AI & Robotics: Lab Course

Brief Notes on Behavior Organization

Marc Toussaint
Technical University of Berlin
Summer 2020

- Disclaimer: This is not at all a proper lecture on behavior organization

- Disclaimer: This is not at all a proper lecture on behavior organization
- Instead, some practical comments to let you consider how to organize your code

Spaghetti Code & Finite State Machines

- All code we've seen in this course was sequential, single-threaded
  - At any time, you are at a single point of execution
  - The location of that point of execution is the 'state' of the execution
  - With if-then-else, and hierarchical method calls, you can navigate that state into any scope, or mode of operation

Spaghetti Code & Finite State Machines

- All code we've seen in this course was sequential, single-threaded
  - At any time, you are at a single point of execution
  - The location of that point of execution is the 'state' of the execution
  - With if-then-else, and hierarchical method calls, you can navigate that state into any scope, or mode of operation

- That's like a FSM!
  - You have one(!) finite state variable
  - At any time, you are in one particular state
  - Some conditions transition you between states

Spaghetti Code & Finite State Machines

- Using a formal FSM library may have advantages, allows for analysis, etc
- But in principle, any coordination you can achieve with a FSM you can also represent directly in non-threaded code

Spaghetti Code & Finite State Machines

- Using a formal FSM library may have advantages, allows for analysis, etc
- But in principle, any coordination you can achieve with a FSM you can also represent directly in non-threaded code

- So, for the sake of flexible refactoring and flexibility, I would recommend against using a formal FSM to organize your behavior

- BUT, it does make a lot of sense to think about what can be a *finite state*, and perhaps introduce variables to explicitly represent the "state" in your code

2 basic paradigms

- Single-threaded, single FSM
- ROS-like: many parallel processes or threads

2 basic paradigms

- Single-threaded, single FSM
- ROS-like: many parallel processes or threads

- Combining these views:
  - Think of each thread as its own FSM, being in a discrete state
  - The overall state is then a distributed state machine, where the state is factored or hybrid
  - Either there is a “master manager” that manages the transitions on the factored state
  - Or the processes themselves transition the factored state

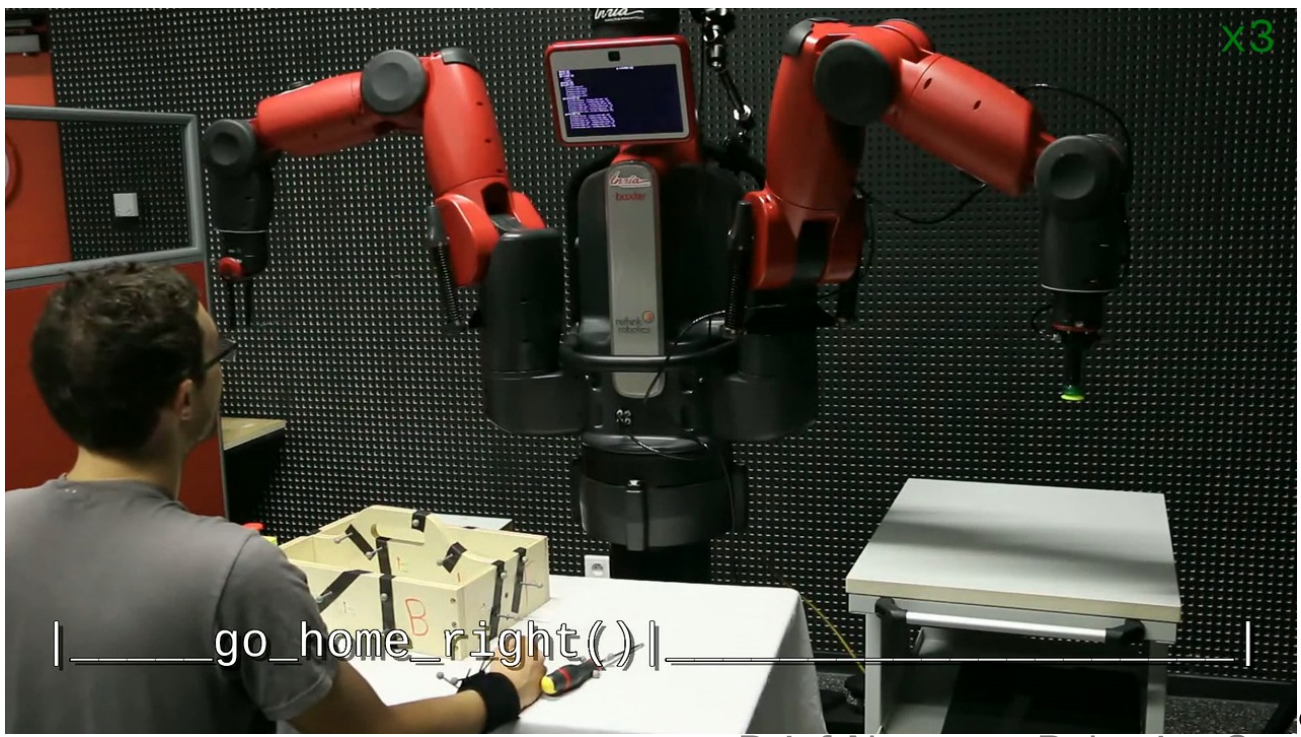
Relational Activity Processes

- Model the state of concurrent cooperation using first order logic, and formulate a semi-MDP

  Toussaint, Munzer, Mollard & Lopes: *Relational Activity Processes for Modeling Concurrent Cooperation*. ICRA'16

- Monte-Carlo Tree Search
- Imitation and Inverse RL in relational domains

  Munzer, Piot, Geist, Pietquin, Lopes: *Inverse reinforcement learning in relational domains*. IJCAI'15



Practical comments for this course

- In this course, go as far as you can with plain sequential code
- Refactor code into **reusable perceptual/manipulation routines**
- Have some main code that sequences the routines
- Perhaps it helps to explictly represent a notion of 'state'