Dynamic Parameter Update for Robot Navigation Systems through Unsupervised Environmental Situational Analysis

Amirhossein Shantia*, Francesco Bidoia† , Lambert Schomaker‡ and Marco Wiering§
Institute of Artificial Intelligence and
Cognitive Engineering, University of Groningen
Groningen, The Netherlands
Email: *a.shantia@rug.nl, †f.bidoia@student.rug.nl, ‡l.r.b.schomaker@rug.nl, §m.a.wiering@rug.nl

Abstract— **A robot's local navigation is often done through forward simulation of robot velocities and measuring the possible trajectories against safety, distance to the final goal and the generated path of a global path planner. Then, the computed velocities vector for the winning trajectory is executed on the robot. This process is done continuously through the whole navigation process and requires an extensive amount of processing. This only allows for a very limited sampling space. In this paper, we propose a novel approach to automatically detect the type of surrounding environment based on navigation complexity using unsupervised clustering, and limit the local controller's sampling space. The experimental results in 3D simulation and using a real mobile robot show that we can increase the navigation performance by at least thirty percent while reducing the number of failures due to collision or lack of sampling.**

I. Introduction

The use of autonomous robots in our daily lives are on the rise. From autonomous drones delivering packages [1], mobile security robots [2], autonomous vehicles [3] to domestic robots that monitor elderly and help them in their activity of daily living [4]. The first and foremost responsibility of all these robotic systems is to navigate safely and efficiently in their environments. The navigation stack usually consists of a global localization module and path planner, a local base controller, and a set of sensor processing systems. The global localization is usually done through a mapping and localization process [5]. When a map is made, these robots estimate their position based on the erroneous movement of the base and precise sensory readings using probabilistic methods such as adaptive Monte Carlo localization (AMCL) [6]. At this point, the only remaining task is to calculate a set of velocities to make sure that the robot reaches its goal safely. For this process, generally a two or three dimensional cost map is made. In the case of three dimensions, all sensor readings, such as rotating lasers, infrared devices, and sonars are added to a three dimensional pointcloud structure called voxel/octo map [7]. For a two dimensional cost map, all sensors information is projected down into a two dimensional array. In each control cycle, the system has to mark or clear these cells using ray tracing techniques [8]. Finally, the robot searches for the best local trajectory by forward simulating a set of velocities over time using discrete sampling, taking into account all the obstacles on the way and the robot footprint and shape. A set of velocities that helps the robot to reach the goal safely will be selected. This process is done multiple times a second to achieve a required control frequency for the robot. This requirement can be different depending on the type and application of the robot. All the parts of the navigation stack have parameters to be tuned. Precision of the cost maps, number of the particles in AMCL, velocity sample rate, simulation time, scoring parameters, etc. It is important to note that these parameters have significant effects on the processing load. Therefore, one can conclude that it is hard to select one set of parameters for all navigation tasks that may differ in complexity. For example, in cluttered environments, tight corners, or doors a higher resolution cost map and velocity sample rate will help the robot to navigate more safely and efficiently, while in larger hallways, the system can use faster speeds and a lower number of samples.

The first two authors contributed equally to this paper.

Contributions: In this paper, we propose a novel approach to automatically identify the complexity of scenes by extracting a customized histogram of oriented gradients (HoG) [9] from a two dimensional projection of sensory readings (cost maps), and clustering them using multiple unsupervised methods such as K-means and agglomerative clustering. In addition, we identify a tuned set of parameters for each of these clusters. Our experiments show that the clustering methods successfully separate the situations into meaningful and human understandable clusters. Therefore, our contributions can be summarized as follows:

- Automatic navigational complexity classification
- Dynamic parameter update for the local navigation system and the cost maps
- Performance increase of the navigation system

Structure of the paper: In section II we describe the full navigation system in detail. In section III we present the customized HoG feature extractor, the used clustering methods, and our approach to extract the best possible parameters for the navigation system. We describe the experiments and the

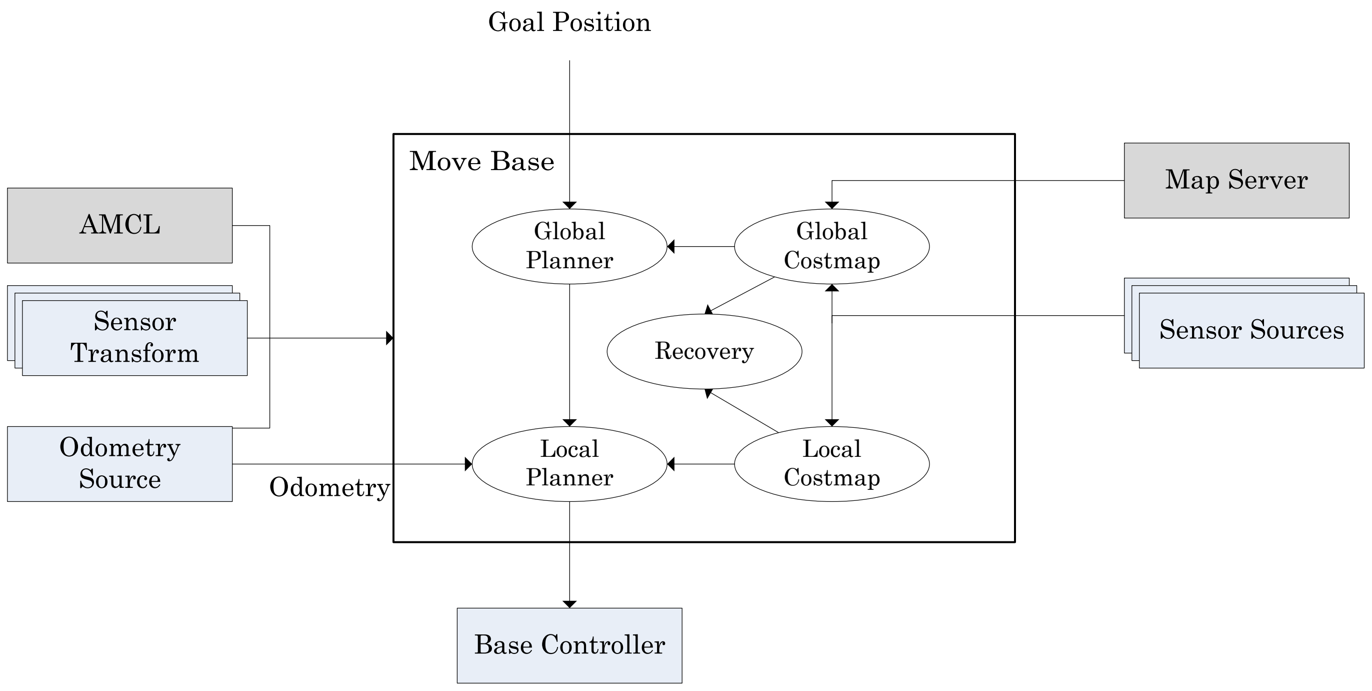


Fig. 1. The structure of the navigation system.

results in section IV. Finally, we conclude the paper in section V and discuss possible future work.

II. Preliminaries

Before continuing with the methodology section, we would like to depict the navigational structure that we use in detail. The used navigational stack is based on the work by David Lu [10]. This work was implemented as a package for the Robot Operating System (ROS) [11]. The general structure of the navigation stack is depicted in Figure 1. This stack requires certain inputs to be able to function correctly. The inputs outside the rectangle in Figure 1 require:

- A pre-generated 2D map of the environment.
- A localization method, in our case AMCL.
- Odometry information
- Base velocity control
- Sensor sources
- A complete transformation function to relate all the robot links in real-time.

Having a pre-generated map, we can use the AMCL method to localize and track the position of the robot. When a destination goal is sent to move base, the global planner will attempt to find a path towards the selected goal using either the $A^\star$ [12], or Dijkstra's shortest path [13] algorithm. The calculated path is then sent to the local planner that will use the projected cost map to find the best set of velocities to approximately follow the global trajectory to reach the goal. In case of failures due to incorrect sensor readings or possible deadlocks, the system can initiate recovery behaviours. In this paper, our optimization is done using the dynamic window approach local planner [14].

A. *Dynamic Window Approach*

We selected the dynamic window approach (DWA) because it is used mostly in robots with good acceleration rates, and has a higher performance due to the window approach in comparison to other methods such as the trajectory roll out planner which simulates the entire path [15]. The DWA needs the local cost map, a window section of the calculated global path, the projected footprint of the robot, robot base capabilities, and a set of simulation parameters. The robot base capabilities are the achievable acceleration and velocities in X, Y, and θ directions. In Table I, the important simulation parameters are described. We omitted parameters with low importance and the ones that we do not optimize[1]. If all the above requirements are fulfilled, the robot navigates safely and reliably. However, different environmental circumstances require different parameters for optimal performance. Simulating unnecessary velocity trajectories in an empty hallway is a waste of resources, and lack of this sampling can be dangerous in crowded and narrow hallways. In section III, we explain how we solve this problem by applying an unsupervised situation detector.

III. Methodology

In this section we present our novel approach for autonomous situation analysis of the environment and a method to select suitable parameters for each situation.

[1]The full list of parameters can be found in http://wiki.ros.org/dwa_local_planner.

TABLE I
DWA PARAMETER STRUCTURE.

Category	Parameter	Description
Robot Configuration	Acceleration Limits Velocity Limits	Rotational and translational acceleration in m/s^2. Rotational and translational speed in m/s.
Goal Tolerance	Yaw Tolerance X-Y Tolerance Latch X-Y	Yaw threshold for the goal in radians. X and Y distance threshold to the goal in meters. If True, after position is reached, only rotation will be checked.
Forward Simulation	Simulation time and Granularity Sampling rates of X-Y-Theta Controller Frequency	Simulation resolution (meters) and length of time (seconds). The number of velocity samples for simulation, limited by robot configuration. Planner's desired loop for driving
Trajectory Scoring	Path Distance Goal Distance Collision Distance	Higher score if the simulated path is close to global path. Higher score if the simulated path is closer to the local goal. Lower score if the simulated path is close to obstacles.

A. *Unsupervised Environmental Situation Analysis*

The cost map is a representation of the surrounding environment of the robot. Therefore, we believe that we can find certain patterns in these costmaps and classify different situations on this basis. The first step is to collect data points. This is done through moving the robot through the environment several times. The next step is to come up with a feature set that can help us in the classification procedure.

1) Customized Histograms of Oriented Gradients: The Histogram of oriented gradients (HoG) [9] is a suitable method to represent the structure in images. It has been used extensively in both two and three dimensional data. The three dimensional case is called the unique signatures of histograms (SHOT) [16]. The two dimensional case (HoG) has been applied to human detection [9], indoor localization [17], object recognition [16], and many other applications. Therefore, we believe that it is also a suitable method for our application. The local cost map is centered on the robot. However, this cost map is invariant to robot rotation, therefore, we first calculate the rotation with a translation matrix using equation 1.

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1-\alpha)\cdot center.y \end{bmatrix} \quad (1)$$

where $center.y$ and $center.x$ are the center of the cost map and

$$\alpha = \cos(angle)$$
$$\beta = \sin(angle)$$

Then, we apply an affine transform to the cost map matrix using the calculated rotation matrix. We lose a part of the image during these operations, but since this information lies on the corners of the cost map, the effect is minimal and can be neglected. Note that the center of the image is not necessarily the center of the robot. It is the main rotation axis of the robot. In addition, we also altered the windowing mechanism of the HoG feature set. We changed the window approach to emphasize this characteristic. Figure 2 shows the new window selection.

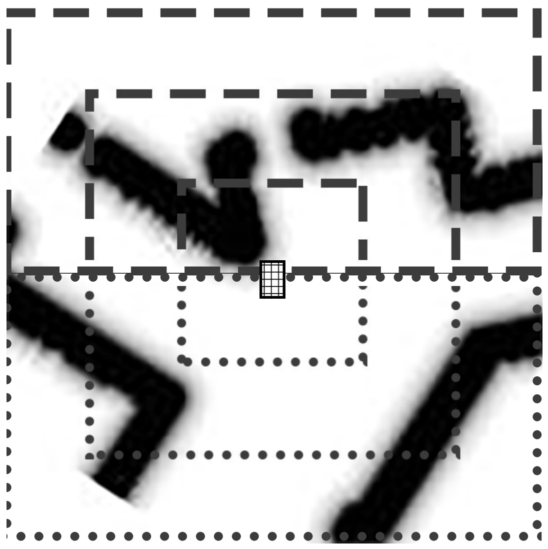


Fig. 2. The window structure of the HoG. The underlying picture is a sample projected cost map. The middle rectangle is the robot footprint and with its rotational axis point in the middle of the image. Each rectangular section is used to calculate the HoG features. This way, the resulting descriptors are more suitable for our problem.

2) Clustering and Center Selection: We use K-means clustering [18] to separate the data into multiple clusters. The problem is to select a good value for the number of clusters. We perform agglomerative hierarchical clustering using the single linkage [19] method to find the best number of clusters which can be seen in Figure 3.

Algorithm 1 Agglomerative Hierarchical Clustering using Single Linkage

```
1: begin initialize $c, \hat{c} \leftarrow n, D_i \leftarrow \mathbf{x}_i, i = 1, \ldots, n$
2: repeat
3:     $\hat{c} \leftarrow \hat{c} - 1$
4:     $find nearest clusters, D_i \ and \ D_j$
5:     $used_{min}(D_i, D_j) = \min_{x \in D_i, x' \in D_j} \|x - x'\|$
6:     $merge \ D_i, and \ D_j$
7: until $c = \hat{c}$
8: end
```

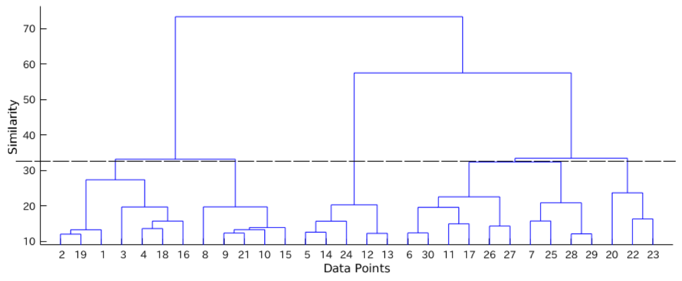


Fig. 3. The dendogram constructed from the single linkage algorithm. There is a large similarity gap for all the joint clusters between $c = 6$ and $c = 5$, and also another gap between $c = 4$, and $c = 3$. By inspecting the average image of each cluster number, we found out that five clusters are the most intuitive number for the k-means clustering.

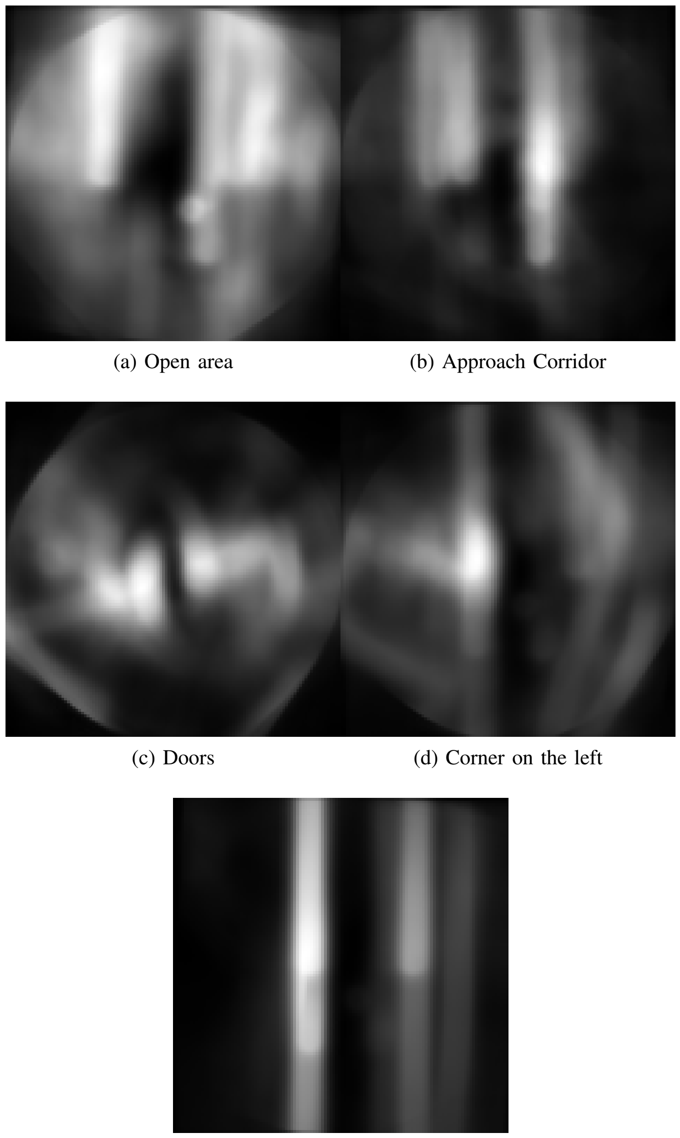
We repeat the above procedure for different numbers of clusters and draw the dendrogram based on the single linkage result which can be seen in Figure 3. Finally, we analyze the similarity values, and select a number of clusters which has a large similarity gap with the next number of clusters. However, since this step requires human observation, it is best to test different clustering numbers and compare the final experiment results. Due to the large required number of experiments, we only observed the members of each cluster for cluster numbers between 3 to 6. With five clusters, the results were most intuitive which can be seen in Figure 4. It is also notable that the current dendrogram uses a binary approach to separate clusters. However, a study by Louis Vuurpijl *et al.* [20] shows that the structure of the underlying data can hinder the effectiveness of a binary approach. Instead, they propose an N-ary approach by comparing the distance of the child clusters to the parent cluster considering the standard deviation between all the child clusters and their respective parent. We left this method for future work.

B. Parameter Selection

When the clustering is finished, we can analyze the results by means of an average image for each cluster. The procedure is to compare the distance between each recorded cost map image to the center of the cluster. Using an exponential function, we give higher weights to the closer points, and lower weights to the points far from the center. The final result is an average image which describes the surrounding environment (Figure 4). Using these images, we arrange the clusters based on danger level. This parameter selection, however, is dependent on the type of the robot used. The requirement is to have one safe parameter setting with a low maximum speed and high simulation sampling, and a fast parameter set with a higher maximum speed and lower sampling rate of velocities. From these two parameters, we can extrapolate the rest of the parameters for clusters based on their danger level.

C. Parameter Update

Every time the cost map is updated, our algorithm analyses and determines the environmental situation. Before proceeding to update the optimal navigation parameters, we make sure that the analysis is correct and coherent. This is done by accepting only results which are coherent over three consecutive cost map updates. This also prevents continuous parameter updates due to outliers which leads to latency in the navigation control loop. For the actual update of the navigation parameters, the method sends the chosen parameters to the navigation stack by calling a specific ROS service. This service updates the parameters in the interval between the navigation control loops. This generates a latency that could lead to the control loop missing the desired control frequency, causing the robot to stop.

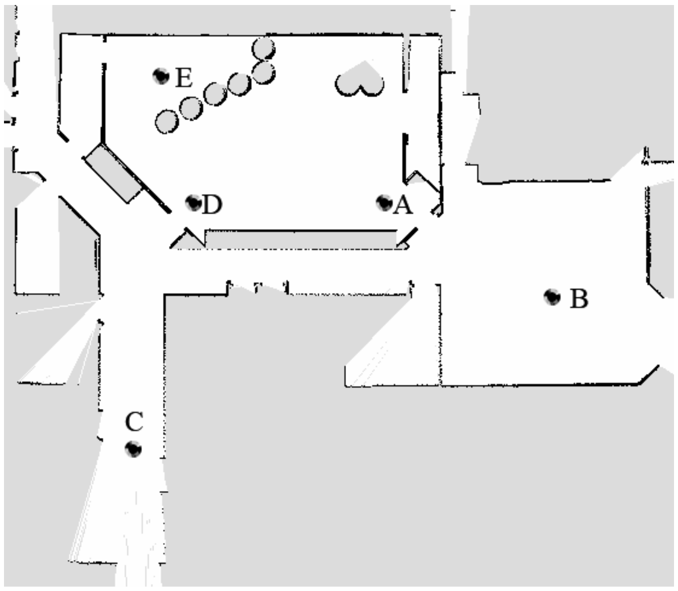


(a) Open area (b) Approach Corridor

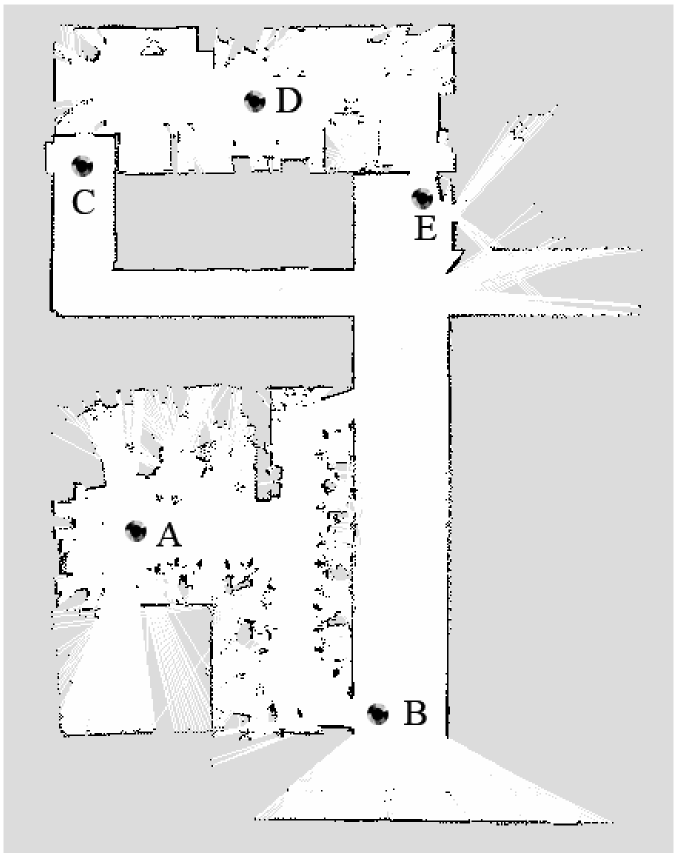
(c) Doors (d) Corner on the left

(e) Hallway

Fig. 4. The weighted average image of all the clusters. Black pixels mean free space, white pixels means obstacles. The intensity level of white pixels show how close they are to the cluster center. (a) shows low congested areas and open spaces, (b) shows that the robot is approaching corridors, (c) shows very dense areas such as doors, (d) shows close proximity to corners, specially on the left side, and (e) shows hallways.



(a) Gazebo Environment



(b) Real Environment

Fig. 5. The map of the environment in Gazebo (a) and for the real experiments (b). The robot starts from checkpoint A, and continues through all the checkpoints. The robot collects the time data when it reaches checkpoint E. If a failure or collision occurs, a penalty of one thousand seconds is recorded, and the experiment is restarted.

IV. Experiments

For the experiments, we use both a 3D simulator and a real robot. The 3D simulator used is an open source program called Gazebo [21] which simulates physics and allows us to have access to sensors, actuators, etc. This simulator is selected because of its wide spread use in robotics, such as the DARPA robotic challenges [2]. The robot is controlled through the robot operating system (ROS) framework [11]. The environment used for the simulation and real life can be seen in Figure 5. We use OpenCV, MATLAB and Numpy libraries for feature extraction and clustering [22], [23],[24].

The robot in Figure 6 is used for our experiments. The robot uses a differential drive base, and a frame which carries the manipulator, two RGB and depth sensors, and a laser range finder. We modeled this robot in Gazebo using the Unified Robot Description Format (URDF), which is an XML format for representing a robot model.

A. Clustering Results

We gathered nine thousand cost map data points, and calculated the customized HoG features from it. A single linkage was done on this set, and we extracted a dendrogram based on the distance values which can be seen in Figure 3. The best number of clusters is 5, since the similarity distance is large between this level and the previous one. We used this number for our K-means clustering method, and Figure 4 shows the average images of these clusters. Hallways, doors, and the situation when approaching doors are evident in these images. When a cluster is detected, the navigation parameters will be changed on the fly.

B. Base Parameter Selection

In order to select the best base parameters, we first hand tuned the values to match the robot's differential drive base capabilities. Then, we sampled exploration values for each of the DWA parameters in Table I. We calculated all possible combinations of these values, and compared the time it took to reach a point. The timeout to reach a point is 200 seconds, if the robot collides with an object or cannot reach the goal in time, it receives a penalty of 1000 and the trial is reset. For each combination, we simulate 100 rounds. The best selected base parameters are depicted in Table II.

[2]http://www.darpa.mil/program/darpa-robotics-challenge

TABLE II

BEST CALCULATED PARAMETERS FOR THE DWA METHOD.

Category	Parameter	Value
Robot Config	Acceleration X-θ Velocity X-θ	$1.0 m/s^2$, $1.0 rad/s^2$ $0.4 m/s$, $1.0 m/s$
Forward Simulation	Simulation granularity, time Sampling rate X-θ Controller Frequency	0.05m, 3.5s 15, 45 5(Hz)
Trajectory	Path Distance Goal Distance Collision Distance	0.2 0.3 0.05

TABLE III

THE SIMULATION RESULTS. ON AVERAGE THE DYNAMIC PARAMETER SETS PERFORM 33.8% BETTER THAN THE BEST STATIC PARAMETER, AND A HIGH SPEED SET. IN ADDITION, THE NUMBER OF FAILURES IS ALSO LOWER THAN THAT OF THE REST.

Param Type	Mean (without Penalty)	Standard Dev.	Failures
Dynamic	**56.32 (37.37) s**	19.22 s	10
Best Static (Safe)	85.08 (60.76) s	30.36 s	13
Fast	89.29 (61.44) s	28 s	15

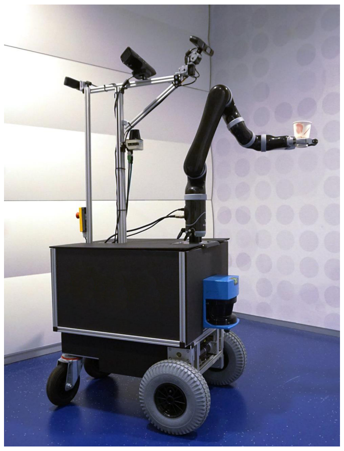


Fig. 6. The robot used for the experiments. The laser in the front of the robot is used for localization and obstacle avoidance. The two 3D sensors on both ends of the top bar are for 3D obstacle avoidance. The front sensor rotates based on the current speed of the robot to better detect obstacles.

C. Simulation

In order to test the performance of our method, we made sure that the navigation environment has different varieties such as narrow hallways, open space, doors, etc. which can be seen in Figure 5a. The robot starts from point A, and continues through all checkpoints until it reaches point E. We measure the time between each pair, and average the results for comparison. We performed 500 trials for each of the best static settings (safe setting), fast parameter and our dynamic parameter setting. If the robot was not able to reach a point, we added it to the number of failures and marked that trial length as one thousand seconds. We then calculated the average time without penalty, and calculated the standard deviation not considering the failures. Table III shows the time average, standard deviation and the number of failures due to collision or lack of sampling in the simulated environment. It is evident that our dynamic situation analysis performs superior to a single parameter set. The reasoning is straightforward, the clustering method correctly detects the majority of situations, and sets the best parameters. For example, close to the doors, the system uses the safest approach. With a limited velocity space, the system can simulate enough velocity points to find the best path in a congested area. On the other hand, in an open space, the system can use a lower number of sample points and reach higher speeds without reliability and safety issues.

D. Real Experiments

We used a similar approach to that of section IV-C. The selected path included narrow and large hallways, congested areas, and doors which can be seen in Figure 5b. We performed 20 trials for each of the base, fast, and our dynamic parameters for A-B, B-C, and C-D-E points. Table IV shows the average time, standard deviation, and the number of failures for each method and their respective pair of points. There were some complications during the real experiments. Unlike the simulator, after online parameter changes of the navigation system, the sensor buffers were frozen for approximately one second which resulted in a full stop of the robot. Depending on the complexity of the scene, the number of parameter change operation varied. However, our dynamic method is still superior (18% faster than the safest method) in comparison to the fast and safest set of parameters with the total time of $116.01s$ without considering collision penalty. We estimate that this number will be closer to that of our simulation results if this sensor buffer failure did not occur. If we take a look at point to point results, we see that the best static method is performing better than our dynamic approach between point A and B. The main reason here was indeed the sensor buffer freezing problem because of high number of parameter changes. In the rest of the path, however, the dynamic parameter system performs better. It is notable that the fast parameter set is very unreliable in crowded areas with 18 total collisions, some of which actually damaged one of our sensors. We can conclude that using our approach, we can achieve a safer, faster, and more reliable navigation.

V. CONCLUSION

In this paper, we introduced a novel approach to automatically analyze a robot's surrounding environmental situation using HoG features and unsupervised clustering. The feature extraction and clustering were done on an obstacle matrix (cost map) which is a two dimensional projection of realtime sensory readings. We first extracted the best safe and fast parameter sets without using our dynamic approach. Then, using our method, we determined the congestion and danger level of these clusters by calculating a scaled average image for each cluster. Finally, we tuned multiple parameters for the local planning module of the robot in order to navigate

TABLE IV
THE REAL EXPERIMENT RESULTS. ON AVERAGE THE DYNAMIC PARAMETER SETS PERFORM 18% BETTER THAN THE BEST STATIC SET OF PARAMETERS, AND A HIGH SPEED SET. IN ADDITION, THE NUMBER OF FAILURES IS SIGNIFICANTLY LOWER THAN THAT OF THE REST.

	A to B			B to C			C to E			Full Path
Parameter Type	Mean (w/o Penalty)	Standard Dev.	Failures	Mean (w/o Penalty)	Standard Dev.	Failures	Mean (w/o Penalty)	Standard Dev.	Failures	Mean (w/o Penalty)
Dynamic	137.71 (26.5) s	16.9 s	2	**90.51 (42.51) s**	5.14 s	1	**94.64 (47.0)** s	23.32 s	1	**326.06 (116.01)s**
Best (Safe)	**120.59 (22.9) s**	5.3 s	2	111.52 (64.27) s	7.0 s	1	196.73 (55.0) s	11.57 s	3	893.151 (118.82)s
Fast	372.96 (35.44) s	26.22 s	8	144.58 (43.9) s	7.31 s	2	375.63 (39.4) s	7.18 s	8	428.38 (142.13)s

with a higher performance and reliability. Using the clusters and danger level scores, we extrapolated new parameters and performed simulation experiments. In simulation, we had a performance increase of 33%, and a reliability increase of 28% in terms of navigation failure. We then performed the experiments on a real mobile robot with the same cluster centers that were learned during the simulation experiments. The performance and reliability increase were 18%, and 30% respectively. We can conclude that by understanding the surrounding environment, we can dynamically change navigational parameters to allow for a faster and more reliable movement.

Future Work: There are several improvements that can be done to further enhance the reliability and performance of the system. Currently, the number of clusters and the danger levels are calculated by manually observing the dendrogram and average images of the clusters. This procedure can be automated by calculating similarity values between cluster levels (Figure 3), and assigning scores to different sections of our windowing approach (Figure 2). In addition, in this research, we only optimize the parameters for the dynamic window approach. However, we can also customize the sampling procedure itself based on the robot and the detected density of the current environment. Finally, we can use a reinforcement learning or other optimization algorithms to tune the parameters for each of the detected environments.

REFERENCES

[1] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, pp. 460–466, 2015.

[2] H. R. Everett and D. W. Gage, "From laboratory to warehouse: Security robots meet the real world," *The International Journal of Robotics Research*, vol. 18, no. 7, pp. 760–768, 1999.

[3] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, "Towards fully autonomous driving: Systems and algorithms," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*, June 2011, pp. 163–168.

[4] SILVER project, *Supporting Independant living for the elderly through robotics*. SILVER consorsium, European Union, 2016. [Online]. Available: http://www.silverpcp.eu/newsletter-july-2016/

[5] S. Thrun, "Robotic mapping: A survey," in *Exploring Artificial Intelligence in the New Millenium*, G. Lakemeyer and B. Nebel, Eds. Morgan Kaufmann, 2002, to appear.

[6] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial intelligence*, vol. 128, no. 1, pp. 99–141, 2001.

[7] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[8] A. S. Glassner, *An introduction to ray tracing*. Elsevier, 1989.

[9] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition. CVPR. IEEE Computer Society Conference on*, vol. 1, 2005, pp. 886–893.

[10] D. V. Lu, "Contextualized robot navigation," Ph.D. dissertation, Washington University in St. Louis, December 2014.

[11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[12] P. N. J. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems, Science, and Cybernetics*, vol. SSC-4, no. 2, pp. 100–107, 1968.

[13] T. J. Misa and P. L. Frana, "An interview with Edsger W. Dijkstra," *Commun. ACM*, vol. 53, no. 8, pp. 41–47, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1787234.1787249

[14] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *Robotics Automation Magazine, IEEE*, vol. 4, no. 1, pp. 23 –33, Mar. 1997.

[15] B. P. Gerkey and K. Konolige, "Planning and control in unstructured terrain," in *Workshop on Path Planning on Costmaps, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA*, 2008.

[16] F. Tombari, S. Salti, and L. Di Stefano, "Unique signatures of histograms for local surface description," in *Computer Vision–ECCV*. Springer, 2010, pp. 356–369.

[17] A. Shantia, R. Timmers, L. Schomaker, and M. Wiering, "Indoor localization by denoising autoencoders and semi-supervised learning in 3D simulated environment," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–7.

[18] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, L. M. L. Cam and J. Neyman, Eds., vol. 1. University of California Press, 1967, pp. 281–297.

[19] J. C. Gower and G. J. S. Ross, "Minimum spanning trees and single linkage cluster analysis," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 18, no. 1, pp. 54–64, 1969. [Online]. Available: http://www.jstor.org/stable/2346439

[20] L. Vuurpijl and L. Schomaker, "Finding structure in diversity: A hierarchical clustering method for the categorization of allographs in handwriting," in *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, vol. 1. IEEE, 1997, pp. 387–393.

[21] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems. (IROS 2004). IEEE/RSJ International Conference on*, vol. 3, 2004, pp. 2149–2154 vol.3.

[22] G. Bradski, "OpenCV library," *Dr. Dobb's Journal of Software Tools*, 2000.

[23] MATLAB, *version 8.4.0 (R2014b)*. Natick, Massachusetts: The MathWorks Inc., 2014.

[24] P. F. Dubois, K. Hinsen, and J. Hugunin, "Numerical Python," *Computers in Physics*, vol. 10, no. 3, May/June 1996.