Autonomous Robots:
Assignment 4

Erik Norlin

June, 2023

a)

The program is divided into four behaviors, namely returnHome, goToRow, cutGrass and chargeBattery. If the battery is lower than a threshold the robot returns home, otherwise it sweep cuts grass. If the robot has returned to the charging station it starts charging until its battery is fully recharged. When it has fully recharged it moves to a specific row to start sweep cutting. Every behavior that involved moving around was designed in a way such that the robot would never get stuck at walls. The threshold for low battery was semi-dynamic. If the robot was further away from the charging station it started to return home sooner. When sweep cutting, the robot cuts everything (command=0) which turned out to give better performance.

When running the program, the robot starts moving all the way to the right of the first row and starts sweep cutting going downwards. It starts returning home when the battery is depleted enough and recharges when it reaches the charging station. When it is fully recharged, the robot moves to the row below the one it started on the last time and the process is repeated. This means that the robot successfully covers the majority of the green area over time.

Running the program for over 70 days shows that the amount of grass converges to a mean of around 0.55, with a smallest recorded mean of 0.45 and a constant max of 1 (see figure 1) without ever running out of battery. This shows that the robot successfully cuts a significant amount of grass and keeps it on a consistent level. However, the robot had a hard time to decrease the max value since it never got below 1. The main reason for this difficulty is that the rain keeps the grass growing fast enough at the same time that the robot has to continuously return home to recharge. Further optimizing would have to be done to decrease the max.

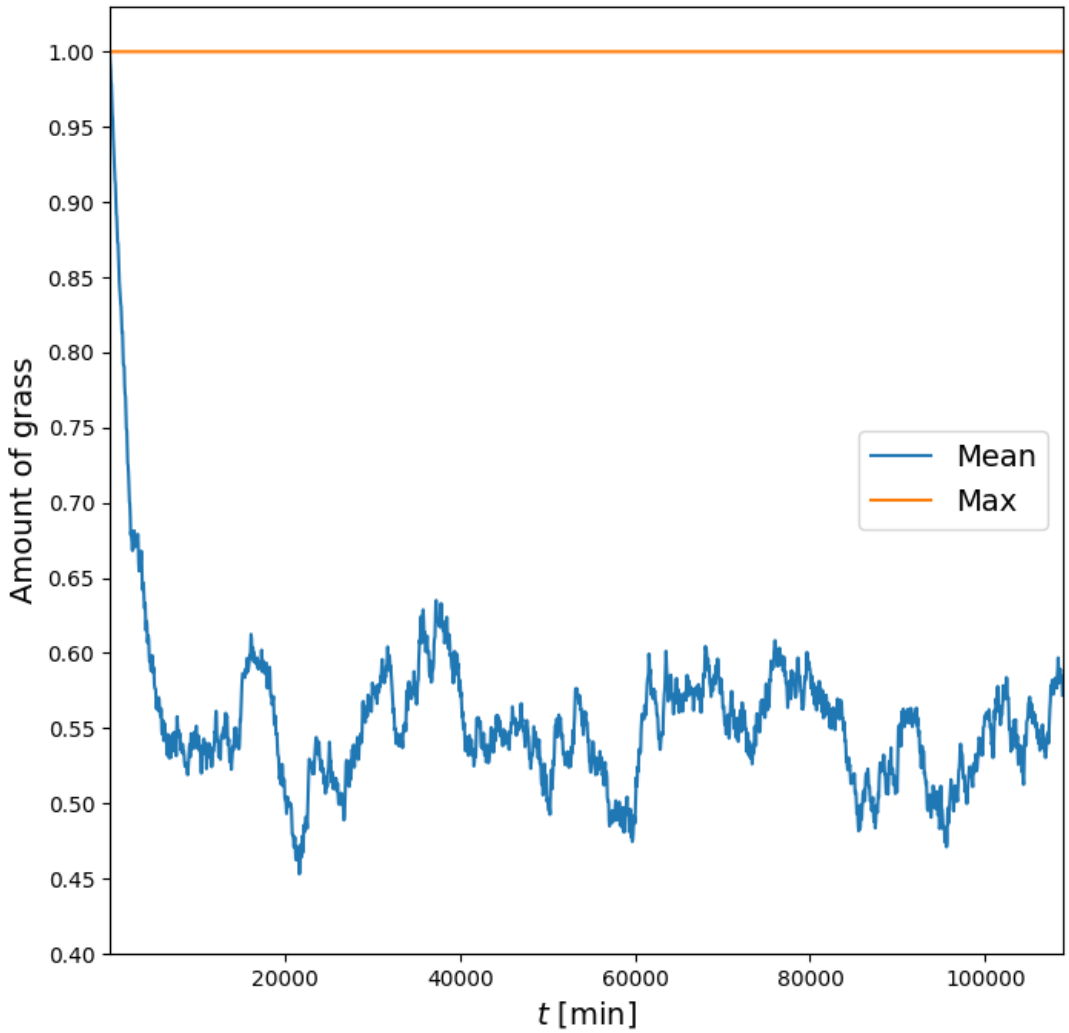

Figure 1: The amount of max and mean grass over time. The smallest recorded mean was 0.45 but stabilized at around 0.55. The max was constantly 1.

The only sensor values that were used for this task were the battery power and the grid position. Using these turned out to give a sufficient solution to cut a significant amount of grass. The solution could probably be further optimized by incorporating more behaviors that would utilize other sensor values such as the amount of grass nearby, rain, coordinate of rain cloud. This was not done due to lack of time.

The way the robot avoids getting stuck at walls are hard coded in this task. Applying these behaviors for other maps may not work properly. It would be better to use distance sensors to avoid walls, but the robot was unfortunately not equipped with this. One could possibly make use of the grass sensor values to judge whether if there is a wall or not. The challenge would then be to distinguish cut grass from walls assuming that both would give sensor values of 0.

Running the program

First login to the registry:
`docker login registry.git.chalmers.se`

Open two terminals. Run the simulation using this command in the first terminal:
`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`

Then run the controller using this command in the second terminal:
`docker run --rm -ti --net=host registry.git.chalmers.se/norliner/opendlv-assignment4:1.3 /usr/bin/tme290-lawnmower --cid=111 --verbose`

Furthermore, the whole source code for the program can be found here.

b)

The solution in part (a) was deemed sufficient but can be improved since there are numerous of possibilities to expand the implementation by finding optimal parameter values for the behaviors as well as implementing more behaviors. These behaviors could then be optimized using an evolutionary algorithm. This was unfortunately not done here but is still worth discussing.

Evolutionary algorithms optimize whatever it is specified to optimize which implies that the factors to be optimized should be carefully thought through. The best way to achieve higher performance is arguably to divide the task into as many behaviors as possible so that the algorithm can explore many combinations of ways of cutting grass and give a more fine tuned optimization. Theoretically, this could correspond to making the fitness landscape more complex and thus allowing the algorithm to find other maxima that would not exist if only optimizing one factor say. Speaking of fitness, to set up the fitness function, involving both the mean and the max grass value would make sense because it is reduction of grass in general that is interesting. Since a minimum amount of grass would correspond to a maximum fitness value it would be appropriate to define the fitness function as 1/(mean*max), or 1/(mean+max) alternatively.

To construct more behaviors for this task, one can make use of the sensor values that were not used for the solution in part (a). For example, given the position of the rain cloud as well as the amount of rain in the robots grid cell, a behavior could be implemented that makes the robot move in the opposite direction of the rain cloud if there is rain in the robots grid cell. This could be useful since cutting grass beneath a rain cloud does not cut any grass most of the times and therefore wastes battery. Another behavior that could enhance the cutting performance could be to move towards the max grass if the grass in the direction of the sweep cutting behavior is zero in order to ensure that battery is not wasted by not moving to empty grid cells. Or perhaps replace the sweep cutting all together and always follow the max grass. As can be noticed, one of the main problems here is that the battery keeps running out. Using an evolutionary algorithm could possibly maximise battery usage by finding an optimal dynamic cutting time such that the robot can sweep cut for a longer time the closer it is to the charging station before returning to recharge.

A potential pitfall of using an evolutionary algorithm is that if it is not set up properly it can lead to pre-mature convergence during training, meaning that the algorithm gets stuck in a local maxima in the fitness landscape. This could result in poor performance or failing to surpass the performance of the solution in (a).