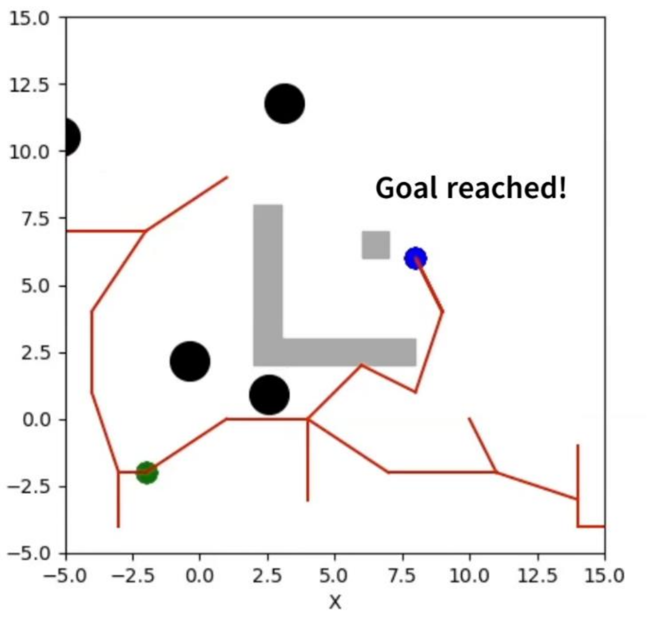
PIYUSH GOENKA

SALIN247 Code Challenge

Path Planning

1.1 Output Video



1.2 Discussion

Method:

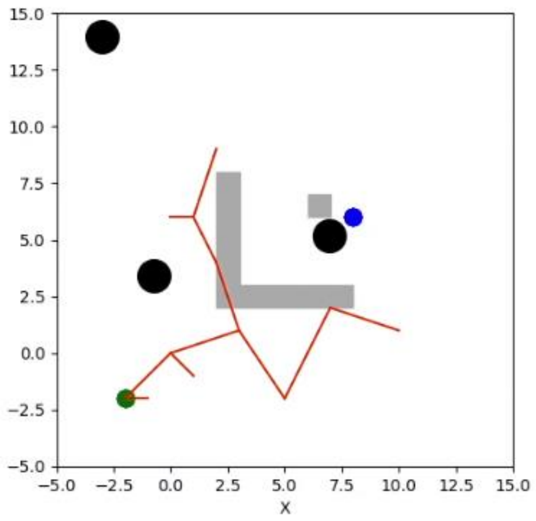
I used the RRT algorithm to find a path from start node to goal node. The generate_path method runs one iteration of the RRT algorithm. The generate_path method is called in the update_animation method for every frame since one of the inputs to the generate_path method is *time of simulation*. This implies that generate_path method needs the current time of simulation. One way of providing that in our context is by giving the current frame number.

If the total number of frames is known before we can compute the path then the path can be computed completely before displaying it in the simulation.

Performance:

The RRT algorithm performs well and manages to find a path most of the times within the 45 frames limit. It is also effective against static and dynamic obstacles.

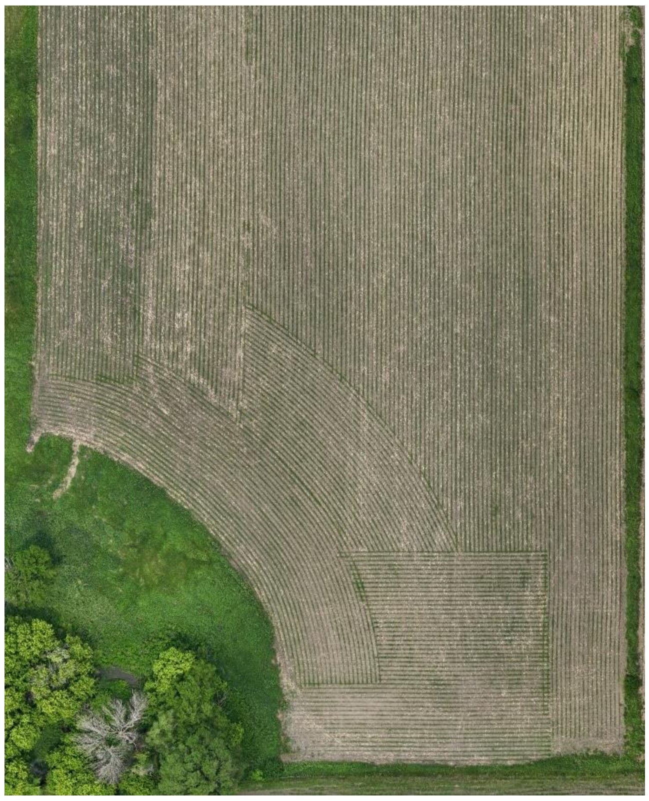
Note: Sometimes a path is seen to be generated through a static obstacle at 45° angle. It happens because I am discretizing all the points on the edge (line between parent and node) and checking for collisions. Hence the line segment that has coordinates with decimal values, for example, (2.546, 2.67) are not checked for collisions. The discretization was done to save computational time.

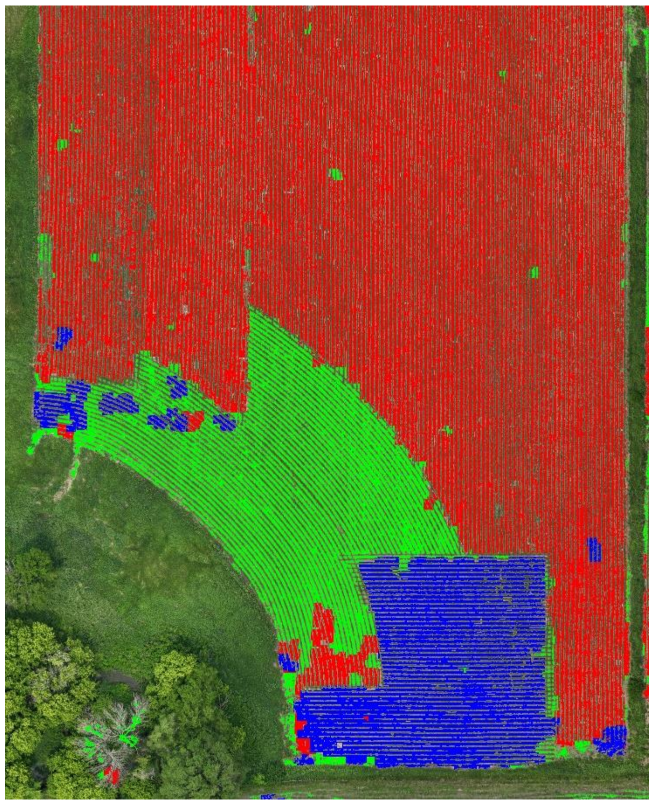


To further optimize the solution, different variants of RRT can be used such as RRT*. Also, to reduce the time to find a path, we can start 2 trees, one from start point as well as goal point.

Instance Segmentation-

2.1 Original image vs Masked image





2.2 Discussion

Method used:

First the RGB image is converted to a binary image with white pixels representing crop rows. To extract the right crop row pixels, an examination of the image was done manually, and certain pixel relationships were extracted that were consistent in all crop row pixels.

- RGB pixels had a brightness level of over 100.
- R brightness was more or equal to G
- G/B ratio was less than 1.2.

Then the binary image is thinned using thinning morphing operation. Here, the crop rows reduce to one-two pixels in their width. Hence, we get the skeleton of the crop rows.

Then, we extract the vertical and horizontal edges using a custom filter. Once the edges are extracted, they are de-noised using another custom filter. Hence, we have two separate images with vertical and horizontal edges.

We then thicken these images with a kernel size of 50. The output consists of a big white patch in the respective edge region. Hence, we get two images, one with a mask for vertical edges and another with a mask for horizontal edges.

To extract the diagonal edges, we subtract the above two images from the image that contains all edges. Hence, we get the diagonal edges in a separate image.

We de-noise and then dilate the diagonal edges to form a mask for diagonal edges.

Finally, we iterate over the original image and identify crop row pixels using the conditions stated in the beginning. Once the crop row pixels are identified, we apply the 3 masks on the image one after another. In the end, we produce the output as seen above.

Advantages:

- Works for other crop row configuration of fields having similar colors.
- Works even if the diagonal edges are rotated in other directions.

Disadvantages:

- Would not work if crop rows were of a different color.

Shortcomings:

- The segmentation is not 100% perfect. It can be made better by using machine learning techniques or by using standard edge detection filters.