Code Manipulation through Interactive Markers in a Live Preview

Thomas Witte, Matthias Tichy | 31.10.2019
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

Prototype

- Rqt plugin as prototype
- Editor that connects to other components
  - Live visualization → InteractiveMarkers
  - Live evaluation → tf, joy, …
  - Simulation/execution on robot → Trajectory planning
- Graphic primitives → visualization_msgs

- Multiple instances of interpreter for preview and simulation
- Different implementation of statements

Code editor

Output console

Connected RViz

Start simulation/robot

```
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
```
Live Preview

- The code is executed after each change and markers are placed
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.
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
- \(5+4=7\): \(5 \rightarrow 3\) | \(4 \rightarrow 2\) | \((5 \rightarrow 4.5, 4 \rightarrow 2.5)\) | …
- 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
**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

• Future Work:
  • Evaluate prototype and DSL
  • Integrate block-based language frontend
  • Integrate Augmented Reality visualization