Toby William Towler

Registration number 100395626

2025

Robot Mower Mapping and Pathing

Supervised by Edwin Ren

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

The abstract of your report summarises your entire work () in no more than half a page. It should include the context of your work including its main objective, what methods you employed, how you implemented these, what the outcomes were and a final statement as a conclusion. It should not contain acronyms, abbreviations, elements of a literature review (though a statement of related work is permissible if it is crucial to your work) or future work. The abstract should be written when everything else has been written up and the project is finished! is this workng

Acknowledgements

Edwin Ren, Eden Attlebourgh

Contents

1. Introduction **5**

2. Background and Related Work **5**

3. System Design **6**

3.1. Map Generation 6

3.1.1. Corners 6

3.1.2. Obstacles 6

3.1.3. Graham Scan 7

3.2. Complete Coverage Path Planning 7

3.2.1. Robot Configuration 7

3.2.2. Headland Generation 8

3.2.3. Swath Generation 8

3.2.4. Route Planning 9

3.2.5. Path Planning 9

3.2.6. Cell Decomposition 9

3.3. Aerial Map Generation 9

3.3.1. Algorithmic Approaches 10

3.3.2. Machine Learning Approach 11

3.3.3. Data Set 11

3.3.4. Training 11

4. Performance Evaluation **11**

4.1. Map Generation 11

4.2. Complete Coverage Path Planning 11

4.3. Aerail Map Generation 11

5. Conclusion and Future Work **11**

References **13**

A. Map Generation **14**

B. Aerial Mapping **16**

List of Figures

1. Opencv example of canny detection, source Bradski et al. (2023) docs . 16
2. OpenCV canny edge detection on an image of a golf course 16

1. Introduction

The robot mower is an already existing project developed by previous masters students from the University of East Anglia. Physically, the mower has 2 tracks for movement on the sides of a metal frame, it is controlled by raspberry pi 4 running the Robot Operating System (Macenski et al., 2022). Sensor wise, the robot is equipped with a 4G dongle, lidar and a GPS chip that was upgraded to an RTK chip in this iteration of the project. The existing code base was mostly written in python with very small amounts of C++. Because of this, all of my code will be written in python to slot into existing code without issue.

My contribution to this project this year will be, regarding the overall movement and guidance of the robot. For ease of planning, I have broken this down into 3 sections:

1. Basic map generation
2. complete coverage path planning
3. map generation from an aerial image

These sections are all modular meaning they can be developed, tested and function independently but still easily be integrated together for the final product. The specified use case of the robot will now be to cut golf courses, this is particularly relevant to the aerial map generation section of my work which will likely use a machine learning model and require relevant training data, while the other sections are not concerned with a real world use case as their algorithms will work be able to tweaked for any applicable use case of this robot.

2. Background and Related Work

Another section that is essential and should keep its title as is! Although you could perhaps call it “Literature Review” instead, this is not advisable as at this stage of your project we do not expect an extensive literature review since this was already done in the second formative assignment. The rationale is simply because you will lose valuable pages that could be used better in the next two sections that will cover the preparation and implementation of actual work done. So just provide the context in which your project operates here, and then provide a brief overview of similar work that is directly

relevant to yours. Try to avoid blatant copying and pasting from the formative literature review as it is bound to read awkwardly.

3. System Design

3.1. Map Generation

Map generation is an important part of testing this system, it is important to test on all scenarios that may occur in the real world. For this reason, random or parameter based map generation is very necessary to guarantee success in every environment. As the outputs of this section will mostly be used for testing the path planning algorithm on regions with differing area, number of corners and complexity, no excessive algorithmic complexity is needed. This program should also be able to create n obstacles within the main field, such areas would represent obstructions in the mowers desired path, for example trees or telephone poles in the real world. This means we need a function with 2 parameters:

- K, number of angles in the outer field
- N, number of obstacles within the field, since it would not be sensible to take a parameter for the number of corners for every hole, we can generate them randomly assuming 3–8 corners staying inline with the complexity of the rest of the field without being unreasonably overengineered.

3.1.1. Corners

The number and distance between corners could be thought to represent complexity of a shape. The number of corners in a shape
KEEPGOING

3.1.2. Obstacles

Obstacles or holes, can be thought to represent real world obstructions for example trees or telephone poles in a real field. Generation of obstacles can be completed using the same function as the outer field generation, simply with different parameters. The

algorithm 2 takes 3 parameters, number of points in the shape, an origin point and the range new points. This allows for variable size, positioning and complexity.

3.1.3. Graham Scan

The Graham Scan (Graham, 1972) is an algorithm to find convex hulls, that is from a set of points the outline which contains all inner points. This algorithm does this by sorting the points by their polar angle to the lowest point, since this is always in the hull. There is then further calculations based on the angle between adjacent points to omit inner points from the outline, however for this use case that is not necessary since all points will be vertexes in a field. For this reason, the algorithm used is not strictly a Graham Scan but rather heavily based on the first stage, as shown in 3 Using this algorithm allows for consistent outlining of any set of points with no crossovers or intersections

3.2. Complete Coverage Path Planning

Complete coverage path planning(CCPP) is "the task of determining a path that passes over all points of an area or volume of interest while avoiding obstacles" Zhao and Hwang (2023) For this module, I have used the Fields2Cover library Team (2023), this library is open source. During the course of this project, I contributed to his library, fixing a bug during the build process. The library splits the task up into several sub-tasks.

3.2.1. Robot Configuration

Robot sizing has 2 important factors, track width, how wide the machine itself is, and blade width, how wide the utility object is. For farming equipment the utility object is usually larger than the vehicle, for example a combine harvester's wheels being narrower than blade, however for this project the blade is within the tracks. For this reason the functions will compute slightly differently to its probable intended use case as the tracks are likely to overlap however this should not cause an issue and all outputs should function as needed.

The robot has 2 more parameters regarding its turning circle, these are minimum turning radius and maximum differential curvature. Both of these are important when

calculating a path for the robot to follow as they ensure the robot can easily follow the path.

The robot is 22cm wide from outer edge to edge and the gap between the tracks is 17cm, with a track width of roughly 2.5cm.

3.2.2. Headland Generation

Headlands are the area in which a vehicle turns, think of the rough edges of a crop field, these exist so that agricultural vehicles do not trample their crops when turning. Although for the use case of a golf course this is not strictly needed as there are no "crops" or areas the robot cannot touch Headlands are also generated around obstacles to allow for suitable turning area around the obstacles if needed, this serves as an area to do a "U-turn", effectively the same as reaching the end of the field.

KEEPGOING

3.2.3. Swath Generation

Swaths are the individual straight lines that make up the complete path, they are often parallel to each other, but this can vary depending on how the shape of the field is segmented into smaller shapes however they will always be parallel while in the same sub area. There are 2 ways of determining the optimal angle; the sum of the lengths of all swaths, or the number of swaths, where lower is better for each. Both of these have pros and cons, for example lower total length offers better time efficiency assuming a near constant travel speed while lower swath count gives reduced turning movements which can be better if turning requires a slower speed or other complexities. Similarly, lower sum length of course means less distance travelled and usually less fuel/energy consumed, however the opposite can be true if the acceleration and deceleration causes greater energy consumption compared to constant velocity. Swaths are generated within the bounds of the headlands to allow for independent handling of the turning and connection of swaths regardless of the rest of the generation process. To find the best angle, each angle is tested and compared. This brute force approach could be considered slow, but it is all the library offers and could be taken into consideration in a future release.

3.2.4. Route Planning

A route is the order in which swaths will be covered. These can be sorted in a number of ways:

- Shortest route to cover all swaths, order depends on field
- Boustrophedon order, a zigzag like pattern covering swaths in order (1,2,3...)
- Snake order, Similar to Boustrophedon but skipping 1 swath each time (1,3,5...)
- Spiral order, a spiral pattern around the field, first swath then last (1, n, 2, n-1...)
- Custom order, user defined

KEEPGOING

3.2.5. Path Planning

Path planning refers to the connection of all swaths in route order. There are a couple of ways the to link neighbouring swaths, these are the Dubins curve and Reeds-Shepp curve.

Both of these compute the shortest path between the 2 points, with the deciding factor between the 2 is the robots ability to move backwards. Reeds-Shepp allows backwards movement

3.2.6. Cell Decomposition

3.3. Aerial Map Generation

Previously, for the program to take in a real world map from an image of the real world, the user would have to upload the picture and manually trace the outline, this could cause some problems. For example, the outline is only as accurate as the user's mouse placement, likely skipping some smaller corners to save time or due to inability to be accurate to the necessary degree. Automating this labourious and time consuming task would positively increase user experience and usability, this can be done using algorithmic and machine learning approaches.

3.3.1. Algorithmic Approaches

There are several algorithmic methods that try to achieve this task from several different librarys. A popular library for image processing is scikit-image.

OpenCV's Canny edge detection implementation Bradski et al. (2023) follows the classic algorithm developed by Canny Canny (1986). The algorithm works through several steps to detect edges in images:

- Gaussian Blur: First, the image is smoothed with a Gaussian filter to reduce noise, as edge detection is highly sensitive to noise.
- Gradient Calculation: The algorithm calculates the intensity gradients of the image using Sobel filters in both horizontal and vertical directions:
- Non-maximum Suppression: This step thins out the edges by keeping only the local maxima. For each pixel, it checks if it's a local maximum in the direction of the gradient. If not, it's suppressed (set to zero).
- Double Thresholding: The algorithm applies two thresholds:
  - High threshold: Pixels above this are considered "strong" edges
  - Low threshold: Pixels between low and high are considered "weak" edges
- Edge Tracking by Hysteresis: Finally, weak edges that are connected to strong edges are kept, while isolated weak edges are discarded. This helps ensure that the detected edges are continuous.

This approach works very well for high contrast images, particularly those black and white 1. The clear separation between foreground and background elements allows the algorithm to detect distinct gradient changes and create clean, continuous edge lines.

As for the similarly coloured golf course images, it can often find the outlines but they are made up of many smaller lines 2. This fragmentation occurs because the subtle color transitions between different areas of the golf course (like fairways, roughs, and greens) create weaker gradient signals that may fall inconsistently above or below the detection thresholds.

It may appear there is a simple solution to this problem; join the connected lines into one, however this proves to be rather problematic in itself for a number of reasons:

1. The desired line does not form a complete shape - Gaps in the detected edges mean that even sophisticated line-joining algorithms may fail to connect all segments belonging to a single boundary
2. The line is connected to other lines outside the desired shape - Many detected edges from shadows, texture variations, or other regions of the golf course can intersect with the boundary edges required to be isolated, creating unwanted connection points
3. Once lines are connected, it is impossible to tell which collection of lines is needed - Without prior knowledge of the expected shape or additional contextual information, there's no reliable way to determine which edge segments represent meaningful boundaries, like the edge of a green compared to an outline of trees or a rough area.

All of these means use of canny edge detection can often cause more problems than it solves, while it may have more accurate outlines in places, they may be connected to unwanted areas, Or they may not be complete and still require further user input to chose a selection or finish a shape.

3.3.2. Machine Learning Approach

3.3.3. Data Set

3.3.4. Training

4. Performance Evaluation

4.1. Map Generation

4.2. Complete Coverage Path Planning

4.3. Aerail Map Generation

5. Conclusion and Future Work

Another essential section that should keep its title as suggested. Briefly discuss your main findings, outcomes, results; what worked and what could have been done differ-

ently. Then summarise your work in a concluding statement by comparing your outcomes against the main and sub-objectives and/or MoSCoW requirements (if used) and suggest potential future work that could be done if more time would be available.

References

Bradski, G., Kaehler, A., and Pisarevsky, V. (2023). Opencv: Open source computer vision library. *https:// opencv.org/*. Accessed: 2025-04-13.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, PAMI-8(6):679–698.

Graham, R. L. (1972). An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074.

Team, F. (2023). Fields2cover: Mission planning for agricultural vehicles. *https:// fields2cover.github.io/*.

Zhao, S. and Hwang, S.-H. (2023). Complete coverage path planning scheme for autonomous navigation ros-based robots. *ICT Express*, 9(3):361–366.

A. Map Generation

Algorithm 1 Point Class Definition

```
1:  procedure CLASS POINT
2:      X ← −1                                  ▷ X-coordinate initialized to -1
3:      Y ← −1                                  ▷ Y-coordinate initialized to -1
4:      angle ← −10                             ▷ Angle initialized to -10
5:      procedure CONSTRUCTOR(x, y)
6:          this.X ← x
7:          this.Y ← y
8:          this.angle ← −10                    ▷ Default angle value
9:      end procedure
10: end procedure
```

Algorithm 2 Generate random points

```
1: function GENPOINTS(num, P, size)
2:     points ← []
3:     for i ← 0 to num − 1 do
4:         randX ← random_integer(P.X + 1, P.X + size)
5:         randY ← random_integer(P.Y + 1, P.Y + size)
6:         points.append(Point(randX, randY))
7:     end for
8:     return points
9: end function
```

Algorithm 3 Sort points by polar angle to origin

```
1:  function SortPoints(points, origin)
2:      hull ← [origin]
3:      Sort points by Y-coordinate
4:      for i ← 0 to length(points) − 1 do
5:          points[i].angle ← CalcAngle(origin, points[i])
6:      end for
7:      Sort points by angle
8:      Append points to hull
9:      return hull
10: end function
```

Algorithm 4 Main function

```
1: function Main
2:     hull ← []
3:     origin ← Point(20, 20)
4:     field ← GenPoints(20, origin, 400)
5:     hull.append(SortPoints(field, origin))
6: end function
```

Algorithm 5 Main function with holes in shape

```
1:  procedure Main
2:      hull ← empty list
3:      origin ← Point(20, 20)
4:      field ← GenPoints(20, origin, 400)
5:      add SortPoints(field, origin) to hull
6:      hole1Base ← Point(100, 100)
7:      hole1Points ← GenPoints(5, hole1Base, 50)
8:      add SortPoints(hole1Points, hole1Base) to hull
9:      hole2Base ← Point(150, 50)
10:     hole2Points ← GenPoints(3, hole2Base, 30)
11:     add SortPoints(hole2Points, hole2Base) to hull
12: end procedure
```

B. Aerial Mapping

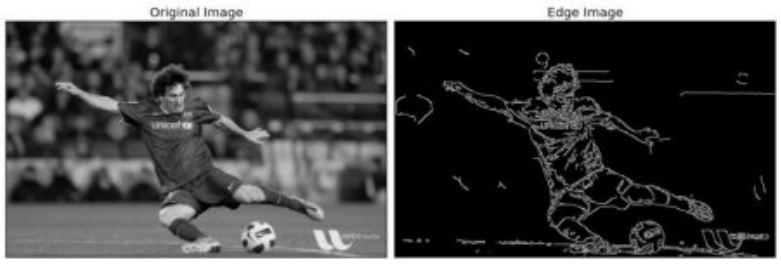

Figure 1: Opencv example of canny detection, source Bradski et al. (2023) docs

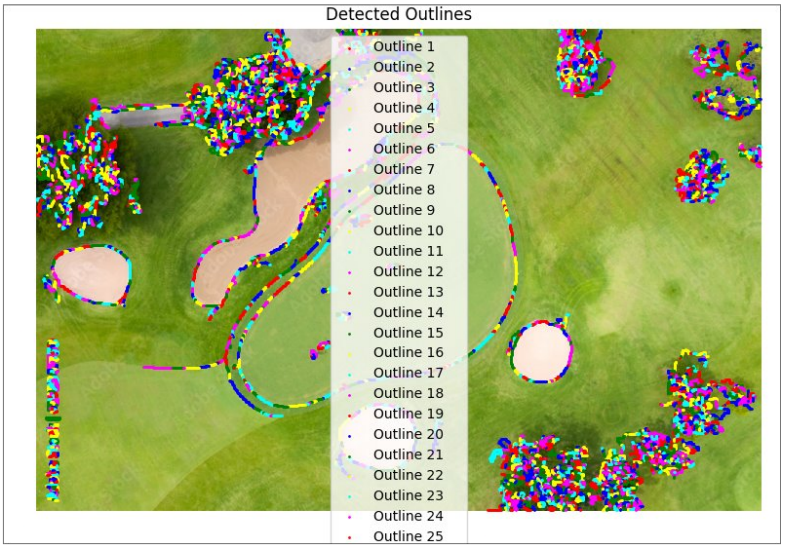

Figure 2: OpenCV canny edge detection on an image of a golf course