

Introduction to Cognitive Robotics

Module 10: Using Turtlesim with CRAM

Lecture 2: Turtlesim with CRAM; creating a CRAM package; controlling a turtle

David Vernon
Carnegie Mellon University Africa

www.vernon.eu

The CRAM Beginner Tutorials

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

Creating a CRAM Package

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

Creating a CRAM Package

Set up a ROS package to use the CRAM plan language within the Lisp REPL

1. Create a ROS package that depends on `cram_language`

```
~$ cd ~/workspace/ros/src
```

```
~/workspace/ros/src$ catkin_create_pkg cram_my_beginner_tutorial cram_language
```

The second argument identifies a dependency on `cram_language`



We name the package `cram_my_beginner_tutorial`



Creating a CRAM Package

Set up a ROS package to use the CRAM plan language within the Lisp REPL

1. Create a ROS package that depends on `cram_language`

```
~$ cd ~/workspace/ros/src
```

```
~/workspace/ros/src$ catkin_create_pkg cram_my_beginner_tutorial cram_language
```

The second argument identifies a dependency on `cram_language`



We name the package `cram_my_beginner_tutorial`




This creates

- a directory `cram_my_beginner_tutorial` to hold the package, containing the following files
- `package.xml` a configuration file containing the manifest we discussed earlier
- `CMakeLists.txt` a script for CMake, an industrial-strength build system used by ROS
- and the following sub-directories `src` and `include`

Creating a CRAM Package

Set up a ROS package to use the CRAM plan language within the Lisp REPL

2. Set up the Lisp infrastructure ...

- Create an ASDF system
 - Create the Lisp package
 - Export the ASDF system to ROS
- Another System Definition Facility
- 

Creating a CRAM Package

Create an ASDF system

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$
```

Creating a CRAM Package

Create an ASDF system

Edit `cram-my-beginner-tutorial.asd`

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



NB: use hyphens, not underscores
Why? Because we will define a system in this file with the same name as the filename and, In general, we don't use underscore characters in Lisp

Creating a CRAM Package

Create an ASDF system

Edit `cram-my-beginner-tutorial.asd`

Enter the following lines

```
(defsystem cram-my-beginner-tutorial
  :depends-on (cram-language)
  :components
  ((:module "src"
    :components
    ((:file "package")
     (:file "control-turtlesim" :depends-on ("package"))))))
```

The name of the system (with hyphens)

We need to create `package.lisp` next

We also need to create `control-turtlesim.lisp`

Creating a CRAM Package

Create the Lisp package

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$
```

Creating a CRAM Package

Create the Lisp package

Edit `package.lisp`

```
~/workspace/ros/src/cram_my_beginner_tutorial/src$ emacs package.lisp
```

Creating a CRAM Package

Create the Lisp package

Edit `package.lisp`


Enter the following lines

```
(defpackage :cram-my-beginner-tutorial  
  (:nicknames :tut)  
  (:use :cpl))
```

The name of the system (with hyphens)



Define a nickname for `cram-my-beginner-tutorial` package



Nickname for the `cram_language` package



Creating a CRAM Package

Create the Lisp package

Edit **control-turtlesim.lisp**

~/workspace/ros/src/cram_my_beginner_tutorial/src\$ **emacs control-turtlesim.lisp**

NB: use hyphens, not underscores



Creating a CREAM Package

Create the Lisp package

Edit **control-turtlesim.lisp**

Enter the following lines

```
(in-package :tut)
```

This just selects the namespace using the nickname



We will add more code later



Creating a CRAM Package

Export the ASDF system to ROS

Make sure you are in the workspace directory

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

Alternatively

```
~/workspace/ros/src/cram_my_beginner_tutorial/src$ roscd  
~/workspace/ros/devel$ cd ..  
~/workspace/ros$
```

Build the workspace to compile the program:
run `catkin_make`

```
~/workspace/ros$ catkin_make
```

Creating a CRAM Package

Launch the Lisp REPL:

```
~/workspace/ros$ roslisp_repl
```

Load the system:

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

Check that everything is fine (it won't do much):

```
CL-USER> (in-package :tut)  
TUT>
```


Controlling Turtlesim from CRAM

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

Controlling Turtlesim from CRAM

Now, let's develop some code to interact with Turtlesim

To do this:

- Update the dependencies in `package.xml`
- Update the dependencies in `cram-my-beginner-tutorial.asd`
- Update the dependencies in `package.lisp`
- Add the new code to `control-turtlesim.lisp`
- Test the code
 - Run the ROS master
 - Run the Lisp REPL, loading the new program, creating a ROS node
 - Run turtlesim
 - Run turtlesim_teleop
 - Calling the new functions

Controlling Turtlesim from CRAM

Update the ROS 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$
```

Controlling Turtlesim from CRAM

Update the ROS dependencies

Edit `package.xml`

```
~/workspace/ros/src/cram_my_beginner_tutorial$ emacs package.xml
```

Controlling Turtlesim from CRAM

Update the ROS dependencies

Edit **package.xml**

Add the following lines

Add after this line

`<exec_depend>cram_language</exec_depend>`

`<depend>turtlesim</depend>`

`<depend>roslisp</depend>`

`<depend>cl_transforms</depend>`

`<depend>geometry_msgs</depend>`

← We will use the turtlesim simulator

← We need to use roslisp

← These are used to specify pose

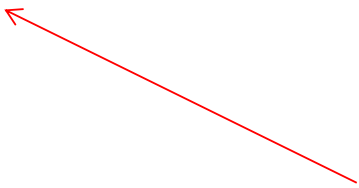
Controlling Turtlesim from CRAM

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$
```

You should be there already
from the previous step



Controlling Turtlesim from CRAM

Update the ASDF dependencies

Edit `cram-my-beginner-tutorial.asd`

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

Controlling Turtlesim from CRAM

Update the ASDF dependencies

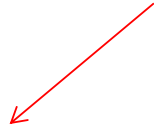
Edit `cram-my-beginner-tutorial.asd`

Replace the `:depends-on (...)` lines with the text below below:

```
(defsystem cram-my-beginner-tutorial
  :depends-on (roslisp cram-language
              turtlesim-msg turtlesim-srv
              cl-transforms geometry_msgs-msg)

  :components
  ((:module "src"
    :components
    ((:file "package")
     (:file "control-turtlesim" :depends-on ("package"))))))
```

The file should now
look like this



Controlling Turtlesim from CRAM

Update the Lisp package to add to `roslisp` and `cl-transforms` to the namespace

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$
```

Controlling Turtlesim from CRAM

Update the Lisp package to add to `roslisp` and `cl-transforms` to the namespace

Edit `package.lisp`

```
~/workspace/ros/src/cram_my_beginner_tutorial/src$ emacs package.lisp
```

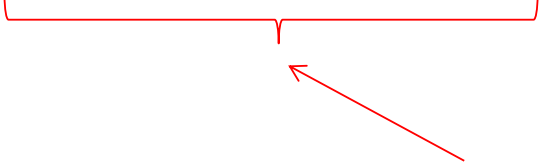
Controlling Turtlesim from CRAM

Update the Lisp package to add to `roslisp` and `cl-transforms` to the namespace

Edit `package.lisp`

Add `:roslisp` `:cl-transforms` to the `(:use :cpl)` line:

```
(defpackage :cram-my-beginner-tutorial
  (:nicknames :tut)
  (:use :cpl :roslisp :cl-transforms))
```



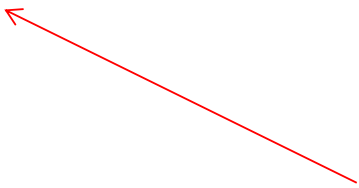
Add these to the namespace so that we don't have to include the package name when we use the functions in the control program

Controlling Turtlesim from CRAM

Update the Lisp package to allow it to communicate with turtlesim

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$
```



You should be there already
from the previous step

Controlling Turtlesim from CRAM

Update the Lisp package to allow it to communicate with turtlesim

Edit `control-turtlesim.lisp`

```
~/workspace/ros/src/cram_my_beginner_tutorial/src$ emacs control-turtlesim.lisp
```

Controlling Turtlesim from CRAM

Update the Lisp package to allow it to communicate with turtlesim

Edit `control-turtlesim.lisp`

Add the code on the next two slides ...

```

(in-package :tut)

(defvar *color-value* (make-fluent :name :color-value) "current color under the turtle")
(defvar *turtle-pose* (make-fluent :name :turtle-pose) "current pose of turtle")

(defvar *color-sub* nil "color ROS subscriber")
(defvar *pose-sub* nil "pose ROS subscriber")
(defvar *cmd-vel-pub* nil "velocity commands ROS publisher")

(defvar *pen-srv* nil "name of ROS service for controlling the pen")

(defun init-ros-turtle (name)
  "Subscribes to topics for a turtle and binds callbacks.
  'name' specifies the name of the turtle."

  (setf *color-sub* (subscribe (format nil "~a/color_sensor" name)
                              "turtlesim/Color"
                              #'color-cb))

  (setf *pose-sub* (subscribe (format nil "~a/pose" name)
                              "turtlesim/Pose"
                              #'pose-cb))

  (setf *cmd-vel-pub* (advertise (format nil "~a/cmd_vel" name)
                                "geometry_msgs/Twist"))

  (setf *pen-srv* (concatenate 'string "/" name "/set_pen")))

```

```

(defun color-cb (msg)
  "Callback for color values. Called by the color topic subscriber."
  (setf (value *color-value*) msg))

(defun pose-cb (msg)
  "Callback for pose values. Called by the pose topic subscriber."
  (setf (value *turtle-pose*) msg))

(defun send-vel-cmd (lin ang)
  "Function to send velocity commands."
  (publish *cmd-vel-pub*
    ;; short syntax:
    ;; (make-message "geometry_msgs/Twist" (:x :linear) lin (:z :angular) ang)
    ;; more understandable syntax:
    (make-message "geometry_msgs/Twist"
      :linear (make-msg "geometry_msgs/Vector3" :x lin)
      :angular (make-msg "geometry_msgs/Vector3" :z ang))))

(defun call-set-pen (r g b width off)
  "Function to call the SetPen service."
  (call-service *pen-srv* 'turtlesim-srv:SetPen
    :r r
    :g g
    :b b
    :width width
    :off off))

```


`(in-package :tut)` ← Already there from when we created this file

`(defvar *color-value* (make-fluent :name :color-value) "current color under the turtle")`
`(defvar *turtle-pose* (make-fluent :name :turtle-pose) "current pose of turtle")` } ← Create two fluents
so that we are informed of any
changes in the color or pose

`(defvar *color-sub* nil "color ROS subscriber")`
`(defvar *pose-sub* nil "pose ROS subscriber")`
`(defvar *cmd-vel-pub* nil "velocity commands ROS publisher")` } ← Create global variables for the `color-sub` and `pose-sub` subscribers
and the `cmd_vel` publisher

`(defvar *pen-srv* nil "name of ROS service for controlling the pen")` ← Create global variables for the `pen` service

```
(defun init-ros-turtle (name)
  "Subscribes to topics for a turtle and binds callbacks.
  'name' specifies the name of the turtle."

  (setf *color-sub* (subscribe (format nil "~a/color_sensor" name)
                              "turtlesim/Color"
                              #'color-cb))

  (setf *pose-sub* (subscribe (format nil "~a/pose" name)
                              "turtlesim/Pose"
                              #'pose-cb))

  (setf *cmd-vel-pub* (advertise (format nil "~a/cmd_vel" name)
                                "geometry_msgs/Twist"))

  (setf *pen-srv* (concatenate 'string "/" name "/set_pen")))
```

```

(in-package :tut)

(defvar *color-value* (make-fluent :name :color-value) "current color under the turtle")
(defvar *turtle-pose* (make-fluent :name :turtle-pose) "current pose of turtle")

(defvar *color-sub* nil "color ROS subscriber")
(defvar *pose-sub* nil "pose ROS subscriber")
(defvar *cmd-vel-pub* nil "velocity commands ROS publisher")

(defvar *pen-srv* nil "name of ROS service for controlling the pen")

(defun init-ros-turtle (name)
  "Subscribes to topics for a turtle and binds callbacks.
  'name' specifies the name of the turtle."
  (setf *color-sub* (subscribe (format nil "~a/color_sensor" name)
                              "turtlesim/Color"
                              #'color-cb))
  (setf *pose-sub* (subscribe (format nil "~a/pose" name)
                              "turtlesim/Pose"
                              #'pose-cb))

  (setf *cmd-vel-pub* (advertise (format nil "~a/cmd_vel" name)
                                "geometry_msgs/Twist"))

  (setf *pen-srv* (concatenate 'string "/" name "/set_pen")))

```

Define a function to initialize the turtle given by name, e.g. `turtle1`

Subscribe to a topic; use name as the prefix, e.g.. `turtle1/color_sensor`

Message type

Use the `color-cb` callback function to handle the messages received (note the sharp quote)

```

(in-package :tut)

(defvar *color-value* (make-fluent :name :color-value) "current color under the turtle")
(defvar *turtle-pose* (make-fluent :name :turtle-pose) "current pose of turtle")

(defvar *color-sub* nil "color ROS subscriber")
(defvar *pose-sub* nil "pose ROS subscriber")
(defvar *cmd-vel-pub* nil "velocity commands ROS publisher")

(defvar *pen-srv* nil "name of ROS service for controlling the pen")

(defun init-ros-turtle (name)
  "Subscribes to topics for a turtle and binds callbacks.
  'name' specifies the name of the turtle."

  (setf *color-sub* (subscribe (format nil "~a/color_sensor" name)
                              "turtlesim/Color"
                              #'color-cb))
  (setf *pose-sub* (subscribe (format nil "~a/pose" name)
                              "turtlesim/Pose"
                              #'pose-cb))
  (setf *cmd-vel-pub* (advertise (format nil "~a/cmd_vel" name)
                                "geometry_msgs/Twist"))

  (setf *pen-srv* (concatenate 'string "/" name "/set_pen")))

```

Subscribe to a topic; use name as the prefix, e.g.. turtle1/pose

Message type

Use the `pose-cb` callback function to handle the messages received

```

(in-package :tut)

(defvar *color-value* (make-fluent :name :color-value) "current color under the turtle")
(defvar *turtle-pose* (make-fluent :name :turtle-pose) "current pose of turtle")

(defvar *color-sub* nil "color ROS subscriber")
(defvar *pose-sub* nil "pose ROS subscriber")
(defvar *cmd-vel-pub* nil "velocity commands ROS publisher")

(defvar *pen-srv* nil "name of ROS service for controlling the pen")

(defun init-ros-turtle (name)
  "Subscribes to topics for a turtle and binds callbacks.
  'name' specifies the name of the turtle."



  (setf *color-sub* (subscribe (format nil "~a/color_sensor" name)
                              "turtlesim/Color"
                              #'color-cb))

  (setf *pose-sub* (subscribe (format nil "~a/pose" name)
                              "turtlesim/Pose"
                              #'pose-cb))

  (setf *cmd-vel-pub* (advertise (format nil "~a/cmd_vel" name)
                                "geometry_msgs/Twist"))

  (setf *pen-srv* (concatenate 'string "/" name "/set_pen")))

```

 Publish on a topic; use name as the prefix, e.g.. turtle1/cmd_vel
 Message type

```

(in-package :tut)

(defvar *color-value* (make-fluent :name :color-value) "current color under the turtle")
(defvar *turtle-pose* (make-fluent :name :turtle-pose) "current pose of turtle")

(defvar *color-sub* nil "color ROS subscriber")
(defvar *pose-sub* nil "pose ROS subscriber")
(defvar *cmd-vel-pub* nil "velocity commands ROS publisher")

(defvar *pen-srv* nil "name of ROS service for controlling the pen")

(defun init-ros-turtle (name)
  "Subscribes to topics for a turtle and binds callbacks.
  'name' specifies the name of the turtle."

  (setf *color-sub* (subscribe (format nil "~a/color_sensor" name)
                              "turtlesim/Color"
                              #'color-cb))

  (setf *pose-sub* (subscribe (format nil "~a/pose" name)
                              "turtlesim/Pose"
                              #'pose-cb))

  (setf *cmd-vel-pub* (advertise (format nil "~a/cmd_vel" name)
                                "geometry_msgs/Twist"))

  (setf *pen-srv* (concatenate 'string "/" name "/set_pen")))

```

← Create the service name, e.g. /turtle1/set_pen

```

(defun color-cb (msg)
  "Callback for color values. Called by the color topic subscriber."
  (setf (value *color-value*) msg)) ← Callback: copy the color message received to the fluent

(defun pose-cb (msg)
  "Callback for pose values. Called by the pose topic subscriber."
  (setf (value *turtle-pose*) msg)) ← Callback: copy the pose message received to the fluent

(defun send-vel-cmd (lin ang) ← Define a function to combine the linear and angular velocities and publish them
  "Function to send velocity commands."
  (publish *cmd-vel-pub*
    ;; short syntax:
    ;; (make-message "geometry_msgs/Twist" (:x :linear) lin (:z :angular) ang)
    ;; more understandable syntax:
    (make-message "geometry_msgs/Twist"
      :linear (make-msg "geometry_msgs/Vector3" :x lin)
      :angular (make-msg "geometry_msgs/Vector3" :z ang)))) } ← Use the lin and ang parameters
                                                                to create the message

(defun call-set-pen (r g b width off) ← Define a function to set the pen parameters using the SetPen service
  "Function to call the SetPen service."
  (call-service *pen-srv* 'turtlesim-srv:SetPen
    :r r
    :g g
    :b b
    :width width
    :off off))

```

Controlling Turtlesim from CRAM

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
```

Controlling Turtlesim from CRAM

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>
```


Controlling Turtlesim from CRAM

Start a ROS node

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

The name doesn't matter



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

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

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

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

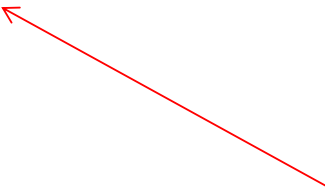
[(ROSLISP TOP) INFO] 1292688669.691: master URI is 127.0.0.1:11311

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

Controlling Turtlesim from CRAM

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

Notice that we haven't launch turtlesim yet but ROS still allows us to subscribe to the turtlesim topics even through turtlesim hasn't yet advertized them

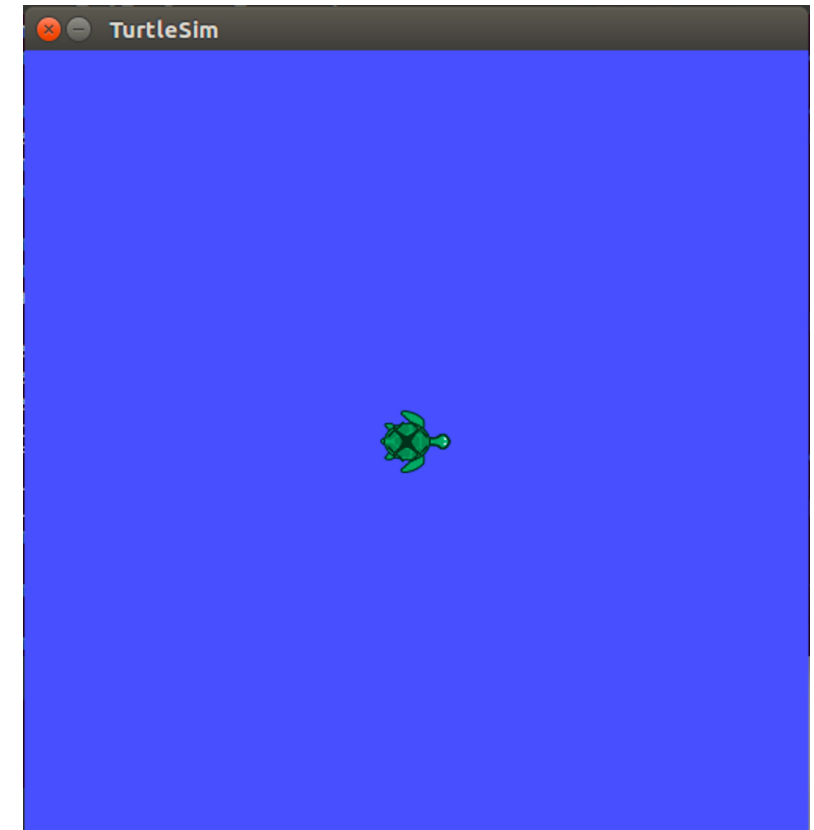
Controlling Turtlesim from CRAM

Now, start turtlesim

Open a new terminal and enter

```
~$ roslaunch turtlesim turtlesim_node
```

This is what you should see



Controlling Turtlesim from CRAM

Let's see if the fluents have the correct values

TUT> (value *turtle-pose*)

[TURTLESIM-MSG:POSE

X:

5.544444561004639d0

Y:

5.544444561004639d0

THETA:

0.0d0

LINEAR_VELOCITY:

0.0d0

ANGULAR_VELOCITY:

0.0d0]

Be careful not to leave out the * characters in the variable name
If you do, you'll get a UNBOUND VARIABLE error
and you'll be transferred into the minibuffer
If that happens, type 1 to abort and return to the main buffer

The turtle is at the centre (5.54, 5.54)

The turtle is facing left

and it is not moving

Controlling Turtlesim from CRAM

Let's see if the fluents have the correct values

TUT> (value *color-value*)

[TURTLESIM-MSG:COLOR

R:

69

G:

86

B:

255]

Be careful not to leave out the * characters in the variable name
If you do, you'll get a UNBOUND VARIABLE error
and you'll be transferred into the minibuffer
If that happens, type 1 to abort and return to the main buffer

Controlling Turtlesim from CRAM

Now, let's move the turtle using turtle_teleop_key

Open a new terminal and enter

```
~$ roslaunch turtlesim turtle_teleop_key
```

Make sure this terminal (the one executing the turtle_teleop_key command) is in focus, (i.e. is selected

Press the Up, Down, Left, or Right arrow key to move the turtle and leave a trail behind it



Controlling Turtlesim from CRAM

Let's see if the fluents have the new values

```
TUT> (value *turtle-pose*)
```

Note: fluents are proxies so, to get the values,
we need to use the value function

```
[TURTLESIM-MSG:POSE
```

X:

```
5.136239528656006d0
```

Y:

```
2.634180784225464d0
```

THETA:

```
0.128000008881836d0
```

LINEAR_VELOCITY:

```
0.0d0
```

ANGULAR_VELOCITY:

```
0.0d0]
```

The turtle is at the new coordinates

The turtle now has an orientation of 0.128 radians

Controlling Turtlesim from CRAM

Aside: coupled with the wait-for function, the fluent allows a process to block until some event occurs

For example, the following blocks until the x coordinate is less than 5.0

```
TUT> (wait-for (< (fl-funcall #'turtlesim-msg:x *turtle-pose*) 5.0))
```

This creates a new fluent that returns the x coordinate of the turtle pose

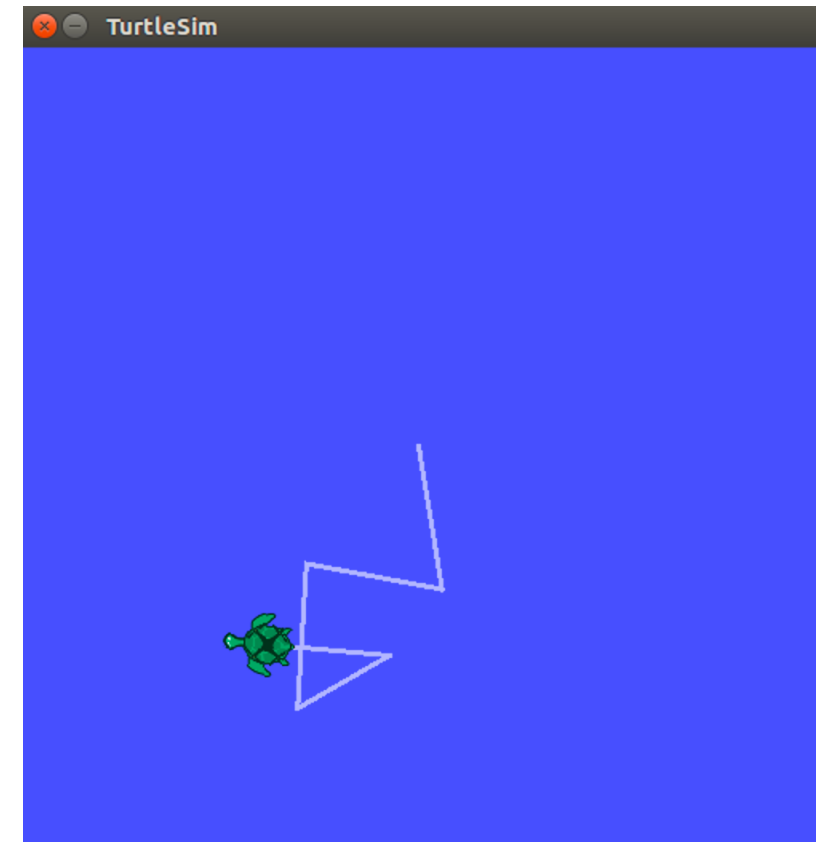
This is also an example of a **fluent network**: the new fluent is created by using the existing fluent

Controlling Turtlesim from CRAM

Aside: execute this expression

```
TUT> (wait-for (< (fl-funcall #'turtlesim-msg:x *turtle-pose*) 5.0))
```

- If the turtle's x position is already smaller than 5, the expression should return T immediately
- If not, the expression should block until you move the turtle far enough to the left using the teleop



Controlling Turtlesim from CRAM

Move the turtle using Lisp

```
TUT> (dotimes (i 10) (send-vel-cmd 1 1) (sleep 1))
```

We defined this in
control-turtlesim.lisp

forward linear velocity of 1
angular velocity of 1

wait 1 second after each velocity command

- The turtle moves in a circle because the angular velocity is non-zero and constant [1]
- with constant forward velocity [1]



Controlling Turtlesim from CRAM

Set the pen colour using Lisp

```
TUT> (call-set-pen 255 0 0 10 0)
```

Red
width of 10
on (the `off` parameter is false, so the pen is on!)

```
TUT> (dotimes (i 10) (send-vel-cmd 1 1) (sleep 1))
```



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