A Live Robot Coding Framework based on ry

Software Project Description

Marc Toussaint, Dec 19, 2018

In music, live coding denotes frameworks that allow an artist to code a piece of electronic music on the fly. We want the same for robotics.

The goal of this project is not a software that solves one particular problem. Instead, a live robot coding (LRC) environment should be developed that allows a coding user to script/operate very efficiently and interactively any robot behavior. The user scenario is a challenge scenario: The user is an expert roboticist, has experience with the coding environment, and is located next to a robot and experimental setup. But the user does not know the task to be solved yet. A judge is defining the task. This could be anything, e.g. the robot is to pick up all pieces and check their color at the bottom; or stack them; or open all boxes. The challenge for the user is to solve this task as quickly as possible by directly coding and interacting with the robot, where the result needs to be a script that performs the task autonomously. The LRC framework should supported him as best as possible during this challenge.

A constrained for this project is that the LRC should wrap nothing but the RAI software (see github) which we are already using in our lab and for which there already are python bindings (see rai-python). The focus is not on extending the already existing functionality of RAI. But instead, to make this functionality accessible in live coding.

Another constraint is that the basic LCR user language should be python. Some established abstractions we already use should also be adopted: the script essentially initiates and terminates concurrent activities or sub-scripts, each of which has a status, and includes language constructs to respond to events (whenever /this/ do /that/; wait for /that/; etc).

This project should explore and develop LCR frameworks beyond this. In particular:

- Visualization is a core to efficiently code robots. E.g., when you've interactively started a method to compute a path, you want to look at the result before sending it to the robot. Generally, during live development one often wants to temporarily display things. A static GUI environment is not flexible enough to allow a live coder to flexibly inspect, debug, and test on the fly. So the LCR framework should support the user to not only print (on console) but also display whatever the user wants to display.
- There are two graphical programming schemes that I find interesting: Behavior Trees, and Blockly:
- Behavior Trees are a very interesting way to represent behaviors. In a sense they are an alternative to scripting with standard programming language constructs. This https://behaviortree.github.io/BehaviorTree.CPP/ is a good library that also includes a GUI to design behavior trees. However, I doubt that using behavior trees ONLY is the right LCR framework. Instead, the project should explore whether it is effective if the python LCR

framework allows the user to also design behavior trees for certain subtasks, and integrate these in the developed solution – all on the fly in interaction with the robot.

- Blockly (https://developers.google.com/blockly/, and checkout these examples: https://blockly-games.appspot.com/) allows kids to learn programming. And although all this mouse-clicking is awkward, it is super simple and effective also for concurrent activities (checkout the music example). Something to explore.

Another dimension of the project is proper evaluation of the LCR. The project should setup such challenge tests with both, experienced and less experienced users, and investigate what truly slows them down, and which abstractions the adopt easily.

Finally, the more lines of codes this project generates overall, the worse. Conciseness and minimalism is the goal.