2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Online Localization with Imprecise Floor Space Maps using Stochastic Gradient Descent

Zhikai Li[1], Marcelo H. Ang Jr.[2] and Daniela Rus[3]

***Abstract*— Many indoor spaces have constantly changing layouts and may not be mapped by an autonomous vehicle, yet maps such as floor plans or evacuation maps of these places are common. We propose a method for an autonomous robot to localize itself on such maps with inconsistent scale using Stochastic Gradient Descent (SGD) with scan matching using a 2D LiDAR. We also introduce a new scale state in 2D localization to manage the possible inconsistent scale of the input map. Experiments are conducted in an indoor corridor using three different input maps - a point cloud, a floor plan, and a hand-drawn map. The SGD localization algorithm is bench-marked to Adaptive Monte Carlo Localization (AMCL). In a point cloud mapped environment, our algorithm achieves 0.264m and 5.26° average position and heading error respectively. On the hand-drawn map, our SGD localization algorithm remains robust while AMCL fails. The role of the scale state in our SGD localization algorithm is demonstrated in poorly scaled maps.**

I. INTRODUCTION

Most autonomous robots must map their environment before they can localize on it. Simultaneous Localization and Mapping (SLAM) with LiDAR [1], [2] or vision [3], [4] is often used to achieve this. However, in new operating environments, such maps may not be available. Furthermore, in dynamic environments such as shopping malls with frequent renovations or cordoned off event areas, re-mapping the environment is needed and is extremely time consuming.

Research exploring alternative maps are uncommon. For indoor spaces, GLFP [5] is proposed for floor plans, but uses hand-picked features. Intention-Net [6] and Nav-Net [7] explores floor plans as well, but does not guarantee generalization of its neural network used to all environments.

In this paper, we use imprecise floor space maps (also referred to as 'improper maps') as a replacement to point cloud maps for indoor 2D localization by utilizing Stochastic Gradient Descent (SGD). These floor space maps can include floor plans, fire escape maps or shopping directories. As opposed to point cloud maps, these maps are targeted at people rather than robots. As a result, they pose a challenge during localization as they may not be drawn accurately, and can be locally distorted at certain portions. Yet, using these maps has advantages. Firstly, no prior sensor mapping with LiDAR or vision is required. The robot can travel from its start to endpoint without exploring the environment. Secondly, for highly dynamic environments, one can easily update these maps as map accuracy is not required. This method can hence be applied on cleaning robots in malls and homes, or automated food serving robots in restaurants.

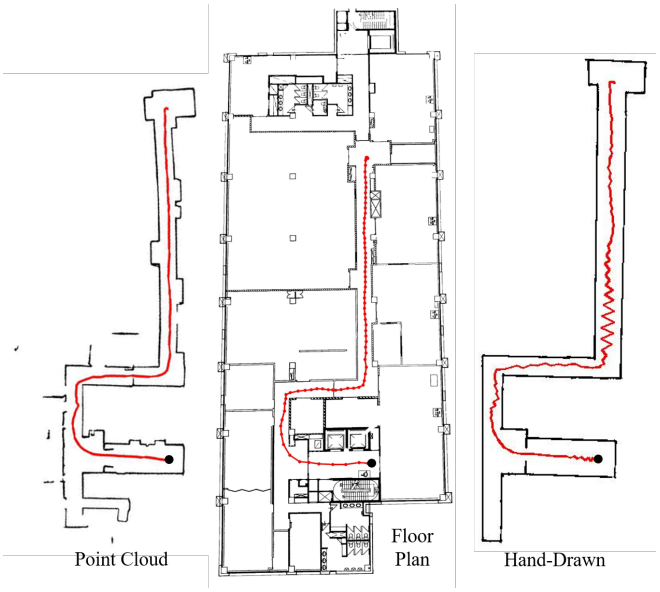

Fig. 1. SGD state trajectory with different input maps.

The goal of the SGD localizer is to track the robot's state and produce a sensible state trajectory when given an improper map, while maintaining localization accuracy in a properly scaled point cloud map, as shown in Figure 1.

Through this paper, we make the following contributions:

- Propose a method of state tracking using SGD that avoids hand-picked features.
- Introduce scale as a state of the agent to handle distorted maps and experimentally show its effects.

The SGD algorithm is also tested on real-world data, demonstrating the robustness of the method.

II. BACKGROUND AND RELATED WORK

Autonomous mobile robots can map their environment by performing SLAM or depending on a reliable source of odometry, such as LOAM [8] or V-LOAM [9], which uses LiDAR and LiDAR-Vision sensor fusion respectively, to track its own state and map its environment. To localize, some methods include using Extended Kalman Filters [10] or using a Monte Carlo approach to approximate the belief posterior [11]. Mono vision with a feature detector [12] or Stereo vision generating a particle swarm [13] can be used for localization as well.

[1]Zhikai Li is with the Singapore-MIT Alliance for Research and Technology (SMART) Centre, Singapore e0004076@u.nus.edu
[2]Marcelo H. Ang Jr. is with the National University of Singapore, Singapore mpeangh@nus.edu.sg
[3]Daniela Rus is with the Massachusetts Institute of Technology, Cambridge, MA, USA rus@csail.mit.edu

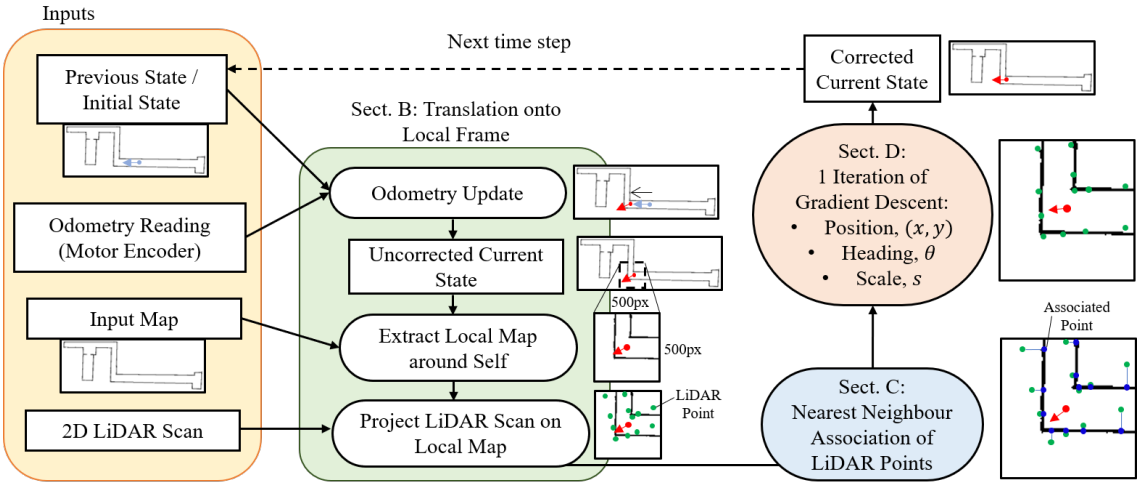

Fig. 2. Architecture of the SGD state tracking algorithm. The state from the previous timestep (in light blue) is first updated to a current state (in red) using odometry measurements. A sub-segment of the global map is cropped with the robot in the middle is scaled up to 500 by 500 pixels and the LiDAR scan is projected onto this local map (in green). Each green LiDAR point is associated with its nearest black pixel (Indicated by blue dots), allowing a cost function to be optimized to minimize the distance between them. One iteration of gradient descent is applied to update the four state components - x, y, θ, s to obtain a corrected state. Through passing the corrected state to the next iteration, the process can be repeated.

Without pre-mapping the environment, a way to navigate the environment is to perform online SLAM. For LiDAR SLAM, a Kalman filter was proposed to localize the agent onto past observations and map out new landmarks [1]. This method is then improved in FastSLAM [2], [14] where k-d trees were used to increase the efficiency of the algorithm. For vision SLAM, Mono-SLAM was proposed to build a map of persistent static obstacles [3]. ORB-SLAM later used the ORB feature detector for the SLAM algorithm to be more robust to motion clutter [4]. SLAM methods work well in exploring new environments and also outputs a map of it. However, in goal-directed applications, one cannot specify a goal position to the robot without a prior map. Loop closures, required for SLAM, is also undesirable when the robot is to travel to the target location as fast as possible. In this paper, an improper map drawn by a person can be given to the robot operating in a foreign environment, thereby avoiding the shortfalls SLAM has in goal-directed operations.

Several works have also explored the use of alternative maps for localization. In [15], a route with multiple checkpoints is planned on a rural road. The autonomous car repeatedly steers towards the next checkpoint while constrained to drivable terrain. In [16], a neural network is used to bridge raw sensor input to steering output, using road topology to form a probability distribution over viable movements. These methods are suited for structured road environments, with a road directory used as the alternative map. IntentionNet [6] and Nav-Net [7] have been proposed in unstructured environments. Both methods use deep learning, by first doing motion planning on a floor plan, then take the local intention on the floor plan with camera vision as inputs to a neural network to output local steering controls. In [5], a floor plan was used for localization using pre-defined features on the floor plan. Whilst this approach is feasible, we would like to avoid the use of hand-picked features.

For localization, gradient descent methods are used in [17] and [18] to optimize the motion model to make it more compliant to the robot's motion constraints. In [19], the gradient descent method is applied to particles on a particle filter to optimize the sub-sampled belief states to better match its observations, so that fewer particles are required for the same quality of localization. We apply SGD to directly correct a single tracked state to match its observations. A scale state is also introduced and optimized so that the approach remains robust in distorted maps.

III. STATE TRACKING ON FLOOR SPACE MAPS USING STOCHASTIC GRADIENT DESCENT

In this paper, a modified 2D state tracking problem will be tackled. Given any map, the goal is to find X_t, a four-dimensional state $X = [x, y, \theta, s]$, at time t, where x, y, θ are the global coordinates and heading of the robot on the map respectively, while s is scale of map. The robot knows all previous states, control actions, and observations.

As the floor space map can be locally distorted, the robot's state contains s, a scale (also commonly known as resolution) which the robot should use for the observation to be consistent with the map. Allowing s to change allows the state tracker to adapt to the varying scale of the map.

A. Architecture of the State Tracker using Stochastic Gradient Descent

The architecture of our approach is summarised in Figure 2. The system accepts the following inputs - an image as the map, a previously known state, X_{t-1}, basic odometry, and the most recent LiDAR sweep of the surroundings. Using this information, the system outputs the most recent state X_t. The process can be repeated by passing the corrected state as the 'previous state' for the next iteration to perform a full online state tracking.

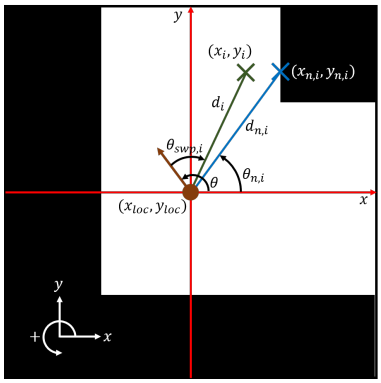

Fig. 3. Local map and frame reference.

In each iteration, the robot first takes the previous state on the global map image and measured changes from the odometry to update the state. A local map centered around the robot's current state is extracted using a sub-segment of the global map. The local map serves as a translated local frame. LiDAR sweep points are then projected on the local map, and the nearest black pixel is associated with each LiDAR point. A cost function is then computed and used to update the four states - x, y, θ, s.

The following subsections describe each component of the architecture in more detail as indicated in Figure 2.

B. Translation onto Local Frame

A Cartesian frame is defined as the global frame with origin is the bottom left corner of the map image.

In Figure 2, the system uses the previous state and basic odometry to obtain a rough estimate of the current state on the map. A sub-segment (size depending on LiDAR max range and current scale, s) of the map is cropped and scaled into a 500 by 500 pixel local map, with the robot placed in the center. The center of the local map is defined to be the origin of the local frame. No rotation transformation is made. The local frame is oriented the same as the global frame, while the LiDAR points are rotated and projected onto the local frame instead. This is to simplify the derivation of the cost function and its derivatives.

The local map is often scaled up when turned into a 500 by 500-pixel image. This increases the local map resolution, preventing LiDAR depth readings from being scaled-down and forced to conform to a low-resolution global map.

C. Nearest Neighbour Association

Black pixels on the map indicate obstacles. LiDAR points are associated with their nearest black pixel as shown in 2.

With reference to Figure 3, denote (x_i, y_i) as the coordinates of the i-th LiDAR point of the sweep in the local frame and $(x_{n,i}, y_{n,i})$ for its nearest associated black pixel. d_i and $d_{n,i}$ are the distances of the i-th LiDAR point and its associated black pixel from the local origin respectively. $\theta_{n,i}$ is the angle of the associated pixel from the horizontal and $\theta_{swp,i}$ is the angle of the i-th LiDAR point taken from the current heading of the robot. Finally, (x_{loc}, y_{loc}) is a variable denoting the position of the robot in the local frame ((0,0) before update) and θ is the heading of the robot.

If we also denote $D_{n,i}$ to be the distance of the i-th associated pixel from the robot in the global frame, then (x_i, y_i) is be dependent on x_{loc}, y_{loc} and θ as denoted in Equations 1 and 2, while $(x_{n,i}, y_{n,i})$ will be dependent on the scale used, s, as denoted in Equations 3 and 4.

$$x_i = d_i \cos(\theta_{swp,i} + \theta) + x_{loc} \tag{1}$$

$$y_i = d_i \sin(\theta_{swp,i} + \theta) + y_{loc} \tag{2}$$

$$x_{n,i} = sD_{n,i} \cos\theta_{n,i} \tag{3}$$

$$y_{n,i} = sD_{n,i} \sin\theta_{n,i} \tag{4}$$

D. Stochastic Gradient Descent for States

The cost function is the sum of squared distances between the LiDAR points and its associated point in Equation 5, where N is the number of LiDAR points in one sweep. Ideally, the cost function should refer to associated distances in a properly scaled map. However, due to random distortions on the floor space map used, each sweep is instead treated as a random observation of the cost function, hence the stochastic nature.

$$Cost, C = \sum_{i=1}^{N} [(x_i - x_{n,i})^2 + (y_i - y_{n,i})^2] \tag{5}$$

(x_i, y_i) and $(x_{n,i}, y_{n,i})$ collectively depends on x_{loc}, y_{loc}, θ and s. To perform gradient descent, the partial derivatives of the cost function with respect to each of the four state dependencies is taken to obtain Equations 6 to 9.

$$\frac{\partial C}{\partial x_{loc}} = 2 \sum_{i=1}^{N} x_i - x_{n,i} \tag{6}$$

$$\frac{\partial C}{\partial y_{loc}} = 2 \sum_{i=1}^{N} y_i - y_{n,i} \tag{7}$$

$$\begin{aligned} \frac{\partial C}{\partial \theta} =& 2 \sum_{i=1}^{N} (x_{n,i} - x_i)(d_i \sin(\theta_{swp,i} + \theta)) \\ &+ 2 \sum_{i=1}^{N} (y_i - y_{n,i})(d_i \cos(\theta_{swp,i} + \theta)) \end{aligned} \tag{8}$$

$$\begin{aligned} \frac{\partial C}{\partial s} =& 2 \sum_{i=1}^{N} (x_i - x_{n,i})(D_{n,i} \sin\theta_{n,i}) \\ &+ 2 \sum_{i=1}^{N} (y_i - y_{n,i})(D_{n,i} \cos\theta_{n,i}) \end{aligned} \tag{9}$$

Equation 10 shows s being updated, while x, y and θ are updated with similar equations. x, y are directly updated in the global frame as changes to x, y are preserved between

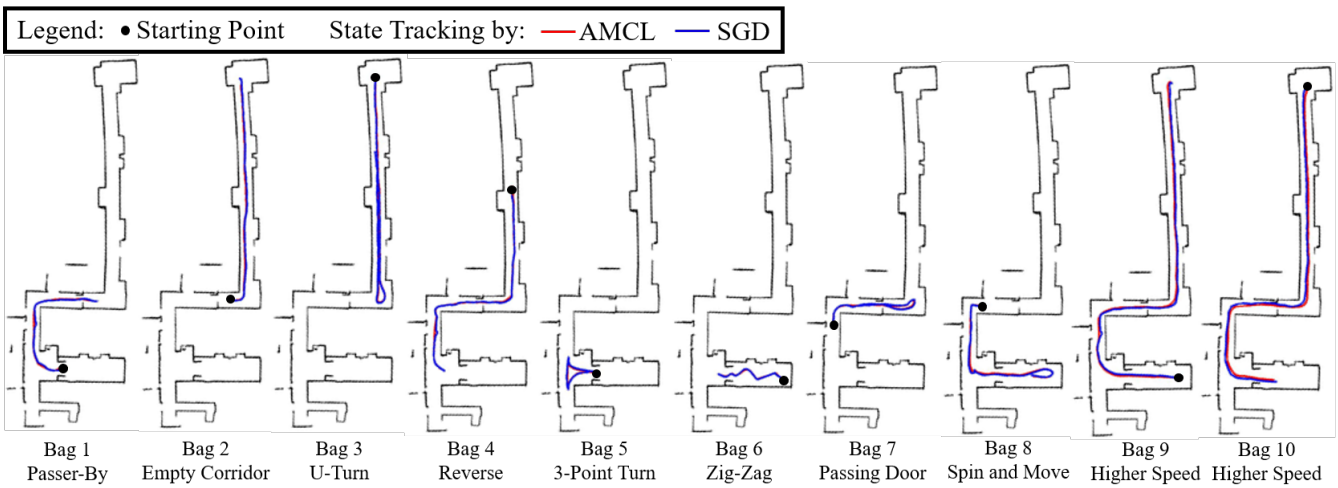

Fig. 4. State trajectories of AMCL and SGD on a point cloud input map. State trajectories produced by our SGD state tracking algorithm is consistent with AMCL.

Algorithm 1: SGD State Tracking

Inputs : map, initial_state $= [x, y, \theta, s]$, learn_rates $=$ (pos_lr, angle_lr, scale_lr)
Output: State Trajectory on Input Map

```
trajectory ← [];
prev_state ← initial_state;
while robot_is_running() do
    odom_reading ← read_from_encoder();
    lidar_sweep ← read_from_lidar();
    uncorr_state ← prev_state + odom_reading;
    subsegment ← crop(map, uncorr_state);
    local_map ← rescale(subsegment, (500, 500));
    lidar_points ← project_to_local_map(lidar_sweep);
    assoc_points ← nearest_obstacle(lidar_points,
      local_map);
    grads ← get_partial_derivs(lidar_points,
      assoc_points);
    corrected_state ← uncorr_state - learn_rates ×
      gradients;
    trajectory ← trajectory.append(corrected_state);
    prev_state ← corrected_state;
end
return trajectory
```

the global and local frame by a factor of s. The output after update is the corrected state as in Figure 2.

$$s_t = s_{t-1} - \eta_s \frac{\partial C}{\partial s} \tag{10}$$

η_s is the learn rate. The learn rates for each state component can be tuned individually. In our approach, the same learn rate is used for both x and y as a positional learn rate, while two different smaller learn rates are set for θ and s. An analysis of the learning rates will be discussed in the results.

E. Algorithmic Flow of SGD Localization

Algorithm 1 shows the iterative flow of the SGD state tracker. Starting with a given initial state, the robot iterates the state tracking process while the robot is running. Every iteration, odometry and LiDAR readings are read from the respective sensors. The odometry reading is first used to update the previous state to an estimate of the current state, $uncorr\_state$. A local map is cropped and rescaled into a 500 by 500-pixel local map for the input map centered around $uncorr\_state$. The LiDAR readings are then projected to this local map, giving a $lidar\_points$ list of coordinates on the local map. Another list, $assoc\_points$ is used to represent the nearest obstacle on the local map to each entry of $lidar\_points$. Using both lists, $grads$, a vector of the partial derivatives of cost with respect to x, y, θ and s is computed using Equations 6 to 9. The partial derivatives are used to update $uncorr\_state$ to $corrected\_state$ using the update equation shown in Equation 10. Finally, $corrected\_state$ is added to the state trajectory and also passed to the next iteration as the 'previous state'.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A. Experimental Hardware and Set-up

Experiments were conducted on a wheelchair equipped with a front-facing TIM551 2D LiDAR with a 270° field of view. Motor encoders are used for odometry. The Robot Operating System (ROS) is used to facilitate the collection of data and replaying the data under different test conditions.

In the experiment, the wheelchair is driven around an indoor corridor at about 1m/s. 10 bags of data were collected, with the wheelchair performing a different maneuver in each bag. The bags recorded include observing passers-by, moving on a straight corridor, U-turn, reversing, 3-point turn, zig-zag motion, passing back-and-forth a door, spinning on the spot, and traveling at a higher speed of 2m/s.

Localization is performed on the 10 bags of data using both our SGD localization algorithm and Adaptive Monte

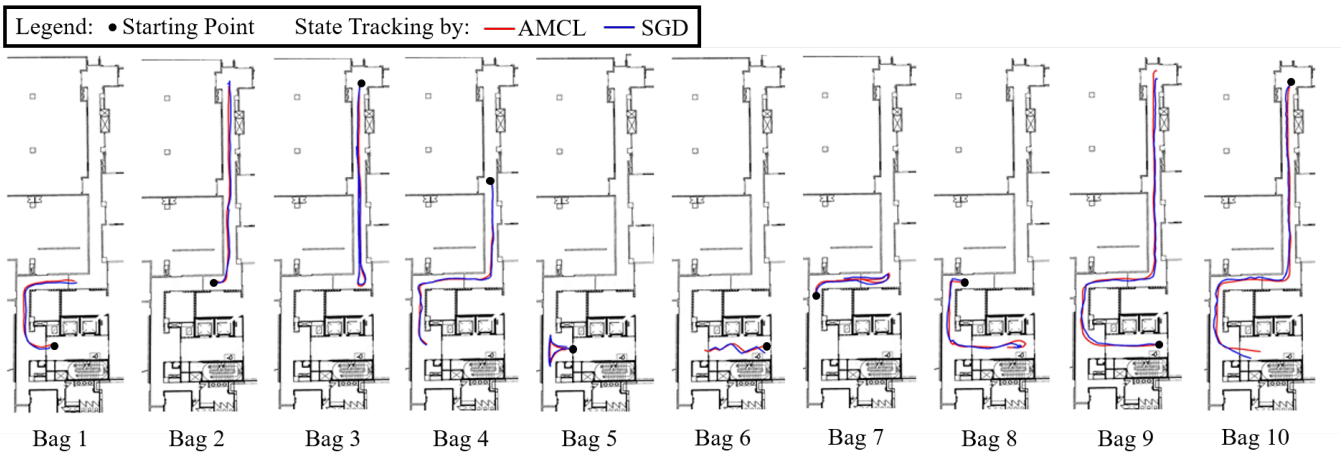

Fig. 5. State trajectories of AMCL and SGD on a floor plan as input map. State trajectory produced by both SGD state tracker and AMCL are still relatively consistent despite minor perspective distortion.

TABLE I
POINT CLOUD MAP LOCALIZATION ERROR

Test Condition	SGD Avg. dist. error/m	SGD Avg. θ error/°
Bag 1	0.316	4.58
Bag 2	0.232	2.53
Bag 3	0.271	2.75
Bag 4	0.336	2.77
Bag 5	0.193	2.57
Bag 6	0.161	11.26
Bag 7	0.156	5.04
Bag 8	0.141	11.80
Bag 9	0.264	5.55
Bag 10	0.417	5.48
On Average	0.264	5.26

Carlo Localization (AMCL) [20]. Both SGD and AMCL algorithms are challenged to localize the 10 bags of data on three different input maps - a point cloud map generated by SLAM, an image of a floor plan, and a hand-drawn map.

B. Localization on a Point Cloud Map

Figure 4 shows the trajectories formed by the states tracked via both SGD (blue) and AMCL (red) when a point cloud map generated by SLAM is used as the input map. This map is to scale with a resolution of $0.05m/px$. Under this input, the SGD Algorithm is set to have a x, y learn rate of 5, θ learn rate of 1×10^{-3} and s learn rate of 3×10^{-7}.

In this map, AMCL localization works extremely well and is taken as the ground truth in localization. Table I shows the position and heading errors of the SGD algorithm compared to AMCL.

Comparing this result with a visual inspection of Figure 4, our proposed SGD algorithm works well in a point cloud input map with an average positional error of 0.264m and heading error of 5.26°, performing better than GLFP that has an average positional error of 0.4m. The abnormally large heading error stands out in Bags 6 and 8, but is explained by the robot rapidly spinning or turning in both bags.

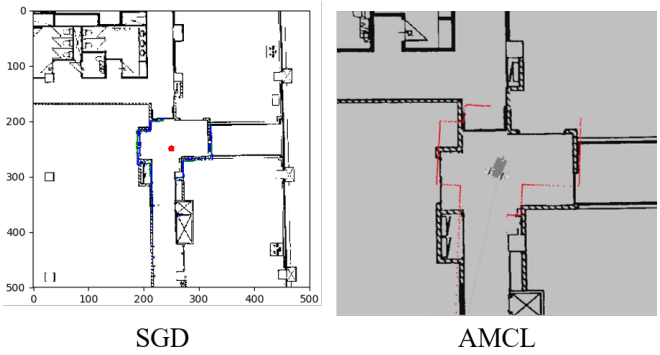

Fig. 6. End states of AMCL and SGD on bag 9 using a floor plan. Green pixels represent LiDAR points of SGD while the red dots are the LiDAR points for AMCL.

C. Localization on a Minorly Skewed Floor Plan

The state trajectories of both SGD and AMCL using an image of a floor plan as an input map is shown in Figure 5. This floor plan is displayed in the lobby and a picture is taken of it. The image has perspective distortion, skewing the floor plan, causing the bottom to appear slightly larger. As AMCL only accepts a fixed scale, a resolution of $0.0175m/px$ is used. The SGD Algorithm is set to have a x, y learn rate of 15, θ learn rate of 2×10^{-3} and s learn rate of 3×10^{-7}.

In the floor plan, AMCL still works decently as details such as bumps in the wall are still present and the severity of the perspective distortion is small. The trajectories in Figure 5 between AMCL and SGD are relatively consistent. However, comparing the end states between both algorithms in bag 9 as shown in Figure 6, the LiDAR scan of AMCL in bag 9 is less consistent with the floor plan than that of our SGD algorithm. This is likely because the AMCL state trajectory has overshot the true endpoint due to the perspective distortion, causing the true resolution of the top portion of the map to be larger than the $0.0175m/px$ used.

In Table II, the states tracked by SGD and AMCL are less consistent than in a point cloud map. The average positional

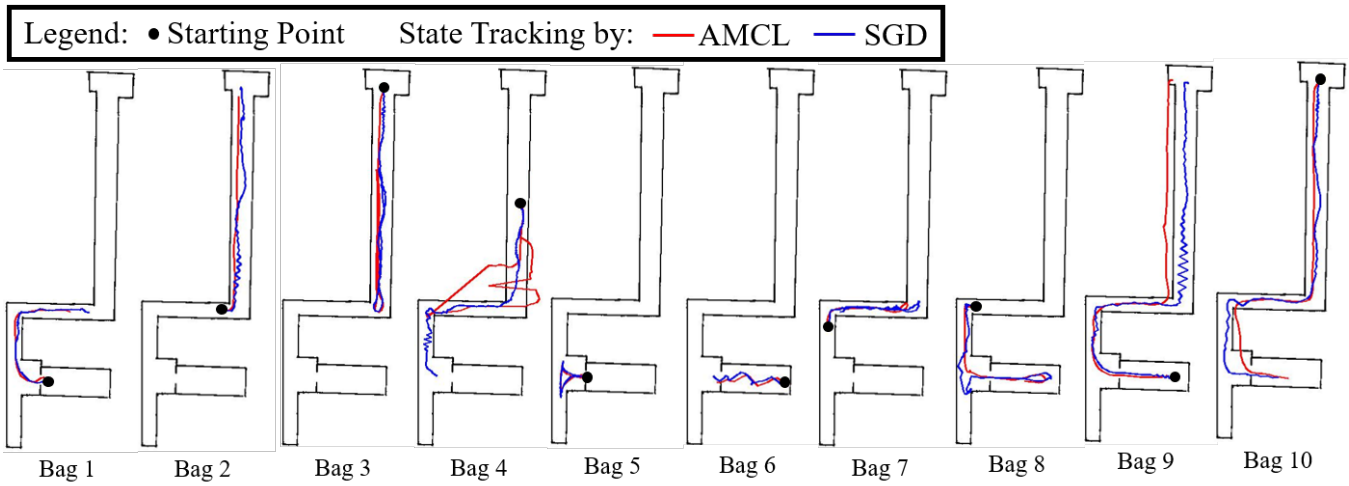

Fig. 7. State trajectories of AMCL and SGD on a hand-drawn input map. SGD state tracker still has a sensible state trajectory albeit jittery. AMCL loses localization in bag 4 and runs into walls in bags 9 and 10.

TABLE II
FLOOR PLAN LOCALIZATION DIFFERENCE

Test Condition	SGD Avg. dist. diff./m	SGD Avg. θ diff./°
Bag 1	0.590	6.05
Bag 2	0.403	3.94
Bag 3	0.539	3.82
Bag 4	0.256	2.85
Bag 5	0.467	8.49
Bag 6	0.516	12.28
Bag 7	0.584	5.98
Bag 8	0.747	13.69
Bag 9	0.735	9.21
Bag 10	1.03	8.62
On Average	0.598	6.88

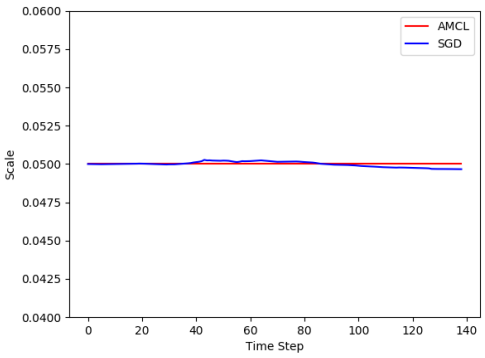

Fig. 8. Scale against time on point cloud map with bag 9. Scale remains relatively constant in a scaled point cloud map.

and heading differences are larger at 0.598m and 6.88° respectively, as the map is no longer well suited for AMCL.

D. Localization on a Hand-Drawn Map

Figure 7 shows the state trajectories of SGD and AMCL on a hand-drawn map of the same environment. Although a human can easily tell that the hand-drawn map represents the same area as the previous maps, many localization algorithms find this challenging as the map is now poorly scaled with many features missing. The corridors are either drawn too long/short/narrow/wide and the small area at the top end of the corridor is also drawn with the wrong shape. Despite these major imperfections, we would still want a sensible state trajectory on the hand-drawn map. For this input map, the AMCL is set to a resolution of $0.08m/px$ while our SGD algorithm is tuned to have a x, y learn rate of 5, θ learn rate of 1×10^{-3} and s learn rate of 5×10^{-6}.

From Figure 7, AMCL often fails in such a map. This is most apparent in Bag 4, where AMCL has completely lost its localization and is attempting to constantly re-localize. In bags 9 and 10, AMCL has traveled into forbidden areas. On the other hand, our SGD algorithm faithfully follows the topology of the hand-drawn map. However, it is not without flaws. When the corridor on the map suddenly changes and appears too wide, our SGD algorithm tends to produce a jittery trajectory while it learns the new scale of the corridor. This can be observed in bags 2, 4 and 9. In bag 8, a false trajectory also appears while the robot is spinning. While these flaws can potentially cause problems to motion planners in autonomous vehicles, the SGD algorithm eventually learns and recovers from them. Furthermore, these shortfalls can also be mitigated by imposing a motion constraint on the robot to smoothen these trajectories.

E. Impact of Varying Scale

A main contributing factor for the success of our SGD algorithm in remaining resilient in improper maps is its scale state, which allows it to adapt to local distortions in the scale of the map.

Figures 8 and 9 shows how the scale of our SGD localization algorithm changes against time when running bag 9 on the point cloud map and the hand-drawn map respectively. When running on a point cloud map which is to scale, Figure 8 shows that our SGD algorithm indeed kept its scale relatively constant to navigate the accurate map.

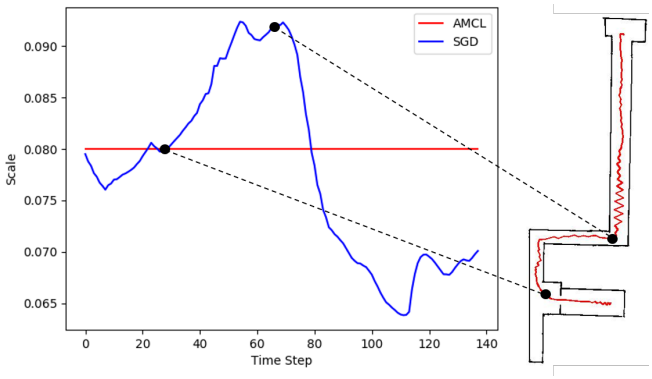

Fig. 9. Scale against time on hand-drawn map with bag 9. Scale increases around time step 30 to compensate a narrowly drawn corridor and falls after time step 70 when the robot turns into a corridor drawn too wide.

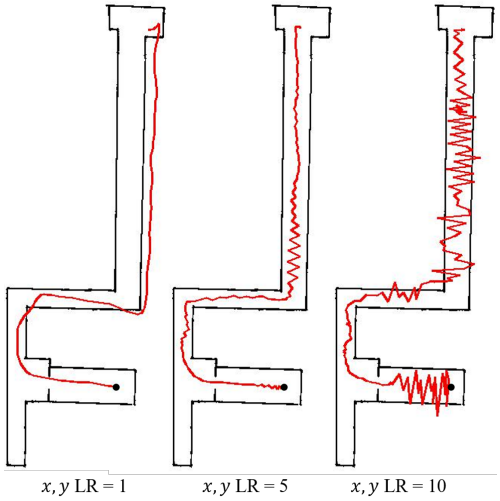

Fig. 10. SGD Bag 9 state trajectory with different x, y learn rates. Trajectory is not properly corrected with low x, y learn rate and becomes extremely jittery when set too high.

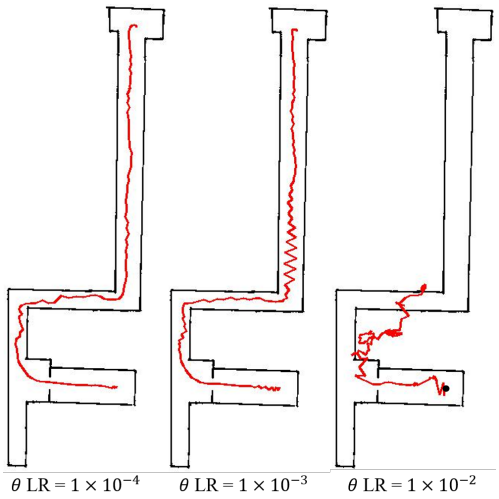

Fig. 11. SGD Bag 9 State Trajectory with Different θ Learn Rates. State tracker loses localization if θ learn rate is set too high. When θ learn rate is set lower, the trajectory appears fine, but there is lack in angular learning as shown in Figure 12.

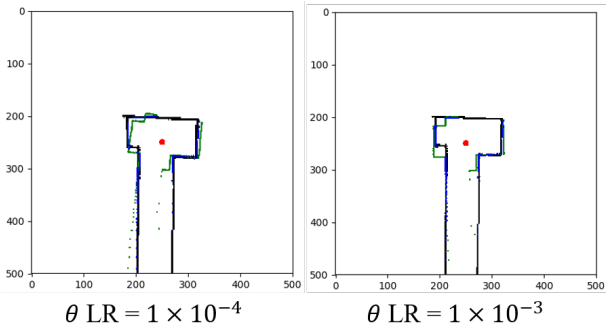

Fig. 12. SGD Bag 9 End State comparison between a small and the optimal θ Learn Rate. The green pixels represent the LiDAR scan when projected onto the local frame.

For the hand-drawn map, the scale is expected to vary widely along with the inconsistent scale of the map, as seen in Figure 9. At around time step 30, the robot performs its first right turn and travels into a set of two corridors that are drawn too narrow. As such, the scale increases sharply so that each pixel in the narrow corridor represents a larger area to make it consistent with what is observed in reality. At about time step 70, the opposite happens, the robot performs a left turn to the corridor that is drawn too wide, and hence the scale sharply decreases to compensate.

F. Analysis of Learn Rates

As part of our SGD algorithm, three learn rates, the x, y learn rate, θ learn rate and s learn rate must be tuned for every new input map. A grid-search approach is used to identify the best combination of learning rates to use. This section describes the observations made that help fine-tune the learning rate by demonstrating on the hand-drawn map.

The optimal learn rates for the hand-drawn map was x, y learn rate of 5, θ learn rate of 1×10^{-3} and s learn rate of 5×10^{-6}. Figure 10 shows the state trajectories of the SGD algorithm on Bag 9, using the optimal, small, and large x, y learning rates. When the x, y learn rate is set to 1, there is not enough positional correction to allow the SGD algorithm to conform to the input map. The state trajectory runs into the wall before the algorithm can correct itself. When the x, y learn rate is set too high at 10, major corrections occur even for minor inconsistencies, resulting in an extremely jittery trajectory that also runs into the walls.

When θ learn rate is set too large in Figure 11, the SGD state tracker loses localization the instant it turns the corridor as θ changes too drastically. When θ learn rate is set too small, an illusion of better state trajectory with less jittering is observed. However, the quality of the LiDAR scan matching is compromised as shown in Figure 12, a small θ learn rate could not properly correct its heading effectively. Instead, the smooth trajectory is a result of the algorithm correcting its position to compensate for the poor angular learning.

As for scale, as seen in Figure 13, if s learn rate is set too low, the algorithm responds very slowly to corridors that suddenly appears too big or small, resulting in persistent

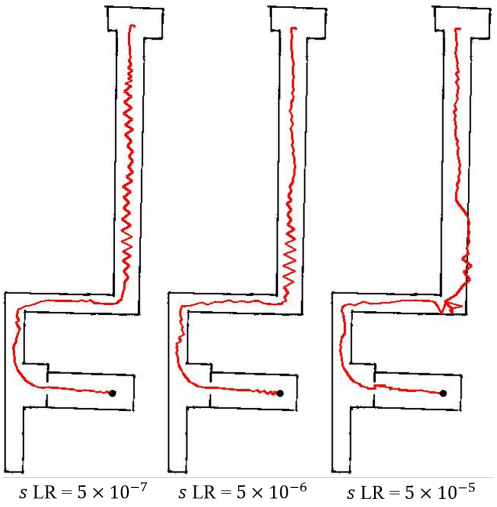

Fig. 13. SGD state trajectory with different s Learn Rates. Low s learn rates cause persistent jittering, high learn rates can cause loss in localization.

jittering that diminishes very slowly over time, as opposed to the optimal θ learn rate where the jittering diminishes quickly as the tracker adapts to the new scale more quickly. However, setting θ learn rate too large can cause s to change drastically when a new corridor is observed, resulting in the temporary loss of localization seen in Figure 13.

V. CONCLUSION

We proposed an SGD state tracking algorithm that is robust to poorly scaled maps and do not use pre-indicated features. The algorithm uses the sum of squared distances between LiDAR scan points and its associated obstacles as a cost function and performs one iteration of descent with every new LiDAR scan. Experimental results show good consistency of the SGD state tracker with AMCL with 0.264m and 5.26° average position and heading error respectively in a scaled point cloud map. In a hand-drawn map, SGD state tracking remained robust while AMCL often fails to produce a sensible state trajectory.

The SGD state tracker can be improved in the future. Although it is robustness to few dynamic obstacles, the algorithm is unsuitable for crowded environments. The learn rates needs to be re-tuned if the operating environments drastically changes. Finally, the algorithm also currently cannot re-localize, and hence uncorrected state estimates cannot stray too far from the true state.

ACKNOWLEDGMENT

This research was supported by the National Research Foundation, Prime Minister's Office, Singapore, under its CREATE programme, Singapore-MIT Alliance for Research and Technology (SMART) Future Urban Mobility (FM) IRG. We also gratefully acknowledge the technical support of Nvidia Corporation through the Memorandum of Understanding with the Advanced Robotics Centre of the National University of Singapore on autonomous system technologies.

REFERENCES

[1] M. W. M. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, June 2001.

[2] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: a factored solution to the simultaneous localization and mapping problem," in *Eighteenth national conference on Artificial Intelligence*, Edmonton, Alberta, Canada, July 2002, pp. 593–598.

[3] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, June 2007.

[4] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[5] X. Wang, R. J. Marcotte, and E. Olson, "GLFP: Global localization from a floor plan," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China, Nov. 2019, pp. 1627–1632.

[6] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian, "IntentionNet: Integrating planning and deep learning for goal-directed autonomous navigation," in *1st Annual Conference on Robot Learning, CoRL 2017*, Mountain View, California, USA, Nov. 2017, pp. 185–194.

[7] P. Karkus, D. Hsu, and W. S. Lee, "Integrating algorithmic planning and deep learning for partially observable navigation," *ArXiv*, vol. abs/1807.06696, July 2018.

[8] J. Zhang and S. Singh, "LOAM: Lidar odometry and mapping in real-time," in *Robotics: Science and Systems Conference (RSS)*, July 2014.

[9] ——, "Visual-lidar odometry and mapping: low-drift, robust, and fast," in *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, Washington, USA, May 2015, pp. 2174–2181.

[10] M. E. E. Najjar and P. Bonnifait, "A road-matching method for precise vehicle localization using belief theory and kalman filtering," *Autonomous Robots*, vol. 19, no. 2, pp. 173–191, Oct. 2005.

[11] S. Thrun, W. B. D. Fox, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, no. 1-2, pp. 99–141, 2001.

[12] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, pp. 1004–1020, Aug. 2018.

[13] V. John, Z. Liu, S. Mita, and Y. Xu, "Stereo vision-based vehicle localization in point cloud maps using multiswarm particle swarm optimization," *Signal, Image and Video Processing*, vol. 13, no. 4, pp. 805–812, 2019.

[14] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *IJCAI'03 Proceedings of the 18th international joint conference on Artificial intelligence*, Acapulco, Mexico, Aug. 2003, pp. 1151–1156.

[15] T. Ort, L. Paull, and D. Rus, "Autonomous vehicle navigation in rural environments without detailed prior maps," in *IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, May 2018, pp. 2040–2047.

[16] A. Amini, G. Rosman, S. Karaman, and D. Rus, "Variational end-to-end navigation and localization," in *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Quebec, Canada, May 2019, pp. 8958–8964.

[17] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard, "A tree parameterization for efficiently computing maximum likelihood maps using gradient descent," *Robotics: Science and Systems*, 2007.

[18] E. Olson, J. Leonard, and S. Teller, "Fast iterative alignment of pose graphs with poor initial estimates," in *IEEE International Conference on Robotics and Automation (ICRA)*, Orlando, Florida, USA, May 2006, pp. 2262–2269.

[19] J. Biswas, B. Coltin, and M. Veloso, "Corrective gradient refinement for mobile robot localization," in *IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011, pp. 73–78.

[20] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, pp. 99–141, 05 2001.