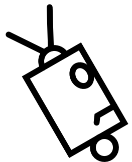
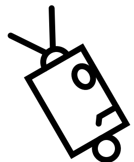
Teaching an Old Robot New Tricks: Learning via Interaction with People and Things

Matthew J. Marjanovic
MIT AI Lab



Embodied AI

Lots of structure *built-in*. . . What about *building structures*?



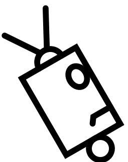
Overview

Three contributions to Embodied AI:

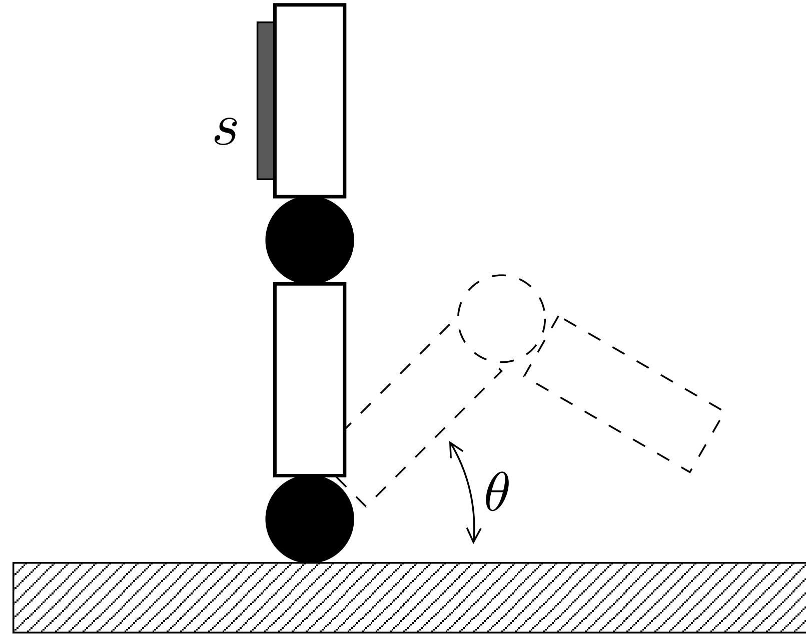
- *sok*: Environment for behavior-based programming.
- *meso*: A biologically-motivated motor-control system (*virtual muscles*).
- *pamet*: A modular system for learning from interaction with the environment and people, with enough modules to learn some simple behaviors.

What you will see:

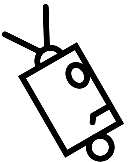
- Cog is taught to perform some simple motor tasks.



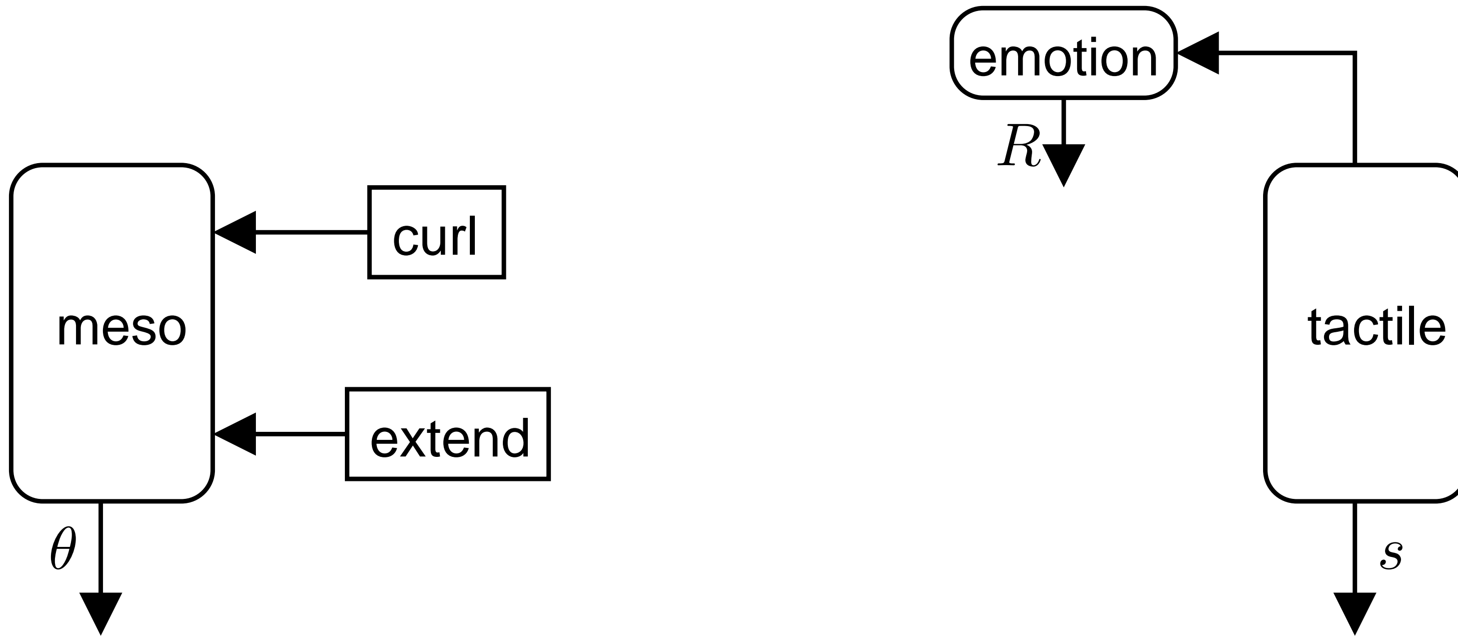
An Example: The FingerBot

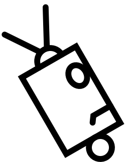


- One degree-of-freedom motion; single tactile sensor.
- Our goal: teach FingerBot to *point in response to touch.*



FingerBot initial state





FingerBot learns to move

emotion

R

meso

curl

extend

tactile

θ

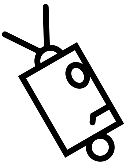
s

modeller

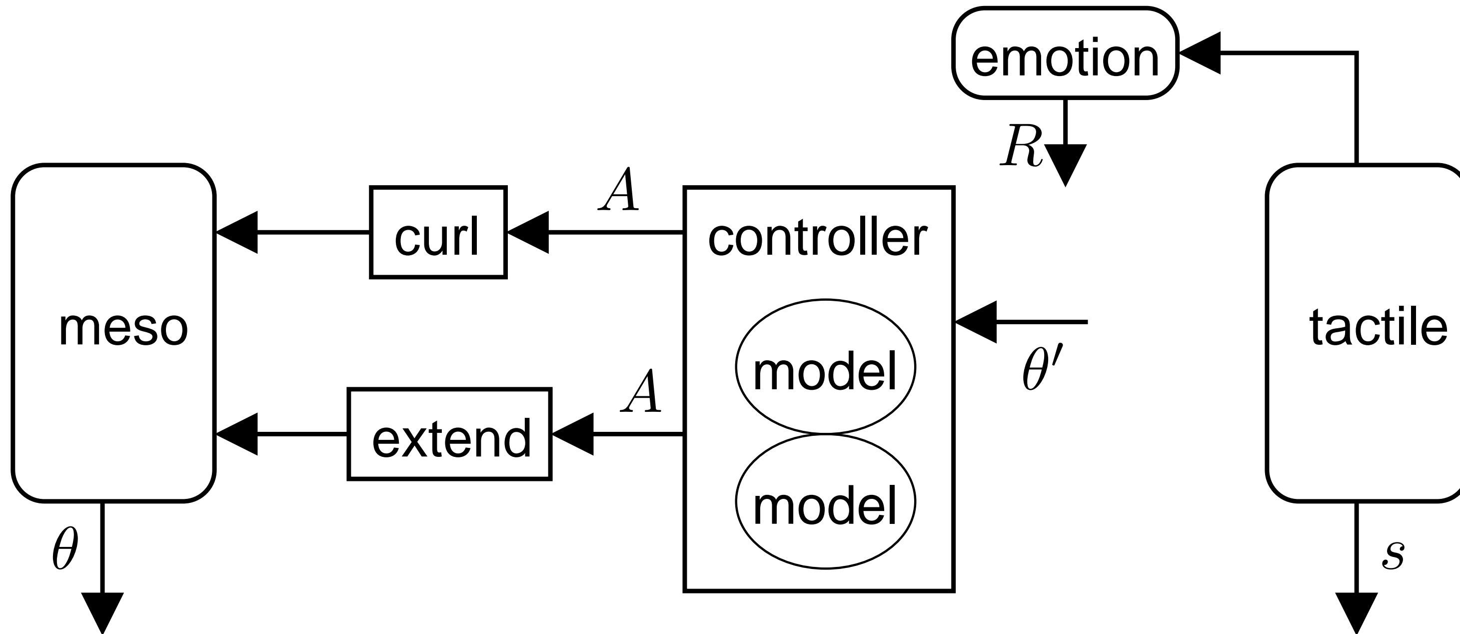
modeller

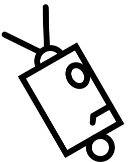
modeller

modeller

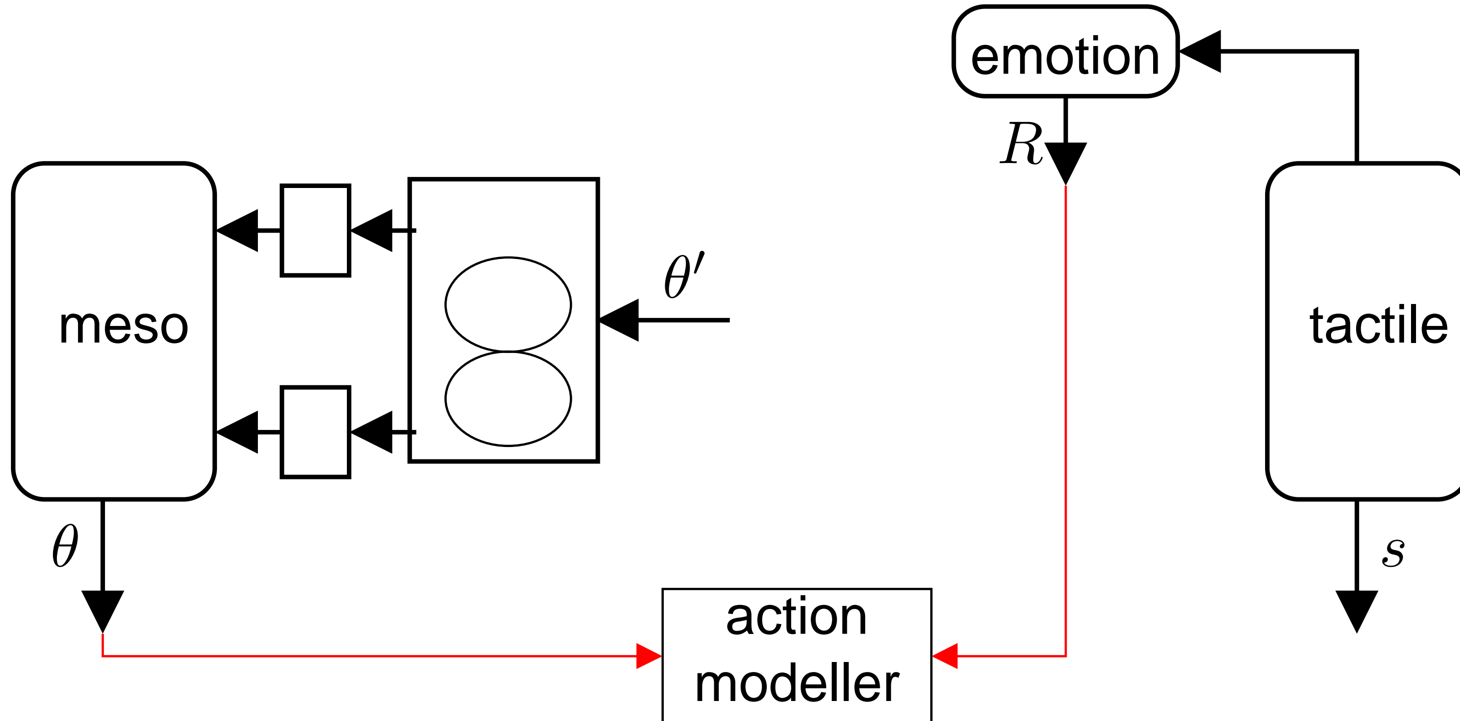


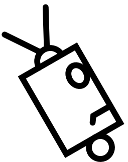
FingerBot controller



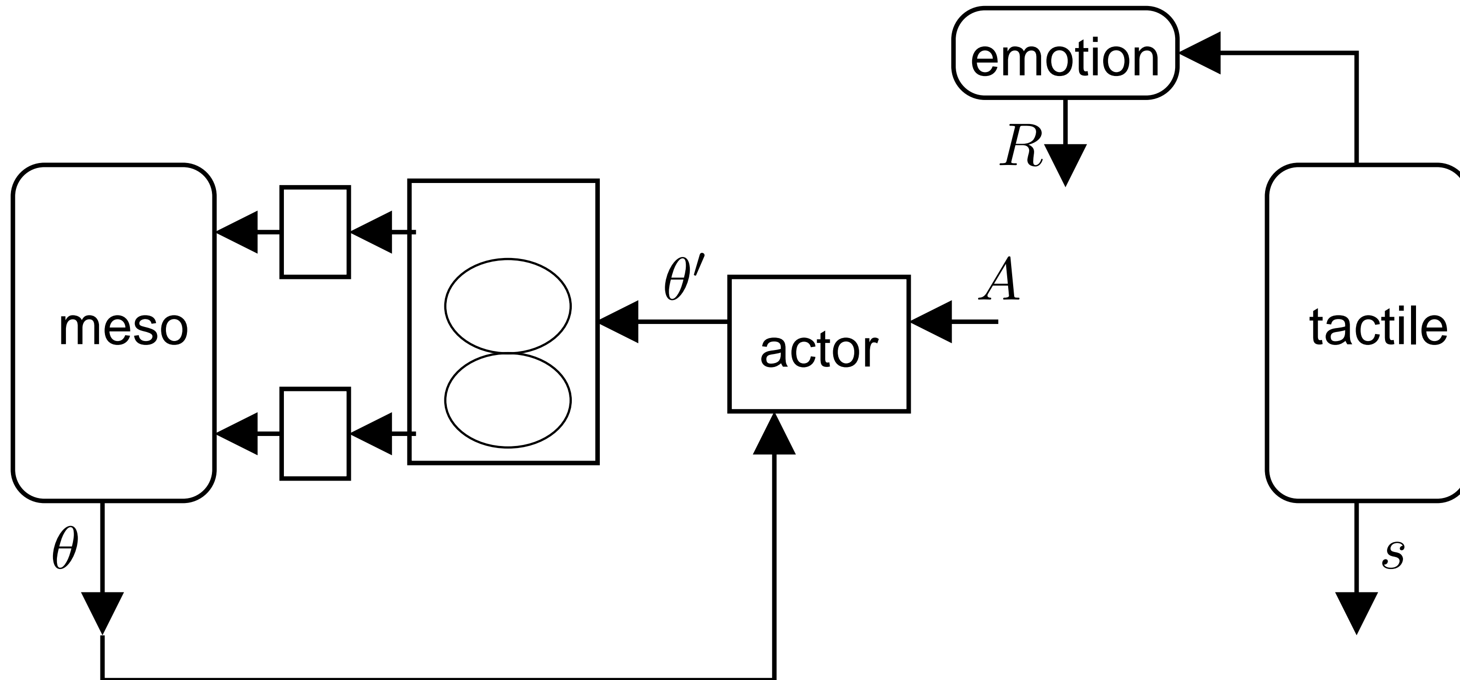


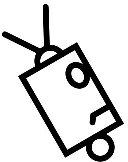
Teach FingerBot *how* to point



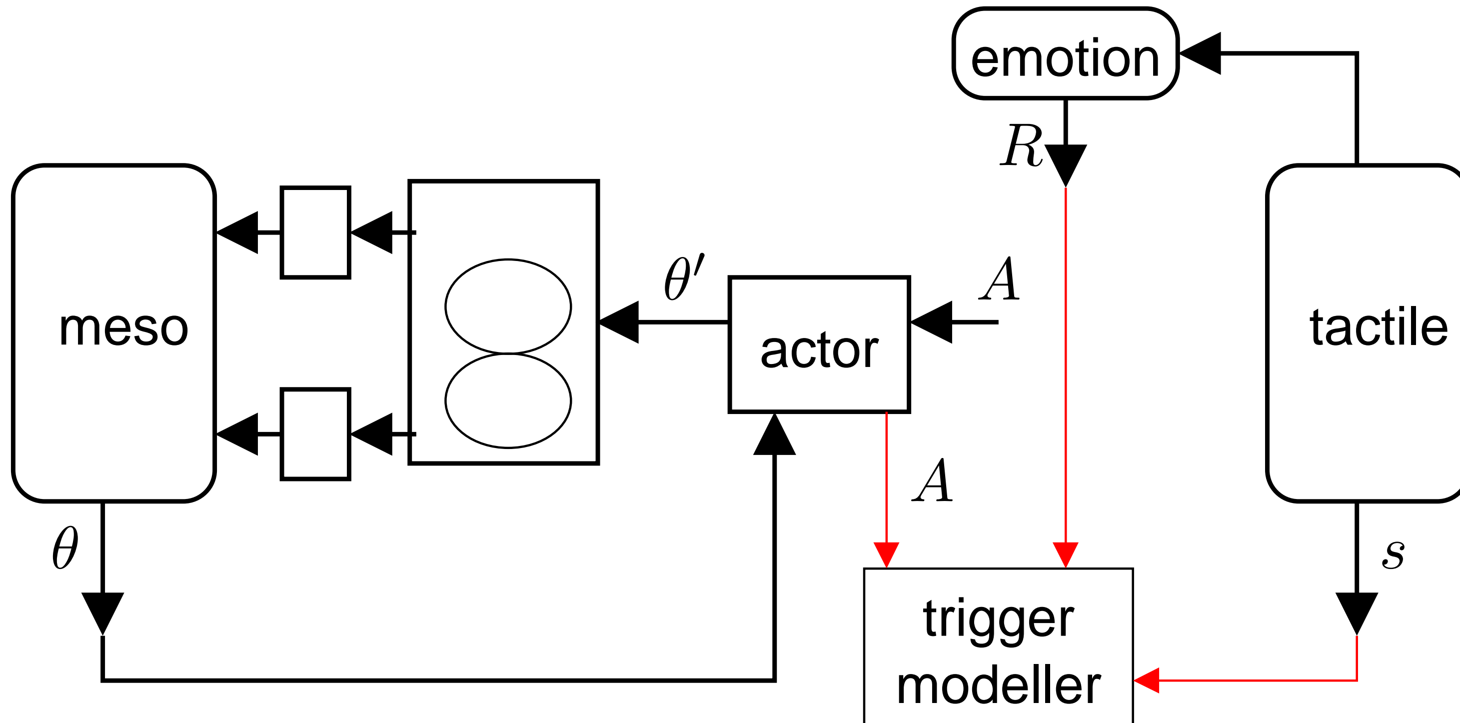


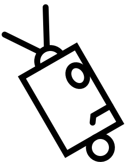
FingerBot points



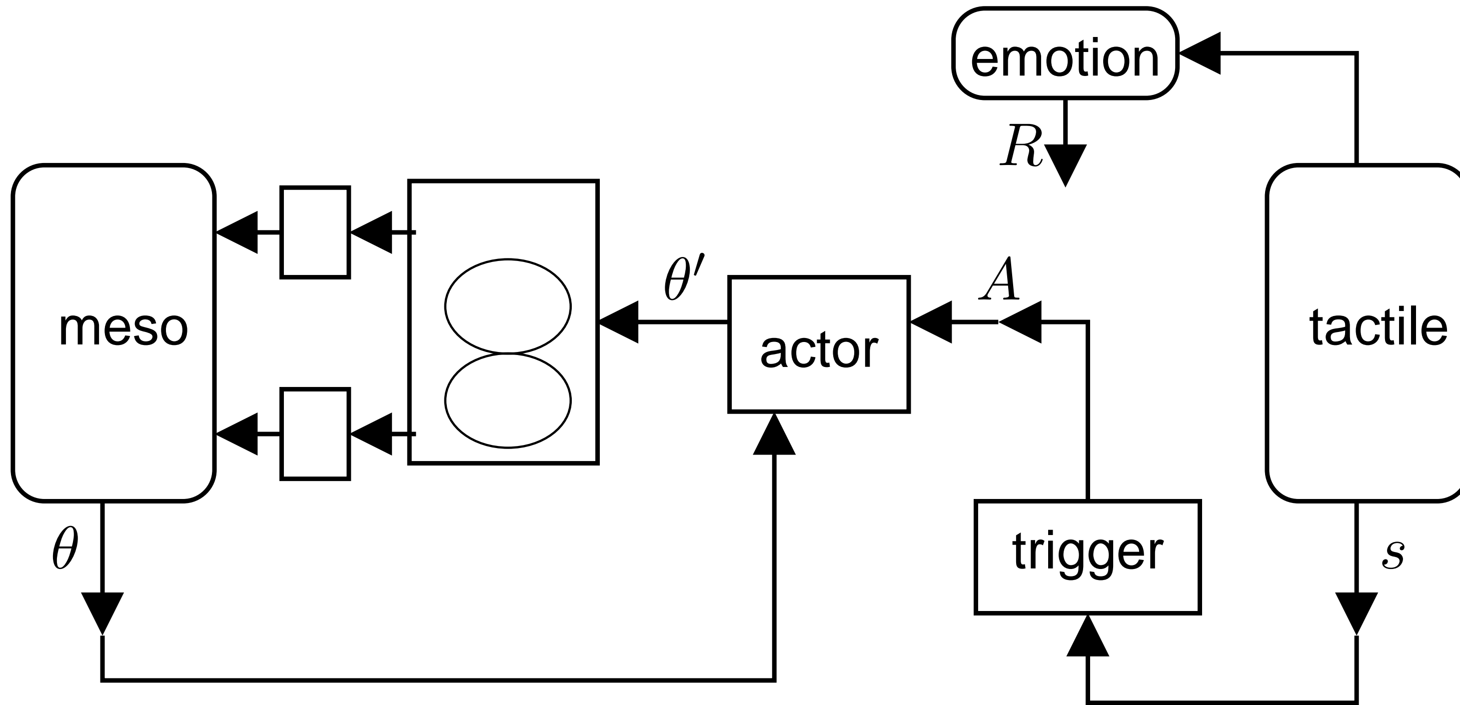


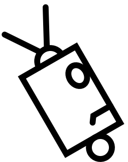
Teach FingerBot *when* to point





FingerBot points when touched





Overview

sok

processes, ports

pamet

models, modellers,
movers, controllers,
actions, triggers,
transformers

vision

attention,
tracking,
head control

emotion

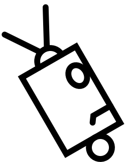
happy
sad

tactile

touch sense

meso

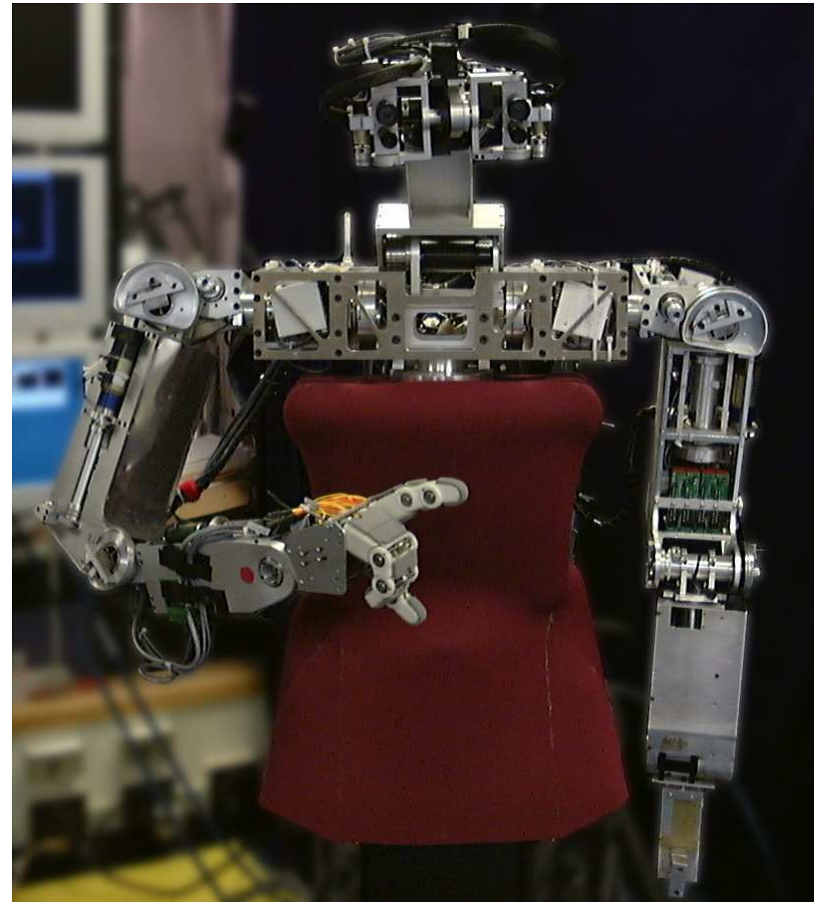
virtual muscles, joint limits, fatigue

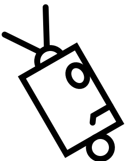


The Robot: Cog

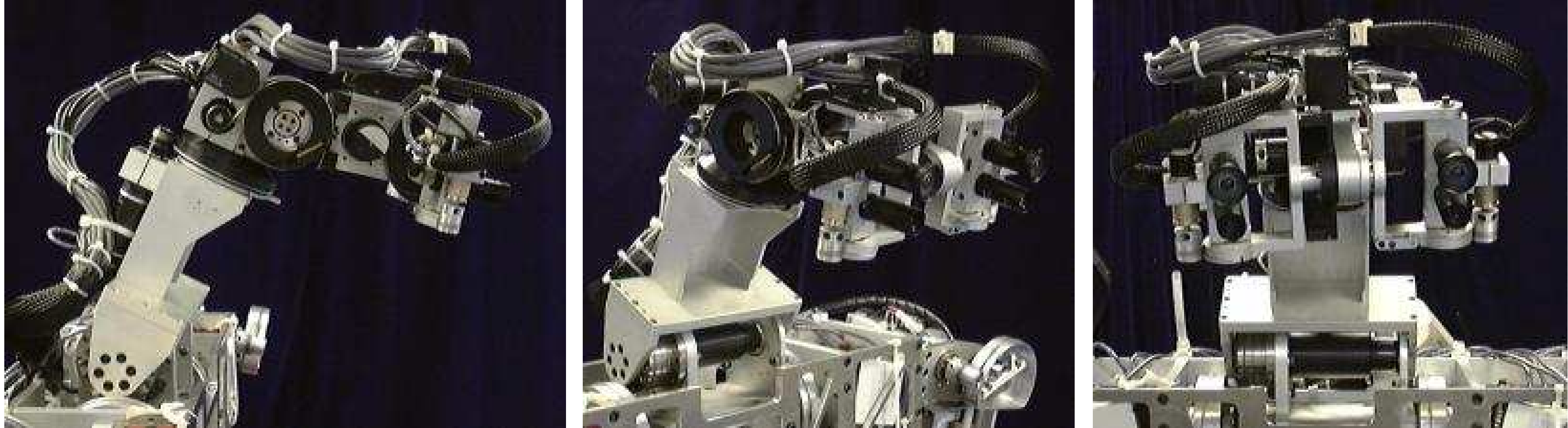
Humanoid, anthropomorphic robot:

- 4-dof head, with 3-dof stereoscopic eyes
- two 6-dof torque-controlled arms
- 3-dof torque control in the torso
- off-board processing: ~28 off-the-shelf x86 PC's running QNX4

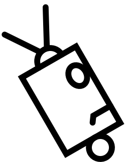




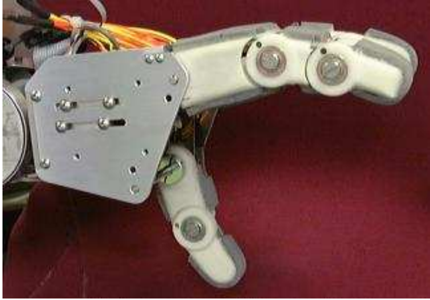
Cog’s Head

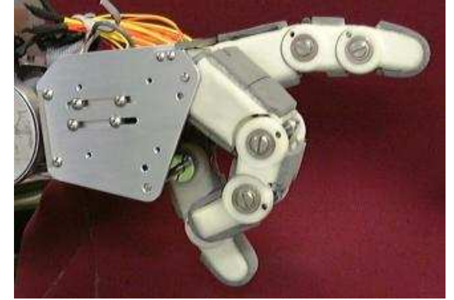


- 4-dof in head: tilt, pan, roll, neck “lean”
- gyroscope: angular velocity, inclination
- 3-dof in two eyes: separate tilt, linked pan
    - dual cameras: wide-angle, narrow-angle “foveal”



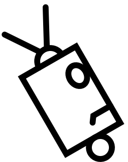
Cog's Hand





reach *grasp* *pinch* *point*

- 2-dof: linked finger/thumb, paddle
- Four recognizable gestures (sorry, no "thumbs up")
- Six surfaces of tactile sensation



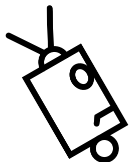
Why build a humanoid robot?

Assorted “practical” reasons:

- Create machines/tools which adapt to human environments.
- Test theories from cognitive science, neuroscience.

Grand philosophical reason:

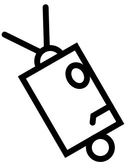
- We want to create a human-level machine intelligence.
- Intelligence is predicated on how an agent interacts with the world.



Metaphors We Live By

[Lakoff & Johnson, 1980]

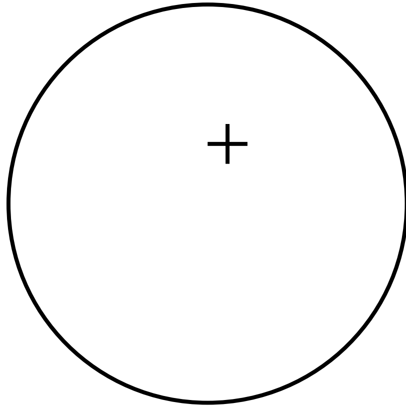
- Metaphors are not just linguistic constructs; we *think* in metaphors.
  - Happy is Up; Sad is Down.
  - Significant is Big.
  - Time is Money.
- No *literal* meaning; no reduction to discrete propositional forms or symbols.
- *Operational* meaning: represent consistent patterns of interaction.
- Grounded in experience.

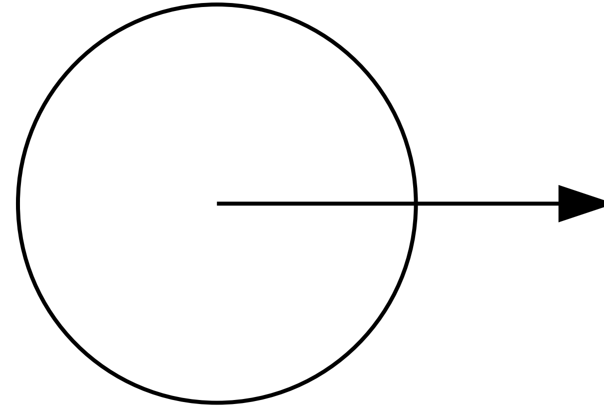


The Body in the Mind

[Johnson, 1987]

Some metaphors are cultural;
some arise from basic human physical interactions: *image schemata.*

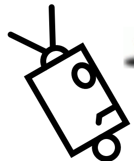




CONTAINER IN–OUT

Simple relationships which appear at all levels of thought:

- Physical: “George put the toys in the box.”
- Abstract: “Donald left out some facts.”

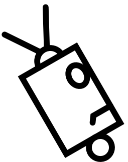


So, what does this mean?

If you want to build a *human-like* machine intelligence:

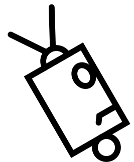
1. You need a system which can develop and manipulate metaphorical structures.
2. Those structures must be grounded in raw sensory and motor experiences.
3. Those experiences must be human-like.

(You need a humanoid robot.)



Cog is a humanoid robot.

- Much work on different aspects of humanoid-ness. . .
  - Matt Williamson: motor control via coupled non-linear oscillators (central pattern generators).
  - Robert Irie: auditory localization.
  - Cynthia Breazeal (Kismet): facial/vocal gestures, social interaction, emotional models.
  - Brian Scasselatti: visual animate/inanimate distinction, theory of body, imitation of simple motions.
  - Paul Fitzpatrick: visuomotor object understanding.
- Isolated parts, lacking direction/means for integration.
- Mostly learning parameters, not structures.

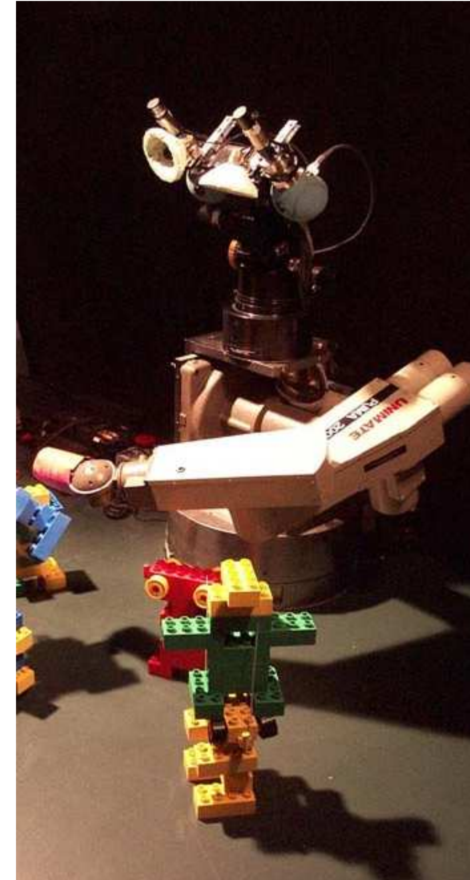


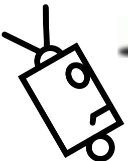
(Where are the metaphors gonna go?!)

Metta's Babybot

[Metta, 1999]

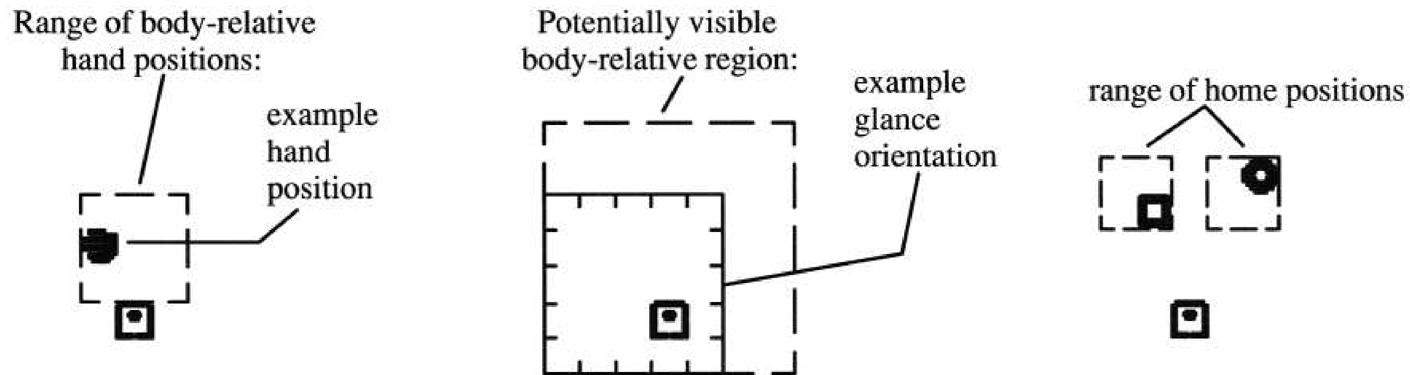
- 5-dof head with stereoscopic vision, gyroscope
- 6-dof torque-controlled arm
- Learns a developmental progression of sensorimotor coordination:
  - Saccades to visual targets via visual feedback (head motion rare).
  - Coordinated head and eye movement (random arm movement).
  - Controlling movement of arm, as a visual target.
- Culminates in reaching out toward moving visual stimuli.



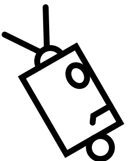


Drescher's Schema Mechanism

- Simulated robot learns progressively more abstract relations by interacting with its world. [Drescher, 1991]



- Mechanism has *items*, *actions*, *schemas*: predictors,
  CONTEXT + ACTION $\Rightarrow$ RESULT
- Learning and abstraction:
  - New schemas for more specific context or result.
  - Composite actions and synthetic items.

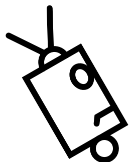


Schema Issues

Completely unrealistic simulation:

- 2-D grid-world with binary features.
- 10 discrete primitive actions.
- Total of 141 bits of primitive binary state.
- No physics, no noise.
- Serialized state-machine for action execution.

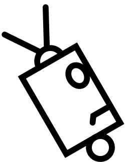
It’s a symbolic AI engine for a symbolic universe.



A Design Philosophy

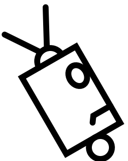
First and foremost, Real-World Constraints:

- Real robot, real physics.
- Real-time operation — human time-scales.
- Distributed computation — expandable, scalable.
- Noise — because separating signal from noise is what it's all about!



A Design Philosophy

- Dynamic: create new models/concepts.
- Transparent: compatible representations across modalities.
  - Reuse same machinery at different levels of development or abstraction.
- Malleable: states/actions/etc allowed to change/evolve.
- Distributed control: no central planner, arbitrator, action selector, modeller.
  - Only serialize operations when required by resource contention.
- Grounded close to the raw physics.



Three Pieces

sok

processes, ports

emotion

happy
sad

pamet

models, modellers,
movers, controllers,
actions, triggers,
transformers

vision

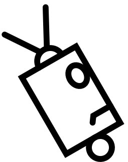
attention,
tracking,
head control

tactile

touch sense

meso

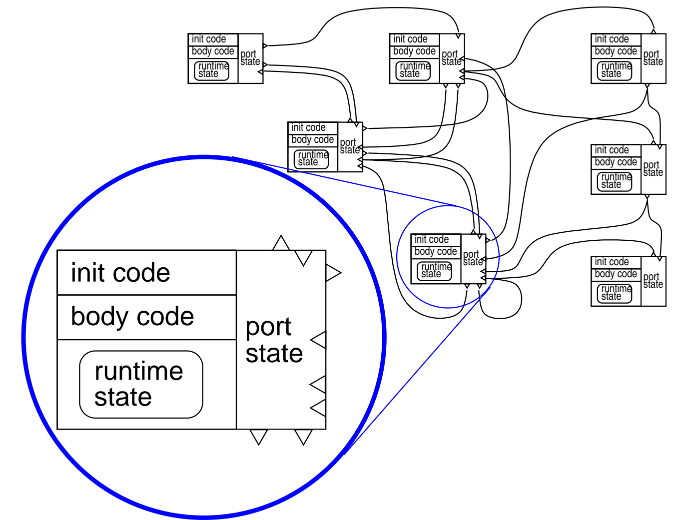
virtual muscles, joint limits, fatigue

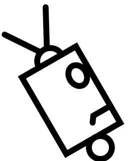


sok: Behavior-based Environment

Dynamic network of *sok processes*, connected via *inports* and *outports.*

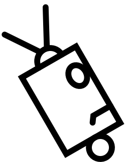
- Provides interprocess communication and process control.
- Connection-based message-passing.
- Event-driven code.
- Distributed: works transparently over a network of processors (via QNX).
- C, C++ libraries, plus utilities.





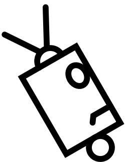
sok Features

- Processes and connections are *dynamic.*
- Network is *tangible.*
  - Processes can be aware of the network topology.
  - Processes can search for other ports/processes by traversing the network.
  - Ports are typed.
- Separation of message-handling code and *body code.*
  - Processes can be killed and restarted without affecting connectivity.
- State of *sok space* can be dumped/restored from disk.



sok Type System

- Ports are typed: C-like (or IDL)
  - 10 atomic `int` and `float` types, plus arrays and structures.
  - Can define constants, too.
- Scheme-based type compiler generates `.h` and `.c` files from `.stc` files.
- Type compiler is embedded in the sok library; type descriptions can be processed at runtime.
- Functions for walking typed buffers, generating composite type identifiers, housekeeping.



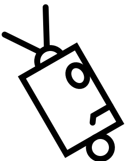
sok Arbitrators

Message passing is asynchronous and anonymous:

- Data received by an inport is buffered, and body process receives an event.
- Default behavior is to overwrite — no queueing.
- Body code doesn't know about multiple connections.

Arbitrator: code which will change reception behavior, e.g.

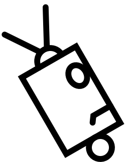
- *Summing*: Inport keeps a running sum of received data.
- *Averaging*: Body process sees the mean of all data received between reads.
- *Blocking*: One connection wins right to monopolize the port; messages from others are discarded.



Dump and Restore

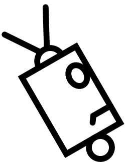
sok provides a dynamic environment, processes and connections come and go as a system develops.

- sok processes can be told to dump state to disk:
  - Connections to ports.
  - Persistent data, arbitrator state.
  - Command-line arguments, etc.
- Entire state of machine can be dumped or restored with single shell command.



Handy Utilities

- `sok` command-line interface:
  - inspect network, processes, ports
  - make/break connections
  - spawn/kill processes, send signals
  - initiate dump and restore
- *Megasliderama* Swiss-army GUI:
  - create panels of widgets connected to ports
  - sliders, stripcharts, buttons, image planes, . . .



Sensory and Motor Boxes

sok

processes, ports

vision

attention,
tracking,
head control

pamet

models, modellers,
movers, controllers,
actions, triggers,
transformers

emotion

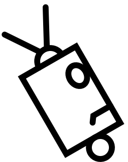
happy
sad

tactile

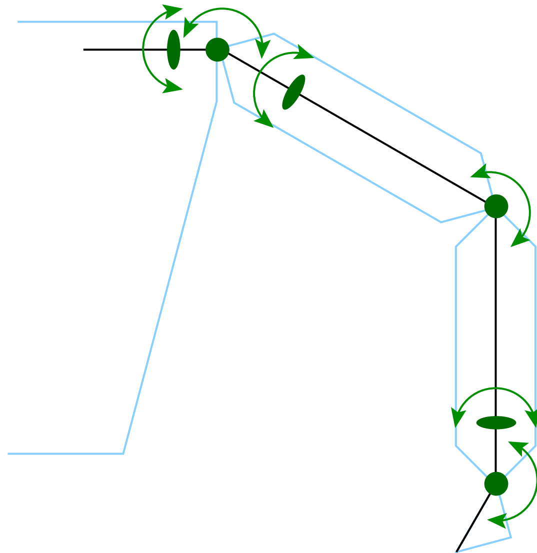
touch sense

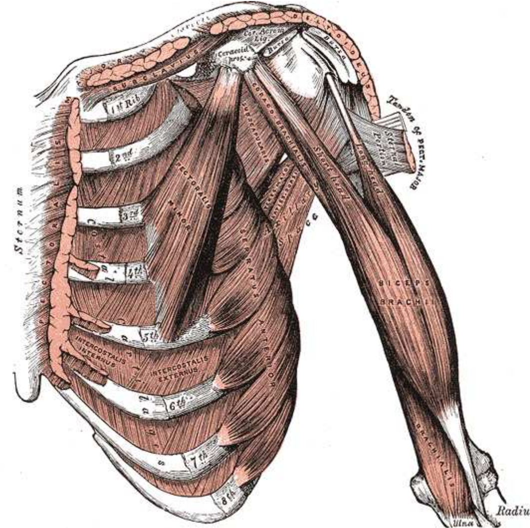
meso

virtual muscles, joint limits, fatigue

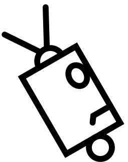


meso: Biological Motor Control



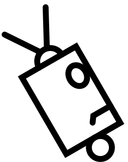


- Implements spring-like *virtual muscles.*
- Allows for *multi-joint coupling*.
- Provides performance feedback:
  - Muscle fatigue.
  - Joint pain.



Why muscles? Fatigue? Pain?

- Movement is influenced by limitations of physiology.
  - We try to minimize energy use, take advantage of physics.
  - We avoid uncomfortable motions.
- Concepts linked to physical interaction are derived from those limitations.
  - “heavy” = “things which make muscles tired”
  - “stretching limits”
- Cog’s electric motors have no innate fatigue.



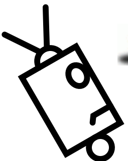
Why spring-like? Multi-joint?

Why spring-like?

- Real muscles are very non-linear force-producing elements, operate in antagonistic pairs.
- At spinal level, feedback loops make muscle groups behave like simple springs.
  - *Equilibrium-point control* hypothesis. [Bizzi, Kelso]

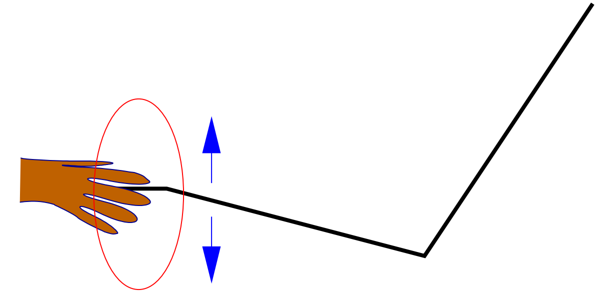
Why multi-joint coupling?

- Real muscles *do* span multiple joints.
- Necessary for complete control of endpoint stiffness. [Hogan, 1985]
- Multi-joint muscles enhance efficiency of certain movements. [Gielen, 1993]

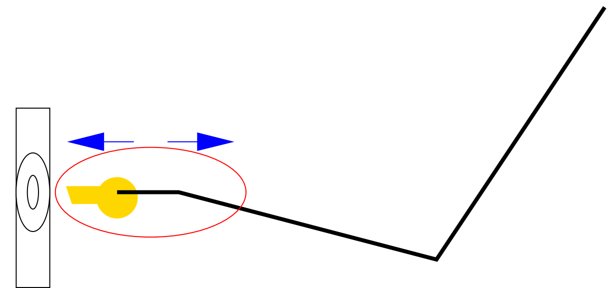


meso: Endpoint Stiffness

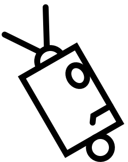
- Single-joint springs do not allow for full control of stiffness in *endpoint space.*
- Endpoint stiffness should be tuned to match the task.



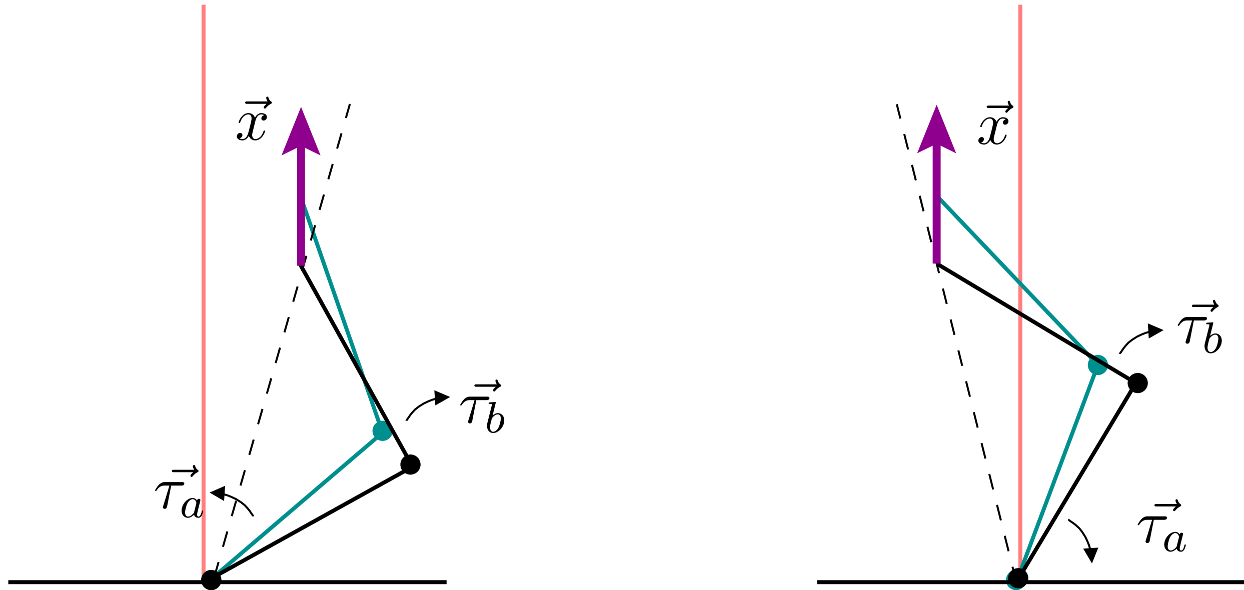
Handshake: low vertical stiffness



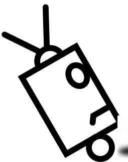
Key in lock: high vertical stiffness



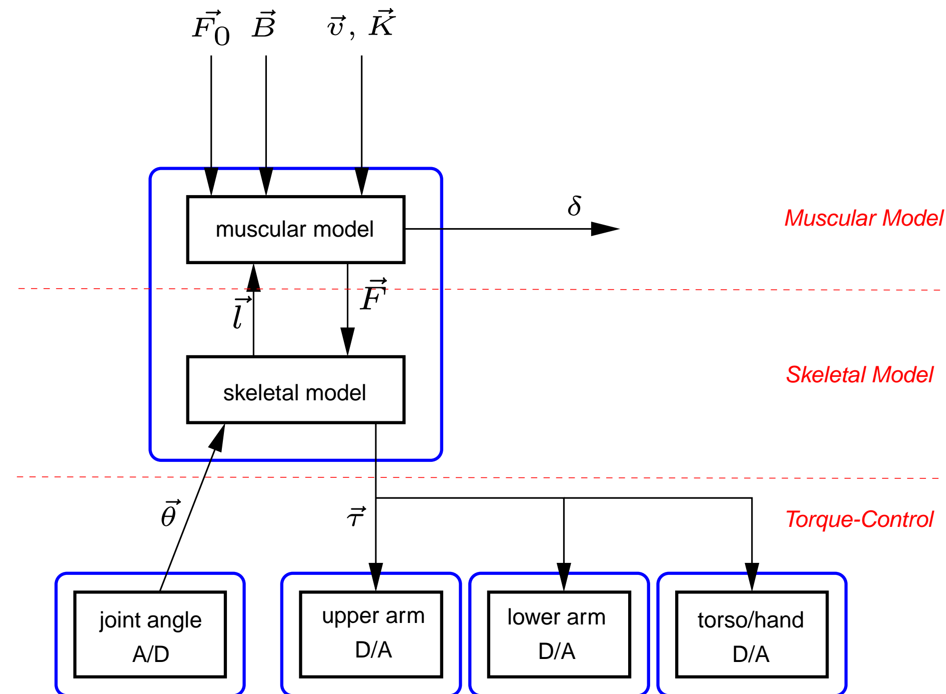
meso: Multi-joint Efficiency



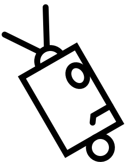
- Depending on configuration, single-joint muscles may undergo *lengthening contraction*, i.e. dissipating energy.
- A multi-joint muscle acting as stiff linkage prevents this, makes motion more efficient.
- To Cog, such motion will *feel* more efficient.



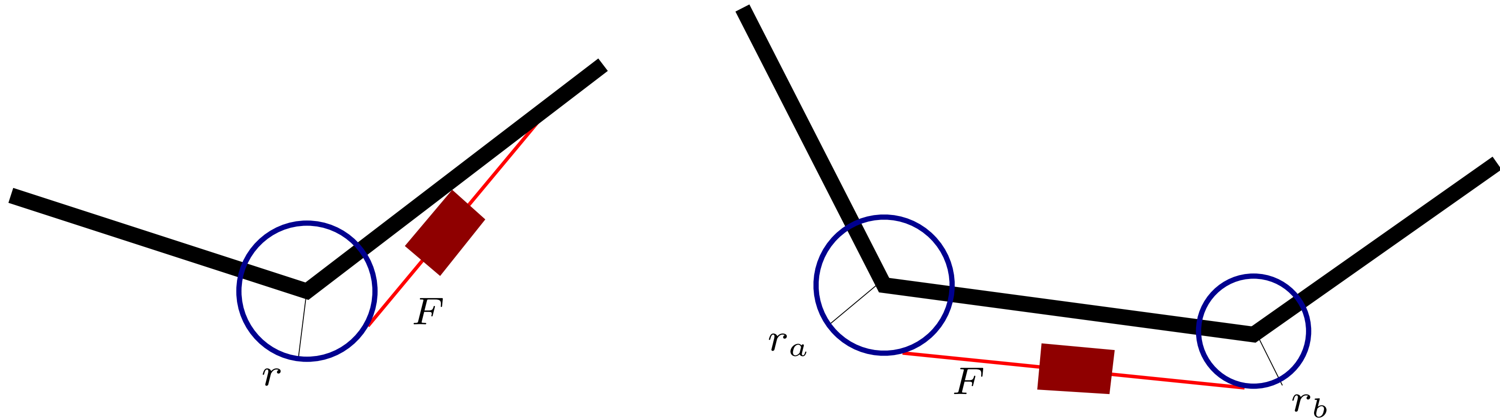
meso: Three Layers



- Low-level torque control.
- Skeletal model (muscle kinematics).
- Muscle model (muscle dynamics, fatigue).

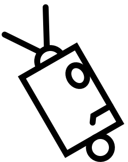


meso: Skeletal Model



- Compute muscle lengths $\vec{l}(\vec{\theta})$, joint torques $\vec{\tau}(\vec{F}, \vec{\theta})$.
- Simple "pulley" model:

$$l_m = \sum_j r_{jm}\theta_j, \quad \tau_j = \sum_m r_{jm}F_m$$

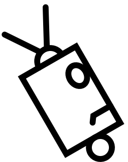


meso: Muscle Model

- Compute forces from lengths, i.e. spring-law:

$$F = K(l - l_0) - B(\dot{l}) + F_0$$

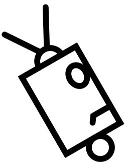
- Setpoint l_0 and stiffness K are control parameters.
- Damping constant B needs to be tuned.
- Bias force F_0 can be used for feedforward control (e.g. gravity/posture).



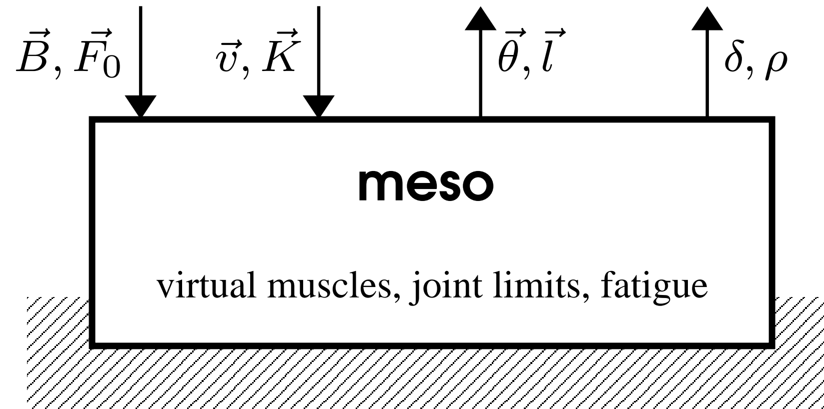
Fatigue Model

- Adams [2000]: detailed biochemical fatigue model:
  - Glucose/glycogen, adipose tissue, heart rate, insulin/glucagon/epinephrine. . .
- Abstracted further:
  - Required power: $P = \alpha|Fv| + \beta|F| + \gamma K$
  - Metabolic supply: P_R
  - Energy reserves: short-term S_S, long-term S_L.
- Discomfort and muscle efficiency due to S_L:

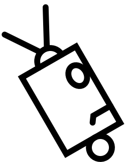
$$\delta = 1 - \left(\frac{S_L}{S_{L0}}\right), \quad \phi = \left(\frac{S_L}{S_{L0}}\right)^{1/2}$$



meso as a Black-box



- Inputs: muscle setpoint *velocity*, stiffness.
- Outputs: joint angles, muscle lengths, fatigue, pain.



Next Box: Vision

sok

processes, ports

emotion

happy
sad

pamet

vision

attention,
tracking,
head control

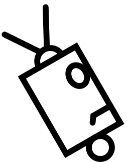
models, modellers,
movers, controllers,
actions, triggers,
transformers

tactile

touch sense

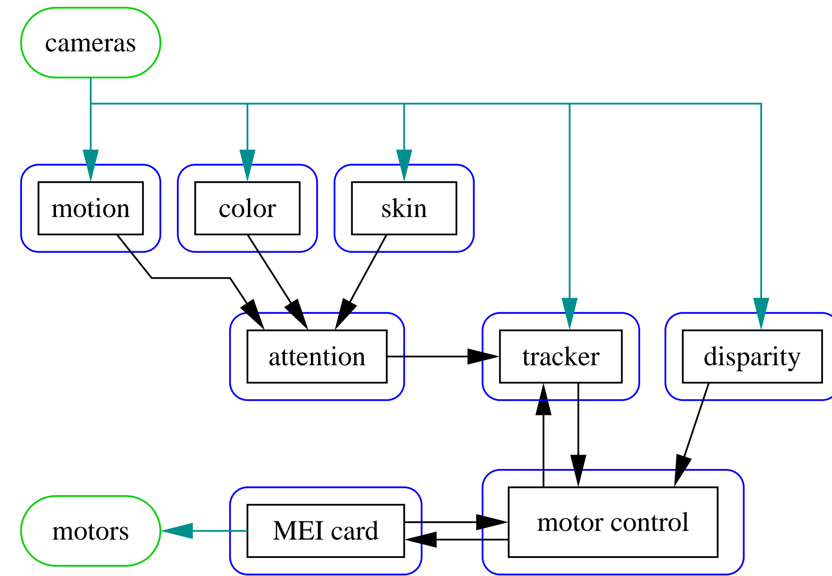
meso

virtual muscles, joint limits, fatigue

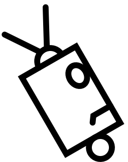


Vision System

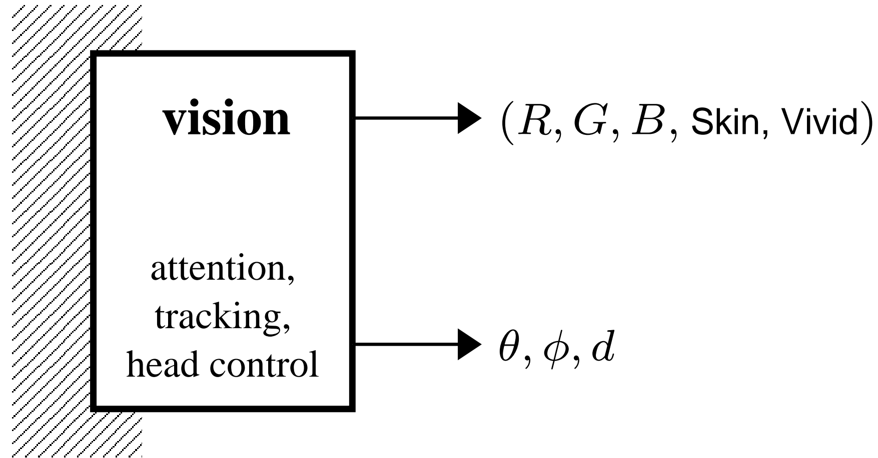
- Saliency filters:
  - Color, skin tone, motion.
- Attention:
  - Weights/adds saliencies
  - Chooses potential targets.
- Tracking system:
  - Locks onto image patch.
  - Follows it around.



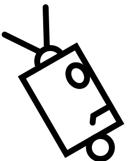
(thanks Paul and Giorgio!)



Vision as a Black Box



- Outputs only:
  - Target gaze-angles, disparity: (θ, ϕ, d)
  - Visual features of target: $(R,\ G,\ B,$ Skin, Vivid$)$



Next Box: Tactile Sense

sok

processes, ports

pamet

models, modellers, movers, controllers, actions, triggers, transformers

emotion

happy
sad

vision

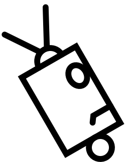
attention, tracking, head control

tactile

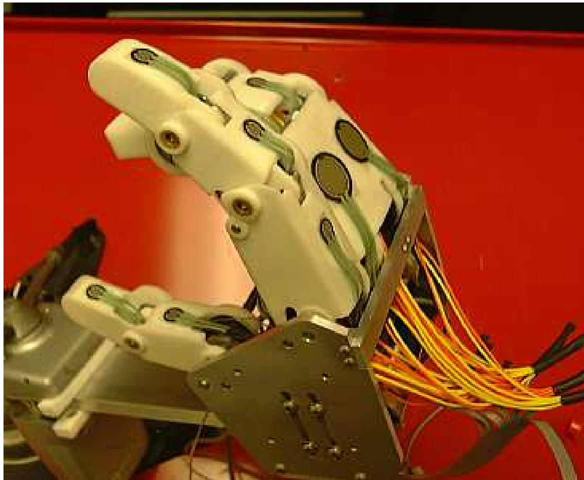
touch sense

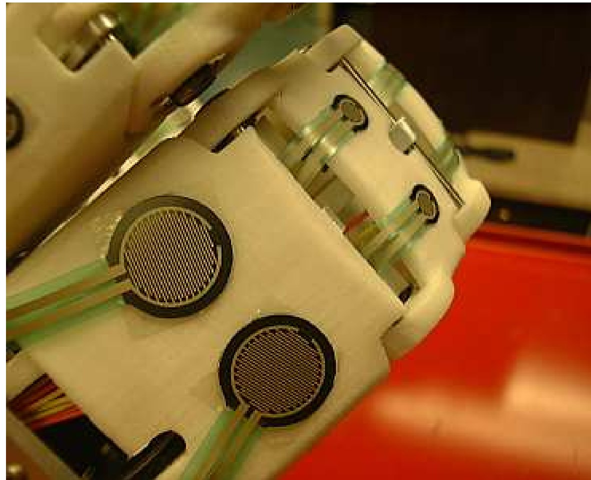
meso

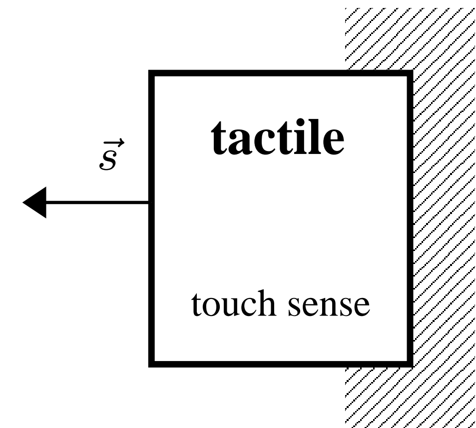
virtual muscles, joint limits, fatigue



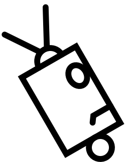
Tactile Sense







- 22 force-sensitive-resistor pads.
- Wired in parallel to yield six sensor surfaces.
  - Vector of six values, [0,1].



Last Box: "Emotion"

sok

processes, ports

pamet

models, modellers,
movers, controllers,
actions, triggers,
transformers

emotion

happy
sad

vision

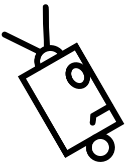
attention,
tracking,
head control

tactile

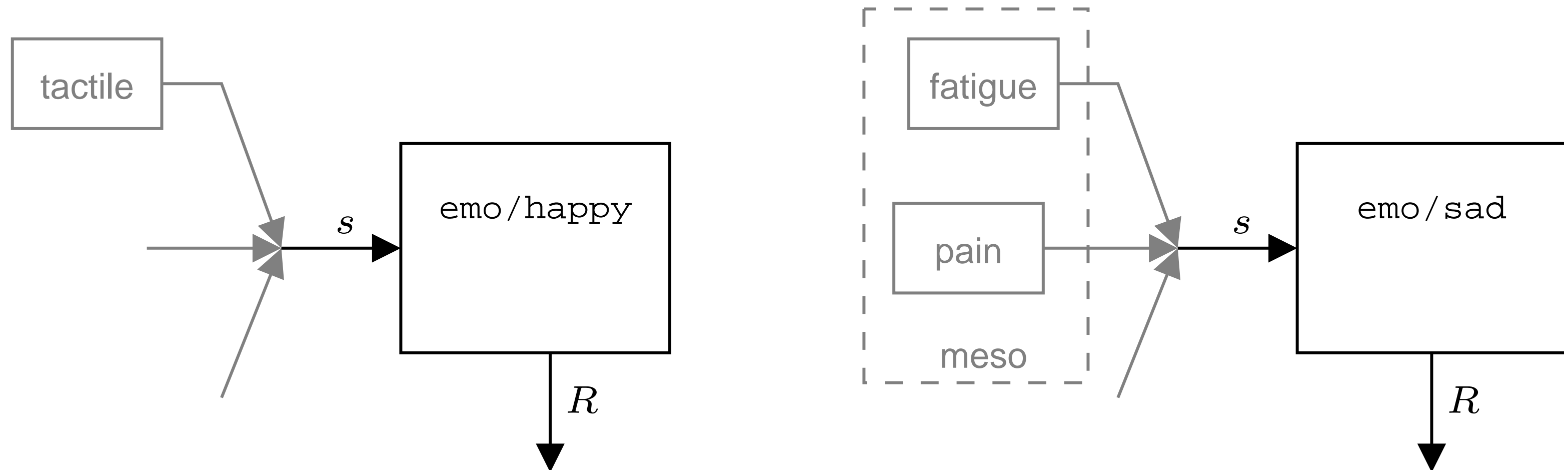
touch sense

meso

virtual muscles, joint limits, fatigue

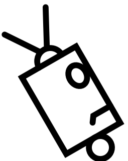


“Emotion”



- Combines sources of innate positive/negative reward.
  - Positive: hand-squeezes.
  - Negative: muscle fatigue, joint pain.
- Leaky integrator dynamics:

$$R(t) = \lambda R(t-1) + \sum r_i(t-1, t)$$



Final layer: pamet

sok

processes, ports

emotion

happy
sad

pamet

models, modellers,
movers, controllers,
actions, triggers,
transformers

vision

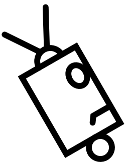
attention,
tracking,
head control

tactile

touch sense

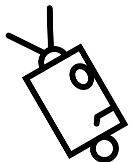
meso

virtual muscles, joint limits, fatigue

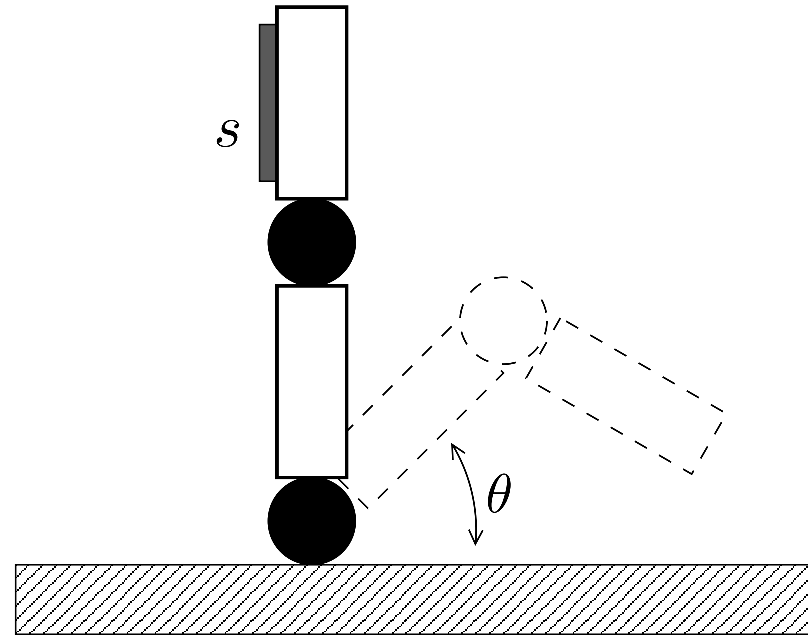


pamet: connecting, learning

- Static modules (sok processes) provide initial structure:
  - Sensory and motor primitives (black boxes, movers).
  - *Proto-modeller* modules to create modellers.
- *Modeller* modules try to discover relations between state parameters, and create models.
  - Mover models, action models, trigger models, transform models.
- *Instance* modules use the models to produce behavior, create new state parameters.
  - Controllers, actors, triggers, transformers.
- Knowledge is acquired in network structure and models.



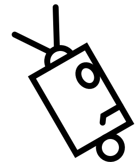
An Example: The FingerBot



- One degree-of-freedom: joint angle θ.
- Controlled by a single virtual muscle, length l.
- Single tactile sensor s.

Goal: *Teach the finger to point in response to touch.*

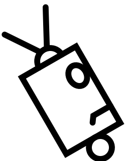
Implementation of the FingerBot



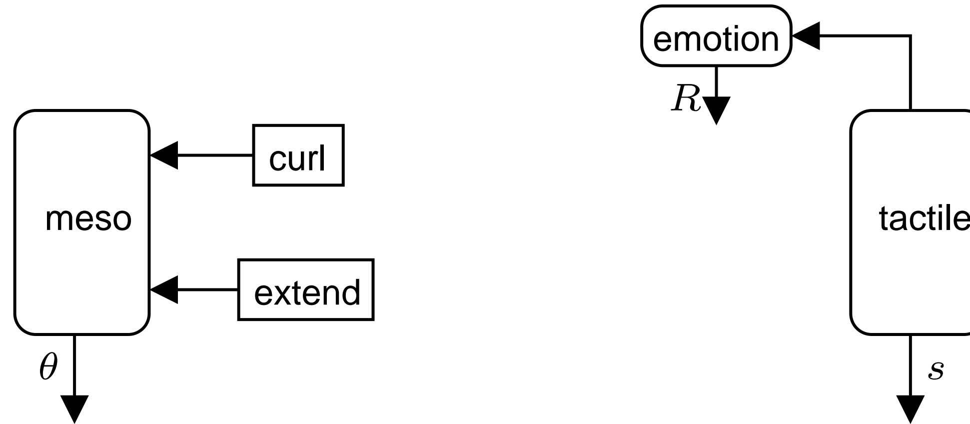
A Menagerie of Modules

Module	*Subclass*	*Inputs*	*Outputs*
mover	meso	A	
controller		$\vec{v}$	A
actor	position-constant	$A, \vec{s}$	$\vec{v}$
	position-parameter	$A, \vec{s}_1, \vec{s}_2$	"
	velocity-constant	$A, \vec{s}$	"
trigger	position	$\vec{s}$	A
	activation-delay	A_1, A_2	A
transformer		$\vec{s}_1, \vec{v}_2$	$\vec{s}_2, \vec{v}_1, A$

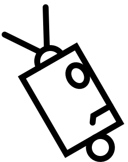
- Basic data types: state parameters $\vec{s}$, activations A, drive signals $\vec{v}$



FingerBot initial state



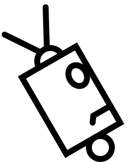
- Two hard-coded *mover modules*: “curl” and “extend”.
- *Mover* modules activate at random: exploration.
- Random activation is supressed by explicit activation.



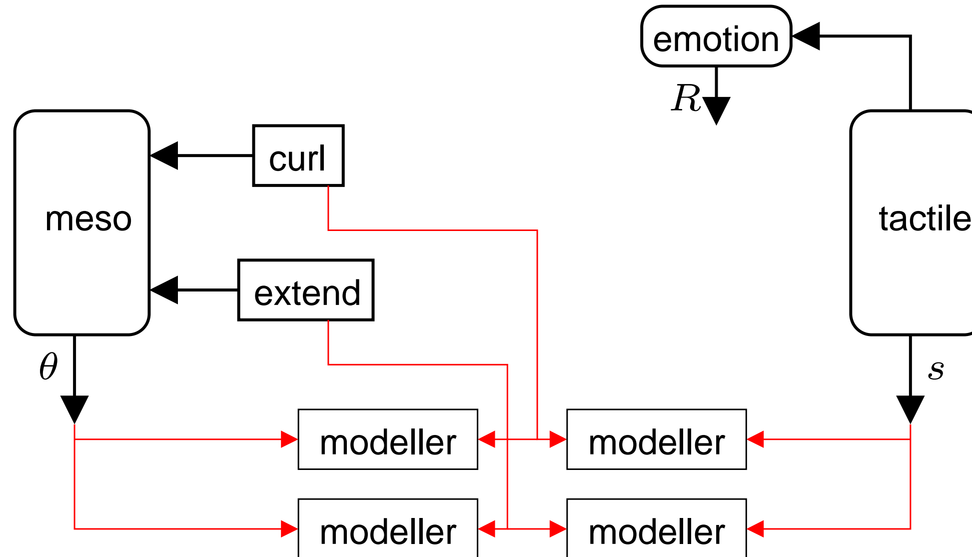
Mover

"Causes movement, rate modulated via a scalar parameter."

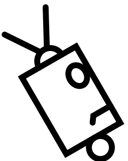
- *meso movers* are the primitive connections to the motor system.
  - Send a vector of muscle velocities, scaled by control parameter (and fixed stiffness).
- Effect on state parameters is modelled by *mover models.*



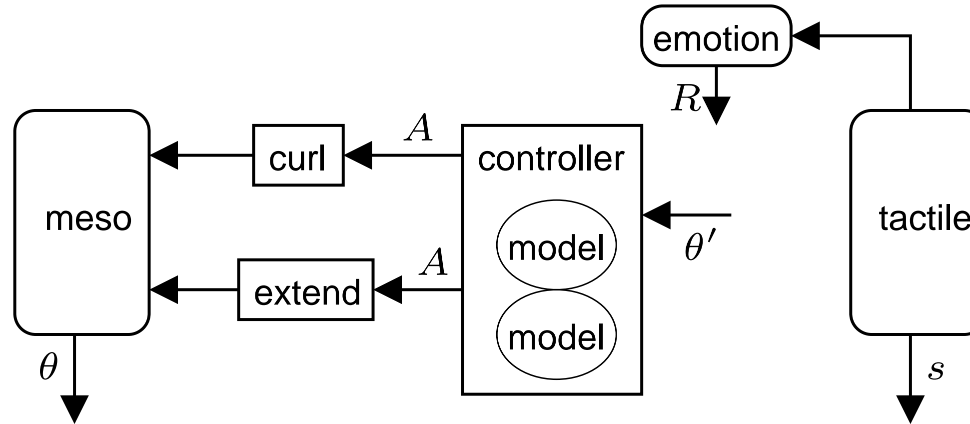
FingerBot mover modelling



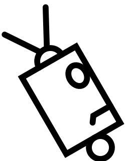
- *Mover modellers* are spawned for each pair of state parameter and mover activation.
- Learn/discover the relationship between the movers and the joint velocity.



FingerBot controller



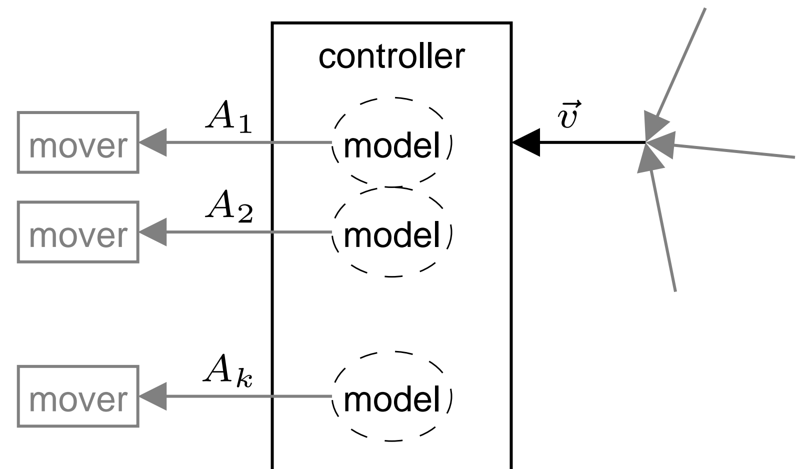
- A *controller* is created for the joint angle: controls joint velocity by activating movers.
- Joint angle becomes a *controllable parameter.*



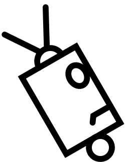
Controller

"Controls the velocity of a state parameter by activating movers."

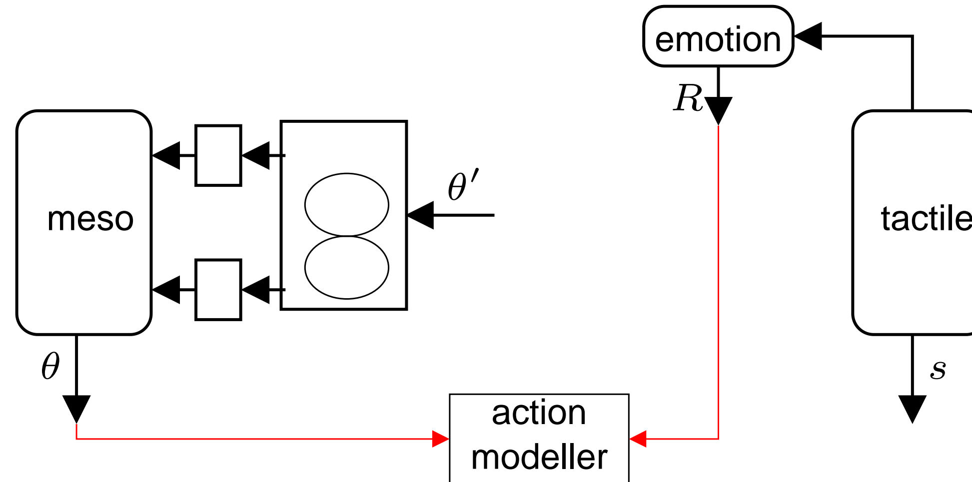
- Uses the mover models to determine how movers will affect state parameter.
- At each instant, activates the mover which will yield the velocity closest to the control input $\vec{v}$:



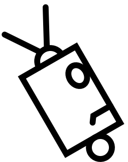
$$m = \arg\max\left(\frac{\vec{v} \cdot \vec{v}_m}{\|\vec{v}_m\|}\right)$$



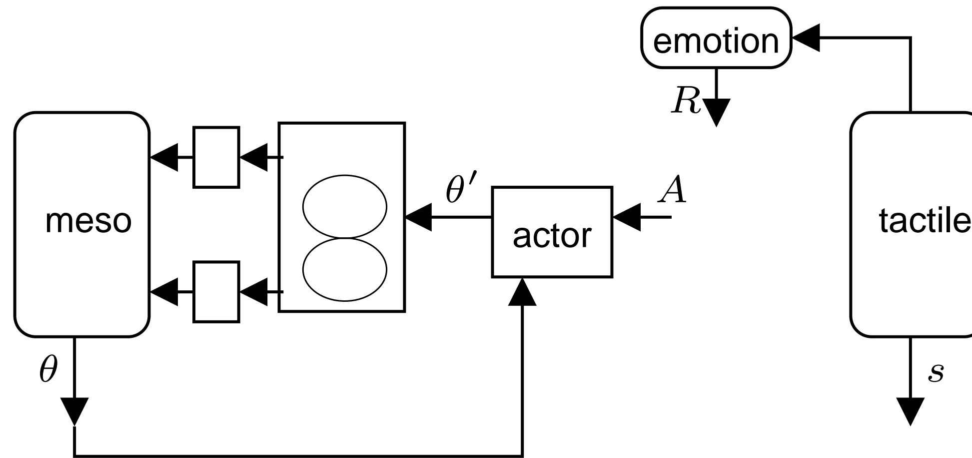
Training FingerBot to Point



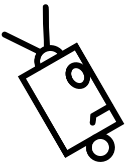
- An *action modeller* is created for each controllable parameter.
- Correlates reward with joint angle; creates an *action model* for the pointing posture.



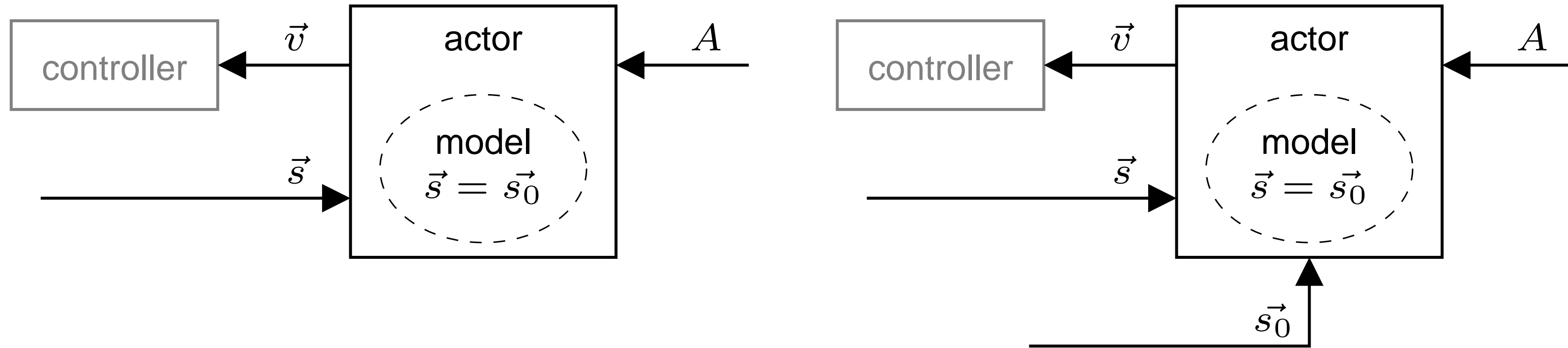
Training FingerBot to Point



- An *actor* is spawned, which moves finger to the pointing position at random times.

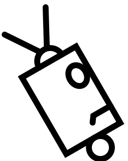


Actor

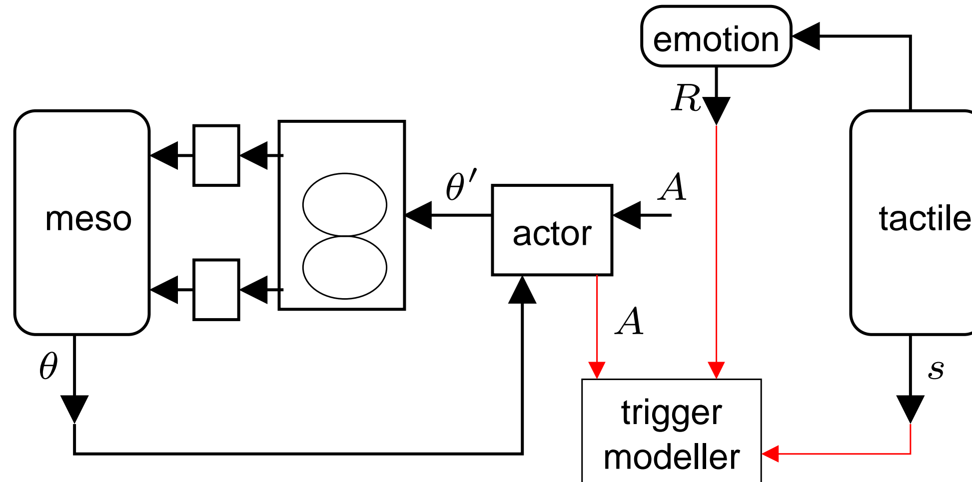


"Drives a controllable parameter to a goal value."

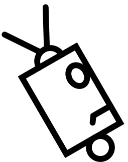
- *position-constant*: constant prototype value in action model.
- *position-parameter*: goal derived from another input parameter.
- Have random activation.
- Unused actors/actions will expire.



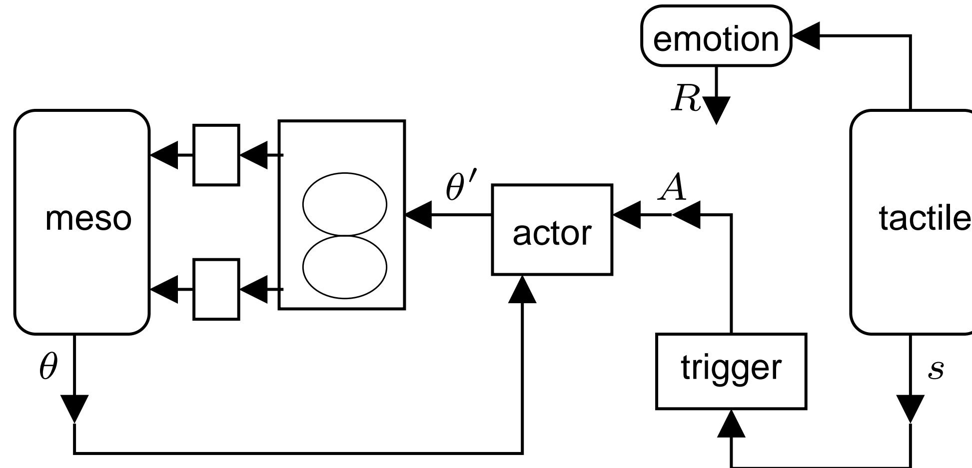
Pointing in response to Touch



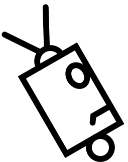
- A *trigger modeller* is created for pairs of actors and state parameters.
- Correlates state parameter with reward and activation.
- Creates a *trigger model* linking pointing action to a range of tactile sense values.



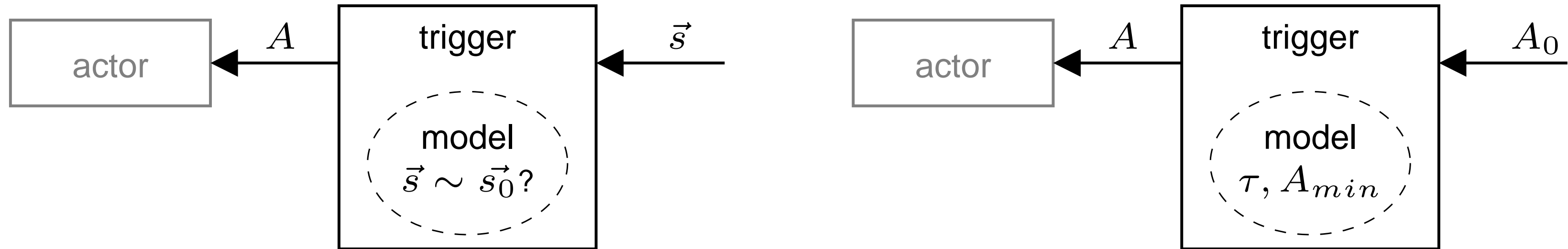
Pointing in response to Touch



- A *trigger* is spawned, which activates the pointing actor in response to touch.

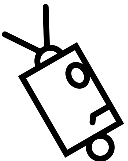


Trigger

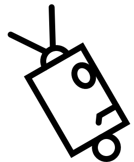


"Activates an actor when input state satisfies a context."

- *position*: context is a region in input state space.
- *activation-delay*: context is another activation signal.
- Activation output indicates how well context is satisfied.



Video of Three Arm Actions

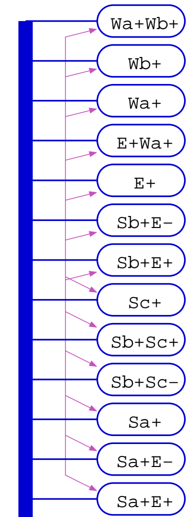


Cog, Initial State

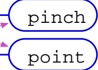
proto–mover–modeller

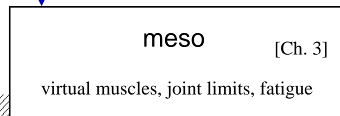
proto–action–modeller

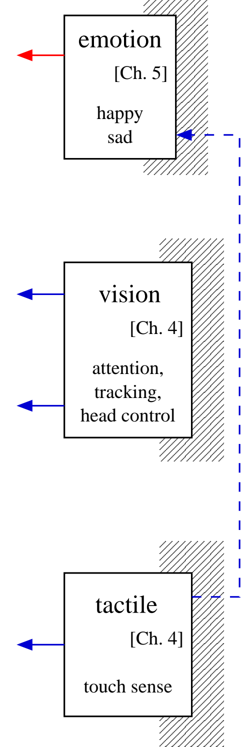
proto–trigger–modeller



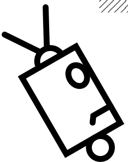
proto–controller







proto–transform–modeller



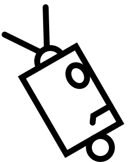
Mover Modelling

Modeller's question:
"Which state parameters are affected by the mover?"

- Linear model of state velocity as a function of activation

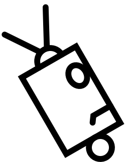
$$\vec{v}_s = A\vec{m}$$

- Modeller operation:
  1. Record samples $(\vec{s}, A)$; calculate velocities $\vec{v}$.
  2. Perform linear regression to estimate $\vec{m}$.
  3. Check correlation coefficient R^2 for each axis of $\vec{v}$.

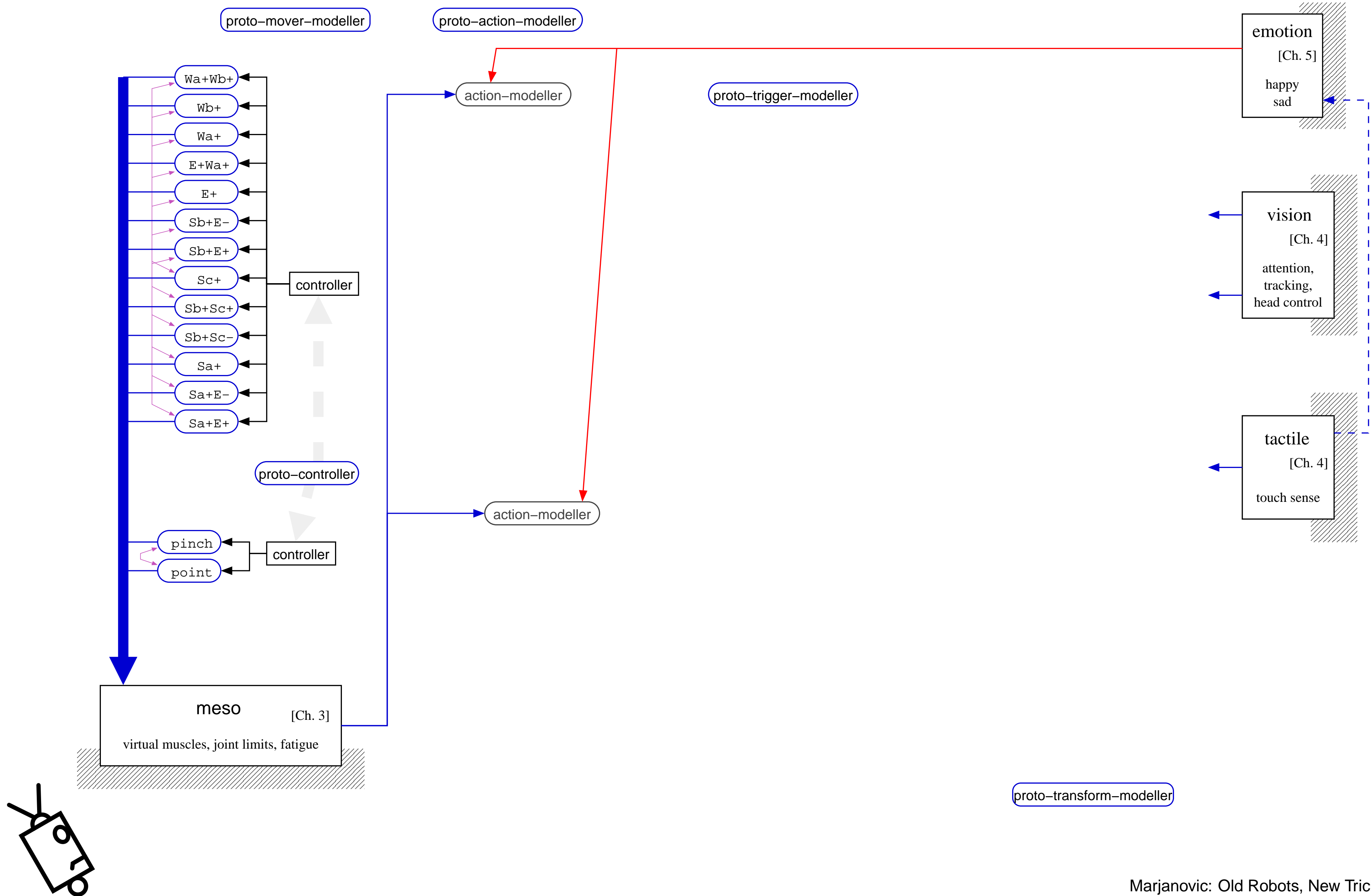


Elbow Mover (vs. Thumb)

Elbow: $R^2 = 0.82$, Thumb: $R^2 = 0.08$

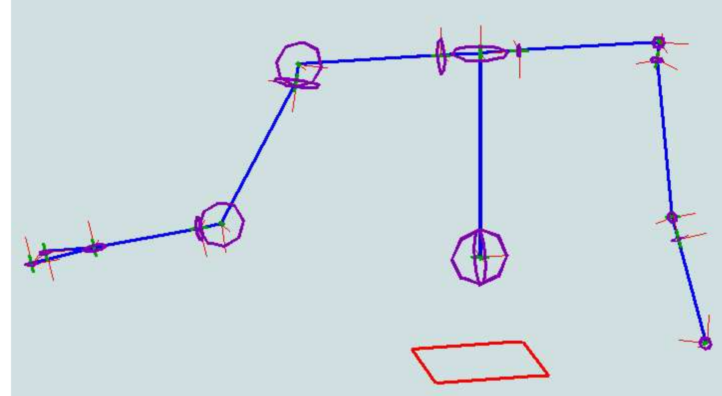


Cog, with Controllers

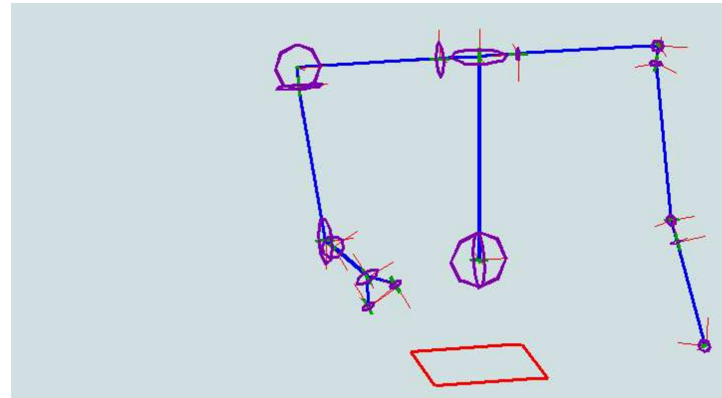


Three Arm Actions

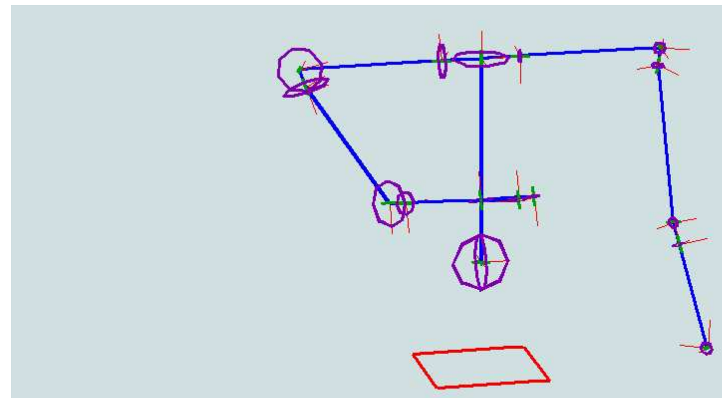
outward

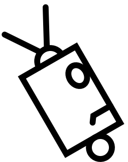


forward

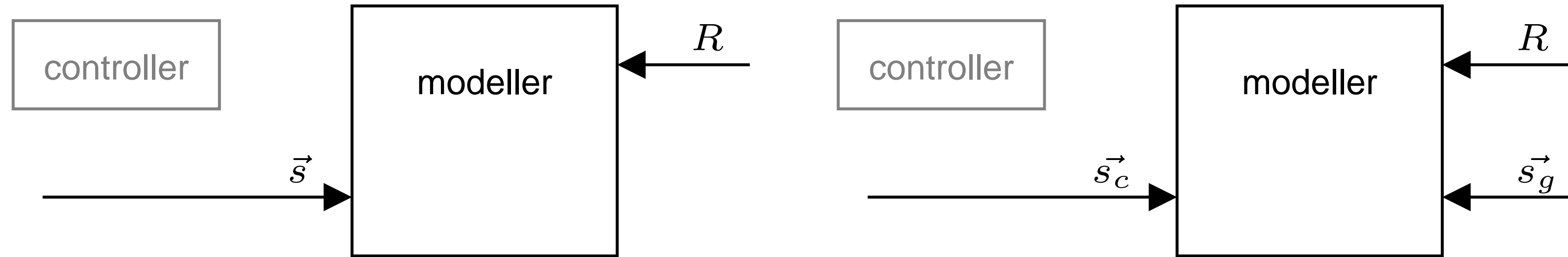


across





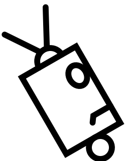
Action Modelling



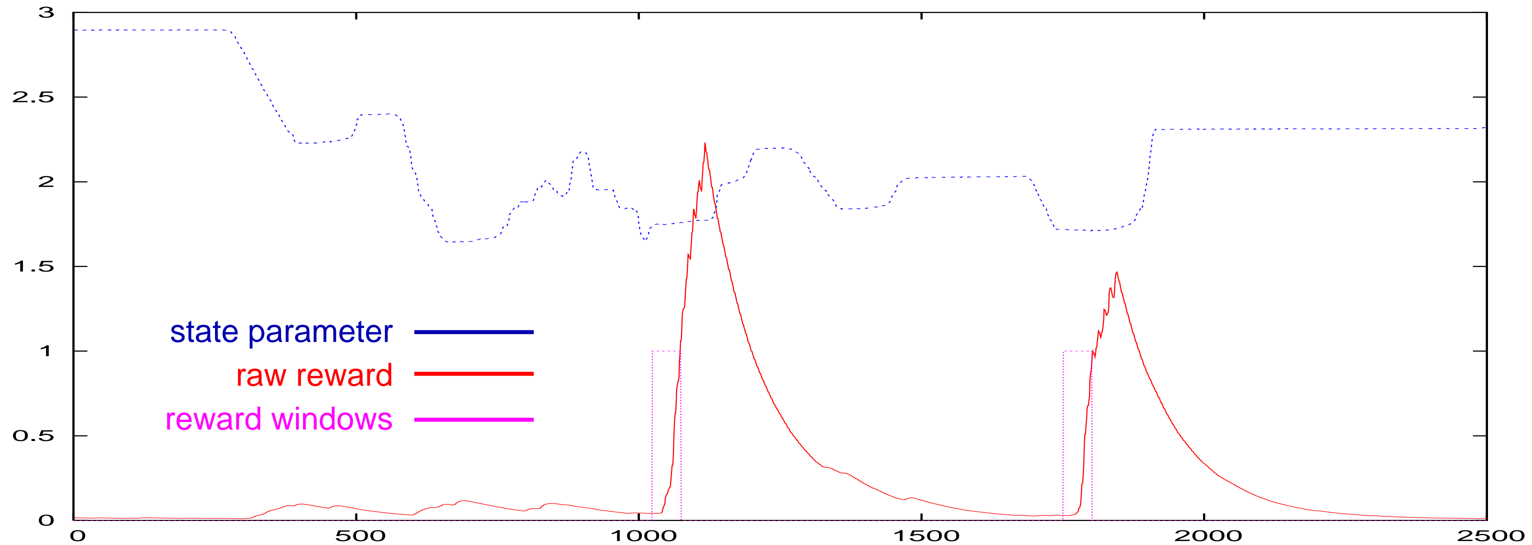
Modeller's question:

- Is there a prototype state sample being rewarded?
- If so, what is it?

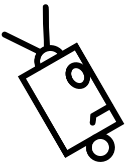
Model acts as a binary classifier: *rewarded* vs. *unrewarded*.



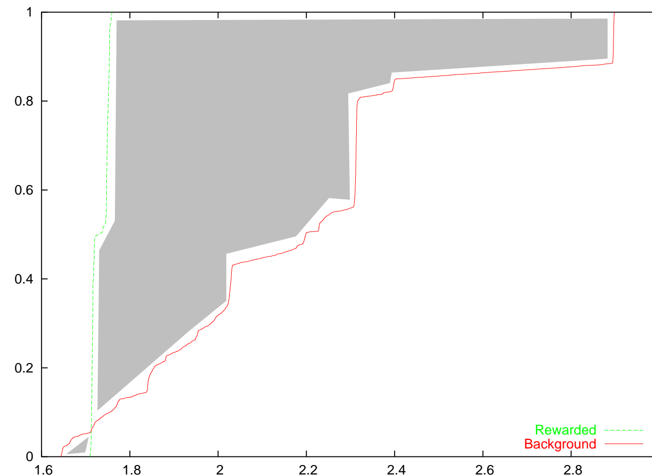
Action Modelling, cont.



- Collect samples $(\vec{s}, R)$.
- Calculate *reward windows*, preceding the reward.
- Label $\vec{s}$ samples:
  - Within windows: rewarded, “T”.
  - Outside windows: unrewarded, “F”.



Is there an Action?



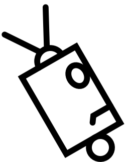
- Compare per-axis distributions $p(s_i|T)$ and $p(s_i|F)$.
  - Measure absolute areal difference of CDF's:

$$\alpha_i = \int |P(s_i|T) - P(s_i|F)| \, ds_i$$

- If $\alpha_i > 0.1$, axis i is *relevant*. (None = no model!)

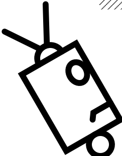
What is the Action?

- Model distributions $p(s_i|T)$ of relevant axes by gaussian.
  - Measure mean and variance of rewarded samples.
- Mean is the *prototype position* — goal of action.

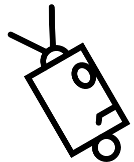


Cog, with Actors

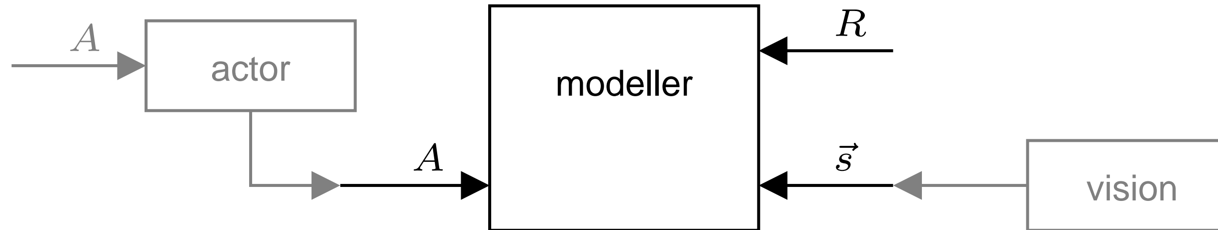
proto–mover–modeller
proto–action–modeller
emotion
[Ch. 5]
happy
sad
action–modeller
proto–trigger–modeller
Wa+Wb+
Wb+
Wa+
E+Wa+
E+
Sb+E–
Sb+E+
Sc+
Sb+Sc+
Sb+Sc–
Sa+
Sa+E–
Sa+E+
controller
actor
trigger–modeller
trigger–modeller
vision
[Ch. 4]
attention,
tracking,
head control
actor
trigger–modeller
trigger–modeller
actor
trigger–modeller
trigger–modeller
tactile
[Ch. 4]
touch sense
proto–controller
action–modeller
pinch
point
controller
actor
trigger
trigger–modeller
trigger–modeller
actor
trigger
trigger–modeller
trigger–modeller
meso
[Ch. 3]
virtual muscles, joint limits, fatigue
proto–transform–modeller



Visual Triggers for the Arm



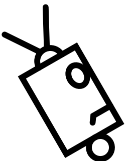
Trigger Modelling



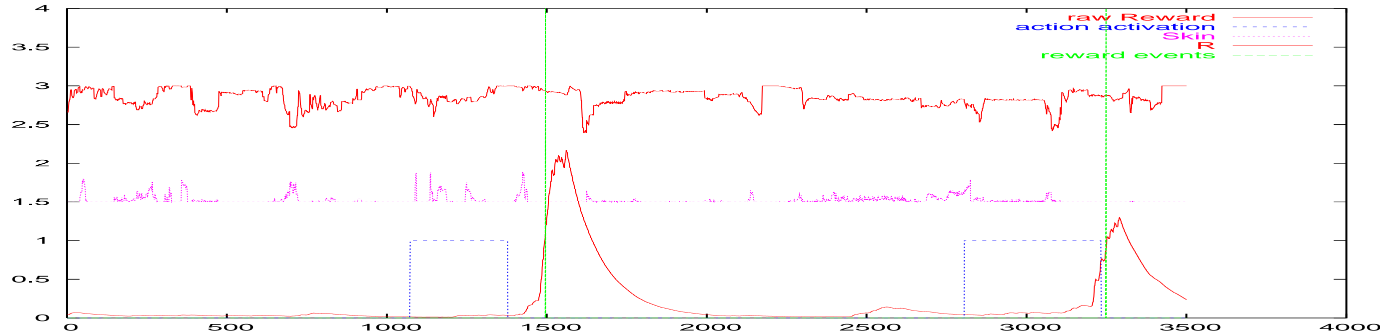
Modeller's question:

- Is there a prototype stimulus coincident with both action and reward?
- If so, what is it?

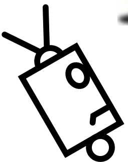
Model acts as a binary classifier: *stimulus* vs. *background*.



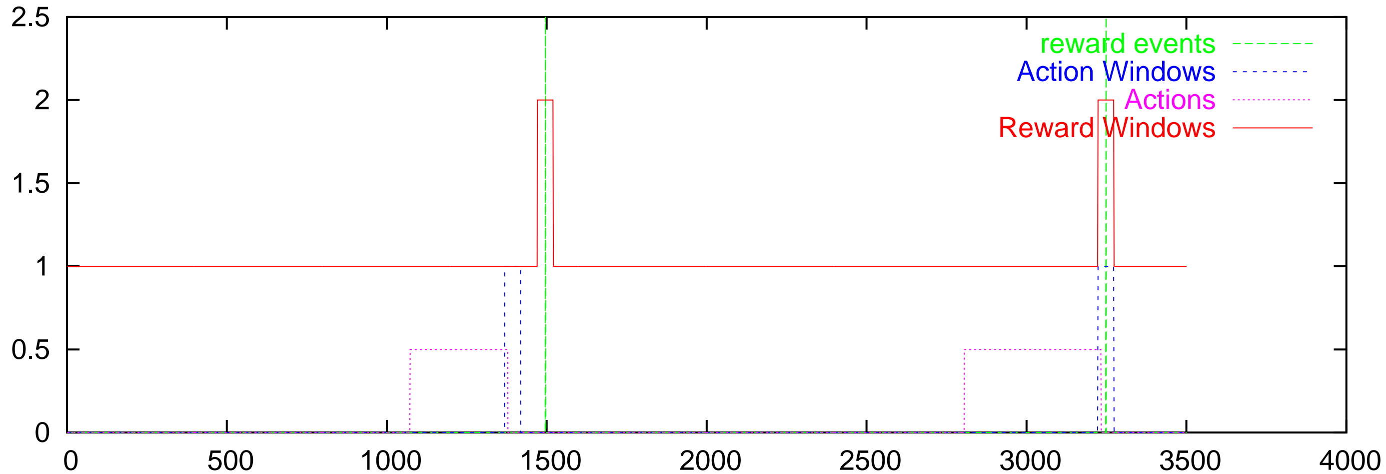
Trigger Modelling, cont.



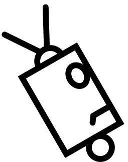
- Collect samples $(\vec{s}, A, R)$.
- Determine *reward windows* and *action windows*.
- Label samples $\vec{s}$:
  - Within both windows: stimulus, “T”.
  - Outside both windows: background, “F”.
- Compare distributions of T and F samples, per axis.



Trigger Modelling Windows



- Reward is “associated” with action if near action endpoint.
- Reward windows relative to reward event.
- Action windows = reward windows for associated rewards.

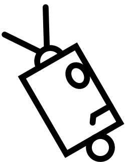


Position Trigger

- Relevant axes of T and F modelled by gaussian distributions.
- “Stimulus” and “background” distributions determine salient region of state space:

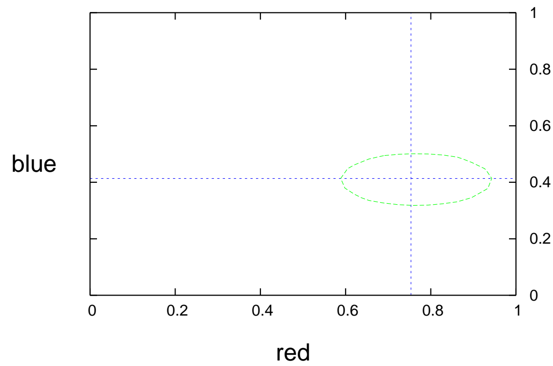
$$\rho = P(T|\vec{s}) = \frac{p(\vec{s}|T)P(T)}{p(\vec{s}|T)P(T) + p(\vec{s}|F)P(F)}$$

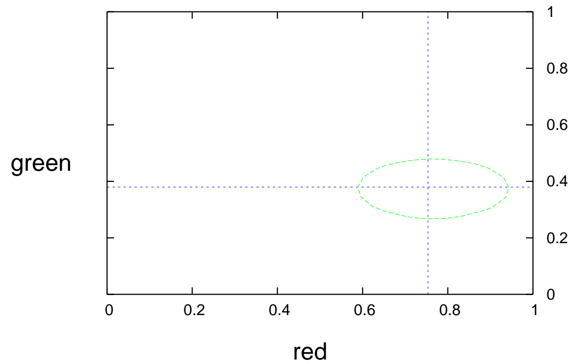
- Trigger context is satisfied when $\rho > 0.5$.
  - ρ sent as activation value to actor.
- Background distribution is updated/adapted over time.
  - Trigger will habituate to a continuous stimulus.

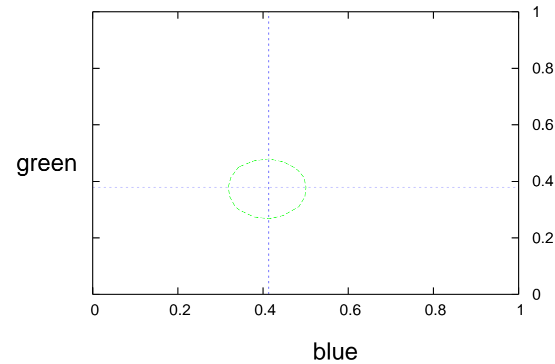


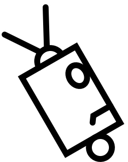
Example Trigger Context

Decision boundary of “Red Ball” trigger:



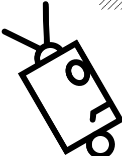




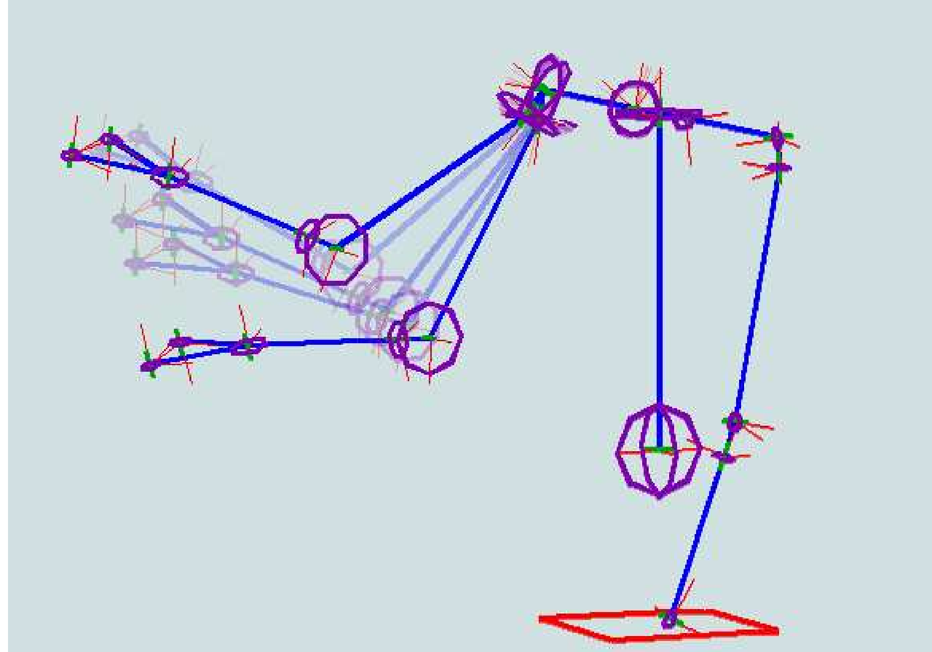


Cog, with Triggers

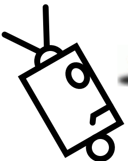
proto–mover–modeller
proto–action–modeller
emotion [Ch. 5]
happy
sad
Wa+Wb+
Wb+
Wa+
E+Wa+
E+
Sb+E−
Sb+E+
Sc+
Sb+Sc+
Sb+Sc−
Sa+
Sa+E−
Sa+E+
controller
action–modeller
actor
proto–trigger–modeller
trigger–modeller
trigger–modeller
actor
trigger
trigger–modeller
trigger–modeller
vision [Ch. 4]
attention, tracking, head control
actor
trigger
trigger–modeller
trigger–modeller
tactile [Ch. 4]
touch sense
proto–controller
action–modeller
pinch
point
controller
actor
trigger
trigger–modeller
trigger–modeller
meso [Ch. 3]
virtual muscles, joint limits, fatigue
actor
trigger
trigger–modeller
trigger–modeller
proto–transform–modeller



Action Model Refinement

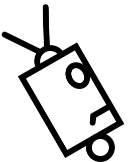


- Action modeller continues to collect data, to discover new actions.
- If a new model would be similar to an existing one, the old model is *refined*.
- Connections to triggers, etc., remain intact.



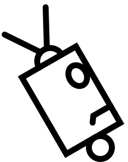
Conclusion

- Robot learns from interaction with self and environment.
- Robot can be taught a few simple behaviors.
- Holds true to design philosophy:
  - Dynamic: new models, structure created.
  - Malleable: models can be refined.
  - Distributed: no central planner/learner.
  - Real robot, real world conditions.



Future Work

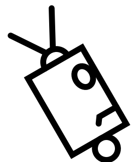
- Negative reward/reinforcement
  - Using the muscle-fatigue/joint-pain signals.
- Inhibition and un-learning
  - Learning when *not* to do something.
  - Forgetting triggers.
- Accounting/distributing reward.
- Consistency, “success” as a reinforcer.



Future Work, cont.

- Models for Johnson's "important" image schemata?

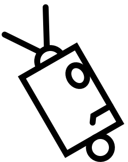
CONTAINER	BALANCE	COMPULSION
BLOCKAGE	COUNTERFORCE	RESTRAINT REMOVAL
ENABLEMENT	ATTRACTION	MASS-COUNT
PATH	LINK	CENTER-PERIPHERY
CYCLE	NEAR-FAR	SCALE
PART-WHOLE	MERGING	SPLITTING
FULL-EMPTY	MATCHING	SUPERIMPOSITION
ITERATION	CONTACT	PROCESS
SURFACE	OBJECT	COLLECTION



Future Work, cont.

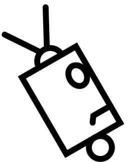
- Incorporate earlier/other work into framework:
  - Richer perceptual primitives.
  - Social primitives; emotional system.
  - Reflexes which encourage interaction.

. . . but, “black boxes” should not be opaque.

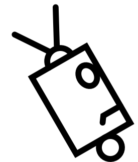


Brain-teasers. . .

- How many primitives do we need to implement?
- How small do the building blocks need to be?
- How dense must the interconnections be?
- Do we need to simulate *cells*? *Proteins*?



EXTRA SLIDES



Phases of Fatigue

discomfort
efficiency

1
0.5
0

Sl
Ss(x10)

60
40
20
0

F produced
F required

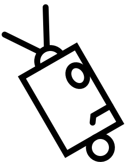
5000
0

P required

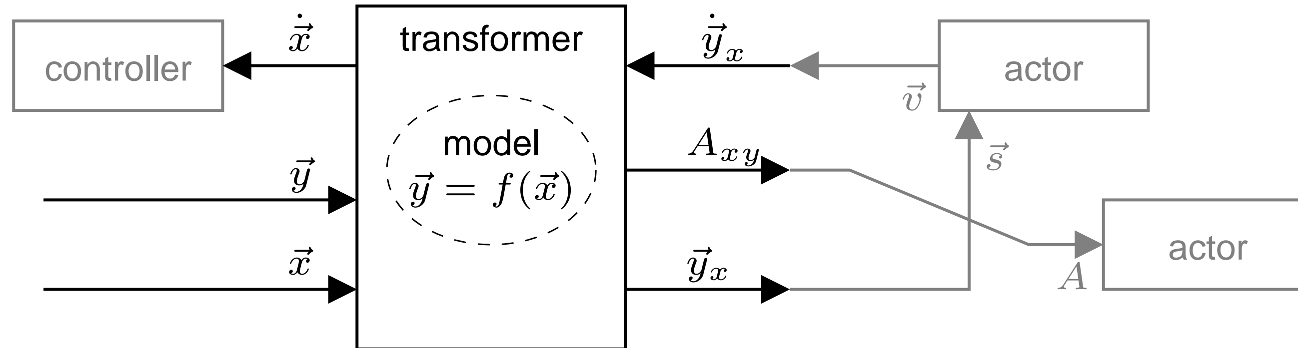
8
6
4
2
0

0 20 40 60 80 100 120 140 160

a b c d

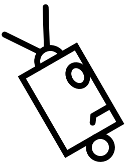


Transformer

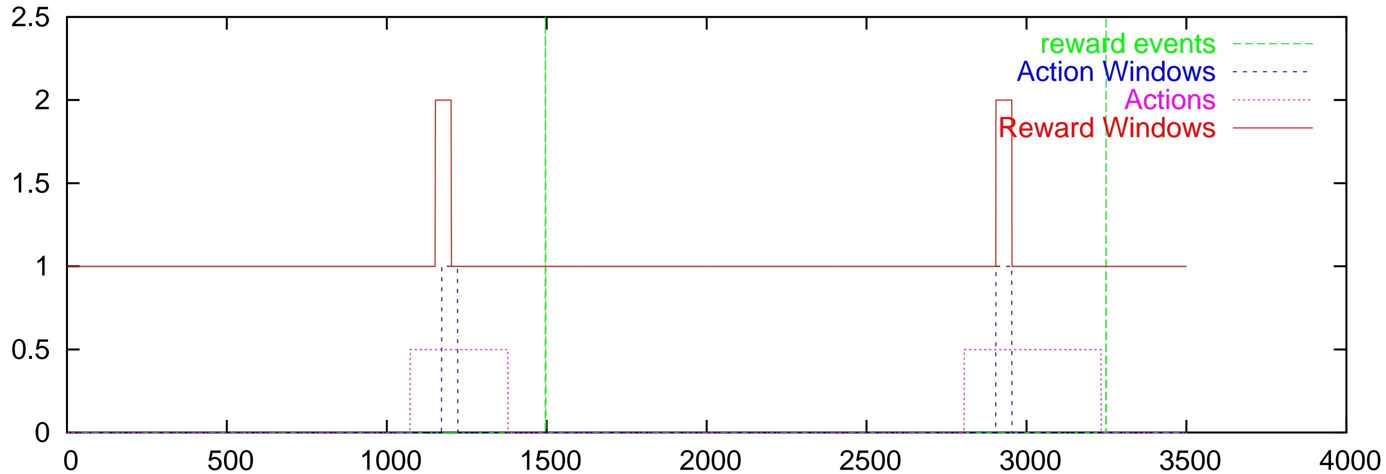


Transformer module can do three things:

- Predict state of $\vec{x}$ in y coordinate frame. ($\vec{y}_x = f(\vec{x})$)
- Decide whether or not $\vec{x}$ and $\vec{y}$ are observing the same process. (A_{xy})
- Control $\vec{x}$ via its velocity $\dot{\vec{y}}_x$ in the y coordinate frame. ($\dot{\vec{x}} = F^{-1}\dot{\vec{y}}_x$)



Trigger Modelling Windows, v1.0



- Action is “rewarded” if reward event soon after action endpoint.
- Action windows near action onset.
- Reward windows = action windows for rewarded actions.
- To be effective, need to slide action windows.