Introduction to Cognitive Robotics

David Vernon

www.vernon.eu/cognitive_robotics

Lecture 39

www.vernon.eu/cognitive_robotics/CR_39.pdf

The CRAM Cognitive Architecture: Cognitive Robot Abstract Machine

1. Overview of CRAM
2. The main tools: Lisp, Emacs, CRAM Language, ROS
3. CRAM Beginner Tutorials with Turtlesim
4. A pick-and-place CRAM plan with a simulation of the PR2 robot

Lecture 39

The CRAM Cognitive Architecture: Cognitive Robot Abstract Machine

1. Overview of CRAM
2. ``The main tools: Lisp, Emacs, CRAM Language, ROS
3. CRAM Beginner Tutorials with Turtlesim
   – Creating a CRAM package
   – Controlling Turtlesim from CRAM
   – Implementing simple plans to move a turtle
   – Using Prolog for reasoning
   – Creating motion designators for the TurtleSim
   – Creating process modules
   – Automatically choosing a process module for a motion
   – Using location designators with the TurtleSim
   – Writing high-level plans for the TurtleSim
   – Implementing failure handling for the TurtleSim

The CRAM Beginner Tutorials

Based on CRAM tutorials
http://cram-system.org/tutorials

Automatically Choosing a Process Module for a Motion

Based on Automatically Choosing a Process Module for a Motion
http://cram-system.org/tutorials/beginner/assigning_actions_2

Automatically Choosing a Process Module for a Motion

In previous example, we called a process module explicitly using
`cram-process-modules:pm-execute` providing it with appropriate designators

Sometimes it is useful to reason about what process module to use to handle a given designator ...

Automatically Choosing a Process Module for a Motion

Setting up fact groups to select a process module

We will put this in a new file `selecting-process-modules.lisp`

Automatically Choosing a Process Module for a Motion

As before, when developing new code, we need to

- (Update the dependencies in package.xml) ← We don't need to do this as there are no new packages being used
- Update the dependencies in cram-my-beginner-tutorial.asd ← We need to do this because we are going to put the new code in a separate file
- (Update the dependencies in package.lisp) ← We don't need to do this as there are no new packages being used
- Add the new code to selecting-process-modules.lisp ← We will place the new code is a separate Lisp file
- Test the code
  - Run the ROS master
  - Run the Lisp REPL, loading the new program, creating a ROS node
  - Run turtlesim
  - Run turtlesim_teleop
  - Call the new functions

Automatically Choosing a Process Module for a Motion

Update the ASDF dependencies

Make sure you are in the cram_my_beginner_tutorial sub-directory

~$ cd ~/workspace/ros/src/cram_my_beginner_tutorial

~/workspace/ros/src/cram_my_beginner_tutorial$

Automatically Choosing a Process Module for a Motion

Update the ASDF dependencies

Edit cram-my-beginner-tutorial.asd

~/workspace/ros/src/cram_my_beginner_tutorial$ emacs cram-my-beginner-tutorial.asd

Automatically Choosing a Process Module for a Motion

Update the ASDF dependencies

```
(defsystem cram-my-beginner-tutorial
  :depends-on (roslisp cram-language
               turtlesim-msg turtlesim-srv
               cl-transforms geometry_msgs-msg
               cram-designators cram-prolog
               cram-process-modules cram-language-designator-support
               cram-executive)
  :components
  ((:module "src"
            :components
            ((:file "package")
             (:file "control-turtlesim" :depends-on ("package"))
             (:file "simple-plans" :depends-on ("package" "control-turtlesim"))
             (:file "motion-designators" :depends-on ("package"))
             (:file "process-modules" :depends-on ("package"
                                                   "control-turtlesim"
                                                   "simple-plans"
                                                   "motion-designators"))
             (:file "selecting-process-modules" :depends-on ("package"
                                                             "motion-designators"
                                                             "process-modules")))))))
```

Add this line

Add these lines

Automatically Choosing a Process Module for a Motion

Create a new Lisp file for the process modules code

Make sure you are in the cram_my_beginner_tutorial/src sub-directory

~$ cd ~/workspace/ros/src/cram_my_beginner_tutorial/src
~/workspace/ros/src/cram_my_beginner_tutorial/src$

Automatically Choosing a Process Module for a Motion

Create a new Lisp file for the process modules code

Edit selecting-process-modules.lisp

~/workspace/ros/src/cram_my_beginner_tutorial/src$ emacs selecting-process-modules.lisp

Automatically Choosing a Process Module for a Motion

Create a new Lisp file for the process modules code

Edit selecting-process-modules.lisp

Copy and paste the code from the following slide

```
(in-package :tut)

(def-fact-group available-turtle-process-modules (available-process-module
                                                  matching-process-module)
  (<- (available-process-module turtlesim-navigation))
  (<- (available-process-module turtlesim-pen-control))


  (<- (matching-process-module ?desig turtlesim-navigation)
    (desig-prop ?desig (:type :driving)))
  (<- (matching-process-module ?desig turtlesim-navigation)
    (desig-prop ?desig (:type :moving)))
  (<- (matching-process-module ?desig turtlesim-pen-control)
    (desig-prop ?desig (:type :setting-pen))))

(defun perform-some-motion (motion-desig)
  (top-level
    (with-process-modules-running (turtlesim-navigation turtlesim-pen-control)
      (cram-executive:perform motion-desig))))
```

This fact group extends `cram-process-modules:matching-process-module`
and `cram-process-modules:available-process-module`

```
(in-package :tut)

(def-fact-group available-turtle-process-modules (available-process-module
                                                  matching-process-module)
  (<- (available-process-module turtlesim-navigation))
  (<- (available-process-module turtlesim-pen-control))
```

There are two process modules available for
automatically matching them to designators

```
  (<- (matching-process-module ?desig turtlesim-navigation)
    (desig-prop ?desig (:type :driving)))
  (<- (matching-process-module ?desig turtlesim-navigation)
    (desig-prop ?desig (:type :moving)))
  (<- (matching-process-module ?desig turtlesim-pen-control)
    (desig-prop ?desig (:type :setting-pen))))
```

Use `turtlesim-navigation` with `(:type :driving)`

Use `turtlesim-navigation` with `(:type :moving)`

Use `turtlesim-pen-control` with `(:type :setting-pen)`

```
(defun perform-some-motion (motion-desig)
  (top-level
    (with-process-modules-running (turtlesim-navigation turtlesim-pen-control)
      (cram-executive:perform motion-desig))))
```

We use `perform` with a motion designator instead of calling `pm-execute` with a process
module *and* an motion designator ... allow the system to decide what process module to use

Automatically Choosing a Process Module for a Motion

Now, let's experiment with this code

First, we need to make sure a ROS master is running

If you have not already done it, open a terminal and enter

~$ roscore

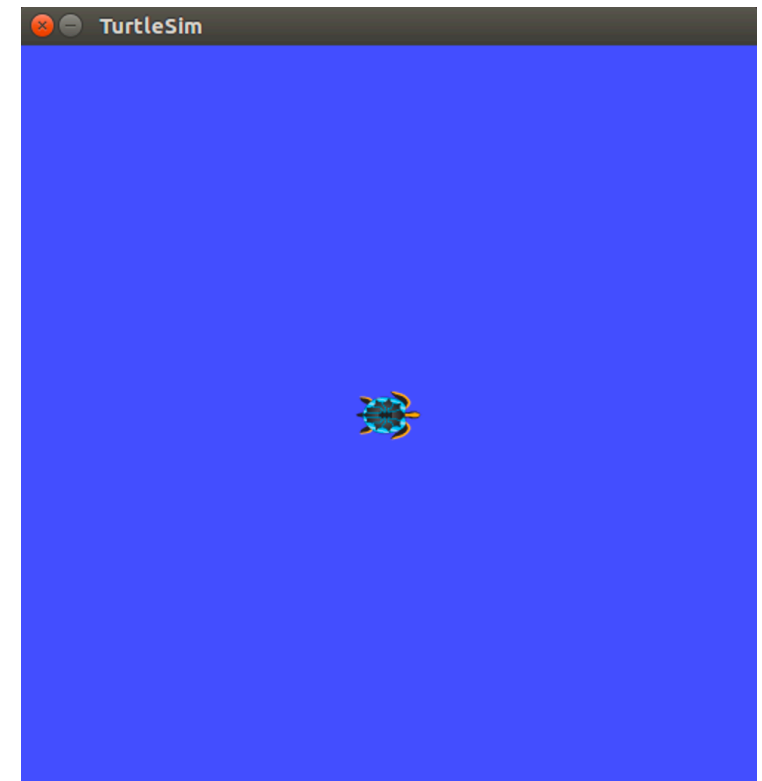# Automatically Choosing a Process Module for a Motion

Now, start turtlesim

Open a new terminal and enter

~$ rosrun turtlesim turtlesim_node

This is what you should see

Automatically Choosing a Process Module for a Motion

Launch the Lisp REPL

If you have not already done it, open a terminal and enter

~/workspace/ros$ roslisp_repl

Load the system

CL-USER> (ros-load:load-system "cram_my_beginner_tutorial" :cram-my-beginner-tutorial)

Switch to the package

CL-USER> (in-package :tut)

TUT>

Automatically Choosing a Process Module for a Motion

Start a ROS node

The name doesn't matter

TUT> (start-ros-node "turtle1")

[(ROSLISP TOP) INFO] 1292688669.674: Node name is turtle1

[(ROSLISP TOP) INFO] 1292688669.687: Namespace is /

[(ROSLISP TOP) INFO] 1292688669.688: Params are NIL

[(ROSLISP TOP) INFO] 1292688669.689: Remappings are:

[(ROSLISP TOP) INFO] 1292688669.691: master URI is 127.0.0.1:11311

[(ROSLISP TOP) INFO] 1292688670.875: Node startup complete

Automatically Choosing a Process Module for a Motion

Call the function we wrote to perform the initialization

TUT> (init-ros-turtle "turtle1")

Use turtle1 ... remember, this forms the prefix on the topic names
This is the name of the first turtle that turtlesim spawns

Automatically Choosing a Process Module for a Motion

First, let's make a couple of designators for testing

TUT> (defparameter *fast-circle* (desig:a motion (type driving) (speed 10) (angle 7)))

FAST-CIRCLE

TUT> (defparameter *goal* (desig:a motion (type moving) (goal (1 9 0))))

GOAL

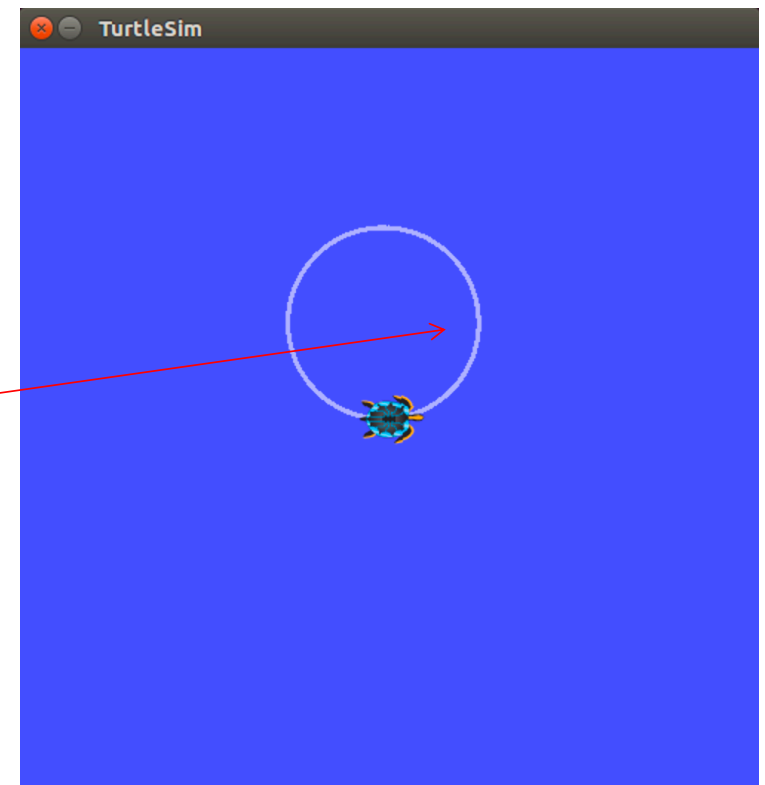TUT> (defparameter *pen-off* (desig:a motion (type setting-pen) (off 1)))

PEN-OFF

Automatically Choosing a Process Module for a Motion

Now, let's perform some motions

TUT> (perform-some-motion *fast-circle*)

The turtle drives in a circular path

Automatically Choosing a Process Module for a Motion

Now, let's perform some motions

```
TUT> (perform-some-motion *fast-circle*)
[(TURTLE-PROCESS-MODULES) INFO] 1501142699.638: TurtleSim navigation invoked with motion designator `#<MOTION-DESIGNATOR ((TYPE
                                DRIVING)
                                (SPEED
                                 10)
                                (ANGLE
                                 7)) {1009D756C3}>'.
```
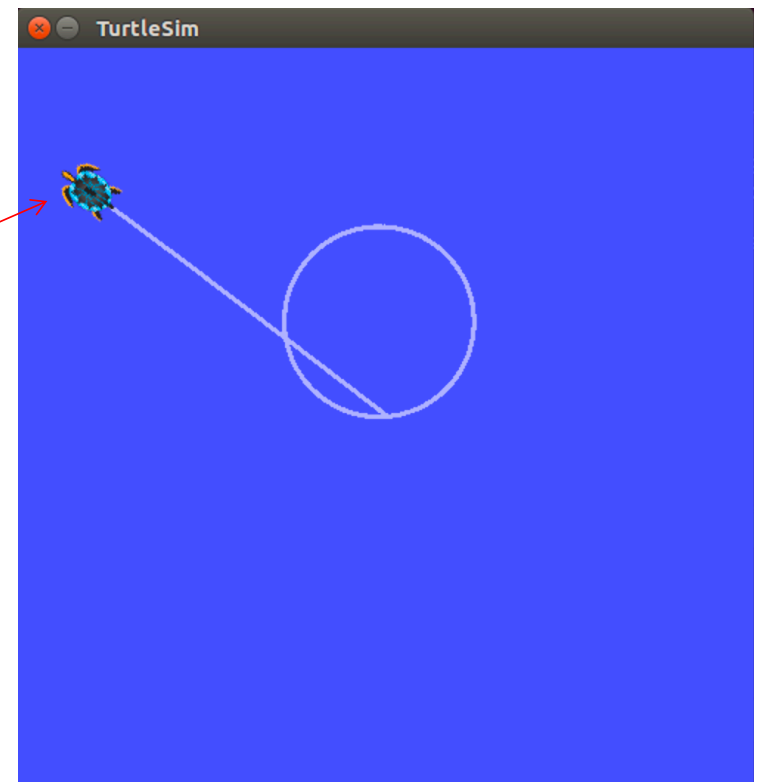
Automatically Choosing a Process Module for a Motion

Now, let's perform some motions

TUT> (perform-some-motion *goal*)

The turtle drives to the goal

Automatically Choosing a Process Module for a Motion

Now, let's perform some motions

TUT> (perform-some-motion *goal*)
[(TURTLE-PROCESS-MODULES) INFO] 1501142702.867: TurtleSim navigation invoked with motion designator `#<MOTION-DESIGNATOR ((TYPE
                         MOVING)
                         (GOAL
                         (1 9
                          0))) {10068602D3}>'.
T

Automatically Choosing a Process Module for a Motion

Now, lets turn the pen off and do the same thing again

- If the turtlesim environment gets a bit messy,
  you can clear the background by entering the following from a terminal

  ~/workspace/ros/src/cram_my_beginner_tutorial/src$ rosservice call /clear

- Or you can reset it completely by entering the following from a terminal
  (this creates a new turtle in the default pose)

  ~/workspace/ros/src/cram_my_beginner_tutorial/src$ rosservice call /reset ⟵——————— Let's do this so that the turtle is repositioned back at the centre

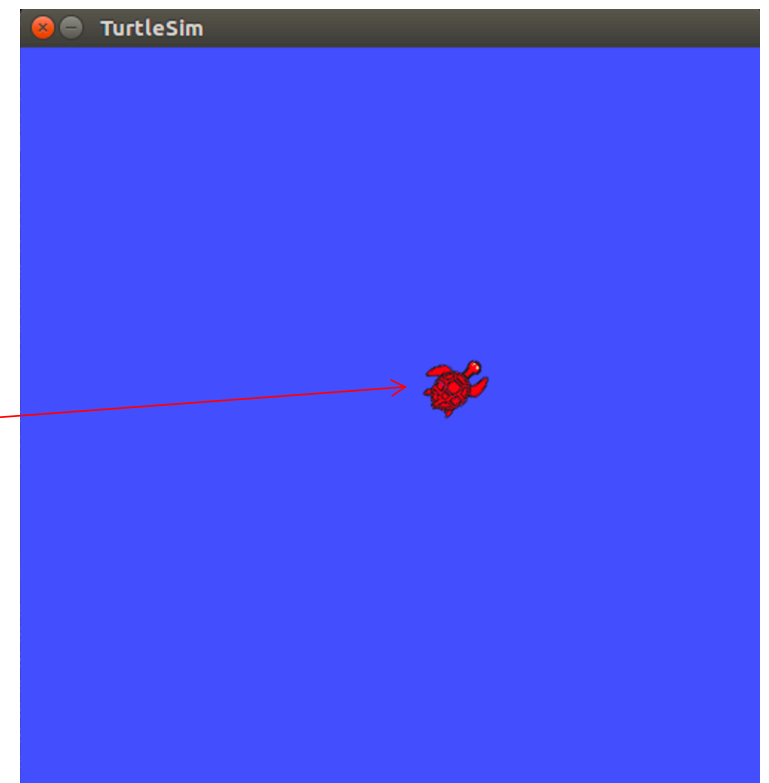# Automatically Choosing a Process Module for a Motion

Now, let's perform some motions

TUT> (perform-some-motion *pen-off*)
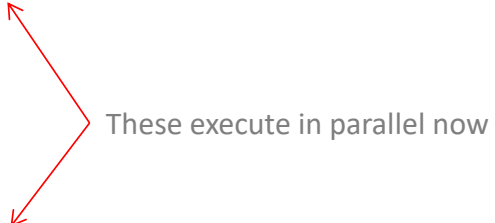
TUT> (perform-some-motion *fast-circle*)

The turtle drives in a circular path (really!)

Automatically Choosing a Process Module for a Motion

Now, lets turn the pen off and do the same thing again

TUT> (perform-some-motion *pen-off*)

[(TURTLE-PROCESS-MODULES) INFO] 1504786623.566: TurtleSim pen control invoked with motion designator `#<MOTION-DESIGNATOR ((TYPE

SETTING-PEN)

(OFF

1)) {1002BB26A3}>'.

[TURTLESIM-SRV:SETPEN-RESPONSE]

TUT> (perform-some-motion *fast-circle*)

[(TURTLE-PROCESS-MODULES) INFO] 1504786630.845: TurtleSim navigation invoked with motion designator `#<MOTION-DESIGNATOR ((TYPE

DRIVING)

(SPEED

10)

(ANGLE

7)) {1002BB2CD3}>'.

These execute in parallel now

Automatically Choosing a Process Module for a Motion

Turn the pen back on

To to this here, we create a new motion designator

TUT> (perform-some-motion (desig:a motion (type setting-pen) (off 0)))

[(TURTLE-PROCESS-MODULES) INFO] 1504786654.744: TurtleSim pen control invoked with motion designator `#<MOTION-DESIGNATOR ((TYPE

SETTING-PEN)

(OFF

0)) {1002D6D763}>'.

[TURTLESIM-SRV:SETPEN-RESPONSE]

CRAM Beginner Tutorials

Create a CRAM Package   http://cram-system.org/tutorials/beginner/package_for_turtlesim

Controlling turtlesim from CRAM   http://cram-system.org/tutorials/beginner/controlling_turtlesim_2

Implementing simple plans to move a turtle   http://cram-system.org/tutorials/beginner/simple_plans

Using Prolog for reasoning   http://cram-system.org/tutorials/beginner/cram_prolog

Creating motion designators for the TurtleSim   http://cram-system.org/tutorials/beginner/motion_designators

Creating process modules   http://cram-system.org/tutorials/beginner/process_modules_2

Automatically choosing a process module for a motion   **http://cram-system.org/tutorials/beginner/assigning_actions_2**

Background Reading

G. Kazhoyan, Lecture notes: Robot Programming with Lisp 7. Coordinate Transformations,
　　　TF, ActionLib, slides 5-8.
　　　`https://ai.uni-bremen.de/_media/teaching/7_more_ros.pdf`

`http://wiki.ros.org/tf/Overview/Transformations`

T. Rittweiler, CRAM – Design and Implementation of a Reactive Plan Language, Bachelor
　　　Thesis, Technical University of Munich, 2010.
　　　`https://common-lisp.net/~trittweiler/bachelor-thesis.pdf`