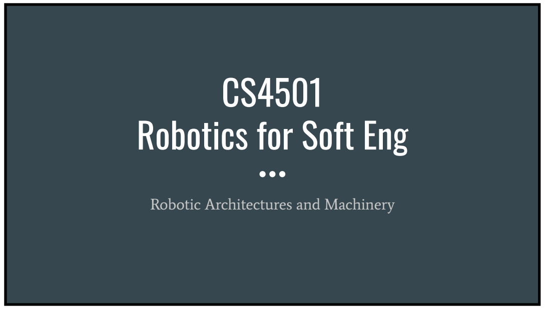
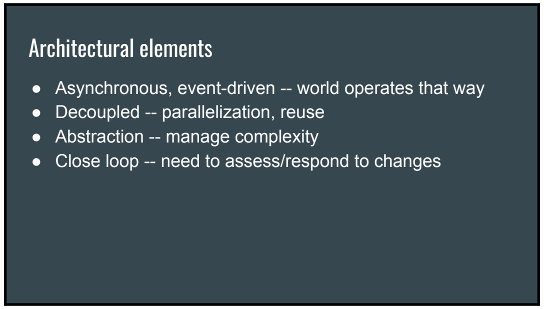


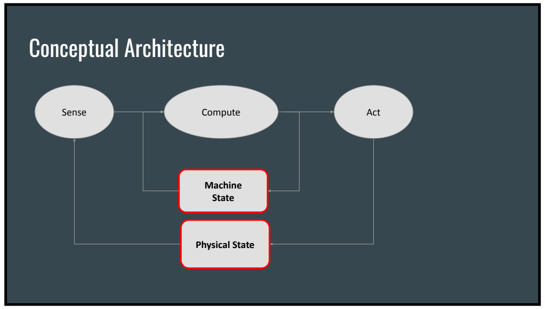
CS4501
Robotics for Soft Eng

Robotic Architectures and Machinery

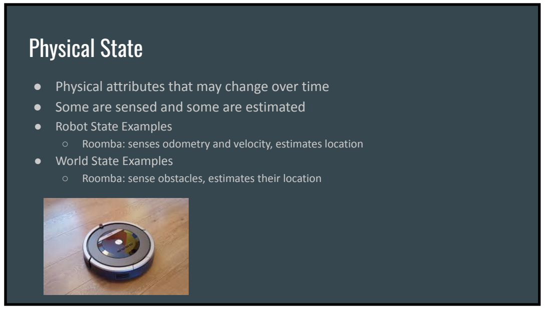


Architectural elements

- Asynchronous, event-driven -- world operates that way
- Decoupled -- parallelization, reuse
- Abstraction -- manage complexity
- Close loop -- need to assess/respond to changes

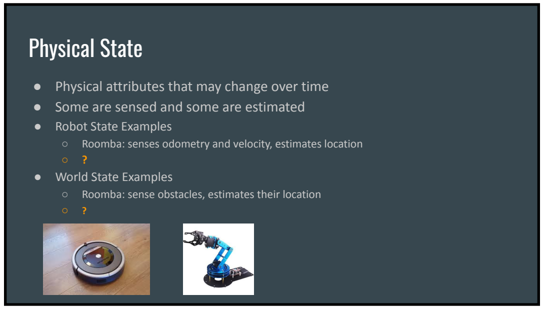


Conceptual Architecture



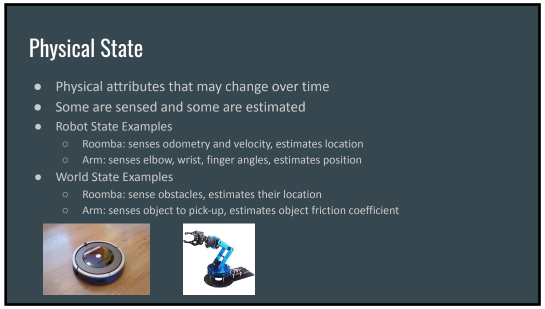
Physical State

- Physical attributes that may change over time
- Some are sensed and some are estimated
- Robot State Examples
  - Roomba: senses odometry and velocity, estimates location
- World State Examples
  - Roomba: sense obstacles, estimates their location



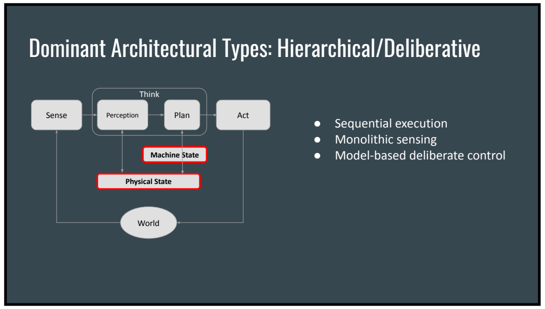
Physical State

- Physical attributes that may change over time
- Some are sensed and some are estimated
- Robot State Examples
  - Roomba: senses odometry and velocity, estimates location
  - ?
- World State Examples
  - Roomba: sense obstacles, estimates their location
  - ?



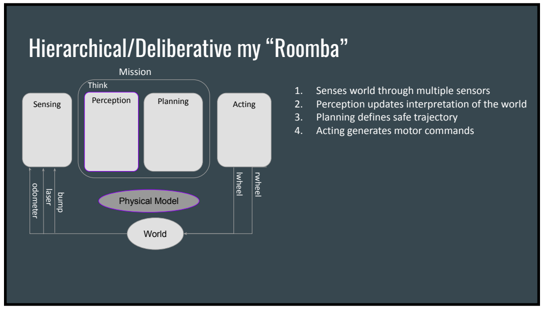
Physical State

- Physical attributes that may change over time
- Some are sensed and some are estimated
- Robot State Examples
  - Roomba: senses odometry and velocity, estimates location
  - Arm: senses elbow, wrist, finger angles, estimates position
- World State Examples
  - Roomba: sense obstacles, estimates their location
  - Arm: senses object to pick-up, estimates object friction coefficient



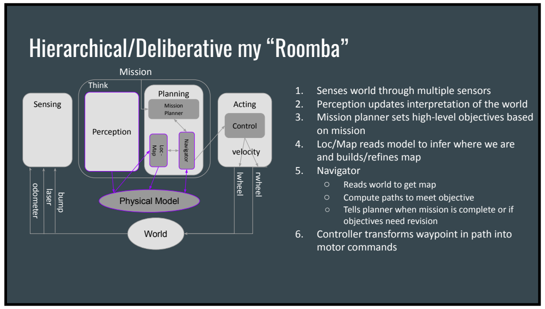
Dominant Architectural Types: Hierarchical/Deliberative

- Sequential execution
- Monolithic sensing
- Model-based deliberate control



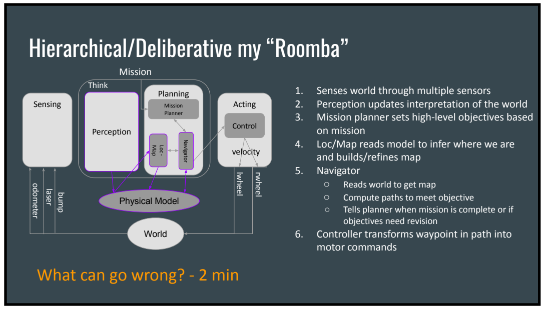
Hierarchical/Deliberative my “Roomba”

1. Senses world through multiple sensors
2. Perception updates interpretation of the world
3. Planning defines safe trajectory
4. Acting generates motor commands



Hierarchical/Deliberative my “Roomba”

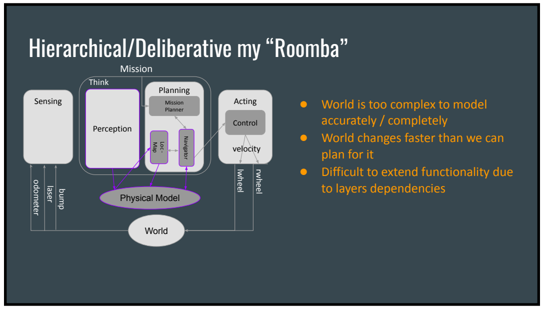
1. Senses world through multiple sensors
2. Perception updates interpretation of the world
3. Mission planner sets high-level objectives based on mission
4. Loc/Map reads model to infer where we are and builds/refines map
5. Navigator
   - Reads world to get map
   - Compute paths to meet objective
   - Tells planner when mission is complete or if objectives need revision
6. Controller transforms waypoint in path into motor commands



Hierarchical/Deliberative my “Roomba”

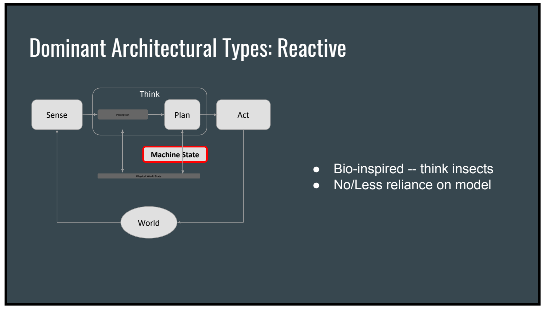
1. Senses world through multiple sensors
2. Perception updates interpretation of the world
3. Mission planner sets high-level objectives based on mission
4. Loc/Map reads model to infer where we are and builds/refines map
5. Navigator
   - Reads world to get map
   - Compute paths to meet objective
   - Tells planner when mission is complete or if objectives need revision
6. Controller transforms waypoint in path into motor commands

What can go wrong? - 2 min



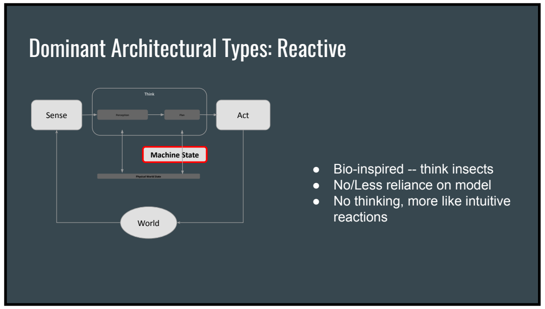
Hierarchical/Deliberative my “Roomba”

- World is too complex to model accurately / completely
- World changes faster than we can plan for it
- Difficult to extend functionality due to layers dependencies



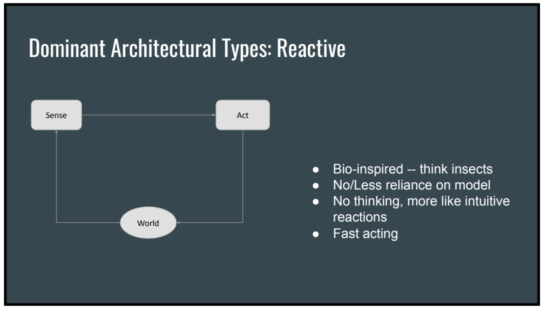
Dominant Architectural Types: Reactive

- Bio-inspired -- think insects
- No/Less reliance on model



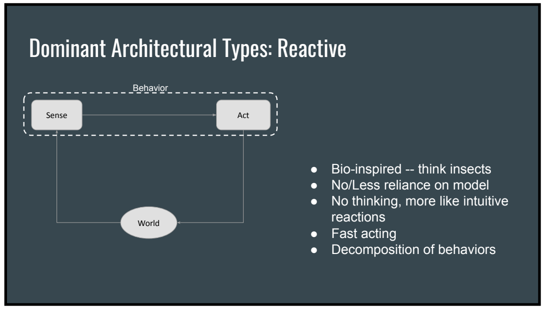
Dominant Architectural Types: Reactive

- Bio-inspired -- think insects
- No/Less reliance on model
- No thinking, more like intuitive reactions



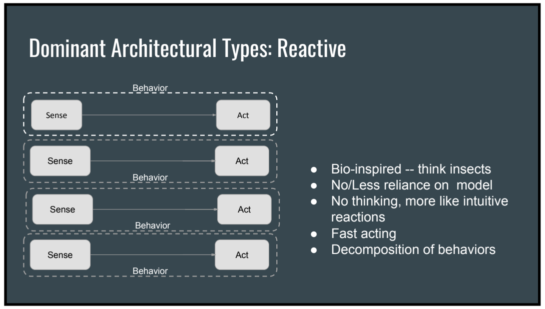
Dominant Architectural Types: Reactive

- Bio-inspired -- think insects
- No/Less reliance on model
- No thinking, more like intuitive reactions
- Fast acting



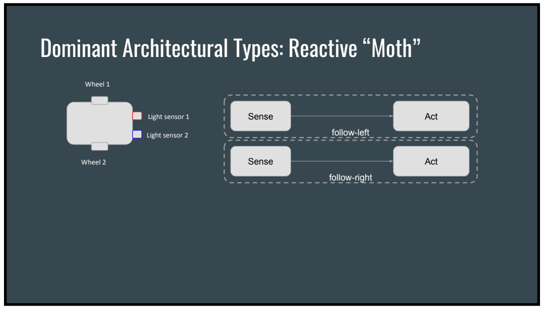
Dominant Architectural Types: Reactive

- Bio-inspired -- think insects
- No/Less reliance on model
- No thinking, more like intuitive reactions
- Fast acting
- Decomposition of behaviors

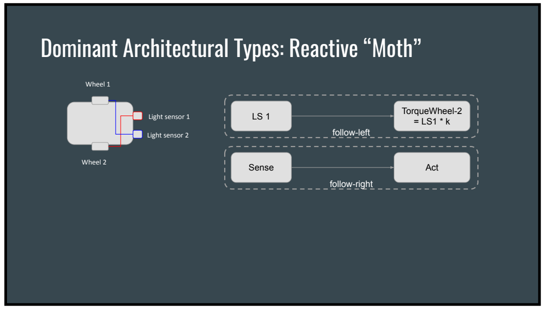


Dominant Architectural Types: Reactive

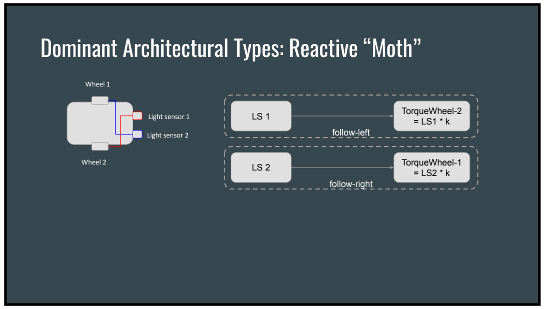
- Bio-inspired -- think insects
- No/Less reliance on model
- No thinking, more like intuitive reactions
- Fast acting
- Decomposition of behaviors



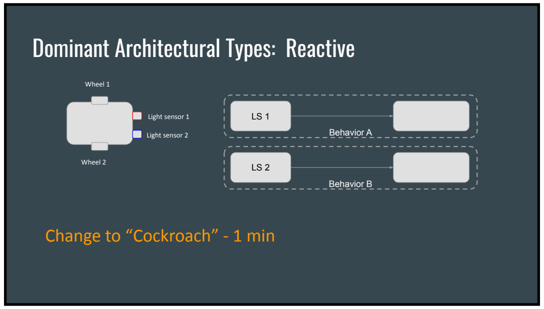
Dominant Architectural Types: Reactive “Moth”



Dominant Architectural Types: Reactive “Moth”

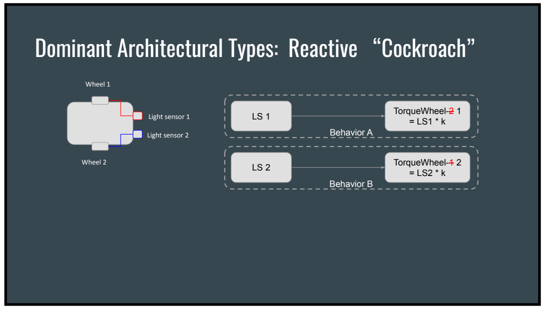


Dominant Architectural Types: Reactive “Moth”

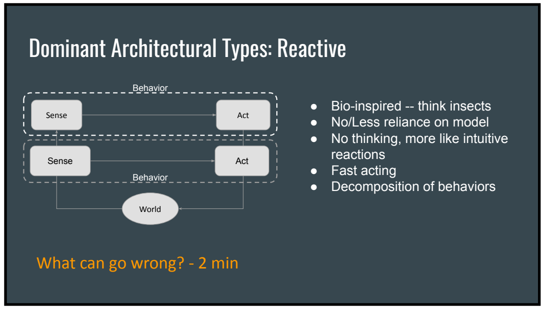


Dominant Architectural Types: Reactive

Change to “Cockroach” - 1 min



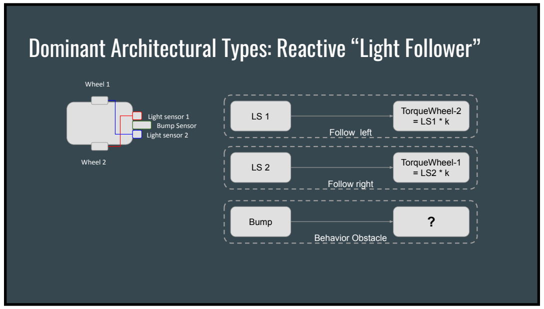
Dominant Architectural Types: Reactive “Cockroach”



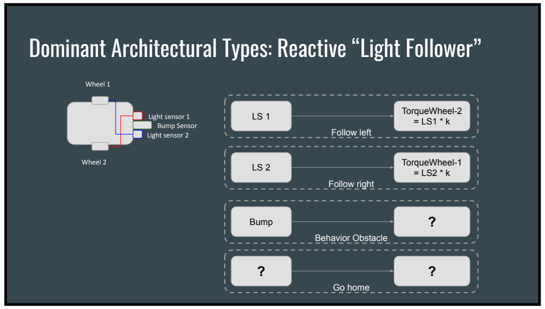
Dominant Architectural Types: Reactive

- Bio-inspired -- think insects
- No/Less reliance on model
- No thinking, more like intuitive reactions
- Fast acting
- Decomposition of behaviors

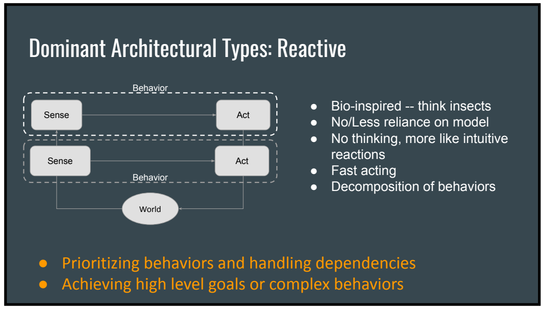
What can go wrong? - 2 min



Dominant Architectural Types: Reactive “Light Follower”



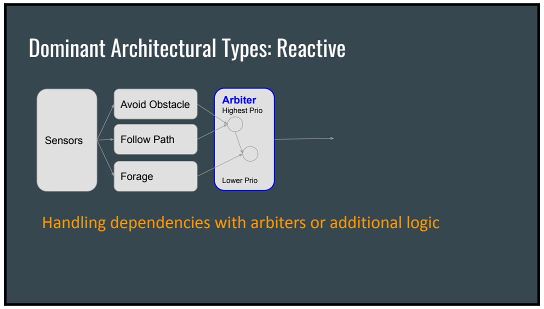
Dominant Architectural Types: Reactive “Light Follower”



Dominant Architectural Types: Reactive

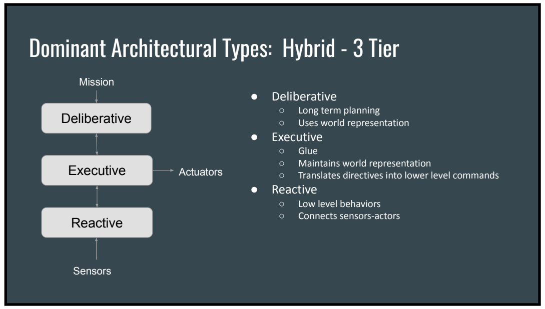
- Bio-inspired -- think insects
- No/Less reliance on model
- No thinking, more like intuitive reactions
- Fast acting
- Decomposition of behaviors

- Prioritizing behaviors and handling dependencies
- Achieving high level goals or complex behaviors



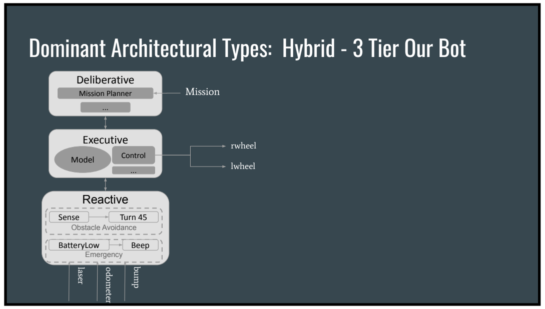
Dominant Architectural Types: Reactive

Handling dependencies with arbiters or additional logic

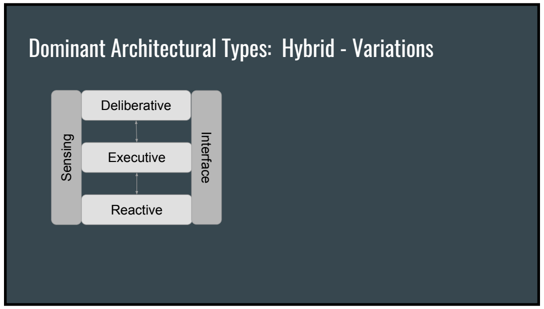


Dominant Architectural Types: Hybrid - 3 Tier

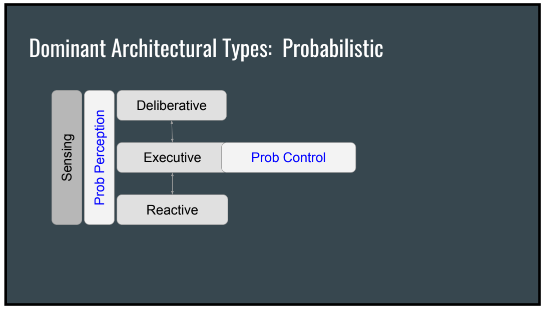
- Deliberative
  - Long term planning
  - Uses world representation
- Executive
  - Glue
  - Maintains world representation
  - Translates directives into lower level commands
- Reactive
  - Low level behaviors
  - Connects sensors-actors



Dominant Architectural Types: Hybrid - 3 Tier Our Bot

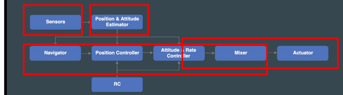


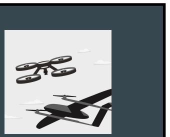
Dominant Architectural Types: Hybrid - Variations



Dominant Architectural Types: Probabilistic

Reality is a bit messier

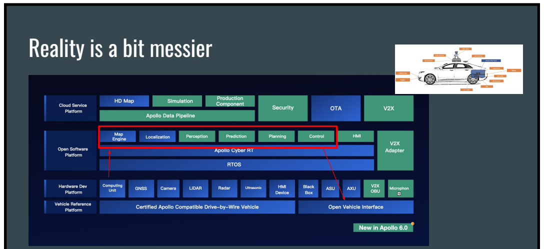




PX4 - Autopilot

https://docs.px4.io/master/en/concept/architecture.html

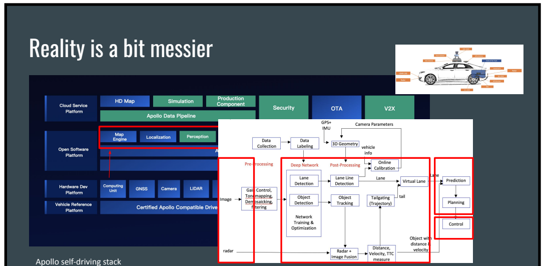
Reality is a bit messier



Apollo self-driving stack

https://github.com/ApolloAuto/apollo

Reality is a bit messier



Apollo self-driving stack

https://github.com/ApolloAuto/apollo

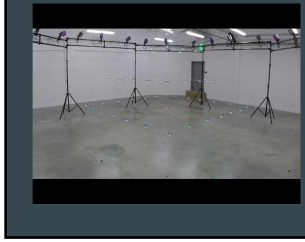
Taking stock

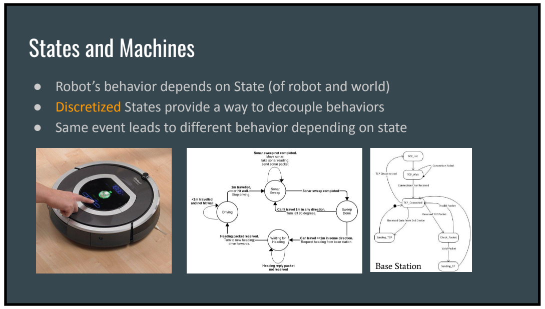
- **Deliberative**
  - Think hard, act later
  - Lots of states
  - Maps of the robot environment
  - Look ahead
- **Reactive**
  - Do not think, react
  - Less/No world states. Less/No maps. No look ahead
  - Reactive + state: Behavior, look ahead only while acting
- **Hybrid**
  - Think and act independently.
  - States. Look ahead in parallel to acting.
  - Combines long and short time scales

States and Machines

- We will learn about state estimation later
- Now states and design
  - Robot's behavior depends on State (of robot and world)
  - States provide a way to decouple behaviors
  - Same event leads to different behavior depending on state

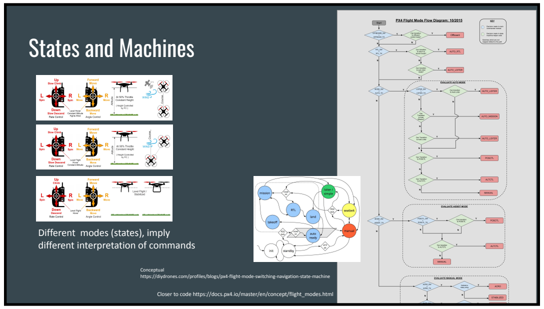
What is State





States and Machines

- Robot’s behavior depends on State (of robot and world)
- Discretized States provide a way to decouple behaviors
- Same event leads to different behavior depending on state

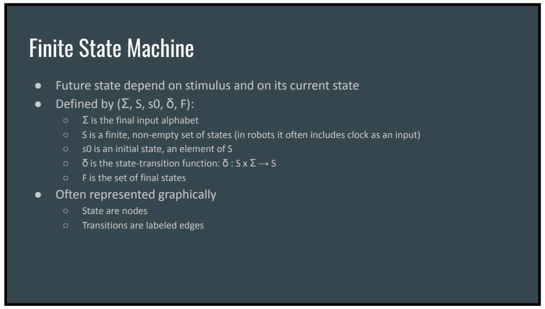


States and Machines

Different modes (states), imply different interpretation of commands

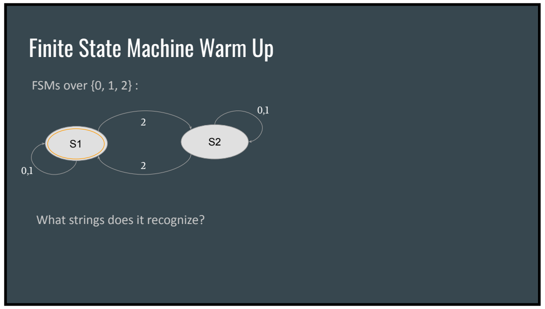
Conceptual
https://diydrones.com/profiles/blogs/px4-flight-mode-switching-navigation-state-machine

Closer to code https://docs.px4.io/master/en/concept/flight_modes.html



Finite State Machine

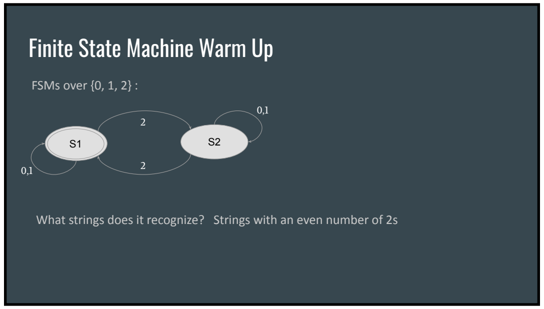
- Future state depend on stimulus and on its current state
- Defined by (Σ, S, s0, δ, F):
  - Σ is the final input alphabet
  - S is a finite, non-empty set of states (in robots it often includes clock as an input)
  - s0 is an initial state, an element of S
  - δ is the state-transition function: δ : S x Σ ⟶ S
  - F is the set of final states
- Often represented graphically
  - State are nodes
  - Transitions are labeled edges



Finite State Machine Warm Up

FSMs over {0, 1, 2} :

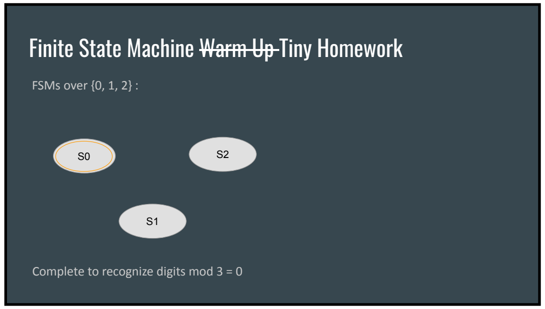
What strings does it recognize?



Finite State Machine Warm Up

FSMs over {0, 1, 2} :

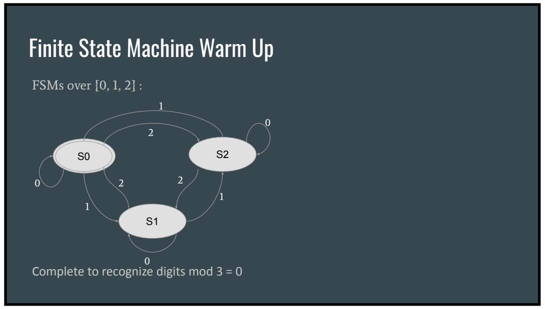
What strings does it recognize?   Strings with an even number of 2s



Finite State Machine ~~Warm Up~~ Tiny Homework

FSMs over {0, 1, 2} :

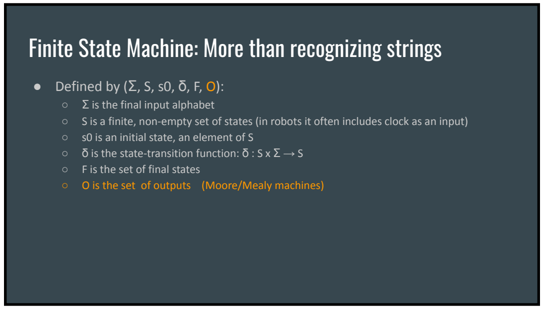
Complete to recognize digits mod 3 = 0



Finite State Machine Warm Up

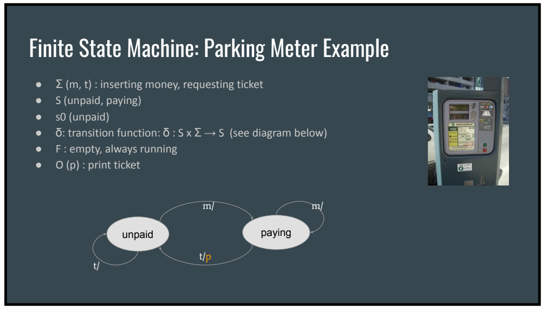
FSMs over [0, 1, 2] :

Complete to recognize digits mod 3 = 0



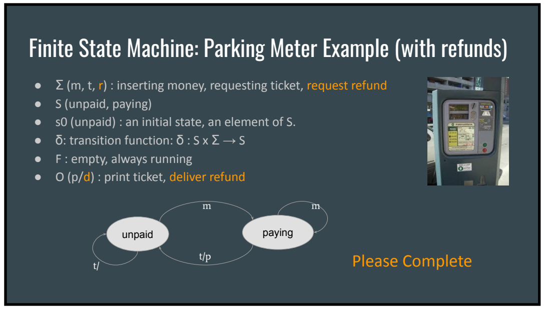
Finite State Machine: More than recognizing strings

- Defined by (Σ, S, s0, δ, F, O):
  - Σ is the final input alphabet
  - S is a finite, non-empty set of states (in robots it often includes clock as an input)
  - s0 is an initial state, an element of S
  - δ is the state-transition function: δ : S x Σ → S
  - F is the set of final states
  - O is the set of outputs   (Moore/Mealy machines)



Finite State Machine: Parking Meter Example

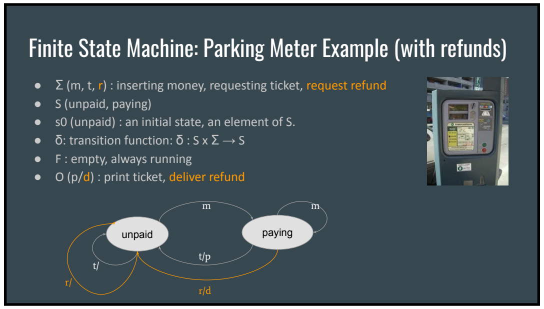
- Σ (m, t) : inserting money, requesting ticket
- S (unpaid, paying)
- s0 (unpaid)
- δ: transition function: δ : S x Σ → S (see diagram below)
- F : empty, always running
- O (p) : print ticket



Finite State Machine: Parking Meter Example (with refunds)

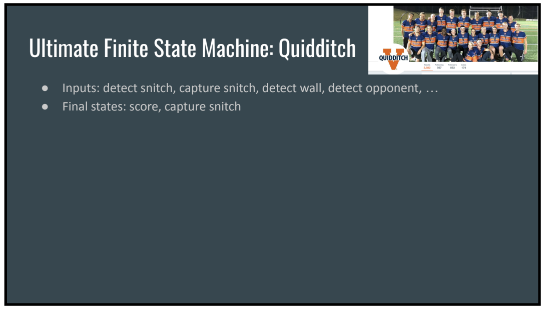
- Σ (m, t, r) : inserting money, requesting ticket, request refund
- S (unpaid, paying)
- s0 (unpaid) : an initial state, an element of S.
- δ: transition function: δ : S x Σ → S
- F : empty, always running
- O (p/d) : print ticket, deliver refund

Please Complete



Finite State Machine: Parking Meter Example (with refunds)

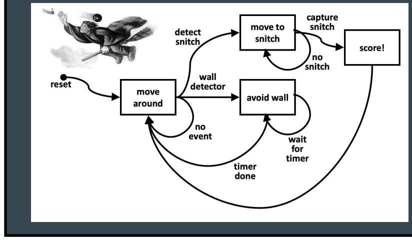
- Σ (m, t, r) : inserting money, requesting ticket, request refund
- S (unpaid, paying)
- s0 (unpaid) : an initial state, an element of S.
- δ: transition function: δ : S x Σ → S
- F : empty, always running
- O (p/d) : print ticket, deliver refund



Ultimate Finite State Machine: Quidditch

- Inputs: detect snitch, capture snitch, detect wall, detect opponent, …
- Final states: score, capture snitch

Ultimate Finite State Machine: HarryPotter



From J. McLurkin lectures
Rice University

FSM Implementation

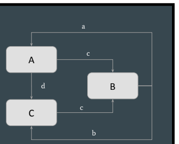
- General requirements
  - Variable to track state
  - Mechanism to set state
  - Mechanism to update a state
  - State encodings
- Styles
  - Nested switch statements (lab)
  - Table-based
  - State-based pattern (next)

FSM are hard to reuse and do not scale well

- Reuse
  - Large impact of changes
    - Adding/removing states causes changes to at least all neighbors
  - High-coupling
    - Conditions for transitions are encoded within states
- Scaling
  - N states
  - N can be large
  - NxN potential transitions

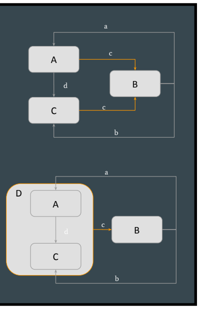
Hierarchical FSMs

- Super states
- Generalized transitions



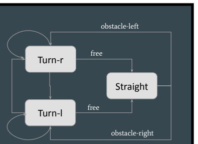
Hierarchical FSMs

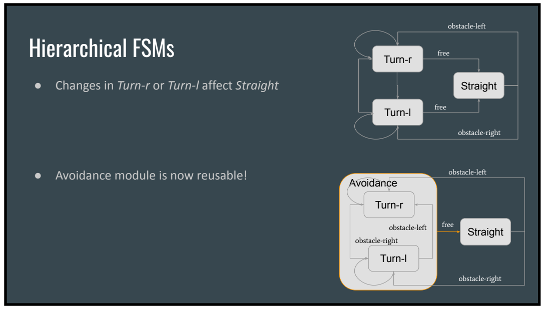
- Super states
- Generalized transitions



Hierarchical FSMs

- Changes in *Turn-r* or *Turn-l* affect *Straight*

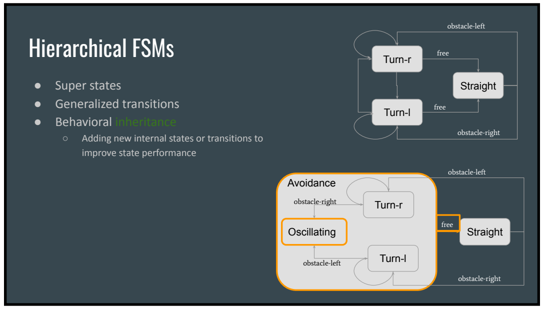




Hierarchical FSMs

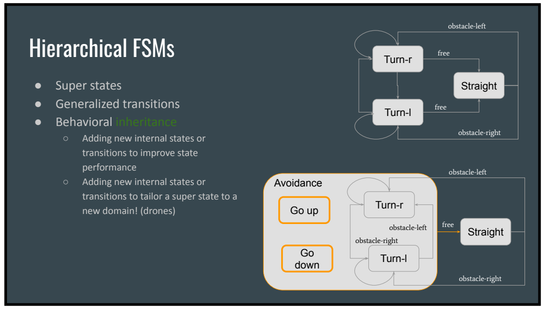
- Changes in *Turn-r* or *Turn-l* affect *Straight*

- Avoidance module is now reusable!



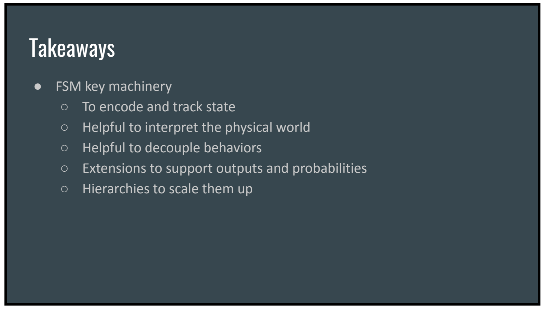
Hierarchical FSMs

- Super states
- Generalized transitions
- Behavioral inheritance
  - Adding new internal states or transitions to improve state performance



Hierarchical FSMs

- Super states
- Generalized transitions
- Behavioral inheritance
  - Adding new internal states or transitions to improve state performance
  - Adding new internal states or transitions to tailor a super state to a new domain! (drones)



Takeaways

- FSM key machinery
  - To encode and track state
  - Helpful to interpret the physical world
  - Helpful to decouple behaviors
  - Extensions to support outputs and probabilities
  - Hierarchies to scale them up