

Code Manipulation through Interactive Markers in a Live Preview

Some context



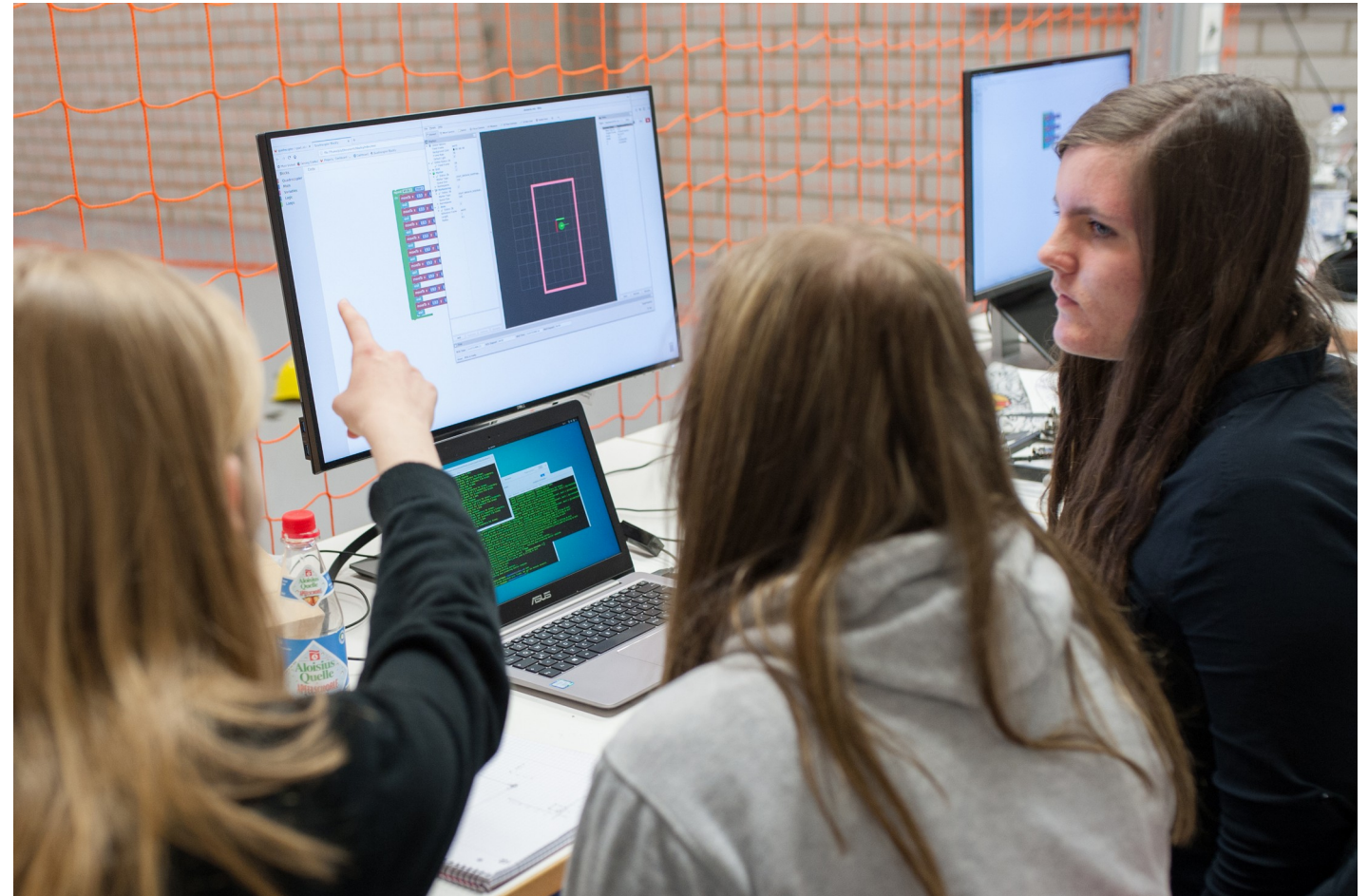
Quadcopter Lab as a demonstrator/student projects

Small Lua based Domain Specific Language (DSL) to define missions/waypoints

Programming novices often hesitant to experiment out of fear to break things
→ Simulation, easier understanding

Defining Quadcopter Missions

- Simplify the language, syntax
- Problems developing a strategy, estimating dimensions etc.
→ live coding / preview



Overview

- 3 main concepts:
 - Live evaluation that previews runtime values and draws visualization
 - Source location tracking that allows to write changes back to the code
 - Mixing visualizations in RViz (integrates into existing visualizations, decoupled)
- Goal:
 - Helping novices exploring, experimenting, understanding code
 - Simplify debugging, remove barrier between development and runtime

Prototype

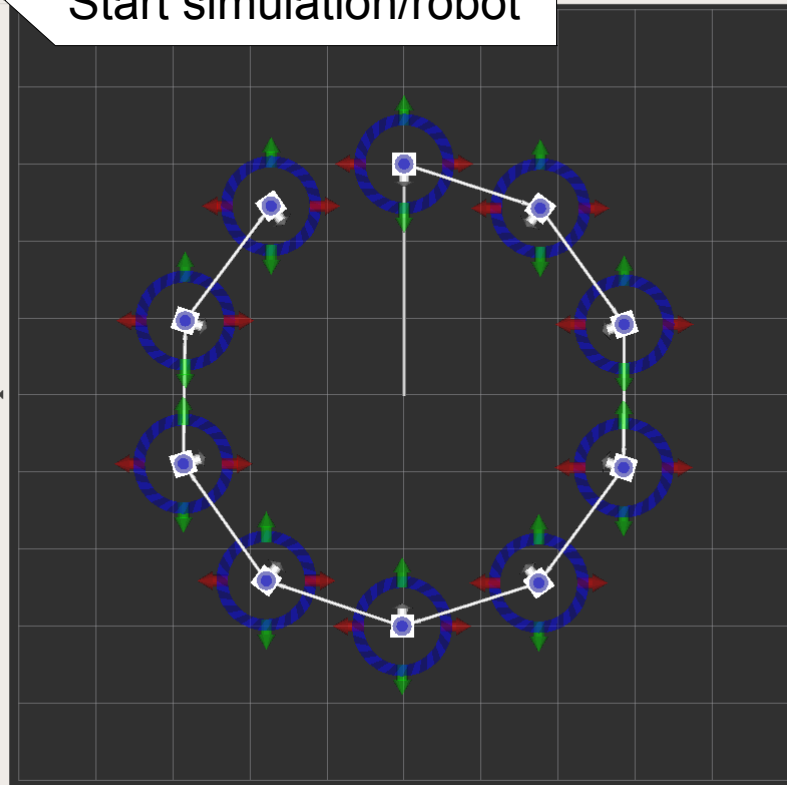
Code editor

```
print('hello world')  
  
for i=0, 9 do  
  local angle = 0.63 * i  
  moveTo(3 * math.sin(angle),  
        3 * math.cos(angle),  
        2, -1.57 - angle)  
  wait()  
end
```

Output console

```
hello world
```

Start simulation/robot



Connected RViz

Live Preview

- The code is executed after each change and markers are placed

The screenshot displays a software interface with two main panes. The left pane, titled 'InteractiveScript', contains a code editor with the following code:

```
print('hello world')  
  
for i=0, 9 do  
  local angle = 0.63 * pi  
  moveTo(3 * math.sin(angle),  
        3 * math.cos(angle),  
        2, -1.57 - angle)  
  wait  
end
```

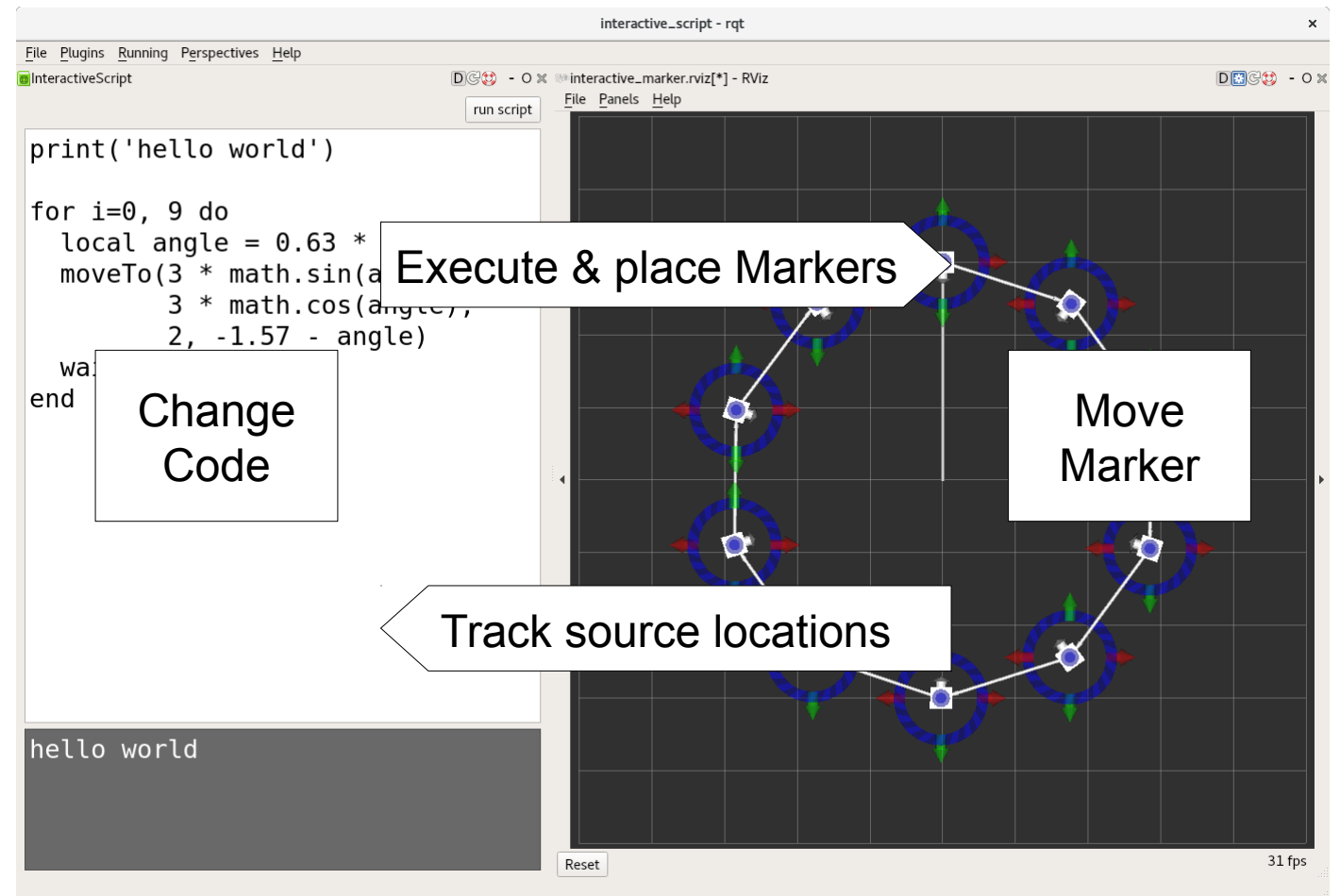
Below the code editor, the output shows 'hello world'. A callout box labeled 'Change Code' points to the code editor. The right pane, titled 'interactive_marker.viz[*] - RViz', shows a 2D plot with a circular path of markers. A callout box labeled 'Execute & place Markers' points to the plot. The plot shows a circular path of markers, with a 'Reset' button at the bottom left and '31 fps' at the bottom right.

Live Preview

- The code is executed after each change and markers are placed
- Allow live modifications of both views

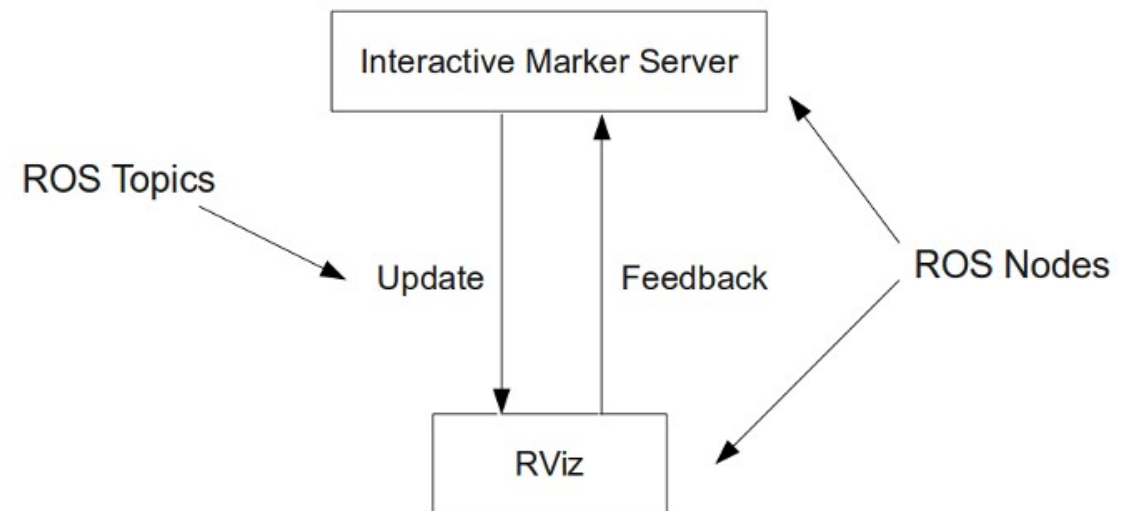
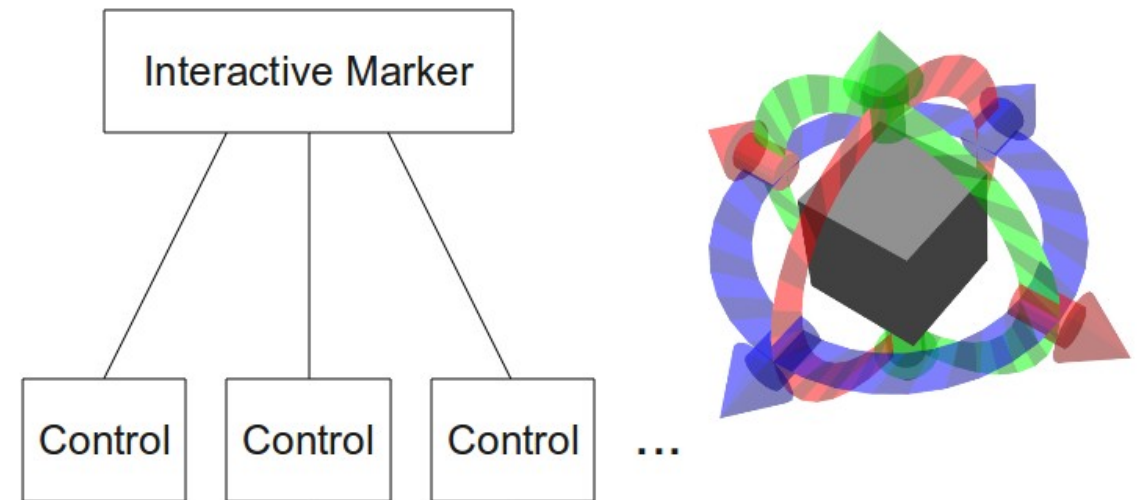
Problem:

imperative language,
changing state



Interactive Markers

- Existing library, implemented in RViz
- Visualization integrates with other ROS nodes
- Can be implemented in other visualization tool (e.g. AR)
- Editor and live preview decoupled



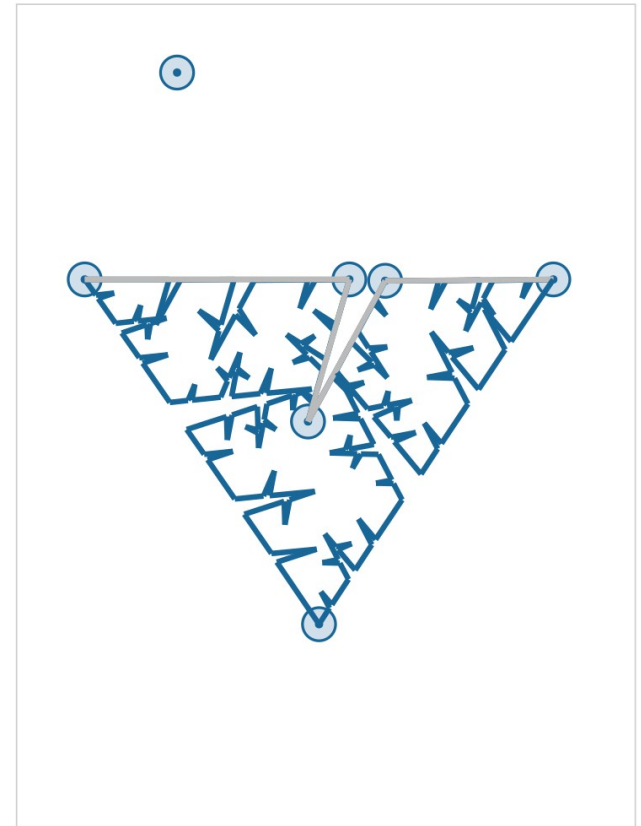
Source Location Tracking (SLT)

- My program returns 4 but I want it to be 6; what do I have to change?
- **Values** can have a source location
- The source location for a literal is its location in the program code.
- The source location is propagated along with the value.

```

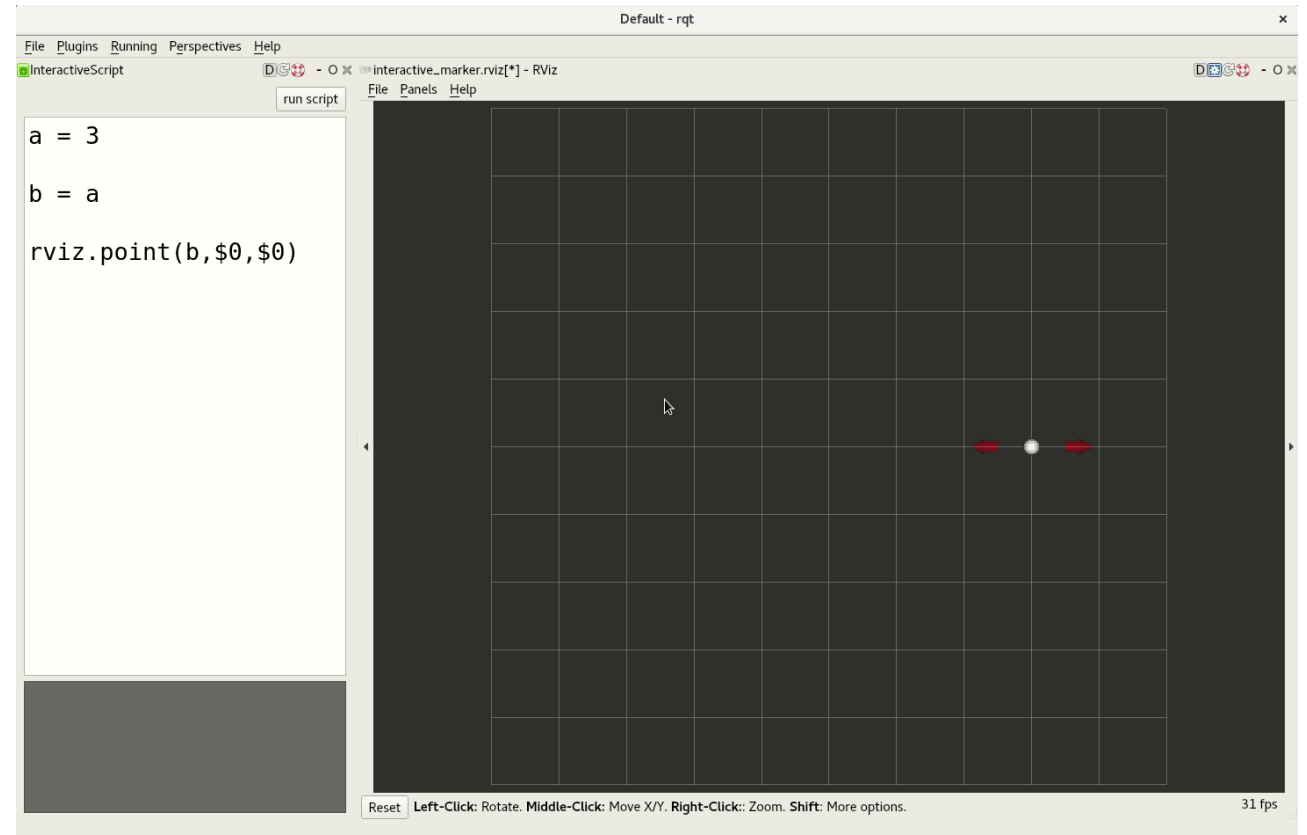
1 (include "prelude.qua");
2 ;
3 (defun (path ps color)
4   (if (< (length ps) 2) ()
5       (begin (line (car ps) (cadr ps) color)
6               (path (cdr ps) color))))
7 ;
8 (defun (fractal a b n ps)
9   (if
10    (<= n 0)
11    (line a b)
12    (begin
13      (define deg (- (angle180 (-- b a))))
14      (define len (lengthv (-- b a)))
15      (define ps' (rotatePath deg ps))
16      (define ps'' (scalePath len ps'))
17      (define ps''' (movePath a ps'''))
18      (map (lambda seg
19            (fractal (car seg) (cdr seg) (-- n 1) ps))
20           (segments ps'''))))
21 ;
22 (defun (koch a b n ps) ()
23   (path ps #bbbbbb)
24   (define ps' (movePath (* -1 (car ps)) ps))
25   (define ps'' (scalePath (/ 1 (lengthv (last ps))) ps'))
26   (define axis (-- (last ps'') (car ps'')))
27   (define ps''' (rotatePath (angle180 axis) ps'''))
28   (fractal a b n ps'''))
29 ;
30 ;
31 (define anchors (list
32 (point 50 200)
33 (point 390 200)
34 (point 220 450)
35 ))
36 ;
37 (define ps (list
38 (car anchors)
39 (point 242 200)
40 (point 212 303)
41 (point 268 201)
42 (cadr anchors)
43 ))
44 (define n (/ (car (point 117 (fix 50))) 50))
45 ;
46 (map (lambda seg (koch (car seg) (cdr seg) n ps))
47      (segments (cons (last anchors) anchors)))
48 ;

```



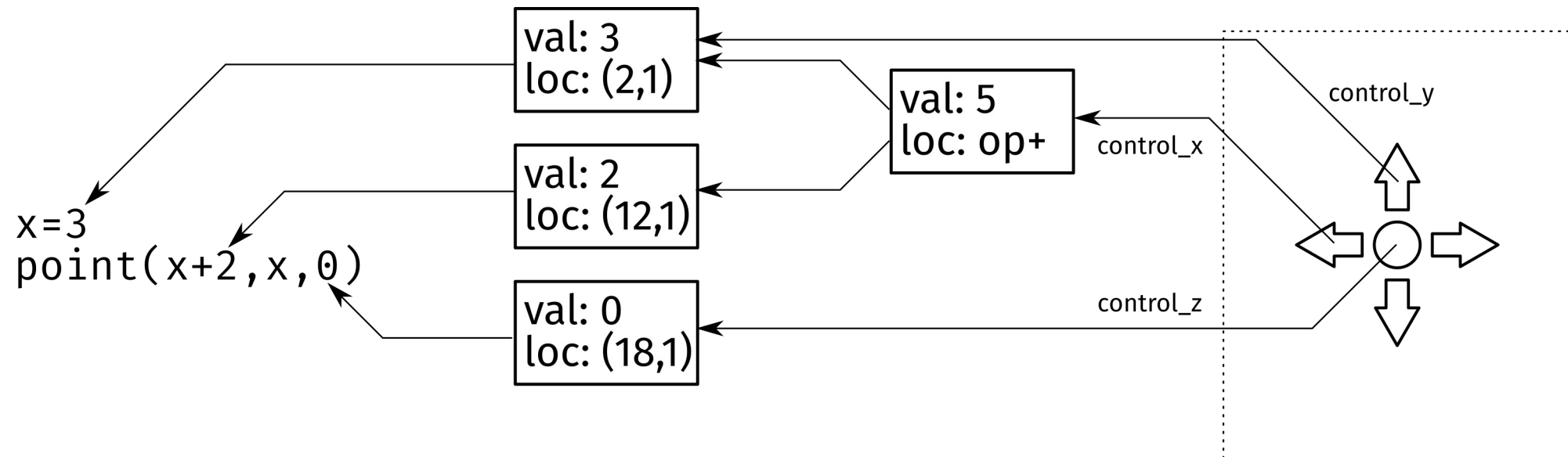
SLT – Simple Example

- Marker control is linked to the **value** that placed the marker
- Assignments etc. just copy the value
- The source is modified at the value's location and reexecuted → the marker is now at the desired position



SLT – Expressions

- Forcing a desired value for an expression yields multiple results
- $5+4=7$: $5 \rightarrow 3 \mid 4 \rightarrow 2 \mid (5 \rightarrow 4.5, 4 \rightarrow 2.5) \mid \dots$
- Left bias implemented: first operand that has a source attached is changed



SLT – Examples, Generalizing effects

- Incomplete heuristic, but it works in many cases
- Nice generalizing effect: Moving a marker keeps the shape elliptical
- Language and Domain agnostic

The screenshot displays a software interface with two main windows. The left window, titled 'InteractiveScript', contains the following code:

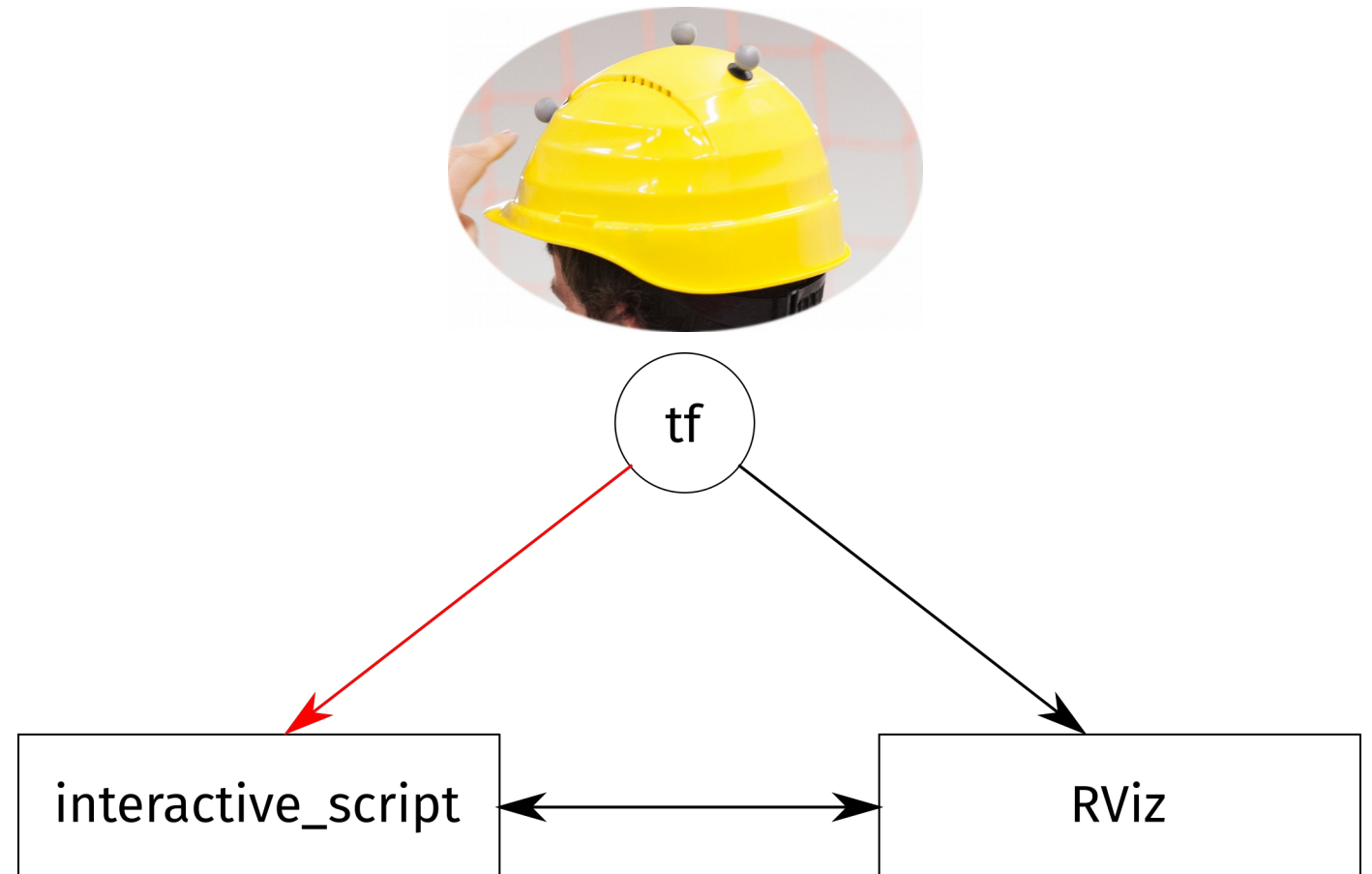
```
print('hello world')

for i=0, 9 do
  local angle = 0.63 * i
  moveTo(3 * math.sin(angle),
        3 * math.cos(angle),
        2, -1.57 - angle)
  wait()
end
```

Below the code, the output 'hello world' is visible. The right window, titled 'interactive_marker.rviz[*] - RViz', shows a 2D plot with a grid. A path of markers is plotted, forming an elliptical shape. Each marker consists of a blue circle with a white square in the center, and red and green arrows pointing outwards. The plot is titled 'Default - rqt' and has a 'Reset' button at the bottom left. The bottom right corner of the plot area shows '31 fps'.

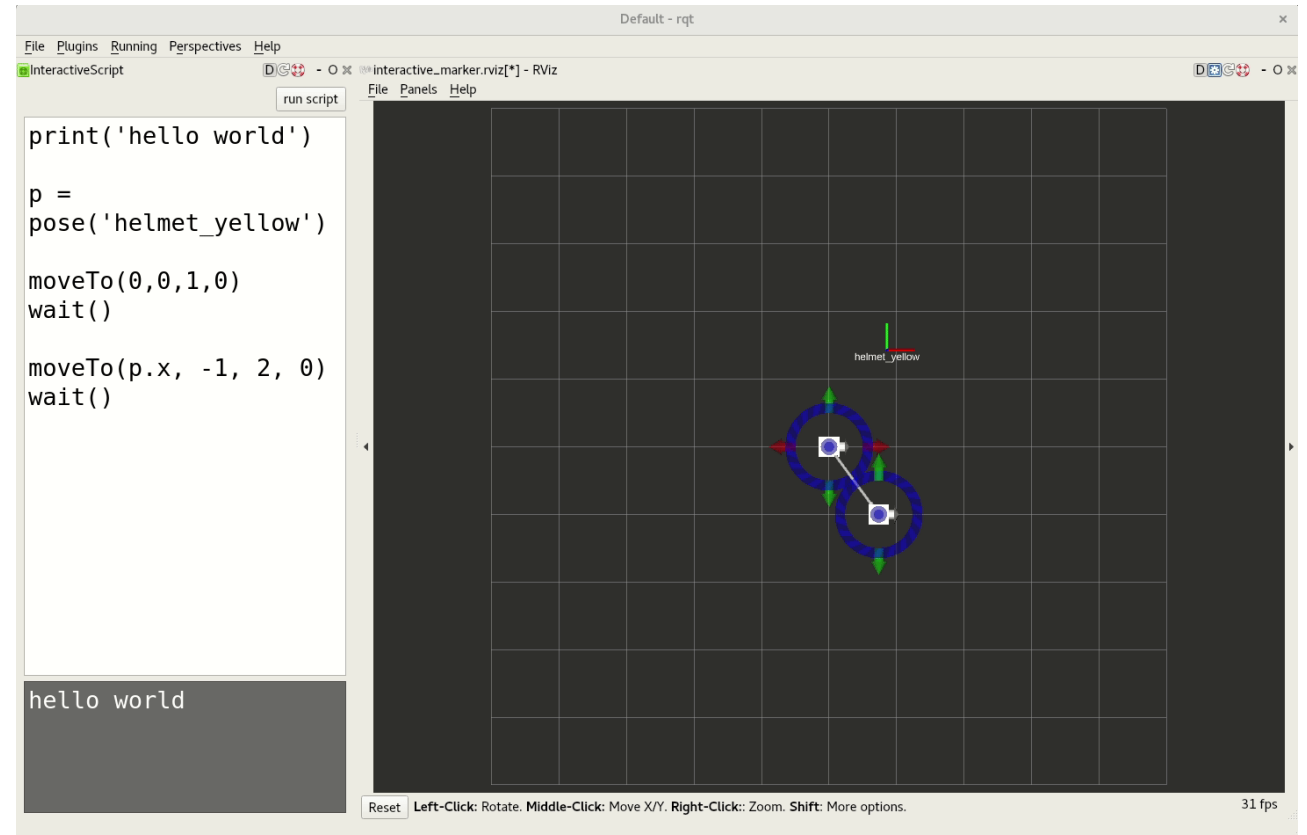
tf Integration

- Interaction with physical objects should be visible in the preview
- Capturing a pose: placing an object and using live evaluation
- Experimentation to explore corner cases



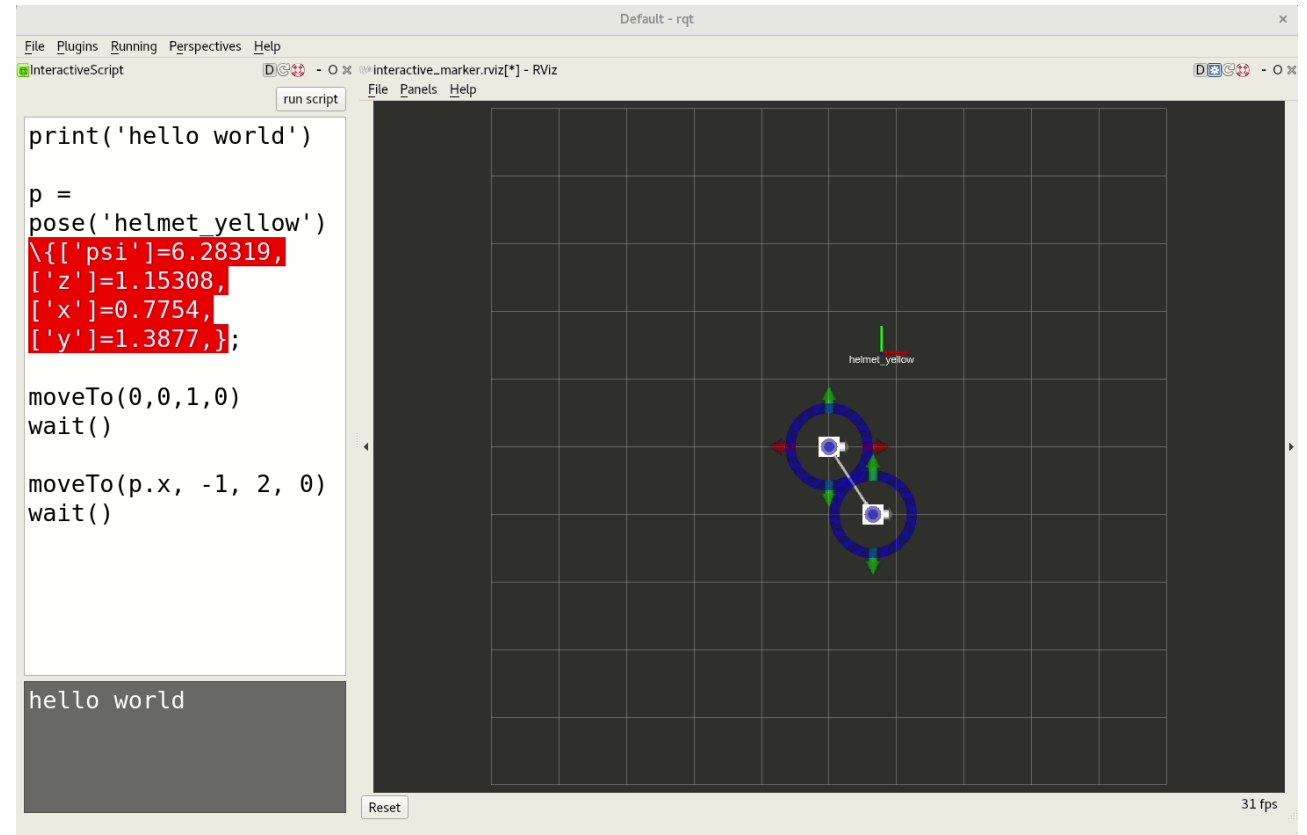
Live Evaluation, \ operator

- Simple script:
fly to x-coordinate of
the yellow helmet
- pose command uses tf
→ preview changes when
helmet is moved
- Show the live coordinates
in the code



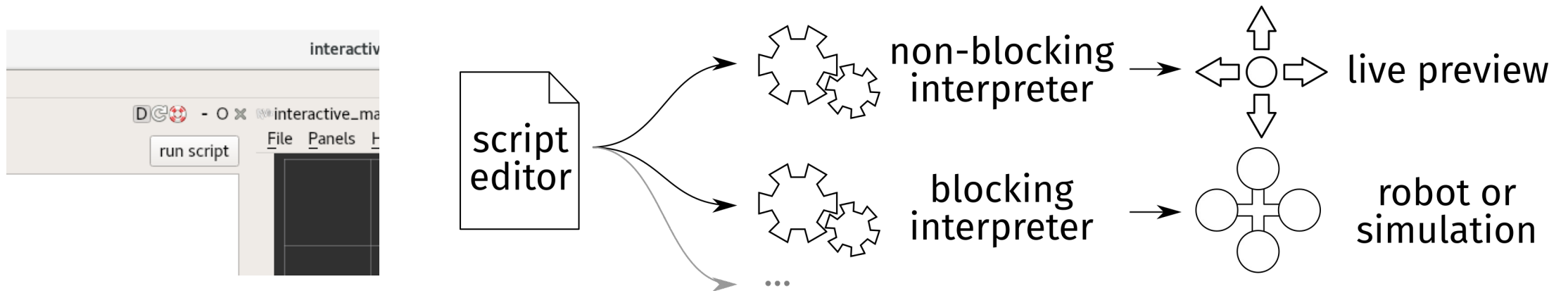
Live Evaluation, \ operator

- Simple script:
fly to x-coordinate of
the yellow helmet
- pose command uses tf
→ preview changes when
helmet is moved
- Show the live coordinates
in the code



Start simulation

- Directly start script on a connected system (simulation/robot)
- Separate interpreter, implementation of functions
- write structure → move markers → see preview → run simulation (→ start robot)



Overview

- 3 main concepts:
 - Live evaluation that previews runtime values and draws visualization
 - Source location tracking that allows to write changes back to the code
 - Mixing visualizations in RViz (integrates into existing visualizations, decoupled)
- Goal:
 - Helping novices exploring, experimenting, understanding code
 - Simplify debugging, remove barrier between development and runtime

Overview

- Prototype implementation:
 - Not yet released
 - Code available at [**github.com/sp-uulm/interactive_script**](https://github.com/sp-uulm/interactive_script)
- Future Work:
 - Evaluate prototype and DSL
 - Integrate block-based language frontend
 - Integrate Augmented Reality visualization