Yann Vonlanthen 260808862

Assignment 4

COMP 417

McGill University, December 2017

Yann Vonlanthen 260808862

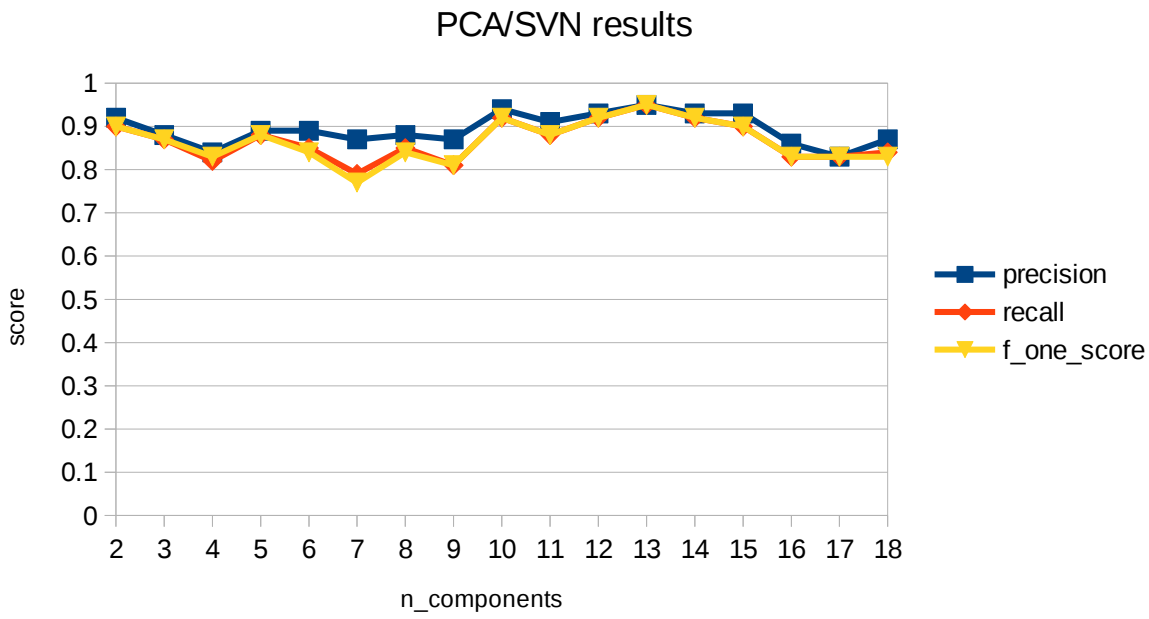
Part 0: Explanation given in Assignment Description

As discussed in class, PCA (principal components analysis) is a dimensionality reduction technique that takes a high-dimensional object (like an image) and allows us to approximate it in al low-dimensional subspace. An SVM (support vector machine) can then be used to define a threshold between different classes. For example, if we were classifying people response to “is it warm or cold” and SVM might discover that the threshold of 20 ◦ C was a good separator between temperatures described as warm and cool. In higher dimensional spaces, this threshold is called a liner separator or a hyperplane. Non-linear separators also exist and geocode.py supports using them, but they are not recommended for this assignment.

GPS coordinates refer to points on a spherical projection. Since we usually think of moving around on a locally- planar region, we need to do some conversion in general and specifically for this assignment. You should recognize that these tiles and the planar approximation you are using relate to the notions of a chart and atlas that we have seen in class long ago.

Part 1: Analysis of SVM / PCA performance

I ran geoclass.py with different number of support vectors on the training data. Below are the results when testing the obtained classifier on data it has never seen before.



The f1_score is probably the most informative, since it is a weighted combination of the obtained precision and recall.

We see that there is not a lot of difference between the different classifiers. For me classifier with 13 components had the best scores across the board. I thus saved it and will used it for the rest of the project.

Note that contrary to a naive intuition, using more components does not necessarily increase the SVM's performance. One possible explanation is given below.

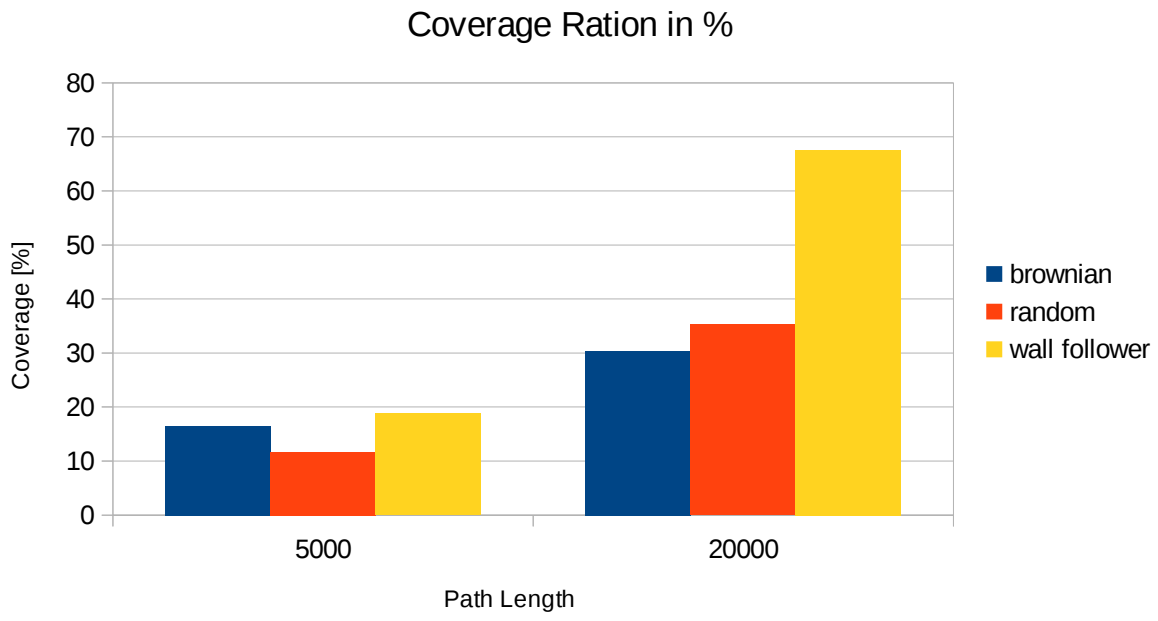
If we use too many points/vectors then our classification we will be able to fit the line / hyperplane perfectly through our training data. However this does not imply better performance on a new dataset, since we might have looked too much into the details of the training data and now remember details instead of having generalized what we saw.

An example I was given is when we have data points that are positively correlated but have some noise. In high dimension we can fit a line perfectly through all the points, but then we take the noise too much into account. If we now observe a new point our classification might be completely off.

Part 2: Map Coverage

I implemented three different Algorithms, two that are random, and one that is deterministic and keeps track of the world it visited. For each I ran the simulation with a for a long path length (20000) and a short path length (5000). This allows for easy comparison between the different methods.

The following chart gives a quick resume of the methods, useful for the analysis. (note that for a better analysis the probabilistic algorithms should have been run more than once)



Yann Vonlanthen 260808862

The Algorithm we want to implement solves a variant of the traveling salesman problem, that is known to be NP-hard, thus no polynomial time solution exists. Even checking whether a path is optimal cannot be done in polynomial time!

Algorithm 1: Brownian

This Algorithm should move the drone in random directions through the world. Doing a motion independent of the last movement results however in a unnatural path, that a drone couldn't realistically execute.

For this I used the concept of acceleration and proceeded to change the acceleration randomly instead of directly affecting the speed. The acceleration then changes the speed. This results in a smoother path.

Analysis and Explanation of Results

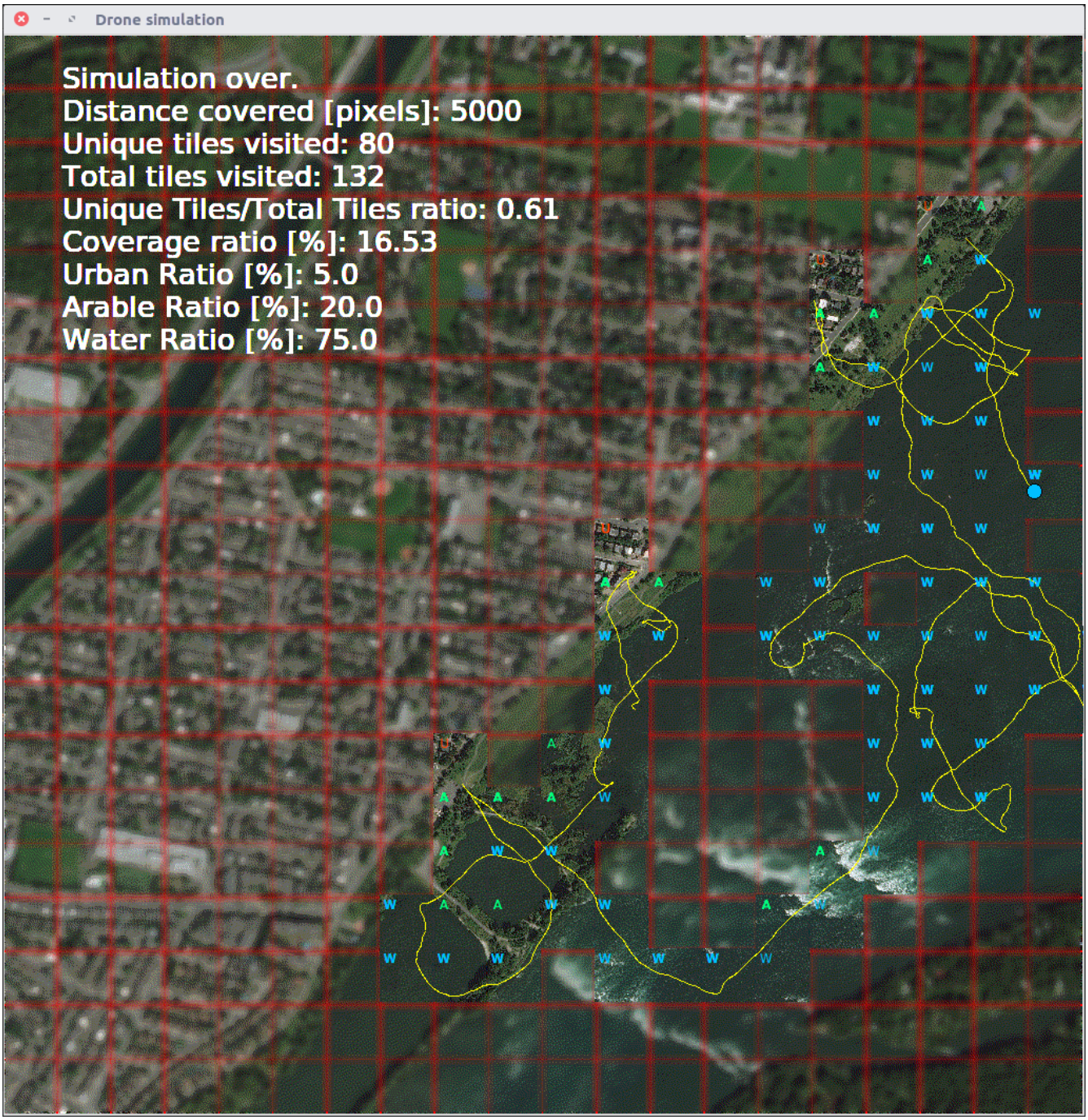
It is not surprising that for short path lengths a random movement can be pretty good, in this example it is just as good as the deterministic one. But since it is expanding outwards very slowly, it becomes less and less efficient with time/pathlength. This is reflected by the unique tiles visited to total tiles ratio dropping from 0.61 to 0.27 !

Advantage and Disadvantages

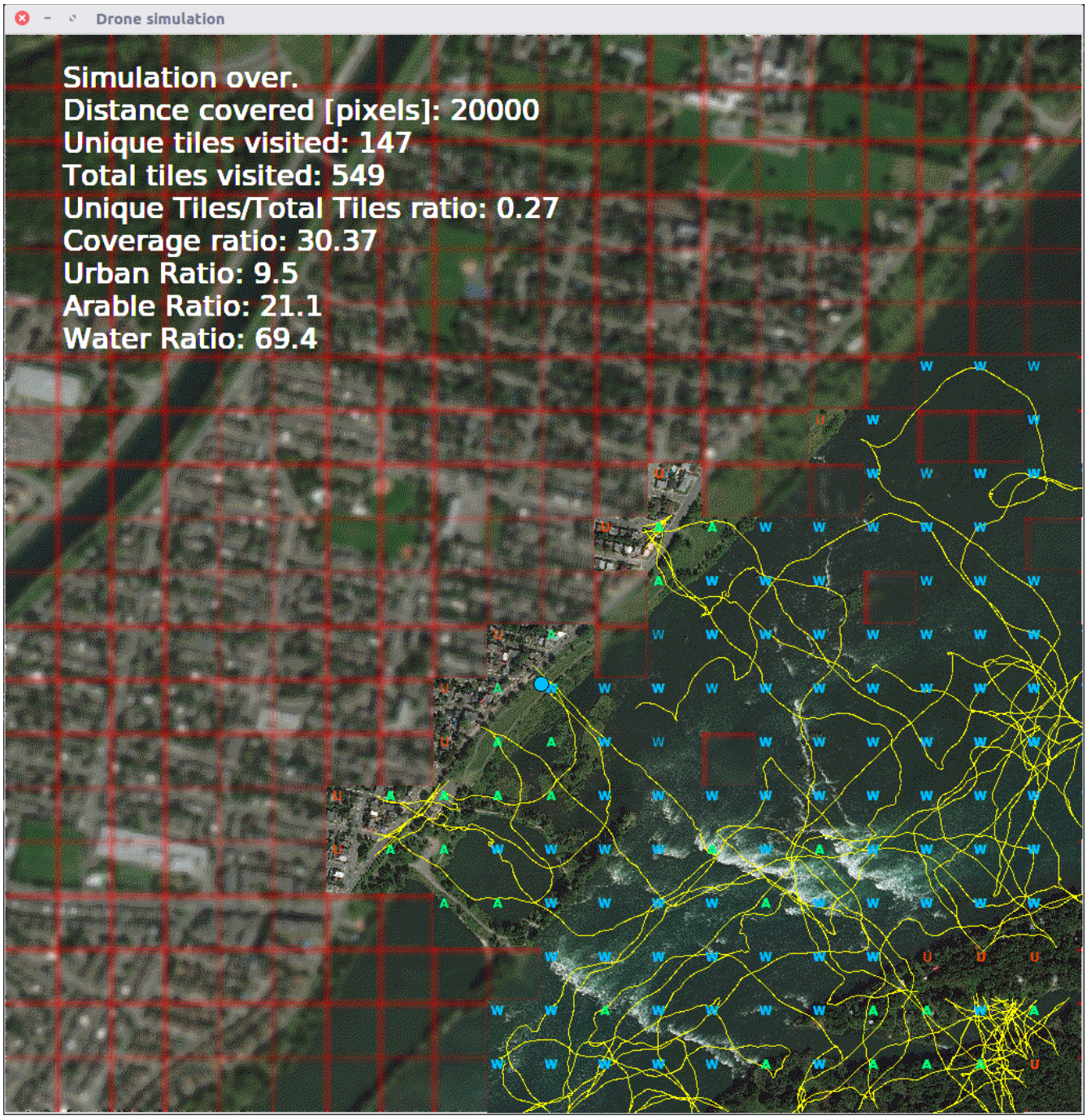
Advantages are easy implementation and the fact that when it is run long enough, the drone will eventually cover the entire map, i.e. it is probabilistically complete.

On the other hand it is extremely inefficient and can be improved easily.

Yann Vonlanthen 260808862



Yann Vonlanthen 260808862



Yann Vonlanthen 260808862

Algorithm 2: Random Lawn Mover

For this algorithm I wanted to copy the commonly seen robotic lawn movers, that pick a random direction when hitting an obstacle, while moving in a straight line otherwise.

Most of the commercial robotic lawn movers implement this strategy, and I thought it would be interesting to let it compete against the other two.

The only hard part here is to know from which direction the obstacle (here urban area, or map boundaries) has been hit. Of course we don't want to pick a direction that leads straight into the obstacle again! Since in this model we work with rectangular tiles we can pick an angle between -Pi/2 and Pi/2 in the direction we came from. (Note that what I refer to as direction here is left, right, top, bottom)

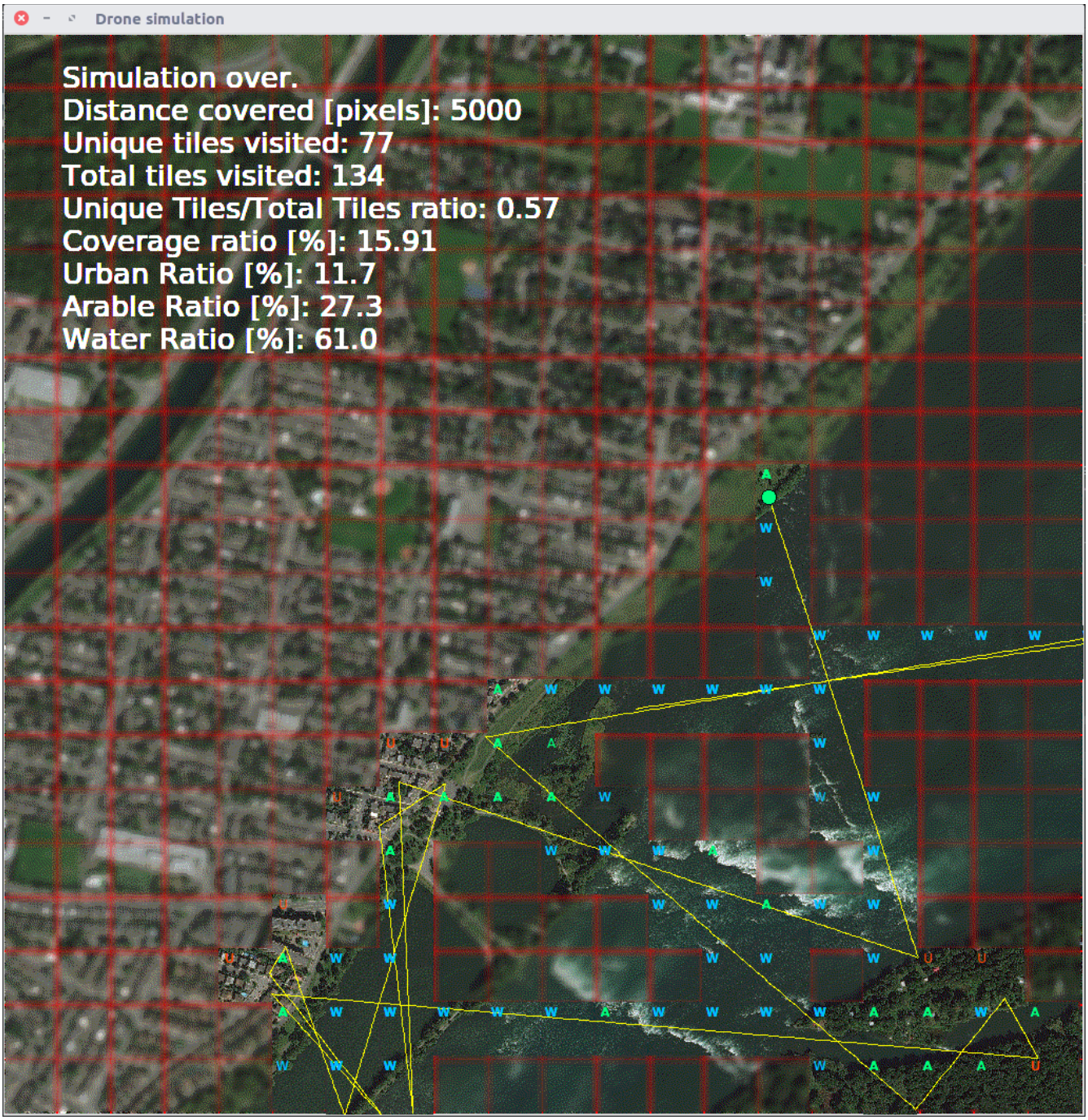
Analysis and Explanations of Results

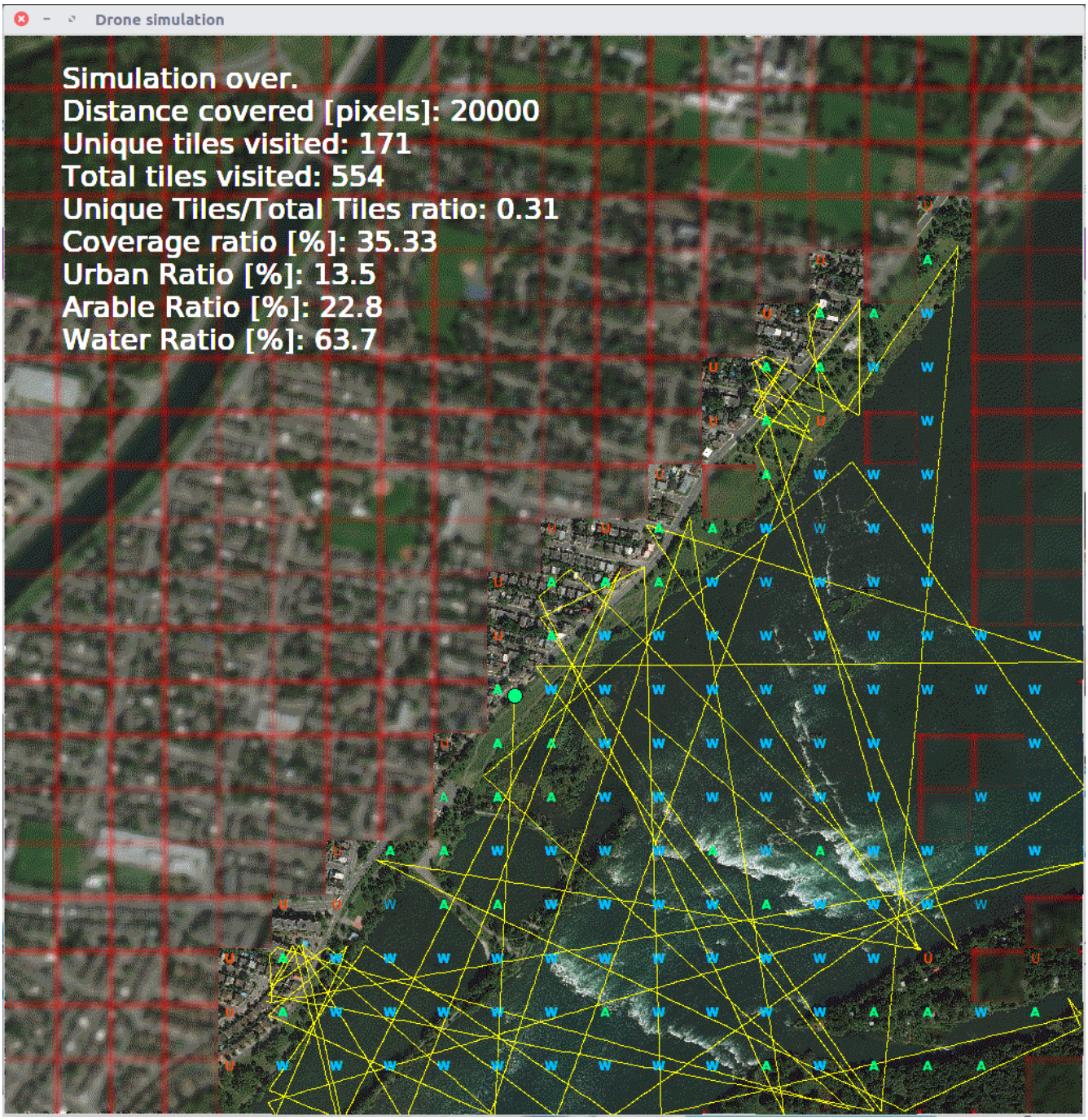
I observed that while similar for short path length to the brownian version, it becomes better with longer path lengths. I suspect this is due to moving in straight paths, which allows the drone to sometimes move straight into uncharted territory in a quick way. Just like the first algorithm it will hardly ever find small passages though and thus not scale very well with larger path length if there are some hidden corners, in comparison to a smarter algorithm. The unique tiles visited to total tiles ratio drops from 0.57 to 0.31!

Advantage and Disadvantages

Again advantages include a clear instruction set, and a very robust system. i.e. the drone is not prone to errors. Moreover it is probabilistically complete, though very inefficient. The ease of implementation is traded for a very sub optimal solution.

Yann Vonlanthen 260808862





Algorithm 3: Deterministic wall following Algorithm

As far as I know most people have two strategies to mow their lawn, that are often combined. First for large patches one often uses a boustrophedon coverage, i.e. going back and forth in straight lines. The second strategy consists of following the borders and obstacles on one side and working its way in a

spiral form towards the center. I chose to implement the latter as a third option. Note that this path planning strategy is completely deterministic contrary to the other two.

The first part of the algorithm consists of finding a wall. Then I mark the tile where we reach the wall as the first tile of the circuit and the coverage algorithm starts.

The basic heuristic is to always checking the right tile, if it is not out of bounds or urban area I continue in that direction.

Now to the tricky bit: In order to gradually move our way in a spiral form to the center (see example below) we need to store the tiles already visited. This has to be done carefully though, since we might block off our way back out of a patch if we refuse to revisit a tile. This is where I used the concept of coming back to the previously mentioned first tile of the circuit. **The drone only avoids visited tiles once it reaches its starting point**, i.e. it completed an entire “circle”. (There are some finer details and bad hacks that I had to do, but I won’t get into that)

Drawbacks and Possible Improvements

In the current version the drone remembers previously visited tiles (and doesn’t turn right on them anymore) but does not remember discovered city tiles. That means that this algorithm does not do any planning, it is purely a state machine. A possible and easy improvement would be to “look ahead” and don’t do any unnecessary right turns that could have been avoided with the previously acquired knowledge.

Additionally, the drone tries to go out of bounds before backtracking. This also could be checked preemptively. This flaw in turn means that my drone discovers a bit more tiles (since it counts tiles out of bounds too) and I chose to not improve this in order to achieve the 200 to 400 tiles mentioned in the description. (note that the coverage ratio stats takes this into account, i.e. the percentage is computed out of 484 and not 400 possible tiles)

A major problem (that appears in the example below too) is cutting off patches of land, that become unreachable afterwards. This happens since we avoid revisiting tiles after each “circle”. I didn’t have time to address this problem, but I think there is a possible fix, probably including path planning and thus not coherent with the idea of state machine.

Analysis

The flaw mentioned above means that the coverage algorithm is **not complete**. (i.e. it doesn’t always find a solution if there is one) If this were to be corrected I believe this algorithm could work well in many practical cases.

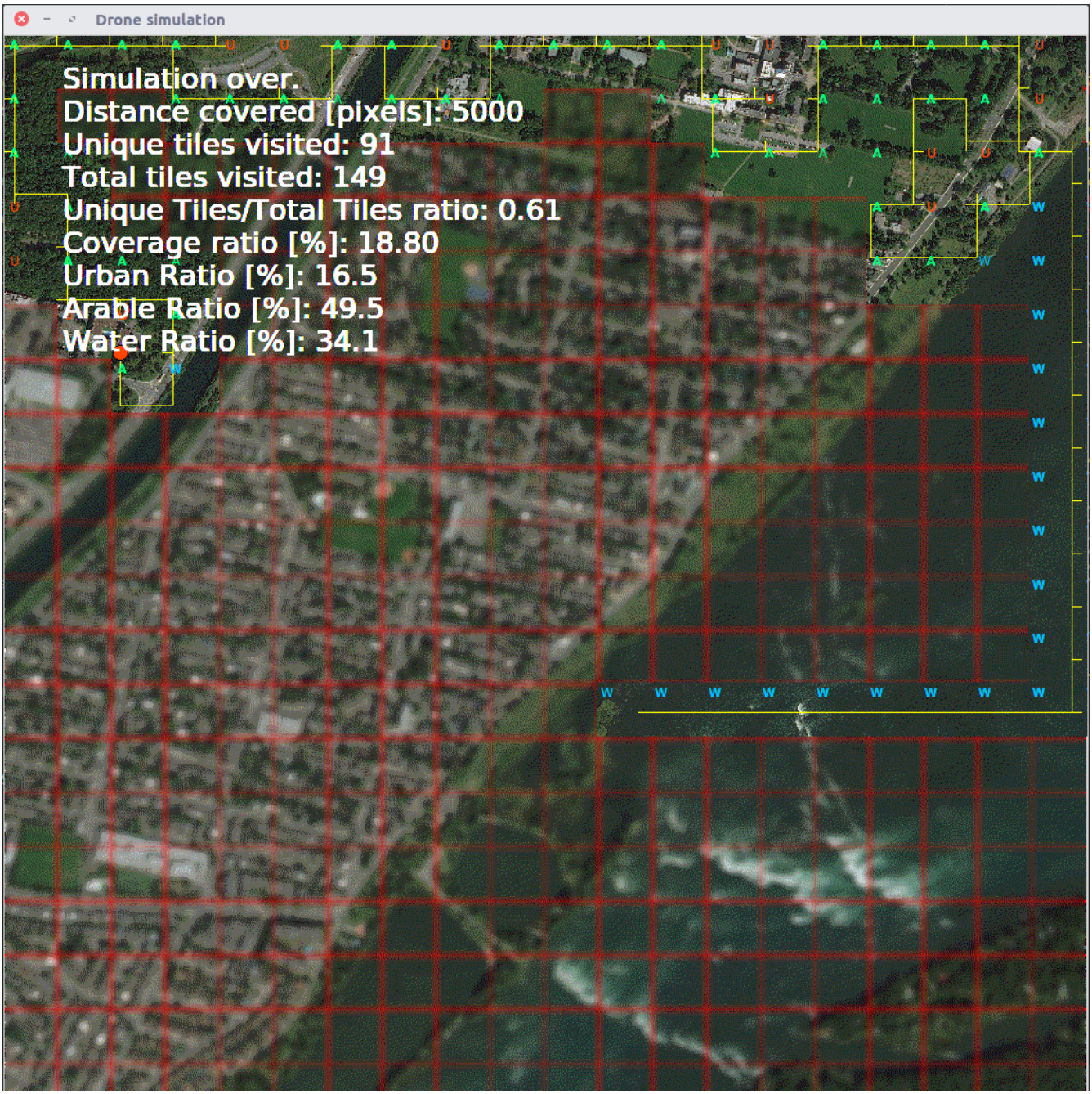
But even then in worst case it can be exponentially bad: Consider the case where we have two patches of land connected by a long, one tile wide path. The drone would take this path many times instead of

first covering the first patch and then the second! Contrary to the previous two algorithms the unique tiles visited to total tiles ration drops only slightly from 0.61 to 0.50.

Advantage and Disadvantages

Advantages include pretty simple implementation, while maintaining good efficiency. On the flip side depending on the map it can be surprisingly bad, and there are a lot of unnecessary drone movements, due to the fact that the drone does not look ahead (i.e. no path planning).

Yann Vonlanthen 260808862



Yann Vonlanthen 260808862