Robot Crowd Navigation through Predictive Pedestrian Modeling

Sebastian Peralta, Raymond Bjorkman, Zachary Fischer

***Abstract*—The issue of seamlessly navigating dynamic environments such as such as crowds of pedestrians is an open research area and one of the bottlenecks to large-scale deployment of delivery service robots. In this paper we propose a local path planning algorithm that generates probabilistic models of the future, thus allowing service robots to temporally reason about challenging environments. We implement and deploy this algorithm on a TurtleBot in a simulation environment and evaluate its efficacy in different crowd scenarios. Compared to a traditional A* method which we use as a benchmark, we find that our planner results in fewer crashes and more efficient paths across a variety of tests.**

I. Introduction

The COVID-19 pandemic has laid bare the demand for robust automated delivery technology; however, significant advances still need to be made. Current robots that would navigate densely populated areas are often slow, careful, and wait for humans to move out of their paths. This behavior makes them difficult to be accepted by the people around them as this overly cautious behavior impedes the flow of foot traffic. For a paradigm shift in this regard, robots should be able to take into account the patterns of movement around it to seamlessly integrate with its environment. Successfully navigating crowds will also result in faster delivery times and higher productivity of service robots. Additionally, a stronger emphasis on understanding the identities of moving obstacles and their trajectories will offer the robot better insight into a wider range of situations.

In the traditional path planning literature, dynamic obstacles such as pedestrians are considered as being instantaneously static [1, 2]. Modeling them in this way is computationally simpler, but results in the robot is only being able to make instantaneously optimal decisions with regard to navigation. If a robot were instead to track these dynamic obstacles over time, it could reason about their future pose and plan optimally over a short time horizon. To this end, we propose a new method for local path planning that is able to reason about path options in terms of an estimated collision probability and make decisions according to a user-defined trade off between safety and efficiency. This algorithm is implemented as part of a full robot stack which we then evaluate in simulation. While the performance of the stack is evaluated holistically, we treat the local path planning module as the primary contribution of this paper.

II. Related Work

Currently there are delivery robots in deployment like the Kiwibot (an outdoor delivery robot being deployed in San Francisco) and the Savioke Relay (being deployed indoors in hospitals and hotels). While these have been somewhat successful in their areas, it is not clear how the software governing these robots would fare in more complicated scenarios, such as in a congested space. Moreover, these robots are likely content with safe navigation rather than concerning themselves with efficiency. One such possible inefficiency is falling victim to the "robot freezing problem", when robots stop when there are people around even if they are not obstructing the robots path.

One potential solution is the IGP (Interaction Gaussian Process) model in which joint cooperation between the robot and agents are included. However live operation requires the robot to replan at around 10 Hz and the inference methods of IGP are computationally prohibitive [1]. Thus, current research has been aimed at discovering more efficient inference methods. Furthermore the same authors have discovered that the freezing problem persists even with perfect prediction of the agents' trajectories [2].

There have been many approaches to solve the more general problem of planning in dynamic environments. The method of safe interval path planning proposes a planner with resolution complete optimal paths with safety guarantees. However, this method is also computationally costly and does not scale well with the high number of dynamic obstacles in real world crowds [3]. Perhaps the most promising application in this space has been the effort to develop an autonomous wheelchair that uses predictive modeling to interact with the environment [4]. Public demonstrations of this system are encouraging however not much else has been revealed.

III. Methods

The below subsections document the major elements of our software stack at a high level. For demonstrations or a more detailed examination, our implementation is publicly available and can be viewed at `https://github.com/dazednconfusing/CrowdNav`.

A. Simulation Architecture and Environment

For our simulation environment, we used PedSim, which is an open source code that helped simulate how a crowd of pedestrians would travel in a multitude of environments. This was useful as it helped give an environment where our robot could then sense and plan its trajectory around both static and dynamic obstacles. Furthermore it allowed for the quick creation of scenarios with tunable pedestrian behavior, including the modeling of social forces between pedestrians

to emulate crowd behavior [5]. The one drawback was that it worked exclusively through RViz, so in order to interface it with other modules we linked it into a Gazebo environment.

In this Gazebo enviornment we also launched an instance of a virtual TurtleBot3 [6], our choice of test platform for this research. Using the ROS node associated with it we queried its sensors for LIDAR scans and controlled its position by going through its low level controller.

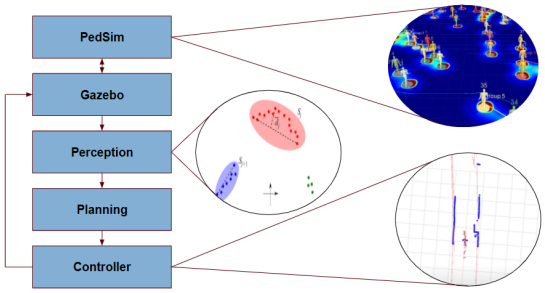

Fig. 1. A high level flowchart of the system architecture. The PedSim, Perception and Controller modules as highlighted as they all represent third party implementations that were integrated into this project.

B. Perception

We are using an open-source package for object detection and tracking based on 2D laser scan data [7]. This package transforms and rectifies the scan for sensor motion and can also merge scans from multiple sensors if this project were ever scaled to a larger robot. Next, the obstacle extraction is performed based on the geometry of the laser points and classified as either a segment or circle. In our case, this simplification did not pose any challenges and was able to accurately distinguish between walls and pedestrians. Over time, a cost matrix is developed to track the disparity between obstacles sets relative to the robot. A Kalman filter is used to sustain the existence of the tracked dynamic obstacles (circles) in the absence of scan data. This results in a data structure containing walls represented by their end-points and pedestrians represented with a position, radius, and velocity. This information was then integrated into the navigation with some parameter tuning to find an ideal balance between accuracy and sensor updates.

C. Planning Algorithm

To generate plans between two locations a hierarchical planner was employed. A global planner running the A* algorithm was used as the top level of the planner. The plans created by this algorithm are then used to guide the lower level local planner which is the focus of this paper and explained in more detail in the subsections below

1) Pedestrian Modeling: The output of the perception module is passed into the planner and corresponds to the position x_0, velocity v_0, and variance σ_0 of nearby pedestrians. This information is then used to make a prediction of the pedestrian's location for times in the near future using the kinematic equation below.

$$x(t) = v_0 t + x_0 \tag{1}$$

Because this equation represents a crude estimation of pedestrian behavior and does not take into account their agency; they are able to suddenly turn or stop and therefor we need to be able to account for this uncertainty. To accomplish this we treat each pedestrian as a random variable with a Gaussian probability distribution $\mathcal{N}(x(t), \sigma(t))$ associated with them. This means that in expectation the pedestrian will follow the prediction given from equation (1) and will have a time varying variance described by the equation below.

$$\sigma(t) = f(t) + \sigma_0 \tag{2}$$

The term $f(t)$ can be chosen to chosen to be any monotonically increasing function and represents the degradation in our confidence that a pedestrian is in a given location.

2) Path Sampling and Collision Probability: We use a sampling-based approach in the style of RRT to generate a set of paths $\mathcal{P}$. In traditional planning literature paths are considered valid if they do not collide with an obstacle; however, we can no longer concretely reason about collisions since we are considering dynamic obstacles to be probabilistic. Therefore we instead seek to evaluate through their probability of collision.

Each path-pedestrian pair has a time varying collision probability. While it might be desirable to evaluate the collision probability over all time, this is an intractable problem. Instead, we seek to find the maximum collision probability for each path-pedestrian pair. This is still a difficult problem to find the true solution to so we make another simplification and instead look to find the point in time at which the robot and pedestrian in question are nearest, t_{min}, then calculate the collision probability at this time. This simplification always represents a conservative approximation of the true collision probability. We can derive an analytical solution t_{min} if we plan with paths through space that we can represent with continuous functions. This is done by minimizing the distance between the robot and pedestrian as shown in the equations below.

$$d(t)^2 = \|x_{ped}(t) - x_{robot}(t)\|_2^2 \tag{3}$$

$$d'(t)^2 = 2(x_{ped}(t) - x_{robot}(t))(x'_{ped}(t) - x'_{robot}(t)) \tag{4}$$

Next we set equation (4) equal to zero and solve for t to find the local minima. We can arrive at t_{min} by clamping the value of t in the range$(0, t_{horizon})$, where $t_{horizon}$ is the maximum time that the planner is predicting in the future.

For simplicity the planner we implemented in this paper only generates paths that follow a straight line but this technique could easily be extended to consider more complicated paths provided that some solution for t_{min} exists.

Once t_{min} is found, the probability of collision for a single pedestrian is found by integrating the 2D Gaussian distribution

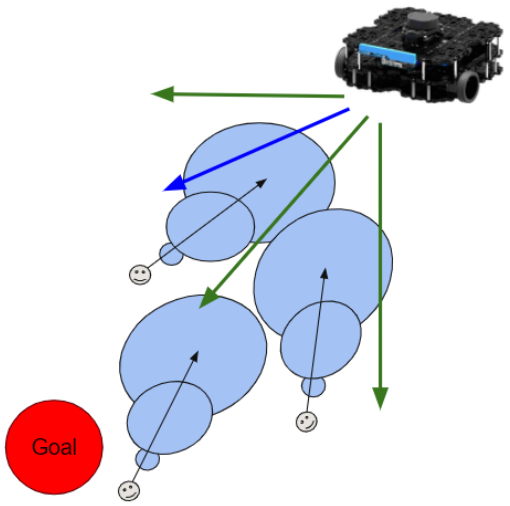

Fig. 2. A cartoon depiction of the local planner. Candidate paths not chosen are shown in green, with the blue path being the cost optimal path. This example illustrates that the robot might decide that some amount of risk is acceptable in order to more quickly reach the goal and therefore it chooses the blue path even though there are paths that have a lower probability of collision.

for the pedestrian at t_{min} over the volume of the inflated robot. The total probability of collision for a path is:

$$1 - \prod_{\text{all pedestrians}} (1 - P_{collision}) \tag{5}$$

3) Evaluating Candidate Paths: In order to evaluate the set of candidate paths, each path is weighted according to a cost function. This cost function balances two objectives: safety (as determined through collision probability) and being goal-oriented. The path that is finally selected is the one that satisfies the optimization below.

$$\min_{p \in \mathcal{P}} \|x_{goal} - x_{p,robot}(t_{horizon})\|_2 + \lambda P_p \tag{6}$$

The first term in this cost function accrues cost for paths that end far from the global waypoint x_{goal} (received from the A* planner). This incentivizes being goal-oriented and efficiently moving towards the goal. The second term corresponds to the collision probability discussed above. It is multiplied by the cost bias λ. This variable is a user-controllable parameter that tunes the planner to either favor safety by weighting paths with higher collision probabilities higher, or efficient travel by placing a bigger emphasis on travelling in the most direct manner. It is recommended that both objectives are normalized so that values of λ have some intuitive meaning in the form of a percentage weight.

D. Controller

The chosen path is then sent to the robot control module. Our implementation utilized the Service Robot Control Stack to accomplish this [8]. The controller implemented by this package was a PID controller for a non-holonomic mobile robot such as the TurtleBot3 we used. There were several parameters that required tuning in order to get desirable performance.

Notably, the controller that we used did not control the robot along the paths we planned to travel along. Since there is some turning radius associated with steering the TurtleBot3, the trajectories were arcs rather than being the idealized straight lines the local planner produced. This made it difficult to track the desired position. One modification we made was introducing another parameter the *forward rotation scalar* to account for the discrepancy between the robot's planned path and the path it followed.

IV. Experimental Evaluations

Several scenarios were created to specifically target and exacerbate important navigation characteristics. The following data tables include a control as well as three additional dynamic environments (upstream, cross-stream, and 2-way). While more scenarios were created and one can be seen in Figure 6, they include compounded traffic patterns that are information qualitatively but were less repeatable and focused. The shortest possible distance the robot can travel in each of the scenarios with recorded metrics is 19 meters. We found that the relative speed between the robot and pedestrians is also critical for successful evasion and ensured that the robot's maximum possible speed is slightly larger than the pedestrians' maximum.

For all the following experiments the "base planner" refers to Kyle Vedder's controller, global A* planner, and a basic collision escape style local planner [8]. The "new planner" is the controller with the local planner that was detailed in Section III.

A. Control Scenario

In order to ensure expected robot behavior, we performed PID tuning and other general adjustments in an empty environment to verify that the robot could travel in a straight line without issue and that there were minimal differences between the planners. The metrics for this scenario can be seen in Table I.

TABLE I
No Obstacles

Metric	Base Planner	New Planner
Total Time (s)	25.50	25.24
Time Stopped (s)	0.595	1.86
Distance Traveled (m)	19.544	19.60

B. Aware Pedestrians

With some modifications to the open-source pedestrian simulator used, the robot could be easily included or removed from the pedestrians' artificial potential field calculations. This change effectively made the pedestrians either aware of the robot or blind to it. This section examines the case of aware pedestrians that is believed to be a closer representation of real human behavior.

TABLE II
UPSTREAM: AWARE PEDESTRIANS

Metric	Base Planner	New Planner
Total Time (s)	35.45	32.04
Time Stopped (s)	0.72	0.57
Distance Traveled (m)	19.57	19.40

TABLE III
CROSS-STREAM: AWARE PEDESTRIANS

Metric	Base Planner	New Planner
Total Time (s)	33.42	31.44
Time Stopped (s)	0.52	0.61
Distance Traveled (m)	19.57	19.89

TABLE IV
2-WAY: AWARE PEDESTRIANS

Metric	Base Planner	New Planner
Total Time (s)	36.05	34.56
Time Stopped (s)	0.70	0.45
Distance Traveled (m)	19.58	19.42

While it is interesting to see how our planner compares to the base planner with aware pedestrians, we can see in Tables II, III, and IV that the results are comparable. This is largely due to the fact that aware pedestrians also are contributing to avoiding collisions rather than putting the onus solely on the robot. Furthermore, an idealized robotic navigation strategy should be as noninvasive as possible to pedestrian motion. While this result is promising for deployment of robots around real pedestrians, we next sought to rigorously stress test our algorithm by using blind pedestrians.

C. Blind Pedestrians

With pedestrians now unaware of the robot but still aware of each other, we ran tests again to compare the base planner with our planner. With this study, we can see how well the robot actively tries to navigate through and around crowds without the safety net of the pedestrians' obstacle avoidance.

In this section, snapshots of the robot navigating the blind pedestrians highlight the scenario structures and traffic flows. The light green circle at the bottom of the environment is the robot's start, the red circle near the top is the goal, and the robot is represented by a coordinate frame with a thick red line as its front. The long blue line extending from the robot is the calculated optimal path that the robot will pursue locally. Pedestrians can be seen as dark green figures with arrows depicting their velocity; they may also have green circles under them when within the robot's sensor range to visualize their estimated location. While all of these depictions are shown through RViz for a clearer understanding, the agents are operating in Gazebo for realistic simulation.

We added a new metric in Tables V, VI, and VII called Failure Rate. With blind pedestrians, it is quite difficult to navigate since they do not avoid the robot. We noticed in most of our simulation runs that the robot would either collide with the pedestrians, or that it took so long to complete that

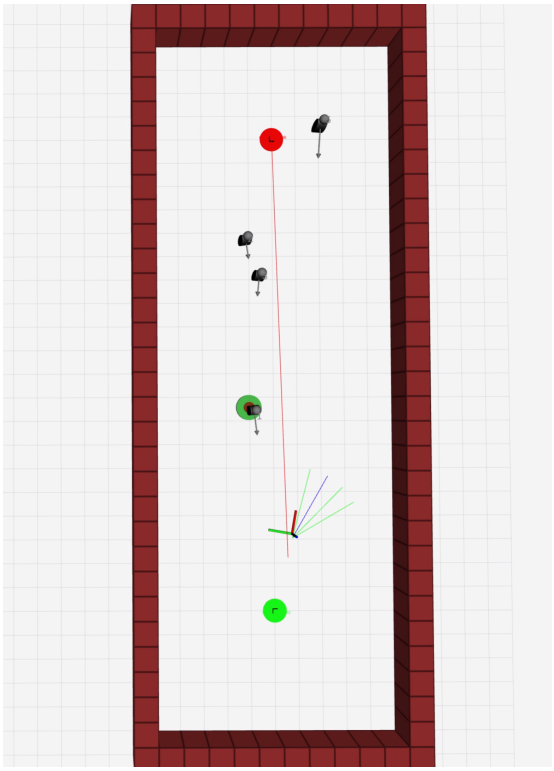

Fig. 3. Blind Peds: Upstream Environment

TABLE V
UPSTREAM: BLIND PEDESTRIANS

Metric	Base Planner	New Planner
Failure Rate (%)	80	10
Total Time (s)	67.82	43.46
Time Stopped (s)	26.62	0.54
Distance Traveled (m)	31.54	24.18

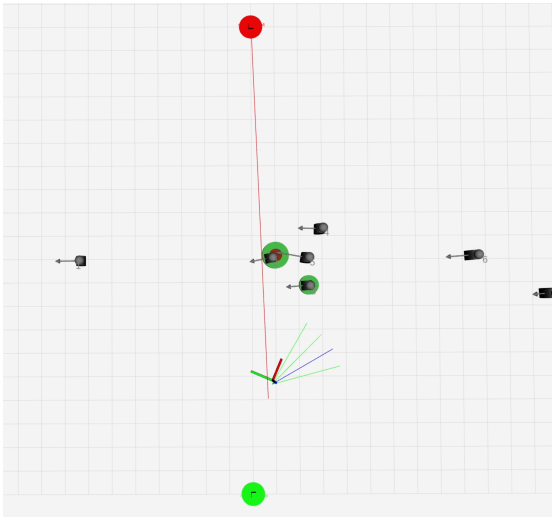

Fig. 4. Blind Peds: Cross-Stream Environment

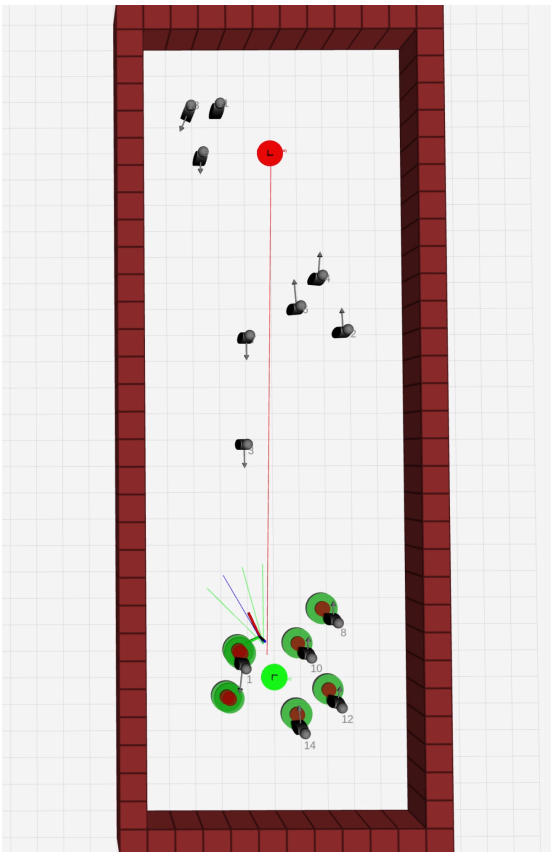

Fig. 5. Blind Peds: 2-Way Environment

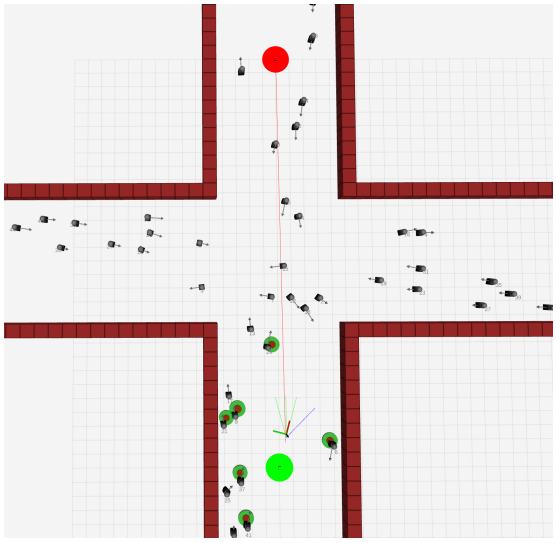

Fig. 6. 4-Way Environment

TABLE VI
CROSS-STREAM: BLIND PEDESTRIANS

Metric	Base Planner	New Planner
Failure Rate (%)	50	10
Total Time (s)	39.47	34.74
Time Stopped (s)	6.93	0.61
Distance Traveled (m)	19.59	20.96

TABLE VII
2-WAY: BLIND PEDESTRIANS

Metric	Base Planner	New Planner
Failure Rate (%)	20	10
Total Time (s)	45.86	47.91
Time Stopped (s)	2.66	0.41
Distance Traveled (m)	20.88	22.06

we listed it as a "Did Not Finish". With this new metric, we can see that the base planner failed often compared to the 0% failure rate in the aware pedestrian scenarios as well as compared to the planner we implemented. In fact, we can see that not only did our planner have a lower failure rate, it also performed better than the base planner, which is most noticeably seen in Table V. In Tables VI and VII, while the original metrics are comparable, we can see that the failure rate and the time stopped are vastly different. The reason the time stopped is different is that instead of staying in place, we forced the robot to move, so it greatly reduced our time spent stopped while increasing the distance travels by a minuscule amount.

D. Pareto Frontier Optimization

Our cost function seeks to optimize two objectives: the time it takes for a robot to reach its objective and the percent of runs that result in a collision. To better understand how parameters of our algorithm affected performance, we collected metrics for 10 runs at each parameter configuration in the two-way hallway scenario. Time of navigation was averaged over the 10 runs.

The first parameter we examined was the cost bias. When selecting the optimal path, the cost bias is the scalar factor that multiplies the probability of a collision occurring in the cost function. The second parameter investigated was the scalar multiplier for the forward velocity if the robot needs to rotate. A higher cost bias in principle makes the robot select more conservative paths at the expense of making less progress towards the goal. A higher forward velocity scalar makes the robot travel faster but at the expense of deviating from the selected optimal path.

The results of the Pareto frontier optimization can be seen in Fig. 7, 8. If we require the collision percentage to be 10% or less, the parameters (20, 0.25) were found to be optimal. However, if we allow the collision percentage to be 20% or less, the parameters (20, 0.5) were found to be optimal.

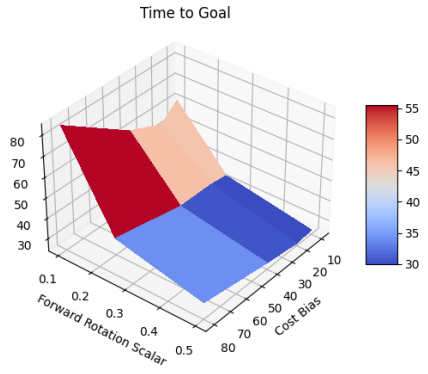

Fig. 7. Visualization of the Pareto frontier for the path efficiency objective.

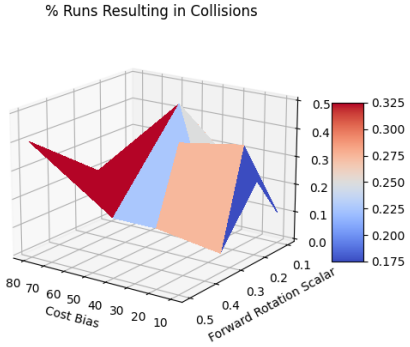

Fig. 8. Visualization of the Pareto frontier for the collision avoidance objective.

V. Conclusion

When comparing the controller and planner that we have implemented to the base planner, we can see that significant improvements were made. As we hoped, we were able to overcome the freezing issue by traveling a bit more distance than the minimal path, but reducing the time taken most of the time. Although our failure rates were around 10%, we were able to navigate around blind pedestrians much better than the base planner.

This system is subject to several limitations, however, which we highlight as potential areas for future work. First, sudden pedestrian velocity change is difficult to account for with our simplistic model of increasing their variance. Future work should seek to find better models for pedestrian probability to reduce the failure rate even further. One could also attempt to find a feasible solution for the more computationally difficult problem of exactly calculating the maximum probability of collision (rather than simplifying the problem by calculating it at t_{min}) as this would provide the planner with more accurate information to act on. Making improvements to the object detection module such as increasing the range and improving velocity estimations could also help as it would also provide the planner with more reliable information resulting in a lowering the failure rate.

On the planning side, a big improvement would be considering the controller and planner as a single entity, and planning solely in terms of dynamically feasible trajectories. This would render parameters such as the forward rotation scalar unnecessary and would greatly improve the system by simplifying it. Another direction could involve learning a more flexible model to calculate collision probability with a neural network. Finally, our local planner lacks any ability to plan nonlinear routes over a longer time horizon, which can lead to it getting trapped in local minima as pedestrians unexpectedly can converge on its location. Imbuing the planner with some forward thinking through a more intelligent global planner would be beneficial to avoid these situations.

References

[1] Trautman, P. et al. “Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation.” The International Journal of Robotics Research 34 (2015): 335 - 356.

[2] P. Trautman and A. Krause, ”Unfreezing the robot: Navigation in dense, interacting crowds,” 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, 2010, pp. 797-803, doi: 10.1109/IROS.2010.5654369.

[3] V. Narayanan, M. Phillips and M. Likhachev, ”Anytime Safe Interval Path Planning for dynamic environments,” 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, 2012, pp. 4708-4715, doi: 10.1109/IROS.2012.6386191.

[4] Park, J. J.. “Graceful Navigation for Mobile Robots in Dynamic and Uncertain Environments.” (2016).

[5] Okal, Billy and Linder, Timm. ”Pedestrian Simulator”. 2015. github.com/srl-freiburg/pedsim_ros.

[6] ROBOTIS-GIT. ”TurtleBot3”. 2020. github.com/ROBOTIS-GIT/turtlebot3.

[7] Przybyla, Mateusz. ”Obstacle Detector”. 2018. github.com/tysik/obstacle_detector.

[8] Vedder, Kyle. ”ServiceRobotControlStack”. 2020. github.com/kylevedder/ServiceRobotControlStack.