User Guide for Swarm Simulator

Rajesh Doddegowda

March 6, 2024

Swarm simulators are sophisticated tools designed to replicate and analyze the collective behavior of multiple autonomous agents, often referred to as a "swarm". These agents, which can represent robots, animals, or any entities capable of collective behavior, operate under a set of rules that govern their interactions. The purpose of a swarm simulator is to observe how individual actions lead to emergent group behavior, allowing researchers and developers to explore various scenarios and applications ranging from robotics to natural phenomena. By utilizing a swarm simulator, one can experiment with algorithms for coordination, collision avoidance, and area coverage, gaining insights into the complexities of decentralized systems and collective intelligence.

The robot simulator features six key areas, referred to as "tabs": Run, Configuration, Robot Properties, Debug, Demo, and Comparison. A thorough guide to these tabs is provided to make the simulator user-friendly and maximize its potential. Each tab has been carefully crafted to fulfill a distinct function, and they are equipped with an array of detailed options to improve both the user's interaction with the simulator and their comprehension of its operations. Let's dive into a straightforward and enjoyable simulation experience.

A User Guide: Run Tab

This Run tab shown in the Figure, 1 is crafted to help you navigate and utilize our simulation tool with ease. Whether you're looking to closely inspect the details by zooming in, add new elements, or control the flow of the simulation, our intuitive interface is designed to provide a seamless experience. You'll learn how to adjust your view, introduce and remove robots, and manage the simulation steps and progression. The guide lays out each feature clearly, ensuring you can fully engage with and understand the dynamics of our algorithms.

Zooming and Panning:

- If you need to zoom in for a closer look, scroll up with your mouse wheel. To zoom out, scroll down.
- To move the view of the canvas, click the right mouse button and drag in any direction you wish to pan.
- To reset the view of the canvas to its default setting, simply click the **Fit Screen** button.
- The canvas features an interactive annotation tool; when you hover the mouse over a robot, it conveniently displays the robot's X and Y coordinates.

Adding and Removing Robots:

- To add a robot, simply click anywhere on the canvas with the left mouse button.
- To remove a robot that you have added, place the cursor over the robot and double-click the right mouse button. Note that you cannot delete the robots that are part of the default simulation setup; this only works for robots you have added.

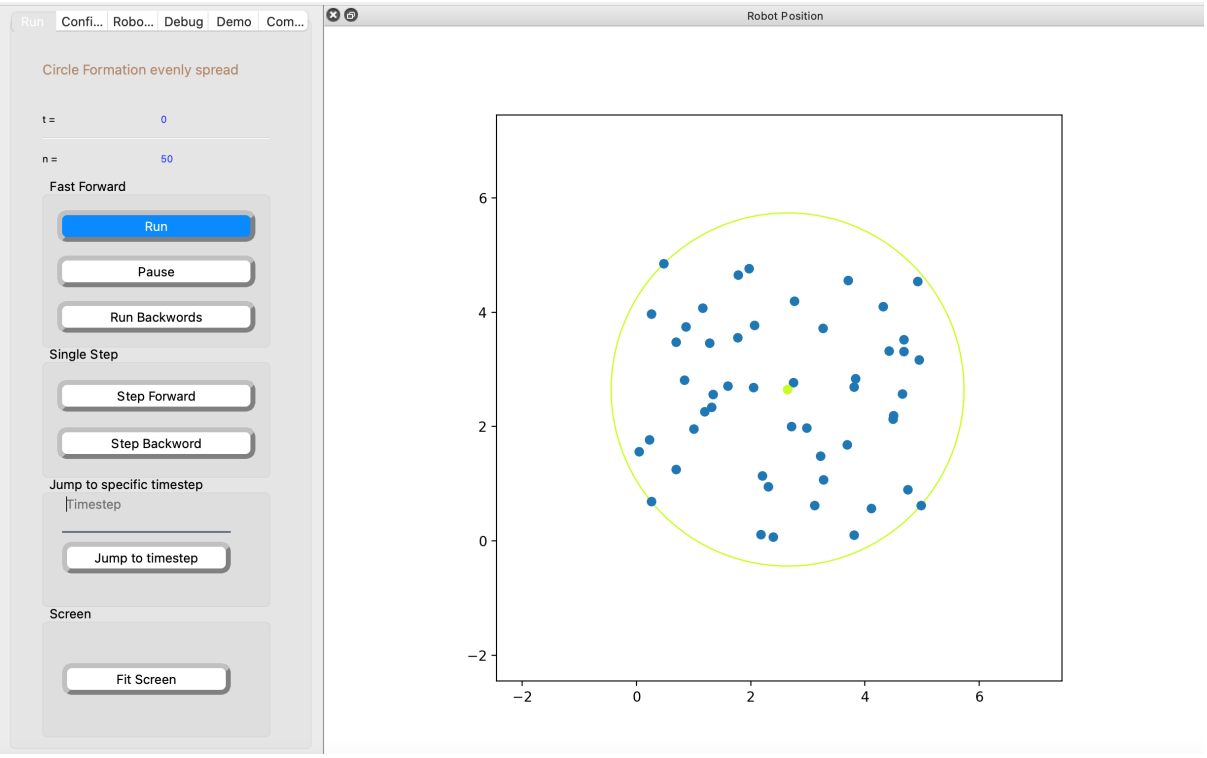

Figure 1: Run Tab in the Swarm Simulator

Running the Simulation:

- The **Run** tab displays the title of the algorithm being used, followed by t which represents the current timestep, and n which indicates the number of robots.
- To start the algorithm, click the **Run** button for forward execution or **Run Backwards** for reverse.
- To halt the algorithm at any point, press the **Pause** button.
- For step-by-step observation, use **Step Forward** to advance by one timestep, or **Step Backward** to go back one timestep.
- To move to a specific point in the simulation, enter the desired timestep number (like 8 or 10) in the *Jump to specific timestep* field and click **Jump to timestep**.

A User Guide: Configuration Tab

The Configuration tab as shown in Figure, 2 is a crucial component of the robot simulator that provides users with the ability to tailor the simulation space to their specific needs. This tab houses essential tools that facilitate the customization of the simulation environment, ensuring a more efficient and precise setup. It includes a user-friendly grid activation feature for precise placement of robots, axis scaling for adjusting the view, and import/export options for managing robot configurations. Whether it's about positioning robots accurately, scaling the canvas to suit the simulation's scope, or saving and loading configurations, the Configuration tab streamlines these processes with its intuitive design.

The Configuration tab in the robot simulator includes several tools for customizing the simulation:

- **Grid On/Off:** A toggle switch that activates a grid overlay on the canvas. With the grid enabled, robots can be positioned precisely by clicking on integer coordinates.
- **Axis Scaling:** This feature allows for the manual scaling of the canvas axes. Users can set the minimum and maximum values up to 10 and apply changes with the **Adjust** button.
- **Import/Export Configuration:**

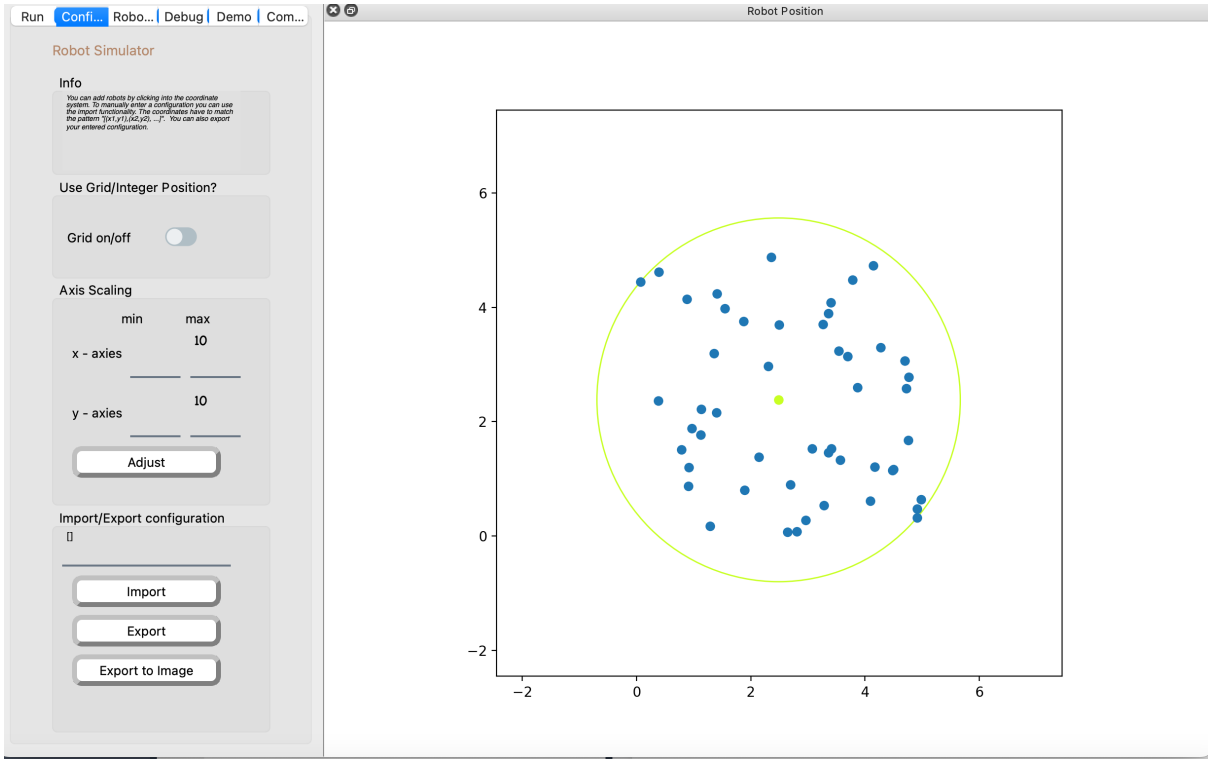

Figure 2: Configuration Tab in the Swarm Simulator

- The **Import** button is used to load a predefined robot position configuration into the simulation in current timestep.
- The **Export** button displays a list of the current robot positions within the simulation in current timestep.
- **Export to Image** saves a snapshot of the current canvas in various file formats such as PNG, JPG, JPEG, or PDF.

These options enhance user interaction with the simulation, offering precise control over the simulation environment and robot configurations.

A User Guide: Robot Properties Tab

The Robot Properties tab is an integral feature of the robot simulator, providing users with detailed control and interaction with individual robot parameters. Activated by a simple toggle, this interactive tab is divided into two key areas: Position Tracking and Selection, and Detailed Robot Properties.

Position Tracking and Selection

Users have the ability to monitor and record the positions of select robots within the simulation. This functionality is critical for tracking robot movement and understanding their behavior within the swarm. To add a robot to the list, one simply needs to hover the cursor over the robot and right-click, prompting the simulator to log its current coordinates.

Detailed Robot Properties

The tab also offers a detailed breakdown of each selected robot's properties, which include:

- **Robot Position:** The current coordinates of the robot within the simulation space.
- **Next Position:** The predicted subsequent location of the robot, determined by the simulation's rules.

- **Left Neighbor:** The position of the closest robot to the left, indicating the robot's spatial context within the group.
- **Right Neighbor:** This property identifies the nearest robot on the right.
- **Left Most Neighbor:** The furthest robot on the left within the robot's range.
- **Right Most Neighbor:** The furthest robot on the right, similarly to the left most neighbor.

These details are vital for users to perform focused observations and to tweak the simulation for desired outcomes.

Focused Debugging

A notable feature of the Robot Properties tab is how it seamlessly integrates with the simulator's debugging functions. By activating this tab and selecting robots, select the debug in the debug tab will display only the selected robots plots. This allows for an in-depth and distraction-free debugging experience, enhancing the overall utility and analysis within the simulation as shown in the Figure, 3.

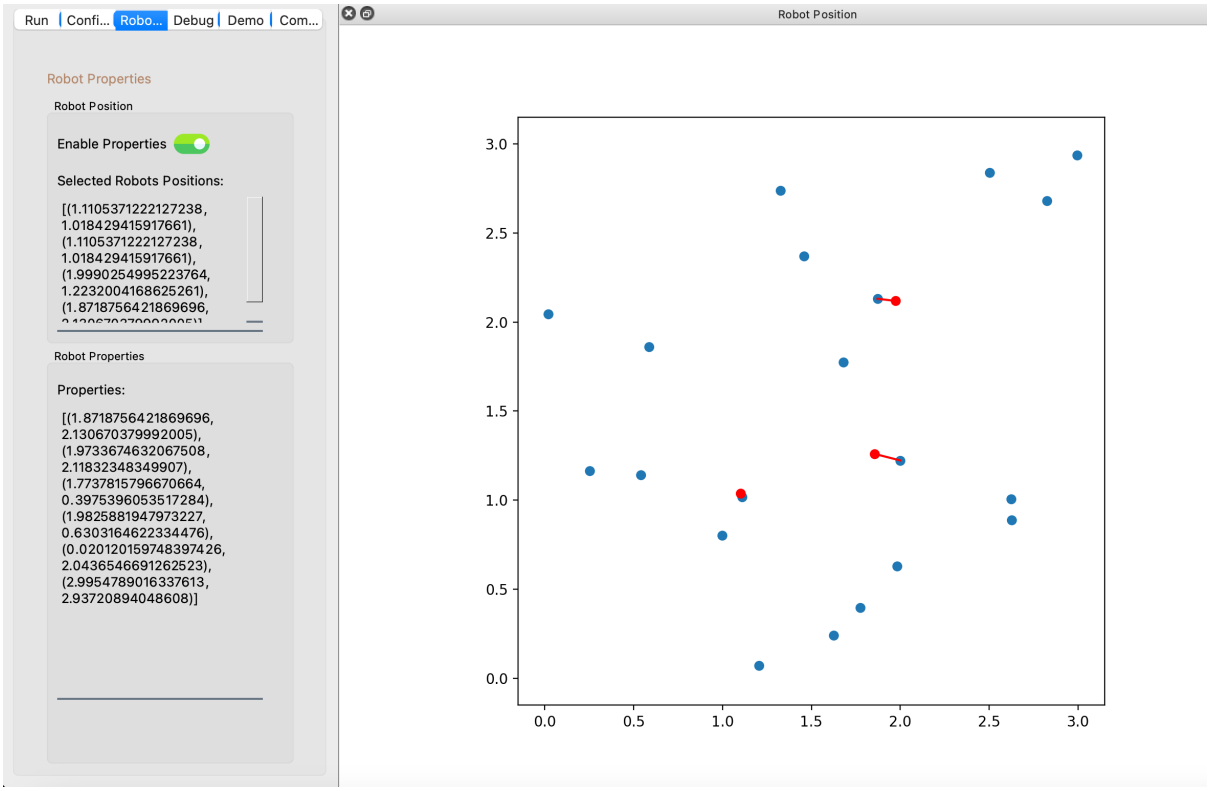

Figure 3: Robot Properties Tab in the Swarm Simulator

With its user-friendly interface and robust set of features, the Robot Properties tab enriches the user experience by offering detailed control over the simulation. It is an invaluable tool for a wide array of uses, from educational demonstrations to complex robotics research.

A User Guide: Debug Tab

The Debug tab serves as a versatile and dynamic tool within the robot simulator, enabling users to visualize debugging plots that are based on the positions of the robots. This functionality is distinct from the `get_debug()` method typically used within specific algorithms. While the `get_debug()` method is limited to providing debug information for a particular algorithm, the Debug tab's plots are used in any algorithms to simulate and can be utilized across various algorithmic contexts.

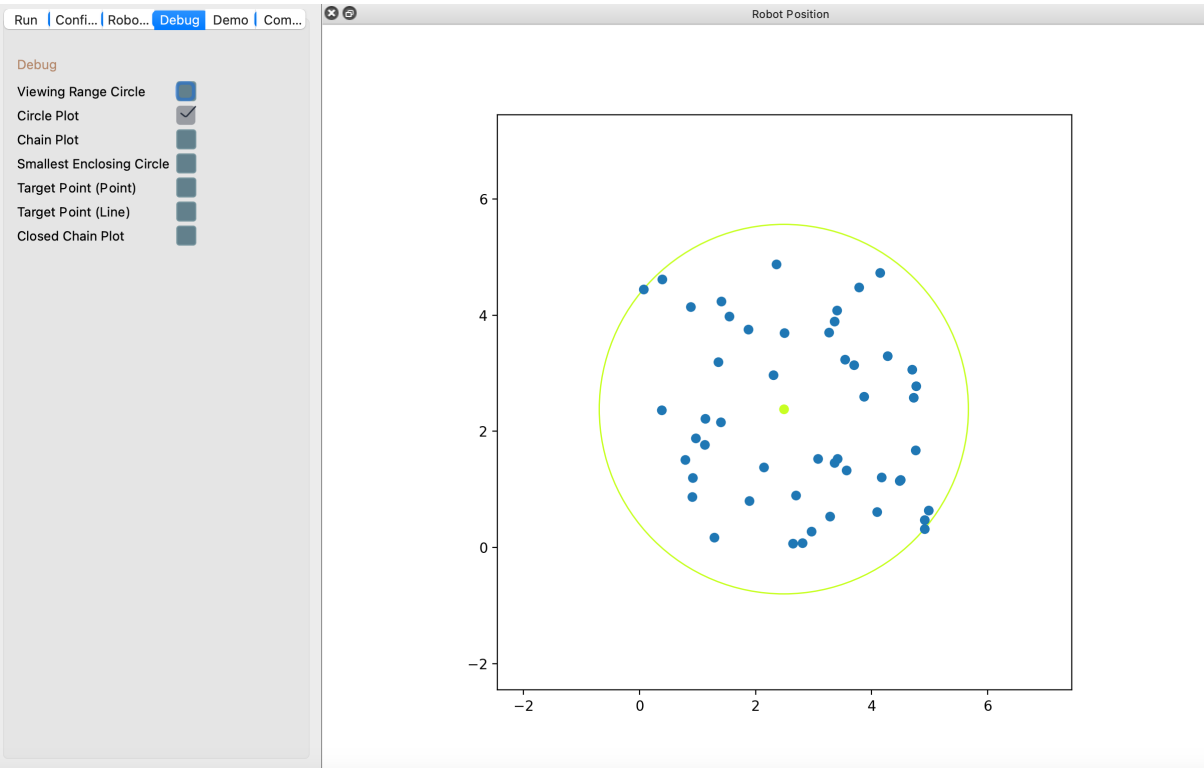

Figure 4: Debug Tab in the Swarm Simulator

To use the Debug tab, one simply needs to check or uncheck the corresponding checkboxes to select or deselect the desired plots. This intuitive interface allows for seamless toggling between different visual debugging aids, making it straightforward for users to adapt the display to their current needs.

One of the key features of the Debug tab is the ability to activate multiple plots simultaneously. This multi-plot capacity is particularly beneficial when users wish to compare and contrast different aspects of robot behavior or when they need to observe the interaction effects between different variables or conditions within the simulation.

Furthermore, using the Robot Properties tab, the Debug tab can be utilized to generate plots for specifically selected robots. This selective plotting is invaluable for in-depth analysis, as it allows users to focus on the behavior of individual robots or a subset of the swarm without the potential clutter or distraction of the full group's data. By isolating the debug plots to selected robots, users can conduct meticulous observations and gain insights into the intricate dynamics that govern robot behavior.

To enhance the functionality of debug plots within the simulation environment, one can integrate a predefined template into the `get_checkbox()` function in the `swarm_interface.py` script as shown in the Figure, 5. This integration will facilitate the addition of a new checkbox in the simulator's debug panel. The inclusion of this checkbox allows for the generation of customized debug plots, tailored to the user's specific requirements for analysis and troubleshooting.

Ensure that the variable i in the template 5 is replaced with the appropriate integer value corresponding to the desired position in the grid layout.

Continuing from the previous step, assign a unique name to each checkbox by adding entries to the label list within the `get_checkbox()` method. This labeling allows for easy identification and reference of each checkbox in the method.

Changes have been implemented in the **debug.py** file, specifically within the **set_debug_plot()** and **get_debug()** methods. These updates enable the addition of debug plots that utilize the positions of the robots, with the modifications visible in the same file.

A User Guide: Demo Tab

The Demo tab in the robot simulator is an interactive component designed for executing demonstrations. This tab is activated by toggling the 'Enable Demo' switch to the on position.

```
self.checkboxdebug2 = QtWidgets.QCheckBox("", self.layoutWidget5)
self.checkboxdebug2.setStyleSheet("QCheckBox::indicator{\n"
                                  "    width:20px;\n"
                                  "    height:20px;\n"
                                  "}\n"
                                  "QCheckBox::hover{\n"
                                  "    color:#B31031;\n"
                                  "}\n"
                                  "QCheckBox::indicator:unchecked{\n"
                                  "    image: url(:/Icons/icon/checkbox.png);\n"
                                  "}\n"
                                  "QCheckBox::indicator:unchecked:hover, QCheckBox::indicator:unchecked:pressed{\n"
                                  "    image: url(:/Icons/icon/square.png);\n"
                                  "}\n"
                                  "QCheckBox::indicator:checked{\n"
                                  "    image: url(:/Icons/icon/check.png);\n"
                                  "}\n"
                                  "")
self.gridLayout_2.addWidget(self.checkboxdebug2, i, 1, 1, 1)
label_list = ['Target Point (Line)', 'Closed Chain Plot']
```

Figure 5: Template to add checkbox

Run All Button

At the heart of the Demo tab is the **Run All** button. Engaging this button selects all algorithms in the list and commences their execution in sequence. This feature is pivotal for showcasing the simulator's range, allowing users to observe a variety of algorithm behaviors in a streamlined manner.

Run Button

Complementing the **RunAll** feature, the **Run** button is tailored for executing a single, selected algorithm. This provides a targeted approach, enabling users to focus their demonstration on a particular algorithm of interest.

Integration with Debug Tab

Accessibility to the Debug tab during a demo allows for on-the-fly plot generation, matching the algorithm in execution. This grants users insight into the algorithm's operation at each step.

Step Controls

The **Step Bwd** (Step Backward) and **Step Fwd** (Step Forward) buttons offer precise control over the demonstration's pacing, allowing users to navigate through the algorithm's execution one step at a time.

Algorithm Navigation

The **Prev** (Previous) and **Next** buttons on the Demo tab enable swift navigation between algorithms, facilitating a smooth transition in multi-algorithm demonstrations.

The Demo tab is a crucial aspect of the robot simulator, providing an intuitive interface for users to demonstrate and scrutinize the intricate functionalities of various algorithms. It is an indispensable tool for education, development, and research in robotic simulations.

A User Guide: Comparison Tab

To begin the comparison of algorithms, activate the comparison feature by enabling the "Enable Compare" switch located at the top of the Compare tab.

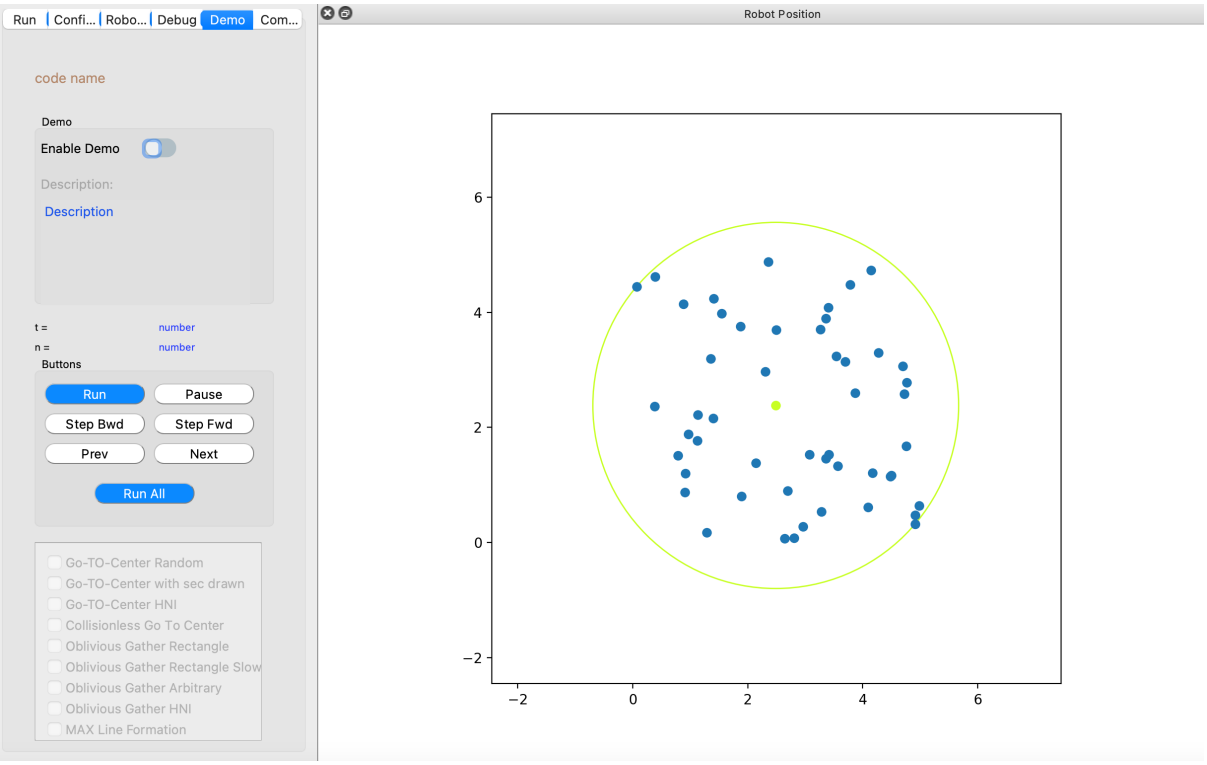

Figure 6: Demo Tab in the Swarm Simulator

Algorithm Selection

Navigate through the list of algorithms in the scrollable area. Click the checkbox next to an algorithm's name to select it for comparison. Note that only one algorithm can be selected at any given time to ensure a focused comparison with the primary visualizer.

Execution Controls

Control the execution of the selected algorithm using the "Run" and "Backward" buttons. These controls allow you to start the simulation and, if necessary, reverse it to analyze the algorithm's behavior over time or under different conditions.

Comparative Analysis

The Secondary Visualizer presents the real-time simulation of the selected algorithm, enabling a direct comparison with the primary visualizer. Analyze the efficiency, execution time, pattern formation, and resource utilization of the algorithm during the simulation.

Key Observables

During the simulation, concentrate on several performance indicators:

- **Efficiency:** Compare how the algorithm's solution speed matches up against the primary algorithm.
- **Execution Time:** Note the execution time and compare it with that of the primary algorithm.
- **Pattern Formation:** Observe the patterns formed by the algorithm and how they differ from those in the primary visualizer.
- **Resource Utilization:** Evaluate the algorithm's resource usage in comparison to the primary algorithm.

In-depth Analysis

For a thorough analysis, adjust the time (t) and number (n) controls to see how the algorithm's performance scales with various input sizes or conditions.

Finalizing Comparison

Once the comparison is complete, deselect the current algorithm or choose another for comparison by toggling the corresponding checkbox.

Returning to Standard View

After concluding your comparisons, disable the "Enable Compare" switch to return to the standard view of the simulation environment.

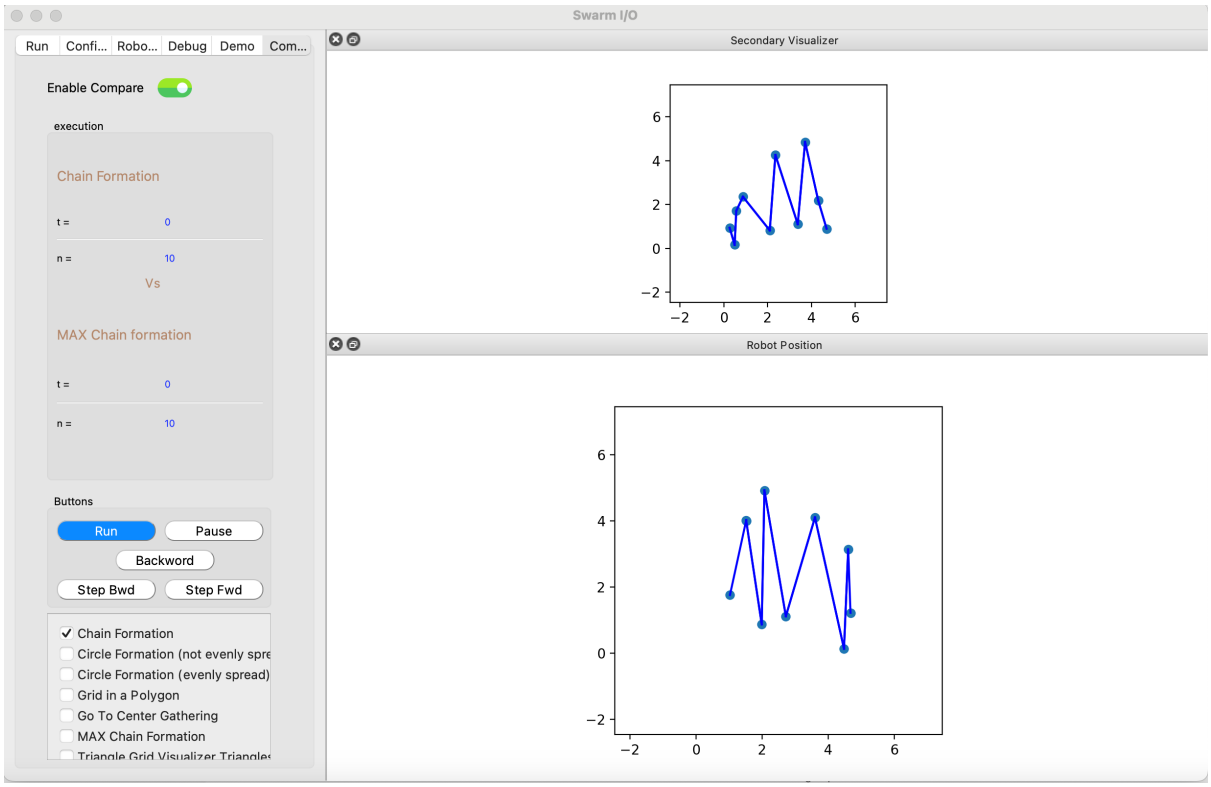

Figure 7: Compare Tab in the Swarm Simulator

User Manual for Manual Execution Template

The manual execution template is designed for the customization and simulation of swarm robotics algorithms, enabling users to define robot model parameters, initial coordinates, robot behaviors, and visualize the swarm's execution for debugging and optimization purposes.

Setting up the Robot Model

At the beginning of the script, instantiate the `RobotModel` class to define the basic properties of your robot model such as `viewing_range`, `dimensions`, and `time_model`. To change the default parameters, provide values to the respective arguments.

The `Robotmodel` class allows for the definition of the robot swarm parameters. These can be customized or set to `None` for defaults:

```
robotmodel_to_update = Robotmodel(
```

```
    viewing_range=None,
    dimensions=None,
    viewing_range_type=None,
    time_model=None,
    grid=None,
    debug=None,
    luminous=None,
    initial_light=None,
    limit_circle_size=None,
)
```

Defining Swarm Coordinates

Use the `rand_coordinates` and `rectangle_coordinates` variables to generate starting positions for the robots. Adjust parameters to set the swarm size and shape according to your simulation needs.

Initial robot positions are set using swarm coordinates functions for generating random coordinates or predefined shapes given below:

```
rand_coordinates = util.SwarmCoordinates.random_swarm_not_connected(3,3,20)
rectangle_coordinates = util.SwarmCoordinates.rectangle_swarm_empty_middle(10, 10)
```

Initializing the Swarm

Instantiate the `Swarm` class with important parameters:

- `robot_coordinates`: Defines initial positions of the robots.
- `RobotClass`: Specifies the class of the robot, determining each robot's behavior in the swarm.
- `rm`: Passes the robot model defined earlier to set properties of each robot in the swarm.

Parameters such as `robot_lights`, `optimize`, `abort_when_execution_finished`, and `max_iterations` control the lighting, optimization, execution abortion, and the maximum number of iterations, respectively.

The `Swarm` class is instantiated with coordinates, robot behavior classes, and other parameters like robot lights and optimization flags given below:

```
s = Swarm(
    robot_coordinates=rand_coordinates,
    RobotClass=MoveNorthRobot,
    rm=robotmodel_to_update,
    robot_lights=None,
    optimize=True,
    abort_when_execution_finished=True,
    max_iterations=np.inf
)
```

Creating a Script for Visualization

The `script_dict` is a Python dictionary defining specific debugging levels at different time steps, allowing granular control over the debugging output. The `ScriptedVisualizer` class takes the `script_dict` and the `Swarm` instance to create a visual simulation. Adjust parameters like `time_offset`, `debug_level`, `debug_ids`, `dynamic`, `xlimit`, and `ylimit` to configure visualization settings.

A script is used with the `ScriptedVisualizer` to control the debugging levels for visualization:

```
script_dict=dict()
script_dict[0] = [([1,2], 2), ([2], 3)]
script_dict[1] = [([1,2], 0), ([2], 3)]
script_dict[3] = [([1,2], 0), ([2], 3)]

vis = ScriptedVisualizer(
```

```
    script_dict=script_dict,
    swarm=s,
    time_offset=100,
    debug_level=0,
    debug_ids=None,
    dynamic=False,
    xlimit=(0, 10),
    ylimit=(0, 5)
)
```

Running the Simulation

To run the simulation, execute the script in an environment where PyQt5 is installed. Save the code in a file named `manual_execution.py` within the `/demo/` directory, adjust parameters as desired, and run the script using a Python interpreter.

The simulation is run using the `RobotSimulator` class which takes the visualizer as a parameter:

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    form = RobotSimulator(visualizer=vis)
    form.show()
    sys.exit(app.exec_())
```

Customization and Execution Tips

Customize the template by adjusting `viewing_range`, `dimensions`, and selecting an appropriate `RobotClass`. Control the plot with `debug_level` and set visualizer bounds with `xlimit` and `ylimit`.

Conclusion

This template serves as a tool for simulating and visualizing swarm robotics algorithms. Customize parameters, present in the template as `manual_execution_template.py` in the `/demo/` directory, adjust parameters as needed, and execute to see the simulation.

Robotclass Example to Initializing the Swarm (Uniform Circular Formation)

The `UniformCircleFormationRobot` class extends the capabilities of a base `Robot` class to enable a swarm of robots to achieve and maintain a uniform circular formation. This guide outlines the key components and functionalities of the `UniformCircleFormationRobot` class, facilitating its integration and utilization in simulations using the manual execution template.

Key Methods

- `get_default_robot_model()`: Initializes the robot model with an infinite viewing range, ensuring all robots are considered neighbors.
- `look_compute()`: Calculates the robot's next position by moving towards a desired position on the circle for uniform distribution.
- `compute_desired_angle()`: Determines the angle for the robot's position relative to the swarm's center to maintain uniform circular formation.
- `angle_from_center()`: Computes the angle of the robot's position relative to the circle's center.
- `get_global_sec()`: Retrieves or calculates the smallest enclosing circle (SEC) for the swarm, defining the formation's center and radius.
- `get_projected_position_on_global_sec()`: Projects the robot's position onto the global SEC, guiding it towards the circle's perimeter.

Debugging and Visualization

The class includes `get_debug_plot` methods for debugging and visualization, offering various insights into the SEC, movement decisions, and projected positions of the robots at different debug levels.

Implementation

To employ this class in a simulation, instantiate the `Swarm` class with `UniformCircleFormationRobot` as the `RobotClass`. This configuration sets up a sophisticated simulation environment where robots autonomously navigate to form a perfect circle, showcasing complex coordinated behaviors in robotic swarms.