ϵ-UCB for Action Selection in Multi Agent Navigation

Julio Godoy, Ioannis Karamouzas, Stephen J. Guy and Maria Gini

Abstract— **In multi-robot systems, efficient navigation is challenging as agents need to adjust their paths to account for potential collisions with other agents and static obstacles. In this paper, we present an online machine learning approach, ϵ-UCB, which improves global efficiency in the motions of multiple agents by building on ORCA, an existing multi-agent navigation algorithm, and on UCB, a widely used action selection technique. With ϵ-UCB, agents adapt their motions to their local conditions while achieving globally efficient motions. We validate our approach experimentally, in a variety of scenarios and with different numbers of agents. Results show that agents using ϵ-UCB exhibit more globally time efficient motions, when compared to just ORCA and to UCB.**

I. INTRODUCTION

Navigating multiple agents through complex environments in real-time and in a decentralized manner has important applications in many domains such as swarm robotics, pedestrian navigation, and traffic engineering. This navigation problem is challenging because of conflicting constraints induced by the presence of other moving agents. As agents plan paths in a decentralized fashion, they often need to recompute their paths to avoid colliding with other agents.

Over the past twenty years, many decentralized techniques for real-time multi-agent navigation have been proposed. These range from force-based approaches to more robust velocity-based solutions, which provide guarantees on collision-free motion for multiple agents. Although these robust approaches generate locally efficient motions for each agent, the overall behavior of the agents can be far from efficient; actions that are locally optimal are not necessarily optimal for the entire group of agents.

One way to improve the global efficiency of the agents' motion is for each agent to learn, through interaction with the environment, what motion is best given its local conditions. Unfortunately, many machine learning algorithms require an offline training phase to achieve learning, or take a long time before converging to the optimal policy. In addition, the computational complexity of learning methods becomes prohibitively high as the number of agents increases.

Our work focuses on *online* learning methods that can be completely distributed and require no communication among the agents. We incorporate online adaptation by formulating the multi-agent navigation problem as a Multi-Armed Bandit problem. The main challenge in these problems is to balance the exploitation of the current best action with the exploration of other actions that may later produce higher rewards [1]. We demonstrate that a widely used method for balancing exploration and exploitation, Upper Confidence Bounds (UCB) [2], is not well suited to the real-time environments of multi-agent navigation tasks. We propose a novel approach, ϵ-UCB, to achieve the desired balance. We combine ϵ-UCB with ORCA [3], a velocity-based local navigation technique. The resulting motion shows a clear improvement over the current state-of-the-art.

*All authors are with the Department of Computer Science and Engineering, University of Minnesota, USA

Partial support is gratefully acknowledged from NSF grant IIS-1208413

This paper makes three contributions. First, we model the problem of achieving globally efficient motions in multi-agent navigation as a multi-armed bandit problem. Second, we propose an online learning approach, ϵ-UCB, which addresses the exploration/exploitation tradeoff via a probabilistic combination of UCB and a greedy method. Third, we experimentally show that our approach leads to more time efficient motions in a variety of scenarios, as compared to using UCB with ORCA and ORCA alone.

II. RELATED WORK

A. Multi-Agent Navigation

Numerous models have been proposed to simulate individuals and groups of interacting agents. After the seminal work of Reynolds on *boids*, many interesting crowd simulation models have been introduced that account for groups [4], cognitive and behavioral rules [5], and other factors [6], [7].

An extensive literature also exists on modeling the local dynamics of the agents and computing a collision-free motion among static and/or dynamic obstacles. Complete approaches have been proposed that pre-compute motions offline [8], however they are not suitable for real-time application. This limitation inspired the development of faster online approaches with applications in real-time domains. For example, Khatib [9] pioneered the use of artificial potential fields, where the robot is treated as a particle which is attracted to the goal and repelled by obstacles. Brooks [10] in his layered architecture, popularized the use of artificial potential fields for robot navigation. Closely related is the social force framework of Helbing for pedestrian simulation [11]. However, all these approaches are reactive and do not account for the velocities of the agents, leading to issues especially in environments where agents move at high speeds. To address this issue, *geometrically-based* algorithms have been proposed [12], which compute collision-free velocities for the agents using either sampling [13] or optimization techniques [3]. In our work, we also use the navigation method proposed in [3] that is robust to different types of environments by accounting for agents' velocities.

B. Learning in Multi-Agent Navigation

We deal with a multi-agent learning problem where agents learn simultaneously and affect the decisions of other agents. We consider this multi-agent system as a group of independent learners, where each agent considers other agents as part of the environment. Extensive work has been done on learning and adapting motion behavior for agents in crowded environments. Depending on the nature of the learning process, this work can be classified in two main categories: offline and online learning. In offline learning, agents repeatedly explore the environment and try to learn an optimal policy given an objective function. Examples of desired learned behaviors include collision avoidance, shortest path to destination, and specific group formations. The approach in [14], for example, applies Q-learning to plan paths for agents in crowds. Similarly, a SARSA-based [15] learning algorithm has been used in [16] for offline learning of behaviors in crowd simulations. Offline learning has limitations arising from the need to train the agents before the environment is known. In online approaches like ours, agents have only partial knowledge of their environment, and need to adapt their strategies to what the other agents do. Recently, an anytime planning-based approach for multi-agent navigation was proposed in which the agents' paths are computed in an iterative manner [17].

III. MULTI AGENT NAVIGATION

In this paper, we introduce an *online* learning technique for multi-agent navigation so that agents can exhibit more intelligent and efficient motion. More formally, in our problem setting, we are given n independent agents A_i $(1 \leq i \leq n)$, each with a unique start and goal positions specified in $\mathbb{R}^2$. The environment for A_i is defined as a 2D virtual or physical space with all the remaining $n-1$ agents, along with a set of static obstacles O. The task is then to steer each of these agents to its goal without colliding with the other agents and the obstacles present in the environment. We also require that the agents navigate *independently* without explicitly communicating with each other.

For simplicity, we model each agent A_i as a disc with radius r_i. The agent A_i has a position $\mathbf{p}_i$, defined by the (x, y) coordinates of the center of the disc, and moves with velocity $\mathbf{v}_i$ that is limited by a maximum speed $\upsilon_i^{\max}$. Furthermore, A_i has a preferred velocity $\mathbf{v}_i^{\text{pref}}$ directed toward the agent's goal $\mathbf{g}_i$ with a magnitude equal to the speed υ_i^{pref} at which the agent prefers to move. We assume that the radii, positions and velocities of the nearby agents can be obtained by local sensing. In this work, agents can sense other agents 15 meters away, and obstacles 1 meter away.

Collision Avoidance and ORCA

Selecting collision-free velocities as close as possible to $\mathbf{v}^{\text{pref}}$ is a challenging problem in and of itself. Collision avoidance methods can be either reactive or anticipatory. In reactive approaches, agents react to avoid collisions only when they are sufficiently close. This can lead to oscillations between agents and to local minima. Another limitation of these methods is that the forces must be tuned separately for each scenario, limiting their robustness. In anticipatory approaches, agents detect and avoid potential upcoming collisions by linearly extrapolating their current velocities. In this category, the principle of Optimal Reciprocal Collision Avoidance (ORCA) for multi-agent navigation was proposed [3]. ORCA provides collision and oscillation-free motion between multiple moving agents, as well as optimality guarantees with respect to the velocities computed. ORCA accounts for anticipatory avoidance, reciprocity among the agents as well as sensor noise and motion uncertainty [18].

Although ORCA guarantees collision-free motions and provides a locally optimal behavior for each agent, it does not account for the aggregate behavior of the agents so it can lead to globally inefficient motions. For example, when agents pass through a narrow bottleneck, ORCA will slow them down to ensure collision-free motion. If all movement leads to potential collisions, the agents may get stuck in local minima resulting in long delays in the navigation. Instead, if agents were able to behave differently in those situations, for example, by selecting a $\mathbf{v}^{\text{pref}}$ in a different direction for a short period of time, they might be able to find a larger region of feasible velocities. This might indirectly help avoid or solve overall congestion, benefiting all agents. Our proposed learning approach addresses these issues, allowing agents to intelligently adapt their preferred velocity in an online fashion and improving their efficiency in challenging environments.

IV. MULTI-ARMED BANDIT FORMULATION

To alleviate the inability of ORCA to produce globally efficient motions, we formulate the navigation problem as a multi-armed bandit problem. In a multi-armed bandit problem, an agent makes sequential decisions on a set of actions (the arms) to maximize its expected reward. To do this, the agent must balance the exploitation of the current best action with the exploration of potentially better ones. The challenge for the agent is to use its past experience to predict the best action at each time, by accounting for uncertainty on future outcomes. In our domain, each arm corresponds to a specific $\mathbf{v}^{\text{pref}}$.

An ORCA-simulated agent has, by default, only one action: a goal-oriented $\mathbf{v}^{\text{pref}}$ with a magnitude equal to υ^{pref}. We would like to enrich the choices that agents have, compared to what ORCA provides, by enabling agents to choose different preferred velocities. This would require each agent to make a choice at every simulation step in two continuous spaces, the space of speeds and the space of directions. However, in an online learning setting each agent is limited in the number of exploration steps to learn from; increasing the number of choices will either reduce the amount of exploration per choice, leading to a larger learning error, or will increase significantly the time needed for learning. This tends to reduce performance, as discussed in previous work [19].

After an experimental evaluation of different sets of actions, we concluded that a set of five fixed velocity choices gave the best performance, as it allowed the agent enough variety of behaviors while avoiding spending too much time in exploration. Specifically, the actions defined correspond to: (1) moving directly straight towards the goal, (2) left 10°, (3) right 10°, all at a speed of 1.5 m/s, (4) a complete stop, and (5) a slow backwards movement (0.5 m/s). This set of actions includes the goal-oriented motion (ORCA's default action) as it is often the best action to take. We also allow the agent to move aside, left or right, in case the goal-oriented motion is not allowed, to stop if no movement is allowed, and to step backwards when no forward movement is possible. Since stopping or moving away from the goal yields to other agents' motion, we refer to this subset of actions as "polite" behavior.

A. Learning Episode

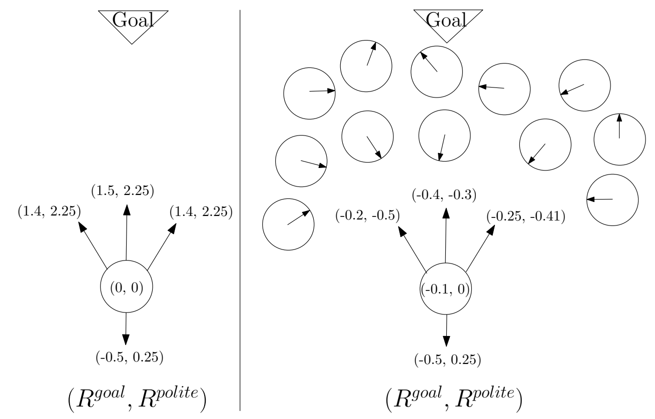
Our agents decide on their motions while the simulation is running. The simulation is divided into time steps, or cycles, each of a fixed time duration. Each simulation cycle corresponds to a learning episode. At each cycle, each agent chooses a specific action, $\mathbf{v}^{\mathrm{pref}}$, based on its rewards and the current simulation state.

An agent uses the state of the simulation (position and velocities of the nearby agents) as input to its learning process. The output of the learning process for the agent is $\mathbf{v}^{\mathrm{pref}}$ which is used as input to ORCA. ORCA analyzes potential collisions and returns the new simulation state (the new position and velocity) based on the chosen $\mathbf{v}^{\mathrm{pref}}$. The union of the updated positions and velocities of all agents constitutes the new global simulation state, which is then used in the next cycle.

B. Optimization Function

Our learning approach ultimately aims at minimizing the total time it takes for all agents to reach their goal. In order to evaluate the action choices of all the agents throughout a simulation (in hindsight), we would normally use the *regret* metric to compare them with the optimal choices each agent should have made to minimize the global travel time. However, measuring the regret in this way as the simulation progresses is impossible, since there is no oracle against which to compare. Therefore, to encourage actions that help reduce the global regret, we propose a reward function that drives the agent towards its goal, while taking into account the interaction with other agents. Specifically, our reward function (see Eq. 3) is a convex combination of a *goal-oriented* component and a *politeness* component. This function is executed after each timestep and evaluates the previously selected action $\mathbf{v}^{\mathrm{pref}}$ with the collision-free velocity $\mathbf{v}$ computed by ORCA.

The *goal-oriented* component R_a^{goal} computes the scalar product of the collision-free velocity $\mathbf{v}$ of the agent with the normalized vector which points from the position $\mathbf{p}$ of the agent to its goal $\mathbf{g}$. More formally, we compute this component as:



(a) Navigation in an open area (b) Navigation in a congested area

Fig. 1: Example of reward values under (a) clear and (b) congested local conditions.

$$R_a^{goal} = \mathbf{v} \cdot \frac{\mathbf{g} - \mathbf{p}}{\|\mathbf{g} - \mathbf{p}\|} \tag{1}$$

The *politeness* component R_a^{polite} compares the preferred velocity given as an action input to ORCA with the resulting collision-free velocity. If ORCA determines that a given preferred velocity may produce collisions, it will produce a different collision-free velocity. Hence, the relation between $\mathbf{v}$ and $\mathbf{v}^{\mathrm{pref}}$ expresses how collision-free $\mathbf{v}^{\mathrm{pref}}$ is. The larger the difference between $\mathbf{v}^{\mathrm{pref}}$ and $\mathbf{v}$, the less polite it is to take the associated action, as potential collisions would also hinder the progress of other agents. Therefore, similar values of $\mathbf{v}^{\mathrm{pref}}$ and $\mathbf{v}$ indicate the action is very polite. Polite actions do not necessarily drive the agent closer to its goal but allow other agents to progress, thereby improving the global simulation state. We compute this component as:

$$R_a^{polite} = \mathbf{v} \cdot \mathbf{v}^{\mathrm{pref}} \tag{2}$$

The reward R_a for the agent for doing action a is given by:

$$R_a = \gamma * R_a^{goal} + (1 - \gamma) \times R_a^{polite} \tag{3}$$

where the parameter γ controls the influence of each component in the total reward ($0 \leq \gamma \leq 1$).

Figure 1 shows an example of two local conditions an agent may encounter. The five actions available correspond to the ones described earlier in this section. The approximate reward values shown are separated according to the two components of the reward function (R^{goal}, R^{polite}). In Fig. 1(a), the agent's movement is not constrained and consequently the reward is higher for the actions that drive the agent towards its goal. In Fig. 1(b), congestion has formed and the goal-oriented actions are now constrained, which results in lower rewards. In this case, the agent will prefer less constrained actions, such as stopping or moving away from the goal, which avoid increasing the congestion and allow other agents to move toward their goals.

To keep an updated estimate of the value of the actions, agents maintain a moving window of the reward obtained in the last 50 timesteps. The estimated value of an action is the average of the reward obtained by performing that action in the last 50 timesteps.

V. Epsilon-UCB

In this section, we briefly describe a widely-used action selection technique called UCB, before fully describing our proposed approach, ϵ-UCB.

A. UCB

UCB works by sampling the actions proportionally to the upper-bound of the estimated value of their rewards $\overline{R_a}$. The UCB estimate $UCB(a)$ of action a for an agent is a sum of two terms (Eq. 4). The first term $\overline{R_a}$ is the moving average of the reward for the agent of performing action a. The second term corresponds to the size of the one-sided confidence interval for the average reward. This confidence interval bounds the true expected reward with very high probability [2]. This term is computed using the number of times action a was selected (n_a) and the total number of action decisions made so far by the agent (n).

$$UCB(a) = \overline{R_a} + \sqrt{\frac{2 \times \ln(n)}{n_a}} \tag{4}$$

At each simulation cycle, UCB updates its estimates of all actions and selects the one with the highest value. Therefore, UCB takes into account the value estimates of the actions and the frequency with which actions have been selected, to encourage the exploration of actions with low value but with potentially outdated estimates.

However, in situations where actions have a fixed known reward, UCB performs unnecessary exploration. In these situations, just exploiting the best action is the best strategy. Our approach takes advantage of both UCB and a greedy action selection method in a probabilistic manner.

B. ϵ-UCB

Our ϵ-UCB approach uses a probabilistic action selection method combining UCB and the best current action to enhance the ability of the agents to adapt. The ϵ-UCB pseudocode can be seen in Algorithm 1. At the beginning of the simulation, we assume that the agent has no previous knowledge of the rewards of its actions. Hence, it first samples the velocity space with its actions to obtain an initial estimate of their values under the current local conditions. This initial sampling is performed by executing ORCA with each action's preferred velocity for a single timestep. Once the initial rewards have been collected, the agent must decide whether to exploit the highest valued action or explore for possible changes in the reward. To do this, the agent probabilistically selects an action, using an exploration rate (ϵ). A key difference between ϵ-UCB and traditional UCB is that explotation is done a fixed fraction of time (lines 8 and 9) and is not dependant on how much better the best action is as compared with the other actions. This increases the amount of time the agent exploits the best current action. After action a is selected, its corresponding $\mathbf{v}^{\mathrm{pref}}$ is passed to ORCA (line 13) to compute a collision-free velocity for execution in the next timestep. This velocity is used to update the estimated reward of the action (line 14).

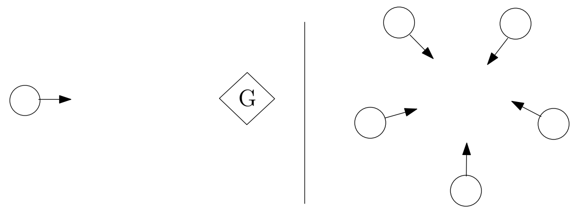
The motivation behind combining UCB and the best known action is twofold: it helps the agent to greedily select the best action when local conditions are stable, reducing the negative effects on reward caused by exploration, while maintaining sensitivity to changes in the local conditions.

Algorithm 1: ϵ-UCB algorithm for an agent

```
1:  **Input:** $ExpRate \in [0, 1]$
2:  initialize simulation, $\mathbf{p}$ = $(x_{start}, y_{start})$
3:  $\epsilon \leftarrow ExpRate$
4:  initialize $R_a$ for all the actions to 0
5:  sample the actions
6:  **while** not at the goal **do**
7:      randomly generate variable p $\in [0, 1]$
8:      **if** $p \leq$ (1-$\epsilon$) **then**
9:          $a \leftarrow$ action with highest estimated reward value
10:     **else**
11:         $a \leftarrow$ action suggested by UCB
12:     **end if**
13:     *ORCA(a)*
14:     update reward estimates
15: **end while**
```

VI. EXPERIMENTAL RESULTS

To analyze the performance of our approach, we compare the regret of the agents using ORCA, ORCA with UCB, and ORCA with ϵ-UCB. However, given the local nature of the actions, the agents cannot directly measure the regret to figure out which action actually minimizes the global travel time. Hence, we use an upper bound of the regret, which we call *Regret**, defined as the difference between the time the last agent reaches its goal and the time that it takes for an agent to traverse at maximum speed (1.5 m/s) the shortest path to the goal. We evaluated the *Regret** of the different approaches in a variety of experiments and scenarios.



(a) Single Agent (b) Small Circle

Fig. 2: Examples of scenarios with and without constraints. (a) A single agent with no obstacles can follow a direct path towards its goal. (b) 5 agents located in a circle cross paths in the middle, inducing constraints on one another.

We begin by evaluating how the exploration strategy affects the time efficiency of the agents. To do this, we measure a single agent's time to reach the goal in a clear

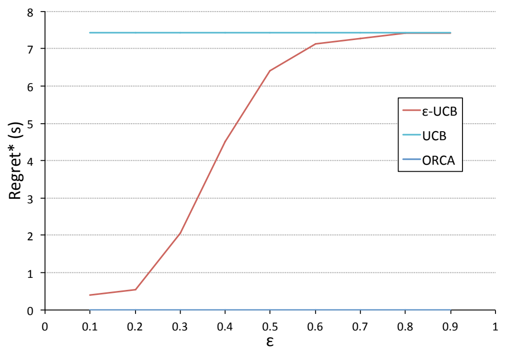


Fig. 3: Total time for a single agent to complete a 100 meter long navigation path, for different values of ϵ.

unconstrained environment. The distance between the agent's starting position and its goal is 100 meters (see Fig. 2(a)). We vary ϵ in the range between 0.1 and 0.9.

We can observe in Fig. 3 that in this simple scenario, the ORCA agent performs the best, as it always takes the action of going full speed towards its goal. This is the optimal policy and the agent has zero *Regret**. UCB and ϵ-UCB are less time efficient as they perform unnecessary exploratory actions. However, ϵ-UCB agents have very low *Regret** when performing little exploration (ϵ=0.1), but their *Regret** increases with more exploration, reaching the upper bound given by UCB. Therefore, lower values of exploration are used in the remaining experiments.

These results are expected as this simple scenario does not have any constraints, so choosing the default ORCA $\mathbf{v}^{\text{pref}}$ is the optimal policy. When the presence of other agents creates constraints, the default behavior is no longer optimal. Fig. 2(b) is an example of this, as 5 agents are located along a circle of diameter of 10 meters, and need to navigate to opposite positions in the circle. In this case, the agents' paths intersect at the center of the circle, which causes slowdowns and increases their *Regret**. The agents' action selection method allows them to find the best action when it dynamically changes. The *Regret** that the agents experience for the three techniques (with $\epsilon = 0.1$), in seconds, is as follows:

Method	ϵ-UCB	UCB	ORCA
*Regret**	1.78	2.56	18.8

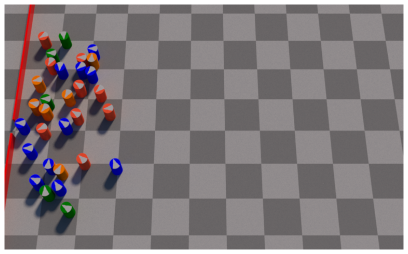
We can observe that ORCA agents exhibit a large *Regret** as their default preferred velocity is not allowed in the middle of the circle. The multiple constraints in this scenario slow down the agents for a large part of the simulation, before their local conditions improve. UCB agents perform better as they find, through exploration, actions that prevent them from slowing down in the middle, and instead they pass by each other decreasing their *Regret** as compared to pure ORCA. However, ϵ-UCB agents perform even better, demonstrating that greedy exploitation of the current best action pays off increasing global time efficiency, as compared to pure UCB. To evaluate the performance of ϵ-UCB in more complex settings, we performed experiments in larger scenarios with many agents and static obstacles. Below we briefly describe each scenario:

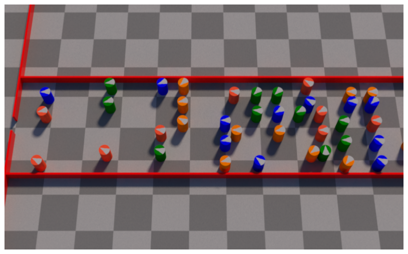
- **Congested**: 32 agents are placed very close to the narrow exit of an open hallway they must escape through this exit (Fig. 4 (a)).
- **Exit**: 48 agents are placed across a narrow corridor with a single exit that allows only one agent at a time to pass (Fig. 4 (b)).
- **Crossing**: Two groups of 100 agents cross paths while moving from opposite directions (Fig. 4 (c)).
- **Circle**: 128 agents are evenly distributed along the circumference of a circle. Each agent has to move to its diametrically opposite position on the circle (Fig. 4(d)).

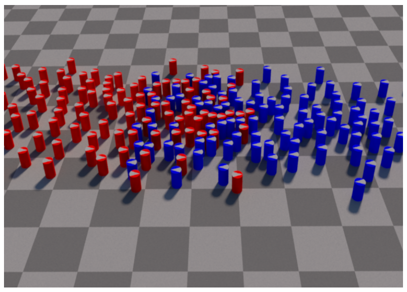
Figure 5 shows the results of ORCA, UCB, and ϵ-UCB in these four scenarios. In all of them, we can observe that ϵ-UCB agents have lower *Regret** than both ORCA and UCB. In the $Congested$ scenario, ϵ-UCB outperforms ORCA by 35% and has 21% lower *Regret** than UCB. In the $Exit$ scenario, the overhead time when using ϵ-UCB is significantly different than for UCB, outperforming ORCA by 41% and UCB by 38%. In both scenarios, the movement of each agent is constrained by the presence of other agents and static obstacles, especially close to the single exit. With ϵ-UCB, agents that fail to make progress are able to learn to step backwards instead of forcing their way towards the congested area. This provides more room for other agents to maneuver, helping to resolve congestion faster as compared to ORCA simulated agents. Consequently, our approach leads to faster exit times in both scenarios. In the $Crossing$ scenario, ϵ-UCB again performs significantly better than UCB, although by a small margin, but it still outperforms ORCA agents by 50%. Finally, in the $Circle$, agents using ORCA immediately converge to the center of the environment. Consequently, they have to frequently speed up and slow down to avoid collisions. UCB performs worse than ORCA, as UCB agents perform too much unnecessary exploration in open areas. Selecting suboptimal actions too frequently increases the agents total time to destination. Overall, our approach has 4% lower *Regret** than ORCA and 24% lower *Regret** than UCB.

VII. CONCLUSIONS

In this paper, we presented a learning algorithm called ϵ-UCB to improve the ability of the agents to navigate efficiently, in real-time, in a crowded environment. We formulate the navigation problem as a multi-armed bandit, where the action space corresponds to a set of preferred velocities. Agents value their actions through a reward function that encourages goal-oriented motion and polite behavior between agents. In ϵ-UCB, agents adapt online to changing environmental conditions. To balance the exploration versus exploitation tradeoff, we blended in a probabilistic manner UCB with the selection of the best known action. We tested our approach across a variety of scenarios and different numbers of agents. Experimental results show that ϵ-UCB significantly improves the time efficiency of the agents, as


(a) Congested


(b) Exit


(c) Crossing

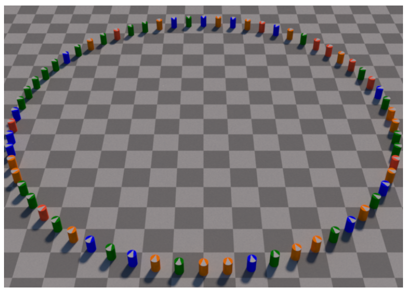

(d) Circle

Fig. 4: Experimental Scenarios. (a) Congested: 32 agents placed close to each other exit an open hallway through a narrow doorway. (b) Exit: 48 agents need to exit a corridor through a narrow doorway that allows one agent at a time to pass. (c) Crossing: two groups of 100 agents cross paths while moving from opposite direction. (d) Circle: 128 agents are placed along the circumference of a circle and must reach their antipodal position in the circle.

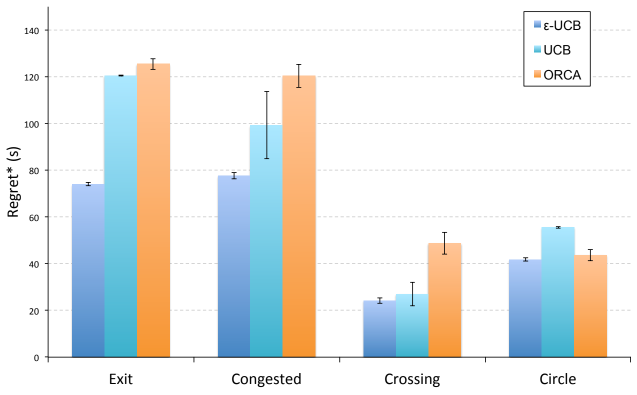


Fig. 5: Comparison of ϵ-UCB with UCB and ORCA in the four scenarios.

compared to UCB or ORCA without learning. ϵ-UCB, which has a low computational complexity, can improve the battery life of real autonomous robots and is ideal for effort-efficient navigation tasks.

References

[1] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *The Journal of Machine Learning Research*, vol. 3, pp. 397–422, 2003.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[3] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research: The 14th International Symposium ISRR*, ser. Springer Tracts in Advanced Robotics, vol. 70. Springer-Verlag, 2011, pp. 3–19.

[4] O. Bayazit, J.-M. Lien, and N. Amato, "Better group behaviors in complex environments using global roadmaps," in *8th International Conference on Artificial life*, 2003, pp. 362–370.

[5] W. Shao and D. Terzopoulos, "Autonomous pedestrians," *Graphical Models*, vol. 69, no. 5-6, pp. 246–274, 2007.

[6] N. Pelechano, J. Allbeck, and N. Badler, "Controlling individual agents in high-density crowd simulation," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2007, pp. 99–108.

[7] S. Guy, S. Kim, M. Lin, and D. Manocha, "Simulating heterogeneous crowd behaviors using personality trait theory," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2011, pp. 43–52.

[8] D. Hsu, R. Kindel, J. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[9] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[10] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. of Robotics and Automation*, vol. RA-2, no. 1, pp. 14–23, Mar. 1986.

[11] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[12] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using Velocity Obstacles," *Int. J. Robotics Research*, vol. 17, pp. 760–772, 1998.

[13] I. Karamouzas and M. Overmars, "Simulating and evaluating the local behavior of small pedestrian groups," *IEEE Trans. on Vis. Comput. Graphics*, vol. 18, no. 3, pp. 394–406, 2012.

[14] L. Torrey, "Crowd simulation via multi-agent reinforcement learning," in *Artificial Intelligence and Interactive Digital Entertainment*, 2010, pp. 89–94.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[16] F. Martinez-Gil, M. Lozano, and F. Fernández, "Calibrating a motion model based on reinforcement learning for pedestrian simulation," in *Motion in Games*, ser. LNCS, vol. 7660. Springer, 2012, pp. 302–313.

[17] J. Godoy, I. Karamouzas, S. J. Guy, and M. Gini, "Anytime navigation with progressive hindsight optimization," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2014.

[18] S. Kim, S. J. Guy, W. Liu, R. W. Lau, M. C. Lin, and D. Manocha, "Predicting pedestrian trajectories using velocity-space reasoning," in *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 609–623.

[19] M. Tokic, "Adaptive ε-greedy exploration in reinforcement learning based on value differences," in *KI 2010: Advances in Artificial Intelligence*. Springer, 2010, pp. 203–210.