Running Robotito Under SCS FAQ

David Ehrenhaft

University of South Florida

Introduction

This document provides a series of questions and answers about running the Robotito as the robot for SCS models. This is meant as an assistance for working with the system, and not as a comprehensive guide or explanation of underlying processes. Refer to the Installation Guide for instructions on installing all necessary software components, to the Run Guide for instructions on running the entire setup, and the Final Report for details regarding the current state of the extension code and suggested fixes.

Table of Contents

How do I change the starting position of the robot?

The starting position is defined by a task in the experimental XML file. This task will occur each episode, and if you modify it, you must modify the task for EVERY GROUP, for which Multiscale Actor Critic Morris Maze Model has 2. Failure to modify both will result in the robot starting different positions depending on the trial.

How do I change the position of the platform?

This depends on whether you are using the simulated robot or the Robotito. In the simulated robot (VirtualStepRobot), you must refer to the maze file listed in the xml and change it there. For the Robotito robot, the platform position is defined by a task (just like the starting position).

How can I change the robot's speed?

The Robotito's speed is defined by a constant in the "/scs/multiscalemodel/src/edu/usf/ratsim/robot/robotito/Robotito.java" file. You can independently adjust the robot's forward speed and rotation speed. If you do so, you may need to adjust the calibration constants for motion. Do note that the current speeds were thoroughly tested, and that faster speeds will increase the likelihood that the robot will run into a wall or flip over.

Why does the robot seem to turn randomly when its facing a wall?

This is a common issue with multiple causes. The first is that the marker cannot be detected, most likely because it is not underneath the camera, or the marker is blocked. You can check whether the markers are detected in rviz. To open, enter "rosrun rviz rviz" in a terminal, then add the visualization markers and the camera stream. The robot's marker lights up blue when detected.

If you have confirmed that the marker is detected, then the Robotito is either too close to a wall, or you have a series of unlucky random turns that cancel each other out. For more details, refer to the final report in the section titled "Known Issues and Necessary Improvements."

Can other physical robots be used than the Robotito?

At this time, no. I have only created the necessary code for the Robotito robot model. It is possible to design the code necessary for other models of robots using the Robotito.java implementation as an example, as most of the code in SCS is not specific to the Robotito, but the firmware and hardware requirements (i.e. Xbee) may take more time to implement with other types of robots.

How do I add obstacles?

The wall detector automatically adds walls when a new marker appears, even if the markers are added between episodes (though you may have troubles if the wall is too close the robot's starting position). Once the robot completes an episode, it will stop and the SCS simulator will freeze until you place it at the origin and press enter twice, so use this time to place in new walls and markers.

How do I print new Markers?

Markers are designed using the Alvar AR tool. I have left these markers in PDF samples with the robotito firmware files. If you wish to use more than the 11 markers, you will need to run the following command to generate the new marker values. You will also need to modify the wall detector java code in SCS (see below).

When cutting the markers out, leave ample whitespace around the black of the marker. The more whitespace, the more easily the ar detector can find and track them.

The walls are not drawn correctly on the screen!

First, run rviz ("rosrun rviz rviz") after launching the pose_detector.launch to confirm that the marker for the wall is properly detected.

Then confirm that the value of the marker is the value you expect. You can confirm this using "rostopic echo walls" when the pose_detector.launch is running. Markers 1 – 4 are reserved for external walls, and markers 5 – 10 are reserved for obstacles. These can be modified in the wall_detector.java file in SCS.

If the wall is correctly placed, but not the correct size, you will need to go to the wall detector code in SCS and modify the wall constants to suit your needs.

Obstacles aren't represented correctly on the screen!

Refer to the above question first.

Obstacles are current implemented as a pair of parallel walls, and all walls are line segments. If this is unsuitable for your needs, refer to "/scs/multiscalemodel/src/edu/usf/ratsim/robot/robotito/ROSWallDetector.java" to modify the wall detection constants. If you need a non-rectangular shape, you may need to make significant modifications to the wall detection and avoidance system.

What can be used as obstacles?

I have performed some tests using black blocks in the lab. Other shapes will be much more difficult to use, since SCS doesn't have an obstacle-tracking system and the markers must be laid flat and in a constant position. I have only modified the wall detector code to mark obstacles as a collection of walls in a rectangular shape, which should be sufficient for simple tests.

Note that the dimensions of obstacles are NOT detected automatically and must be modified by changing constants in "/scs/multiscalemodel/src/edu/usf/ratsim/robot/robotito/ROSWallDetector.java".

Can I pause during an episode?

No, there is currently no way to completely suspend the operation (either to pause for a moment between cycles or to completely stop the experiment and restart from that point again later). If you need to do something to maintain the robot, do so after it has found the platform (when the main thread is waiting for user input to continue).

However, you can slow the pace down significantly. The robot moves in cycles – one action is taken per cycle. You can adjust the length of time to wait between cycles by adjusting the slider below the world visualization in the SCS java applet. Note that doing this will NOT reduce the robot's motion speed when it moves, but the merely the rate of packets being sent to move the robot.

Why does only one of the cameras work?

As described in the "Suggested Improvements" section of the main report, I have modified the ROS code handling the camera to remove the camera further from the lab's door from the physical setup. This is because the old wall detector and pose detector nodes would fail to compile while combining two cameras. Both cameras can work together but you would need to handle the camera overlaps in the pose and wall detector python code in ~/robotito/ (NOT in SCS).

Where can I adjust the camera settings?

The camera configuration settings can be found in /robotito/config/ (NOT in SCS). The Stringray1's settings are currently set to optimize the AR marker detection but may be too slow for your needs. You can view the camera stream in rviz after running the pose detector code ("roslaunch robotito pose_detector.launch").

To test different configuration values, you can try using Coriander (enter "coriander" in a terminal, install if necessary).

What machines can I run this software with?

The SCS software is designed to run in a Unix environment, and all of the testing I performed was done in Ubuntu.

CMAC1 is a machine in the USF Biorobotics lab that is connected directly to the Stringray cameras. Though you don't necessarily need to use this machine, an overhead camera's video stream is necessary to run the application with a physical robot, so you will need to adjust the "/robotito/launch/pose_detector.launch" file to accommodate the changes. Note that you don't necessarily need to be directly connected to the camera to access a valid video stream, but modifications to the above file would be necessary.

What sensors does the robot use?

The robot is functionally blind. Although the Robotito design contains a dozen IR sensors, none of these are used. Instead, the Global Camera (the Stingray camera(s)) handles all wall and pose detection.

Where are the initial position and the platform?

Currently, these are not marked by any sort of marker. The positions are defined only in code (refer to above questions to change). If you need to identify the exact positions, you should run pose_detector.launch and use the command "rostopic echo /robotito/pose" to see the current position of the robot's marker. Then use the marker to map the point you want to the coordinate grid used by the setup.

How close to the platform/starting position does the robot need to be?

For the starting position, the exact degree of correctness isn't extremely relevant. As long as you are within a few centimeters, it should all be fine.

For the platform, it depends on whether the robot is in simulation or physical. In the simulation, look at the maze file. For the physical robot, it is currently set to the radius of the platform (found in the XML file) + the FOUND_PLAT_DIST, a constant in multiscalemodel/src/edu/usf/ratsim/robot/robotito/Robotito.java, which is 0.1 m by default.

Sometimes the robot stops moving briefly, is there a reason for that?

The robot has no “thinking time” at least from a human’s perception. If the robot doesn’t appear to be moving, and it hasn’t hit the goal, there is probably a missed rotate (as rotates fail more often than forwards).

How can I debug SCS?

You should refer to “scs_platform/src/edu/usf/experiment/utils/Debug.java” to interact with multiple debugging flags for the SCS platform. Setting these to true will add new print statements to the console, helping to debug across multiple scattered parts of the platform.

If you need to debug the Robotito code, a main method also exists within “multiscalemodel/src/edu/usf/ratsim/robot/robotito/Robotito.java”. You must first set the constant Boolean titled “ROBOT_DEBUG” on line 104 to true before attempting to run it. There is also another flag just underneath that can be used to print information about the actions send and waits associated with the estimation methods.

Why are the forward and rotate methods estimation-based?

The only input to the SCS program is through the /robotito/pose and /walls topics, which are both dependent on the marker detector node. This node only updates once every 100 ms or so and even these aren’t necessary up-to-date data, making the updates excessively slow – too slow to use a PID approach with. The estimation approaches are generally very accurate, though sometimes the wheels don’t turn for a given rotate. These errors are controlled, however, since the robot’s current position is determined by the camera, and steps are relatively small. If a cycle completes in under 100 ms, the extra step won’t force the robot to hit a wall.

It would be optimal to use a less noisy approach, but you would need to speed up the marker identification code or replace the markers with another localization system. Also consider that some episodes may take over 1000 steps, so be careful with PID. It will be more accurate, but may take far more time to actually run with than its worth.

How does the coordinate system work?

If you are in the lab, the origin of the universe is between the two stingray cameras. The x-axis is aligned with the door. As you approach the door, the x-value increases. The y-value increases as you approach the CMAC1 machine.

To test these values, you can enter “roslaunch robotito pose_detector.launch” in a terminal and in a new terminal enter “rostopic echo /robotito/pose” and you can see the Coordinate of the robot’s marker. You can lay the marker in desired positions to identify the equivalent place in the simulator’s coordinate scheme.

How do I change parameters of the experiment?

The first thing you’ll want to look at is the experimental XML file. For the Robotito, you’ll want to be using “/scs/multiscalemodel/src/edu/usf/ratsim/model/morris/multiscale/Robotito_Experiment.xml”.

From there, you can change the size of steps, angle to rotate per rotation, the Robot to use, the Universe to use, the Mazefile to use (used for the simulation only), as well as all the place cell parameters (number of layers, cells per layer, size of cells, etc.). This also allows you to define the model to use.

You will also be able to change the number of episodes to perform per trial (40 episodes of training for Learning group as default, 10 episodes of Recall for the Learning and Control groups as default).

When using ROS, you can also modify the initial and platform positions in the task sections (make sure you change these for both the training and recall trials if you want the positions to be constant across the experiment).