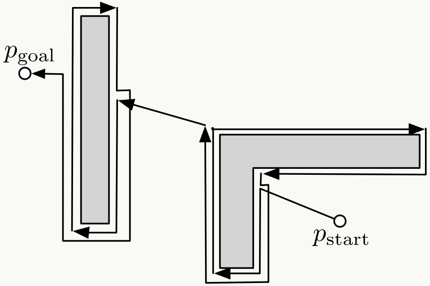
Challenges

- Plenty of magic constants in both reference and neural implementation
- State transition network

Thank you for your attention!

The Bug 1 algorithm

```
while not at goal do
    move towards the goal
    if hit an obstacle then
        at the same time
            circumnavigate the obstacle
            track minimum distance to goal
        follow boundary back to minimum
    end if
end while
```



Source: Lectures on Robotic Planning and Kinematics, Bullo and Smith, 2016

- ⊕ One can show that this algorithm is *complete*
- ⊖ Relatively complex, paths are far from optimal

media/video/demo_bug_1.mp4

Spiking Neural Networks: Why Bother?

- Can implement linear algebra and dynamical systems in spiking neural networks

Spiking Neural Networks: Why Bother?

- Can implement linear algebra and dynamical systems in spiking neural networks

$\Rightarrow$ Why not just use the mathematical description?

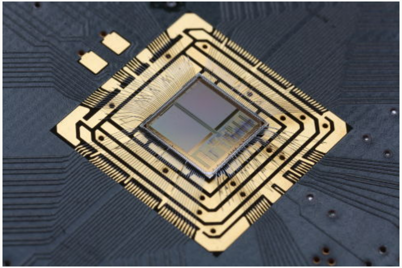
Spiking Neural Networks: Why Bother?

- Can implement linear algebra and dynamical systems in spiking neural networks

$\Rightarrow$ Why not just use the mathematical description?

ENGINEERING

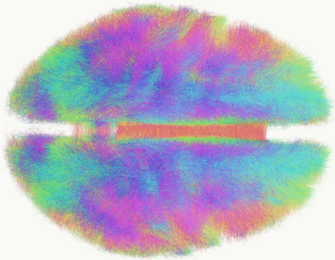
Use neuromorphic hardware



Spikey Neuromorphic Chip
Source: Electronic Visions Group, KIP Heidelberg

COMPUTATIONAL NEUROSCIENCE

Build biologically constrained models of cognition



Human Connectome. Data by Horn A. et al., 2014
Source: Wikimedia Commons