Assignment Report

for COMP4034 Autonomous Robotic Systems
December 2020

Callum Gooding
psycg2@nottingham.ac.uk
14298750

***Abstract*—This paper presents a system for using a robot to find four objects in a room. The resultant system performs well and has a high likelihood of finding all the objects in the room. The system uses a point clustering algorithm to select checkpoints within the room that allows for coverage of all areas and can generalise to any enclosed room. The system also uses it's depth camera to effectively determine the location of all of the objects in the room and can approach them, confirming their locations. The system also provides a comprehensive GUI that allows for monitoring and debugging.**

I. Introduction

The problem addressed in this paper involves using the ROS Melodic[1] framework to guide a simulated TurtleBot3 Waffle[2] around a room whilst finding a series of objects:

- A fire hydrant
- A green box
- A mail box
- A number 5

The criteria for having found an item is as follows:

1) Stopping at an appropriate distance from an object, and
2) Providing some indication that the object has been found

II. Methodology

A. Room Exploration

In order to ensure the entire room is explored, a set of points must be calculated that, when visited, provide a view of all possible locations an object may appear. This is accomplished using a point clustering algorithm. This section outlines the algorithm used.

1) Data Acquisition: The only data provided to the algorithm is the data published on the `map` topic. The `self.handle_map` function handles this data by mapping the following attributes within the message `msg`[3] to attributes in the `ObjectFinding` class:

- `msg.data` is mapped to `self.map` after being reshaped to fit `[msg.info.width, msg.info.height]`
- `msg.info.resolution` is mapped to `self.map_resolution`

2) Room Identification: The `self.map` matrix contains data that identifies three types of cell:

- Unknown cells, represented by -1
- Walls, represented by 100
- Free space, represented by 0

The first step is to retype the data from the `int8` to the `uint8` datatype. This serves to set the values of unknown cells to 255, allowing a simple thresholding pass to identify all the free space cells. This pass sets all the cells with values greater than 50 to 1. After subtracting the output element-wise from 1, the resulting matrix represents free space with a value of 1, and all other cells with a value of 0, as shown in Figure 1.

[1]http://wiki.ros.org/melodic
[2]https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
[3]http://docs.ros.org/en/melodic/api/nav_msgs/html/msg/OccupancyGrid.html

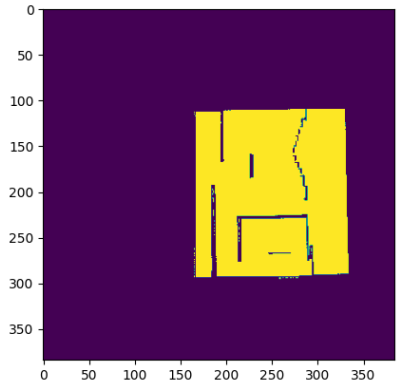

Fig. 1. Image of the processed map data. Free space is shown in yellow, walls and unknown cells are shown in purple.

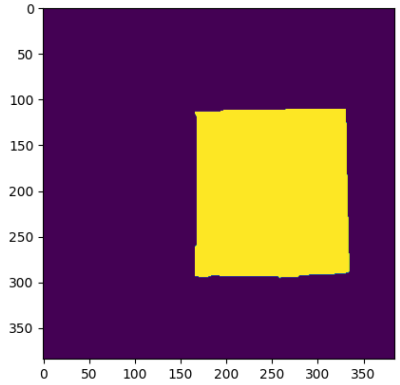

Fig. 2. Image of the morphological opening of Figure 1.

Next, the internal walls and voids are removed by a morphological closing of the data. The kernel used is an ellipse of radius 5 pixels. The purpose of this opening is to aid in identification of the room boundaries by making the contour simpler. The result of this closing is shown in Figure 2

After this step, the `cv2.findContours` function is used to find a rectangular contour that encapsulates the room. A frame of 5 pixels is then added to each side of this rectangle to ensure all of the room is included. With this frame, the image is then cropped to produce Figure 3

In order to populate the room with seed points, the disparate islands of space in the room image must first be removed. This is done by a morphological opening with an circular kernel of radius 3 pixels and is shown in Figure 4.

3) Seed Generation: With the matrix representing all available grid cells within the room, seed points are randomly distributed within the room. This is done by first calculating the real world size of the room, and then using a constant of 2 points/m^2, calculating the total amount of seed points to place within the room.

After the points are placed, those lying within a wall of unknown

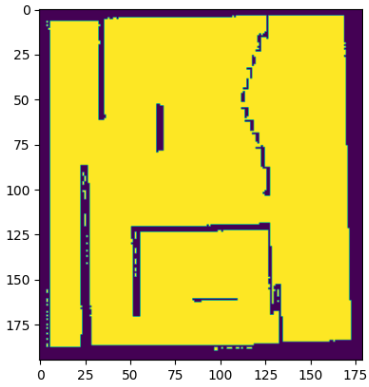

Fig. 3. Cropped image of the map data including only the room to explore.

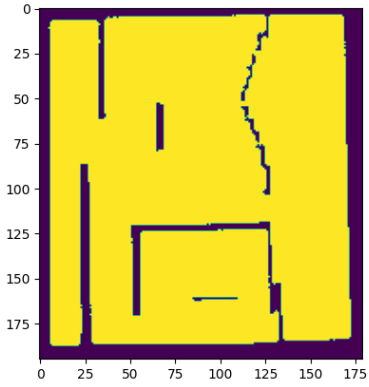

Fig. 4. Morphological opening of Figure 3

cell are removed, leaving the seed points shown in Figure 5.

4) Point Clustering: The seed points are next clustered by moving them away from the nearest wall cells. At each iteration of clustering, the nearest 5 wall cells W to point P are recorded, their normalised offset vectors averaged and then multiplied by the cluster distance per pass d, and then subtracted from P to provide P':

$$P' = P - \left(\sum_{w \in W} \frac{w - P}{|w - P|} \right) \frac{d}{|W|} \tag{1}$$

This calculation is performed over each point 5 times. The total cluster distance and amount of cluster passes is configurable, meaning the cluster distance per pass d is calculated to be 3 cells. The paths taken by the points during clustering is shown in Figure 6.

In order to speed up computation of the clustering vectors, a list of neighbouring point differences is computed before clustering. This

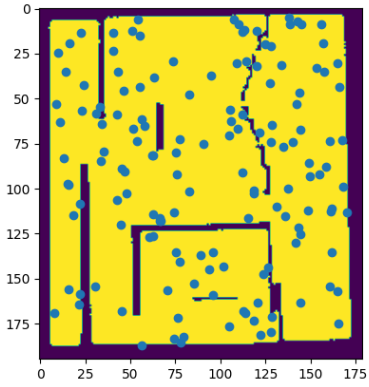

Fig. 5. Seed points randomly placed within the room after filtering those within walls or unknown cells.

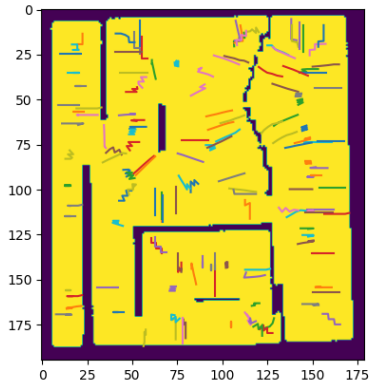

Fig. 6. Paths taken by the seed points during clustering.

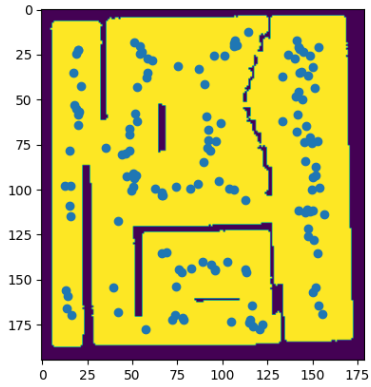

Fig. 7. Seed points after clustering

list contains all of the points with a magnitude of less than the clustering radius and can be used to replace $w - P$ in Equation 1. This list also stores the magnitude of the vectors, replacing $|w - P|$ in Equation 1. This list is also sorted by magnitude such that the wall finding loop can break after finding enough points, eliminating unnecessary computation.

The clustered points can be seen in Figure 7

5) Neighbour Location: The landmark points in the room are the locations of the most clustering. In order to find these points, "clustering" must first be measured. This first requires calculating the amount of neighbours each point has. Each point is hence looped over and, using the pre-computed offset magnitudes, neighbours counted. Neighbouring a point P is defined as lying within the clustering radius, defined as 30, of point P. Figure 8 shows the same points as Figure 7 but coloured to show the amount of neighbours they have.

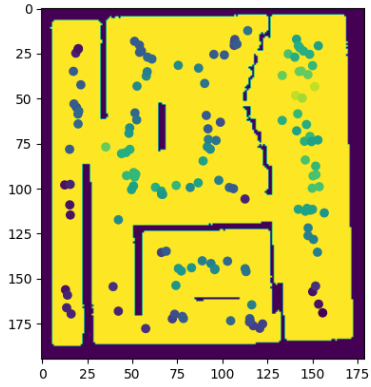

Fig. 8. Clustered points, points get lighter the more neighbours they have.

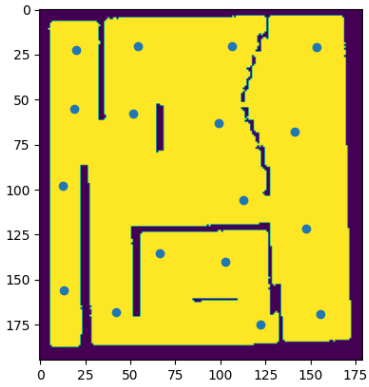

Fig. 9. The final points after local minima location.

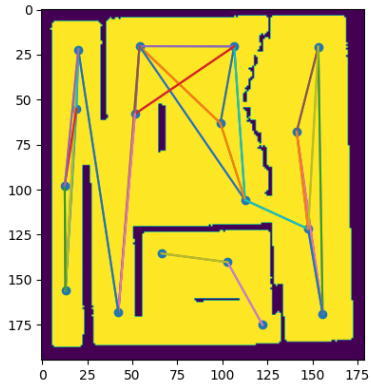

Fig. 10. Checkpoints with lines drawn between co-visible points.

6) Minima Location: The clustered points are now filtered based on whether they are a local minimum. Point P with P_n neighbours is a local minimum if all of its neighbours N have a neighbour count greater than itself:

$$M = \forall n \in N : n_n > P_n \tag{2}$$

Where M is $\top$ if P is a local minimum, $\bot$ otherwise and n_n is the neighbour count of point n, a neighbour of P.

After this, all points are iterated over and all those that neighbour points with a higher index yet equal neighbour count are removed. This is to handle a case where multiple points of equal rank neighbour each other. After this final pass, only the points in Figure 9 remain. These are the checkpoints to be used by the robot in order to find the objects.

B. Searching

1) Checkpoint Selection: A new checkpoint must be selected at three events:

- At the beginning of the search
- After a checkpoint has been visited
- After an object approach is completed, or aborted

As the exact length of time required to travel to any given checkpoint is unknowable, euclidean distance is the metric used by which to decide the next checkpoint to select. However, in order to account for the increased time requirement when moving around walls, an optimisation is made to the pool of candidate checkpoints.

At any given point P at which a new checkpoint must be chosen, all of the visible checkpoints are calculated and, if non-empty, is used as the candidate pool from which the closest checkpoint to P is chosen. If no checkpoints are visible from P, all of the unvisited checkpoints are used as the candidate pool.

The visibility of any given checkpoint from any given point is determined by walking the sight line between them. A point is considered non-visible if a wall cell is found at ant point along the line. Figure 10 shows this algorithm used to determine which checkpoints can be seen by each other. However, this graph is never fully computed and only the nodes visible from the points at which the robot must find another checkpoint are required.

2) Object Localisation: As the robot moves between checkpoints, it loops over it's object finders. These are functions that take the RGB image provided by the camera and, if possible, return the position of an object in camera-space. This position is then queried in the image provided by the depth camera. If the position is at a valid range, the real world position of the object can be estimated. If, for ten consecutive frames, position estimations are acquired that have a maximum variance of 30cm, the average of these estimations are used as the final localised position of the object. When an object is localised, the robot uses the navigation stack to approach it and mark the object as successfully approached.

C. Object Detection

The object detection functions operate by first converting the provided RGB image into HSV space. After this, a colour range is chosen for each object:

- The green box uses values between HSV(50, 0, 0) and HSV(70, 255,255)
- The fire hydrant uses values between HSV(0, 1, 0) and HSV (1, 255, 255)
- The mail box uses values between HSV(110, 1, 0) and HSV(113, 255, 255)
- The number 5 uses only HSV(0, 0, 0)

The object finders for the fire hydrant and post box also make use of a morphological opening followed by a closing in order to filter out noise.

III. Evaluation

The system is highly resilient to changes in robot starting position. Additionally, as the checkpoints are chosen algorithmically, the system can perform in any enclosed room provided it has a map. In testing, the system is able to consistently locate the objects and make sucessful approaches. However, when the objects are moved around, the detection functions can produce false positives, leading to object guesses in the wrong place. If the robot cannot approach all of the objects, it revisits all of the checkpoints again until it succeeds. This ensures eventual success in approaching the objects.

The system is very error resilient and can handle errors in the navigation stack, failed object approaches and blocked checkpoints.

The system's main failure mode is the approach to objects. This seems to be due to confusion in the navigation stack when switching between goals. There are two main solutions to this:

- More granular control of the navigation stack. Currently, the system uses the `SimpleActionClient` to interface with the navigation stack. A more granular controller, possible on the ROS publisher level, would allow for easier rectification of these errors.
- Waiting until all of the objects are located before approaching any of them. This would remove the need to change goals whilst moving between checkpoints. However this would additionally increase the total time between system start and end.

A. Checkpoint Selection Refinement

The constant with most effect on the selection of the checkpoints is the initial density of the seed points, shown in Figure 5. Too

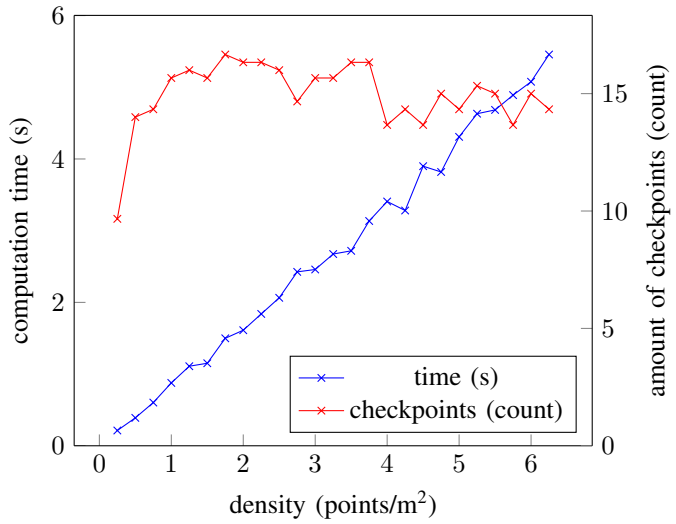

Fig. 11. Graph showing computation time and amount of computed checkpoints as seed density increases.

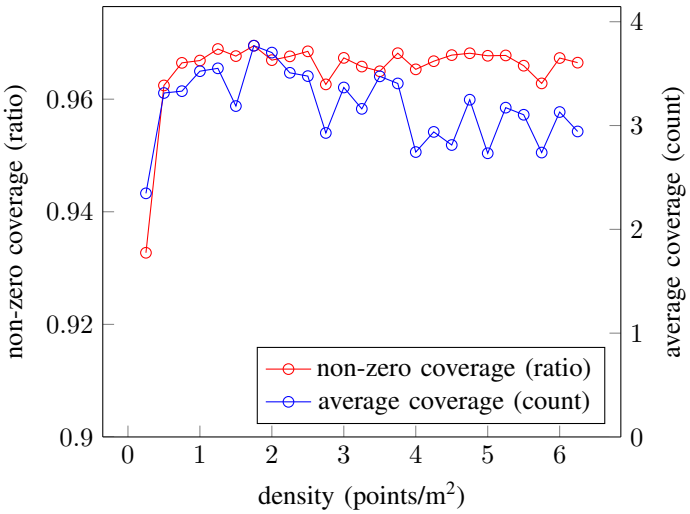

Fig. 12. Graph showing coverage metrics as seed density increases.

few seed points would result in a sub optimal checkpoint selection, possibly leading to unsearched areas and hence unfound objects. However computation time for the selection algorithm has an O(n) time complexity, where n is the seed density. This is shown by —×— in Figure 11.

The amount of checkpoints, shown by —×— in Figure 11, drops slightly as the density of points increases. This may be due to the convergence of the behaviour of an increasingly dense set of points to that of a field. As the density increases, the effects of noise in the random distribution of points is reduced and the resultant checkpoints converge to a single ”solution”. This then implies that the increased amount of checkpoints at lower seed density levels are a result of artefacts made more obvious by the increased effects of seed position noise at lower density levels.

In order to measure the coverage of the room provided by the checkpoints, two metrics are used:

- Non-zero coverage: the ratio of the amount of cells covered by at least one checkpoint to the total amount of cells within the room. Shown by —○— in Figure 12.
- Average coverage: the average amount of checkpoints covering any given cell in the room. Shown by —○— in Figure 12.

A cell is ”covered” by a checkpoint when it visible from it.

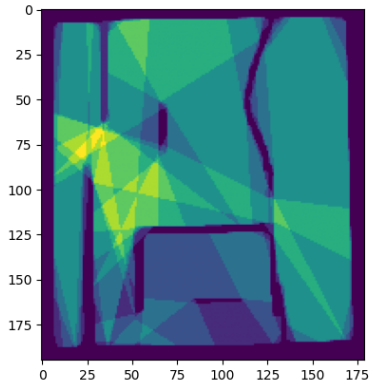

Fig. 13. Coverage map of the room, lighter cells have more coverage

Figure 13 shows a coverage map of the room. This map is used to calculate the coverage metrics.

During testing, the seed density was chosen empirically, by starting at a low density level and increasing until effective solutions were reliably found. The value of 2 seed points per square meter was chosen and is backed up by the data as this represents the peak of average coverage whilst keeping a low computation time.

The data for Figures 11 and 12 was collected by computing the checkpoints five times per seed density. The highest and lowest values for each metric were discarded and the mean average of the remainders was taken as the final value.

IV. Reflections

A. Improvements

1) Checkpoint Selection: One major improvement to the checkpoint selection algorithm would be to more locally normalise the density of points. At present, point density is only consistent over the whole room. There are two methods that would ensure density is constant over smaller areas:

- Relaxation of the points before filtering and clustering
- Placing points in square meter sections, ensuring density is constant

The latter of these two optimisations would be preferred as it would be less computationally expensive. However the former would ensure a more even point distribution, but this gain would not be worth the increased time requirement, especially in mobile applications.

An improved density consistency would result in fewer artefacts caused by gaps in seed point placement, such as that centring around point (25, 130) in Figure 5. This could reduce the overall seed density required to produce the the same quality of final checkpoints.

2) Object Detection: In order to improve the object detection logic, analysis of the shape of the objects would be the most effective improvement. This would result in fewer false detections and could be aided by the the use of the depth camera to further refine analysis of the shape.

B. Contributions

The author contributed all aspects of the system, apart from the object detection functions. This includes checkpoint selection, object localisation and approach, GUI and movement.