Online Learning for Crowd-sensitive Path Planning

Robotics Track

Anoop Aroor
The Graduate Center, City University of New York
aaroor@gradcenter.cuny.edu

Susan L. Epstein
Hunter College, City University of New York
susan.l.epstein@hunter.cuny.edu

Raj Korpan
The Graduate Center, City University of New York
rkorpan@gradcenter.cuny.edu

ABSTRACT

In crowded environments, the shortest path for an autonomous robot navigator may not be the best choice — another plan that avoids crowded areas might be preferable. Such a *crowd-sensitive* path planner, however, requires knowledge about the crowd's global behavior. This paper formulates a Bayesian approach that relies only on an onboard range scanner to learn a global crowd model online. Two new algorithms, CUSUM-A* and Risk-A*, use local observations to continuously update the crowd model. CUSUM-A* tracks the spatio-temporal changes in the crowd; Risk-A* adjusts for changes in navigation cost due to human-robot interactions. Extensive evaluation in a challenging simulated environment demonstrates that both algorithms generate plans that significantly reduce their proximity to moving obstacles, and thereby protect people from actuator error and inspire their trust in the robot.

KEYWORDS

path planning; robot navigation; online learning; human crowds

ACM Reference Format:
Anoop Aroor, Susan L. Epstein, and Raj Korpan. 2018. Online Learning for Crowd-sensitive Path Planning. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

Now that robots have begun to serve as museum tour guides [18], telepresence robots [21], and assistants in hospitals and offices [24], autonomous navigation in crowds has become an increasingly important problem. Crowds can slow, divert, or halt the robot, and thereby increase its *navigation costs* (travel time and distance). A service robot should not only travel economically, however. It must also respect humans' personal space and safety, even as its very presence may attract some people and repel others. The thesis of our work is that a Bayesian model for the robot's experience facilitates online learning about crowd behavior and can support crowd-sensitive planning. This paper reports on two new methods tested extensively in a challenging simulated environment. The principal results reported here are statistically significant improvements in the robot's ability to maintain a safe distance from pedestrians while it travels among them.

Local planners generate short-term plans focused on the robot's immediate vicinity, while *global planners* generate complete end-to-end plans. A common approach to crowds initially generates a global plan that ignores them, and then adjusts that path locally with human-aware planners [11]. Given a graph of nodes that represent locations and edge weights that represent distances, the A* algorithm finds a shortest path [9]. Because it ignores the costs of navigation through a crowd, however, such a plan may prove globally inefficient. For example, the robot in Figure 1 has two plans; the uninformed plan is shorter but ignores the crowd, while the crowd-sensitive plan is less direct but aware of the likely congestion ahead. One approach might be to anticipate these global costs through a dataset of end-to-end human trajectories observed by external sensors (e.g., wall-mounted or overhead cameras) [26]. This is impractical or infeasible in many environments.

Instead, we consider here how a robot limited to only an onboard 2D range sensor can learn a *cost map*, a grid-based spatial model of global navigation costs in an indoor environment. This paper describes two methods that learn a cost map and plan from it. The first method detects dynamic changes in crowd-movement patterns; the second records the impact of the robot's presence on those patterns.

Over time, crowds behave differently. A shopping mall crowd, for example, is likely to be larger on some evenings and weekends, as well as for holidays or special events. Such spatio-temporal changes make navigation costs dynamic. To plan effectively, a robot must learn and update a crowd model that records where people are likely to be found. The first method, CUSUM-A*, learns a crowd-density cost map that records the average number of people in each grid cell per unit of time. It uses CUSUM, a statistical change detector [16], to track spatio-temporal changes in crowd density that are likely to impact navigation cost.

Crowd behavior also changes when a robot is present [15, 20]. An action is deemed *risky* if it brings the robot within some specified

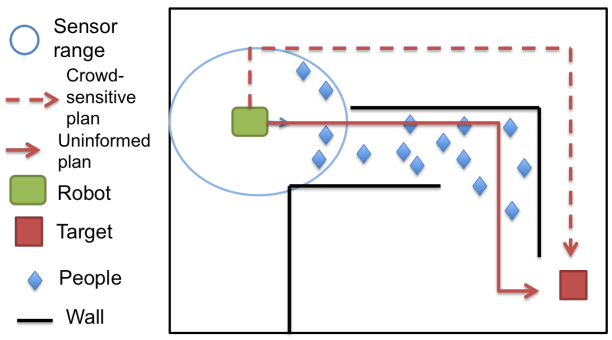

Figure 1: Because A* lacks knowledge of global crowd costs, it travels through the crowd

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

distance of a person. Some people make way for a robot; others approach it, hindering its motion. This changes *risk*, the frequency with which the robot comes unacceptably close to a person. In a shopping mall, for example, areas that attract children might incur more risk than other crowded areas, despite similar crowd density. The second method, Risk-A*, combines crowd density and observations of human-robot interaction to build a cost map that records the level of risk in each grid cell.

This paper applies both approaches to path planning in crowded indoor environments. The algorithms, CUSUM-A* and Risk-A*, learn their cost maps online, as the robot travels through a crowd. CUSUM-A* uses its cost map to plan crowd-sensitive paths that respond to dynamic crowd density, while Risk-A* responds to human-robot interaction effects. The next two sections discuss related work and provide background on robot navigation, the cost maps, and Bayes filters. Subsequent sections describe CUSUM-A* and Risk-A*, their implementation in ROS, and their integration into a cognitively-inspired robot controller. These are followed by the experimental design, empirical results, and a discussion.

2 RELATED WORK

Early robot research used sensors to detect local obstacles in the environment and then sought to avoid them with local planners; it did not predict obstacles' future motion [8]. Many recent methods make local, short-term predictions about the future trajectories of moving obstacles and use them to improve collision avoidance. These approaches have predicted human trajectories and planned a path around them with a Gaussian process [19], with neural networks [1], or with bio-mechanical turn indicators [22]. One local path planner learned reward functions on data from human experts who controlled the robot [12]. Yet another approach used pedestrian trajectory datasets to learn a model that jointly predicted the trajectories of both a robot and nearby pedestrians, and then generated socially compliant paths [13]. All of these human-aware planners improved collision avoidance and navigation, but they were restricted to local scenarios where the robot could completely observe its surroundings. In contrast, the work reported here addresses a complementary problem: how to learn global navigation costs in a given crowded environment and then use them to improve global path planning.

Other work has made global, long-term predictions about the behavior of a crowd, and adapted navigation behavior accordingly, typically with end-to-end pedestrian trajectories. One approach treated a single trajectory as a Markov decision process, learned a distribution over trajectories, applied inverse reinforcement learning to find the reward function that best fit those trajectories, and used it to predict new ones [26]. Another approach used an end-to-end simulated pedestrian trajectory dataset to initialize a Gaussian-process-based model, updated it from local sensor observations, and then used inverse reinforcement learning to make the robot's behavior more human-like [10]. In contrast, the approach proposed in this paper learns online and does not require end-to-end pedestrian trajectory datasets.

The online learning problem addressed here was originally proposed in [2], which also introduced the CSA* algorithm. CSA* learns a cost map from local data collected by range sensors as it moves through a simulated environment. For each cell in a grid superimposed on the footprint of the environment, CSA* calculates a running average for crowd density and records it in a *crowd density map*. CSA*, however, does not work well in environments where crowd-movement patterns change over time, and it ignores the impact of the robot's presence on human behavior. Both are necessary considerations in real-world navigation, addressed here with a novel Bayesian formulation. The resultant algorithms, CUSUM-A* and Risk-A*, prove effective in scenarios where crowd-movement patterns change over time, and incorporate human-robot interaction effects to learn more accurate cost maps for global navigation.

3 BACKGROUND

Our approach makes several fundamental assumptions. The robot's laser range sensors are mounted at a uniform level near the floor, and it has a two-dimensional map of its environment with static features. The robot can *localize*, that is, it knows its *pose* (location and orientation) with respect to an allocentric coordinate system. Within its sensor range, the robot can also detect *local crowd data*, the location and orientation of each person [14].

The robot receives an ordered set of *targets* ($< x, y >$ coordinate pairs) in its environment. Its task is to *reach* (arrive at a location less than 0.5m from) these targets in their specified order. Moreover, the robot should do so quickly, travel relatively short distances, avoid collisions, and take relatively few risks. We also assume that the robot, as it visits its targets, learns online about the crowd, without separate phases for learning and testing.

For a crowd density map, each cell is modeled as a Poisson distribution with rate λ, both because the crowd density is always non-negative and because empirical observations in the simulator indicated Poisson was plausible. The likelihood that a crowd of size z is present in an individual grid cell for a fixed duration is thus

$$P(CrowdSize = z|\lambda) = \frac{\lambda^z e^{-\lambda}}{z!} \tag{1}$$

Within a given grid cell, let $z_{1:n}$ represent the sequence of n observations $z_1, z_2, ..., z_n$ made by the robot. Each observation z_i represents the number of people detected from the range scan data. To estimate the value of λ for a particular grid cell from $z_{1:n}$, we assume that $z_1, z_2, .., z_n$ are conditionally independent given λ. This yields a recursive Bayesian filter that computes $P(\lambda|z_{1:n+1})$ given $P(\lambda|z_{1:n})$ and a new observation z_{n+1}, with normalization constants η, as follows.

$$P(\lambda|z_{1:n}) = \eta_1 P(z_{1:n}|\lambda)P(\lambda) \quad \text{(Bayes rule)} \tag{2}$$

$$\begin{aligned} P(\lambda|z_{1:n+1}) &= \eta_2 P(z_{1:n+1}|\lambda)P(\lambda) && \text{(Bayes rule)} \\ &= \eta_2 P(z_{1:n}, z_{n+1}|\lambda)P(\lambda) \\ &= \eta_2 P(z_{1:n}|z_{n+1}, \lambda)P(z_{n+1}|\lambda)P(\lambda) \\ &= \eta_2 P(z_{1:n}|\lambda)P(z_{n+1}|\lambda)P(\lambda) \\ &= \eta_3 P(\lambda|z_{1:n})P(z_{n+1}|\lambda) && \text{(by (2))} \end{aligned} \tag{3}$$

Let $P(\lambda)$ be a Gamma distribution with (prior) parameters α and β, and let $P(z_{n+1}|\lambda)$ be the Poisson distribution (evidence). Then given

$$P(\lambda|z_{1:n}) = \Gamma(\alpha + \sum_{i=1}^{n} z_i, \beta + n)$$

the update rule that tracks the crowd density in a given cell is

$$\begin{aligned} P(\lambda|z_{1:n+1}) &= P(\lambda|z_{1:n})P(z_{n+1}|\lambda) && \text{(by (3))} \\ &= \Gamma(\alpha + \sum_{i=1}^{n} z_i, \beta + n) * (\frac{\lambda^{z_{n+1}} e^{-\lambda}}{z_{n+1}!}) && \text{(by (1))} \\ &= \Gamma(\alpha + \sum_{i=1}^{n} z_i + z_{n+1}, \beta + n + 1) \end{aligned}$$

Thus the new values of α and β after a new observation z_{n+1} are readily computed from the old values of α and β, and the expected value of λ is the ratio of α and β:

$$\alpha_{new} = \alpha_{old} + z_{n+1} \quad (4)$$

$$\beta_{new} = \beta_{old} + 1 \quad (5)$$

$$E(\lambda) = \frac{\alpha}{\beta} \quad (6)$$

For a given grid cell, update rules (4) - (6) mean that the crowd density can be estimated as the running average of observations there, the same way CSA* computes it. This assumes, however, that λ is constant, which is not the case in most real-world environments. The next section explains how CUSUM can monitor changes in λ and take corrective action whenever change occurs.

4 CUSUM-BASED CROWD DENSITY MAP

CUSUM is an online statistical change detector. Given a sequence $z_1, z_2, .., z_n$ of observations of a random variable Z that follows some probability distribution, CUSUM detects a change in Z's distribution parameter θ . Because $z_1, z_2, .., z_n$ come from a probability distribution, some variation is to be expected. What CUSUM seeks to identify, however, is a persistent and significant change.

Assume that a change in the distribution parameter happens before the observation z_c. Let θ_0 be the distribution parameter before that change and θ_1 the distribution parameter after it. CUSUM tracks the *summation score*, the sum of the log-likelihood ratios. For observations before z_c, the likelihood ratio $\frac{p(z_i|\theta_0)}{p(z_i|\theta_0)}$ is 1, and the log-likelihood ratio $ln\frac{p(z_i|\theta_0)}{p(z_i|\theta_0)}$ is 0. For observations from z_c onward, the likelihood ratio is $\frac{p(z_i|\theta_1)}{p(z_i|\theta_0)}$, and the log-likelihood ratio is $ln\frac{p(z_i|\theta_1)}{p(z_i|\theta_0)}$. If the observations match the changed distribution parameter θ_1, the log-likehood ratio will eventually become positive. CUSUM uses the summation score to guarantee that a change has occurred. Because the precise instant c of the change is unknown, CUSUM assumes c is the instant when the summation score is maximized:

$$G[n] = \max_{1 \le c \le n} \sum_{i=c}^{n} \ln \frac{p(z_i|\theta_1)}{p(z_i|\theta_0)} \quad (7)$$

To be certain that the change at c is significant, CUSUM requires that $G[n]$ exceed some threshold τ.

To detect a change in the distribution of the crowd density in a given cell, we take θ_0 as λ_0 and θ_1 as λ_1, where λ_0 and λ_1 are the parameters of the Poisson distribution for that cell before and after the change. By substitution from (1), the instantaneous log-likelihood ratio $s[i]$ of i^{th} observation is

$$\begin{aligned} s[i] &= \ln \frac{p(z_i|\lambda_0)}{p(z_i|\lambda_1)} \\ &= \ln(\frac{\lambda_0^{z_i} e^{-\lambda_0}}{z_i!}) - \ln(\frac{\lambda_1^{z_i} e^{-\lambda_1}}{z_i!}) \\ &= z_i \ln(\frac{\lambda_0}{\lambda_1}) - (\lambda_0 + \lambda_1) \end{aligned}$$

A *cumulative sum* $S[j]$ from 1 to j is defined as

$$S[j] = \sum_{i=1}^{j} s[i]$$

$S[j]$ can be used to reformulate $G[n]$ as a minimization problem, a common CUSUM technique for more efficient computation:

$$G[n] = S[n] - \min_{1 \le j \le n} S[j-1]$$

Before the robot travels, α is initialized to 0 and β to 1 for each cell. Whenever CUSUM detects a change in the crowd recorded for a cell, the parameters of that cell are reinitialized. Given a current observation z_n, Algorithm 1 estimates the crowd density map with CUSUM, where γ is a discount factor in (0,1).

5 RISK MAP

In real environments, where people behave differently in the presence of a robot, the crowd density map computed in Algorithm 1 is not an accurate reflection of the true cost of navigation. If, for example, people consistently make way for a robot as it travels, the crowd density map would overestimate navigation difficulty. Instead, we estimate the expected number of risky actions in a given grid cell per unit time as a proxy for the cost of navigation through that cell. The expected risky action rate λ_r in a grid cell depends

Algorithm 1: CUSUM crowd density map($grid, \gamma, \tau, z_n$)

```
/* Initialize                                          */
for each cell in grid do
    (α, β, S[0], SMIN^old) = (0, 1, 0, 0);
end
/* Online update step                                  */
for each cell in grid do
    α^new = (α^old * γ) + z_n;
    β^new = (β^old * γ) + 1;
    S[n] = S[n − 1] + z_n ln(λ0/λ1) − (λ0 + λ1);
    SMIN^new = min(S[n − 1], SMIN^old);
    G[n] = S[n] − SMIN^new;
    /* Reset if CUSUM detects change                   */
    if G[n] ≥ τ then
        α^new = 0;
        β^new = 1;
    end
    E(λ) = α^new / β^new;
end
```

both on the crowd density there, as estimated from the laser range data, and on the crowd behavior when a robot is present in that cell. This section describes an online algorithm for learning λ_r.

To examine risky actions across time, for each grid cell we count the *risk experience* a_i, the number of risky actions the robot took when it moved through that cell for the i^{th} time. Let $a_{1:k}$ be the sequence of risk experiences that the robot observes in the same cell from time 1 to time k. We assume that the occurrence of one risky action does not influence the next, given the crowd density in that cell. We then model the risky actions in any given grid cell as a Poisson distribution with rate λ_r. To learn λ_r, let $z_{1:n}$ be the sequence of crowd counts in that grid cell based on laser scans, and let λ be the estimated crowd density there. If we assume that λ_r is conditionally independent of $z_{1:n}$ given λ, the probability $P(\lambda_r|\lambda, z_{1:n}, a_{1:k})$ of risky actions after k experiences is equal to $P(\lambda_r|\lambda, a_{1:k})$. We can then compute $P(\lambda_r|\lambda, a_{1:k})$ recursively, again with normalization constant η, as follows.

Let the prior $P(\lambda_r|\lambda, a_{1:k})$ be a gamma distribution with parameters λ and c, where λ is the current crowd density estimate for the cell and c indicates how much crowd density affects risky actions. Then, given a new experience a_{k+1} where the robot moves through a crowd in that cell, we can construct a Bayes filter by reasoning similar to the derivation in the previous section:

$$P(\lambda_r|\lambda, a_{1:k}) = \eta_1 P(a_{1:k}|\lambda, \lambda_r)P(\lambda_r|\lambda) \tag{8}$$

$$\begin{aligned} P(\lambda_r|\lambda, a_{1:k+1}) &= \eta_2 P(a_{1:k+1}|\lambda, \lambda_r)P(\lambda_r|\lambda) \\ &= \eta_2 P(a_{1:k}, a_{k+1}|\lambda, \lambda_r)P(\lambda_r|\lambda) \\ &= \eta_2 P(a_{1:k}|a_{k+1}, \lambda, \lambda_r)P(\lambda_r|\lambda)P(a_{k+1}|\lambda, \lambda_r) \\ &= \eta_2 P(a_{1:k}|\lambda, \lambda_r)P(\lambda_r|\lambda)P(a_{k+1}|\lambda, \lambda_r) \\ &= \eta_3 P(\lambda_r|\lambda, a_{1:k})P(a_{k+1}|\lambda_r) \end{aligned} \tag{9}$$

with risky-action update rule:

$$\begin{aligned} P(\lambda_r|\lambda, a_{1:k+1}) &= \Gamma(\lambda + \sum_{i=1}^{k} a_i, c + k) * (\frac{\lambda_r^{a_{k+1}} e^{-\lambda_r}}{a_{k+1}!}) \\ &= \Gamma(\lambda + \sum_{i=1}^{k} a_i + a_{k+1}, c + k + 1) \\ E(\lambda_r) &= \frac{\lambda + \sum_{i=1}^{k} a_i}{c + k} \\ E(\lambda_r) &= \frac{\frac{\sum_{j=1}^{n} z_j}{n} + \sum_{i=1}^{k} a_i}{c + k} \end{aligned} \tag{10}$$

Given current observations for crowd density z_n and risky action count a_k, Algorithm 2 estimates the *risk map* with equation (10), where γ is a discount factor in (0,1), α_r^{new} replaces the sum on a_k, and β_r^{new} counts k. This approach balances the evidence collected from the laser range scanner about crowding in a given location against the realized cost of navigation. Given a cell where the robot has no risk observation a_k, the algorithm depends on evidence from the laser scanner to estimate navigation cost. As the robot gathers more evidence about how crowds in that cell behave, it progressively adjusts the risky action count. When crowds vary in their response to a robot, this should make risk-sensitive planning more robust than CUSUM alone.

Algorithm 2: Risk map($grid, \gamma, z_n, a_k$)

```
/* Initialize                                          */
for each cell in grid do
    (α, β, α_r, β_r) = (0, 1, 0, 1);
end
/* Given a new Observation z_n                         */
for each cell in grid do
    α^new = (α^old * γ) + z_n;
    β^new = (β^old * γ) + 1;
    E(λ) = α^new / β^new;
end
/* Given a new Risk experience a_k                     */
for each cell in grid do
    α_r^new = α_r^old + a_k;
    β_r^new = β_r^old + 1;
    E(λ_r) = (E(λ) + α_r^new) / (c + β_r^new);
end
```

6 ONLINE CROWD-SENSITIVE PLANNING

Algorithms 1 and 2 both estimate the added costs of navigation incurred by crowds. This paper leverages that knowledge to formulate crowd-sensitive plans with an A* planner. A regular grid is superimposed upon the map, and a weighted graph is built that represents each grid cell as a node. A node in the graph has edges to at most eight cells that adjoin it in the grid. (There are fewer than eight if the cell lies on the border of the grid or if a wall intervenes.) The weight of edge e_{mn} that connects nodes m and n is the Euclidean distance between their centers. A* finds an optimal (shortest) path in this graph.

When presented with its first target in a new environment, a crowd-sensitive robot makes an A* plan in the same weighted graph and retains that plan until it reaches the target. As it travels and observes the crowd, however, the robot updates its cost map (here, either CUSUM's crowd density map or the risk map). Then, before each subsequent target, the robot estimates from its cost map the likely impact of crowds on its navigation, and updates the graph's edge weights. Let the cost recorded for grid cell i in the cost map be c_i. Then the new, crowd-sensitive edge cost to travel from node m to node n is

$$e'_{mn} = e_{mn} + w * ((c_m + c_n)/2)$$

where the parameter w controls the severity of the penalty on the edge due to crowding (here, $w = 0.5$). A* returns a plan that avoids highly crowded areas unless the alternative is a very long path.

CUSUM-A* uses the CUSUM crowd density map with A*, and Risk-A* uses the risk map with A*. Recall that the robot's task is to navigate to a sequence of targets. Initially, when the robot has no information about the crowd, both CUSUM-A* and Risk-A* generate plans identical to those of A*. Then, as the robot gathers information about the crowd, the relevant cost map is continuously updated by Algorithm 1 or Algorithm 2. The computational complexity of both update algorithms is $O(n + q)$, where n is the number of people in the environment and q is the number of grid cells. For fixed n and

q these updates are fast. (We have used as many as 3600 grid cells with no noticeable computation delay.) Once enough information is gathered through map updates, the cost maps force the planner to avoid crowded areas, as the next sections demonstrate.

7 IMPLEMENTATION

7.1 ROS, the Robot Operating System

ROS is the state-of-the-art operating system for robot navigation [17]. Crowd-sensitive planning is implemented here as three interacting ROS nodes, shown in Figure 2.

MengeROS. To simulate crowding and the robot in a single environment, we use *MengeROS* [3], a ROS extension of the open-source crowd simulator *Menge* [4]. MengeROS requires a map of the environment, a robot, and crowd specifications.

SemaFORR. The SemaFORR node controls the robot by sending it actuator commands. It receives the simulated robot's position and laser scan data from the MengeROS node, and returns to the MengeROS node the actuator commands that drive the robot. The MengeROS node then simulates these actions on the robot.

Crowd Learner. The Crowd Learner is a standalone ROS node; it too receives the robot's position and laser scan data as messages from MengeROS. The Crowd Learner uses Algorithm 1 or Algorithm 2 to build a cost map, and forwards the current cost map to the SemaFORR node upon request. This modular implementation is important because it allows the learner node to be used with any other ROS-compatible simulator and any other ROS-compatible controller.

7.2 SemaFORR navigation architecture

In this work, the robot's decision algorithm is *SemaFORR*, a controller for autonomous navigation [7]. SemaFORR is a cognitively-based hybrid architecture that involves both reactive and deliberative reasoning. The deliberative reasoning component generates a plan that is a sequence of intermediate locations (*waypoints*) on the way to the target. As in Figure 3, SemaFORR's input includes the actions available to the robot, its pose and current target, the current laser scan data, and the cost map. Because SemaFORR is based on the FORR cognitive architecture [6], it uses a combination of heuristic procedures called *Advisors* to choose an action. SemaFORR's Advisors form a three-tier hierarchy.

Tier-1 Advisors are reactive decision-making rules that assume perfect knowledge. As a result, they are fast and correct. Each Advisor can either choose an action to execute or eliminate actions

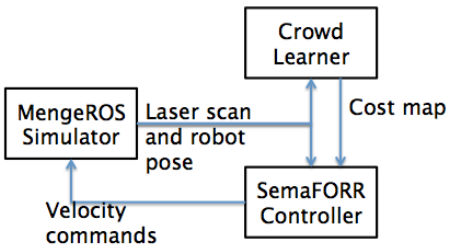

Figure 2: ROS node interactions

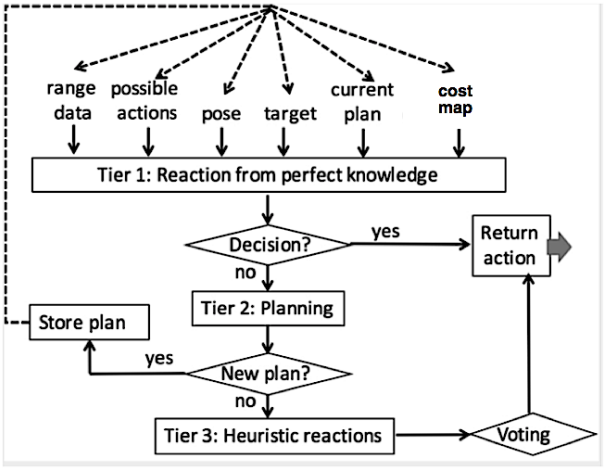

Figure 3: SemaFORR's decision cycle

from further consideration. Victory is a tier-1 Advisor; it chooses the action that gets the robot closest to a target within sensory range when no obstacles block the robot's path. If the robot has a plan to reach the target, at least one of its unvisited waypoints is within sensory range, and no obstacles block the robot's path there, Enforcer chooses the action that best approaches the waypoint closest to the target. Otherwise, AvoidObstacles, another tier-1 Advisor, eliminates actions that would cause a collision. It uses the laser range scan data to remove actions that would bring it too close to static or dynamic obstacles. If only one action remains, it is returned. Otherwise, SemaFORR forwards the remaining actions to the next tiers.

Tier-2 Advisors are the four deliberative planners: A*, CSA*, CUSUM-A*, and Risk-A*. When control reaches tier 2, if there is no current plan, the user-specified planner generates one and forwards it to the plan store and the cycle ends. Otherwise, if a plan is already in place, SemaFORR forwards control to tier 3, which selects an action from the remaining, collision-free, plan-compliant actions.

Each *tier-3* Advisor makes heuristic recommendations based on its own rationale. Given the current plan, tier-3 Advisors treat the next waypoint as if it were the target. For example, Greedy prefers actions that move the robot closer to that waypoint, and Explorer prefers actions that keep the robot away from previously visited areas. To express its preferences, a tier-3 Advisor assigns a numeric value to each possible action that survived tier 1. A voting mechanism aggregates the preferences of all tier-3 Advisors and returns the most preferred action. Further details on SemaFORR are available in [7].

8 EXPERIMENTS

This section describes experiments conducted in simulation to evaluate CUSUM-A* and Risk-A* on navigation through crowded environments. MengeROS simulated both the robot and the crowds. The system's parameters support a broad variety of crowd scenarios, including not only the number of pedestrians in an environment, but also their initial positions, their intended goals, their decision-making strategy, and their collision avoidance strategy. Maps of

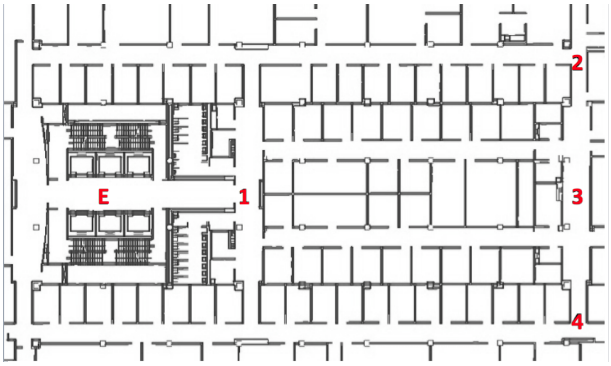

Figure 4: The fifth floor with elevators at E

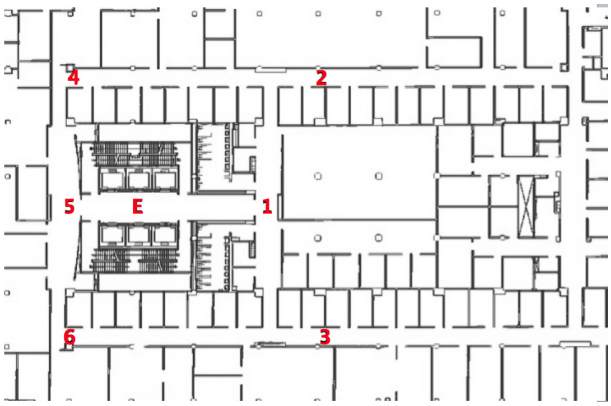

Figure 5: The fourth floor with elevators at E

the fourth and fifth floors of a 110m × 70m urban office building served as the environments. Both floors support alternative routes when crowds obstruct an area.

MengeROS requires that all pedestrians use the same collision avoidance strategy to avoid one another and the robot. In preliminary work we tested two of these, ORCA[23] and PedVO [5], both based on velocity obstacles. The velocity obstacle (*VO*) of a person is the set of all velocity vectors that will result in a collision. Collision-free motion requires that every agent has a velocity vector outside its VO. ORCA has each agent address this problem equally to produce an optimal solution. PedVO adapts ORCA to incorporate such human behaviors as aggression, social priority, authority, and right of way. Because we detected no significant difference in the robot's behavior or performance on our tasks when confronted by PedVO or ORCA crowds, we report here only on ORCA.

In these experiments, MengeROS simulates the footprint and sensor readings of Freight, a standard platform for mobile service robots [25]. Freight has a 2-D laser range scanner with a range of 25m and a 220 degree field of view, a 15Hz update rate, and an angular resolution of 1/3 degree. Each experiment sets a task for Freight in an environment that it shares with a crowd controlled by MengeROS. SemaFORR controls Freight, with a 500-step limit per target, that is, a target is abandoned if the robot has not reached it after 500 decisions. The global cost map is represented as a discrete grid with 2m × 2m cells.

8.1 CUSUM-A* experiment

We compared CUSUM-A* to CSA* and A* in the simulated fifth floor of Figure 4 while the crowd changed its flow pattern across time. Initially, 40 pedestrians begin from the elevators, move along the hallway that follows the arrows in Figure 6, and return to their starting position. Whenever a simulated pedestrian returns to the elevators, it is chosen to repeat the trip with probability 0.98; otherwise, the pedestrian leaves the environment. As a result, the pedestrian population gradually decreases. The robot's task is to visit targets 1, 2, 1, 3, 1, 4, 1 in Figure 4 in that order. This is repeated until the robot has attempted to reach 40 targets. When the robot has addressed about half its targets, a second group of 40 pedestrians enters the environment; they too begin and end at the elevators, but they cycle instead through a smaller inner path, shown by the arrows in Figure 7, and always repeat their trips.

Table 1: CUSUM-A* improves navigation performance

	A*	CSA*	CUSUM-A*
Risky actions	4309.40	3475.67	2635.33
Failures	9.00	11.21	6.50
Distance (m)	2460.16	2518.70	2755.71
Total time (sec)	4452.49	4244.01	4597.19
Mean success time (sec)	99.68	95.00	105.00

This change in crowd pattern, where the first group gradually leaves and another group is suddenly introduced, tests an algorithm's ability to track changes in crowd behavior efficiently. CUSUM's parameters were set to detect a sudden increase in the crowd in any cell. The threshold τ was 10, λ_0 was the current mean arrival rate at the cell, and $\lambda_1 = \lambda_0 - 4$.

Targets were chosen to force the robot to interact with the crowd, and there was always more than one path from one target to the next. We evaluated the robot's performance under A*, CSA* and CUSUM-A* planners by total travel distance, total travel time, total number of risky actions (those that brought the robot within 0.5 meters of a pedestrian), and number of failures (targets the robots failed to reach within its decision limit). The results of the experiment, averaged over 15 trials, appear in Table 1.

Because it has no knowledge of the global cost incurred by the crowds, A*'s plans move through them. Distance traveled was not statistically significantly different with any of the planners. The most noteworthy differences were in risky actions and failure rate; CSA* took fewer risks than A* ($p < 0.001$) and CUSUM-A* took even fewer than CSA* ($p < 0.001$). This is because CSA* relies on a running average for crowd density, and therefore is slower than CUSUM-A* to recognize sudden changes in crowd patterns. Time on task is discussed in the final section.

Figure 6 overlays the 15 plans (one for each trial) generated by A*, CSA*, and CUSUM-A* on the 15th task as the original crowd circles along the arrows. On the 20th task, the second group of pedestrians enters and circles along the arrows in Figure 7. Figure 7 overlays all 15 plans generated by A*, CSA*, and CUSUM-A* on the 29th task. CUSUM-A* plans avoid the crowd flow, while CSA* plans go through the inner hallway, even though it is crowded.

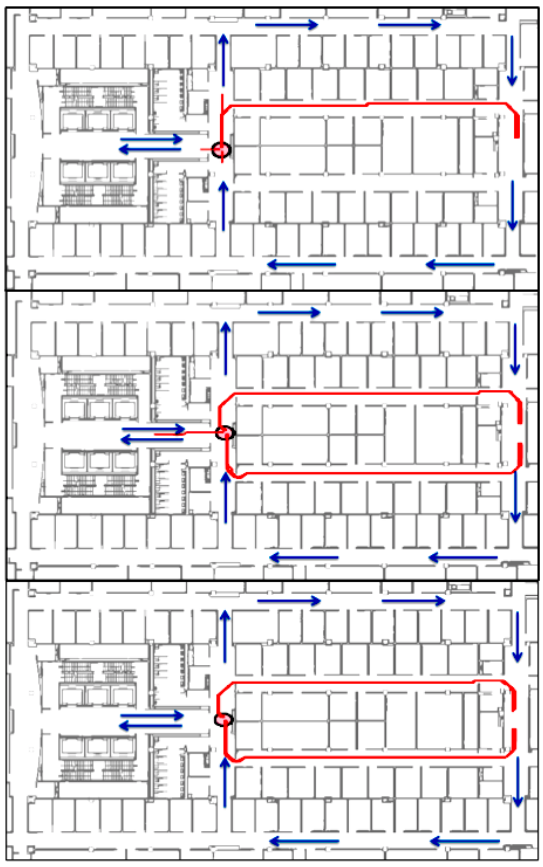

Figure 6: All plans generated by A*(top), CSA*(middle) and CUSUM-A*(bottom) to reach the (circled) 15th target

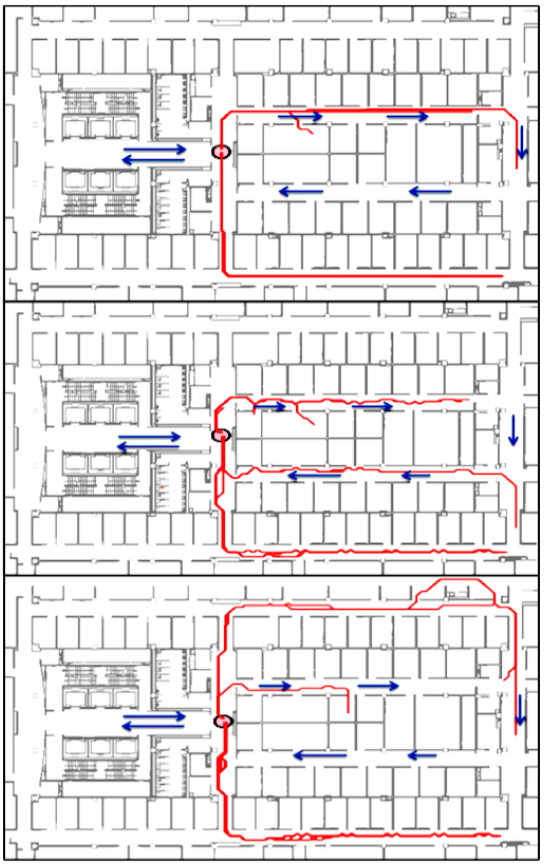

Figure 7: All plans generated by A*(top), CSA*(middle) and CUSUM-A*(bottom) to reach the (circled) 29th target

8.2 Risk-A* experiment

We compared Risk-A* to CSA* and A* in the simulated fourth floor environment in Figure 5. The robot's task here is to visit target positions 2, 5, 3, 4, 1, 6, 2 in Figure 5 in that order until the robot addresses 40 targets. This time, two groups of 35 people move as shown in Figure 8, where the blue arrows describe the first group's path $\langle E, A, B, E\rangle$ and black arrows describe the second's $\langle E, D, C, E\rangle$. Both groups begin and end at the elevators, but they visit different parts of the space. Agents in both groups always repeat their trip when they return to the elevator area; this ensures a steady stream of crowd flow. The important difference between the two groups is that the agents in the second group (black arrows) deviate from their normal path and come closer to the robot. When an agent in the second group comes within 1m of the robot, it generates a new path that brings it closer to the robot. This makes it more difficult for the robot to navigate through the second group.

Risk-A* addresses risky actions in its cost map when it plans. We compared the performance of Risk-A* to A* and to CSA*, which only considers the crowd density estimate when it predicts the cost. The results, averaged over 15 trials, appear in Table 2. There are no statistically significant differences in distance. Again the focus is on risky actions. CSA* takes fewer than A* ($p < 0.001$), and Risk-A*, takes fewer than CSA* ($p < 0.001$). Figure 8 compare the differences in plans generated by A*, CSA*, and Risk-A* as the robot navigates to the 4th target. Risk-A* now mostly generates plans that seek to avoid the second group. Time is discussed in the next section.

Table 2: Risk-A* improves navigation performance

	A*	CSA*	Risk-A*
Risky actions	2394.57	1760.93	1260.64
Failures	6.57	5.21	4.93
Distance (m)	1946.47	1976.50	2199.76
Total time (sec)	4373.02	4654.84	4660.85
Mean success time (sec)	96.76	100.08	108.54

9 DISCUSSION

Because the robot learns online, it can use CUSUM-A* or Risk-A* in a new indoor environment without previous training. This is a significant advantage over methods that learn crowd behavior offline from pedestrian datasets. Another drawback of pedestrian datasets is the assumption that they sufficiently capture future crowd behavior. In most indoor environments, new crowd patterns arise quite frequently. Each time CUSUM-A* resets its model, the robot begins

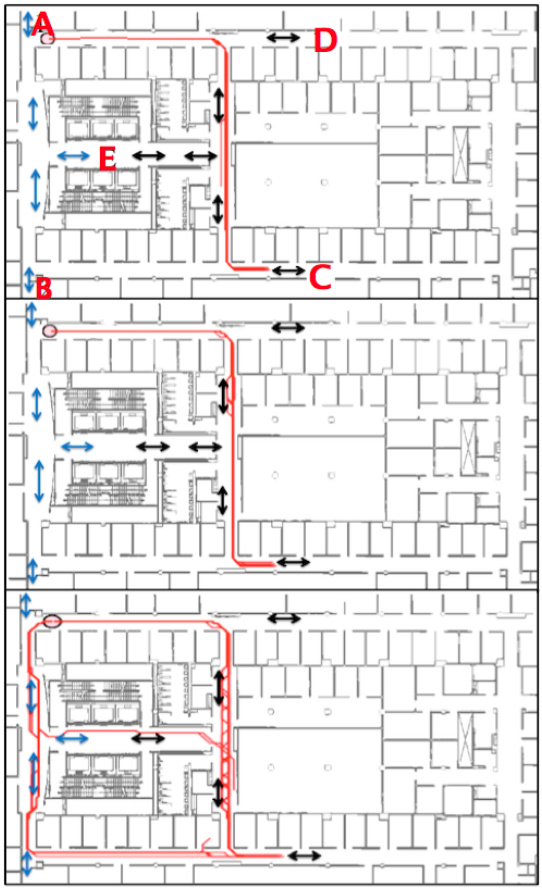

Figure 8: All plans generated by A*(top), CSA*(middle) and Risk-A*(bottom) to reach the (circled) 4th target. Arrows and numeric labels describe crowd behavior. (See text.)

to detect and adjust to these unpredictable changes. Moreover, both CUSUM-A* and Risk-A* are efficient; their runtime complexity is constant for a given map and a pedestrian crowd size. This makes both methods real-time and suitable for onboard processing. Moreover, the algorithms themselves are easy to implement. We tested CUSUM-A* and Risk-A* separately here, to evaluate their individual performance. CUSUM-A* was tested on the 5th floor because it provides more alternative paths than the 4th floor. Realistically, both dynamic changes and human-robot interaction effects can occur simultaneously, and it is possible to apply both algorithms at once. CUSUM would then detect changes in both the crowd density map and the risk map, and adjust edge weights appropriately.

CSA* finishes all 40 tasks more quickly than the other two planners in Table 1, and A* finishes more quickly than the other two in Table 2. The average time only on tasks completely successfully, however, is not statistically significantly different ($p = 0.05$). This is because a robot guided by CSA* in the first experiment or by A* in the second often becomes mired in the crowd. When unable to move, it quickly and repeatedly selects "pause" as its action, until it exhausts its 500 decision-step limit, and fails.

CUSUM-A* detects only sudden increases in the crowd, but it can be easily extended to detect sudden decreases by running a second instance of CUSUM. Although the threshold τ that registers change is hard-coded into CUSUM, that does not limit the size of the crowd that CUSUM can detect. Because each grid cell has its own CUSUM, the arrival of a large crowd that covers multiple grid cells will be detected by multiple CUSUMs, while a smaller crowd that covers fewer grid cells will be detected by fewer CUSUMs. Increased granularity in the cost map would increase the number of CUSUMs and provide more accurate change detection, but it would also incur increased computation cost. CUSUM is tuned to detect change if it collects enough evidence in a given grid cell for a change in crowd dynamics. The threshold for evidence τ can be decreased to make CUSUM highly sensitive to changes.

This work assumes that the robot has access to precise localization and laser scan values. Because the algorithms presented here learn a cost map on a grid, localization or sensor errors will not affect the final cost map, as long as the error in the exact location of a person is within the grid size (here, 2m × 2m). Additional onboard sensors, such as a camera or a Kinnect, could also improve the accuracy of both localization and crowd detection.

The Bayesian framework in Section 3 could also address navigational challenges other than crowds: travel time, travel distance, or energy usage. A learned global cost map applies to other important problems for indoor service robots as well. For example, a service robot that distributes pamphlets or offers directions in a shopping mall should plan a path though the mall's most crowded areas from a global cost map. Current work also includes testing γ values to decay knowledge over time. (These experiments set $\gamma = 1$.)

Both CUSUM-A* and Risk-A* are global planners, but they do not preclude concurrent local collision-avoidance planners. Risk-A* adapts the robot's behavior based on the number of risky actions it has experienced. Since the choice of a local collision-avoidance planner directly impacts the number of risky actions, the choice of a local planner influences the performance of Risk-A*. If a robot uses a local planner that does not perform well in specific areas of the environment, the number of risky actions in those areas would increase, and Risk-A* would make plans that avoid those areas. In this sense, Risk-A* is an example of metacognition.

In summary, path planning in crowded environments is essential for autonomous service robots, but it requires knowledge about the global costs of navigation. Previous approaches that learned the global cost map online assumed simple static crowd patterns [2]. Our novel Bayesian formulation supports two algorithms that learn global cost maps online and are therefore more responsive to the dynamics of a crowded environment. We have tested the algorithms in simulation on complex real-world maps, and demonstrated statistically significant reductions in the number of risky actions, despite realistic changes in the behavior of the crowd.

ACKNOWLEDGMENTS

The authors thank Dr. Olympia Hadjiliadis for suggesting CUSUM to detect changes over time, and Gil Dekel for maps of the environment. This work was supported in part by the National Science Foundation under NSF 1625843.

REFERENCES

[1] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. 2016. Social LSTM: Human Trajectory Prediction in Crowded Spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 961–971. https://doi.org/10.1109/CVPR.2016.110

[2] Anoop Aroor and Susan Epstein. 2017. Toward Crowd-Sensitive Path Planning . In *Proceedings of the AAAI Fall 2017 Symposium on Human-Agent Groups*. 238–245.

[3] Anoop Aroor, Susan Epstein, and Raj Korpan. 2017. MengeROS: A Crowd Simulation Tool for Autonomous Robot Navigation. In *Proceedings of the AAAI Fall 2017 Symposium on AI for HRI*. 123–125.

[4] Sean Curtis, Andrew Best, and Dinesh Manocha. 2016. Menge: A Modular Framework For Simulating Crowd Movement. *Collective Dynamics* 1 (2016), 1–40.

[5] Sean Curtis and Dinesh Manocha. 2012. Pedestrian Simulation Using Geometric Reasoning in Velocity Space. In *Pedestrian and Evacuation Dynamics 2012*, Ulrich Weidmann, Uwe Kirsch, and Michael Schreckenberg (Eds.). Springer International Publishing, Cham, 875–890.

[6] Susan L. Epstein. 1994. For The Right Reasons: The FORR Architecture For Learning In A Skill Domain. *Cognitive Science* 18, 3 (1994), 479–511. https://doi.org/10.1207/s15516709cog1803_4

[7] Susan L. Epstein, Anoop Aroor, Matthew Evanusa, Elizabeth I. Sklar, and Simon Parsons. 2015. Learning Spatial Models for Navigation. In *Proceedings of the 12th International Conference on Spatial Information Theory - Volume 9368 (COSIT 2015)*. Springer-Verlag New York, Inc., New York, NY, USA, 403–425. https://doi.org/10.1007/978-3-319-23374-1_19

[8] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. 1997. The Dynamic Window Approach To Collision Avoidance. *IEEE Robotics & Automation Magazine* 4, 1 (1997), 23–33.

[9] P. E. Hart, N. J. Nilsson, and B. Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (July 1968), 100–107. https://doi.org/10.1109/TSSC.1968.300136

[10] P. Henry, C. Vollmer, B. Ferris, and D. Fox. 2010. Learning To Navigate Through Crowded Environments. In *2010 IEEE International Conference on Robotics and Automation*. 981–986. https://doi.org/10.1109/ROBOT.2010.5509772

[11] Harmish Khambhaita and Rachid Alami. 2017. Assessing the Social Criteria for Human-Robot Collaborative Navigation: A Comparison of Human-Aware Navigation Planners. In *Proc. IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. 6p.

[12] Beomjoon Kim and Joelle Pineau. 2016. Socially Adaptive Path Planning in Human Environments Using Inverse Reinforcement Learning. *International Journal of Social Robotics* 8, 1 (2016), 51–66.

[13] Henrik Kretzschmar, Markus Spies, Christoph Sprunk, and Wolfram Burgard. 2016. Socially Compliant Mobile Robot Navigation via Inverse Reinforcement Learning. *The International Journal of Robotics Research* 35, 11 (2016), 1289–1307.

[14] A. Leigh, J. Pineau, N. Olmedo, and H. Zhang. 2015. Person Tracking and Following with 2D Laser Scanners. In *IEEE International Conference on Intelligent Robots and Systems*. 726–733. https://doi.org/10.1109/ICRA.2015.7139259

[15] Tatsuya Nomura, Takayuki Uratani, Takayuki Kanda, Kazutaka Matsumoto, Hiroyuki Kidokoro, Yoshitaka Suehiro, and Sachie Yamada. 2015. Why Do Children Abuse Robots?. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts (HRI'15 Extended Abstracts)*. ACM, New York, NY, USA, 63–64. https://doi.org/10.1145/2701973.2701977

[16] Ewan S Page. 1954. Continuous inspection schemes. *Biometrika* 41, 1/2 (1954), 100–115.

[17] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: An Open-Source Robot Operating System. In *ICRA workshop on open source software*. ICRA, 5.

[18] Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin B Cremers, Frank Dellaert, Dieter Fox, Dirk Hahnel, Charles Rosenberg, Nicholas Roy, Jamieson Schulte, et al. 1999. MINERVA: A Second-Generation Museum Tour-Guide Robot.. In *IEEE International Conference on Intelligent Robots and Systems*, Vol. 3.

[19] Peter Trautman and Andreas Krause. 2010. Unfreezing The Robot: Navigation In Dense, Interacting Crowds. In *Intelligent Robots and Systems (IROS)*. IEEE, 797–803.

[20] Peter Trautman, Jeremy Ma, Richard M Murray, and Andreas Krause. 2013. Robot Navigation In Dense Human Crowds: The Case For Cooperation. In *IEEE International Conference on Intelligent Robots and Systems*. 2153–2160.

[21] Katherine M. Tsui, Munjal Desai, Holly A. Yanco, and Chris Uhlik. 2011. Exploring Use Cases for Telepresence Robots. In *Proceedings of the 6th International Conference on Human-robot Interaction (HRI '11)*. 11–18. https://doi.org/10.1145/1957656.1957664

[22] Vaibhav V Unhelkar, Claudia Pérez-D'Arpino, Leia Stirling, and Julie A Shah. 2015. Human-robot Co-navigation Using Anticipatory Indicators Of Human Walking Motion. In *IEEE International Conference on Intelligent Robots and Systems*. 6183–6190.

[23] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. 2011. Reciprocal n-Body Collision Avoidance. In *Robotics Research: The 14th International Symposium ISRR*. Springer Berlin Heidelberg, Berlin, Heidelberg, 3–19.

[24] Manuela M Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. 2015. CoBots: Robust Symbiotic Autonomous Mobile Service Robots.. In *IJCAI*. 4423.

[25] Melonee Wise, Michael Ferguson, Derek King, Eric Diehr, and David Dymesich. 2016. Fetch And Freight: Standard Platforms For Service Robot Applications. In *Workshop on Autonomous Mobile Service Robots*.

[26] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa. 2009. Planning-based Prediction for Pedestrians. In *2009 IEEE International Conference on Intelligent Robots and Systems*. 3931–3936. https://doi.org/10.1109/IROS.2009.5354147