Implementation of SARL* Algorithm for A Differential Drive Robot in a Gazebo Crowded Simulation Environment

Seif Eddine Seghiri, Noura Mansouri, Ahmed Chemori

To cite this version:

HAL Id: lirmm-03826760
https://hal-lirmm.ccsd.cnrs.fr/lirmm-03826760

Submitted on 24 Oct 2022
Implementation of SARL* Algorithm for A Differential Drive Robot in a Gazebo Crowded Simulation Environment

1st Seif Eddine Seghiri  
LARC Laboratory  
University of Constantine 1  
Constantine, Algeria  
seifeddine.seghiri@umc.edu.dz

2nd Noura Mansouri  
LARC Laboratory  
University of Constantine 1  
Constantine, Algeria  
nor_mansouri@yahoo.fr

3rd Ahmed Chemori  
LIRMM Laboratory  
University of Montpellier  
CNRS, Montpellier, France  
Ahmed.Chemori@lirmm.fr

Abstract—Because of the stochasticity in people’s behaviors, autonomous navigation in crowded environments is critical and challenging for both the robot and people evolving around. This paper deals with the implementation and effectiveness evaluation of the Socially Attentive Reinforcement Learning star algorithm, namely SARL*, which is an extended version of the state-of-the-art socially compliant navigation algorithm SARL. It introduces a dynamic local goal resetting mechanism. The Simulations were conducted in the Robot Operating System (ROS) and the Gazebo simulator is used to test the human-aware navigation in different scenarios. Simulation results illustrate the efficiency of SARL* in terms of navigation around people in a socially acceptable manner. Nevertheless, it could not navigate efficiently when the goal position is located behind static or quasi-static obstacles.

Index Terms—SARL*, Social Robotics, ROS, Deep Reinforcement Learning, Navigation

I. INTRODUCTION

A robotic framework is often constructed around three main entities, including (i) guidance, (ii) navigation and (iii) control (GNC) systems. These systems mainly interact with each other through the transmission of data and signals. In its basic form, GNC is a reference model (the guidance system), a measurement/sensor system (the navigation system) and a controller (the control system) [1]–[4].

Mobile robot navigation has been intensively researched as a basic problem in robotics. Today, more service robots are being created to function in human-robot coexisting situations, including, autonomous vehicles, smart wheelchairs, luggage collecting and hospitality robots [5]–[10].

In traditional navigation frameworks typically, Collision avoidance modules treat dynamic obstacles as static, such as the Dynamic Window Approach (DWA) [11] or simply focus on the next step of action based on certain interaction rules, such as Reciprocal Velocity Obstacle (RVO) [12] and Optimal Reciprocal Collision Avoidance (ORCA) [13]. Because these strategies prevent collisions by passive reaction and typically rely on manually programmed functions to assure safety, it has been revealed that they lead the robot’s movement to be awkward, short-sighted, and dangerous.

In congested environments, a robot must be able to observe, interpret and predict the behavior of the surrounding pedestrians in order to move in a socially acceptable manner. In the human aware navigation problem, the integration of human motion prediction with robot motion planning remains a difficult problem. One of the existing approaches consists to plan after prediction, i.e., to choose a safe path after predicting the future trajectories [14]. To anticipate pedestrian trajectories, some hand-crafted models (e.g., constant velocity model [15], discrete choice model [16], social force model and its variations [17], [18]) and data-driven approaches (e.g., social LSTM [19], [20], social GAN [21], [22]) have been presented. However, the significant stochasticity of the crowd behavior frequently affects the computing cost and reliability of pedestrian trajectory prediction. Such planning-after-prediction approaches are still hard for practical implementations in the context of human-aware navigation for mobile robots, which demands both security and time efficiency [23].

Deep Reinforcement Learning (DRL) is another category of algorithms for human-aware navigation that includes human motion prediction into the decision-making process, i.e., the robot learns from experiences to comprehend crowded environments and encodes crowd-robot interaction in the navigation strategy. Recent studies [24]–[29] shown greater ability to create crowd-aware navigation rules, with the Socially Attentive Reinforcement Learning (SARL) algorithm obtaining cutting-edge performance for an effective DRL based human-aware navigation.

II. PROBLEM FORMULATION

Let us consider a robot and $n$ humans as agents, in a 2D workspace. Each of these agents has an observed states such as position $\mathbf{p} = [p_x, p_y]$, velocity $\mathbf{v} = [v_x, v_y]$ and radius $r$ (i.e. agents are represented as circles in their workspace). For each agent the rest of the states, such as the orientation $\theta$, goal position $\mathbf{g} = [g_x, g_y]$ and the preferred speed $v_{pref}$, are not observed by other agents.
The robot’s full state at time $t$ can be defined as $S_t = [x_t, y_t, v_x, v_y, \theta, g, \gamma, r, \delta_{pref}]$, and the $k$-th human’s observable state at time $t$ can be defined as $O_t^k = [x^k_t, y^k_t, v^k_x, v^k_y, \gamma^k, r^k]$. To make the state representation more general, the current and goal positions $(p, g)$ are replaced by the distance between them and denoted $d_g$. The distance between the robot and the $k$-th human is denoted by $d^k$. As adopted in [26] the new state is defined as follows:

$$ S_t = [d_g, \delta_{pref}, x_t, y_t, \gamma] $$

$$ O_t^k = [d^k, \gamma^k, x^k_t, y^k_t, \gamma^k, \delta^k + r^k] $$

A joint state of all $(n + 1)$ agents at time $t$ is constructed by concatenating the state $S_t$ with all of the $O_t^k$:

$$ J_t = [S_t, O_1^t, O_2^t, ..., O_n^t] $$

Differential drive robot is controlled by linear and angular velocity commands (i.e., actions $a_i$) according to a specified navigation policy $\pi(J_t)$: $\pi(J_t) = a_i = v_i$. At each time the robot is awarded by $R(J_t, \pi(J_t))$. The form of the reward function is as following:

$$ R(J_t, a_t) = \begin{cases} 
-0.25, & \text{if } d_{min} < 0; \\
0.5 \cdot d_{min} - d_c, & \text{if } 0 < d_{min} < d_c; \\
1, & \text{if } d_g = 0; \\
0, & \text{otherwise.} 
\end{cases} $$

Where $d_{min}$ represents the smallest separation distance between the robot and people during the decision interval $\Delta t$, and $d_c$ is the shortest comfortable distance that humans can handle.

The optimal value of $J_t^*$ at time $t$ can be formulated as:

$$ V^*(J_t) = \sum_{i=0}^{K} \gamma^i \cdot \Delta t \cdot v_{pref} \cdot R(J_t, a_t^i) $$

where $a_t^i$ is selected according to a given optimal policy $\pi^*(J_t)$. $K$ is the total number of decision steps from the state at time $t$ to the final state, $\Delta t$ is the decision interval between two actions $a_t$ and $\gamma \in [0, 1]$ is a discount factor in which the preferred speed $v_{pref}$ is introduced as a normalization parameter.

Maximizing the cumulative reward yields the optimal policy, which is as follows:

$$ \pi^*(J_t) = \arg \max_{a_t \in A} R(J_t, a_t) + \gamma^\Delta t \cdot v_{pref} $$

The computation of the optimal strategy defined in (5) is simplified as follows:

$$ \pi^*(J_t) = \arg \max_{a_t \in A} R(J_t, a_t) + \gamma^\Delta t \cdot v_{pref} $$

### III. DEEP REINFORCEMENT LEARNING (DRL)

In Reinforcement Learning (RL), value functions $V(J_t)$ can be evaluated in a variety of ways. One approach is to utilize tables to record the values for each state $J_t$ or action-state pair $(J_t, a_t)$. However, this method, does not scale with the sizes of state and action spaces. Another approach is to employ neural networks to estimate value functions or, to generate an action distribution given input states. Artificial neural networks can be viewed as universal function approximators [30], which means that they can represent arbitrarily complicated mappings between spaces if properly trained.

The trained deep neural network takes as inputs the state $J_t$ and some features that could be collected from agents (e.g. range measurements), and outputs the estimated value function namely Value Network.

The training of the value network used in the SARL* algorithm is summarized by the Algorithm 1.

**Algorithm 1 Value Network Training**

- Initialize experience replay memory $E$ with demonstrations.
- Initialize the value network $N$ with memory $E$.
- Initialize the target value network $N' \leftarrow N$.
- for episode $= 1, ..., M$ do
  - Initialize $J_0$ randomly.
  - repeat
    - $a_t \leftarrow \text{EpsilonGreedyActionSelection}()$
    - value $\leftarrow R(J_t, a_t) + \gamma^\Delta t \cdot v_{pref} \cdot N'(J_{t+\Delta t})$
    - state $\leftarrow J_{t+\Delta t}$
    - Enrich experience $E \leftarrow (\text{state}, \text{value})$
    - Optimize value network $N$ with experience $E$
    - $t \leftarrow t + \Delta t$
  - until $J_t = J_{end}$ or timeout
- if episode $| TargetUpdateInterval $ then
  - Update the target value network $N' \leftarrow N$
- end if
- end for
- return $N$

### IV. SARL* ALGORITHM

The difference between SARL and SARL* is that SARL* introduces a dynamic goal resetting due to the SARL algorithm’s training conditions [23], [29]. The DWA local planner [11] is used in each $\Delta t$ to reset a local goal which is considered as a target goal for the SARL planner. The algorithm keeps controlling the robot to move towards the local goal till it reaches the global goal of the robot.

Furthermore, the learning of SARL algorithm does not consider the static nor the quasi-static obstacles in the 2D environment. Therefore, they introduced a procedure for checking whether the best selected action $a_t^i$ according to the optimal
policy $\pi^*(J_t)$ will drive the robot to an obstacle or not. The action will be aborted if it will drive the robot to an obstacle and the robot will choose rotating in place in order to find another valid optimal action.

**Algorithm 2** SARL*

Initialize $A$;
Load the pre-trained value network $N$;
Build the 2D map;
Set global goal position $g_{global}$;

while $d_g > \text{goal\_tolerance}$ do
    Update $J_t$;
    $\text{GlobalPlan} \leftarrow \text{Dijkstra}(p_t, g_{global}, \text{map})$;
    $g_{local} \leftarrow \text{FindLocalGoal}(p_t, \text{GlobalPlan}, d)$;
    $a_t \leftarrow \arg \max_{a_t \in A} R(J_t, a_t) + \gamma^{\Delta t} v_{pref} N(J_{t+\Delta t})$
    with $g \leftarrow g_{global}$
    $A_s = \text{FindSafeActionSpace}(p_t, A, \text{map})$
end while
return $N$;

As it can be seen from the ROS RViz application depicted on Fig. 1, humans are represented by the blue dots. The robot is moving towards the red dots (local goals), so that it will eventually reaches the global goal.

V. Description of the Simulation Environment

The mobile robot used for simulation is the Turtlebot2, as illustrated in Fig. 2. It is a low-cost differential-drive mobile robot built out of a Yujin Kobuki mobile platform plus sensors and actuators such optical encoders and DC motors. It is equipped with a Microsoft XBOX 360 Kinect along with a Hokuyo lidar sensor.

The robot’s simulation ROS package integrates the Kobuki platform with the lidar and Kinect ROS modules. Using the ROS navigation stack, these modules enables simulation and implementation of many mobile robotics tasks such as perception, localization, mapping, navigation, and control.

**Fig. 1.** Illustrative view of the robot, humans, global and local paths in the 2d map environment. The optimal action is selected with respect to the red points (local goals). Eventually, the robot will arrive at its final destination, while avoiding humans navigating around based on the SARL policy.

**Algorithm 2** SARL*

Initialize $A$;
Load the pre-trained value network $N$;
Build the 2D map;
Set global goal position $g_{global}$;

while $d_g > \text{goal\_tolerance}$ do
    Update $J_t$;
    $\text{GlobalPlan} \leftarrow \text{Dijkstra}(p_t, g_{global}, \text{map})$;
    $g_{local} \leftarrow \text{FindLocalGoal}(p_t, \text{GlobalPlan}, d)$;
    $a_t \leftarrow \arg \max_{a_t \in A} R(J_t, a_t) + \gamma^{\Delta t} v_{pref} N(J_{t+\Delta t})$
    with $g \leftarrow g_{global}$
    $A_s = \text{FindSafeActionSpace}(p_t, A, \text{map})$
end while
return $N$;

As it can be seen from the ROS RViz application depicted on Fig. 1, humans are represented by the blue dots. The robot is moving towards the red dots (local goals), so that it will eventually reaches the global goal.

The Robot Operating System (ROS) melodic is installed on an 8GB of RAM and i5 CPU PC running under Ubuntu 18.04 operational system along with Gazebo simulator, enables the creation of a realistic human-robot coexisting environment. The simulation environment depicted in Fig. 3 could be a good testing place where there are humans navigating alongside the robot. They move randomly in the environment and do not follow a certain navigation policy.

Each human is tracked using the people ROS package [31]. The robot and the humans states $S_t, O_t$ are used to update the joint state $J_t$. The SARL* algorithm is then started controlling the robot to move towards its goal while avoiding humans and obstacles.

**Fig. 2.** View of the Turtlebot2 Differential drive mobile robot used in simulation including its basic components and sensors.

**Fig. 3.** Top view of the human-robot simulation environment used to test the SARL* navigation efficiency and robustness.

VI. Numerical Simulation Results

In order to test the efficiency of the SARL* algorithm, we conducted the simulation in two steps. In the first one, we give the robot a goal position in front of it, and near its
current position. In the next step, the goal position is placed behind and a little bit far from the robot. Each of these two scenarios is tested by running the algorithm 5 times and then another 5 times while placing new static obstacles along each trajectory in such a way that the robot will encounter them. These obstacles are not present in the 2D occupancy grid map. The time limit for the navigation is defined as $t_{\text{max}} = 200 \text{ sec}$. It is valuable to mention that the robustness of SARL* is also considered, by controlling the simulated humans to randomly walk in different speeds rather than navigating according to the same training conditions such as circle crossing or the Optimal Reciprocal Collision Avoidance (ORCA) [13].

### A. Scenario 1

In this scenario, the robot is located initially at $p_0 = [0.0, 0.0]$ and receives a goal position $p_g = [6.22, 0.51]$. The SARL* policy will drive the robot from $p_0$ to $p_g$ while avoiding humans. Fig. 4 illustrates how SARL* is a good human-aware navigation policy in terms of finding the solution, time and path length. We also indicate on Fig. 4, the time required for the robot to arrive at the goal position at each run. The two curvatures generated by the SARL* policy are resulting from the humans passing beside the robot, where it successfully navigates through them. The same scenario is running with a static cylindrical obstacle placed along the trajectory (i.e. along the global path). Fig. 5 Illustrates the length of the trajectory taken by the SARL* policy to converge towards the goal position and each curve is associated with the time required to arrive at the final state.

### B. Scenario 2

This scenario will be more challenging than the previous one, due to the distance between the initial robot’s pose $p_0 = [0.0, 0.0]$ and the target position $p_g = [-6.00, -8.92]$. Like in the first scenario, each curve is associated with the time taken by the robot’s navigation. In this first part of the second scenario the robot is placed at $p_0$ and receives the target pose $p_g$. The SARL* algorithm will then drive the robot towards $p_g$. Another challenging problem in this part is that the robot will encounter an L-shaped obstacle as depicted in Fig. 6. The results will be illustrated through Fig. 7. The figure describes the trajectories taken by the robot to go from $p_0$ to $p_g$. In addition to the time required for it, in each run.
It can be seen from Fig. 7 that the robot makes a lot of detours inside the L-shaped wall. This is because the SARL* navigation dynamic goal resetting mechanism is not taking the occupancy map in account. Therefore, it will obviously try to move toward the dynamic local goal even if it is placed behind and obstacle. The second part of this scenario is that a cylindrical obstacle and cubic obstacle are placed at $p_{obs}^1 = [2.96, -6.11]$ and $p_{obs}^2 = [-4.12, -5.6]$ respectively. The navigation policy will drive the robot like in the first part of the scenario. However, it has to avoid the new obstacles. Fig. 8 is an illustration of the trajectories and the their time taken by the robot while following the SARL* navigation policy.

The time required for the robot to reach its goal is highly dependent on how many humans it encountered during its travel. One can imagine a scenario where the robot avoids a human by taking a right turn. Yet, if the robot is trying to avoid a human to the right, the right side has to be clear (i.e. it could get stuck). This explains the reason behind the high variance in the travel time.

The robustness of SARL* is quite acceptable. Nevertheless, in each one of the simulation scenarios when new obstacles are added to the environment, the robot did collide with them one or two times. This can bee deduced from the trajectories overlapping obstacles in Fig. 5 and Fig. 8. However, obstacles are not fixed and the robot could push them away from its path and continue moving along its trajectory.

The approach proposed by Li et al. in [23] is only effective when the goal location is unobstructed. It can be seen in Figs. 7 and 8, the detours made by the robot that are caused by the L-shaped wall which is a common obstacle in all real life human-robot coexisting indoor environments.

VII. CONCLUSIONS AND FUTURE WORK

We evaluated the effectiveness and robustness of SARL* in dealing with real-life socially indoor scenarios. Following a considerable amount of simulations, we found that SARL* policy is an effective technique for socially acceptable navigation in crowded environments. However, a 2D cost map of the surroundings is indispensable. To effectively deploy a socially attentive reinforcement learning-based human-aware navigation in real-world applications, static, quasi-static and dynamic obstacles must be integrated into the network training approach. As a future work, we suggest that the reward function design and the training environment must consider both social and indoor environmental information, such as walls and doors.

REFERENCES


