Robot Navigation for Object Localization

MEMBERS *

Tiago Trocoli - 226078 (Master's Student). E-mail: t226078@dac.unicamp.br
Jing Yang - 262891 (PhD. Student). E-mail:j262891@dac.unicamp.br
José Nascimento - 170862 (Undergraduate Student). E-mail:j170862@dac.unicamp.br
Camila Moura - 211845 (Master's Student). E-mail:c211845@dac.unicamp.br

***Abstract* – In this project, the robot needs to move to all predefined positions to localize plants in the scenario. To move to all positions only once and follow the shortest path, we ended up with the famous Traveling Salesman Problem, an NP-Hard problem. To overcome it, we relax our restriction and developed a path planning heuristic based on a shortest path algorithm, graph theory and robot behavior methods done on our previous projects. To have a better estimation of robot localization, we implemented the particle filter. To localize plants, the robot needs first to detect and classifies them, we handle this by using a machine learning algorithm. As a result, our project is capable of visualizing the environment and navigating a path based on the Bellman-Ford algorithm and the state machine, by applying deep learning, it is possible to detect objects and measure the distance between the frontal sonar sensor and the vase of the plant.**

***Keywords* – Robot Path Planning, Object Detection, Object Classification, Particle Filter.**

I. INTRODUCTION

This project concerns primarily with path planning and computer vision. The objective is make the robot go to all positions (nodes) of the map, possibly calculating the shortest paths, while obeying the graph topology, and also it must detects, classifies and estimates plants' positions and concurrently creating a map of the environment. By obeying a graph topology, we mean that the robot can only go to one node to another if they are connected by an edge.

In the path planning part, the robot needs to go to all predefined designated positions following specific path given by a graph. These paths are generated by an combinatorial optimization algorithm that calculates an optimal path in real time. While moving, the robot detects plants and takes photos of them, so it can estimates their positions on the map. The robot also generates a map and mark the plants on this map. To better track its positions on the scenario, the robot uses gyroscope and particle filter along with encoders.

An intelligent robot is expected to not only be able to navigate on the environment, but also to see and interact with it. Among all these abilities, object localization and recognition is fundamental and significant in that it is bringing enormous productivity to the society. For example, in a disaster scenario the robot can go to dangerous places searching for casualties. That what happened in Fukushima nuclear accident. Industrial robots are helping warehouses and supply chain doing laborious tasks once did by humans. Arm robots can detect defective products on the chain line increasing companies productivity.

Path planning is a common problem find in robots in which its objective is to find a path from one location to another, simultaneous avoiding obstacle and trying to find a shortest path, or at least, a "good" one. This task presents challenges that each wheeled mobile robot needs to overcome to become autonomous. The robot uses different kind of sensors to perceive the environment, such as ultrasonic, laser and vision that help it to make decisions while moving, notably, avoid obstacles. In the process of path planning, the robot's kinematic and dynamic constraints should be considered and also how it estimate its location.

Path planning can only be applied when a map of the environment is known [1]. There are basically two types of problems. In the optimum path, the goal is to find the optimum path between the initial and goal points. While complete coverage is to find the optimum path so the robot passes to all predefined positions. Our project deals with complete coverage, an NP-Hard problem know as Traveling Salesman Problem.

Tacking mobile objects in indoor environment requires sophisticated mathematical algorithms and this problem cannot be solved using GPS. They are based on probabilistic, linear algebra, markov chain and statistics analysis. They operate better by fusion data from many sensor, method called sensor fusion. In the literature, it is common to see Kalman Filter, the most famous method invented in 1960 by Rudolf E. Kalman [2]. It is a quadratic linear estimation motion model that is used today. Since then, many variations were developed in which the most notorious is Extended Kalman Filter and Distributed one. The first tracks using nonlinear motion models while the second is a distributed algorithm [3]. Markov and Particle filter was developed later and although is computationally more expensive than Kalman, they outperform Kalman filter in some scenarios [4].

Tracking mobile objects today is an application of Internet of Things (IoT). This application is known as Indoor Positioning System (IPS). It consists of devices that are employed to locate or track people and objects where GPS and other

*Contributions:
Tiago Trocoli: path planning for robot navigation and map building
Jing Yang: particle filter for robot localization
José Nascimento: object detection (ROI) and object localization (bounding box method)
Camila Moura: real time visualization, object detection (pre-trained network), gyroscope method and localization based in robot space (using sonar sensor).

satellite technologies fail to do so, such as inside buildings, airports, hospitals, parking garage, alleys, and underground metro station. A large number of technologies and devices can be used and integrated into this system, such as smartphones, smartwatch, WiFi, Zigbee, Bluetooth antennas and digital cameras. IPS has many applications in commercial, military, retail, and inventory tracking industries [5]. Examples, include tracking of products through manufacturing lines, operation of indoor unmanned vehicles, first-responder navegation, asset navegation or people-movers [6].

II. Path Planning Approach

The objective of path planning is to visit all nodes (positions) of a graph in a scenario, as show in figure 1. To do so, the path planning algorithm is based on a few steps. First, the robot decides which unvisited node it will go to. Then, it calculates the shortest path between its current node to the destination one. Therefore, the robot follows the shortest path marking each intermediate nodes as visited. After reaching its destination, it decides again which unvisited node it will go to, closing the loop. The algorithm stops when all nodes have been visited. The diagram is shown in figure 2. For example, if the robot is in node 0 (start) and decides to go to the unvisited node 4, it follows the shortest path $(0-2-3-4)$. Thus, the nodes 0, 2, 3 and 4 will be considered visited during its execution.

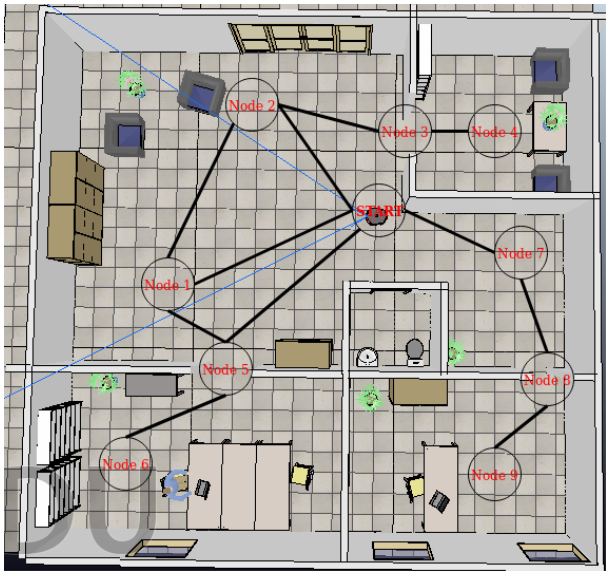

Figure 1. The scenario with a graph representing all positions the robot needs to visit following the edges.

The robot decides its destination based on the Euclidian distance of its current position and all unvisited nodes. The most distant of them is chosen to be its destination. Mathematically, let's suppose the robot is in position (x, y) and wants to decide its destination $\mathbf{x} = (x_d, y_d)$ from a set of unvisited nodes $U = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$, to do so, an algorithm executes the following math equation

$$\mathbf{x} = \underset{\mathbf{x}}{\operatorname{argmax}} \sqrt{(x - x_i)^2 + (y - y_i)^2}, \ \forall \ (x_i, y_i) \in U \quad (1)$$

Therefore, the robot must decide the minimum path from (x, y) to (x_d, y_d). It is a polynomial complexity problem, so it is possible to find a solution in polynomial time by applying any known shortest path graph algorithm in literature. We chose Bellman-Ford [7]. Algorithm 1 shows its implementation.

Algorithm 1 Bellman-Ford Algorithm

Require: list vertices, list edges, vertex source.
1: **create** distance and predecessor vectors.
{Step 1: initialize graph}
2: **for each** vertex v **in** vertices **do**
3: distance[v] = **inf**
4: predecessor[v] = **null**
5: **end for**
6: distance[source] = 0
{Step 2: relax edges repeatedly.}
7: **for** i from 1 **to** size(vertices)-1 **do**
8: **for each** edge (u, v) with weight w in edges **do**
9: **if** distance[u] + w < distance[v] **then**
10: distance[v] = distance[u] + w
11: predecessor[v] = u
12: **end if**
13: **end for**
14: **end for**
15: **return** distance[], predecessor[]

In which the distance vector is the minimum cost distance of a source to every node. Predecessor vector records the parent of each vertex, which is the parent in the shortest paths tree. By "cost", we mean the Euclidean distance of node to another.

After finding the minimum path, the robot moves to each node of the shortest path, until reaching its destination. To go to one node to another, it uses the state machine (Algorithm 2) that we implemented in project 2. Our state machine is a combination of goToGoal [8] and wall-following [9] behaviours.

Algorithm 2 State Machine Algorithm

Require: Current position (x_i, y_i) and destination (x_d, y_d)
1: stop = **false**
2: **while** $\sqrt{(x_i - x_d)^2 + (y_i - y_d)^2} > 0.1$ **do**
3: **Execute** goToGoal behavior.
4: stop = **false**
5: **while** if there's any wall near **or** stop == **false do**
6: **Execute** wall-following behavior.
7: $best_d$, stop = foundDistance($best_d$, (x_d, y_d))
8: **end while**
9: **end while**

In line 7, the function $foundDistance$ updates the shortest distance ($best_d$) from its destination that the robot has reached

so far during wall-following (line 6). If the robot reaches a shorter distance than its previous one, then *stop* becomes true and $best_d$ is updated. Thus, the robot executes goToGoal method. Otherwise, the robot is being forced to go to another direction because of an obstacle, so the code remains on the wall-following loop (lines 5-8)

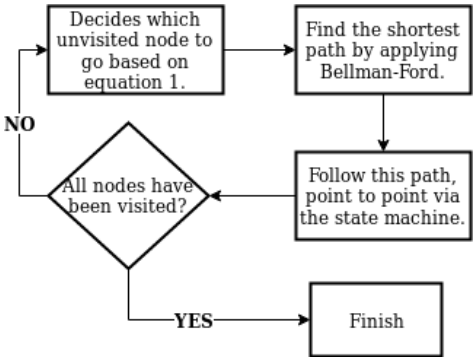

Figure 2. Diagram of Path Planning Method.

It worth mentioning that our ideal scenario of path planning is to make the robot follow the shortest possible path that visits each node only once and returns to the initial one. However, it is the famous Travelling Salesman Problem, an NP-Hard problem. So, we overcame with another approach. The robot could visit the same node more than once. It follows the shortest path of its current node to another unvisited one. And it does that until all nodes are visited. Of course, this heuristic doesn't find the overall shortest path, but at least find locals ones.

III. Object Detection

A. Region of Interest

A *Region of Interest* (ROI) is a user specifier rectangle that can be used to highlight a specific area of the image [10]. In this method, for each input image, the model generates a binary image, where the region of interest is white, and the other regions are black.

We used the ROI concept to train a neural network **to generate a ROI image detecting *indoor plants* in the images**. The methodology is depicted below.

1) Dataset: The first requirement for this task is a dataset with different images and its *ground truth*. The ground truth of an image is a user-generated binary image which contains the expected ROI retangle for the image. Example of ground truth is depicted in Figure 3 and 4. We have in this dataset 800 images depicting the indoor plant. For each image we have its real distance from the indoor plant and its equivalent user-generated ground truth.

Figure 3. Indoor Plant Photo

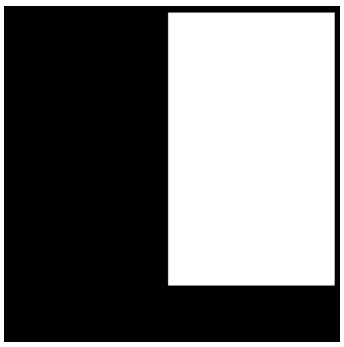

Figure 4. User-Generated Ground Truth of Figure 3

Using the robot, we acquired and labelled 800 indoor plant images *. The pictures followed the pattern below:

- Distances of 0.8, 0.9, 1, 1.1, 1.5, 1.6, 2.1, 2.5, 2.6, 3, 3.1, 3.5, 3.6, 4 and 4.1 meters.
- For each distance, we set the robot (in the simulator) to walk around the indoor plant, and to acquire images from different angles (keeping the same distance).

Plus, we defined that the region of interest should be the whole indoor plant, including both the vase and the plant. Otherwise, an empty vase could be identified as a plant.

2) VGG16-UNet: To train the model, we used the VGG16-UNet [11]. It is a fully convolutional network that does image segmentation. We implemented this neural network based on [12]. The architecture of the VGG16-UNet is depicted in Figure 5.

3) Training: To train the model, we used every image in the dataset, and its respective ground truth. We used 80% as training set and 20% as validation set. We trained the model twice: Firstly, we set 5 epochs for the training, and the model's accuracy was *0.9504*; Then, we set 200 epochs, and the model's accuracy was *0.9694*.

*Link to the dataset: https://drive.google.com/open?id=13qrefE4zj_RDfq302psH67T27n4v1gb8

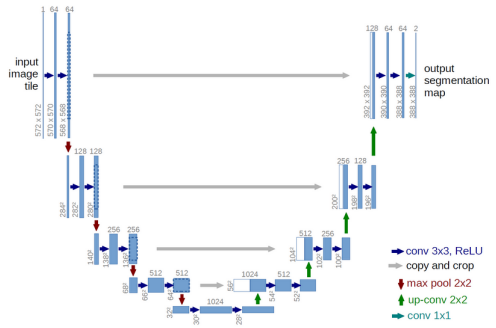

Figure 5. VGG-16 UNet Architeture [11]

B. Pre-trained Network

As a method of comparing our network, we developed a deep learning-based algorithm for object detection, using an SSD network based on MobileNet V1 architecture pre-trained on the MSCoco dataset which contains objects labeled in 90 different classes [13]. This network was implemented in TensorFlow and its output is the bounding box coordinates. With this, we were able to detect, in real-time, all objects present in the environment and draw a bounding box.

IV. Plant Position

A. Using Sonar Sensor

Based on the bounding box coordinates, we calculate the center of the object, and based on the position in which it is associated with its size, we can calculate which side the robot should rotate and the position it should move to approach the object, so that it remains centered on the image resulting from the visualization of the robot.

Using fuzzy technique, we select a vase of the plant as a target to direct the robot. We detected the plant continuously and direct the robot until the area occupied by the bounding box is up to 25% of the total image captured by the robot camera. Then it stops and calculates the distance between the robot and the inner plant by means of the frontal sonar sensor.

As long as we have (i) the distance between the object and the frontal sonar sensor, and (ii) an accurate pose (x, y, $theta$) of the robot, we can measure the global position of the plant by the coordinate transformation matrix.

B. Bounding Box Method

In addition, we implemented a method † in which the robot would be able to infer the indoor plant's distance by the bounding box shape.

1) Rate: Given an indoor plant photo and its ground truth, we acquire the following information.

- *Image Area* - In this case, as every image is 256x256 pixels, this value always is *65536*
- *Bounding Box Area* - In pixels (as the Image Area)
- *Image Centroid Point* - Point in image which is equivalent to the vertically lowest point and horizontally the mean point. As every image is 256x256 pixels, this value is (256,128.5). See Figure 6.
- *Bounding Box Centroid Point* - See Figure 7.

Figure 6. Indoor Plant Photo - Red indicates the Centroid Point Location.

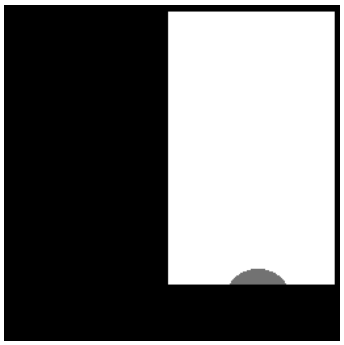

Figure 7. Ground Truth of Figure 6 - Gray indicates the Centroid Point Location.

Firstly, we calculate the *extent ratio* of image area to bounding box area as:

$$Extent = \frac{image\ area}{bounding\ box\ area} \quad (2)$$

Then, we calculate the *distance* between both centroid points:

$$D = dist(image\ centroid, bounding\ box\ centroid) \quad (3)$$

Finally, we calculate the *rate* as:

$$Rate = Extent * D \quad (4)$$

2) Distance Prediction: It is expected that (i) images which similar indoor plant distance have similar rate, and that (ii) the rates have some proportion among themselves. Hence, as we have the distance information for every image in the dataset with 800 images, we implemented both linear regression and polynomial regression models to estimate the distance of new photos.

†We borrowed the basic idea of this method from a colleague. The paper which concerns it is still in development.

V. Localization Improvement

In Robotics, localization aims to infer the location of the robot in its environment. To update its location, the robot usually needs sensors to measure its motion and the environment. Odometry is one common way to achieve localization. However, this method requires initial location of the robot, which is sometimes unknown, such as a kidnapped robot. In addition, odometry becomes very inaccurate after a few iterations because of accumulative errors. To overcome the drawbacks of robot odometry, there are several localization methods, including histogram filter, kalman filter and particle filter. Among them, particle filter is one of the most efficient and scalable solutions.

In our project, we tried particle filter to improve the localization. Particle filter is based on Monte Carlo methods, which manage to handle non gaussian problems by discretizing original data into particles(each of them representing a possible state of the robot). The general algorithm of particle filter is described as follows:

Algorithm 3 Particle Filter Algorithm

Require: Landmarks M, Robot motions U

1: Initialize N particles
2: **for** i from 1 to size(U) **do**
3: robot **Execute** motion U_i
4: Z = robot sensor observation
5: **for** k from 1 to N, for every particle **do**
6: **Execute** motion U_i + certain noise
7: compute distance from particle position to landmark positions
8: calculate particle weights by measuring the distance and observation Z
9: resample particles based on their weights
10: **end for**
11: **end for**

Figure 8 depicts the main process of particle filter. First the particles are sampled following a certain distribution, then based on the measurements of robot sensors, particles are assigned with different weights, the higher the weight, the closer the particle is to the robot. Thus, particles are resampled based on their weights.

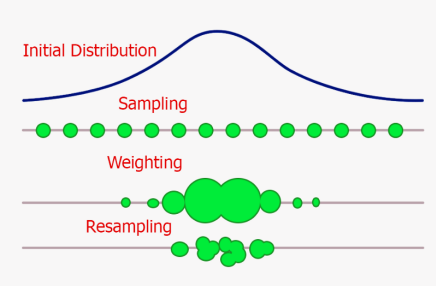

Figure 8. particle filter illustration example[14]

In algorithm 3, the resampling algorithm in line 9 is described in the following:

Algorithm 4 Particle Resampling Algorithm

Require: Particle weights w

1: Generate a random index i from uniform distribution ranging from 0-N
2: Initialize $beta$ = 0.0
3: **for** k from 1 to N, for every particle **do**
4: Generate a random number r from uniform distribution ranging from 0-maximum particle weight
5: $beta$ = $beta$ + r
6: **while** $beta$ is greater than $w[i]$ **do**
7: $beta$ = $beta$ - $w[i]$
8: i =(i + 1) mod N
9: **end while**
10: obtain sample particle i
11: **end for**

The major difficulty of particle filter in our project is lack of observation data obtained from sensors. To localize the robot, we first need to acquire the map, then collect sensor information while the robot is moving around the environment. In the path planning, we choose to make the robot follow the wall of its left side. After collection of robot motions (which are the left and right wheel radius) and sensor information along with each motion, we apply particle filter to estimate robot location. However, we did not achieve feature matching for the sensor data and the map. There are several ways to tackle this: 2D scan matching[15], deep learning based matching[16], etc.

VI. Experiments and Results

A. Path Planning

In the experiment the graph topology is given in figure 1. The robot indeed visited all positions. The robot follows the paths below, these paths are optimal cost by Bellman-Ford. A video showing the robot path can be found in [‡]

$0 \rightarrow 2 \rightarrow 3 \rightarrow 4$
$4 \rightarrow 3 \rightarrow 2 \rightarrow 0 \rightarrow 8 \rightarrow 9$
$9 \rightarrow 8 \rightarrow 7 \rightarrow 0 \rightarrow 5 \rightarrow 6$
$6 \rightarrow 5 \rightarrow 1$

The map data collected by laser scanner through the robot path is shown in Figure 9

B. Region of Interest

Even though the good accuracy presented in the training, the method usually did not performed good results. Examples of outputs are Figure 10 and 11. At least two improvements could be researched in future works in order to (hypothetically) increase the accuracy:

[‡]https://drive.google.com/open?id=1GUcv4zkem2GDr2UgBpVgDAESiRZ531LD

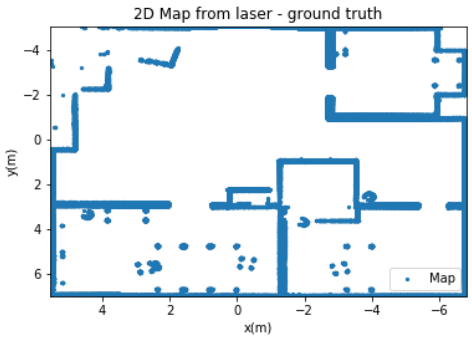

Figure 9. Map created by laser scanner through robot path

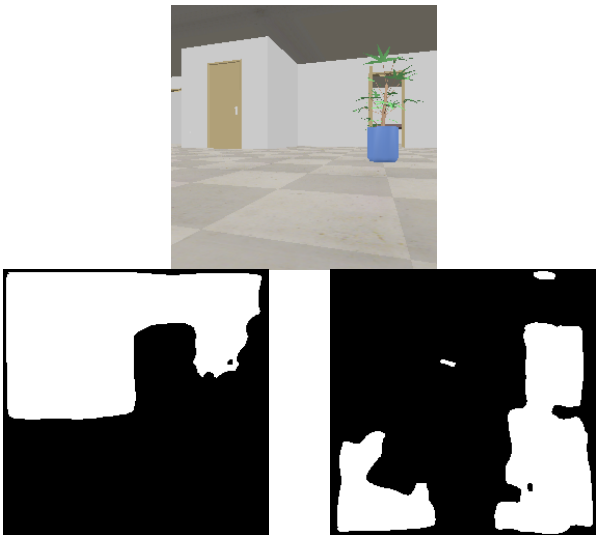

Figure 10. Predict Ground Truth (5 and 200 epochs, respectively)

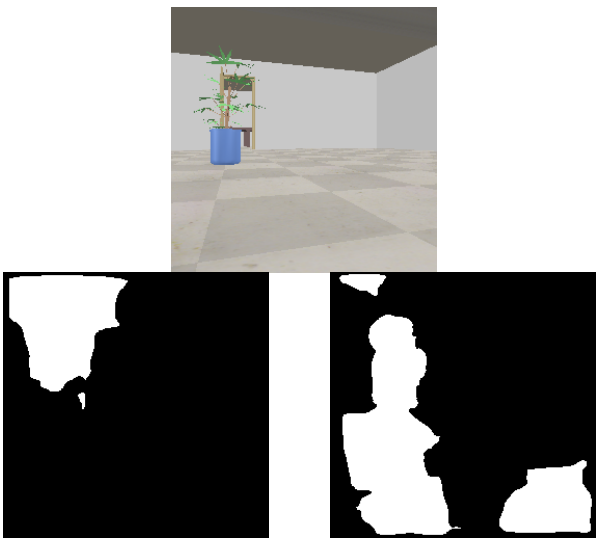

Figure 11. Predict Ground Truth (5 and 200 epochs, respectively).

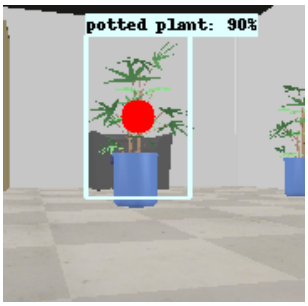

Figure 12. Object detection with label.

Figure 13. Vision of robot to decided when calculate distance.

- *Acquiring more photos*: 800 may be a small number of photos for training a neural network, even in a simulator context.
- *Acquiring random photos*: The photo's acquirement process was based in specific distances and angles. Making this process random would increase the variability of images in the training set, hence (theoretically) improving the method.

C. Object Detection and Plant Position by the Sonar Sensor

We implemented the camera view on our robot, where we can detect objects in the scene and label them in real time, as shown in Fig 12. As we obtained good accuracy in this function, we incremented the method by defining an object of interest and orienting the robot to follow it. it and avoid obstacles in its path. In Fig 13, we can observe the robot's view as it moves towards a plant, as well as identify the bounding box around the vase and the center of the object, circle in red.

As a method of calculating the distance, related in the Subsession IV-A, the robot stops in front of the vase and printed its distance. For the case, presented in Fig 13 this happened at a distance of 0.51 cm. Then, the robot is able to infer the global position of the plant using Local-to-Global transformation matrix. After that, the robot continues its path, until find another plant. Examples of this methods can be seen in [§]

[§]https://drive.google.com/open?id=1OztAkR7L0x_SgwaujcpT8o7PbBN_x98T
https://drive.google.com/open?id=1MfBPVgZiakzVQ8POBWblcRbRIIgQbuD6
https://drive.google.com/open?id=1ymYMa63_-cfExl3K3KT8zgTDPblrc87E

Table I
LINEAR REGRESSION

Linear Regression (Ax + B)				
Images	A	B	MSE	Variance
All	-4.15	3.37	0.21	0.80
Centered	-3.36	3.32	0.21	0.81

Table II
POLYNOMIAL REGRESSION

Polynomial Regression (Ax^2 + Bx+C)					
Images	A	B	C	MSE	R^2
All	6.93	-9.44	3.96	0.06	0.94
Centered	7.76	-10.07	4.00	0.07	0.94

D. Plant Position by the Bounding Box Method

After inspecting the rate values, we concluded that (i) the rate could be approached by both linear and quadratic functions; (ii) the error is larger when the indoor plant is not in the center of the image; and (iii) the error is considerably larger when the indoor plant is located 1 meter or closer from the robot. One of the reasons of it is that if the plant is closer, it is more likely that some part of it is not in the picture;

To estimate the distance from the robot to the indoor plant, we tested both linear and polynomial regression (quadratic function) in two scenarios: (i) the dataset contained all images and (ii) the dataset contained only images which indoor plant is closer to the center (henceforth called *centered images*). We used 80% of the dataset as training set, and the remaining 20% as test. Finally, we decided in this initial test to use the User-Generated Ground Truth to measure the bounding box proprieties. We depict the results for Linear Regression in Table I and Pictures 14 and 15, and we depict the results for Polynomial Regression in Table II and Pictures 16 and 17.

In order to determine whether the image is a centered image, we first determined the center of the bounding box in the mask. For each line in axis X, we measured the mid point between the first and the last white pixel (if there is at least one white pixel, otherwise this line is not considered). The mid X is the mean of every mid point calculated. Then we calculated an analogous method for Y axis, and we obtained the center of the bounding box (X_b, Y_b). Finally, we decided that a bounding box is close to the center if Y_b is between 115 and 142 (as the center of the image is 128.5). Among the 800 images, 260 were categorized as centered images.

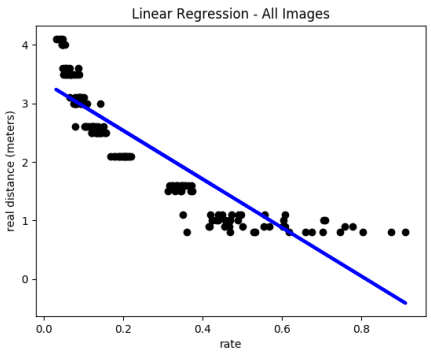

Figure 14. Linear Regression - All Images

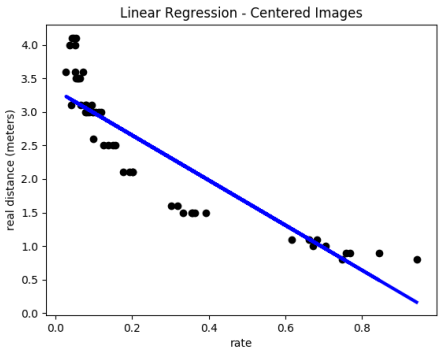

Figure 15. Linear Regression - Centered Images

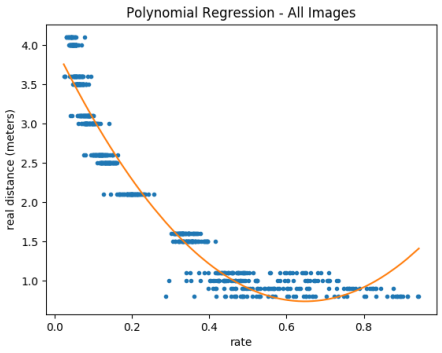

Figure 16. Polinomial Regression - All Images

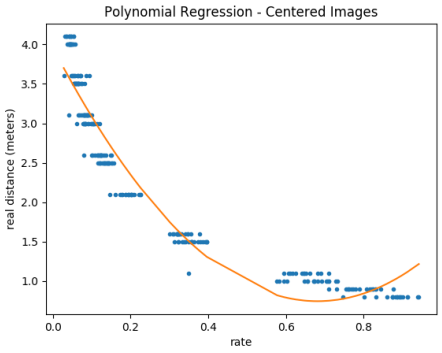

Figure 17. Polynomial Regression - Centered Images

The accuracy for polynomial regression would be enough for the problem, but this work lacks an experiment with a larger and more variate dataset. Plus, we estimate that this

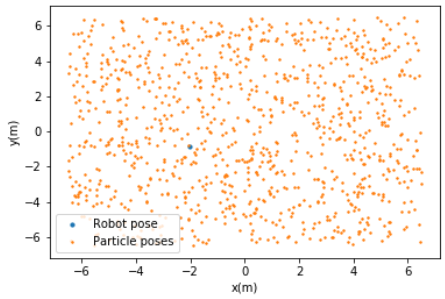

Figure 18. particles initial states

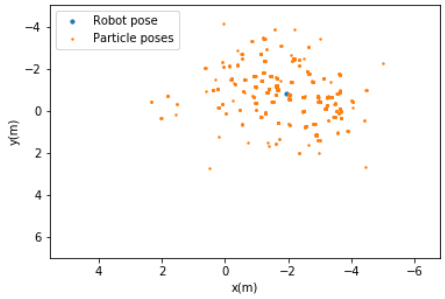

Figure 19. particles states after 4 iterations

accuracy may be improved if an additional method to infer the angle of the object in the camera referential given an image is developed.

E. Particle filter localization

In this part, we will show some results of particle filter localization. In the experiments, we assume the noise of wheel encoder reading follows gaussian noise. As the robot moves along the planed path, we collect its wheel encoder data every 0.5s, and calculate the difference of radius in every time interval. Then we compute robot odometry adding noise. Assuming the noise of radius difference for two wheels follows zero mean gaussian distribution, and their variations are both 0.1.

In this experiment, for the sensor measurements, we are not using actually any sensor, instead we just calculate the distances between noisy odometry position and landmarks. We assume the noise distribution of sensing measurement is N(0, 3).

The number of particles in the experiment is 1000. The initial locations of particles and robot is shown in Figure 18. After 2 seconds (4 iterations), the distribution of particles is shown in Figure 19, after 120 iterations, the distribution is shown in Figure 20.

From Figure 18 - Figure 20, we can see that the particles are congregating around robot location very fast. The paths of ground truth, odometry and particle filter after 120 iterations are shown in Figure 21

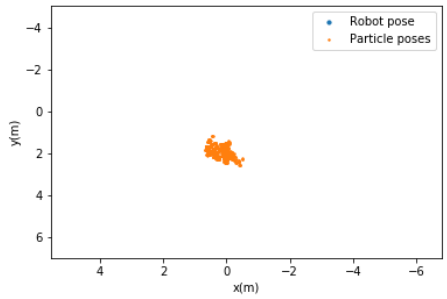

Figure 20. particles states after 120 iterations

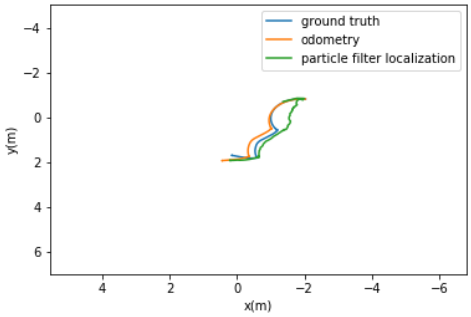

Figure 21. paths of ground truth, odometry and particle filter after 120 iterations

As we can see from the Figure 22, particle filters can achieve very good results in a short time. However, it is only based on odometry without any other sensors, which results in accumulated errors similar to original odometry. Furthermore, it is only heuristic to assume the distribution and parameters of the noise model of the odometry, which may not be accurate. Finally, we conclude that even with particle filter, with only odometry data is not enough for robot localization.

VII. Conclusion

This project designed a robot navigation path based on Bellman-Ford algorithm and state machine. However, both

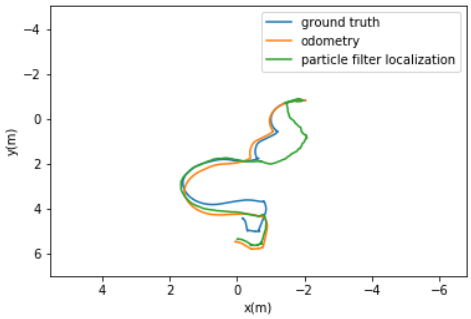

Figure 22. paths of ground truth, odometry and particle filter after 300 iterations

object detection and robot localization presented problems. In object detection, the model's ground truth may present errors far from an acceptable result. Meanwhile, particle filter accumulated errors similar to original odometry. To detect the position of the object, we still need to rely on a pre-trained network, as even though the results for the bounding box method are promising, it lacks tests with a larger dataset.

Finally, this work lacks an implementation of a behavior which uses the methods developed here altogether.

REFERENCES

[1] G. Klancar, A. Zdesar, S. Blazic, and I. Skrjanc, *Wheeled mobile robotics: from fundamentals towards autonomous systems*. Elsevier, 2017. i

[2] "Kalman filter," https://en.wikipedia.org/wiki/Kalman_filter, accessed: 2019-09-30. i

[3] S. Shen, C. Xia, R. Sprick, L. C. Perez, and S. Goddard, "Comparison of three kalman filters for an indoor passive tracking system," in *2007 IEEE International Conference on Electro/Information Technology*. IEEE, 2007, pp. 284–289. i

[4] N. Y. Ko and T. G. Kim, "Comparison of kalman filter and particle filter used for localization of an underwater vehicle," in *2012 9th international conference on ubiquitous robots and ambient intelligence (URAI)*. IEEE, 2012, pp. 350–352. i

[5] Wikipedia contributors, "Indoor positioning system," 2019, [Online; accessed 10-September-2019]. [Online]. Available: https://en.wikipedia.org/wiki/Indoor_positioning_system ii

[6] D. Dardari, P. Closas, and P. M. Djurić, "Indoor tracking: Theory, methods, and technologies," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 4, pp. 1263–1278, 2015. ii

[7] A. Goldberg and T. Radzik, "A heuristic improvement of the bellman-ford algorithm," STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, Tech. Rep., 1993. ii

[8] "Robot navigation," https://medium.com/@aniket0112/navigation-of-mobile-robots-351648def6cf, accessed: 2019-11-25. ii

[9] R. Braunstingl, P. Sanz, and J. M. Ezkerra, "Fuzzy logic wall following of a mobile robot based on the concept of general perception," in *Procs. of the 7th Int. Conf. on Advanced Robotics (ICAR'95)*, 1995, pp. 367–376. ii

[10] R. Brinkmann, *The Art and Science of Digital Compositing*. Morgan Kaufmann, 1999. iii

[11] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. iii, iv

[12] "Unet segmentation," https://github.com/nikhilroxtomar/UNet-Segmentation-in-Keras-TensorFlow/blob/master/unet-segmentation.ipynb, accessed: 2019-11-27. iii

[13] "Single shot detection," https://github.com/osrf/tensorflow_object_detector/, accessed: 2019-11-25. iv

[14] "Optimal estimation algorithms: Kalman and particle filters," https://pierpaolo28.github.io/blog/blog26/, accessed: 2019-11-25. v

[15] K. Yoneda, C. Yang, S. Mita, T. Okuya, and K. Muto, "Urban road localization by using multiple layer map matching and line segment matching," in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 525–530. v

[16] W. Lu, Y. Zhou, G. Wan, S. Hou, and S. Song, "L3-net: Towards learning based lidar localization for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6389–6398. v