Almir Mehanovic
almeh12@student.sdu.dk

Individual Study Activity
Software Enginering Master
3rd. semester

January 4. 2017

Evolving robot morphology with genetic algorithms

Supervisors:
Tomas Kulvicius <toku@mmmi.sdu.dk>
Poramate Manoonpong <poma@mmmi.sdu.dk>

Figure 1: Leg-wheel robot

Introduction

This report is the result of an individual study activity in which evolution of robot morphology was conducted through simulations and genetic algorithms. Specifically the shape of the wheels for a simulated two wheeled robot were investigated.

While round wheels are good for moving in even terrains, they are quite bad for rough terrains, requiring large wheel sizes in order to climb obstacles. Walking robots are better and more adaptable on rough terrains but can not compete with regular wheels on smooth terrain, and they require complex limbs and controllers. The leg-wheel is a hybrid between a leg and wheel which keeps some advantages of both. *Figure 1* illustrates a robot with four leg-wheels, each with four spikes, climbing an obstacle.

The leg-wheel keeps the simplicity of a regular wheel, while giving robots the ability to better climb obstacles and maneuver on irregular ground. In order for a leg-wheeled robot to perform well on both smooth and rough terrains, the configuration of the leg-wheel must be optimised in respect to number and length of spikes.

Theory

Genetic algorithms are capable of solving optimization problems, using an approach inspired by natural selection. The approach works by encoding the problem solution in some string format called the genome. After generating an initial population of candidate solutions, bio inspired operators like mutation, crossover and selection are applied to evolve better

solutions over a number of generations. A fitness function is used to obtain a score for each genome, which determines its probability of moving to the next generation. The fitness function serves as heuristics to guide the algorithm towards convergence, without performing an exhaustive search. Genetic algorithms are applicable to any search problem as long as a genome representation and fitness function can be defined. The main drawback however is that a potentially heavy fitness computation needs to be performed for each genome generated as the algorithm progresses, thus making the genetic algorithm slow. From the above we can conclude that there are three steps which are required in order to use a genetic algorithm:

1) Define a genome representation of the problem
2) Define the genetic algorithm's operators and parameters
3) Define the fitness evaluation

Method

Approach

The overall approach of this project is to evolve the optimal leg-wheel configuration by simulating a robot in different environments. The robot would have a genome representing its leg-wheel configuration, and it would be evaluated in each environment. It's score would be the sum of distances travelled after 1 minute in each environment. Only distance travelled along an axis parallel to the robot's initial heading is measured.

Robot

Figure 2 illustrates the simulated robot used in this project. The robot consists of a flat cylindrical body, with two wheels and a tail attached. At the end of the tail is a small ball which can roll around freely. The tail ensures that the robot body is not dragged across the ground, provides stabilization when climbing obstacles and ensures that the robot can still move if the radii of its leg-wheels exceed the radius of the body. The tail length was experimentally determined to be best when it was 1.5 times the length of the longest leg-wheel spike.

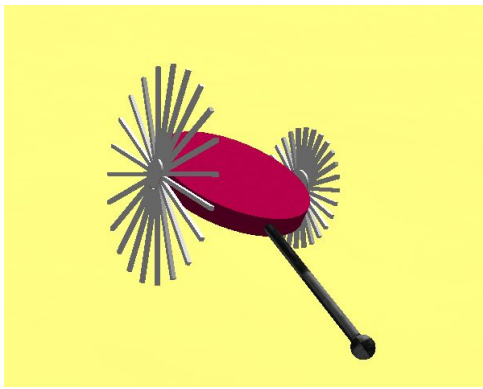

Figure 2: Simulated robot used in this project

Each leg-wheel consists of a small cylinder at the center, with any number of spikes attached around it. The radius of this cylinder is chosen automatically such that it has just enough circumference to hold all the spikes. The spikes on a single leg-wheel are evenly distributed and all have the same length, which however can be chosen arbitrarily. The other two dimensions of the boxes comprising the spikes are fixed.

The robot is equipped with an orientation and a position sensor enabling it to know its exact location and heading in 3d space. Noise, which would normally be part of any sensory input, was not simulated here since we are only interested in the performance of the leg-wheel, and not the robot's ability to maneuver with noisy sensor input. A motor is attached to each leg-wheel allowing the robot to move and turn by varying the power to the two motors.

Environment

6 different terrains were simulated in order to evaluate how well a given leg-wheel configuration performs. These terrains vary in their:

- Roughness
- Slip
- Hardness
- Elasticity

Figure 3 provides a visual overview of the different environments.

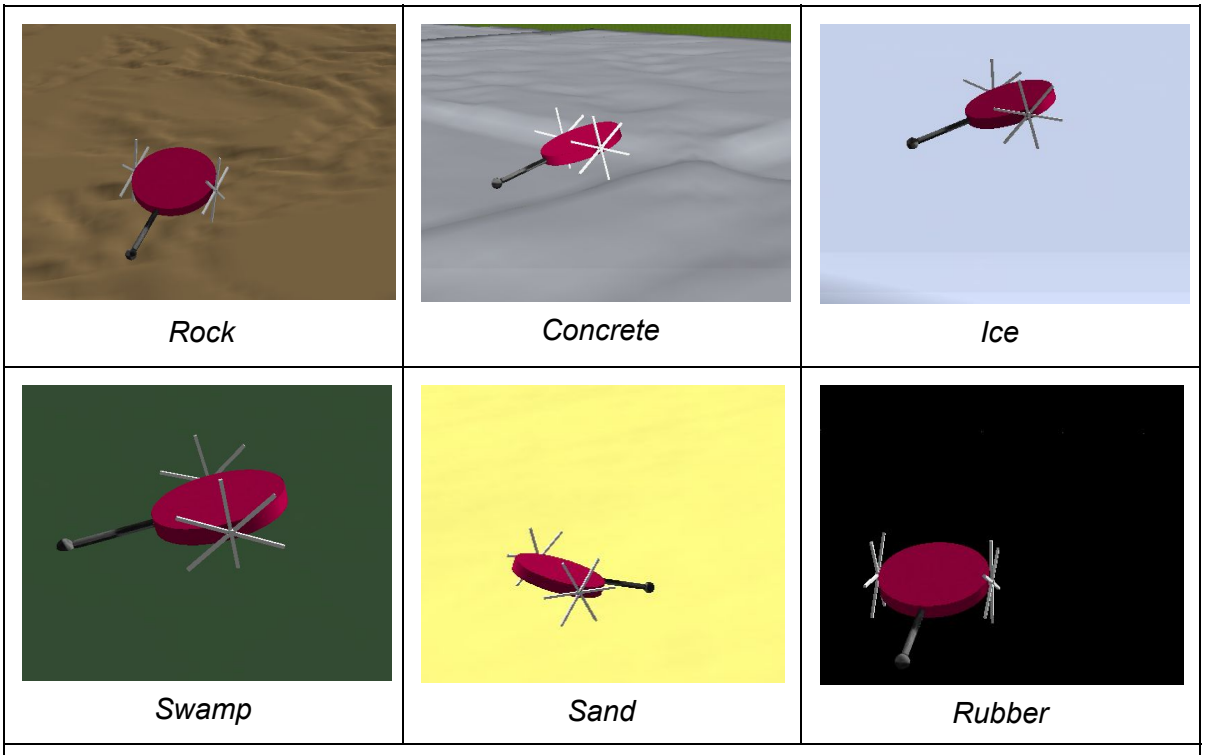

Figure 3: Illustrations of the simulated terrains

These six terrain types were chosen because they have quite differing properties, and provide a good coverage of different terrains. The rock terrain is a hard uneven terrain type in which it is easy to get stuck, either by a leg-wheel being stuck in a groove or by the robot body resting on an elevation from where it is unable to gain traction with the ground. The

concrete terrain is similar to the rock in texture, but consists of fairly plane surfaces arranged in different elevations. This tests the robot's ability to climb up and down steep obstacles. The ice terrain is a plane, hard and very slippery surface, which will test how different leg-wheel configurations affects the robot's ability to maneuver in slippery terrains. The swamp is a slippery very soft terrain, making it difficult to move around as the leg-wheels sink into the terrain. The sand terrain is slightly uneven, soft and slippery. The rubber terrain represents the ideal conditions for a round wheel as it is a flat non slippery surface. Testing that the leg-wheel configuration also works well on a rubber surface, ensures that it is adaptable.

Controller

In order for the robot to get a high score it has to travel in 1 minute as far as possible in the direction of its initial heading. To do so it needs a controller able to drive as fast as possible while keeping the initial heading. As the leg-wheels axles are fixed on the robot body, a differential drive controller is used to steer the robot.

In order to avoid that the controller influences the performance of different leg-wheel configurations a highly adaptable controller was needed. Many hours were spent trying out different approaches to create a controller which performed equally well on all terrains and with all kinds of leg-wheel configurations. I learned the hard way that there is no silver bullet robot controller which just works universally (*or at least it would take very long time to manually design one*). Instead of a handcrafted controller which would work universally, a better approach would have been to utilize some learning algorithm to create a separate controller for each terrain type. However, this realization came too close to the project's deadline, and thus a manually crafted controller was used in the end for the results obtained here. Pseudo code representing the essential behaviour of the controller can be seen in Figure 4. Many details making the controller better in certain terrains are omitted for clarity.

```
heading //heading direction of the robot
desired_heading //the heading of the axis on which travelled distance is measured
heading_error = desired_heading - heading;

if(heading_error < 0)
        // Means the robot's heading is off to the left. Eg. needs to reduce power on right motor
        motor_power_right = p(heading_error) // p returns a reduced power based on heading error

        if(heading_error > previous_heading_error)
                // Still drifting left so give left motor full power to stop the drift
                motor_power_left = max_power
        else
                //We are off to the left but are going towards desired heading.
                //In this case don't do max_speed on left motor in order to avoid overcompensation
                motor_power_left = motor_power_right + (max_power - motor_power_right)/2

else if(heading_error > 0) // Means the robot's heading is off to the right
//Identical to the above but mirrored
…
```

Figure 4: Pseudocode of the robot controller

Genetic Algorithm

Each leg-wheel configuration required four parameters to be represented in a genome. Those were the spike length and number of spikes for each wheel. These four parameters were encoded in a binary string of 30 bits. 10 bits for each leg-wheels spike length and 5 bits for their number of spikes. *Figure 5* illustrates how the genome encoded the leg-wheel configuration.

<table>
<tr><td colspan="10">Left wheel spike length
Range: 0.5 * robot_body_height
to 1.5 * robot_body_radius</td><td colspan="5">Left wheel no of spikes
Range: 1 to 30</td><td>Repeat for right wheel</td></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16 - 30</td></tr>
</table>

Figure 5: Genome representation of leg-wheel configuration

A binary-to-decimal mapping was made from the binary string to decimal values within the specified ranges. For example, a mapping of 8 bits and range of 0 to 255 would use 8 bits to represent the numbers from 0 to 255, inclusive.

While a genome should be able to represent any solution to a problem, it should still be designed such that it cannot represent infeasible solutions. This minimizes the amount of infeasible candidate solutions generated and evaluated by the fitness function. The limits on spike lengths are imposed because any leg-wheel with spikes shorter than half the body height won't be able to reach the ground, and leg-wheels with spikes longer than 1.5 times the body radius make the robot unstable and tilt to the side. Limits on number of spikes for each leg-wheel are specified because the robot can't move with no spikes, and anything more than 30 spikes for each wheel would make the fitness evaluation unfeasibly slow.

When the genetic algorithm begins it will initialize a population of 150 randomly generated genomes. Simulations on different terrains are performed in order to evaluate the fitness score of each genome. Then selection of genomes is performed based on the magnitude of the fitness score relative to the rest of the population. A genomes probability of being selected is equal to the genome's fitness score divided by the sum of fitness scores for all genomes.

Once Individuals for the next generation are selected, single point crossover at a random point is performed for each genome with a probability of 0.65. Finally a single random bit flip mutation is performed with a probability of 0.05. Now a new generation is ready to be evaluated by the fitness function.

Software and setup

The genetic algorithm used in this project was provided by the open source C++ library GAlib[1]. By default this library is not capable of running parallelized, thus a modified version of the library called GAlib-mpi[2] was used, with a few additional modifications to make it run in the setup of this project. With these modifications it was possible to run the genetic algorithm in parallel on many computational nodes using MPI.

[1] http://lancet.mit.edu/galib-2.4/GAlib.html
[2] https://github.com/B0RJA/GAlib-mpihttps://github.com/B0RJA/GAlib-mpi

Simulations were performed using the open source C++ robot simulator lpzrobots[3]. Sometimes lpzrobots would crash randomly with an unrecoverable segmentation fault, meaning the whole process running the simulator is killed by the OS. Thus in order to avoid the MPI job failing each fitness evaluation was performed by spawning a separate process which could be respawned in the event of a crash without causing the MPI node to fail.

The genetic algorithm was run on 900 cpu cores for 10 hours on the Abacus 2.0 supercomputer, hosted at the DeIC National HPC centre, SDU. 900 cores were used because the population size of the genetic algorithm was set to 150, meaning that the total number of simulations required pr. generation is 900 when each individual needs to be evaluated in 6 different terrains.

Results

During the 10 hours run of the evolution, the genetic algorithm evolved the initial population of 150 individuals for 422 generations. As each individual was evaluated in six different terrains, the total number of simulations performed was $150 \times 422 \times 6 = 379{,}800$. For every individual evaluated by the fitness function the individual's generation, leg-wheel configuration and travelled distance within each terrain was written to a file, which was loaded into R for analysis.

Figure 2 from the previous section is a visual representation of the highest scoring robot configuration, which was found at the 348'th generation. *Figure 6* shows the details of the overall best configuration, and compares it with the highest scores obtained by any individual across the different terrains.

<table>
<tr><td colspan="6">Leg-wheel configuration of highest scoring individual</td></tr>
<tr><td colspan="2">Spike length</td><td colspan="2">Left = 1.46 Right = 2.5</td><td colspan="2">Pct. diff (L/R) = 53%</td></tr>
<tr><td colspan="2">Number of spikes</td><td colspan="2">Left = 29 Right = 22</td><td colspan="2">Pct. diff (L/R) = 27%</td></tr>
<tr><td colspan="6">Score in different terrains (Total = 31.06)
x = score obtained by above overall best individual
(y) = highest observed score in the terrain by any individual during evolution
z % = x decrease compared to y (Eg. ((y-x) / y) * 100)</td></tr>
<tr><td>Rock
2.88
(7.33)
61%</td><td>Concrete
2.56
(5.32)
52%</td><td>Ice
5.43
(7.35)
26%</td><td>Swamp
6.08
(6.83009)
11%</td><td>Sand
7.03
(10.08)
30%</td><td>Rubber
7.08
(10.63)
33%</td></tr>
</table>

Figure 6: Individual with highest overall score

Two things can be observed from *Figure 6.* Firstly the robot is highly asymmetric with respect to spike length and number of spikes on each leg-wheel. And secondly the performance seems to be on average 36% worse compared to the highest observed scores

[3] http://robot.informatik.uni-leipzig.de/software/

in the different terrains. In response to the first observation *Figure 7* provides two scatter plots of spike length difference and number of spikes difference in pct plotted against the total fitness score.

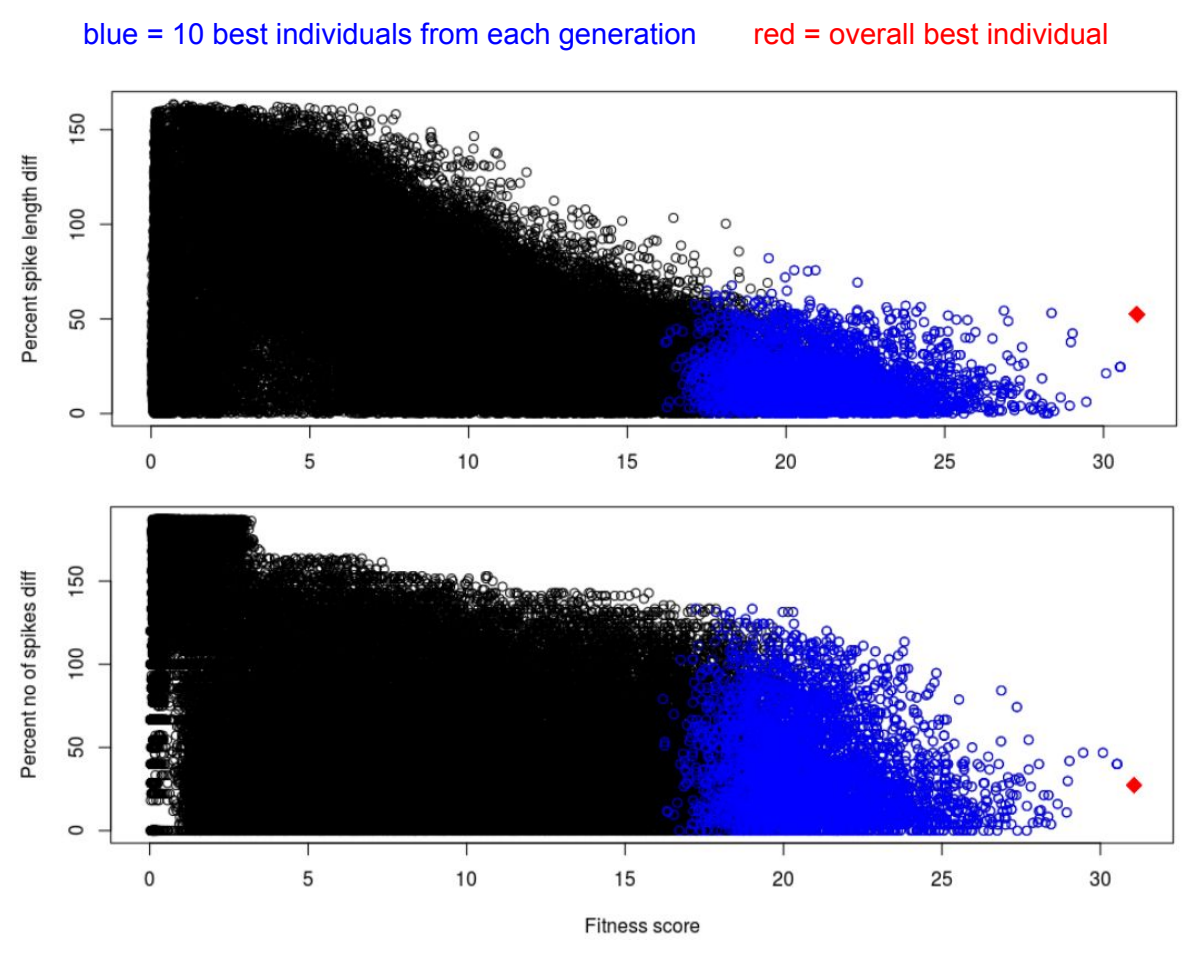

Figure 7: Scatterplots showing relationship between left/right symmetry and fitness

It becomes clear from those plots that there is a negative relationship between the percentwise difference and fitness score. The red dot on the plots indicates that the overall best scoring individual is somewhat an outlier, and that symmetry is in general preferred. The six different individuals holding records in one of the terrain types also have low spike length difference and number of spikes difference in pct. Namely, 13% and 11% respectively.

The second observation (the on average 36% worse performance of the overall best individual compared to the highest observed scores in the different terrains) could indicate that the configuration of the overall best individual is more adaptable, compared to the record-holders in each terrain which would have configurations more suited for the given terrain. However this explanation seems to be countered by the high similarity between the six record holders, as seen in *Figure 8.* Thus, a more likely explanation is that the environment is highly stochastic making small variations in configuration yield very different results. Thus instead of considering the single individual with overall best score as the

optimal configuration, a statistical method should be used to derive the configuration which gives the highest probability of a good fitness score.

Terrain	**Avg spike length**	**Avg no of spikes**
Rock	2.22	28.5
Concrete	1.95	20.5
Ice	2.14	20.5
Swamp	2.04	19
Sand	2.37	20.5
Rubber	2.46	24
Average of all	2.198	22.167
Standard deviation	0.195	3.516

Figure 8: Leg-wheel configurations of record holders

Knowing that the optimal configuration seems to be a symmetric one, and by looking at the average of all the record holders we could consider two symmetric leg-wheels with 22 spikes of length 2.12 the optimal configuration. Another approach would be to take the average of the overall best individual from each generation. This would give a configuration of 23 spikes with length 1.73. *Figure 9* provides six scatter plots visualizing the relationships between spike length and fitness score in different environments, and can be used to explain this difference.

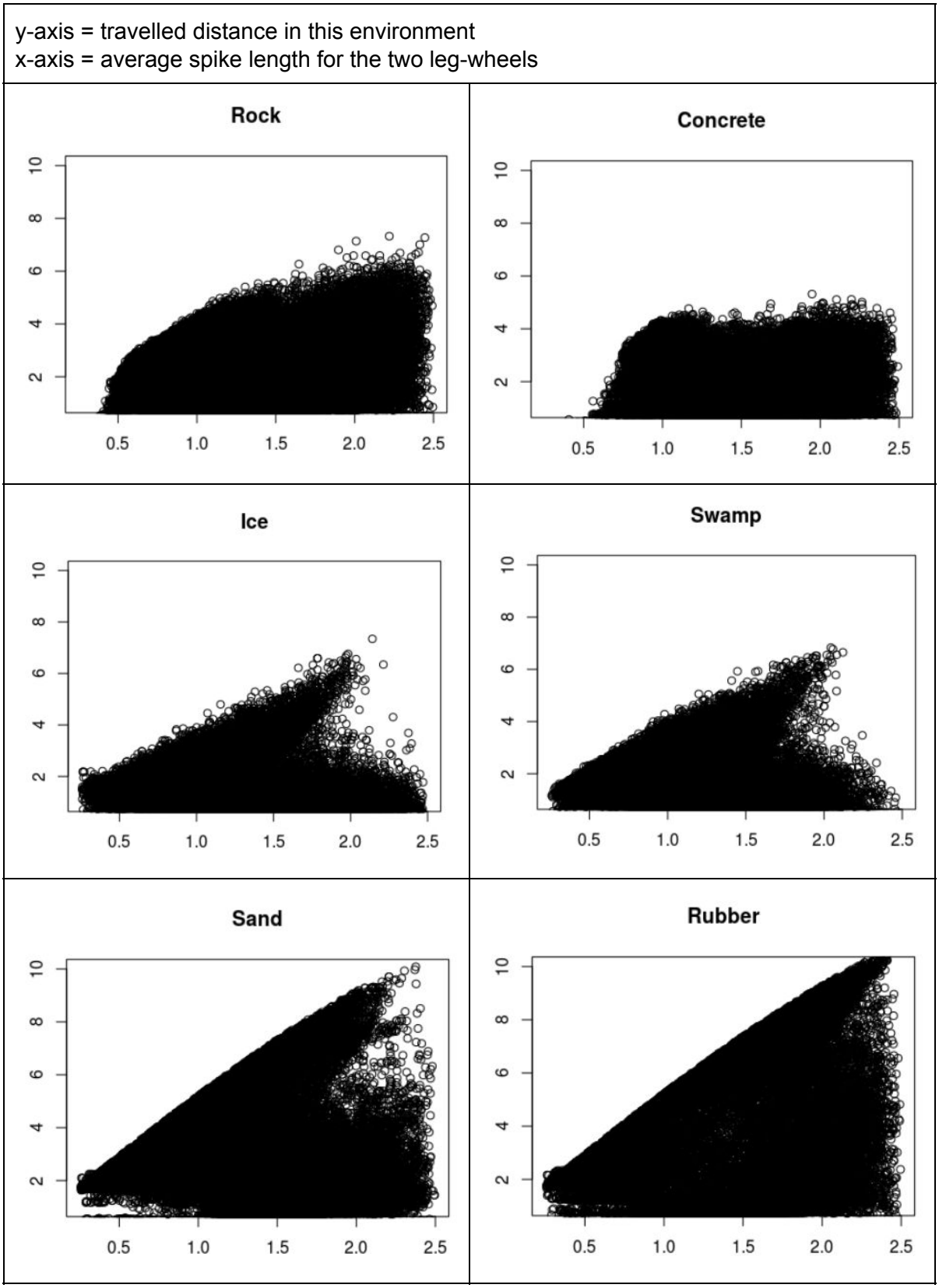

Figure 9: Spike length influence of performance in different terrains

From *Figure 9* we see that in general longer spikes are better, which makes sense as it would allow the robot to go faster. However, spike lengths above 1.7 seems to either yield very good or very bad fitness scores (especially in slippery terrains). This is because it increases the probability of the robot tilting over and rendering it unable to move further forward. Thus determining the optimal spike length becomes a tradeoff between stability and the possibility to go fast. The safe bet would be to have a spike-length at around 1.7. Increasing the spike-lengths beyond this point means the robot could go further, but with a high probability of tilting over.

Conclusion

As this is a learning project as much as a scientific one the focus was on the learning outcomes more than obtaining groundbreaking scientific results. The project did meet its goals, and activities relating to all learning objective were carried out in the project.

The project successfully investigated how an optimal leg-wheel configuration should be, but it is doubtful that the results can be used outside the setup of the experiments carried out in this project. In order to obtain more generalizable results the influence from the robot controller on the results should be minimized. Furthermore, this project's leg-wheel design was very limited. More interesting results would be obtained by allowing the leg-wheel to evolve more freely. For instance by letting each individual spike evolve a different size, shape, material and position on the wheel axle.

The main findings of this project are that a symmetric left and right leg-wheel are better for going fast. The optimal spike length becomes a tradeoff between stability and the possibility to go fast. On all the simulated terrains a leg-wheel with around 22 spikes yielded the best results, however it is not tested how the width of spikes might impact this number.