Assignment 4

TME290Autonomous robots

Axel Johansson
axejoh@student.chalmers.se

June 2, 2023

Solution proposal

How to run the code: Create a build folder and use cmake.

```
mkdir b
cd b
cmake ..
make
```

Then run

```
./tme290-lawnmower --cid=111
```

For the simulation run in one line

```
docker run -ti --rm --net=host -e COLUMNS=$COLUMNS -e LINES=$LINES -e TERM=$TERM
olbender/tme290-grass-simulator-amd64:v0.0.7 tme290-sim-grass --cid=111
--time-limit=0 --dt=0.1 --verbose
```

Task 1

Method

When the robot has charged the behaviour is set to *GoToTarget*. This behaviour generates a random valid coordinate on the grid and computes the path with **A***. The A* path is converted to commands. This is done based on the difference of the coordinates (dx, dy) from which an index is computed as

$$\text{index} = 3 \cdot (dx + 1) + dy + 1. \tag{1}$$

This index mapped in a vector that holds the corresponding behaviour.

When the path is reached the lawnmover can start to cut grass. The *CutGrass* behaviour stops every other turn and every other it moves to the highest grass. If rain is above a threshold is tries to *AvoidRain* by just driving and not stopping every other turn.

When the battery gets bellow a threshold, A* is run for each iteration to see if it can go another step before activating the *GoToChargingStation* behaviour. As before from the path control commands is created to get it home. When home it starts *Charging* until full.

Results

The mean grass length stabilizes at 0.5 when the simulation is run for shy of 3 weeks.

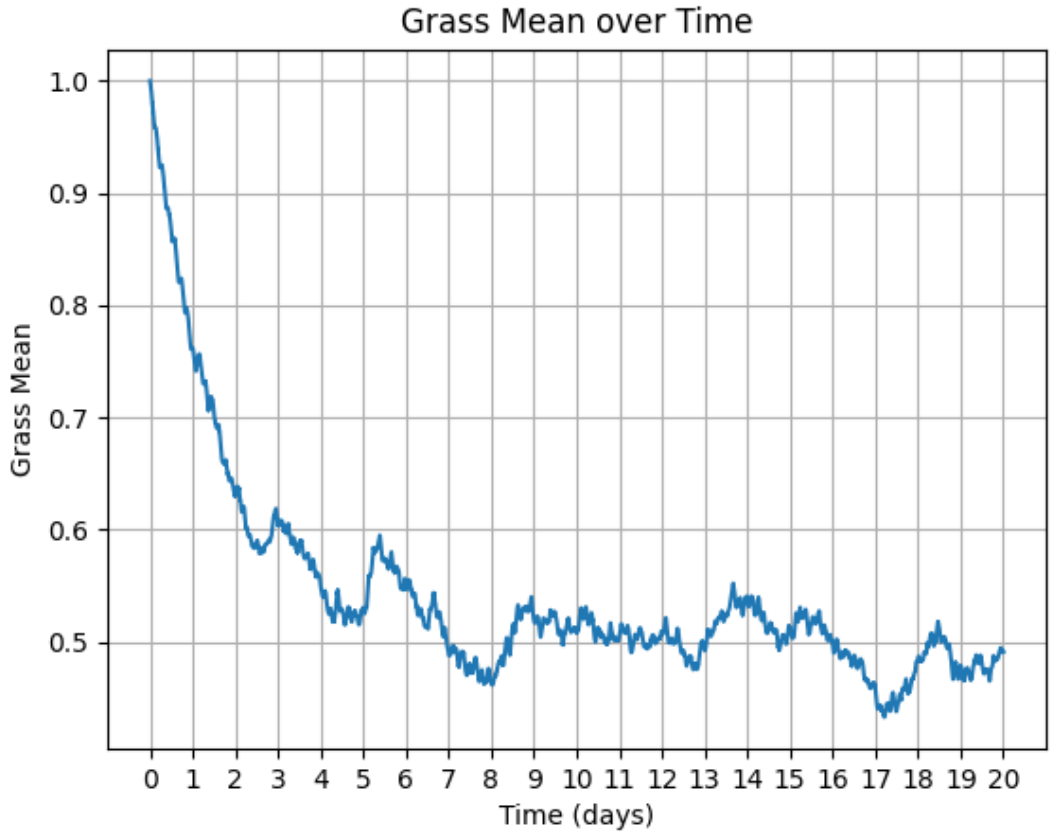

Figure 1: Simulation is run for 20 days.

Task b

This task was taught in Stochastic optimization algorithms.

Optimize BBR with ER

To optimize the solution using evolutionary robotics (ER), one can use the evolutionary algorithm to evolve the parameters of the behavior-based controller. ER or EA allows you to automatically optimize the parameters of your behavior-based controller, instead of manually setting them. To set up the evolutionary robotics (ER) to optimize the behavior-based robotics (BBR) controller, several key steps are involved.

The first step is determining the appropriate chromosome structure. This involves carefully selecting the sensors and behaviors that will be included in the encoding scheme. By representing the controller parameters within a chromosome, the optimization algorithm can manipulate and evolve these parameters to enhance the controller's performance.

The next step involves initializing the population. A population consists of a collection of individuals, where each individual represents a specific set of controller parameters encoded within their chromosomes. To introduce genetic diversity and enable exploration of the search space, the values within the chromosomes of the individuals are randomized. This initial population serves as the starting point for the optimization process.

The optimization process relies on a fitness function which has to be constructed. The fitness function evaluates the performance of each individual in the population in achieving the object. More about that in a later section.

With the fitness function in place, the next step is to implement a selection mechanism. The selection mechanism determines which individuals will be chosen for reproduction, based on their fitness values. These selection methods, like tournament selection or roulette wheel selection, assign probabilities to individuals based on their fitness scores, favoring individuals with higher fitness, gradually improving the overall population.

To create a new population with genetic diversity, reproduction operators such as crossover and mutation are implemented. This allows for exploration of different combinations of controller parameters and potential improvements. Mutation introduces random changes in the genetic material, enhancing diversity and preventing the algorithm from converging to suboptimal solutions.

Lastly, a termination criterion needs to be specified. This criterion determines when the optimization process should stop. It can be based on a fixed number of generations or when the fitness value reaches a satisfactory threshold.

By iteratively applying the selection, reproduction, and replacement steps, the population evolves over time. With individuals gradually improving their ability to keep the grass as short as possible. This lets the optimization algorithm effectively explore the solution space, evaluate different combinations of controller parameters, and hopefully converge towards a superior BBR controller.

Fitness function

The fitness function should reflect the effectiveness of keeping the grass as short as possible. One way to measure this effectiveness is by evaluating the average grass height. Additionally, including the maximum grass height value in the fitness function can serve as a penalty if the

robot fails to mow the entire area, which normally is desired.

Since the robot should never run out of battery this requirement could potentially be included directly or indirectly into the fitness function. Directly incorporating this requirement would assign very low fitness if the battery level reaches 0 during the task. Indirectly, this aspect can be covered by evaluating the average grass height over time. Since if the robot discharges the grass will grow back to mean 1.

By including these metric, the fitness function encourages the robot to balance its energy consumption with efficient grass cutting. This approach ensures that the evolutionary process generates controllers that are not only effective in grass cutting but also mindful of battery usage to accomplish the task without depletion. The fitness function would combine these two objectives, either by assigning weights to each objective or using a multi-objective optimization approach. The evolutionary algorithm would then search for parameter values that maximize the fitness function.

Pitfalls

ERs have several potential pitfalls that need to be considered for optimal performance.

Premature Convergence

The evolutionary algorithm may converge to suboptimal solutions (local maximum). This could happen if there is not enough exploration in the search space. This is especially true when using crossover which is a potent operation that causes fast convergence. To mitigate this, you can adjust the mutation rates, exploration strategies, or introduce mechanisms like elitism to preserve the best solutions.

It is possible for the evolutionary algorithm to converge to suboptimal solutions (local maxima). This could happen if there is not enough exploration in the search space, particularly when using powerful operations like crossover, which can lead to fast convergence.

To mitigate the risk of getting trapped in local maxima, several strategies can be employed. They include: Adjusting mutation rates can introduce additional variability, allowing for exploration of different regions in the search space. Dynamic mutation rates based on population performance or convergence rate can promote exploration. Introducing elitism preserves the best solutions across generations, preventing the loss of promising individuals and providing a basis for further exploration. Implementing methods like fitness sharing or crowding penalizes similar individuals, encouraging diversity within the population. Another technique is periodically resetting the population to a diverse state allows exploration of different regions and helps escape from local optima.

By implementing these strategies, the evolutionary algorithm can effectively explore the search space, maintain diversity, and avoid premature convergence, leading to improved performance of the behavior-based controller.

Overfitting

While the fitness function in this case may not be heavily tied to the specific environment, there is still a risk that the controller may struggle on new scenarios. If the objective is to optimize performance on this specific map, this is not a problem. However, if the goal is to create a more versatile solution, it is important to balance achieving the primary objectives and adaptability.

To address this challenge training the controller on multiple maps or varying environments can help enhance its ability to generalize and perform well in different scenarios. Another method is to incorporate penalty terms into the fitness function that encourage simpler solutions or penalize complex individuals.

Computational cost

Whilst its important that the individuals are not overly complex its important to find the right balance in the chromosome length. It is important to avoid overly complex individuals, as they can lead to overfitting. However, it is arguably more important to ensure that the *chromosome length is not too short*, as this can limit the capacity of the evolutionary algorithm to learn. This is because a chromosome that is too short does not have the number of sensors and behaviors required to grasp the complexity of the problem. This can impede the ability of the ER to discover effective solutions and hinder the learning process.

Hence, to determine an appropriate chromosome length that balances complexity and sufficiency is important. It should be long enough to provide the necessary genetic information to encode relevant traits and behaviors, while avoiding unnecessary complexity causing inefficiency.