Final Report: RoboViz Capstone Project

Boyd Kane
KNXBOY001
KNXBOY001@myuct.ac.za

Imaad Ghoor
GHRIMA002
GHRIMA002@myuct.ac.za

Jesse Sarembock
SRMJES001
SRMJES001@myuct.ac.za

The RoboViz Project extends an existing open source genetic evolution platform (RoboGen) to permit the visualisation of multiple robots, called a swarm. The RoboViz Project accepts multiple parameter to fine-tune the simulation of the swarm, and adheres to OOP best practices.

Contents

1. Introduction

This project - RoboViz - involves extending an existing visualiser [1] to enable the visualisation of multiple of multiple robots simultaneously. Robogen allows researchers to define a robot structure and then make use of genetic algorithms (paired with a fitness function) to evolve robots that that gradually perform better (as a measured by the relevant fitness function) as more generations of robots are simulated.

After a set number of generations, the final robot can also be visualised, although currently the software only allows for the visualisation of a single robot. The software also generates STL files which describe the 3D body parts of the robot, such that a 3D printer can take those files and 3D print the body components of the evolved robot. INO files are also generated, which can be loaded onto the Arduino platform and define the robot's 'brain' as a software defined artificial neural network which was evolved by the genetic algorithm.

This project involves modifying the source code of the RoboGen software and proving that the modifications are efficient enough for at least 3 (but preferably more) robots to be simulated at once.

An agile software development approach was taken to develop this project. The project team has worked over WhatsApp and MS Teams, informing each other on their work and designating tasks from there. Time was spent on creating functional code to implement a swarm of robots and testing that code before adding more functionality. The team has had frequent meetings with project stakeholders, receiving feedback directly from the stakeholders while developing the project. Throughout development, there have been a significant number of changes needed as development progressed and these were added to the project plan over time.

A vertical prototype was chosen for this project since it focuses on implementing a specific feature - swarm of robots in the visualiser - this was the most appropriate prototype as it tests key components during early stages of the project to check key functions.

The Doxygen documentation tool [2] was also added to the project, in order to automatically generate documentation from the source code of the project. Following this, the source code was annotated with well formatted comments that could be parsed by Doxygen. The html documentation can be found in `src/docs/html/index.html` (but it is not kept under version control).

2. Requirements Captured

2.1. Functional Requirements

The final project must be able to visualise at least 3 robots simultaneously. The morphology and neural network defining these robots must be defined in a file as per existing RoboGen guidelines for defining robots (as either `json` or specially formatted `txt` files). The user must be able to zoom in and out of the simulation, as well as pan across to view different parts of the simulation with more clarity. The user must also be able to pause and unpause the simulation.

2.2. Non-functional Requirements

The user should be able to interact smoothly with the simulation once started. That is, the simulation should not close before the specified simulation duration is over. The simulation should start within 2 minutes of running the `robogen-file-viewer` executable. On an adequately powerful machine, the swarm should be simulated at more than 15 frames per second.

2.3. Usability Requirements

While the simulation is running, console based output should inform the user of the details of the simulation, for example, when robots are added to the swarm or which configuration files are being read.

This output should be well formatted and provide information on various levels useful for finding problems, warning the user about potential issues, and if an error occurs, providing the user with sufficient information to solve the error.

2.4. Use Case Narratives

See Figure 1 for the Use Case Diagram.

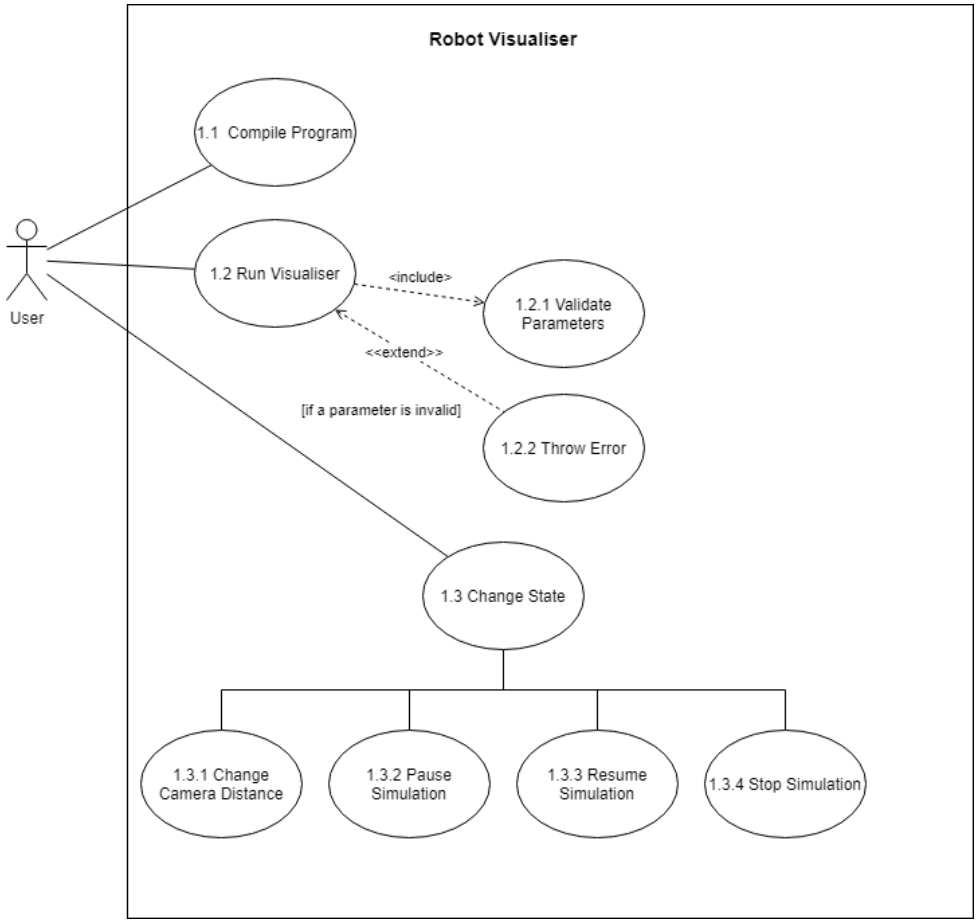

Figure 1: Use Case Diagram

2.4.1. Start Simulation of the Swarm. Actor: User

Primary Path: The user runs the program and is prompted to enter the name of a robot file and a configuration file (which includes how many robots to simulate in the swarm). The program loads a new window that is the visualiser, and displays the specified number of robots performing their tasks from the same robot file. While the simulation is running, the user may pause the simulation and zoom into the area where the robots are performing for a closer view.

Alternative Path: If the robot file entered is incorrectly formatted or does not exist then the program will throw an error and return the command line help page.

2.4.2. Run Visualiser. Actor: User

Primary Path: The user starts the visualiser, after that, the user needs to enter 2 parameters, location of the file that defines the robot and the location of the file that defines requirements and configurations for the file viewer. Should one of the parameters entered be invalid or non-existent, an exception will be thrown, requiring the user to enter those parameters again.

2.4.3. View Simulation. Actor: User

Primary Path: Once the simulation displays the robots performing their tasks in the run-time environment. The user may change the state of the view, by pausing,resuming and stopping the simulation. The user may also zoom in closer to the robots or zoom further away. The simulation will stop once the time limit specified in the configuration file has passed.

3. Design Overview

Given that the project authors were extending the existing RoboGen [1] code base, changing some existing aspects of the existing software was considered out of scope due to time constraints and for existing RoboGen users to easily learn to use the RoboViz project.

3.1. Layered Architecture Diagram

3.2. Swarm Class

3.3. SwarmPositionsConfig Class

3.4. Analysis Class Diagram

See the Analysis Class Diagram in Figure 2

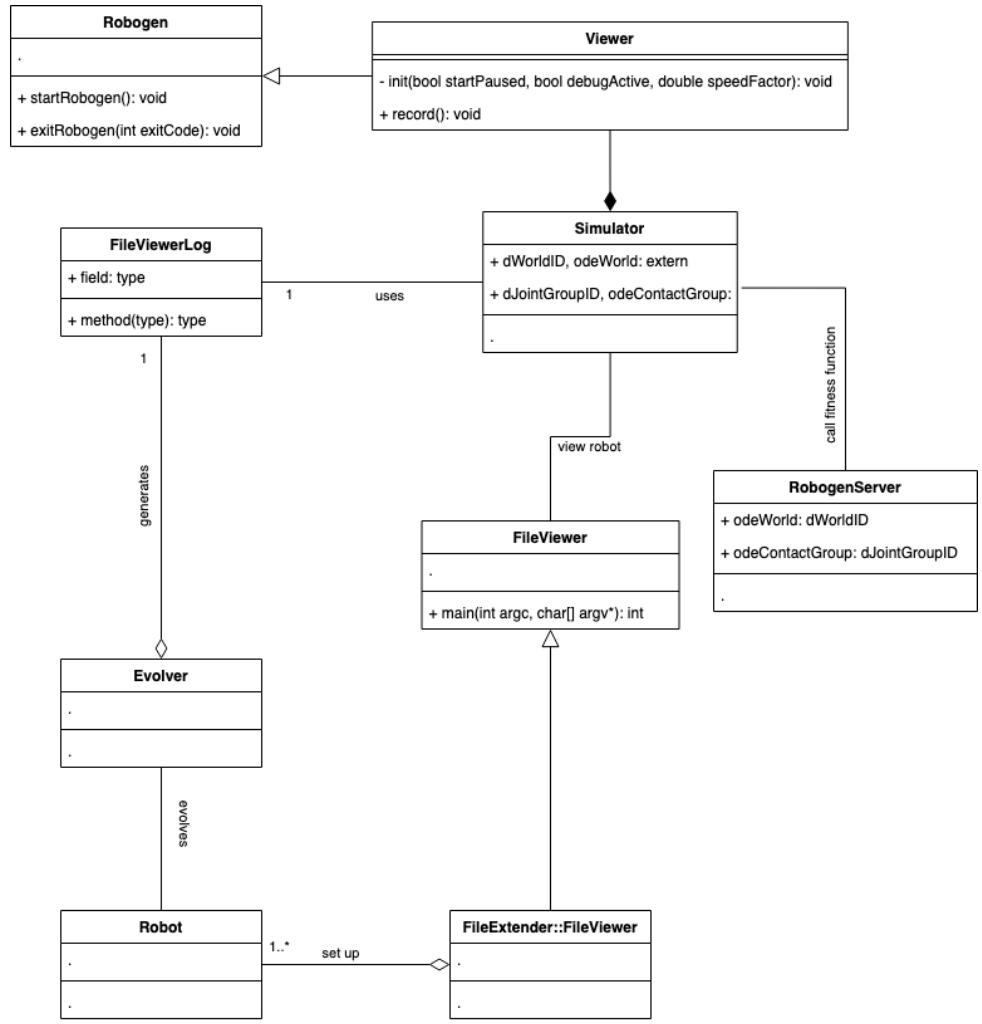

Figure 2: Analysis Class diagram

4. Implementation

4.1. Overview

The changes made to the existing project will be covered in the same order as they are encountered when a user runs the final executable.

The `robogen-file-viewer` is the executable used to run the simulation such that the swarm can be watched in real time. This executable had to be modified to read in some additional parameters (for example: `swarmSize` , `swarmPositions`) via configuration files, and then to store those parameters in the existing `RobogenConfig` object along with the other configurations. Since the positions of the robots in the swarm has to be specified in a separate file, this also required creating a new file configuration reader (`SwarmPositionsConfig`) to parse the swarm positions. This file was created to be similar to the existing `ObstaclesConfig` and `LightSourcesConfig` objects, and was integrated to the existing `ConfigurationReader`.

Once the configurations for a swarm are read in, error checking is done on those configurations and the program exits if the user requests an invalid combination of options (for example, a negative `swarmSize`). At this point various logging objects are initialised, and these were modified to log the details of every robot in the swarm.

At this stage the robots are initialised, with source files and swarm size specified in the above mentioned configuration files. Each robot's sensors, actuators, and its neural network are also initialised here. This can take some time, and so sufficient log output is printed to keep the user informed.

Now that everything is initialised, the main loop of the simulation begins. The simulation can run once (as is the case for a simple visualisation) or it can run multiple times (as is the case for when a population of swarms are being evaluated by the evolver. There are various protections in place to ensure the swarm is not taking advantage of errors in the physics simulation, which were developed to handle more than one robot. Additionally the evaluation of the individual robot's neural networks had to be extended to work for more than just one robot.

On completion of the simulation, the memory required by the dynamics engine [5] and the RoboViz swarm is freed.

4.2. Data Structures Used

4.2.1. Swarm Class

The Swarm class (`src/Swarm`) is the internal representation of a collection of robots. The members of the swarm are assumed to be attempting to cooperate with one another, and receive a single fitness score at the end of a scenario.

4.2.2. RobogenConfig Class

The simulator is designed to be flexible, and as such requires a lot of configuration parameters to be specified. These configurations are stored in the `RobogenConfig` object, often named `config` in the code base. However, the `RobogenConfig` object is only responsible for storing the well formatted and easily accessible parameters. The responsibility of parsing those parameters is handed to the `ConfigurationReader` class:

4.2.3. ConfigurationReader Class

The configuration parameters for the scenario (like the type and size of terrain, the starting positions of the robot(s), the fitness function to use) are parsed by the `ConfigurationReader`. Some of the parameters are specified directly in the configuration file as key-value pairs separated by an equals = symbol, while other parameters have values referencing relative file paths where a list of values can be found. These external parameters are parsed separately by different config classes (for example, `ObstaclesConfig`, `LightSourcesConfig`, and `SwarmPositionsConfig`.

4.2.4. SwarmPositionsConfig

The x,y,z locations of the individual members of the swarm can be specified in a separate file, and this file is parsed by the `SwarmPositionsConfig` object. The object is instantiated as a member of the

`ConfigurationReader`, and these configurations can later be accessed during simulations.

4.2.5. Scenario

A scenario is the combination of a swarm in rigid body simulation with a fitness function. When a scenario is initialised, the swarm is instantiated into the task environment, and when the simulation starts the swarm will be monitored by the scenario so that it's fitness can be calculated. A custom fitness function can be defined by creating a javascript file with the appropriate callbacks, or one of `racing` (fitness is proportional to average distance from the starting position) or `chasing` (fitness is inversely proportional to distance from the nearest moving light source) can be chosen. These fitness functions are defined in the various files found in `src/scenario/`

4.2.6. FileViewer Class

The `FileViewer` is the command line entry point for starting a simulation. It takes in two command line arguments (the path of the file defining the robots, and the path of the configuration file), parses those arguments using `boost` program options [4]. This is the file that becomes the executable used to start the simulator for the purpose of viewing a swarm.

4.2.7. IViewer Class

This is an interface defining common functionality for classes wishing to view the current state of a simulation. For example, the simulation can be viewed in the web browser or on desktop.

4.2.8. Simulator

The `Simulator` has a single (overloaded) method, `runSimulations` which receives a scenario, a RobogenConfig, and optionally a viewer. This method then initialises the viewer (if applicable), each robot in the swarm, the rigid body simulator, ODE, and sets up the scenario in anticipation of calculating the fitness of the swarm. The simulation is then started up, and run for a duration specified in the `RobogenConfig` object. This duration from starting the simulation in the world until the simulation is destroyed is called a trial. The simulator can be specified to perform multiple trials, or just one.

When the specified number of trials has been completed, the dynamics engine is closed, and all resources are cleaned up.

4.2.9. robogen.proto

Most objects in the Robogen project can be serialised into Google Protocol Buffer Messages [3], which provide a language and platform neutral method for serialising structured data, similar to XML but faster and simpler. The file `robogen.proto` contains definitions of the structure of the data, and was extended to allow the swarm class to be serialised.

4.3. Overview of the User Interface

4.4. Most Significant Methods of Each Class

4.5. Special Relationships between Classes

5. Program Validation and Verification

6. Conclusion

Here we must summarize everything, and provide a conclusion to the project's initial aims and goals.

7. Bibliography

References

[1] Joshua Auerbach. *Robogen – Robot generation through artificial evolution*. URL: `https://github.com/lis-epfl/robogen`. (accessed: 2021-08-01).

[2] Dimitri van Heesch. *Doxygen - documentation generator*. URL: `https://www.doxygen.nl/index.html`. (accessed: 2021-08-01).

Table 1: Summary Testing Plan. TODO: complete caption

Process	**Technique**
1. Class Testing: test methods and state behaviour of classes	TODO
2. Integration Testing: test the interaction of sets of classes	TODO
3. Validation Testing: test whether customer requirements are satisfied	TODO
4. System Testing: test the behaviour of the system as part of a larger environment	TODO

Table 2: A table of tests. TODO Complete Caption

Data Set and reason for its choice	**Test Cases**		
	Normal Functioning	*Extreme boundary cases*	*Invalid Data (program should not crash)*
Preliminary test (see Appendix 3)	Passed	n/a	Fell over

[3] Google Inc. *Google Protocol Buffers*. URL: `https://developers.google.com/protocol-buffers`. (accessed: 2021-08-01).

[4] The Boost Organisation. *Boost Standard Libraries*. URL: `https://www.boost.org/`. (accessed: 2021-08-01).

[5] Russ Smith. *Open Dynamics Engine*. URL: `http://ode.org/`. (accessed: 2021-08-01).

A. User Manual

This is the User manual