EU Horizon 2020 Research & Innovation Program

Advanced Robot Capabilities & Take-Up

ICT-25-2016-2017

Safe Robot Navigation in Dense Crowds

http://www.crowdbot.org

Technical Report

D 4.3: Crowd Simulator – Final Version

Work Package 4 (WP 4)

Simulation Tools for Robot Navigation in Crowds

Task Lead: INRIA France

WP Lead: INRIA France

Table of Contents

List of Figures

List of Tables

Executive Summary

This report details the crowd simulation software and tools that were developed for the CrowdBot project and presents their last version reached at the end of the project, up to month M42. The development of the crowd simulation tools was mainly performed in the frame of Workpackage 4 of the project. The simulation tools for robot navigation in crowds were developed with two key roles in the project:

1 Crowd simulation for short-term prediction of the evolution of the situation of people in the vicinity of the robot.
2 Crowd simulation for testing and evaluating the navigation functions of a robot in a densely populated environment.

This deliverable reports on the developments made by the partner Inria to meet these two objectives. This deliverable is the last one about WP4, thus providing a description of simulation tools in their last version. All these tools are open source, freely accessible to the community.

The report starts with simulation tools for the purpose of performing short term prediction of the trajectory of people around the robot. This problem is known in the community as the one of human trajectory prediction HTP. Since the beginning of the CrowdBot project, in 2018, the development of deep learning modeling approaches has revolutionized the field, moving it from knowledge-based models to data-driven ones. CrowdBot actively participated to this change, and the current report details our HTP solution called "Social-ways" (Section 2.1). Social-ways is based on a GAN approach (Generative Adversarial Networks) that is trained in a special way to avoid the mode-collapse problems, that tend to average together possible trajectories corresponding to different choices (e.g., turning left or right).

The report then continue with Section 2.2 that is dedicated to another kind of prediction problem, which is the one of the presence of the human in zones which are occluded to the robot sensors. This problem is quite new in spite of its importance to perform more accurate prediction of the short term trajectories of people around the robot. One can understand that those trajectories can be influenced by their own neighbors, potentially unseen by the robot. It should be said as well that a robot of moderate height, such as the ones considered in the CrowdBot project, will be able to perceive the position of only a few people at direct proximity. The approach we present to solve the problem is data-driven, and based on prior knowledge on the patterns of people that crowds may exhibit.

Section 3 is dedicated to simulation-based evaluation tools. The objective is slightly different from the previous one. The objective is to study, in simulation, the behavior of robots in a large set of different situations, covering as far as we can all possible configurations of the neighborhood. Unlike HTP problems where an accurate prediction of individual trajectories is expected, we here try to cover all possible evolutions of a large set of configurations. Section 3.1 and 3.2 present the objectives and overview of the Crowd-robot simulation.

Section 3.3 details UMANS, our internal crowd simulation software the specificity of which is to be able to reproduce a number of existing algorithms in one single architecture. Section 3.4 presents how crowd simulation is extended to include robot navigation as well as immersive capabilities. Immersion of real users in the crowd-robot simulation is presented as a way to palliate limitations of humans simulation that can't cover all possible human behaviors and specific reactions to a robot. Section 3.5 contains information about the CrowdBot Challenge developed during the project.

Finally, conclusion discuss direction for future work in the area.

1 Introduction

The CrowdBot project addresses the problem of robot navigation in crowds. This problem is specific in two ways. The first is the dynamic nature of the environment. The presence of moving people around the robot makes its environment almost entirely dynamic. The close proximity of the people around the robot may also prevent it from perceiving its distant environment. The great variety of human behaviors and movements makes it very difficult to plan the robot's action and movement, which is subject to numerous potential collisions. Second, the development of robot navigation techniques is, itself, made difficult by the ethical issues raised by the need to test the robot in potentially dangerous situations.

The need for human motion simulation techniques is therefore useful in two ways in the project. Firstly, to enable the robot to predict the evolution of its environment. From a situation perceived by the robot, even if it is very partial because of the limitations of the perception techniques, it is interesting to make a short-term prediction of this situation. It will be useful to detect the most critical cases of human-robot interactions, and to avoid causing the robot to cause dangerous collisions with its human neighbours, to begin with. This prediction is limited by the predictive capabilities of the robot. Based on a set of on-board sensors, the dimensions of the robot do not generally allow it to perceive the presence or situation of humans beyond a first row of people around the robot. Thus, in parallel with a temporal evolution of the situation of the people around the robot, it is relevant to also make a prediction of the presence of humans beyond the robot's perception threshold. The presence of these people also influences the evolution of the situation of the people around the robot.

The first usage of crowd simulation techniques is thus to perform predictions on the state of the crowd around a given robot at a given time, and is used to plan and to control the robot motion in such an environment. It can also be used to test and evaluate the robot capabilities in various situations, in place of real state. Instead of performing predictions from a given state, the idea is thus to generate situations and compute their evolution to evaluate if the robot is capable of handling them correctly and safely. The difficulty in predicting a large variety of human behaviors and interaction modes with robots is however quite limiting studies in simulation. Virtual Reality is a mean to immerse real users in the simulation to extend its results with behaviors performed by real users.

In this document, we report on simulation tools development efforts along all these directions. Section 2 is dedicated to spatial (beyond perception capabilities) and temporal predictions of an observed situation. Section 3 is dedicated to evaluation tools with possibilities of immersive VR.

2 Crowd Prediction

2.1 Human Trajectory Prediction

2.1.1 Introduction

In this section, we address the problem of (short-term) human trajectory prediction (HTP) in crowded scenarios: "Given the motion trajectories of one or multiple pedestrians in a scene, surrounding a robot, it needs to predict, in real-time, the Spatio-temporal state of the pedestrians, and infer their short or mid-term intentions (destinations)".

The predictions will provide an interface for the Navigation module of the robot, to decrease the risk of collision with pedestrians in the environment. In addition, an accurate prediction can help improve the visual tracking of these individuals.

The trajectory prediction is based on the history of the two-dimensional location of pedestrians within the world- or robot- coordinate system, which is assumed to be provided by the perception and tracking system of the robot.

Due to the unpredictable nature of human motion, the prediction of crowd motion could be very challenging. Trajectories of single pedestrians are affected from both internal variables of the person of interest (POI) and the environmental factors. Without claiming exhaustivity, we could mention the nature of the surrounding obstacles and their spatial distribution, the nature of the ground, the long-term goal of the pedestrian, their age and mental state, and etc. Then, to make things even more difficult, the motions of a whole set of agents sharing a common space are dependent, through a whole range of interactions that can go from avoidance to meeting intention or person following.

2.1.1.1 System Input

We assume that each scene is first preprocessed to obtain the spatial coordinates of all people at different time instants, in a global coordinate system. At any time-instant *t*, the *i*_th person in the scene is represented by his/her XY-coordinates x_t^i, y_t^i. In this document, we denote a trajectory from time instant t_0 to t_1 as $\mathbf{x}_{t_0:t_1}$. We give all the N observed trajectories $\mathbf{x}^i_{-T_{obs}:0}$ to the system.

Note that due to implementation complexities, the system only accepts trajectories with length $T_{obs}+1$. The longer trajectories should be cut and the pedestrians appeared in the scene at t: $-T_{obs} < t < 0$, they need to be extrapolated (e.g., with a constant-velocity model), otherwise they will be skipped.

2.1.1.2 System Output

The output of the system is the most likely locations of the N detected pedestrian in the desired time window i.e., time instants 1 to T_{pred}.

We set $T_{obs} = 3s$ to use the last 3 second observations of the pedestrians for making the predictions, and the trajectory predictions are done for approximately the next 5 seconds. This prediction can be used by the robot to make safe and efficient short-term motion plans.

2.1.2 Trajectory Prediction with GANs

In our proposal we considered three design principles:

a Modeling social interactions between agents:

As explained above, the motion of a pedestrian might be impacted by its surrounding agents, so we need to consider the interaction between agents in a prediction system. Collision avoidance, grouping behavior, and leader-follower behavior are few examples of interactions between agents.

b Data-driven prediction:

If we look at the new trends in this field, we will see a growing tendency in the research of data-driven methods. The main advantage of this method is that it allows the natural human-human interaction to be captured and learned automatically by using real-world pedestrian data. Machine learning techniques and in particular recursive neural networks (RNNs) such as Long Short-Term Memory (LSTM) networks are a suitable choice to learn pairwise human-agent interactions.

c Handling multi-modality:

Furthermore, we know that the prediction problem does not have a unique solution. There are multiple plausible future trajectories at any given instant due to latent goals of agents and multiple paths to each goal. An example is shown in **Figure 1**.

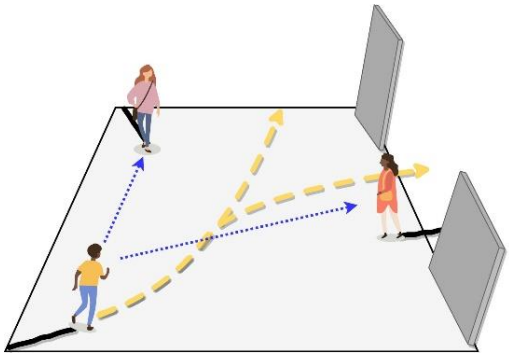

Figure 1. Illustration of the multi-modal trajectory prediction problem. Having the observed trajectories of a pedestrian of interest (yellow-shirt), and that of the surrounding pedestrians, the system should be able to build a predictive distribution of possible trajectories (here with two modes in dashed yellow lines).

Therefore, the models that can generate multiple 'plausible' predictions will be more interesting. This is most commonly done by sampling generative models such as Generative Adversarial Networks (GANs).

A GAN is a class of machine learning systems composed of two separate neural networks: a Generator and a Discriminator. The two networks compete with each other in a game. Given a training set, this technique learns to generate new data with the same statistics as the training set. If the model is conditioned on a given variable, such as an observed trajectory, the model will be called "conditional GAN" (see **Figure *2***).

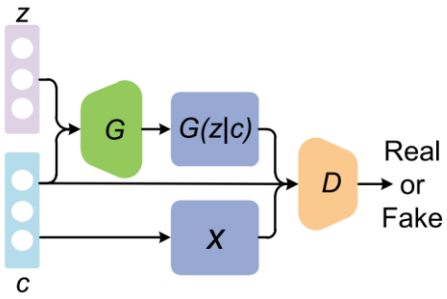

Figure 2. Overview of a general conditional GAN for handling multi-modality problem.

Very briefly, here are the steps that happens in a conditional GAN:

- The generator takes in a vector of random numbers, aka noise signal (z) and the observation, aka condition (c) and returns a prediction trajectory, which is called the fake prediction ($G(z|c)$).
- This prediction is fed into the discriminator alongside the real prediction taken from the database (x).
- The discriminator takes in the observation and both real and fake predictions and returns probabilities, a number between 0 and 1, with 1 representing a prediction of authenticity and 0 representing fake.

Accordingly, we propose a GAN solution for proposing plausible future trajectories of pedestrians in the horizon of a few seconds, given a set of observations of their own past trajectories and of those of the pedestrians sharing the same space. It naturally encompasses the uncertainty and the potential multi-modality of the pedestrian steering decision, which is of critical importance when using this predictive distribution as a belief in higher level decision-making processes.

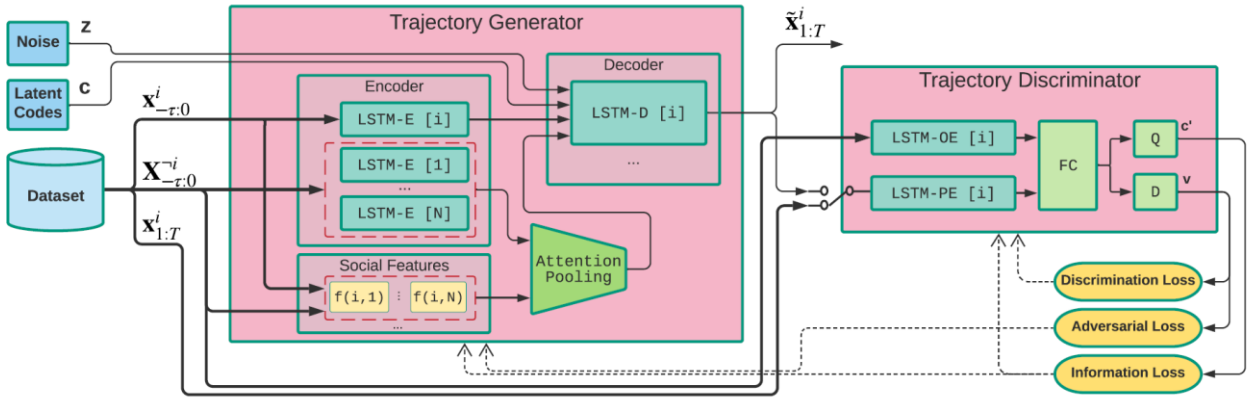

Figure 3. The proposed GAN-based system architecture for human trajectory prediction.

2.1.2.1 System Components:

The architecture of the proposed system for human trajectory prediction is shown in **Figure *3***. Here we overview the system components.

2.1.2.1.1 Trajectory Generator

The trajectory generator is designed to receive the following four inputs:

1 The observed trajectory of POI: x^i ($.^i$ denotes the pedestrian of interest or POI).
2 The observed trajectory of the surrounding pedestrians: (denotes all pedestrians except *i*).
3 The latent codes (**c**) : this signal will ideally control some properties of the generated samples, and by changing it we can obtain different prediction samples (it will be explained in the training section).
4 The noise signal (**z**) : this signal will give some other degrees of freedom to the system to generate various samples.

The generator is composed of four sub-networks:

1 Encoder (LSTM-E's): for encoding the observed trajectories, regardless of their interdependencies.
2 Social Features: for computing the reciprocal features between each pair of the pedestrians (i, j).
3 Attention Pooling: to find the impact of each surrounding neighbors on each agent, using the social features.
4 Decoder (**LSTM-D**): the block that generates the prediction sequence.

2.1.2.1.2 Social Features

We calculate the following values between each pair of agents and feed them to our architecture as social features:

- Euclidean distance between pair of agents
- Bearing angle
- And distance-to-closest-approach (DCA), a concept from biomechanics, illustrated in figure below.

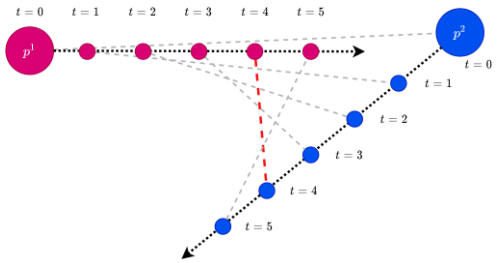

Figure 4. Illustrating DCA (distance-to-closest-approach) between two agents p^1 and p^2. The closest approach takes happen at t=4 which is shown by the red dashed line.

2.1.2.1.3 Attention Pooling

The influence of the other agents on agent i is evaluated by encoding the vector through LSTM-E, and by applying an attention weighting process that produces weights $\mathbf{a}^i[a^{i1}, .., a^{ij}, ..., a^{iN}]^T$ for agent i. They are defined for $j \neq i$, based on predefined geometric features. A social feature vector between agent i and j $\delta^{ij} \in \mathbb{R}^3$ is computed by stacking the social features, described above. In **Figure *5***, an crowd interaction example is shown in which the impacts of pedestrians p_2, p_3, p_4 on pedestrian p_1 are estimated and showed using arrows with different widths.

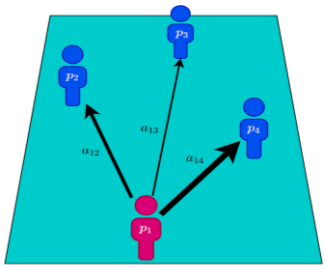

Figure 5. Attention Pooling mechanism in the prediction system assigns importance weights to interaction between agents.

2.1.2.1.4 GAN Discriminator

On the other side there is Discriminator which is situated on the right part of Figure 3. It takes as input either:

- a composite candidate trajectories for agent i, $[\mathbf{x}^i_{-\tau:0}, \tilde{\mathbf{x}}^{i,k}_{1:T}]$, or
- a ground truth full trajectory, $[\mathbf{x}^i_{-\tau:T}]$

For each of them it decides whether the prediction is fake or real. It will obtain this ability through learning the network. At the same time the objective of Generator is defined to 'fool' the discriminator, by improving its predictions.

2.1.2.1.5 Loss function for training

The discriminator outputs a value D(x) indicating the chance that the prediction is real or not. Our objective is to maximize the chance to recognize ground truth prediction as real and generated predictions as fake. i.e. the maximum likelihood of the observed data. To measure the loss, we use Binary-Cross-Entropy (BCE) between the discriminator outputs and the expected labels.

2.1.2.2 Encourage Diversity

One of the problems that we have considered in our architecture is sample diversity. In the context of generative networks, when the generator generates a limited diversity of samples, or even the same sample, regardless of the input, we call it a mode collapse. We implemented and tested different ideas to overcome this problem, and finally we chose to apply the idea of InfoGAN. We will see in the results section, the positive impact on avoiding the mode collapsing problem with respect to other versions of GANs. Info-GAN learns disentangled representations of the sources of variation among the data, and does so by introducing a new coding variable c as an input (see **Figure *3***). The training is performed by adding another term to maximize a lower bound of the mutual information between the distribution of c and the distribution of the generated outputs.

2.1.3 Implementation

First, note that all the internal FC layers of both the Generator and the Discriminator are associated with "LeakyReLU" activation functions, with slope 0.1.

Generator: comprises a first FC linear embedding μ of size 4 × 128, over positions and velocities. The Encoder block in Generator contains one layer of 128 LSTM units (LSTM-E). Using 2 continuous latent code, noise vector with length of 62, and pooling vectors of size 64, which totally gives a 256-d vector, the Decoder LSTM (LSTM-D's) then uses 128 LSTM units in one layer and 3 FC layers with size of 64, 32, 2 to decode the predictions. Weights are shared among LSTM layers with the same function.

Discriminator: uses two LSTM blocks (LSTM-OE and LSTM-PE) with hidden layers of size 128 to process both the observed trajectories (size $4 \times \tau + 4$) and the predicted/ "future" trajectories (size $4 \times T$); these outputs are processed in parallel with two 64×64 FC layers. Then they are concatenated in fed to two separate FC blocks: soft-classifier (D) [64×1] and latent-code re-constructor [64×2] (Q). Finally, τ and T are set to 7 and 12 respectively.

2.1.3.1 Training

We trained the GAN network with the following hyper-parameters setting: mini-batch size 64, learning rate 0.001 for Generator and 0.0001 for Discriminator, momentum 0.9. The GAN is trained for 20000 epochs. The system should be trained for use in a new environment, owing to the fact that it learns the statistical information from the observed trajectories. According to our tests, the training data should include a few minutes of recording (> 10 minutes) and the trajectories should ideally cover different plausible paths in the environment.

2.1.3.2 Source code

The network is implemented in Python and using PyTorch library (a scientific computing package for development of deep neural networks). The code is published on GitHub and is available to the public at the following address:

https://github.com/amiryanj/socialways

2.1.3.3 How to setup

As any other python code, this code can be run on any platform with python version >= 3.5. The required packages can be installed using pip package-management system:

```
$ pip install torch torchvision numpy matplotlib tqdm nose
```

For running the visualization scripts the following packages should also be installed:

```
$ pip install seaborn python-opencv
```

2.1.3.4 How to train the network

In order to train the model, the dataset should be given as a **csv** file containing all the annotated trajectories. The address to this file should be given to the training script (train.py). We use public HTP datasets to train the system (ETH + UCY). It is recommended to use a system that supports CUDA framework. The system needs to be trained for at least 100 epochs and this may take few hours on a standard computer with Nvidia GPUs.

2.1.4 Experiments

For evaluation of the proposed system, we tested both of them on real and artificial datasets. In this section we will first describe the datasets, then we will clarify the evaluation metrics and then we show the results.

2.1.4.1 Testing on real datasets

2.1.4.1.1 ETH BIWI Walking Pedestrians dataset:

This is a pedestrian trajectory dataset, containing two different footages (ETH Univ and ETH Hotel), recorded from birds-eye view and annotated manually with frame-rate of 2.5 fps. The dataset is composed of 650 tracks over 25 minutes. A sample image can be seen in Figure 6. In terms of crowd density these datasets are categorized as low-density.

Figure 6. Sample frames from ETH dataset. ETH-Univ (left) Hotel (right).

2.1.4.1.2 Crowds-by-Example Pedestrian dataset (UCY):

This is another pedestrian trajectory dataset containing 3 footages (Zara01, Zara02 and university students) totaling around 16 minutes. The videos are labeled manually with a rate of 2.5 fps (see **Figure *7***).

Figure 7. Sample frames from UCY dataset. Zara (left) University Students (right).

2.1.4.2 Evaluation Metrics

For evaluation of the accuracy of predictions we use the following two metrics:

1. Average Displacement Error (ADE): Average Euclidean (L2) distance between ground truth and prediction over all predicted time steps:

$$\mathrm{ADE}(\mathbf{x}^i_{-\tau:T}) = \frac{1}{T}\sum_{t=1}^{T} \|\mathbf{x}^i_t - \hat{\mathbf{x}}^i_t\|$$

where x is ground truth and x is predicted location at time t for ped(i).

2. Final Displacement Error (FDE): The distance between the predicted final destination and the true final destination at the end of the prediction period.

$$\mathrm{FDE}(\mathbf{x}^i_{-\tau:T}) = \|\mathbf{x}^i_T - \hat{\mathbf{x}}^i_T\|$$

2.1.4.3 Performance of the Prediction Model

In Figure 8, we give qualitative examples of the outputs and intermediate elements in our approach. We generated 128 samples with our method and the predictive distribution is shown with magenta points. In most of the scenarios (including non-linear actions, collision avoidance and group behaviors), the distribution has a good coverage of the ground truth trajectories and also generates what seems to be plausible alternative trajectories.

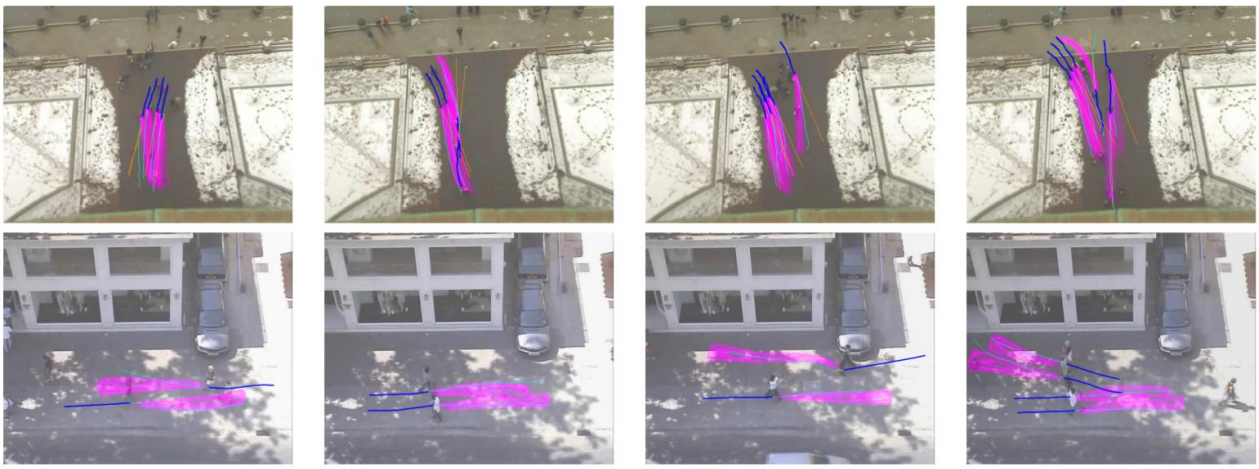

Figure 8. Illustration of our sample outputs (in magenta color) for ETH dataset - Univ (top) and UCY dataset Zara (bottom). The observed trajectories are shown in blue and ground truth prediction and constant-velocity predictions are shown in cyan and orange lines, respectively.

2.1.4.4 Toy Dataset to evaluate the quality of predictive distribution

As commented before, the GAN-based predictor and its training process are designed to preserve the modes of the predictive trajectory distribution and avoid the mode collapse problem. However, in the above datasets, there are very few examples of clearly multi-modal predictive trajectory distributions. Hence, we have created a toy example dataset to study the mode collapsing problem with stochastic predictors. This toy example is depicted in **Figure *9***.

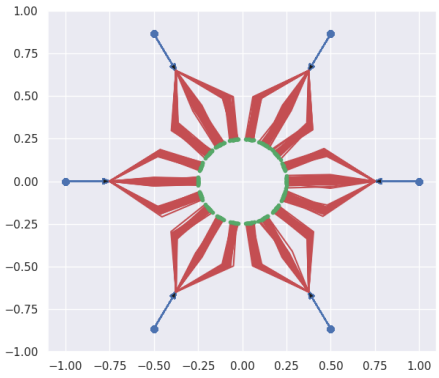

Figure 9. Toy trajectory dataset. There are 6 groups of trajectories, all starting from one specific point located along a circle (blue dots). When approaching the circle center, they split into 3 subgroups. Their endpoints are the green dots.

Given an observed sub-trajectory (blue lines), the Generator should predict the rest of the trajectory (red lines). Each of the 6 groups represents one separate condition to the system ($\mathbf{x}^i_{-\tau:0}$), and each of the 3 sub-groups represents a different mode in the conditional distribution $p(\mathbf{x}^i_{1:T}|\mathbf{x}^i_{-\tau:0})$.

2.1.4.4.1 Multi-modal Prediction Evaluation Metrics

For a more quantitative evaluation of generative models, we used the following two metrics to assess the set of fake trajectories versus the set of real samples. Given two sets of samples $S_r = \{\mathbf{x}^i_r\}$ and $S_g = \{\mathbf{x}^j_g\}$ with $N = |S_r| = |S_g|$ and $\mathbf{x}^i_r \sim P_r$ and $\mathbf{x}^j_g \sim P_g$ (generated samples):

1 Nearest Neighbor classifier (1-NN), used in two-sample tests to assess whether two distributions are identical. We compute the leave-one-out accuracy of a 1-NN classifier trained on S_r and S_g with positive labels for S_r and negative labels for S_g. As you can see in **Figure *10***, the classification accuracy for data from an ideal GAN should be close to 50% when N (the size of the samples) is large enough. Values close to 100% mean that the generated samples are not close to real samples enough. Values close to 0% mean that the generated samples are exact copies of real samples, and that there is a lack of innovation in such systems.

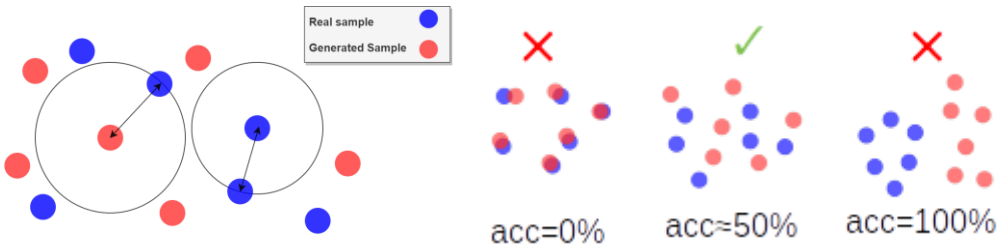

Figure 10. Evaluation metric using a Nearest Neighbor classifier.

2 The Earth Mover's Distance (EMD) between the two distributions. Which measures the minimum required effort to turn one distribution to another (see **Figure *11***).

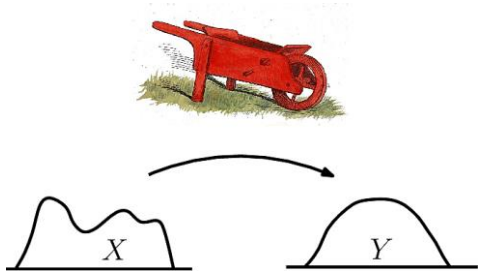

Figure 11. Evaluation metric using Earth Mover's Distance.

2.1.4.4.2 Other baselines for comparisons:

In order to compare our approach with other GAN-based techniques, we implemented several baselines. In all of them, the prediction architecture is the one we proposed without the attention-pooling; the GAN subsystem changes:

- Vanilla-GAN: This is the simplest baseline, where the Generator is just trained with the adversarial loss.
- L2-GAN In addition to adversarial loss, a L2 loss is added to the Generator optimizer.
- S-GAN-V20: The Variety loss proposed in Social-GAN method [Social-GAN] is added to the adversarial loss. This L2-loss only penalizes the closest prediction to ground truth among V = 20 predictions and gives more freedom to choose prediction samples.
- Unrolled10: Vanilla-GAN with the unrolling mechanism proposed in [Unrolled-GAN]. The number of unrolling steps is 10.

The results of the mode preserving for different baselines are shown in **Figure *12***.

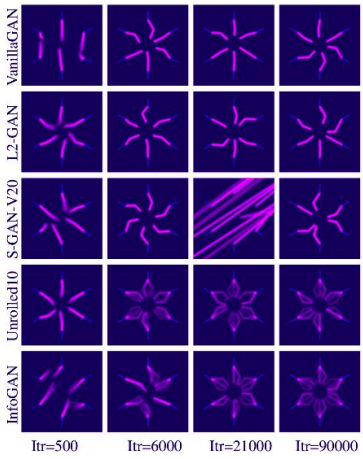

Figure 12. Results of learning baselines on Toy Example, for different numbers of iterations.

We computed both 1-NN and EMD metrics on our toy dataset with $|S_r| = |S_g| = 20$ (see **Figure 13**). We added evaluations for a few combinations of the aforementioned baselines (e.g., Info-GAN+unrolling steps or Unrolled+L2). The lower 1-NN accuracy of our approach using Info-GAN shows its higher performance for matching the target distribution, compared to Vanilla-GAN and other baselines. It is worth noting that the fluctuations in the accuracy are related to the small size of the set of samples.

As it can be seen, Unrolled10 and Info+Unrolled5 have also better performances, while it is obvious that by adding L2 loss, the results are getting worse. The results of the EMD test also proves that both Info-GAN and Unrolled10 offer more stable predictors with lower distances between the fake and real samples. There is no evidence that the Variety loss offers better results than a Vanilla-GAN.

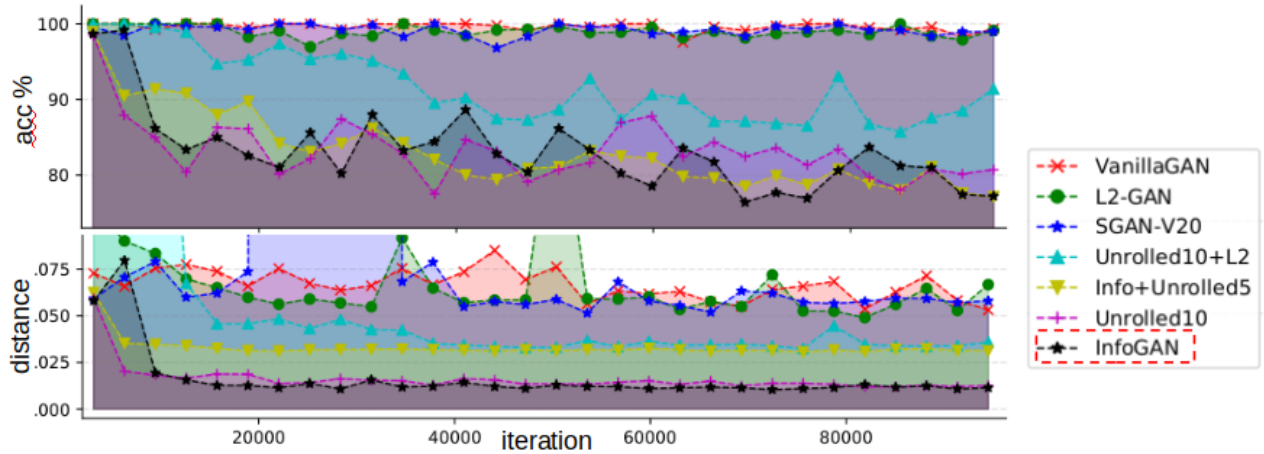

Figure 13. Statistics for different GAN implementations over training iteration. Top: 1-NN accuracy metric (closer to %50 is better). Bottom: Earth Mover's Distance between generated and ground truth samples (the lower, the better).

2.2 Handling crowd occlusion by Crowd-Imputation:

In this section we describe another contribution of the CrowdBot project which is about addressing the problem of crowd occlusion in robot navigation. We consider the navigation of mobile robots in crowded environments, for which onboard sensing of the crowd is typically limited by occlusions. We address the problem of inferring the human occupancy in the space around the robot, in blind spots, beyond the range of its sensing capabilities. We call this mechanism crowd imputation.

2.2.1 Introduction

When dealing with this problem from the robot perspective, the input data have properties that make the prediction much harder, e.g. as opposed to more classical motion prediction problems with data coming from a static, bird-view surveillance camera. In particular, due to the low height of the sensor (e.g. LiDAR) installed on the robot, noticeable parts of the crowd can be occluded. Moreover, many pedestrians may remain undetected for long sequences of frames. Depending on the density of the crowd and the characteristics of the robot's sensor, the proportion of non-detected people can be negligible or severe.

This can impact the performance of the robot in predicting future collisions and building a safe and valid motion plan. An example of typical crowd perception from a mobile robot, with multiple occlusions, is presented in Figure 14. for a simulated mobile robot within a high-density crowd.

2.2.2 Problem Statement

"Given the robot observations about the crowd surrounding it and given a query point $\mathbf{x}$ (potentially out of robot's sight), what is the probability of presence of a person, that is not already detected, at $\mathbf{x}$?".

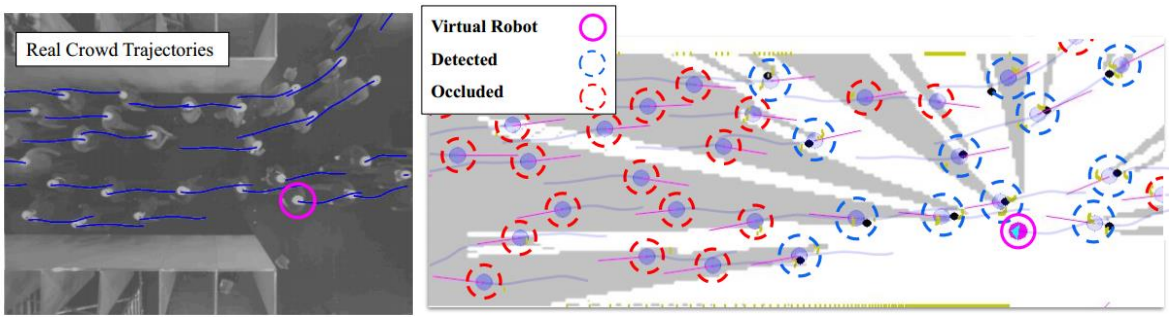

Figure 14. Occluded crowd: In the top left, a top-view image of a crowd is shown. The pedestrians form a bidirectional flow in a narrow corridor. If we simulate a robot through one of the pedestrians, there would be noticeable occlusion (gray areas in the right image) where the pedestrians are un-detected (red circles). The black dots within the blue circles show the detection locations, before filtering. Note: the crowd activity is recorded in the form of xy trajectories (blue lines).

2.2.3 Proposed Method

As we know a crowd can be associated with different properties such as the type of flow (laminar flow, turbulent flow, crossing flow, etc.), density (sparse, low-density, high density, etc.), the presence of groups and size and shape of the groups (see Figure 15).

Our proposal is to leverage the statistical patterns extracted from past observations over a surrounding crowd to estimate the probability of the presence of people in unseen areas, i.e., we perform statistical imputation of the occupancy levels in these areas.

Our model takes as input the stream of range data and the positions of the detected persons and gives as an output a prediction of the surrounding crowd motion.

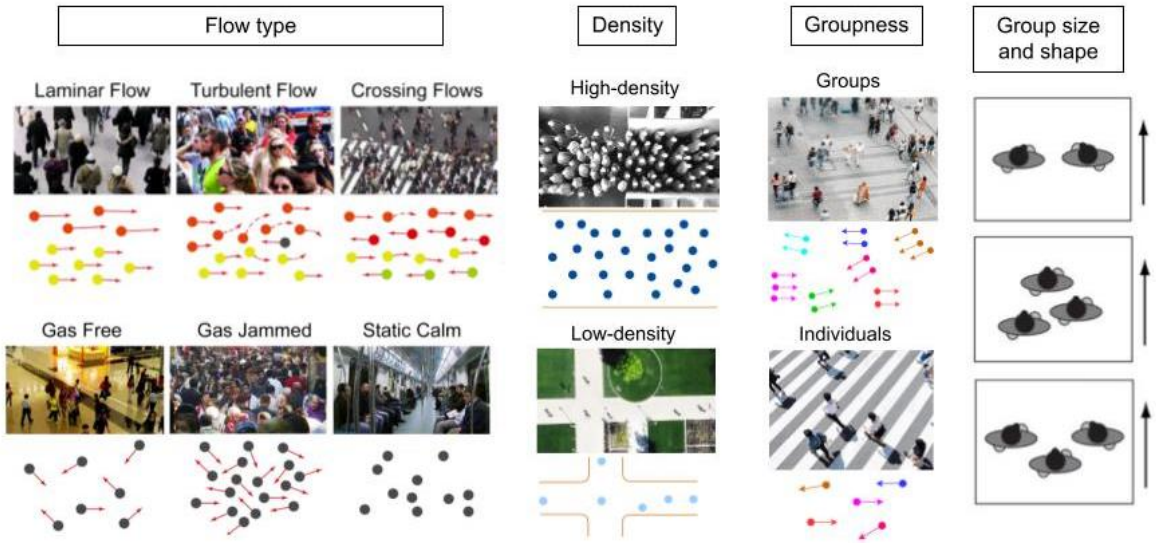

Figure 15. Different Crowd Properties and Patterns.

2.2.4 Problem Formulation

Suppose that a robot navigates in an environment shared by n pedestrians. Each pedestrian state is described through its position $\mathbf{x}_i \in \mathbb{R}^2$ and instantaneous velocity $\mathbf{v}_i \in \mathbb{R}^2$. The robot uses its sensors to get raw measurements and passes them to a human detection unit that returns a set of m detections as $\mathbf{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_m\}$ (see **Figure *16*** which are handled as noisy observations of $\{\mathbf{x}_1, ..., \mathbf{x}_m\}$.

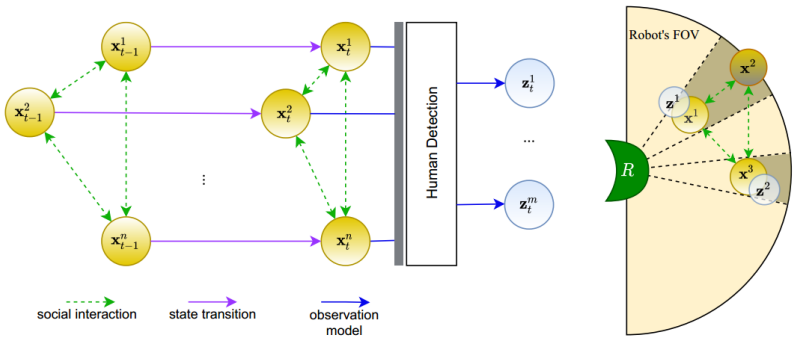

Figure 16. The graphical representation for Occlusion- and Socially-aware Object Tracking. The yellow circles show the state of each agent at two consecutive time instants t-1 and t. The blue circles are the observations at the current instant t. The dashed green arrows represent the social interaction between the agents and the blue arrows represent the observation model.

We assume there are $u = n - m$ unobserved pedestrians, either due to errors by the detector or because the pedestrians are not visible by the sensors, e.g. because of an occlusion or because they are out of the robot's field of view.

Our algorithm leverages the information about geometric relations between the people in the surrounding crowd, extracted from previously observed trajectories, to infer a probabilistic occupancy map covering occluded regions and to impute the position of unobserved pedestrians. This way, we can improve the robot's knowledge about the environment and build more reliable motion plans, estimate and plan beyond what it can see. Before detailing our imputation algorithm, we first introduce the concept of Social Tie.

2.2.5 Social Ties

As a modeling tool to understand the geometrical structure of the flows within crowds, we introduce the notion of Social Tie. Inspired by ideas from other works from Alahi et al., we define a Social Tie as a displacement vector between a pair of agents, expressed in the local frame of the first one:

$$\delta(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{R}_i)^\top (\mathbf{x}_j - \mathbf{x}_i)$$

where $\mathbf{R}_i^t$ is the rotation matrix giving the global orientation angle of the agent i at time t.

2.2.5.1 Strong and Absent Ties

We further classify the social ties as **strong ties** and **absent ties**. The terms are borrowed from social networks literature to represent, respectively, long-term interactions (e.g., grouping or leader-follower behaviors) versus instant interactions (e.g., collision avoidance overtaking) between pairs of agents.

The motivation behind this modeling choice is that these two categories (strong/absent) define two distributions of displacements δ that are better treated dissimilarly. The classification between the two tie types is based on the history of the distance between two agents. We keep the history of the distance vector between all pairs of detected agents. A tie is labeled as **strong**, if:

1 During the last t_c time-steps, the two agents are closer than a threshold distance r_{max},
2 in the same time interval, the Euclidean distance between the agents remains fixed (up to a tolerance threshold ϵ_l on distance variations) over the last second, and
3 the agents have similar orientations (within another threshold ϵ_θ).

If (1) holds but either (2) or (3) does not, then the link is categorized as **absent**.

While we found these simple rules to be enough in our setup, more advanced classification rules can also be used. An example is depicted in **Figure *17*** to illustrate the classification process.

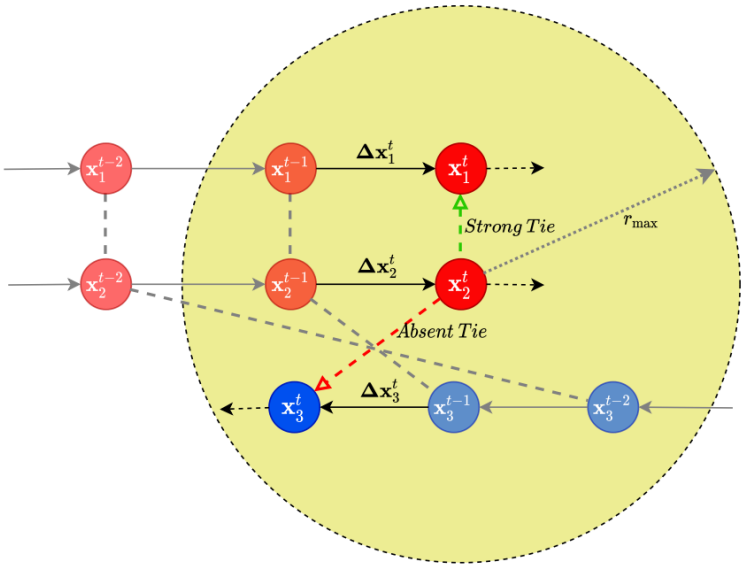

Figure 17. Social Ties: Social Ties. The tie between $\mathbf{x}_1$ and $\mathbf{x}_2$ is classified as a strong tie at time t, (shown in green). On the other hand, the link between $\mathbf{x}_2$ and $\mathbf{x}_3$ is absent (red dashed line).

The above definition implies that a tie (strong or absent) is assigned to any pair of agents that have a distance lower than r_{max} during the last t_c time-steps.

Next, we evaluate the distribution of social ties across agents by classifying and accumulating the observed ties and then taking a polar histogram for each. The two histograms represent the empirical distribution of the strong and absent ties. They are denoted by $p(\delta|\tau = S, \mathcal{H})$ and $p(\delta|\tau = A, \mathcal{H})$, respectively, where τ is the tie type (strong or absent) and $\mathcal{H}$ denotes the historical observations used for training.

Note that these distributions on displacements give us access to the conditional distribution of the absolute location of a queried position $\mathbf{x}$, conditioned on seeing a pedestrian at $\mathbf{x}_i$, and on the type of social tie τ: $p(\mathbf{x}|\mathbf{x}_i, \tau, \mathcal{H})$.

Some examples of $p(\delta|\tau, \mathcal{H})$ are shown in Figure 18, where one can observe that the different collective patterns lead to significantly different social tie patterns in the HERMES-Bottleneck, SDD, ETH and Zara Datasets. For example, in Zara (upper right), the strong ties are mostly observed as side-by-side configurations, while in HERMES-Bottleneck (bottom right), the strong ties come mostly as lines of people (because of the nature of the dataset). We will discuss in the following how these patterns can help us in estimating the location of people in unseen (occluded) areas.

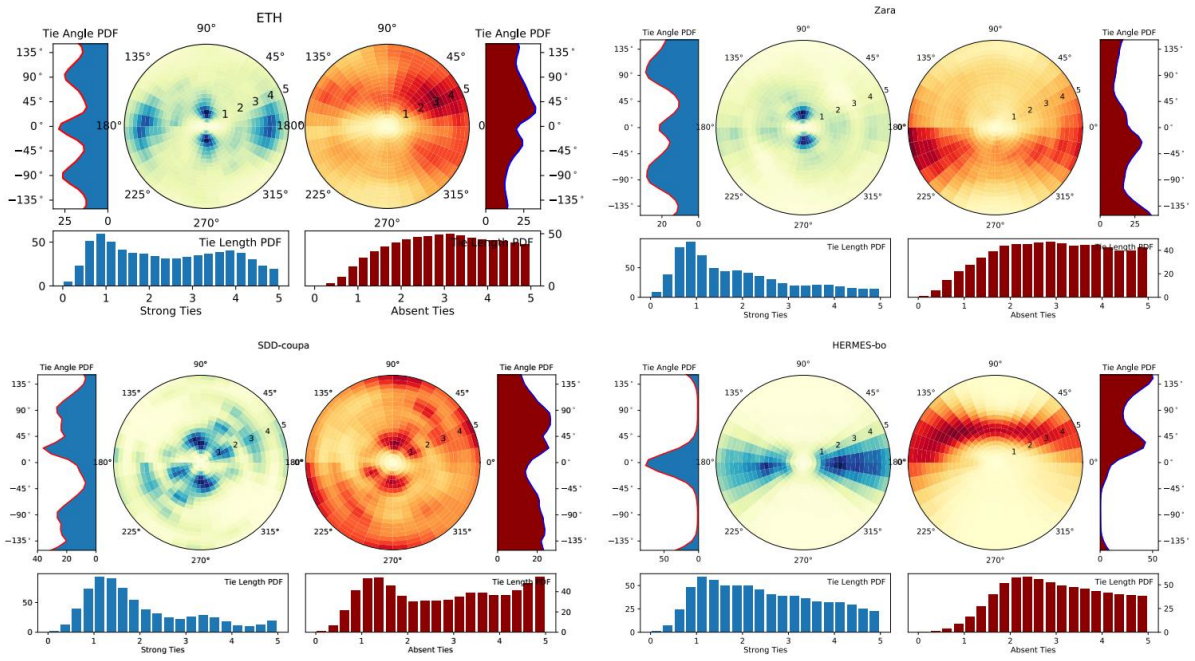

Figure 18. Distributions $p(\delta|\tau = S, \mathcal{H})$ and $p(\delta|\tau = A, \mathcal{H})$ of strong (left side) and absent (right side) social ties for 4 different datasets. The different flow structures lead to very distinct distributions along the datasets. The graphs emphasize some structural elements of the crowd flow, such as the community's width (very small in the Hermes-Bottleneck case), the permanent asymmetries of the absent ties (Zara or Hermes-Bottleneck).

2.2.5.2 Communities (groups)

We define *communities* (or groups) as subsets of pedestrians that are connected (directly or indirectly) by strong ties. They can also be seen as clusters of people moving together as a group, or as a continuous flow of people moving in one direction (see examples in Figure 19).

The m observed agents are partitioned into K communities: $\mathbf{C} = \{\mathbf{c}_1, ..., \mathbf{c}_K\}$. Note that a community can be as small as containing only one person.

The *velocity* of a community $\mathbf{c}_k$ is defined as the average velocity of its members.

We also define a *territory* area for each community, by running a k-nearest neighbor classifier that assigns, to every location in the plane, a label that represents the nearest community. The probability of being, at a location $\mathbf{x}$, in the territory of $\mathbf{c}_k$ is denoted by $\phi_k(\mathbf{x})$. To take the orientation of each agent into account, the k-nearest neighbor is not implemented with the Euclidean distance but with a Mahalanobis distance $d(\mathbf{x}, \mathbf{y})$ with a 2×2 covariance matrix Σ chosen with its first eigenvector aligned with the agent velocity $\mathbf{v}_i$ and assigned to an eigenvalue $\alpha\|\mathbf{v}_i\| + \beta$ while the second one is assigned an eigenvalue β:

$$d(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})$$

The coefficients α and β are set to 0.2 and 0.1 respectively.

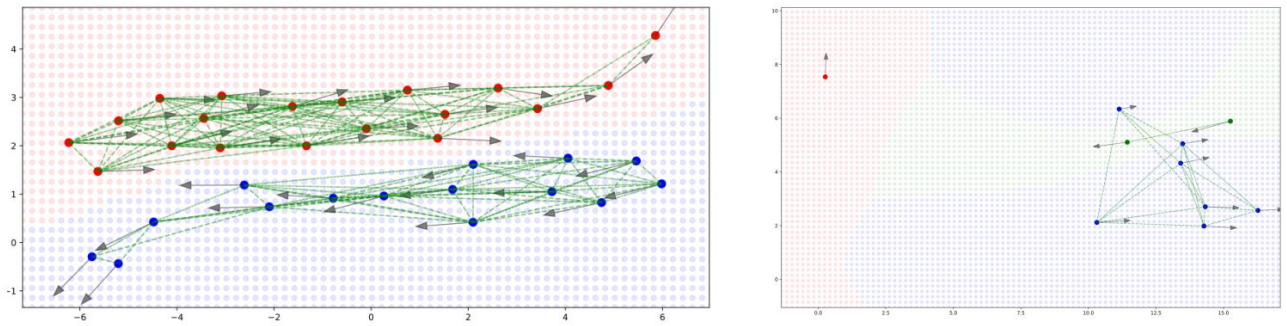

Figure 19. Communities. Green lines: strong ties, black arrows: velocity vector of the agents. Left: Bottleneck dataset, a bi-directional flow of pedestrians that form two communities. Right: A frame of UCY dataset (Zara) with 3 communities moving in different directions. The territory of each community is shown with different colors on a mesh.

2.2.5.3 Imputing New Pedestrians

We explained above that the social ties are extracted from crowd activity observations. Here we leverage this data to predict plausible positions for the unobserved agents that might exist in the blind spot areas and beyond the field of view of the sensor. This idea is inspired by *inpainting* technique in Computer Graphics, where Pair Correlation Functions (PCF) are used to detect textures from an image and propagate them to other areas.

A PCF measures the probability density of the distance between pairs of particles. However, in our context, considering only the distance between agents is difficult, since the orientation of the agents is critical in modeling human collective activities. Hence, we propose an extension of the concept of PCF with the distribution of social ties.

We are interested in finding $p(o(\mathbf{x})|\mathbf{Z})$, the probability of the presence of an unobserved agent at a query point $\mathbf{x} \in \mathbb{R}^2$, given the m detected agents $\mathbf{Z}$. We approximate this distribution using the computed territories and strong and absent ties distributions, which are previously extracted from historical observations:

$$p(o|\mathbf{Z}) = \sum_{w\in\{0,1\}} \sum_{k} p(o, \mathbf{c} = \mathbf{c}_k, v = w|\mathbf{Z})$$
$$= p(v = 0) \sum_{k} p(\mathbf{c} = \mathbf{c}_k) p(o|v = 0, \mathbf{c} = \mathbf{c}_k, \mathbf{Z})$$

The details of the equation can be found in the paper [2].

By interpreting this distribution as a likelihood function $q(\mathbf{x}) = p(o(\mathbf{x}) = 1|\mathbf{Z})$ over $\mathbf{x}$ and by discretizing the $\mathbf{x}$ along a regular grid, we can sample a new agent at location $\mathbf{x}_s$.

We tested different sampling strategies, and found out the following sampling to be more effective.

You can see the pseudo-code of the proposed algorithm, below in **Table *1***.

- We draw a sample from $q(\mathbf{x})$ and then we search within a small disk r_s for the local maximum of q in this region.
- After this, a virtual agent is created at this location $\mathbf{x}_s$, and assigned to community $\mathbf{c}_k p(\mathbf{c}(\mathbf{x}_s) = \mathbf{c}_k)$, with velocity $\mathbf{v}_k$.
- By iterating the sampling, we create more agents in the occluded area. After each sampling iteration #i, we add the sample $\mathbf{x}_s^{(i)}$ to $\mathbf{Z}^{(i-1)}$ (having $\mathbf{Z}^{(0)} \equiv \mathbf{Z}$) and update $q^{(i)}$ using the equation above.
- We repeat the process until $\max_{\mathbf{x}} q^{(i)}(\mathbf{x}) < \epsilon_s$ in a neighborhood of radius r_{nav} around the robot.
- By repeating the sampling process, explained in the previous section, we obtain an augmented set $\mathbf{Z}^+$ of detected and virtual agents.
- We repeat this entire process, for H times to get multiple sets of hypotheses $\mathbf{Z}^{+(h)}$ for $h = 1...H$.

Table 1. Occlusion-Aware Crowd Imputation Algorithm.

Algorithm 1 Occlusion-Aware Crowd Imputation

Inputs: $\mathbf{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_m\}$ ▷ detected persons
Output: Sample sets $\{\mathbf{Z}^{+(h)}\}_h$

1: Compute Ties and Tie types between detections (Eq. 5.1)
2: Find Connected Components (Communities) $\{\mathbf{c}_1..\mathbf{c}_K\}$
3: Compute Territory of each Community $\{\phi_1..\phi_K\}$
4: Compute Velocity of each Community $\{\bar{\mathbf{v}}_1..\bar{\mathbf{v}}_K\}$
5: Initialize: $\forall \mathbf{x}\ q(\mathbf{x}) \leftarrow p(v(\mathbf{x}) = 0)$
6: **for** $\mathbf{x}$ **do**
7: $\quad k' \leftarrow \mathbf{c}(\mathbf{x})$
8: $\quad$ **for** $\mathbf{z}_i \in \mathbf{Z}$ **do**
9: $\quad\quad \delta \leftarrow \mathbf{R}_i^T(\mathbf{x} - \mathbf{z}_i)$
10: $\quad\quad k \leftarrow \mathbf{c}(\mathbf{z}_i)$
11: $\quad\quad$ **if** $k = k'$ **then**
12: $\quad\quad\quad q(o) \leftarrow q(o) * p(\delta|\tau = S, \mathcal{H})$
13: $\quad\quad$ **else**
14: $\quad\quad\quad q(o) \leftarrow q(o) * p(\delta|\tau = A\dot{t}, \mathcal{H})$
15: $\quad\quad$ **end if**
16: $\quad$ **end for**
17: **end for**
18: **for** $h \in 1..H$ **do**
19: $\quad \mathbf{Z}^{+(h)} \leftarrow \mathbf{Z}$
20: $\quad$ **while** $\max q(\mathbf{x}) > \epsilon_s$ **do**
21: $\quad\quad \mathbf{x}_s \leftarrow$ Sample virtual pedestrian
22: $\quad\quad$ add $\mathbf{x}_s$ to $\mathbf{Z}^{+(h)}$
23: $\quad\quad$ assign $\mathbf{x}_s$ to $\mathbf{c}_k = \arg\max_{\mathbf{c}} p(\mathbf{c}(\mathbf{x}_s) = \mathbf{c})$
24: $\quad\quad$ update $q(\mathbf{x})$ for all $\mathbf{x}$ (Lines 6:17)
25: $\quad$ **end while**
26: **end for**
27: Return $\{\mathbf{Z}^{+(h)}\}_h$

2.2.6 Experimental Results

In this section we first study the feasibility/importance of the proposed method on multiple trajectory datasets and then we discuss the results.

2.2.6.1 Implementation Details

We developed our algorithm using Python. For 3D simulation of the crowd motions and the robot perception, we used the CrowdBot Simulator, which is built on top of the Unity game engine and ROS system. Perceptions are obtained from a simulated 360-degree LiDAR with a resolution of 0.5^o and working range of $[0.05m, 8m]$ installed at a height of $40cm$.

We use DR-SPAAM, a deep learning-based person detector that detects persons (legs) in 2D range data sequences. We couple the detector to a Const-Acceleration Kalman-Filter to track multiple targets. In our algorithm, the gridmap has a resolution of 8 cells per meter.

2.2.6.2 Hyper-parameters of the algorithm:

- $r_{max} = 5m, t_c = 1s, \epsilon_l = 0.5m$ and $\epsilon_\theta = 45^o$ for classifying the social ties.
- Radial and angular resolution of the polar histogram used for the representation of strong and absent tie distributions: 25cm ($[0 - 5m]$) and 10^o ($[-180, 180^o]$) respectively, with a constant-padding $p = 1$ for any bin out of this range.
- ϵ_s is set to 50% and r_s (the radius of the search disk) is set to 2m.

2.2.6.3 Real Crowd + Simulated Robot

To evaluate the proposed method and validate the system, it is critical to work with real datasets. However, to the best of our knowledge, there are no public robot-crowd datasets in which the ground truth annotations of occluded pedestrians are available. Hence, we use crowd-only datasets, select one random pedestrian in the crowd and replace it by a simulated robot. The robot traverses the same trajectory taken by the pedestrian. The motivation for not simulating the robot navigation is to make the system independent of the navigation algorithm and to be able to compare the performance against new methods. An example of simulating Bottleneck dataset is shown in **Figure *20***.

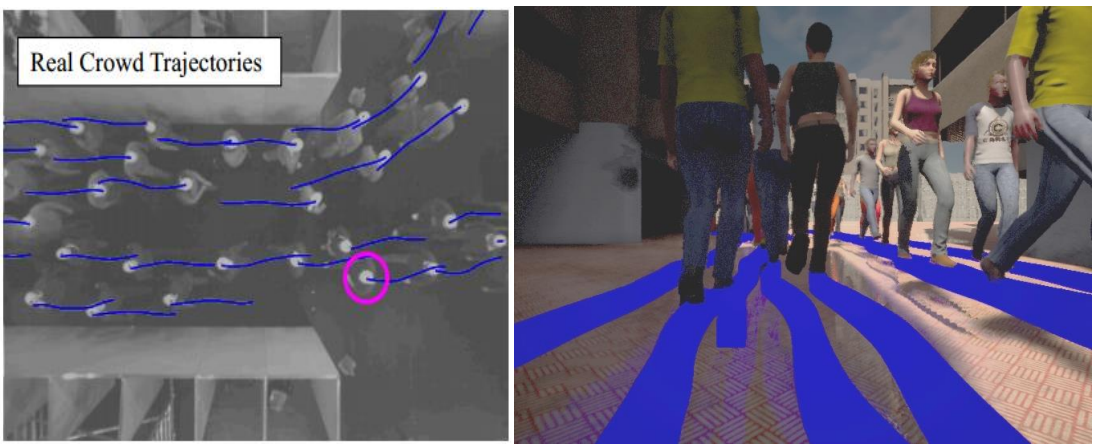

Figure 20. Left: Real crowd trajectories. Right: Simulation from robot's perspective.

2.2.6.4 Testing on HTP datasets

2.2.6.4.1 Occlusion Severity in Crowd

We have measured the severity of occlusions for multiple datasets, using the simulation explained above. This is done by repeating the simulation, each time replacing one pedestrian with the robot. Then we estimate the percentage of sensing rays that are occluded for each pedestrian. We have classified occlusion values into 4 categories: Fully-Visible (0-15%), Partially-Occluded (15-50%), Largely-Occluded (50-85%) and Fully-Occluded (85-100%). The results are shown in Figure 21. As expected, the HERMES sequences exhibit severe occlusions, while ETH/Zara do not. We will see in the next sections that our crowd imputation algorithm does not improve the baseline in low/medium-density situations.

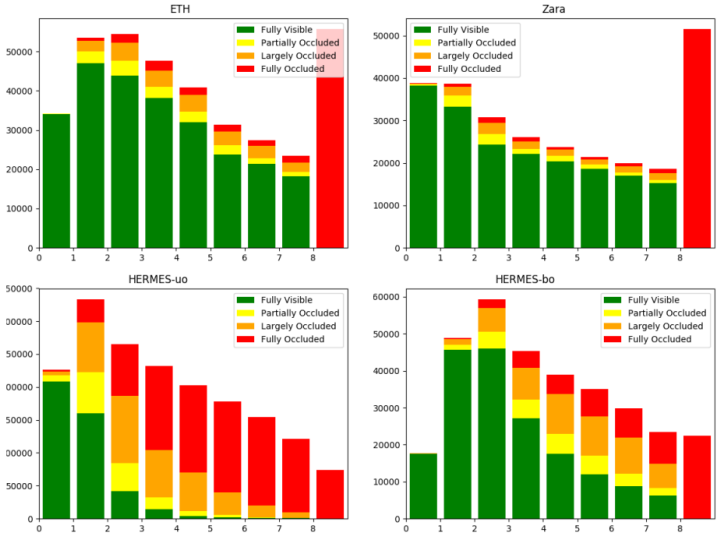

Figure 21. Occlusion Severity: ETH / Zara / Hermes (Uni-directional and Bi-directional flows).

2.2.6.4.2 Analysis of Tie Patterns

In order to measure the amount of information captured by the tie patterns, we calculate the entropy of each pattern for each dataset. The equation of normalized entropy for the distribution is derived by considering different bin sizes in a polar histogram:

$$\bar{H}(p) = -\sum_{r,\theta} p^{r,\theta} \log\left(\frac{p^{r,\theta}}{A^{r,\theta}}\right) / H_{max}$$

where $p^{r,\theta}$ and $A^{r,\theta}$ are the normalized value and the area of the bin (r,θ). The total value is divided by $H_{max} = \log(A^{R,2\pi})$, the maximum entropy of a uniform disk to obtain a normalized entropy between $[0,1]$.

In **Figure *22***, we see the entropy values for different datasets.

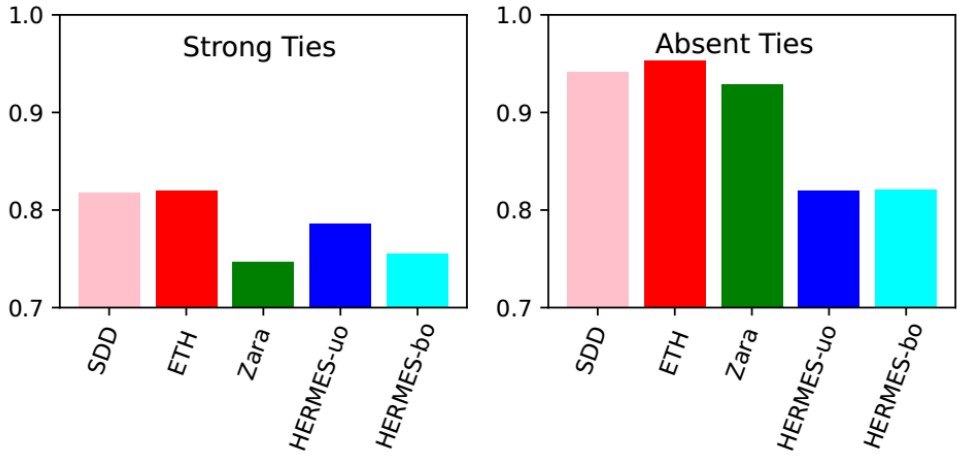

Figure 22. Entropies of Strong/Absent Ties distributions for different datasets. Lower entropy means the distribution contains more structured patterns.

2.2.6.4.3 Performance Evaluation

In order to evaluate the results, we first perform Kernel Density Estimation from the ground truth distribution of agent's location, by using Gaussian kernels centered at the location of each agent with $\sigma = 0.5m$. We denote the result (a probabilistic occupancy map) by π. The Mean Squared Error (MSE) is calculated by averaging the squared difference of predicted and ground truth occupancy grid maps.

We run the simulation, for each of the trajectory #i in the test set, and execute the crowd prediction algorithm to obtain $\hat{\pi}_i^{(h)}$ at each time-step. The prediction errors for Hermes and ETH datasets are shown in **Figure *23***. As you can see the proposed algorithm has improved the Vanilla-MOT baseline on Hermes dataset and also outperforms the PCF baseline. On the other hand, on the ETH dataset the algorithm has not improved the results which means it's better not make no prediction in scenarios with few-occlusions. Due to this issue, we do not report the prediction results on other low-occlusion datasets (SDD and Zara).

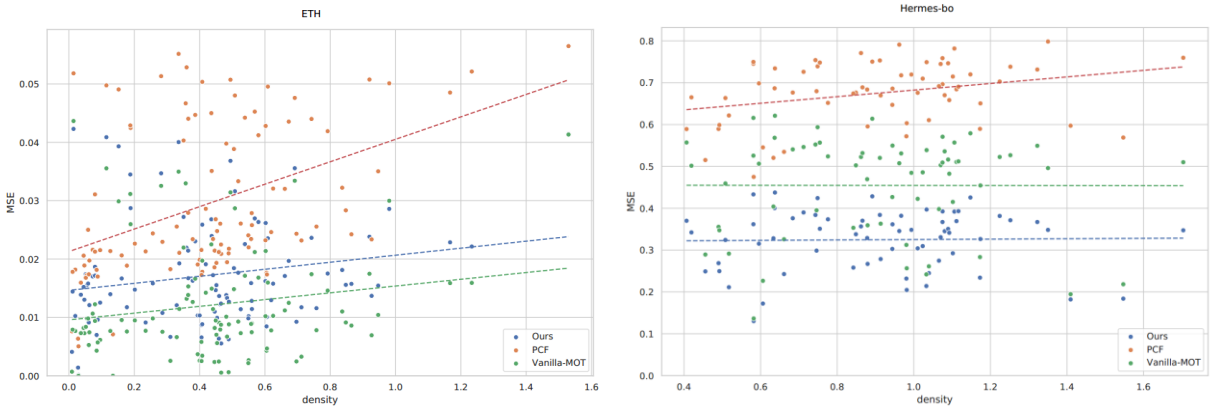

Figure 23. Prediction MSE on Hermes and ETH datasets. Each point on the plot represents the average error of the predictions for one trajectory, sorted by the average crowd density around the robot.

In the figures below some imputation examples are shown for the Hermes dataset. The algorithm has proposed interesting samples in many cases using the social-tie patterns it has learned

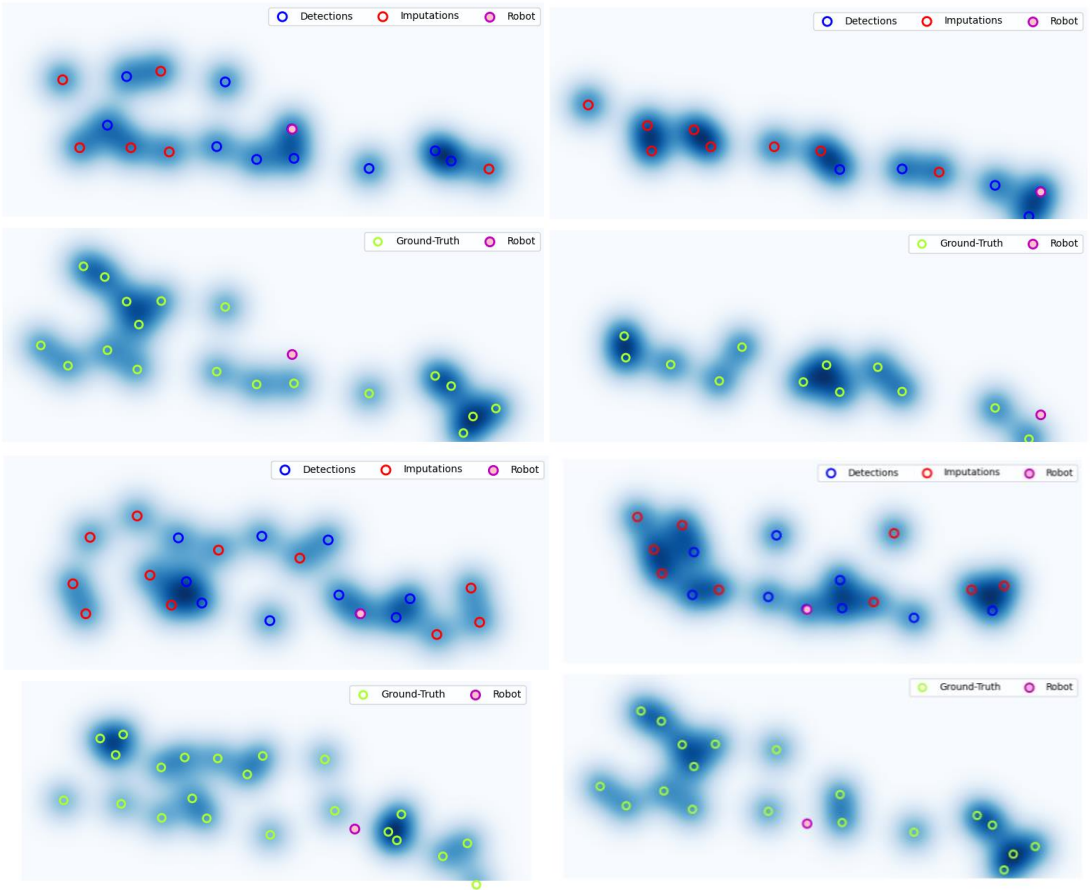

Figure 24. Qualitative Results, on imputing occluded crowd.

3 Crowd-robot simulation

This section describes the Crowd-robot simulator and associated software, and provides updates details for each of the technical components that make up the simulation platform. The simplified architecture of the platform is represented by **Figure *25***. As shown, the general objective of the platform is to create this virtual space where virtual robots (left box) and crowd of virtual humans (right box) can interact so as to evaluate robot navigation capabilities. One originality of the CrowdBot simulator is also to consider the possibility to immerse real humans (bottom box) as well as real robots (upper box) into this virtual space. The diagram also displays the nature of data transiting between each component. The components of the simulation platform are grouped according to their role in the architecture. The main purpose of this section is to familiarize the reader with the solutions chosen to achieve the objectives defined by the project. Therefore, this section first breaks down the objectives for each of the architecture's components. This has been done in the deliverable D42 "Crowd simulator - intermediate version", so the first part is mainly a reminder. Then provides a technical description of the proposed solutions.

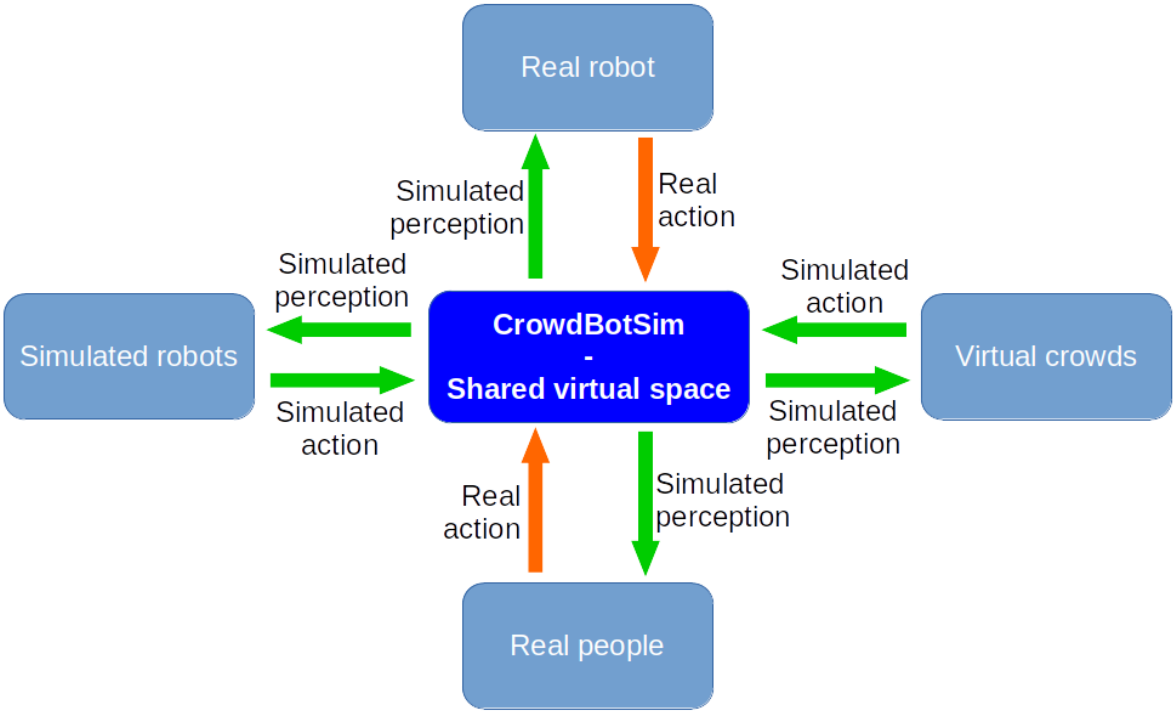

Figure 25. Crowd-Robot simulator diagram.

3.1 Problem statement & objectives

The main objective of the CrowdBot simulator is to provide a test environment for robots intended to navigate densely populated public environments. Test environments can be used at the earliest stage of the development of new robot navigation techniques with several objectives in mind: evaluate the efficiency of navigation techniques, their capabilities to deal with complex environments, study robot behaviors in specific situations, etc. In the frame of CrowdBot, one main objective is to evaluate the level of safety of robot navigation techniques among many people, with possibly high levels of densities.

One goal is to limit the need for real experiments, involving real human participants, because of all the raised ethical and safety issues. In addition to this, the second goal of the CrowdBot simulator is to offer a better coverage of all the possible scenarios in which robots can interact with humans, that would be difficult and expensive to get based on experiments with participants. Finally, another goal of the CrowdBot simulator is to provide detailed and deep analysis of some factors, with possibilities to endlessly replay, reproduce or slightly adjust some parameters of studied situations.

Whilst simulation environments are more and more used in the task of designing robot techniques, especially with the emergence of machine learning approaches that are data-intensive, we are not aware of a simulator that can generate realistic populated environments for robot tests, training and evaluation. In the CrowdBot project, we want to fill this gap.

The CrowdBot simulation toolbox proposes a wide variety of components and features:

- The implementation of robot models used in the frame of the Crowdbot project. We also provide the easy integration of new models.
- A variety of crowd simulation algorithms, that are used to populate and simulate human activity in 3D environments. By combining different models, we can offer a wide range of scenarios, using each algorithm in its range of scenarios where it is most realistic.
- A physics engine allowing in particular to analyze the physical contacts between the crowd and the robots
- An easy-to-use interface and the ability to create scenarios quickly.
- The ability to integrate real robots, or real people, into a virtual scenario by using Virtual Reality to immerse them in the simulation space.

The following sections detail each of these features.

3.2 Overview

With respect to the objectives of the project, and after review of the current state of the art, our solutions are at the interface of mobile robotics, crowd simulation, and virtual reality, as illustrated in **Figure *26***

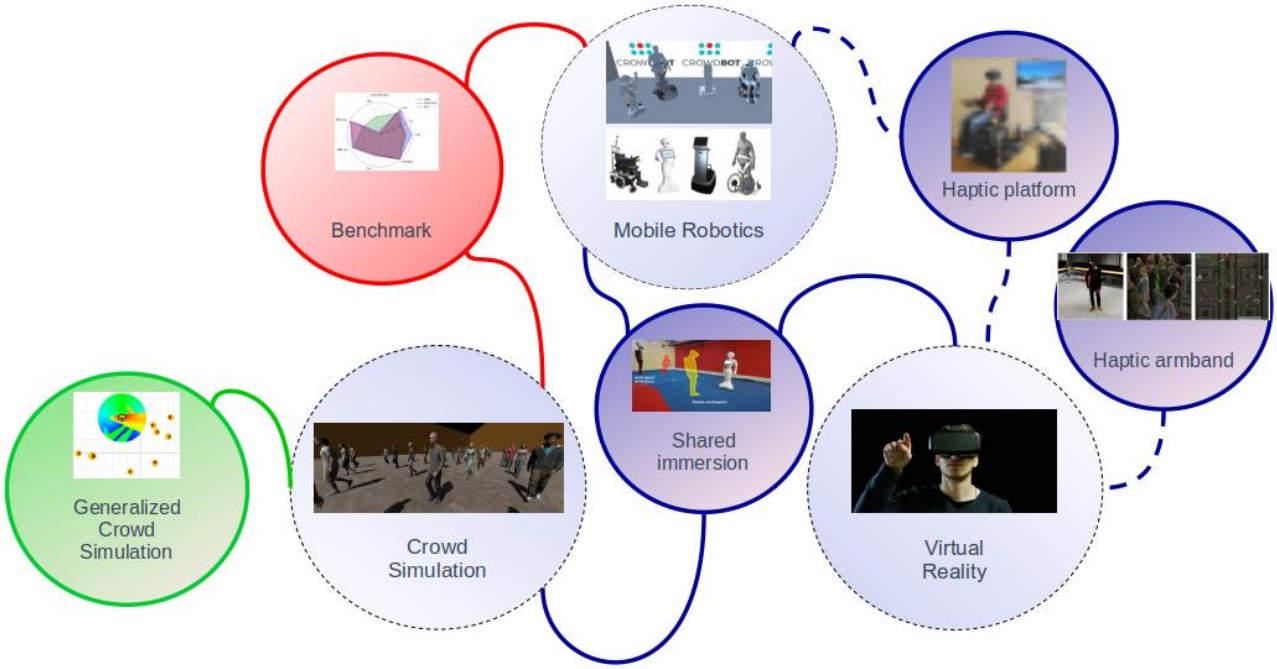

Figure 26. Overview of the crowd-robot simulator solutions.

Our solutions are:

- UMANS: a crowd simulator that reproduces many crowd simulation algorithms using a single principle.
- CrowdBotSim: a crowd-robot simulation software, a "sandbox" for the creation of crowd-robot scenarios.
- CrowdBotChallenge: a benchmark tool for the evaluation of robot navigation in crowds.

The remainder of this section will describe those 3 components.

3.3 UMANS

3.3.1 Overview

UMANS (Unified Microscopic Agent Navigation Simulator) is a crowd-simulation framework that can reproduce many different algorithms for local navigation behavior (such as collision avoidance) in crowds. UMANS translates these algorithms to a single principle, while unifying as many simulation settings and details as possible. This makes it a suitable platform for experimentation and for honest comparisons between algorithms.

To simulate the low-level ('microscopic') behavior of human crowds, a local navigation algorithm computes how a single person ('agent') should move based on its surroundings. Many algorithms for this purpose have been proposed, each using different principles and implementation details that are difficult to compare. *UMANS* describes local agent navigation generically through the optimization of a cost function in a velocity space. We show that many state-of-the-art algorithms can be translated to this framework, by combining a particular cost function with a particular optimization method. As such, we can reproduce many types of local algorithms using a single general principle. UMANS is freely available online. This software enables easy experimentation with different algorithms and parameters.

UMANS is a collaborative work between CrowdBot team and the INRIA Rennes Rainbow team.

3.3.2 What can UMANS do?

- **Run a crowd simulation:** Given a scenario file containing agents and their properties, UMANS can output the trajectories of all agents during a period of time. These trajectories can then be loaded into other applications for visualization or analysis.
- **Basic visualization:** You can use the *UMANS-GUI* application for a simple 2D visualization of a scenario.
- **Model many kinds of behavior:** A scenario file should define the so-called *policies* that agents use for local navigation. Overall, a policy consists of a cost function (possibly with parameters) and an optimization method. You can mix and match these elements as you wish. No programming knowledge is required for this.
- **Support new behavior:** If you are a programmer, you can clone the UMANS repository and *add your own cost functions* (and/or optimization methods), to add new types of behavior to the system. Adding new cost functions is easy thanks to the software's architecture.

3.3.3 What can UMANS *not* do?

- **Global path planning:** The UMANS library deliberately focuses on *local navigation only*, i.e. the local interactions between agents. The simulation environment may contain static obstacles, but the agents in UMANS *do not* plan a global path around these obstacles.
- **Fancy visualization:** UMANS focuses on the simulation itself. For nice 3D animations of a crowd, refer to CrowdBotSim.

3.3.3.1 Core

This subsection describes the core of the framework.

3.3.3.1.1 Unification principle for local avoidance

The idea behind UMANS is to take advantage of the fact that many microscopic (*ie* agent centered) crowd simulators can be described as the combination of a cost function in the velocity space and an optimization method.

Using a single algorithm, we can reproduce many crowd simulators. For instance the **Figure *27***shows the use of this single principle to reproduce 8 different crowd simulation algorithms.

For more information on the theory behind UMANS, please see the I3D 2020 publication "Generalized Microscopic Crowd Simulation using Costs in Velocity Space"[1].

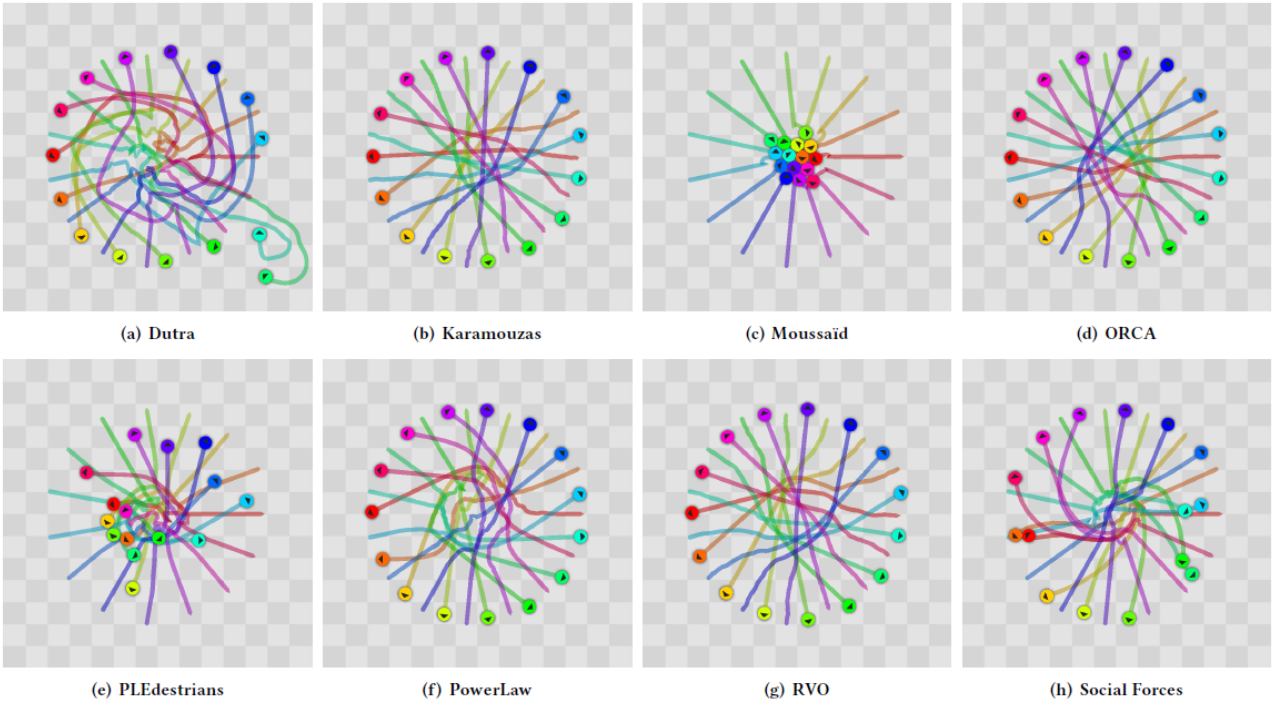

Figure 27. One algorithm reproduces 8 different crowd simulators.

3.3.3.1.2 Grouping

Umans supports grouping by defining a modified goal reaching force toward a leading agent.

Note: This feature is *not* part of the contributions in the I3D publication, thus is not scientifically validated.

3.3.3.1.3 Hybrid simulators

Policies define a set of cost functions, associated to an optimization method, that agents can follow. By construction, UMANS supports multiple cost functions by linear combination. It is used for combining a goal reaching cost function and an avoidance function. This can also be used to define *hybrid* cost functions, which can show various behaviors according to various variables (density, distance to collision, time to collision...).

1 https://project.inria.fr/crowdscience/generalized-microscopic-crowd-simulation-using-costs-in-velocity-space-i3d-2020/

For instance, an agent could follow a smooth vision-based method for long term avoidance where agents are far from each other, and could use velocity based and/or force-based methods for "reflex", last second navigation.

Note: This feature is *not* part of the contributions in the I3D publication, thus is not scientifically validated.

3.3.3.2 User interfaces

There are currently 3 ways to use UMANS.

- Command line: UMANS can be used directly in a terminal. This uses xml files as input describing the scenario (position of agents, goals, properties), and generates csv files which contain the trajectory of each agent.
- Using a graphical user interface: thanks to the framework Qt, we created a user interface that allows quick visualization of trajectories in real time. The **Figure *28*** shows this user interface.
- A library and an application programming interface (API): The core of the framework is compiled into a library which can communicate with other tools through interfacing function (API). This allows us to use UMANS in other tools such as Unity (see CrowdBotSim section).

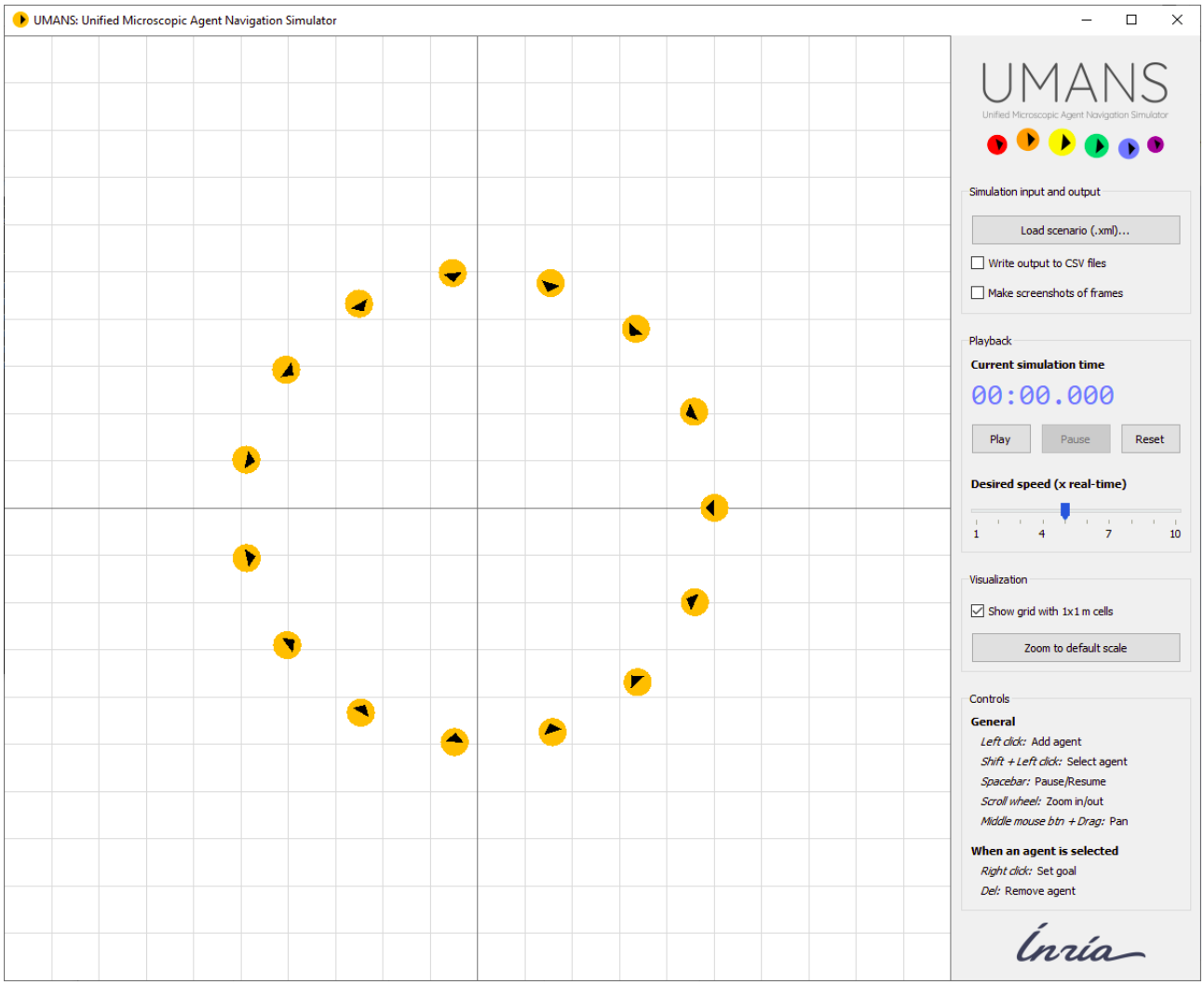

Figure 28. UMANS graphical user interface.

3.3.4 Sources

The Git repository[2] of UMANS is public, so everyone can download the latest version of the source code. We have also published pre-compiled binaries. The latest stable release can be downloaded via the download page.[3]

3.3.5 Wiki and documentation

The UMANS repository comes with a Wiki[4] that helps you install and use the UMANS software. It contains the following pages:

- Getting started[5] - This page explains how to download or compile the UMANS binaries, what their main input and output is, and what to do if you run into problems.
- Configuration files[6] - This page explains the XML files that UMANS takes as input. It describes all options for these input files, so that you can create new scenarios yourself.
- For developers[7] - This page is for those who want to dive into the C++ code itself. For example, it explains how you can write your own cost functions.

Also, most of the source code of UMANS has been carefully documented in a style compatible with Doxygen.[8] This documentation is mostly meant for developers who intend to use/extend the UMANS codebase itself. The Doxygen documentation also facilitates development in an IDE such as Visual Studio. If you run the Doxygen program on the UMANS root folder, it will generate a *html* folder with all documentation pages. (This folder is not part of the repository on purpose.)

3.3.6 License

The software is under the open source MIT license:

[2] https://gitlab.inria.fr/OCSR/UMANS
[3] https://project.inria.fr/crowdscience/download/
[4] https://gitlab.inria.fr/OCSR/UMANS/-/wikis/
[5] https://gitlab.inria.fr/OCSR/UMANS/-/wikis/Getting%20started
[6] https://gitlab.inria.fr/OCSR/UMANS/-/wikis/Configuration%20files
[7] https://gitlab.inria.fr/OCSR/UMANS/-/wikis/For%20developers
[8] http://www.doxygen.nl/

3.4 CrowdBotSim

CrowdBotSim is our solution for crowd-robot simulation: it is designed to provide a test environment for robots intended to navigate densely populated public environments.

CrowdBotSim proposes a wide variety of components and features:

- The implementation of robot models used in the frame of the Crowdbot project. We also provide the easy integration of new models.
- A variety of crowd simulation algorithms, that are used to populate and simulate human activity in 3D environments. By combining different models, we can offer a wide range of scenarios, using each algorithm in its range of scenarios where it is most realistic.
- A physics engine allowing in particular to analyze the physical contacts between the crowd and the robots
- An easy-to-use interface and the ability to create scenarios quickly.
- The ability to integrate real robots, or real people, into a virtual scenario by using Virtual Reality to immerse them in the simulation space. Also, the ability to extend virtual reality tools with haptic rendering armbands.

Most of the description of this tool is done in the deliverable D42 “Crowd simulator intermediate version”. The remainder of this section details the parts that were updated.

3.4.1 Updates

3.4.1.1 Robots' models

CrowdBotSim most recent updates regarding robots focus on improving the overall performances of our robots. In particular, we worked on the stabilization of the physics behavior when controlling a robot.

3.4.1.2 Human models

CrowdBotSim uses a set of human avatar, presented in **Figure *29***, which are then animated, to create a realistic crowd. Those avatars are available online under the project “Rocketbox”. This project is now the property of Microsoft © .

Figure 29: "Rocketbox" open source avatars by microsoft.

This project is open source under a MIT license which allow us to use it freely and open our own CrowdBotSim sources:

DEALINGS IN THE SOFTWARE.

3.4.1.3 Crowd simulation

In the deliverable D42, we described our crowd simulators implemented in the crowd-robot simulator earlier version. While those crowd simulators still exist in the source code of CrowdBotSim, we now use the library UMANS (described in the previous section) which is able to reproduce many crowd simulators and create original behaviors such as grouping, or various avoidance behaviors within one crowd.

3.4.1.4 VR extensions

The deliverable D42 presented our use of virtual reality as a way to create virtual human-robot interactions by having a shared immersion of a human and a robot together in the simulation, as shown in **Figure *30***.

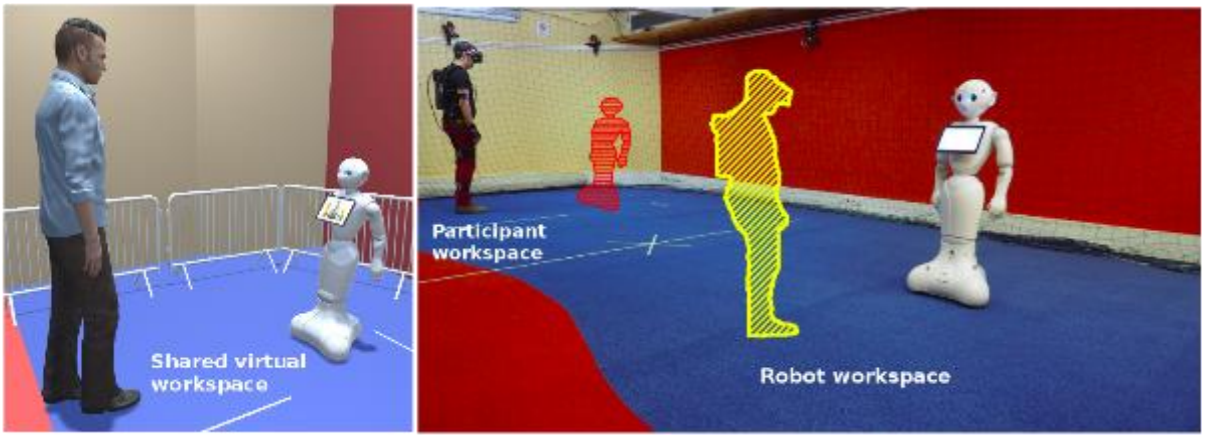

Figure 30: CrowdBotSim is used for virtual reality experiments involving humans and robots.

We extended the immersion of humans in the simulation through the use of haptic rendering devices:

- We performed an experiment, illustrated in **Figure *31***, in which humans had to move through a crowded environment. They were equipped with haptic feedback armband that were activated when a collision between the participant and a virtual agent of the crowd occurred. This collaborative work (CrowdBot/Inria Rennes Rainbow) was accepted at TVCG "Crowd Navigation in VR: exploring haptic rendering of collisions"[9].

Figure 31: Exploring haptic rendering during a task of crowd navigation in virtual reality.

[9] https://ieeexplore.ieee.org/abstract/document/9273221/

- We collaborated with the European project Adapt in the design of, and experimentation with, a mechanical platform that simulates a wheelchair, shown in **Figure *32***. Using virtual reality to immerse the wheelchair driver, the platform moves according to the motion of the virtual wheelchair. This platform is used as a tool to teach people how to drive a power wheelchair safely. This collaborative work (Adapt/CrowdBot) was accepted at ICORR "User-centered design of a multisensory power wheelchair simulator: towards training and rehabilitation applications"[10].

Figure 32: A mechanical simulator of a wheelchair improving user immersion.

3.4.2 Sources and documentation

The sources for CrowdBotSim are public, and are available in two places:

1 CrowdBotUnity[11]
2 CrowdBotSim[12]

CrowdBosUnity contains all the code developed since the project CrowdBot started. CrowdBotSim is a lighter, more stable version of the crowd-robot simulator, for which obsolete code have been removed. In CrowdBotSim, components are separated into git submodules which need to be downloaded to access all the features.

For instance, in CrowdBotUnity, you will find all the versions of the robot 3d models, that evolved during the project, while in CrowdBotSim, you will have to download them separately, and only the latest, stable version, is available (Previous version are available in CrowdBotUnity only).

Each project has its own wiki in the repository where you will find details regarding installation and tutorials on Gitlab[13] and on Github[14].

[10] https://ieeexplore.ieee.org/abstract/document/8779496/
[11] https://gitlab.inria.fr/CrowdBot/CrowdBotUnity
[12] https://github.com/FabienGrzeskowiakInria/CrowdBotSim
[13] https://gitlab.inria.fr/CrowdBot/CrowdBotUnity/-/wikis/home
[14] https://github.com/FabienGrzeskowiakInria/CrowdBotSim/wiki

3.4.3 License

The project CrowdBotSim is under the open source license MIT:

3.5 CrowdBotChallenge

The CrowdBotChallenge is our solution for evaluation of robot navigation in crowds. The idea is to define standard scenarios and performance evaluation metrics, constituting the benchmark tool shown in **Figure *33***, in order to compare various robot navigation techniques fairly. The deliverable D73 "CrowdBot Challenge" describe the problem that this solution solves, as well as its objectives. The deliverable D73 also give our definition of a standard scenario and our evaluation metrics.

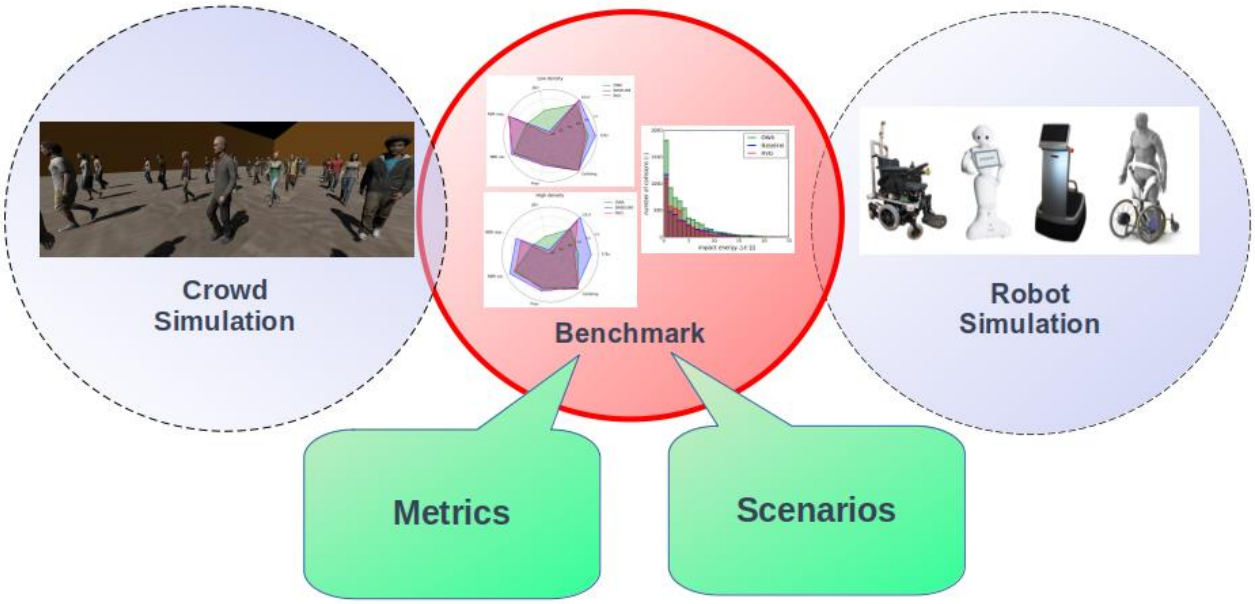

Figure 33: The benchmark, at the junction between crowd simulation and robot simulation, propose a set of metrics and scenarios.

3.5.1 Architecture

The CrowdBotChallenge is a built version of CrowdBotSim. The creation of standard scenarios was done using CrowdBotSim, which handle crowd simulation using UMANS.

For this solution we developed a python module that handle communication between Unity (the executable generated by CrowdBotSim) and the user inputs. It also generated results and visualizations based on our evaluation metrics. Finally, it creates datasets resulting from simulations. This module comes as an interface between Unity and external tools such as ROS of pythonRobotics. Those external tools are used for sensor vizualisation and standard implementations of usual navigation techniques. Those tools communicate natively with the python module, which then handle the Unity simulation accordingly. The **Figure *34*** shows the software architecture of the CrowdBotChallenge.

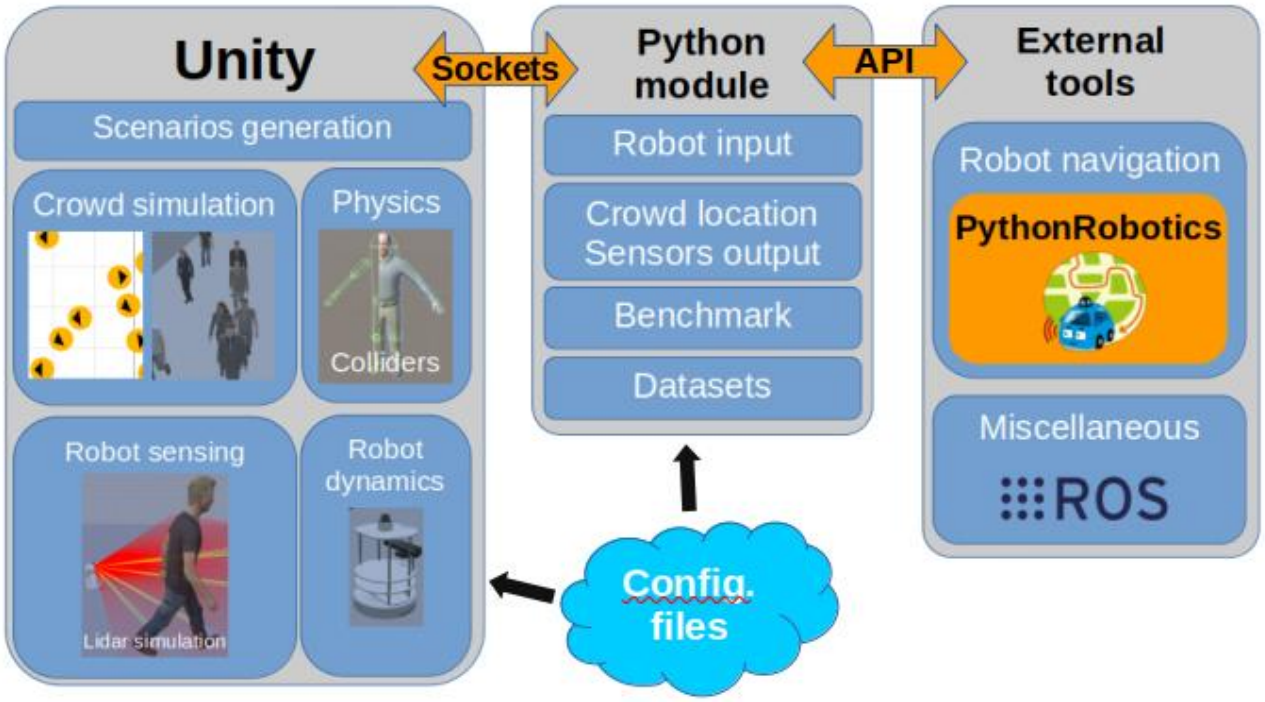

Figure 34: CrowdBotChallenge detailed architecture.

3.5.2 Results

We evaluated 3 different navigation techniques through 160 scenarios:

- A baseline, which consists in having the robot going straight toward the goal without taking the crowd into account.
- Dynamic window approach, which consider the crowd as static obstacles and is less aggressive (it takes into account the robot mechanical properties for smooth trajectories)
- Reciprocal velocity obstacle, which makes a prediction of the crowd trajectories.

The profiles are generated by the python module and are shown in **Figure *35*** and **Figure *36***.

This evaluation has for objective to do a sanity check of the benchmark: it can be used as an evaluation solution for robot navigation in crowd. This work was presented at ICRA "Crowd against the machine: A simulation-based benchmark tool to evaluate and compare robot capabilities to navigate a human crowd"[15].

[15] https://arxiv.org/abs/2104.14177

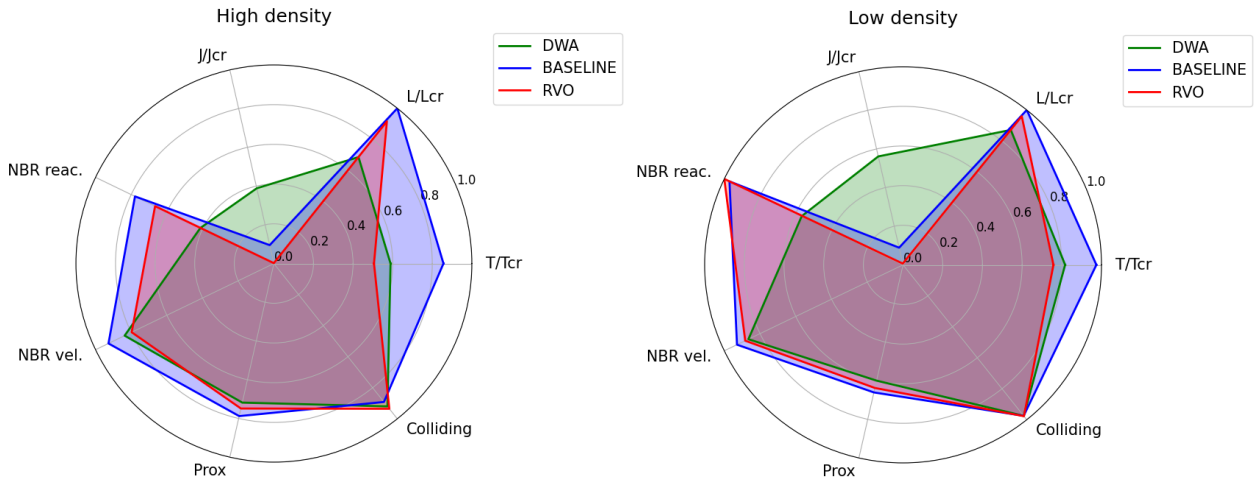

Figure 35: General profiles of our navigation techniques.

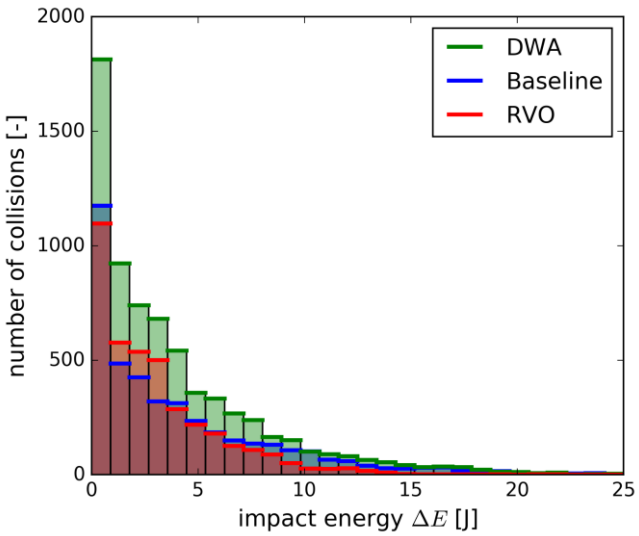

Figure 36: Energetic profiles of our navigation techniques.

3.5.3 Updates

The new version of the challenge has a major change: it's environment.

The environment is new in two ways: first, the robot and the crowd move in a "S" shaped corridor, as shown in the **Figure *37*** and **Figure *38***, so that disappearing agents can disappear without the robot noticing. Second, we define scenarios where the environment can contain static obstacles such as pillars walls and or static agents.

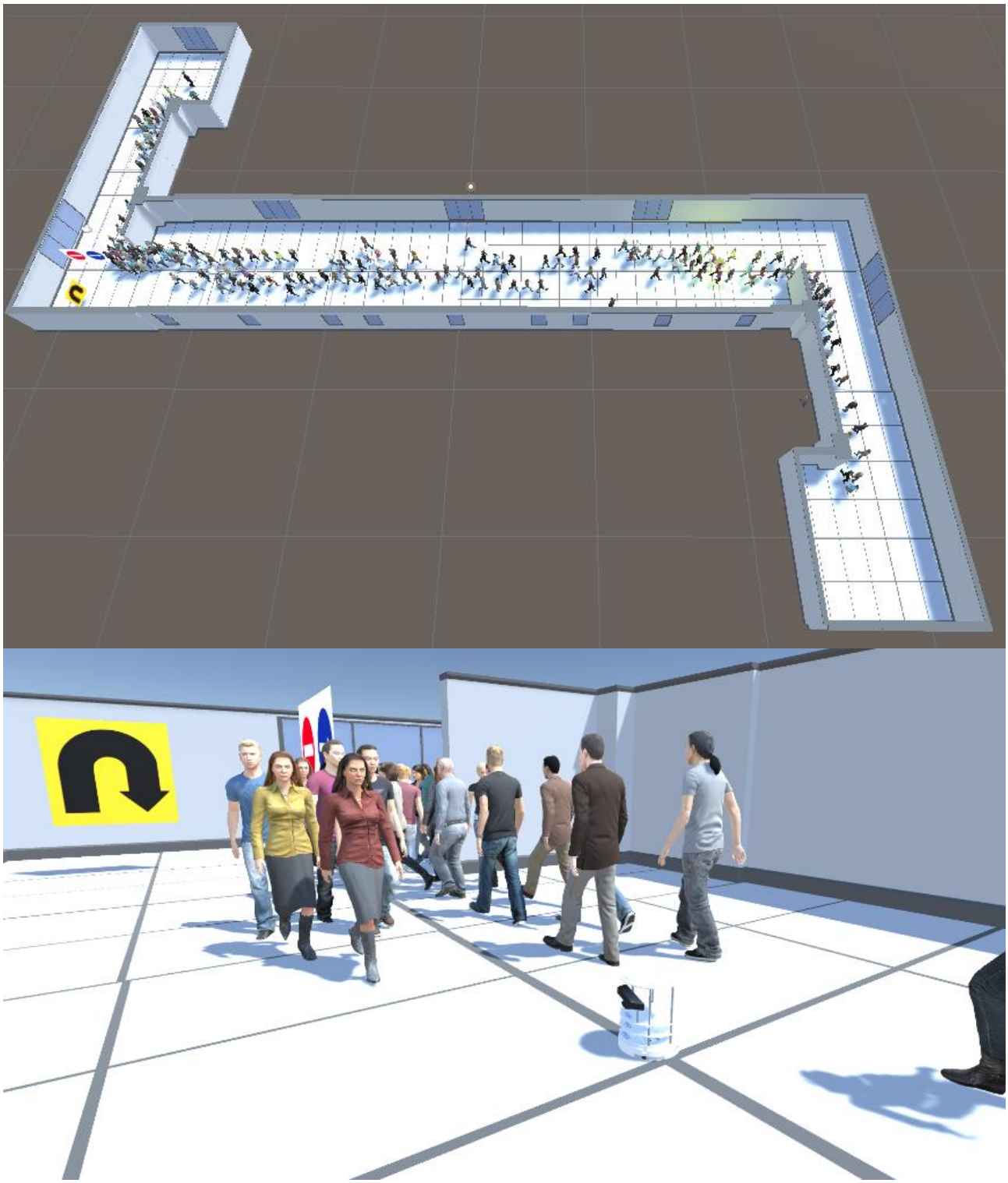

Figure 37: Snapshot of the updated Crowdbot challenge (back view).

Figure 38: Snapshot of the updated CrowdBotChallenge (view from strating point).

3.5.4 Sources & documentation

The CrowdBotChallenge has 2 versions. The first version, used in the IEEE ICRA article, can be found on Gitlab.[16]

It is also reachable through the CrowdBot website.[17]

The second version of the challenge is on GitHub.[18]

Each git repository has a wiki associated so that users can run the challenges.

3.5.5 License

The CrowdBotChallenge is a desktop executable of CrowdBotSim. Therefore, the CrowdBotChallenge is under the scope of CrowdBotSim, thus uses the same license (Chapter 3.4.3).

[16] https://gitlab.inria.fr/CrowdBot/CrowdBotChallenge
[17] http://crowdbot.eu/crowdbot-challenge/
[18] https://github.com/FabienGrzeskowiakInria/CrowdBotChallengePublic

4 Conclusion

In this deliverable, we have described in detail our solutions for 1) situation prediction around a robot based on what it can perceive, and 2) objective evaluation of the robot's ability to navigate in a crowd.

The work presented is in a completed version at the end of the CrowdBot project. In particular, emphasis has been placed on making our tools available in an open manner to benefit the community concerned. The information in this document allows everyone to access the code of our simulators and to continue the development of these tools to make them even more useful to the community.

The CrowdBot project took place during a scientific shift in the themes addressed, linked in particular to the development of deep learning. Whereas the simulation tools required were essentially based on knowledge-driven approaches, they are clearly evolving towards data-driven approaches based on machine learning.

This evolution is noticeable in our work. Given the potential of data-driven approaches to provide accurate predictions at the individual level, we have favoured these approaches. As far as assessment tools are concerned, we felt that the existing more classical approaches were sufficient to provide a representative variety of human behaviour. We focused on: generalising existing methods by unifying them in a single architecture, proposing benchmarking tools by addressing the issue of the coverage of situations evaluated as well as the evaluation metrics. Finally, we addressed the idea of immersive simulation to overcome the limitations of human behaviour simulators, whose real behaviours in all their complexity can never be imitated or predicted.

Our future work is oriented towards a more intensive use of Virtual Reality, in particular with the idea of generating data sets useful for training the artificial intelligence of robots with their own data. For example, constitution of datasets with sensor positioning that is specific to the robot design. Virtual Reality allows the constitution of synthetic data sets with great ease, in an automated way, and possibly with rich annotations.

References

1 Amirian, Javad, Jean-Bernard Hayet, and Julien Pettré. "Social ways: Learning multi-modal distributions of pedestrian trajectories with gans." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019.

2 Amirian, Javad, Jean-Bernard Hayet, and Julien Pettre. "What we see and What we don't see: Imputing Occluded Crowd Structures from Robot Sensing." *arXiv preprint arXiv:2109.08494* (2021).

3 Amirian, Javad, et al. "Opentraj: Assessing prediction complexity in human trajectories datasets." *Proceedings of the Asian Conference on Computer Vision*. 2020.

4 Grzeskowiak, Fabien, et al. "Crowd against the machine: A simulation-based benchmark tool to evaluate and compare robot capabilities to navigate a human crowd." *ICRA 2021-IEEE International Conference on Robotics and Automation*. 2021.

5 van Toll, Wouter, et al. "Generalized microscropic crowd simulation using costs in velocity space." *Symposium on Interactive 3D Graphics and Games*. 2020.

6 Grzeskowiak, Fabien, et al. "Toward virtual reality-based evaluation of robot navigation among people." *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, 2020.