Auto-tuning Particle Filter Parameters Under Changing Environments

Taijing Chen[1*], Xuefei Zhao[1*], Varshinee Sreekanth[1*]

***Abstract*— In mobile robot localization, a significant portion of time and effort is spent in tuning particle filter parameters. In this paper, we aim to demonstrate a methodology for generating and automatically tuning particle filter parameters. The goal of this is to reduce the need for human supervision in parameter tuning, as well as allow for the robot to adjust parameters as the environment around it changes. In our experiment, we tuned for different observation and obstacle contexts and noise levels.**

Experimental results indicate that robot localization performance with autotuning is comparable to performance with hand-tuned parameters. The applications of this methodology include particle filter tuning in noisy and varied environments.

I. Introduction

A particle filter is a popular approach to solving state estimation problems for dynamic systems. Owing to its proven-success in inferring non-linear non-Gaussian states from measurements, it's been widely deployed in many engineering fields. However, a particle filter with high accuracy and robustness requires a good set of parameters, which consumes much time and human supervision for at least two reasons.

Firstly, the same set of parameters can have a different localization performance due to the particle filter's random sampling process. Under error-prone situations (noisy environment, abrupt transformation, etc), particle filters require multiple trials on the same environment to evaluate a single set of parameters. Moreover, users need to verify the quality of the same set of parameters on other environments as well. If the parameters fail to demonstrate desired consistency, the whole process needs to be repeated.

Secondly, a universal best set of parameters is not guaranteed to exist. Parameters that work well in one environment may fail to perform in other environments. Therefore, users need to compromise particle filter performance for general cases in exchange for good performances on edge cases.

To tackle these difficulties, we asked 2 main questions: 1) Can we automate the parameter evaluation process so that we can find best parameters without human supervision? If so, 2) Can a robot auto-tune its parameters at run time to adapt to the changing environment?

*Equally contributing authors

[1]Taijing Chen, Xuefei Zhao, and Varshinee Sreekanth are with Department of Computer Science, University of Texas at Austin, Austin, TX 78712 {taijingchen, zhaoxuefei, varshinee}@utexas.edu

II. Related Work

Previous works have resolved similar problems in robotics. Xiao et al. [1] designed an adaptive navigation planner. In this model, environments are categorized into various contexts, and human demonstration is set to be the "ground truth" that the algorithm learns the parameters from. In context of particle filters, it is hard, if not impossible, for humans to inform the robot on its ground truth localization. Therefore, human demonstrations are not used as the ground truth in our auto-tuning model. However, inspired by the ideas of contexts, we defined contexts for our particle filter to limit the parameter search space.

Additionally, Lima and Ventura [2] have investigated parameter optimization for localization. In the paper, they proposed SMAC (Sequential Model-Based Optimization for General Algorithm Configuration), a random-forest-based, general Java tuning software. To use SMAC, users need to define a heuristic to evaluate the quality of each run. It is feasible to optimize parameters for different contexts using SMAC offline. However, to auto-tune the parameters at run time, the algorithm still needs to rely on our framework to identify the contexts in order to select the best-fitting parameters. Furthermore, the major portion of our parameter-finding time is spent in recording bag files. Despite not having explicitly tested on SMAC's training time, we hypothesize that SMAC would also require a long time to generate bag files and evaluate each run. As a result, we decide not to adopt a SMAC-like approach.

III. Method

We broke down our project into 3 tasks:

1) Classify all possible environments into well-defined contexts.
2) Find the best set of parameters for each context.
3) Identify the current context during run-time and use the best-fitting parameters

A. Contexts

In this project, we categorize different types of environments into *contexts*. The goal of our training is to find the best-performing parameters for each of these contexts. At run-time, the particle filter algorithm identifies the contexts from the its observations online, and changes its particle filter parameters to the predetermined set of parameters that are known to be the best fit for the current state.

For now, cutoffs for each of these contexts are hard-coded and heuristic-based. For the purposes of our experiments, we don't evaluate the quality of the cutoffs. It is very likely that the "actual" best parameters around two sides of the cutoff are in fact the same, or indistinguishable, but that is out of scope of the purposes of this paper.

We defined the following conditions to classify the environment:

1) Observation noise: Past works often treat obstacles as observation noise. However, in our model, LIDAR noise and obstacles are separated into two distinct sources of noise. Environments can be classified as either a low-noise context or high-noise context. Observation noise reflects hardware errors or other LIDAR errors. It is determined by the LIDAR point cloud's standard deviation, which must be high in order to classify the environment as high-observation noise. (Fig. 1).

2) Obstacles: Obstacles are laser observations inconsistent with the map. An environment can be classified as either a few-obstacles or many-obstacles context. Obstacles are defined as unforeseen laser readings that do not appear on a predetermined map. If a laser reading is more than a certain distance from the closest wall, it is counted as an obstacle. If more than a fixed percentage of the laser readings at a given time are determined to be obstacles, the environment would be classified as a many-obstacle context (Fig 2).

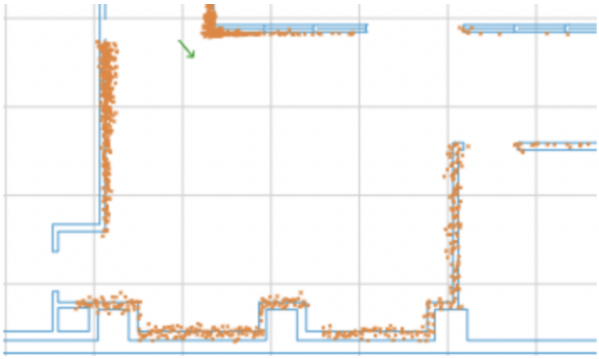

Fig. 1. An environment with high observation noise

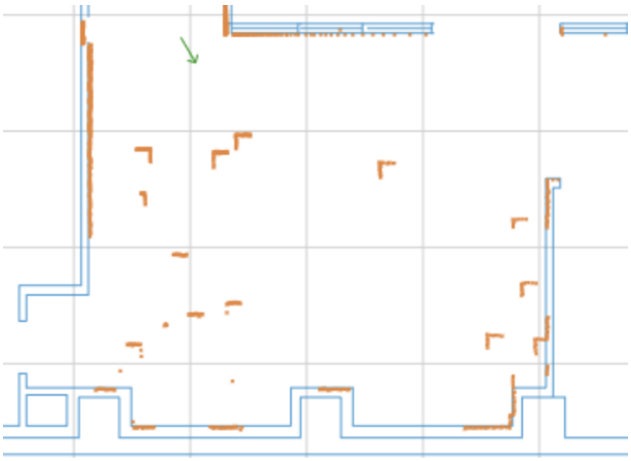

Fig. 2. An environment with high obstacles

Integration of the 2 types of conditions gives 4 possible contexts: low-noise and few-obstacles, low-noise and many-obstacles, high-noise and few-obstacles, and high-noise and many-obstacles. Our training and evaluation focuses on finding and using the best parameters for these four contexts.

B. Dataset

The simulator (with `--localize` flag) is used to generate the dataset in order to gain denser, reliable ground truths. We wrote shell and python scripts to automate generation random datasets for both training and testing. The script is designed to 1) adjust the laser reading noises in the simulator configuration file, and 2) simulate random obstacles on the map. The scripts can also generate multiple rosbag files of different low and high observation noise settings (instead of a single bag file) for each context for better robustness.

The navigation module is used on the simulator to generate bag files. The simulator localization is treated as the ground truth, which is later compared to trajectories localized by different versions of the particle filter for training and evaluation of the auto-tune module.

C. Training

To find the best parameters for each context, we took a brute-force approach. Our process break-down is below:

1) Record ground truth files: To generate trial runs, multiple bag files are recorded from simulation. Each bag file has various amounts of either observation or obstacle noise added to it to simulate different contexts. The rosbag files store `odom`, `scan`, and `localization` topics. `odom` and `scan` are fed into the particle filter, and `localization` is used to form the ground truth. This allows us to have denser ground truths, as mentioned previously.

2) Select parameters to tune: In the interest of time, we discretized the parameter search space into sets of potential values and limit the number of parameters to tune. Parameters included in the search space are:

- Sensor standard deviation: the standard deviation of sensor readings.
- d_long: the case where observations are further away than expected in the robust observation likelihood model.
- d_short: the case where observations are closer than expected in the robust observation likelihood model.

3) Evaluate each parameter set: To determine the efficacy of a certain set of parameters, we evaluate the performance of our auto-tuned particle filter as follows:

(a) For each context, run every set of parameters from the parameter search space.
(b) For each set of parameters, play all bag file in that context.
(c) For each bag file, calculate the average distance between the ground truth and auto-tuned particle filter localization.
(d) For each context, find the set of parameters that gives the lowest mean of the average distances (the mean of the value from (c) for all bag files).

D. Run-time

During run-time, the auto-tune algorithm determines the current context and uses the corresponding parameter set for each timestamp. The algorithm makes decision based on the following error types:

1) Observation error: The standard deviation of the observation noise (excluding obstacles) along the observation line is used to determine the level of observation noise. We segment the laser readings into small chunks, and apply iterative linear regression to find a line of best fit for observations within each segment. Then the mean of standard deviation of the distance to the line of best fit across the segments is computed. If this value is higher than a cutoff, the robot will assume that this is a high-observation-noise context.

2) Obstacles: The proportion of laser readings that are more than a certain distance away from the predefined map is used as the heuristic to determine the density of obstacles.

IV. Experiment

A. Experiment Setup

1) Datasets: 16 rosbag files are generated to train for the best parameters, with 4 for each context. For the low-noise contexts, the sensor standard deviation is set to be 0 and 0.02; for the high-noise contexts, the sensor standard deviation is set to be 0.07 and 0.1. To simulate obstacles, we generated 100 random boxes of side length 0.5 meters on the map. The few-obstacle contexts contain 0 and random 20 boxes; the many-obstacle contexts contain random 50 and 100 boxes.

We then generate 4 files for the testing set, 1 for each context, following a similar methodology as above. The obstacle placement is randomized and sensor noise is varied from the training set.

2) Parameter Cutoffs: During run-time, to determine the context at a given moment of time, the following cutoffs are set:

- Observation noise: If the detected sensor standard deviation is greater than 0.03m, then the context will be classified as high-noises.
- Obstacles: If a laser reading is further than 0.5m from a wall, then the reading will be classified as an obstacle.
- Obstacle proportion: If more than 10% of laser readings are obstacles at a given timestamp, then the environment will be classified as high-obstacles.

B. Evaluation Methods

In the experiment, the auto-tuned parameters are compared with the control set – the manually-tuned parameters that were previously hand-tuned by eye in Assignment 2. The evaluation metrics are as follows:

1) Accuracy: Similar to our training metric (section III-C.3), the average distance between the ground truth locations and particle filter localization is measured. We then calculate the mean distance across runs of different bag files in the same context to ensure that the auto-tuned particle filter is able to yield comparable results to the vanilla particle filter.

2) Precision/Consistency: Additionally, we also run multiple trials with the same bag file and measure the standard deviation across runs to ensure that our result is consistent and not just a fluke because of the inherent randomness in particle filters.

C. Results

Overall, our auto-tuned particle filter model successfully identifies the current context and picks the best parameters for the context. It's worth noting that all bag files have high motion noise. This is to ensure particle filters to mainly rely on observations to locate themselves.

Table I lists all the auto-tuned parameteres for each context. We can see the values are reasonable.

TABLE I
Auto-tuned Parameters for Each Context

	sensor_stddev	d_long	d_short
Low-Sensor Few-Obstacle	0.05	1.5	1.0
Low-Sensor Many-Obstacle	0.15	2.5	1.5
High-Sensor Few-Obstacle	0.15	2.0	1.0
High-Sensor Many-Obstacle	0.2	2.5	1.5

1) Training: In this section, we evaluate particle filter's performance on the training set. These results (Tables II and III) aim to show the validity of our auto-tuning approach and the correctness of our implementation.

(a) Accuracy: Overall, the auto-tune particle filter has small localization errors. This shows our particle filter's ability in recognizing the environment and switching its parameters to adapt to the environment.
(b) Consistency: Across different runs on the same bag file, auto-tune particle filter performs consistently, while the performance of the vanilla particle filter greatly fluctuates.

TABLE II
VANILLA PARTICLE FILTER (M)

	Low-Sensor	High-Sensor
Few-Obstacle	1.016 ± 1.136	1.217 ± 0.553
Many-Obstacle	1.282 ± 0.206	0.806 ± 0.555

TABLE III
AUTO-TUNED PARTICLE FILTER (M)

	Low-Sensor	High-Sensor
Few-Obstacle	0.116 ± 0.033	0.083 ± 0.024
Many-Obstacle	0.131 ± 0.032	0.154 ± 0.057

2) Testing: In this section, we evaluate the particle filter's performance on a testing set (Tables IV and V) where all observation standard deviations, motion noise, and obstacles' placements and distributions are different from the training set.

Note that in this dataset, the motion noise is smaller than the ones used for training. Therefore, both particle filters generally have less localization errors compared to the ones in Tables II and III.

(a) Accuracy: Trajectories localized by the auto-tuned particle filter deviate from the ground truths less than the ones localized by the vanilla particle filter, especially for the high-sensor, few-obstacles case.
(b) Consistency: Results from the auto-tuned particle filter have lower standard deviation for all cases.

TABLE IV
VANILLA PARTICLE FILTER (M)

	Low-Sensor	High-Sensor
Few-Obstacle	0.246 ± 0.087	1.925 ± 1.740
Many-Obstacle	0.491 ± 0.178	0.146 ± 0.063

TABLE V
AUTO-TUNED PARTICLE FILTER (M)

	Low-Sensor	High-Sensor
Few-Obstacle	0.120 ± 0.040	0.070 ± 0.020
Many-Obstacle	0.099 ± 0.044	0.115 ± 0.016

V. DISCUSSION

A. Significance

Particle filter localization is a math-heavy algorithm, which may prevent some people from using it. Auto-tuning allows people with no prior knowledge of the inner workings of particle filters to use localization. Users can run our auto-tune module to find the best parameters without knowing the mathematical impacts of the parameter values themselves.

The other benefit of our auto-tuning is that this allows the robot to adapt to its surroundings and change parameters when the environment changes. Auto-tune particle filter is able to yield consistent performance under various environments, better ensuring the reliability of our robots.

Finally, our auto-tuning's primary benefit is that it can produce comparable results to hand-tuned parameters *without human supervision*. The entire process, start to finish, is automated, and thus there does not need to be any manual tuning by humans.

This project is mostly a proof-of-concept – We have tried to show that a particle filter's parameters can be found without human supervision, and adjusted them accordingly during run time. Although we ran our experiment in simulation, we expect this concept transferable to real-world robots with following extensions.

B. Future Work

Although the results look promising, this paper is still only the first step in auto-tuning parameters. The following are some potential work to be done in the future.

1) Expand contexts: We can expand the auto-tune module to include different types of contexts. Currently we are mainly concerned with the various types of observational noises, as they are most directly related to particle filter parameters. However, other types of contexts can significantly affect a particle filter's performance as well. For example, when going down a long hallway, a robot cannot determine its location based purely on observation - Therefore, we want to trust the motion model more in the context of a long hallway.

2) Expand the scope of errors: We can expand types of errors as well. More specifically, we want to take motion errors into account. For our project, we didn't consider the motor commands issued to the robots

at each time stamp. We want to modify our context detection to pick up on motion noise in addition to observation noise. We would do this with Correlative Scan Matching, and by comparing the odometry readings to the motor commands.

3) Sparser ground truth: Real robots often don't have dense ground truth. For the purposes of this paper, we used a dense ground truth with the simulator as a proof-of-concept. We want to expand this implementation to work with sparser and more error-prone ground truths that are available on the real robots.

4) Add ML: Currently, our context cutoffs are determined arbitrarily, with minimal tuning. Although it didn't make too much of a difference for our purposes, a follow-up study could use clustering algorithms or support-vector-machine-style algorithms to determine the cutoffs for each context type to determine a separate set of parameters for them.

VI. Appendix

Github repo: `https://github.com/TieJean/particle-filter-autotuning`

References

[1] Xuesu Xiao, Bo Liu, Garrett Warnell, Jonathan Fink, and Peter Stone. 2020. APPLD: Adaptive Planner Parameter Learning from Demonstration. CoRR abs/2004.00116 (2020). arXiv:2004.00116 https://arxiv.org/abs/2004.001164

[2] O. Lima and R. Ventura, "A case study on automatic parameter optimization of a mobile robot localization algorithm," 2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), 2017, pp. 43-48, doi: 10.1109/ICARSC.2017.7964050.