

Socially Compliant Navigation in Dense Crowds

Roman Bresson, Jacques Saraydaryan, Julie Dugdale, Anne Spalanzani

▶ **To cite this version:**

Roman Bresson, Jacques Saraydaryan, Julie Dugdale, Anne Spalanzani. Socially Compliant Navigation in Dense Crowds. IV 2019 - 30th IEEE Intelligent Vehicles Symposium, Jun 2019, Paris, France. pp.64-69, 10.1109/IVS.2019.8814288 . hal-02096006

HAL Id: hal-02096006
https://inria.hal.science/hal-02096006

Submitted on 11 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Socially Compliant Navigation in dense crowds

Roman Bresson[1], Jacques Saraydaryan[3], Julie Dugdale[2], and Anne Spalanzani[1]

[1]Univ. Grenoble Alpes, Inria, 38000 Grenoble, France
[2]Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble France
[3]University of Lyon, CITI, CPE Lyon France

***Abstract*— Navigating in complex and highly dynamic environments such as crowds is still a major challenge for autonomous vehicle such as autonomous wheelchairs or even autonomous cars. This article presents a new way of navigating in crowds by using behavioral clustering for the surrounding agents and representing the crowd as a set of moving polygons. Once the environment has been modelled in his way and the robot has all the information it needs, we then propose a navigation algorithm that is able to guide the vehicle through the scene. The key-points of this algorithm are that (1) it can avoid densely-populated areas in order to minimize the risk of being on a collision course with any of the surrounding dynamic obstacles, (2) it generates socially compliant trajectories.**

I. Introduction

The motivation of this work is to be able to navigate an autonomous vehicle/robot in a highly dynamic and densely human populated environments such as crowds. This type of situation has posed many problems for classical path planning algorithms, e.g. freezing problems, heavy recomputation of the route, zigzagging, and a lack of consideration for normative social behaviors. A critical aspect of the work is to derive socially compliant trajectories that are accepted by the passengers inside of the vehicle and by the pedestrians around the vehicle. In everyday situations, humans do not necessarily take the shortest path when navigating through a crowd. Instead, they take into account social norms, such as not walking through and breaking up a family group. We also have to overcome the so-called *Freezing Robot Problem* [1]. This happens when a robot cannot find a safe (i.e. collision-free) path and, thus, either stops moving or starts engaging in excessive avoidance manoeuvres in order to reach its destination. While autonomous vehicles perform well on standard roads and junctions, they succumb to the freezing robot problem in spaces that are densely populated by pedestrians. Shared spaces is a relatively new urban design approach that minimises the segregation between different types of road users (e.g. cars, bicycles and pedestrians) by removing features such as kerbs, road surface markings, traffic signs, and traffic lights. Our goal is for autonomous vehicles to be able to navigate safely, whilst adhering to social norms, in shared urban spaces.

II. Related Work in Motion Planning Strategies Among Crowds

Autonomous navigation in dense human-populated environments is at the crossroad of a number of fields, the most pertinent ones being robotics and multi-agent systems. Robotics research concerns managing perception, for example with lasers and depth maps, and control engineering. Multi-agent systems involves managing the evolution of the robot in relation to the other agents, such as vehicles, robots, and pedestrians, that populate the environment. Experimentation with real-life crowds and robots is very time-consuming, resource-dependant (since many volunteers are needed and the robot is often running for several hours) and potentially dangerous. Therefore most of the early tests are done in agent based simulation environments with crowd simulation models being used to initially validate the approach.

The approach used in [2] and in [3] uses rapidly expanding random trees applied in a 2D space cost grid representing the environment. Firstly, at each time-step, the robot observes its surroundings and detects any obstacles that it can find. Then, given the observations it estimates their trajectories as Gaussian processes. This generates a 2D space map that represents, for any point of the environment, the risk of a future collision. Using this map, a tree is built in the following manner: at each iteration, a point is randomly selected in the map (usually, there is a bias on the selection of this point so that it is in the general direction of the final objective). Several trajectories are built to reach that point from the closest node of the current tree; the one chosen is the one that minimizes the cost (here, the probability of collision given by the space risk map). Once the objective is reached by a node of the tree, the path is rebuilt from start to goal and the robot starts following it. If a new obstacle is detected during the run, or if an already known obstacle has changed its behaviour, and if that triggers a sufficient risk of collision, the tree will be partially rebuilt from that point on until a new path is found.

Another approach that relies on a 2D cost map is in Jumel et al. [4]. In that paper, the authors assume that the whole environment is known. Then, based on the observations of the robot, a 2D flow grid is built. For each square of the grid, the probability of an agent's direction is stored. Based on these observations, an orientation estimation is build, call a virtual flow grid, based on Von Mises distribution probabilities [5]. The authors use a standard A* short path

computation but modify the cost estimation for selecting neighbor cells. The cost becomes higher if the cell transition is facing the opposing flow direction (or estimated direction. Indeed, the computed short path avoids opposite flows and in a crowd environment and follows existing human flows.

While formalizing the *Freezing Robot Problem*, in 2010 Trautman and Krause [1] reached the conclusion that a viable path-planning model had to take joint avoidance into account. Indeed, the state of the art at the time considered each agent independently: the robot predicted the trajectory of other agents assuming that agents would not change their behaviors when crossing the robot. Based on this result, the Optimal Reciprocal Collision Avoidance [6] was developed. ORCA can be used on top of a classic path-planning algorithm, such as A* or Dijkstra. If a collision is predicted in the near future both agents will avoid the collision by each of them taking evasive manoeuvres. The objective is to avoid collision while minimizing the effect on each agent, rather than forcing a single agent to take all of the responsibility (thus triggering the freezing robot problem). The work by Stein et al. [7], like the previously described learning approaches, draws upon the idea that humans are able to effectively navigate crowds. One finding by [8] is that in crowds people walking in the same direction tend to follow each other, thus creating corridors of unidirectional flow. [7] exploits this by looking around for humans and selecting one as a leader, which is then followed through the crowd, exploiting the gap created by the leader's passage. When the selected pedestrian is lost, or is no longer deemed to be a good leader, the robot either selects another pedestrian or adopts another navigation strategy until a new leader is found. The leader is selected through a set of criteria (e.g. walking towards the desired goal, not moving too fast or too slow, not visibly being made uncomfortable by the robot).

However, these methods have strong limitations when they are used in dense crowds. The approach of Stein et al. relies on finding a leader, or at least another agent that is going in the same direction as the robot. In cases where the robot is alone and facing a contra-flow, the approach would be limited. The approach of Jumel et al. assumes that the environment is known, which is not entirely realistic since real life situations are non-deterministic and irregular behaviours are common. Although reciprocal avoidance (ORCA) largely mimics real life situations where people jointly share collision avoidance, it cannot be completely replied upon; as we see in real life, although collisions between people are rare, they do happen. Finally, previous approaches have problems generating socially acceptable trajectories where pushing through a social group, such as a family contravenes social norms.

III. Effective Avoidance Combination Strategy

Effective Avoidance Combination Strategy (EACS) is a middle-term planning method proposed by Pettré and Bruneau in [9]. It is a multi-agent method originally aimed at crowd-simulation. In order to take the best decision, the agent (i.e. a pedestrian) must take into account all of its immediate surroundings and evaluate the consequence of each avoidance move in order to make the best choice, rather than being near-sighted and dealing with anticipated collisions one by one.

[9] concerns crowd simulation and tries to make simulated pedestrian movements as close to reality as possible. Considering the assumption that a human-imitating robot will travel effectively among a crowd, it was natural to consider adapting this method for robot navigation. In their paper, the authors proposed a middle-term strategy in order to find the best succession of avoidance manoeuvres for a short sequence of moves. The number of moves is determined by the field of view and occlusion of the agent, and the final objective is to reach a goal, i.e. a defined position.

To do so, the agent first computes a path to its goal (figure 1.a). Then, it considers the first agent, in green, and evaluates if a collision will occur with the computed path. If it is the case, candidate waypoints are considered around the potential collision point (4 cardinal points, figure 1.b). 4 candidates are respectively selected at a time by applying the following robot actions: accelerating, decelerating, turning left, turning right. Each of these candidates are then tested to find a valid path to the goal (without a potential collision with other agents,figure 1.c and d). If so a path is found, elsewise four new waypoints are created and the algorithm re-iterates.

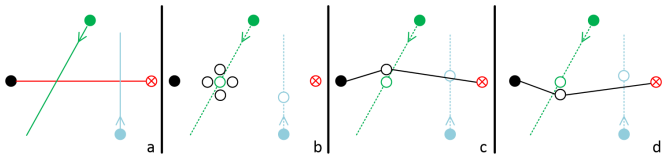


Fig. 1: EACS path selection steps. The black dot is the current agent, the red dot is the agent's objective, the green and blue agents are other agents that cross the path

This approach allows for more complex behaviours than immediate social force-based collision avoidance (e.g. choosing an empty but further away corridor if the closest one is crowded, or finding a sparse "corridor" among a moving crowd and taking it).

In this approach there is no uncertainty about the position and velocity of other agents. A direct consequence is that the four new way-points are built as the close to the original collision point but still avoiding any collision. The other agent will be avoided without any "safety margin": the EACS-running agent will respect the minimal distance with regards to any other agent, and nothing more.

By dodging the collisions with minimal margin, even the slightest reorientation of the other agent could put both agents on a collision course again. As a consequence, EACS becomes very unstable in dynamic environments. This will force the EACS-running agent to constantly recompute its path. This is a problem, not only because of the high computational cost, but also because it totally negates the point of EACS, which is to be a middle-term planner. Moreover, the resulting trajectories are usually far from adhering to socially acceptable behavior. The path

it produces should have at least some consistency and be valid for several seconds (the span of such middle-term planning). The algorithm does not avoid high-density areas, as it often finds an opening in the crowd. However, such openings are always short-lived, disappearing long before they are reached by the robot. As a consequence, a new path needs to be found, leading to either a total change of direction without any efficient progression towards the goal or getting stuck in high-density areas, zig-zagging and causing great disturbances to the agents around it.

In order to overcome such behaviour, we developed the following method that tries to avoid such high density and highly dynamic areas.

IV. Clustered EACS based navigation

The aim is to create a socially acceptable trajectory. The approach can be succinctly described as follows. We first gather agents that share the same behaviors (velocity, orientation, position) into clusters. The resulted clusters intend to average individual behaviors. These clusters are treated as convex polygons. This effectively treats the crowd both discretely (as each unclustered agent will be treated individually) and as a continuum (since the polygons are seen as one single mass, independently of how many agents they contain). Finally, we navigate through this new representation of the crowd by dodging the polygons with as safe a margin as possible. Finally a best path is selected. The basic algorithm is shown below, with further details given in Algorithm 2 concerning collision checking.

Algorithm 1: Social Aware Trajectory computation

input : $start_pose$: current robot position,
$goal$: goal targeted

begin
- Compute trajectory from $start_pose$ to $goal$
- Detect pedestrian positions and velocities ($a \in A$) Fig. 2 a)
- Compute Cluster $c \in C$ with DBSCAN Fig. 2 b)
- Compute polygon $p \in P$ associated to each cluster c Fig. 2 c)
- Create a container of possible paths $path = []$
- CheckCollision($start_pose$, $goal$, P,$path$, [])
- Get best path $best$ in $path$
- Apply best path $best$

end

A. Clustering the crowd

1) DBSCAN Clustering Method: We apply the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) defined in [10]. This clustering algorithm will, in an unsupervised way, classify groups of agents (clusters) that are close to each other with regards to a given distance, noted $dist(a_1, a_2)$ where a_1, a_2 are agents. The DBSCAN method offers good performance and is faster than other approaches like EM-algorithms, and detects complex pattern

Algorithm 2: CheckCollision algorithm

input : $start_pose$: current position,
$goal$: goal targeted,
P list of polygons,
$path$ list of valid paths,
$current_path$ current path in construction

begin
- collision=false
- **for** p_i ***in*** P **do**
  - **if** *Collision between* $start_pose \rightarrow goal$ *and* p_i **then**
    - Compute dodge way points W Fig. 2 d)
    - **for** w_i ***in*** W **do**
      - $current_path.add(w_i)$
      - CheckCollision(w_i, $goal$, P,$path$, $current_path$) Fig. 2 e)
    - **end**
    - collision=true
  - **end**
  - **if** *!collision* **then**
    - $path.add(current_path)$ Fig. 2 f)
  - **end**
- **end**

end

without the need for kernel-clustering. In addition it can also define elements that do not belong to any cluster.
The DBSCAN algorithm ascribes one of 3 status to each agent: "CORE" if the agent is on the inside of its cluster; "BORDER" if the agent is on the edge of its cluster; and NOISE if the agent is isolated with regards to the given distance, i.e. it does not have enough neighbours. At the start of the algorithm, all of the agents are marked with the status not_treated. The algorithm takes two parameters: $MinNeigh$ and $NeighDist$. Let A be the set of all agents, and $N(a_1)$ with $a_1 \in A$ the set of neighbours of a_1. $NeighDist$ is the considered radius of the neighbourhood around an agent :

$$\forall a, b \in A, b \in N(a) \iff dist(a, b) < NeighDist$$

$MinNeigh$ refers to the minimal number of neighbours an agent should have to be considered as a CORE agent of its cluster.

$$\forall a \in A, status(a) = CORE \iff |N(a)| \geq MinNeigh$$

A high value of $NeighDist$ leads to a low-density large clusters (in distance), Whereas, a high value of $MinNeigh$ leads to more dense clusters.

An agent will be classified as a BORDER agent if at least one of its neighbours is a CORE, and it will be put in the same cluster as the CORE. An agent that is not neighbour with a CORE will be classified as NOISE.

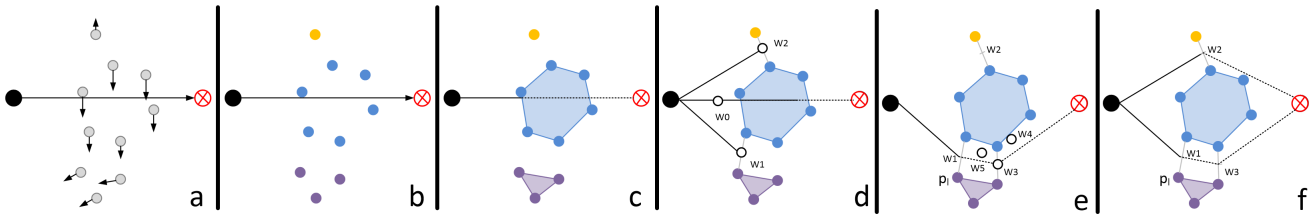


Fig. 2: Social aware trajectories steps. a) refers to the agent position and velocity detection (Alg. 1 l.3), b) clusters agents through position and velocity (Alg. 1 l.4), c) computes polygon associated to cluster (Alg. 1 l.5), d) detects collision (Alg. 2 l.4) and computes dodge points if needed (Alg. 2 l.5), e) and f) iterates until all dodged points are computed (Alg. 2 l.8)

2) DBSCAN in the navigation problem: In our method we first apply the DBSCAN algorithm to the list of all the pedestrians (or any agent in a case where other robots are present). The idea of using DBSCAN to cluster crowds comes from [11].

In our context, the clusters represents pedestrians sharing a common behavior, evolving as a continuum. For this purpose, the Euclidean distance could not be the only parameter to characterize the clusters. The following crosswalk example highlights why: Two groups of roughly the same number of people cross each other in opposite directions. Their position when they reach the middle of the crosswalk will be very close, so if the DBSCAN used only the Euclidean distance it will see one big group, of average speed zero, leading to a prediction of one huge immobile group. In this situation, the different behaviours of the two groups (different orientation and velocity) are not detected. Thus, the velocity parameter needs to be taken into account in our DBSCAN distance computation.

For A the set of agents, let $v(a)$ be the velocity of a given agent a and $p(a)$ be the position of that agent. The distance we chose in this work is a linear combination of velocity and position. λ is a parameter that weights the importance of the velocity in the computation of the distance.

$$dist_\lambda : a, b \rightarrow \frac{1}{1+\lambda} \left(\|p(a) - p(b)\|_2 + \lambda \|v(a) - v(b)\|_2 \right)$$

B. Turning the Clusters into Polygonal Continua

The crowd is now clustered into groups that share a common behaviour. All of the members of each group have a certain coherence in terms of position and a similar speed. These similarities are used to remove some of the uncertainty of the problem. To do this we turn the clusters (sets of discrete agents) into polygons (continua). This defines, for each cluster, an area on the map that contains all of the agents of the cluster, like a footprint. The polygon we chose to represent each cluster is the convex envelope of the set of positions of each of the members (agents) of the cluster. These are the polygons that we will navigate around. A cluster represents high pedestrian density that an agent has to avoid. Geometric operators can be applied on such polygons, allowing an agent to easily detect group borders.

There are several advantages in choosing to work with complex polygons: First of all, convex polygons are simple to manipulate. They have a small number of corners yet provide a lot of information (unlike bounding boxes, which have fewer corners, but are very coarse). This eases the computational charge during the next steps of the method. Moreover, when viewed from a certain point in space (e.g. the robot's sensor), the leftmost and rightmost agents of the cluster (with regards to the said point) are part of the convex hull, and are therefore easy to find.

We have represented the crowd as a set of polygons and lonely agents. Therefore, instead of individually avoiding all of the agents contained in the clusters, we define an average behaviour that will be applied to the polygon in order to predict the future of that polygon. We consider that the polygon will travel in a linear motion at the average speed of the agents in the cluster for at least a few seconds. Future work could take acceleration into account, or anticipate the member's goals in order to predict the polygon's future position.

C. Cluster dodge distance

As presented in section III, a main limitation of the EACS method is that it does not produce a socially acceptable trajectory since the robot moves uncomfortably close to humans when trying to avoid them. In our approach, when a the robot detects a collision (figure 3.a) with a cluster (polygon), it computes different avoidance manoeuvres (as in EACS). The acceleration and deceleration operations are similar to the original algorithm. Concerning the "turning left" and "turning right" operations, our method detects the upper a_u and lower a_l pedestrians in the cluster (figure 3.b). In order to define new possible way points, the closest pedestrian of other clusters (respectively p_u and p_l) to the points a_u and a_l are retrieved. The resulting possible way points (w_u and w_l) are then located in the middle of the segments [p_u,a_u] and [p_l,a_l] (figure 3.c).

During this process, some exceptions could appear. If the robot could not reach the new way point with an acceptable speed (< max speed), a maximum dodge distance is applied on the segment (e.g [p_l,a_l]). If the segment crosses the cluster (e.g in the case of no close neighbor cluster), the maximum dodge distance is applied.

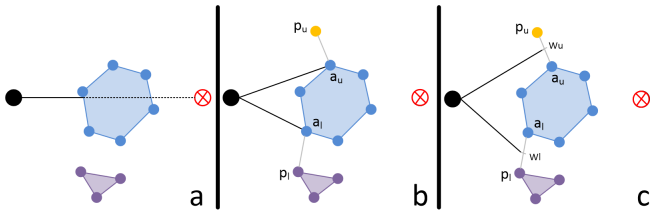


Fig. 3: Cluster dodge process. The black dot represents current robot position, the red dot is the robot's goal, other dots are pedestrians.

Such approach is close to the Voronoi edges for navigation found in [12] and [13]. However, these articles deal with a static environment, with immobile obstacles (e.g. walls, furniture). In our approach, we consider moving polygons, as well as estimations of where and of what shape the polygons will be in the near future.

What motivates this representation is seeing the polygons as moving high-density areas of the map. Such areas should be avoided due to the high uncertainty of the environment inside them. Indeed, going inside a polygon will expose the robot to highly unstable terrain, where all path planning would be canceled by a flurry of immediate collision avoidance measures.

V. Simulation and Results

A. Crowd simulation

All of the experiments were run using Pedsim [14], an open source crowd simulator that implements Helbing's social force model [8]. This allowed us to tune our algorithm in an environment filled with agents (pedestrians) with seemingly realistic behaviours. We worked with a pedestrian-heavy environment. Predicting the behaviour of a human in a crowded environment will in most cases not yield good results for more that a few seconds. Pedsim uses a random force that allows the simulated agents not to have too rigid a behaviour. In addition the force model in itself already yields satisfactory behaviours with sharp turns and sudden collision avoidance. In this regard, the simulator implements uncertainty very well.

B. The robot

The simulated robot, which executes our navigation algorithm, is a human sized, circular shaped robot with a tunable radius. We chose a radius around 30 cm, which is realistic for robots such as SoftBank Robotics' Pepper or autonomous wheelchairs. Pedsim implements only forces between people. However, we are currently working on extending Pedsim in two directions: firstly so that human-vehicle interaction is taken into account and secondly in increasing the size of the robot in order to represent an autonomous vehicle. Currently though, other shapes are easily implementable by modifying the collision-detecting function, given that the right forces are also coded as an extension of the simulator. The robot navigates at a human-like speed, with a configurable maximum speed around 1.5 m/s. It is also considered holonomic, (i.e. when the number of controllable degrees of freedom is equal to the total degrees of freedom) and can turn sharply if necessary. Finally, physical constraints such as inertia and bounded acceleration were not taken into account due to:

- the reduced maximum speed (thus the low maximum amplitude of changes in velocity) of the simulated test subjects that allows for extremely quick variations in velocity
- the light weight of the simulated test subjects that allows great manoeuvrability

C. Results

The parameters, λ, $MinNeigh$ and $NeighDist$ of the DBSCAN were tuned by hand on the crosswalk scenario inside the simulator. To do this we created two groups of ten actors with opposite objectives, and ran the algorithm, comparing it with human observations from the angle of the robot.

The values that yielded the best results were:

- $\lambda = 5$
- $MinNeigh = 1$ or $MinNeigh = 2$
- $NeighDist = 1$

The big value for λ can easily be explained by the fact that velocity is the preponderant feature of an actor's behaviour, and is usually more stable than precision. Indeed, a given speed indicates a desired objective (e.g. the other side of the street, a specific street entrance, a specific shop).

Position still needs to be taken into account, nonetheless, in order to give an upper bound to the distance. Indeed, even though two pedestrians walk at the exact same speed, we can pass between them if there are several free meters available to the robot.

The small value for $MinNeigh$ signifies that an agent needs to be close (in terms of our custom distance function) to another agent in order to be in the same cluster. In a dense crowd if we consider people as being in "rows", only the closest and part of the second row to the robot will probably be seen and the rest will be ignored. Agents deeper into the crowd will not be taken into account.

This limits the clustering to the border of the crowd closest to the robot, as only a "line" of people will be detected. Thus, the robot will only see one or two neighbours even if there are many more agents hidden behind them.

A video showing the differences between EACS and Clustered EACS on 3 scenarios is available [1].

Figure 4 shows the trajectories generated by both EACS and Clustered EACS. In this simulation, 2 groups of people are walking, one from left to right, the other from the bottom to the top of the image. The robot starts from the top of the image to reach the bottom. The robot using EACS (top images) finds its way among the group of people going up, then crosses the other group and after a few oscillations,

[1] https://youtu.be/2OOaXblkqOs

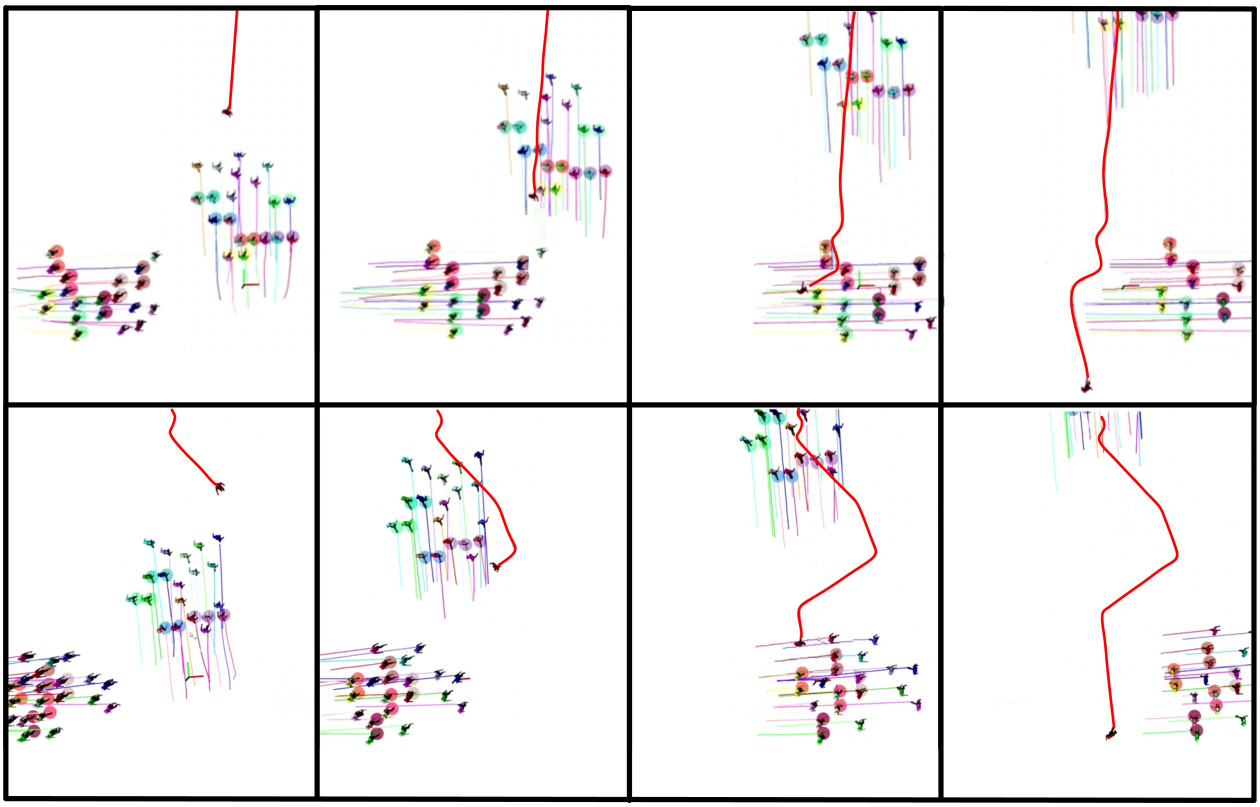


Fig. 4: Comparison of a trajectory produced by EACS (top) and Clustered EACS (bottom)

reach its goal. The robot using the clustered EACS method, anticipates the incoming flow of people and avoids the group by passing on its right and the second group by passing on its left. It does not try to enter the group. Although the trajectory produced using EACS is more direct, it is clear that the robot passes extremely close to people, possibly breaking through social groups. Although the trajectory of the clustered EACS method is longer, it respects social groups and is more socially compliant by adhering to social norms.

VI. Conclusion

In this article, we proposed two main concepts: a new way to represent the crowd as seen by the robot, and an algorithm to navigate this representation. The first point has the objective of obtaining, from incomplete observations, a reliable and dynamic way of anticipating the shifts in density around the robot. This allows the robot to avoid the most dense areas, which are the most dangerous in terms of the probability of collision. These are areas in which no safe path can be computed with any certainty and that will last long enough for the robot to travel.

We also implemented an algorithm that is able to- https://www.overleaf.com/project/5bf66905a5705a1596087f74 navigate such a representation. This was done by maximizing the avoidance margin with the dynamic obstacles previously computed, and sliding over dynamic Voronoi edges.

VII. Acknowledgement

This research has been conducted as part of the HI-ANIC (Human Inspired Autonomous Navigation In Crowds) project, funded by the French Ministry of Education and Research and the French National Research Agency (ANR-17-CE22-0010).

References

[1] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot navigation in dense human crowds: the case for cooperation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 2153–2160.

[2] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier, "Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes," in *IEEE/RSJ 2008 International Conference on Intelligent RObots and Systems*, Nice, France, 2008.

[3] C. Fulgenzi, A. Spalanzani, C. Laugier, and C. Tay, "Risk based motion planning and navigation in uncertain dynamic environment," INRIA Rhone-Alpes, Research Report, Oct. 2010.

[4] F. Jumel, J. Saraydaryan, and O. Simonin, "Mapping likelihood of encountering humans: application to path planning in crowded environment," Sep. 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01588815

[5] K. Mardia and P. Jupp, *Directional Statistics*. John Wiley and Sonsons Inc., 2000.

[6] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. A. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Int. Symp. on Distributed Autonomous Robotics Systems (DARS)*, 2010.

[7] P. Stein, A. Spalanzani, V. Santos, and C. Laugier, "Leader following," *Robotics and Autonomous Systems*, vol. 75, no. PA, pp. 79–95, Jan. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2014.09.028

[8] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical Review E*, vol. 51, pp. 4282–1286, 1995.

[9] J. Bruneau and J. Pettré, "EACS: Effective Avoidance Combination Strategy," *Computer Graphics Forum*, vol. 36, no. 8, pp. 108–122, Dec. 2017. [Online]. Available: https://hal.inria.fr/hal-01392248

[10] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 226–231. [Online]. Available: http://dl.acm.org/citation.cfm?id=3001460.3001507

[11] I. Chatterjee and A. Steinfeld, "Low cost perception of dense moving crowd clusters for appropriate navigation," in *Workshop on Social Norms in Robotics and HRI, IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2015.

[12] A. T. Rashid, A. A. Ali, M. Frasca, and L. Fortuna, "Path planning with obstacle avoidance based on visibility binary tree algorithm," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1440–1449, Dec. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2013.07.010

[13] N. S. V. Rao, "Robot navigation in unknown generalized polygonal terrains using vision sensors," *IEEE Transactions on Systems Man and Cybernetics*.

[14] B. Okal, T. Linder, D. Vasquez, S. Wehner, O. Islas, and L. Palmieri, "https://github.com/srl-freiburg/pedsim_ros," 2017.