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 7.3: CrowdBot Challenge

Work Package 7 (WP 7)
Exploitation, Dissemination & Communication

Task Lead: INRIA, France
WP Lead: Locomotec, Germany

Table of Contents

Table of Figures

Executive Summary

This deliverable introduces the CrowdBot Challenge and presents its level of advancement before launching the first round. This document first presents the objective of the challenge, which is, in brief, to allow teams working in the field of crowd robot navigation to evaluate their navigation techniques and to compare their methods for moving robots in crowds with those of other teams. To do this, we propose a tool to evaluate these algorithms using different metrics on defined scenarios. Introduction explains the context, the path and the objectives of this challenge. Part 1 describes the scenarios and metrics defined for the Toward CrowdBot Challenge 2020. Part 2 explains the improvements planned for Toward CrowdBot Challenge 2021. Part 3 proposes the dissemination strategy envisaged for this challenge (partners involved, dissemination channels, etc.). Annex 1 is a copy of the wiki of our tool.The whole of this document, together with the download of our tool, should enable interested teams to take up this challenge and make it a reality.

Introduction

One main concept of CrowdBot is to use crowd simulation techniques to provide robots with the capacity of predicting short term evolution of the crowd state around the robot and, consequently, to improve the ability of robots to navigate in crowds. The use of crowd simulation techniques also serves an evaluation purpose, to test in situ the risks raised by a given navigation strategy. As a result, the objective of the work package 4 is two-fold:

1. To simulate the crowd sensed around the robot and to perform (on line and on board the robot) a short-term forward simulation of this crowd to predict its future state. This requires fitting a crowd simulation to the sensed data and to guarantee good simulation performances (accuracy, computation time), which can be deployed rapidly with an accurate enough estimation.
2. To design a crowd simulation software to simulate the navigation of a virtual robot in a virtual crowd for evaluation purposes in terms of collision risks as well as perturbation to the crowd traffic. This requires coupling a crowd simulation technique with a robot simulation technique and to add a physical simulation layer to the simulator to enable it reporting on collision forces.

To meet these two objectives, more than two years of work have already been achieved in the CrowdBot project. We started by studying physical interactions between mobile robots and humans and possible injuries resulting from collisions. The result of this initial work was delivered in July 2018 and is available in the deliverable D4.1[1]. Then, we have extended an existing crowd simulator by the addition of a physical simulation layer during the task T4.2. We are currently working on the tasks T4.3 to couple this simulation to sensed data for navigation purpose and T4.4 to couple this simulation with a robot simulator for evaluation purpose. In October 2019, we have published an intermediate version of the Crowd simulator in the deliverable D4.2[2]. These different works have already allowed us to make several demonstrations (SiDo 2019[3], Project Meeting Lausanne 2020[4], etc.), experiments (INRIA Rennes 2020[5], etc.) and a paper (IEEE VR 2020[6]). INRIA is working with the project partners to improve this crowd simulator.

In order to disseminate this simulator to the scientific community and to allow other teams to test it and use it for their work, we have developed the CrowdBot challenge, whose main tool is the crowdbot benchmark: a set of imposed scenarios to test the robots moving through crowds. Moving robots in dense crowds is new. One reason this topic has not advanced as much as it could have is that it is difficult to evaluate technologies. There

[1] http://crowdbot.eu/deliverable-4-1-physical-interactions-between-robots-humans/

[2] http://crowdbot.eu/deliverable-4-2-crowd-simulator-intermediate-version/

[3] http://crowdbot.eu/demonstration-at-sido-2019/

[4] http://crowdbot.eu/demonstrations-at-lausanne-january-2020/

[5] http://crowdbot.eu/experiments-in-virtual-reality-at-inria-rennes/

[6] http://crowdbot.eu/publication-ieee-vr-2020/

is no crowd easily available in labs to perform tests of the technologies under development. It is not easy to gather crowds to perform user-studies. All these factors slow the progression of research on this topic. A good solution to alleviate these difficulties is to perform tests in simulation environments, but they are not available now. To accelerate dissemination of our results in simulation environments and to motivate research on the topic, we organize a challenge on robot navigation in crowded environments based on the CrowdBot simulation platform. Of course, simulation results cannot directly transfer to real environments, but this will start a virtuous cycle: simulation environments will boost research on navigation in crowds; back, this research activity will always require a more realistic simulation environment.

The main objectives of this challenge are:

- To allow benchmarking,
- To collect feedback,
- To create awareness,
- To initialise collaboration networks.

This document consists of three parts. The first describes the scenarios and metrics defined for the Toward CrowdBot Challenge 2020. The second explains the improvements planned for Toward CrowdBot Challenge 2021+. And the last part develops the dissemination strategy that we currently imagine. Moreover, a copy of the wiki (version April 2020) is available in Annex 1.

1. Toward CrowdBot Challenge 2020

The CrowdBot Challenge consists in providing a set of imposed scenario on simulation tool that allows to test, evaluate and compare different robot navigation techniques adapted to navigate crowds of humans. The challenge is thus presented as a benchmarking tool. The tool is a free and open source software, which gathers different main components:

- A robot simulator, which allows to reproduce "in silico" the behavior of a robot, in particular its perception part and the navigation technique to be evaluated.
- A crowd simulator, which makes it possible to simulate the dynamic world around the robot,
- A set of scenarios against which teams can compete,
- A set of metrics to evaluate the robot's performance in crowd navigation.

Our goal is to organize annual evaluation campaigns in workshops at major conferences in the field. The tool underlying the CrowdBot Challenge is based on two main software bricks that are ROS and Unity, the first one because it is a base of robotics and a tool most widely used for the integration of robotic functionalities, and the second one to facilitate the simulation of a dynamic scene populated by virtual humans. This architecture is shown in Figure 1.

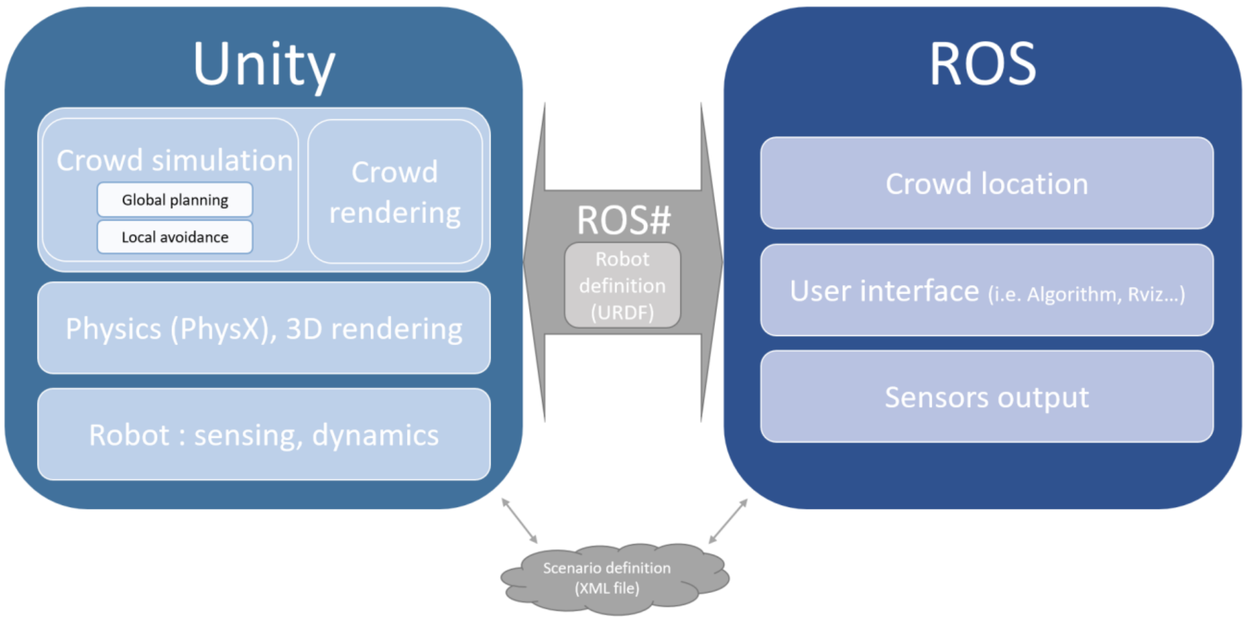

Unity
Crowd simulation
Global planning
Local avoidance
Crowd rendering
Physics (PhysX), 3D rendering
Robot : sensing, dynamics
ROS#
Robot definition (URDF)
ROS
Crowd location
User interface (i.e. Algorithm, Rviz...)
Sensors output
Scenario definition (XML file)

Figure 1. Scheme of the CrowdBot Challenge simulator

1.1. Scenarios for the CrowdBot Challenge

The CrowdBot Challenge objective is to compare the performances of various robot navigation techniques in standardized situations, that we call scenarios. Scenarios are mainly defined by the activity of the simulated crowd around the robot. In the beta version of the challenge (that is to be tested by users before the first actual CrowdBot Challenge takes place, cf. Section 3 Dissemination strategy), we do not try to design a precise list of scenarios, but we demonstrate how to formalize a scenario description through a simple example of a unidirectional crowd flow around the robot.

The activity of crowd movement can be infinitely varied. For simulation, simplification and selection of scenarios is necessary. For this first challenge, we determined four variables: the environment, the crowd dynamic, the local avoidance and the robot. We set the possible parameters for these different variables.

Even with this limitation (Figure 2), 162 combinations are possible for the tests.

	Possible values		Parameters	Values		
Environement	Corridor 50*10 m Obstacles : boundary walls only		Corridor dimensions	[NA]		
			Areas size and location	[NA]		
Crowd dynamics	1D flow		Flow direction	In flow	Counter Flow	
			Density	0.125 p/m^2	0.25 p/m^2	0.5 p/m^2
			Speed (average,std dev)	slow (0.8m/s,0.03)	fast (1.6m/s,0.03)	dispersed (1.2m/s,0.1)
Crowd simulator (local avoidance)	UMANS	TtcaDca	Costfunction coeff	1	20	50
			Range	2m	5m	10m
		RVO	[NA]	[NA]		
Robot	Turtlebot2 + add. Sensors		[NA]	[NA]		

Figure 2. Summary table of possible values

1.1.1. Environment

1.1.1.1. Description

The environment of the simulation is a corridor closed by four walls with different groups of humans. The only obstacles (apart from humans) are the boundary walls. The robot must cross the corridor, from the beginning zone to the ending zone, avoiding the different people/obstacles in it. The following images, extracted from the simulator, show the example of 1 scenario.

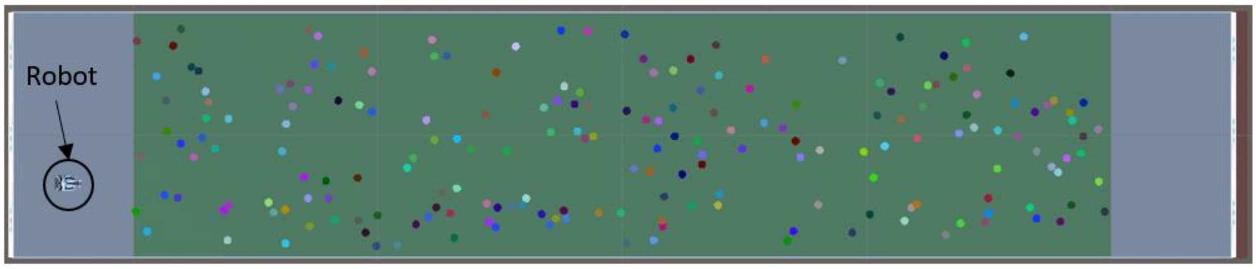


Figure 3. Screenshot of the environment from the software

On Figure 3, in a scenario example with 200 pedestrians, the robot moving with the flow (from left to right). The walls with the CrowdBot logo are a portal: an agent reaching one appears through the other. Some steps of this simulation are available on Figure 4, Figure 5, Figure 6 and Figure 7.

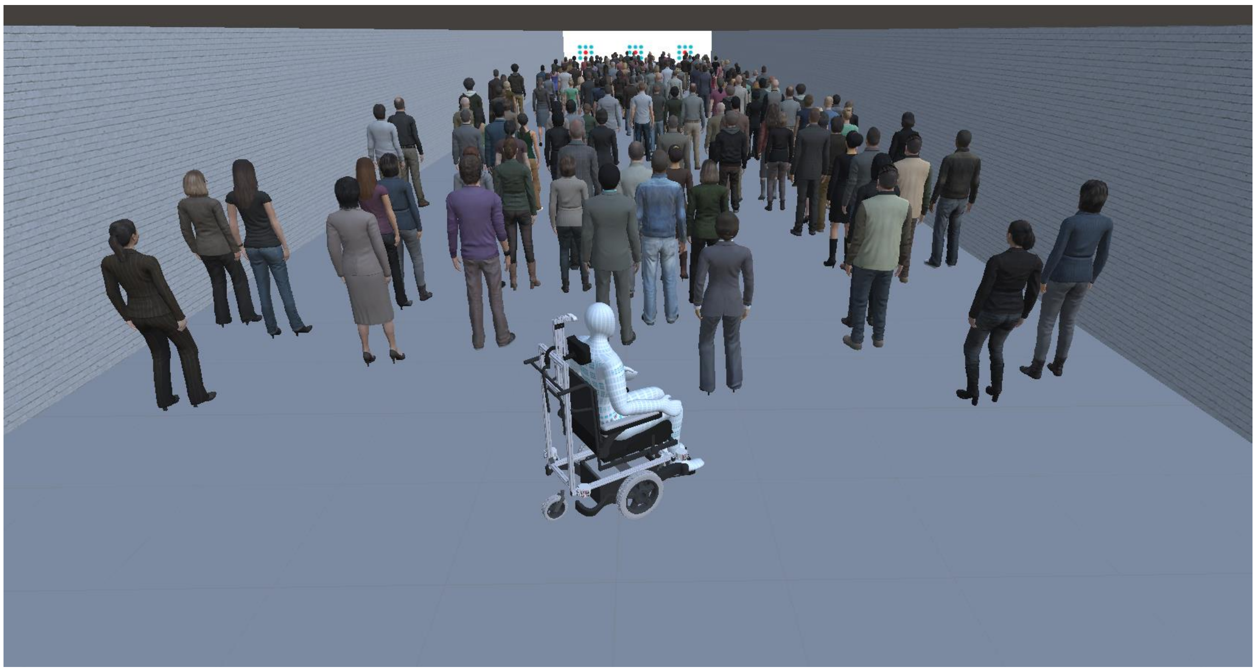


Figure 4. First step of the simulation: Robot spawn

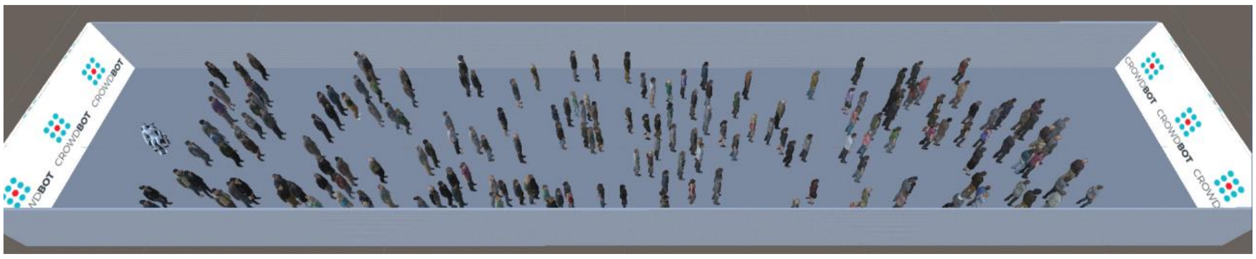


Figure 5. First step of the simulation: top view

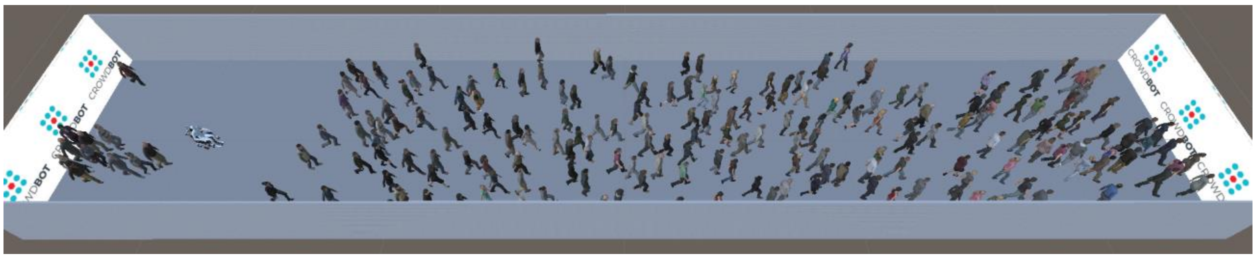


Figure 6. Intermediate step

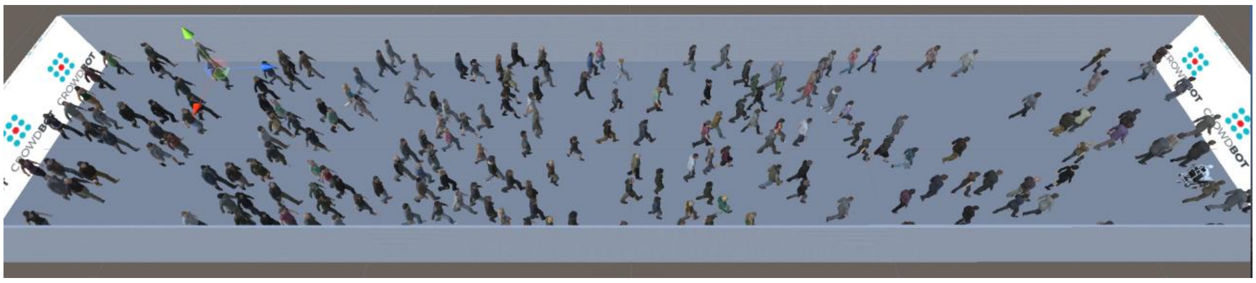


Figure 7. Final step: the robot reached its goal

1.1.1.2. Parameters

The two main parameters adjustable in the environment are the **corridor dimensions** and the **areas size and location**. For this first challenge these two parameters are set.

For the scenario determined in the challenge, the corridor is **50 meters long** and **10 meters wide**. Between 0 and 5 meters, it's the robot spawn zone. Then, between 5 and 45 meters it's the crowd sprawn. And finally, between 45 and 50 meters, it's the robot goal zone. Just before the robot spawn zone, it's the crowd "portal A" and just after the robot goal zone it's the crowd "portal B". Depending on crowd flow direction, the crowd goal area or "portal out" is the portal A or B. The crowd dematerializes in the "portal out" to rematerialize in the "portal in". In the example on Figure 8, portal A is portal out and portal B is portal in because flow direction is "counter flow". This technique, we call "Toric world"," allows us to have a continuous crowd flow with a constant crowd density in the corridor.

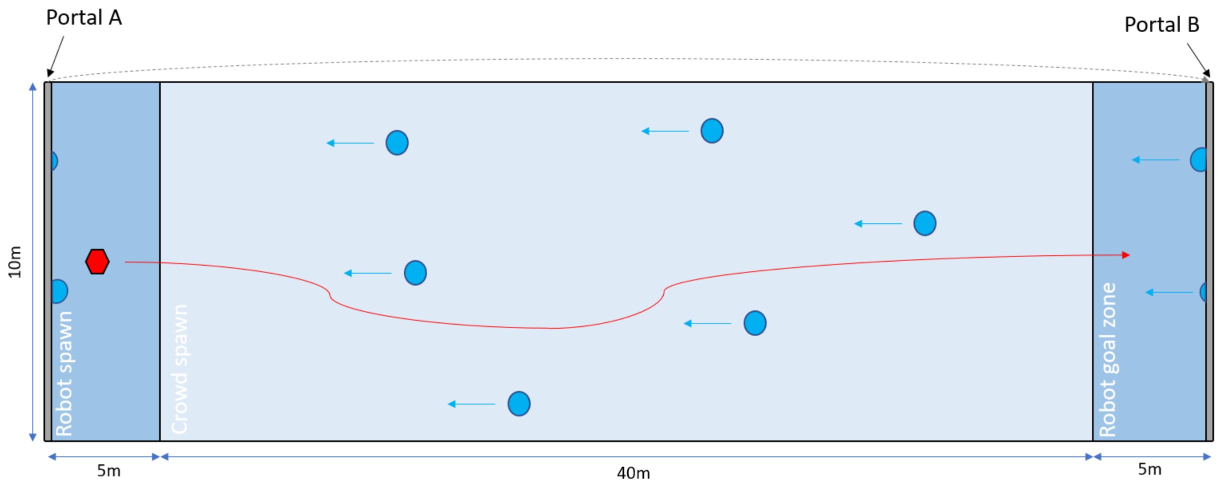


Figure 8. Scheme of the environment (with a "counter flow" crowd)

1.1.2. Crowd dynamics

1.1.2.1. Description

Each agent of the crowd has a goal randomly chosen in the goal area (portal A or B depending on crowd flow direction). Once they reach it, they are teleported in a randomly selected point in the portal out area (portal A or B depending on crowd flow direction), and move toward their goal again. This goal is defined randomly but takes into account the parameter flow direction defined by the user in order for the crowd to have the expected behaviour. The number of agents in the corridor is defined according to the setting of the density parameter while the walking speed of each agent is defined according to the speed parameter.

For each agent in a given scenario, the starting position, the goal, and other randomized parameters such as speed parameters, are generated and fixed for the challenge. Thus, challengers will have the exact same scenarios.

1.1.2.2. Parameters

The crowd dynamic for this first challenge is a **1D flow** only. The four parameters for setting up the crowd dynamics are the **flow direction**, the **density** and the **speed average and standard deviation.**

The two possibilities for flow direction are "**in flow**" (robot and crowd move in the same direction) or "**counter flow**" (robot and crowd move in two opposite directions). If the flow direction is "in flow", the crowd goal area is portal B. Else, if the flow direction is "counter flow", it's the portal A (see Figure 9).

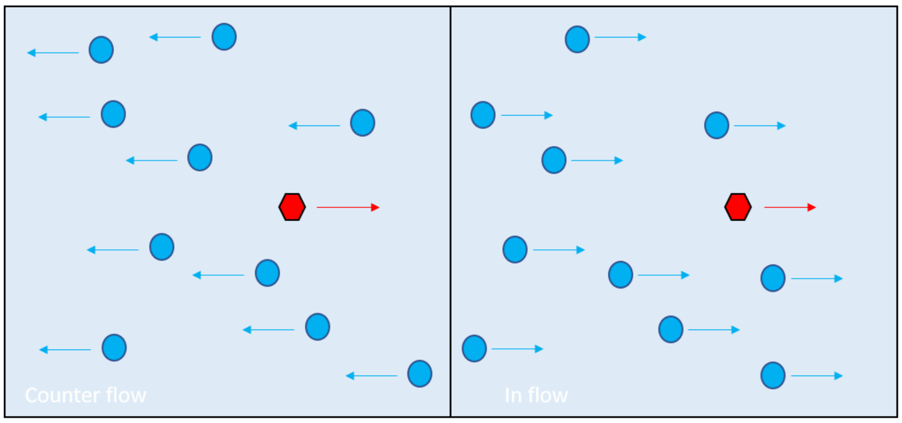


Figure 9. Scheme of flow direction

Density is defined by the number of people per square metre. The possible values are **0.125 p/m²**, **0.25 p/m²** or **0.5 p/m²** (see Figure 10).

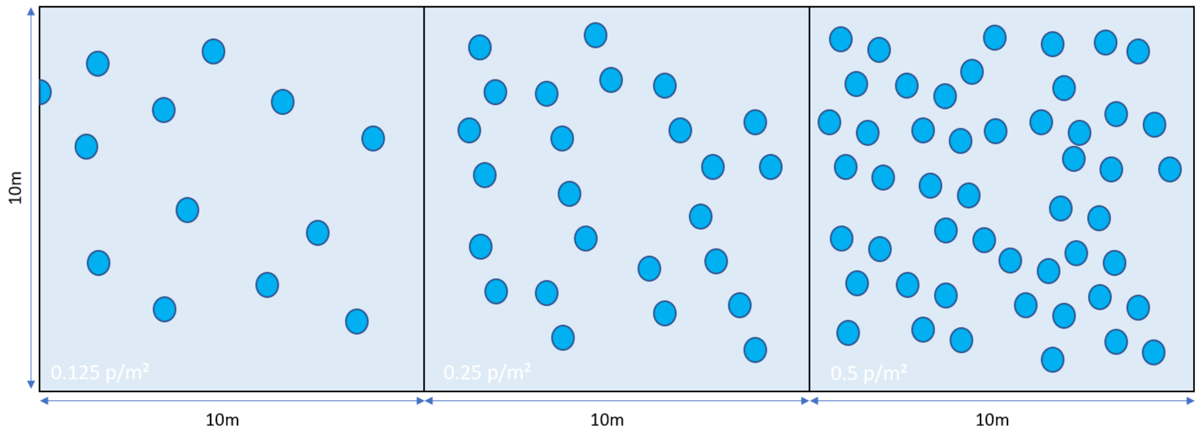


Figure 10. Schemes of the 3 possible density values

Average speed is an average of the speed of all the agents in the simulation. Standard deviation is an evaluation of the dispersion of the measurements around the mean value.

The three possibilities for speed are:

- **Slow**: all the crowd walk slowly with an average speed of 0,8 m/s and a standard deviation of 0,03.
- **Fast:** all the crowd walk fast with an average speed of 1,6 m/s and a standard deviation of 0,03.
- **Dispersed**: each agent of the crowd has a different walk speed with an average speed of 1,2 m/s and a standard deviation of 0,1.

1.1.3. Crowd simulator for local avoidance

1.1.3.1. Description

For the local avoidance, INRIA has developed its own tool UMANS[7] (Unified Microscopic Agent Navigation Simulator, see Figure 10). It’s 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.

[7] https://project.inria.fr/crowdscience/project/ocsr/umans/

UMANS

Unified Microscopic Agent Navigation Simulator

Figure 11. UMANS logo

1.1.3.2. Parameters

Umans allows mixing algorithms togethers, to create an original crowd behavior. In this first challenge, we choose to simulate the crowd with a mix of two crowd simulators available in the UMANS framework: “TtcaDca” and “RVO”.

The first method, TtcaDca, is used for mid-term avoidance and is the one that will have its parameters “Range” and “Coeff” changed in the different scenarios.

The second method, RVO, is used for reflex behavior, and is set to very short term avoidance. It is active when two agents (virtual human or robot) are only a few centimeters away.

The parametrization guarantees absence of collision between the robot and virtual human when the robot is stopped. Thus, collisions occur only when the robot is moving.

The possibilities for the costfunction coeff are 1, 20 and 50. The possibilities for the range are **2 meters**, **5 meters** and **10 meters**.

The parameters set are:

- name="TtcaDca"
- sigmaTtca="0.3"
- sigmaDca="0.3"
- sigmaAngle_goal="3"
- sigmaSpeed_goal="0.01"
- costfunction range="0.3"
- name="RVO"
- time to horizon=“5s”

To understand the meaning of these different parameters, you can read the Umans presentation paper.[8]

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

1.1.4. Robot

1.1.4.1. Description

The robot used in the first challenge is TurtleBot2 (view Figure 14). It's a classic open source robot in the robotics community. It was developed by Willow garage (a.k.a ROS creators). The TurtleBot2 is the 2nd generation of the TurtleBot following within the REP 119 specification[9]. It is a low-cost, personal robot kit with open-source software. With TurtleBot, everyone is able to build a robot that can drive around his house, see in 3D, and have enough horsepower to create exciting applications.[10] It is for this accessibility that we decided to work with this robot for the first challenge. This allows a larger number of teams to have access to it.

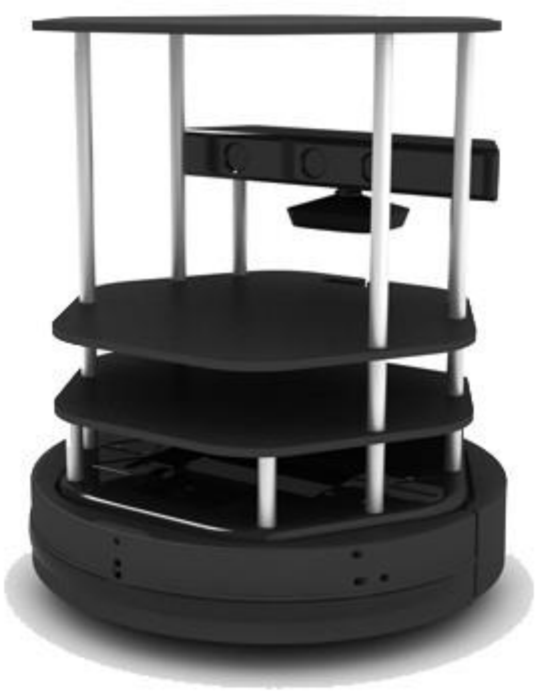


Figure 12. Picture of TurtleBot2

The robot modeling comes (view Figure 12 and Figure 14) from the turtlebot_description[11] package which provides a complete 3D model of the TurtleBot for simulation and visualization.

[9] https://www.ros.org/reps/rep-0119.html

[10] https://www.turtlebot.com/turtlebot2/

[11] http://wiki.ros.org/turtlebot_description

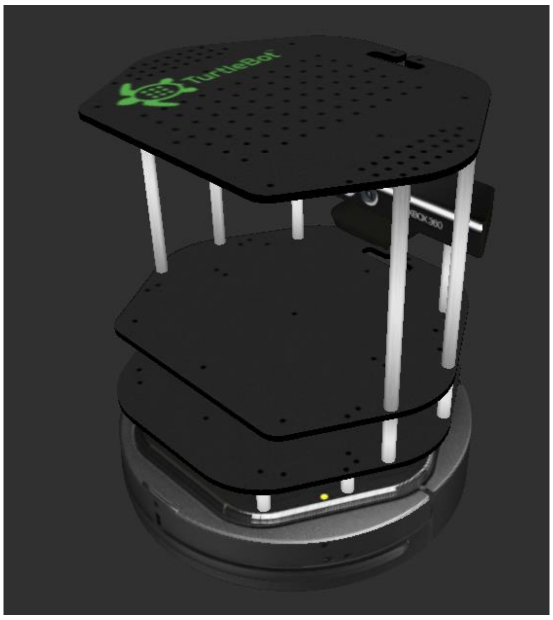


Figure 13. Image of TurtleBot2 in virtual reality

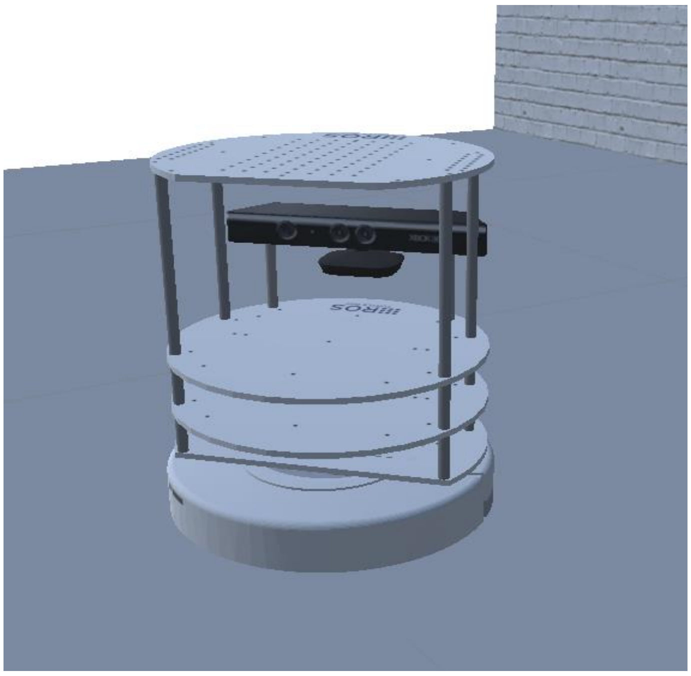


Figure 14. Screenshot of TurtleBot2 from the simulator

1.1.4.2. Parameters

The robot parameters cannot be changed for this first challenge. Some additional sensors are used to perceive datas from the environment:

- Twelve ultrasonic sensors, all around the robot,
- Two Lidars, one front and one back,
- One RDBG camera in front for front detection,
- One odometry system.

1.2. Evaluation Metrics

For the first challenge, we determined 3 categories of evaluation metrics in order to analyze the path in different ways. Indeed, a particular path might be very efficient but have a substantial effect on the crowd.

These 3 categories are Path efficiency, Effect on crowd and Collision, and have been described in the deliverable 1.3: Specification of Scenarios Requirement Update.[12] A copy of these descriptions is available in Parts 3.2.1 to 3.2.6 below.

The aim of these categories is to classify various metrics which will provide the possibility for challengers users to evaluate their algorithm on the proposed scenarios.

Challengers will have to run the challenge with calibration scenario which are used as reference for the metrics: a scenario with the robot alone moving with their algorithm, and scenarios with the crowd alone, with the different densities specified earlier.

For the first published version of the challenge, the metrics implemented will be the following:

1.2.1. Path efficiency

The path efficiency regroups the metrics which evaluate the robot efficiency to reach the goal when it is alone in its environment, compared to its efficiency when it is doing the same task surrounded by a crowd.

- Relative time to goal

The relative time to goal compares the time taken by the robot to reach its goal when it is alone to the time it takes when it is in a crowd. The metrics is given below:

$$T_{rtg} = \frac{T_{robot\ alone}}{T_{crowded}}$$

- Relative path length

The relative path length compares the length of the path taken by robot to reach its goal when a crowd is present or not. Mathematically it is given by the following formula:

$$L_{rp} = \frac{L_{robot\ alone}}{L_{crowded}}$$

- Relative jerk

The relative jerk evaluates the smoothness of the path taken by the robot to reach its goal when it is alone compared to when it is in a crowd. It is given below:

$$J = \int_0^{T_{robot}} (\dot{a} + \ddot{\theta})$$

$$J_{rp} = \frac{J_{robot\ alone}}{J_{crowded}}$$

1.2.2. Effect on crowd

This category regroups the metrics evaluating the effects the robot has on the crowd.

- Effect on time

The effect on time is measured by comparing the average time taken by the agents of the crowd to reach their goal when there are no robots and the average time taken by the crowd when the robot is present. It is given by the following formula:

[12] http://crowdbot.eu/deliverable-1-3-specification-of-scenarios-requirement-update/

$$E_t = \frac{T_{crowd\ alone}}{T_{crowd\ +\ robot}}$$

- Effect on velocity

The effect on velocity compares the average velocity of the crowd when there is a robot on not. The formula is given below:

$$E_v = \frac{V_{crowd\ alone}}{V_{crowd\ +\ robot}}$$

- Neighbor's time to goal

The neighbor's time to goal compares the average time taken by the neighbors of the robot to the one taken by the crowd. The neighbor is the set of agent in the vicinity of the robot (3m circle around the robot). The mathematical formula is given below:

$$N_{ttg} = \frac{T_{robot\ neighbors}}{T_{crowd}}$$

- Neighbor's velocity

The neighbor's velocity compares the average velocity of the neighbors of the robot to the one of the crowds, on average. The mathematical formula is given below:

$$N_v = \frac{V_{robot\ neighbors}}{V_{crowd}}$$

1.2.3. Collision

This category of metrics is regrouping anything collision related.

- Risk

We first define the notion of distance to closest approach (dca), which is, at a given time, the minimal distance in meters between an agent of the crowd and the robot if both keep their current velocity in the future. If this distance is 0, then they will eventually collide if they do not try any avoidance maneuver. The time of closest approach (ttca) is the corresponding time in seconds, at which the collision happens.

The risk is defined as the sum of the minimal ttca if dca is 0 at each time step.

$$R = \sum_0^{T_{robot}} (ttca_{min}\ if\ dca = 0, 0\ otherwise)$$

- Expected Severity

We define the severity in a similar way as the risk, but we take the relative velocity into account:

$$S_R = \sum_0^{T_{robot}} (v_{relative}^2 \times ttca_{min}\ /if\ dca = 0, 0\ otherwise)$$

- Number of collisions

This metric is just the count of collisions, noted C, that occurred in the scenario. Note that a collision is not necessarily dangerous, as a touch is already a collision.

- Collision severity

We define collision severity as the sum of the momentum of each collision that actually occurred during the scenario. It is defined as below:

$$S_C = \sum_0^{T_{robot}} (m_{robot} + m_{agent})/2\ \times (v_{robot} - v_{agent})^2\ \ ||when\ colliding\ (ttca = 0)$$

1.2.4. Running the evaluation

The challenge's main purpose is to generate a report that gives, for each available metrics, a score. These scores can be used as an objective measure of an algorithm. It is the responsibility of the challenger to analyze the results with their interpretation.

The data necessary to generate the report will be automatically recorded when the challenger starts the challenge.

Each completed scenario will generate a CSV file with the scores of the metrics, and a rosbag (record file for ROS) with the raw data of the simulation (sensors, position of robot and agents…).

2. Toward CrowdBot Challenge 2021+

This part explains evolutions for the next challenge version. The extended simulator scheme with its different blocks is available in Figure 15.

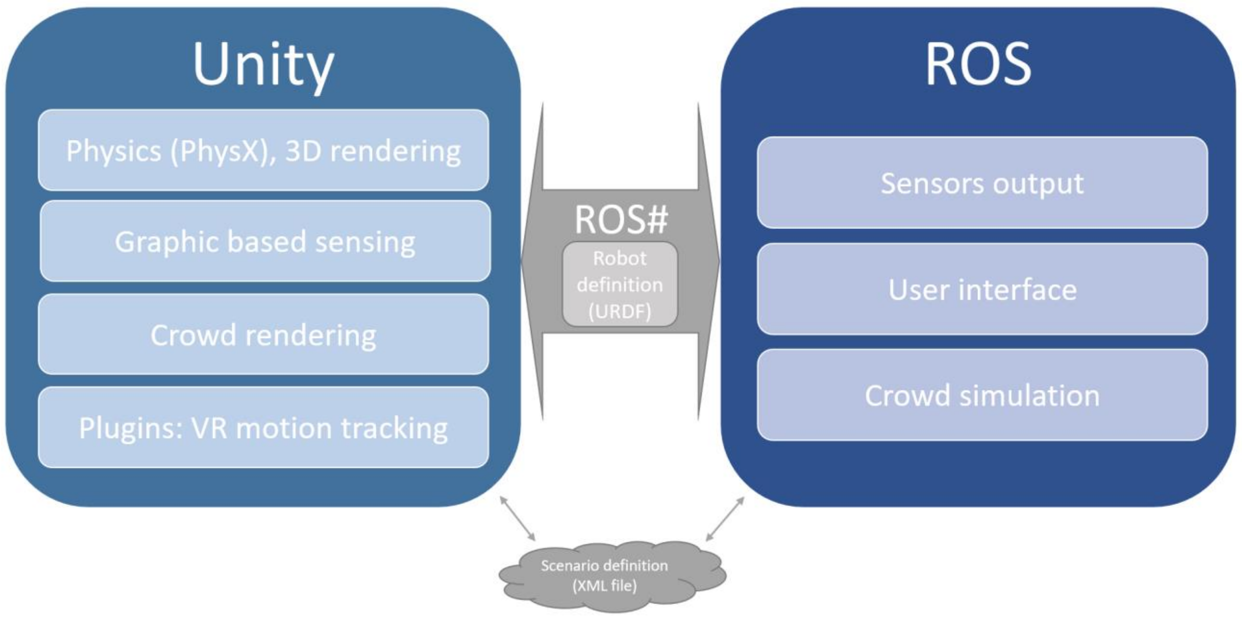


Figure 15. Scheme of extended simulator

2.1. Scenarios

2.1.1. Environment

For the next challenges, different propositions for environments are being considered.

The first one is to use the previous corridor with modification of its dimensions and its areas size and location. The second is to use a crossing corridor. In these different cases the boundary walls will always be present and pillars can eventually be added.

For more reality, the environment could also evolve to a large shed with gates/law walls instead of walls. Or for a classic street where the current walls would be replaced by the buildings on the street.

2.1.2. Crowd dynamics

The objectives for the crowd dynamics is to add the crossing flow and the 2D flow (see Figure 16).

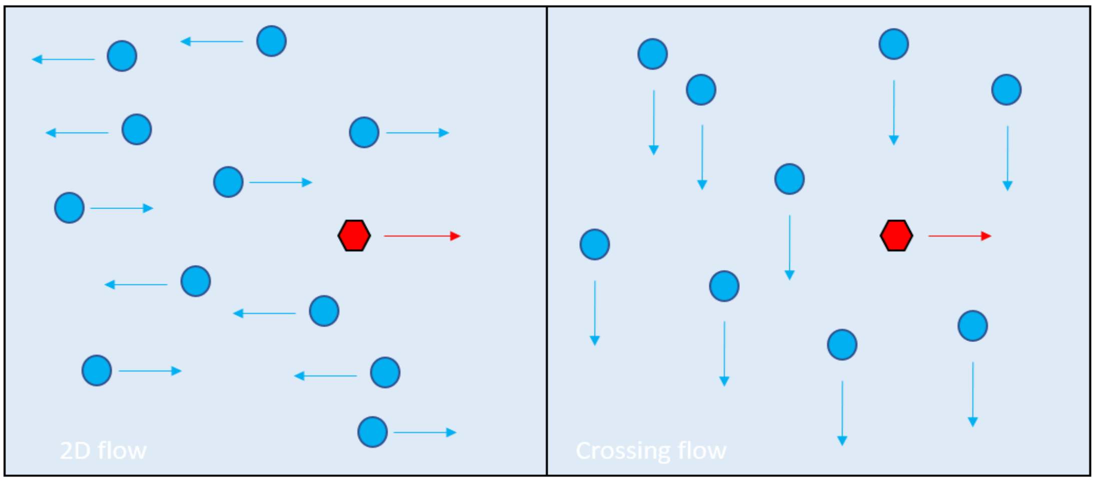


Figure 16. “2D flow” and “crossing flow”

2.1.3. Robot

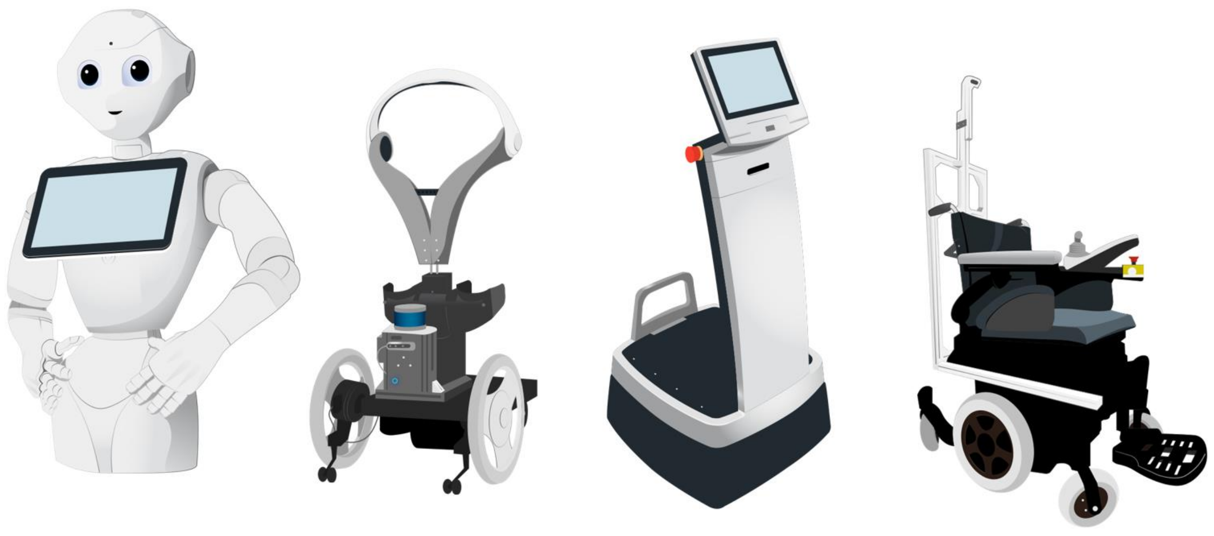


Figure 17. Robots proposed for next challenges (from left to right; Pepper, Qolo, CuyBot and Smart Wheelchair)

For the next challenges, the objective is to add in the simulation the different robots of CrowdBot (see Figure 17 and Figure 18):

- Pepper, a humanoid robot that must navigate in a dense crowd while actively approaching people to assist them,
- Qolo, a device that combines active powered wheels and passive exoskeleton,
- CuyBot, a robot under development which will adapt to compact crowd, being touched and pushed by people,
- Smart Wheelchair, a semi-autonomous wheelchair, that must adapt its trajectory to unexpected movements of people in its vicinity.

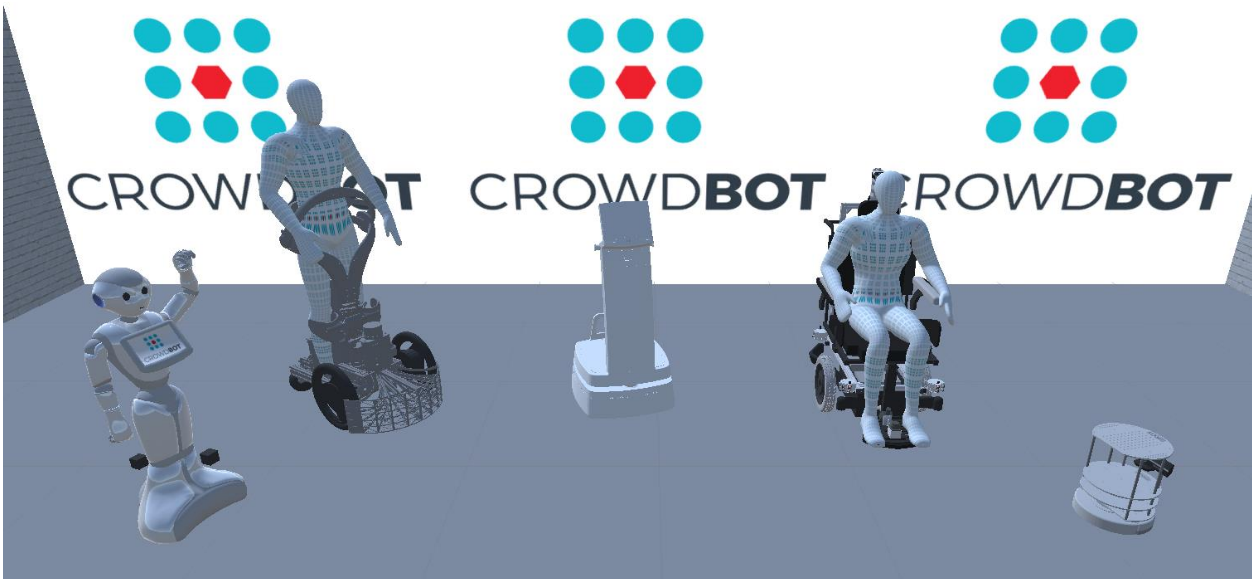


Figure 18. Simulation of the robots proposed for the next challenges

2.2. Evaluation Metrics

For our challenge, we determined 3 additional categories of evaluation metrics in order to analyze the path in different ways. Indeed, additional metrics might refine the analysis of challengers.

These 3 additional categories are Pedestrian-robot similarities, Crowd-robot interaction, and Shared control quality, and have been described in the deliverable 1.3: Specification of Scenarios Requirement Update.[13] A copy of these descriptions is available in Parts 3.2.1 to 3.2.6 below.

The final goal is to implement each of these evaluation metrics in the challenge. Thus, users will be able to evaluate their algorithm on the proposed scenarios using various metrics. The new metrics should be implemented in the same workflow as the ones described in part 1.2.

2.2.1. Pedestrian-robot similarities

This section of metrics compares directly the robot path to the one of the agents, showing how similar a robot path and an agent path are. We use statistics on the crowd as a reference for the agent. A value of "1" for means a high similarity between the robot path and the average agent of the crowd.

- Time to goal similarity

The time to goal similarity compares the time taken by the robot to reach its goal to the time taken by an agent of the crowd (average time of the crowd). The metrics is given below:

$$\Delta_{ttg} = \frac{T_{robot}}{T_{agent}}$$

- Path length similarity

The path length similarity compares the length of the robot path to the length of the average agent path. The formula for this metric is given below:

$$\Delta_L = \frac{L_{robot}}{L_{agent}}$$

- Heading similarity

The heading similarity shows similarities between the variation of rotation between the robot and the average agent of the crowd. The formula for this metric is given below:

$$\Delta_\theta = \frac{1}{T_{robot}} \sum_{0}^{T_{robot}} \left(\frac{\dot{\theta}_{robot}}{\dot{\theta}_{agent}}\right)$$

- Velocity similarity

The velocity similarity compares the robot velocity to the one of the average agent. The mathematical formula is given below:

$$\Delta_v = \frac{1}{T_{robot}} \sum_{0}^{T_{robot}} \left(\frac{V_{robot}}{V_{agent}}\right)$$

In the same idea we compare the velocity of the robot to the average velocity of its neighbors with the following formula:

[13] http://crowdbot.eu/deliverable-1-3-specification-of-scenarios-requirement-update/

$$\Delta_{v|N} = \frac{1}{T_{robot}} \sum_{0}^{T_{robot}} \left(\frac{V_{robot}}{V_{neighbors}}\right)$$

- Overtaking count

This metric counts the number of overtakes the robot performs, to be compared with the average number of overtakes of an agent of the crowd.

A high score means the robot is more aggressive than the rest of the crowd, while a low score means the robot is blocking the crowd.

2.2.2. Crowd-robot interaction

- Proximity

We define proximity as the shortest distance to agents during the whole trajectory of the robot. The formula is given below:

$$P = \sum_{0}^{T_{robot}} d_{min}^{-1}$$

This metric is robot dependent and crowd dependent, but can be used to compare different robots in similar situations.

- Normalized proximity

This metric is defined as the normalized distance between the robot to the barycenter of its neighbors.

This metric is density independent, thus relevant in scenarios with different crowd density.

$$\hat{P} = \frac{std(d1, \ldots, dn)}{avg(d1, \ldots, dn)}$$

Where “std” is the standard deviation and “avg” is average for distances between the robot and its neighbors.

2.2.3. Shared control metrics

- Agreement

We defined agreement in terms of the deviation of the direction of user’s commands from the direction of the final shared control’s velocity. Mathematically, agreement is given below:

Define direction

$$\theta(u) = tan^{-1}\left(\frac{v}{w}\right)$$

$$a_i = 1 - \frac{|\theta(z_h^i) \ominus \theta(u_{SC}^i)|}{\pi}$$

$$agreement = \frac{\sum_{i=0}^{N} a_i \cdot \Delta t_i}{\sum_{i=0}^{N} \Delta t_i}$$

where v and w are the translational and rotational velocities $u \sim [v\ w\ ]$, a_i is the normalised agreement at time step, u_{SC}^i is the final output of the shared control (PSC or linear blending). N is the number of samples available in which data from both the measured input of the user, z_h^i coincide in time with u_{SC}^i . Δt_i is the duration of user’s input command z_h^i.

For the same wheelchair user, we expect the agreement to be high when there are no or only a few obstacles in proximity. On the contrary, when there are obstacles in proximity, agreement would be lower as the wheelchair is providing more assistance. Indeed, this metric would be more informative if analyzed with other metrics such as user input entropy and subjective metrics.

- User input entropy

Entropy determines how much information or disorder is in the user's command. The more erratic and jerkier the user's input, the more entropy the user's input exhibits. Thus, entropy can indicate the degree of difficulty in a similar manner to jerk. Entropy is estimated from the angular component, $tan^{-1}(\frac{v(z_h)}{w(z_h)})$ of the user's input [49].

- Fluency of Commands

We observe the fluency of commands of the user (temporal continuity) given that the reactive navigation is intervening with its desired motion, as a second reference measurement of the disagreement with the decisions of the assistive navigation.

$$F \quad = \frac{1}{N} \sum_{t=t_0}^{t_N} 1 - \ |u_h^t - u_h^{t-1}|$$

where u_h^t represents the user given command at time t, and N the number of samples taken in the interval $t_0 \ \ to \ t_N$.

- Objective metrics

In a controlled physical experiment, self-assessed metrics of performance can be used to evaluate shared control from the perspective of the participant [50]. The perceived workload is captured by the NASA-TLX. The idea here is that better performing shared control will reduce the perceived workload in comparison to other shared control. The USE questionnaire captures how well a user is satisfied with the shared control, and it is important because we want users to be content using our shared control.

2.2.4. Additional metrics

The challenge is currently in its early days. It is subject to evolve with the remarks and feedback of challengers. We expect challengers to request additional metrics for extended analysis.

Such metrics will be classified as "Experimental" and will be tested and validated with CrowdBot teams, before being officially integrated to the Challenge 2021+.

2.2.5. Updated report

The data generated by the simulation will serve as an input of the "challenge summary", a tool that will be made available for challengers to visualize their performances in scenarios. They will be able to visualize their score by metrics and by parameters (i.e. with or counter flow, low density only, medium and high density…).

They will have the option to compare their scores with reference scores made available for them.

3. Dissemination strategy

3.1. Dissemination objective

The main objective of the CrowdBot Challenge is to stimulate research on robot navigation in human crowds, in order to accelerate the emergence of technologies that guarantee safety while allowing robots to perform their tasks efficiently. The objective of the challenge is to stimulate the teams through a competition that will be initiated during the CrowdBot project, looking for it to be continued beyond the end of the project.

Such a form of stimulation is classic in the field of robotics. We can take the example of the RoboCup which has been stimulating robotics enthusiasts for more than 20 years. These competitions are as much at the service of promoting robotics among young people as they are used in research to accelerate the development of certain robotic functionalities. This is for example the case of the DARPA Challenge.

In order to have a lasting impact on the field of mobile robotics, we hope that the CrowdBot challenge will involve a critical number of research teams over a significant period of time. However, unlike other challenges, such as the DARPA Challenge for example, we cannot guarantee the success of this competition by attracting funding. We need to convince the community of the merits and interests of our challenge in order to hope for its success and sustainability.

We believe that the CrowdBot challenge can remove some of the obstacles to the development of solutions. The first difficulty is linked to experimentation. CrowdBot challenge provides a simulator to overcome the difficulties of experimentation, the simulator provides this "crowd" that can be difficult to gather around a robot, and does not raise ethical issues related to the risks of experimenting robots near humans. The second difficulty relates to the demonstration of the superiority of certain navigation solutions over others, a demonstration that is essential to enable, for example, the publication of results. The CrowdBot Challenge offers standardised situations for experimentation, which can be repeated at will to facilitate comparisons.

Our criteria for evaluating the success of this objective are:

1) To make the CrowdBot simulator a standard and unavoidable tool in the evaluation of robot navigation functionalities in human-populated environments,
2) To attract more and more research teams related to the subject of robot navigation each year,
3) To make the challenge last beyond the duration of the CrowdBot project.

3.2. Dissemination principles

In order to meet the criteria listed above, we set the following principles in the development of the CrowdBot Challenge.

3.2.1. An open source crowd simulator

In the CrowdBot Challenge, the purpose of the crowd simulator is to simulate the activity of a crowd around the robot. The risk raised by the use of a simulator is not to be able to realistically simulate the reactions of virtual humans around the robot.

In order for the community to accept to use this artifact for evaluation and comparison of navigation techniques, it is important that the operation of the simulator be transparent, its capabilities understood by all, and its possible evolution.

For this, it is important to be as transparent as possible about its operation, allowing access to its code and the richest possible documentation. It is with this objective in mind that we are now basing the simulator on

UMANS[14] and ChAOS[15] software whose codes are open, and which allow us to reproduce several state of the art crowd simulation techniques and to generate character animations in a very easy way.

3.2.2. Use of standard robotics tools

To be easily taken in hand by the community, the CrowdBot Challenge is aligned with the standard tools of the robotics community. In particular, the tested navigation functions can be integrated into ROS[16]. The Challenge also relies on Unity[17], which is one of the most popular game engines and increasingly known by the robotics community to perform visual simulations of robotic environments.

3.2.3. Propose scalable scenarios and evaluation metrics

It would be counterproductive to align on a single line the way in which methods are evaluated and compared. Each robot, each navigation technology may attempt to improve a specific performance, e.g., safety, efficiency, etc. The CrowdBot challenge must offer a rich range of scenarios and metrics, and we must let the community evolve these standards.

3.2.4. Involve external teams (to the CrowdBot project) in the organization of the Challenge

In the interest of openness and fairness of the Challenge, it is important to involve outside teams in its organization. For example, teams from different continents, as well as men and women researchers, or senior and younger researchers, could be involved.

3.3. Strategy for the organization of the first CrowdBot Challenge

For the launch of the first CrowdBot Challenge, we do not wish to launch a real campaign of evaluations of different robot navigation techniques, but rather a first test challenge where we associate on invitation some international teams focused on the subject of robot navigation in crowds with the aim of :

1) Collect their experience on the challenge.
2) Start to involve them eventually in the organization of the evaluation campaigns to come.

More precisely, the launch of the first CrowdBot challenge will coordinate 3 actions for its diffusion:

1) The submission of a conference paper describing the principles underlying the evaluation of navigation techniques through a simulation tool such as that of the CrowdBot Challenge,
2) The invitation of some international teams to test our simulator and to test their own navigation techniques,
3) The organization of a workshop to share the feedback on our simulator, and to present the tracks of each one for the evolution of the evaluation tool, as well as the organization of the first official challenge.

The targeted event for these three events is the ICRA 2021 conference. If they are refused, a submission will be made to IROSS, HRI and RSS.

[14] Van Toll, W., Grzeskowiak, F., López, A., Amirian, J., Berton, F., Bruneau, J., ... & Pettré, J. (2020, May). Generalized Microscropic Crowd Simulation using Costs in Velocity Space. In *Symposium on Interactive 3D Graphics and Games (I3D'20).*

[15] https://project.inria.fr/crowdscience/download/

[16] http://wiki.ros.org/

[17] https://unity.com/

3.4. CrowdBot Challenge webpage

The CrowdBot Challenge and the related tools are progressing on a daily basis to meet the target of the ICRA 2021 conference. The required information to test or participate, presented along this deliverable as well as the online resources (git and wiki) will be updated on the web. We have created a dedicated webpage on the CrowdBot website: www.crowdbot.eu/crowdbot-challenge/

Conclusion

Our goal for the CrowdBot Challenge is ambitious: we want to organize around the simulation tools developed in the CrowdBot project an annual campaign and become the standard for evaluating robot navigation in populated environments. There is still a long way to go to reach this goal, and we need to think about how to meet this objective beyond the end of the CrowdBot project.

With this goal in mind, the adhesion of multiple teams to our benchmarking method is an essential step that we are seeking to achieve within the terms of the project. It is a delicate task to provide both quality usage support, and to move the challenge towards more features, more scenarios, in order to expand the community of people interested in its use.

This deliverable has allowed us to take stock of our technical progress towards this goal. An important milestone has been reached: our simulator is technically ready for use in the challenge. We still have to develop the scenarios and evaluation metrics in collaboration with the teams involved in the challenge.

We have also exposed our communication strategy around the robot. The next step is to disseminate these tools (Toward CrowdBot Challenge 2020) through a draft paper and workshop at the ICRA 2021 conference, hoping that the quality of the submissions will allow us to be selected for this edition.

Annex - CrowdBot Challenge Wiki

CrowdBot Challenge is available and free on INRIA's GitLab. To understand and use the Challenge, a wiki documentation is available on the GitLab.

On the GIT, different things are available:

- Source code
- Unity simulator
- ROS tools essential to the challenge
- 3D models of the challenge
- Scenarios
- Analysis tools
- Documentation to link the different elements
- A tool for reporting bugs via issues and user requests

Executables are available via download links in the wiki. These wiki[18] consists of 3 parts:

- Installation and configuration
- Tutorials
- User cases

This annexe is a copy of our wiki in April 2020. This wiki will evolve and remain linked to the version of the challenge available throughout its evolution.

1. Configuration and Installation

1.1. General recommendations

The CrowdBot simulator requires Unity and ROS. The recommanded operating system is Ubuntu 18.04 LTS (long time support). However, it is possible to use the simulator with Windows 10 (with Windows subsystem Linux) and IOS (with a virtual machine).

The CrowdBot simulator itself needs 30 GB of free space on your hard drive.

You will actually need more space for Unity, ROS, a (eventual) virtual machine or Windows Subsystem Linux.

We recommand to use this simulator with the minimum configuration:

- RAM : 32GB
- Processor : Intel® Core™ i7-7920HQ CPU @ 3.10GHz × 8
- Graphics : Nvidia Quadro M2200 (or GTX1050)

[18] https://gitlab.inria.fr/crowdbot/crowdbotunity/-/wikis/home

- Disk : minimum 30GB

1.2. Table of Content

1.3. Using Linux (recommanded)

1.3.1. Ubuntu (recommanded)

- Version : Ubuntu Bionic Beaver 18.04 LTS

- Installation:

You will find an image of the OS in this Download section of the Ubuntu website.

Please refer to the Ubuntu website for tutorials and questions regarding the installation of this distribution.

1.3.2. ROS

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.

- Version: Melodic (LTS)

- Installation: Use the following link fory a full detailed installation tutorial: Melodic (LTS)
- Quick installation:

In a terminal:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
main" > /etc/apt/sources.list.d/ros-latest.list
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
sudo apt update
sudo apt install ros-melodic-desktop-full
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator
python-wstool build-essential
sudo rosdep init
rosdep update
```

You will also need to install the ROS package "rosbridge" which connects ROS and Unity together.

```
sudo apt-get install ros-melodic-rosbridge-suite
```

1.3.3. Git clone the project

```
mkdir -p ~/CrowdBot_simulator/
cd ~/CrowdBot_simulator
git clone git@gitlab.inria.fr:CrowdBot/CrowdBotUnity.git
git submodule init
git submodule update
```

1.3.4. Catkin

catkin is the official build system of ROS and the successor to the original ROS build system, rosbuild. catkin combines CMake macros and Python scripts to provide some functionality on top of CMake's normal workflow. catkin was designed to be more conventional than rosbuild, allowing for better distribution of packages, better cross-compiling support, and better portability. catkin's workflow is very similar to CMake's but adds support for automatic 'find package' infrastructure and building multiple, dependent projects at the same time.

- Version: see ROS version
- Installation: If not installed with ros, run the following command in a terminal.

```
sudo apt-get update && sudo apt-get install ros-melodic-catkin
```

Also, you should create a catkin workspace, where your ros nodes will be built. A submodule in the CrowdBotUnity project is present for this purpose.

```
cd ~/CrowdBot_simulator/catkin_crowdbotsim
catkin_make
```

1.3.5. Unity Hub

- Version: Latest
- Installation:

The best way to use Unity under Linux is by installing Unity Hub which allows you to manage your different projects and the different versions of Unity installed on your hard drive.

First download the AppImage file by clicking on "Download Unity Hub" on this page.

In a terminal:

```
mkdir ~/Softwares
mv ~/Downloads/UnityHub.AppImage ~/Softwares/
chmod +x ~/Software/UnityHub.AppImage
```

You can now launch Unity Hub like a standard application by clicking on it or executing the command `~/Software/UnityHub.AppImage &` in a terminal.

1.3.6. Unity

- Version: at least 2018. (Highly) Recommanded: latest 2019 version.
- Installation

In the Unity Hub, click on:

Installs => Add => Select Unity 2019.X.X (the latest 2019 version) => Next => select "Documentation" => Done

This manipulation will install Unity on your computer.

1.3.7. Mono

Sponsored by Microsoft, Mono is an open source implementation of Microsoft's .NET Framework based on the ECMA standards for C# and the Common Language Runtime. A growing family of solutions and an active and enthusiastic contributing community is helping position Mono to become the leading choice for development of cross platform applications.

- Version : see Ubuntu version
- Installation:

Install Mono here

- Quick install:

In a terminal, run the following commands

```
sudo apt install gnupg ca-certificates
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF
echo "deb https://download.mono-project.com/repo/ubuntu stable-bionic main" |
sudo tee /etc/apt/sources.list.d/mono-official-stable.list
sudo apt update
sudo apt install mono-devel
```

1.4. Using other Linux distribution

Other Linux distros have not been tested. However, it should be possible as both ROS and Unity provide their software for other distros.

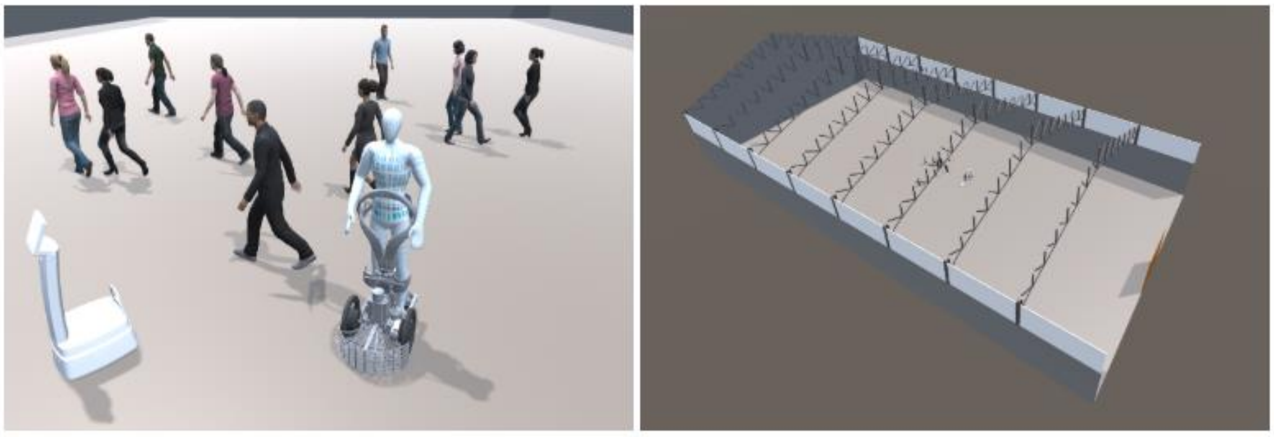
1.5. Windows

1.6. IOS

2. Tutorials

2.1. Tutorial 1 - Qolo and Cuybot in a crowded Hangar

Note: if you are totally new with Unity, you can explore the set of tutorials available here.



2.1.1. Table of content

2.1.2. Objective

The objective of this tutorial is to recreate from scratch the scene "Tuto1" which you can find in the directory **Assets/Scene/Tutos/**

Doing so, you will learn the important components necessary to run a simulation.

First, create a new scene (File -> new scene) and save it in the Asset/Scenes directory. Remove everything that is in the scene (camera and light).

2.1.3. Configuration files

The philosophy of the CrowdBot simulator is to set up everything with several layers of configuration files. It standardizes the notion of scenario defined in the CrowdBot project. It also allows you to have control over the experiment when using a build version of the CrowdBot simulator.

More details are given in the section Configuration files.

2.1.4. Main manager

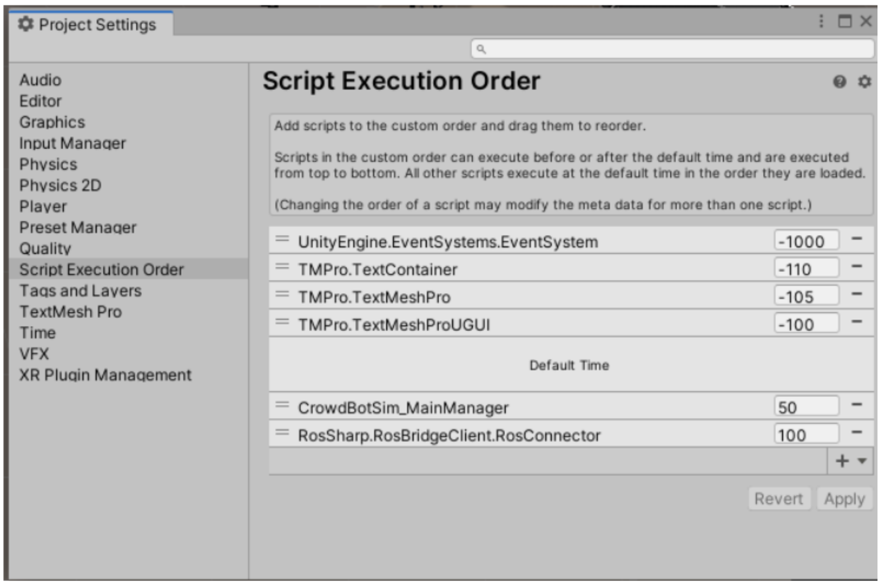
2.1.4.1. Description

The MainManager is the prefab in charge of managing the experiment.

It reads the configurations files and loads the relevant assets in the simulation.

It is a mandatory asset which have the tag "GameManager" in the inspector.

Pro-tip : the main manager must have the highest priority in the script execution order, after default time.



Script Execution Order

Add scripts to the custom order and drag them to reorder.

Scripts in the custom order can execute before or after the default time and are executed from top to bottom. All other scripts execute at the default time in the order they are loaded.

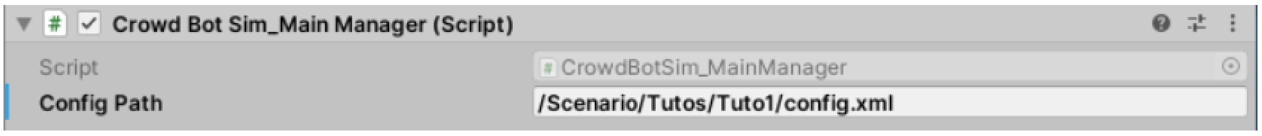
(Changing the order of a script may modify the meta data for more than one script.)

UnityEngine.EventSystems.EventSystem	-1000
TMPro.TextContainer	-110
TMPro.TextMeshPro	-105
TMPro.TextMeshProUGUI	-100
Default Time	
CrowdBotSim_MainManager	50
RosSharp.RosBridgeClient.RosConnector	100

In the "Project" window, search for the **MainManager** prefab in the search bar, or in the directory *Assets/Prefabs/Simulation*, and Drag-and-drop it in the Hierarchy window.

Then, in the inspector, complete the path toward the main config file you plan to use. This path is concatenated to the current directory: ${path_to_CrowdBotUnity_project}${config_path}.

Pro-tip: don't forget the '/' at the beginning of the path


Crowd Bot Sim_Main Manager (Script)
Script CrowdBotSim_MainManager
Config Path /Scenario/Tutos/Tuto1/config.xml

2.1.5. Stage

The stage is the asset that contain the objects that compose your environment.

For instance, select the prefab "Hangar" in the directory "Assets/Stages/Tuto1/", and drap-&-drop it in the scene.

It contain the 3d elements for the scene: the floor, walls and obstacles, and the lights.

It has a "CrowdBotSim_TrialManager" script that has an obstacle container attribute: it is a child object of the stage that would contain all obstacles. An obstacle can be either a pillar or a wall, and in each case, it has be tagged (in the inspector) as such. Such tagged obstacles are taken into account in the crowd simulator (for local avoidance).

If you create your own scene, you would need to create a NavMesh, a component used by Unity for global planning for the crowd simulation. To do so, select the floor and walls, and in the "Navigation" window, bake the NavMesh. More details about this procedure here.

For a more precise NavMeshn you might need to select "Height mesh" in advanced options.

The Stage also contain the light of your scene. The automatic rendering of the light is deactivated in this project. You might need to generate the light in "Window > rendering > lightning setting".

2.1.6. Agents Models

In the scene, you will need to insert some animated characters to be controlled by the crowd simulation.

The CrowdBot simulation toolbox is the first robotics simulator that propose a high level of realism of a crowd. The human of this simulator uses the same two components as a robot: a visual representation and a collider. We use animations in order to have a realistic movements.

The prefab "PlasticMan"in "Prefabs/HumanModels" is a faceless character that is usable by default in the simulator. You can drag&drop it in the scene.

It has an animation controller script attached to it, and an animator with the "LocomotionFinal" controller.

You can use the more realistic set of characters "RocketBox" which is an addon to the project, for which you should request access here.

The RocketBox, provide more diversity: it is a set of 40 realistic characters (20 male and 20 female). For each character, we have different levels of resolution (i.e more or less complex mesh). Also, we have textures to apply on the mesh, which gives a realistic human visual representation. We also have variation on the texture (hair, skin, clothes).

2.1.7. "Player"

The CrowdBot Simulator is designed for user studies, which means that you can have an actual person interaction with the elements of the simulation.

For more details, please refers to Tutorial 3 and the Inria use case.

In this tutorial, we are not interested into immersing someone. However, we still want a free camera that render the scene from were we want: we can use the prefab "CamPlayer" following the directory Assets/MainAssetS/Agents/Player.

A player should be tagged as player and have a "Head" component tagged as HeadPlayer.

The CamPlayer defines a camera which render on the display 1 and is tagged as MainCamera.

2.1.8. Robots

The main Robots available in the simulator are : Qolo_sensors, Cuybot_sensors, Pepper_sensors, Wheelchair_sensors. Other robot models exist, which are a partial version of the previous ones.

Select the prefab Qolo_sensors (Assets/Robots/RobotsModels) and drag and drop it in the scene.

The root object of a robot is tagged "Robot". It contain 2 childs: a RosConnector and a base_link. The root of the object have a script "URDF Robot" which allow you to control the parameters of the base_link.

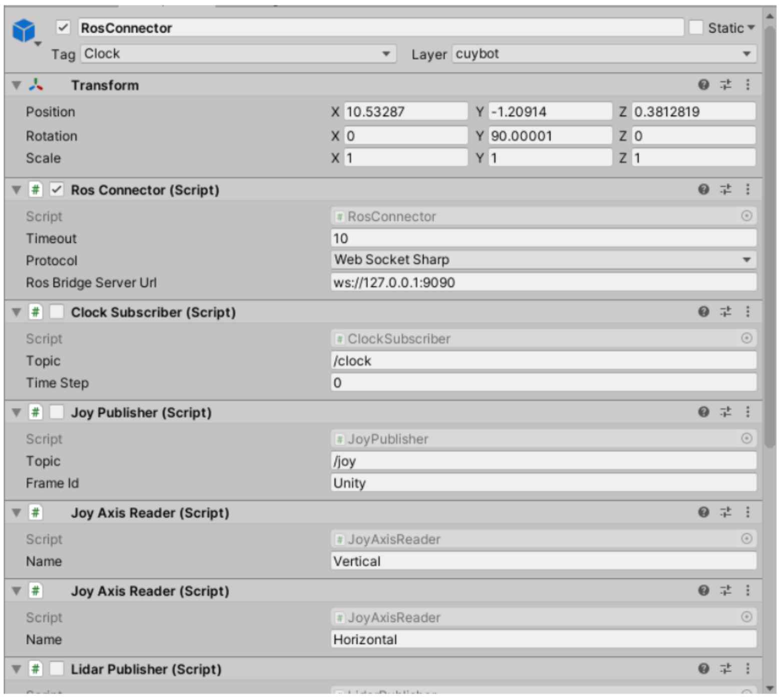
2.1.8.1. ROS

The RosConnector is the script connecting the simulator to ROS using rosbridge. If you are unfamiliar with ROS, then check the tutorials.

The simulator uses the library ROS# which communicates with ROS. It means that you can run the simulator on one computer or server and run your ros nodes on another computer.

The RosConnector script have a "Ros bridge server url" which correspond to your $ROS_MASTER_URI parameter.

Important note: the RosConnector script is the only one of the scripts attached to the RosConnector that is enabled.



RosConnector Static
Tag Clock Layer cuybot

Transform
Position X 10.53287 Y -1.20914 Z 0.3812819
Rotation X 0 Y 90.00001 Z 0
Scale X 1 Y 1 Z 1

Ros Connector (Script)
Script RosConnector
Timeout 10
Protocol Web Socket Sharp
Ros Bridge Server Url ws://127.0.0.1:9090

Clock Subscriber (Script)
Script ClockSubscriber
Topic /clock
Time Step 0

Joy Publisher (Script)
Script JoyPublisher
Topic /joy
Frame Id Unity

Joy Axis Reader (Script)
Script JoyAxisReader
Name Vertical

Joy Axis Reader (Script)
Script JoyAxisReader
Name Horizontal

Lidar Publisher (Script)

You should run the command

```
roslaunch crowdbotsim unity_sim.launch
```

in a terminal on the computer running the ROS core.

2.1.8.2. The Clock

The CrowdBot simulator gives you full control of the time.

To use an external clock, the RosConnector have to be tagged as "Clock" and have must have a ClockSubscriber script attached to it, with a topic name "/clock".

On the computer running ROS, you can publish a default clock using the following command

```
rosrun crowdbotsim clock_publisher.py ${sleep_time} ${delta_time}
```

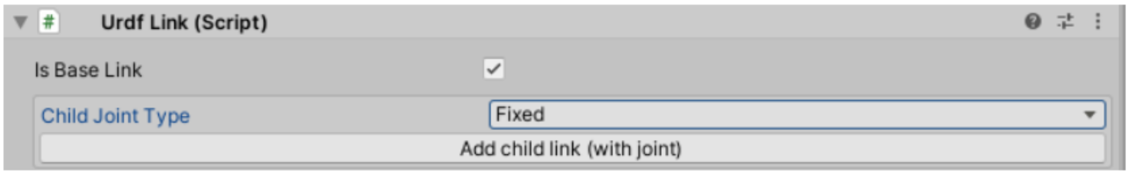
Which publish a clock every "sleep_time" seconds (first parameter) updated of "delta_time" seconds (second parameter).

Important note: the clock controls the time step of the whole simulation: the crowd, the sensors, physics... Which means that a high delta time might give absurd results (robot going through floor, *people not avoiding...), and a small one might be far from real time if it is smaller than the time required to compute a step of the simulation. These parameters depends on the simulation itself and must be tuned for each case. Generally speaking, we recommand a delta time between 0.01 s and 0.1 s.*

2.1.8.3. The URDF Robot

The rest of the game object consist in virtually building the robot, following the URDF standard. It starts with the base_link.

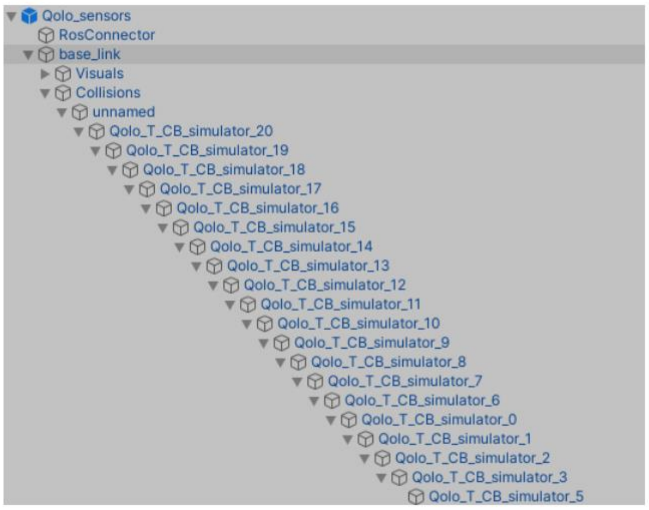
Important note: the name convention base_link, as a child of the object tagged robot, is important for the crowd simulator. Its position in the scene is considered to be the position of the robot


Urdf Link (Script)
Is Base Link ✓
Child Joint Type: Fixed

In the URDF standard, a link is a combination of visual elements and collision element.

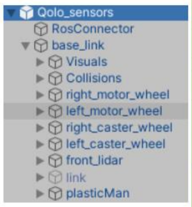
We can connect links together with joints by clicking on "add link with joint" to create a child link.

note : The physics engine limits the colliders (collision elements used to compute contacts) to convex shapes when using non kinematic rigid body. To improve precision of the collision elements, we divide the complex 3d models in smaller pieces and link them in a child/parent relationship in the Collision element. For instance with the base_link of Qolo, the structure of the robot is divided in 20 pieces.


```
▼ Qolo_sensors
    RosConnector
  ▼ base_link
    ▶ Visuals
    ▼ Collisions
      ▼ unnamed
        ▼ Qolo_T_CB_simulator_20
          ▼ Qolo_T_CB_simulator_19
            ▼ Qolo_T_CB_simulator_18
              ▼ Qolo_T_CB_simulator_17
                ▼ Qolo_T_CB_simulator_16
                  ▼ Qolo_T_CB_simulator_15
                    ▼ Qolo_T_CB_simulator_14
                      ▼ Qolo_T_CB_simulator_13
                        ▼ Qolo_T_CB_simulator_12
                          ▼ Qolo_T_CB_simulator_11
                            ▼ Qolo_T_CB_simulator_10
                              ▼ Qolo_T_CB_simulator_9
                                ▼ Qolo_T_CB_simulator_8
                                  ▼ Qolo_T_CB_simulator_7
                                    ▼ Qolo_T_CB_simulator_6
                                      ▼ Qolo_T_CB_simulator_0
                                        ▼ Qolo_T_CB_simulator_1
                                          ▼ Qolo_T_CB_simulator_2
                                            ▼ Qolo_T_CB_simulator_3
                                                Qolo_T_CB_simulator_5
```

The rest of the Qolo hierachy are the motorised wheels (see section controllers) which use continuous joints, and the caster wheels which use, for the moment, a floating joint with translation constraints, and a sphere as collider. Also, the sensors are added as a new links with fixed joint (see section sensors).

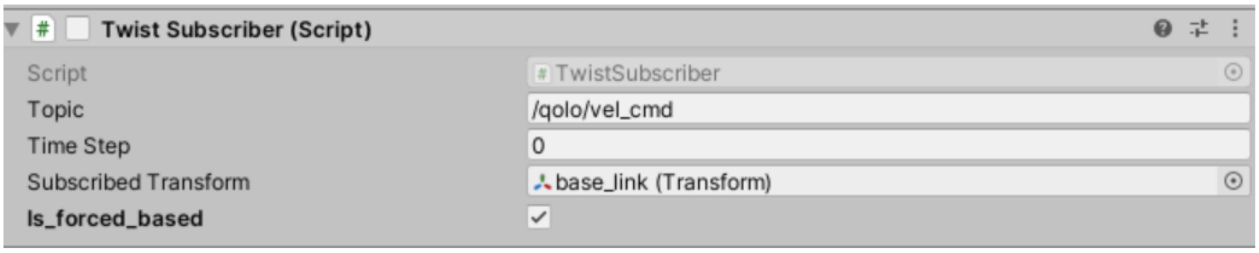
For Qolo, We added a dummy plastic man for the pilot.



2.1.8.4. Controllers

The default way to control a robot is by publishing a twist message through ROS.

In the RosConnector, you should use a TwistSubscriber script which subscribe to the the published tiwst with the corresponding name.



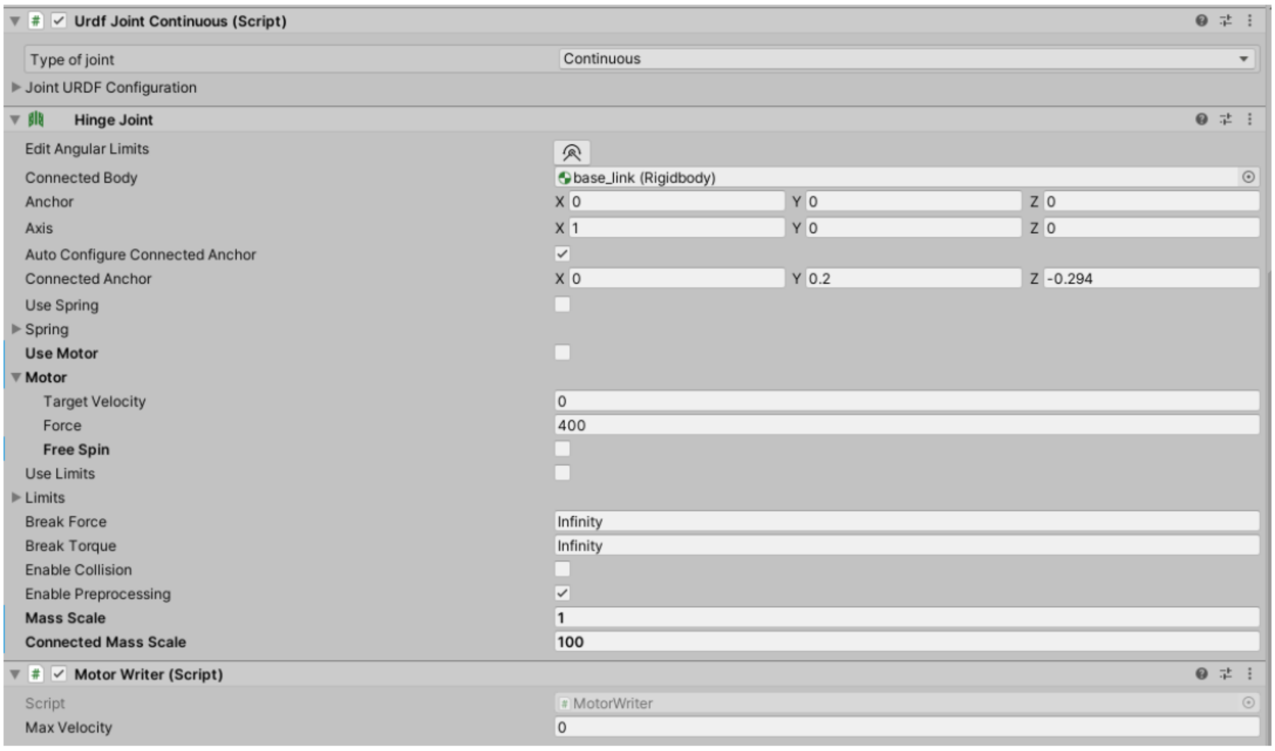
Twist Subscriber (Script)

Script	TwistSubscriber
Topic	/qolo/vel_cmd
Time Step	0
Subscribed Transform	base_link (Transform)
Is_forced_based	✓

The subscribed transform should be the base_link of the robot.

You can select the modes of controller:

- Kinematic: This mode ignore physics, thus will move the robot precisely but without realism. It is a good mode to begin with
- Forced based: This mode use physics and apply a second Newton law F = m * dv/dt where m is the mass of the base_link, dv is the difference between the base_link rigidbody velocity and the desired velocity, and dt is the time spent between two computations. A similar treatment is apply to the torque. Due to approximation of the mass, plus friction constraints on the wheels, this mode will not be precise, the desired velocity will not be reached. However, it is more realistic than kinematic mode for contacts.
- Differential drive (Fix in progress): * (this mode is only available for non-holonomic robots)* In this mode we use the virtual motors on the motorised wheel, i.e. a continuous joint with a "Motor Writer" script, and the odometry of the robot (see sensors). This mode require the tuning of the controller (PID) but will provide the best compromise between precision and realism.



Urdf Joint Continuous (Script)

Type of joint	Continuous
Joint URDF Configuration	

Hinge Joint

Edit Angular Limits	
Connected Body	base_link (Rigidbody)
Anchor	X 0 Y 0 Z 0
Axis	X 1 Y 0 Z 0
Auto Configure Connected Anchor	✓
Connected Anchor	X 0 Y 0.2 Z -0.294
Use Spring	
Spring	
Use Motor	
Motor	
Target Velocity	0
Force	400
Free Spin	
Use Limits	
Limits	
Break Force	Infinity
Break Torque	Infinity
Enable Collision	
Enable Preprocessing	✓
Mass Scale	1
Connected Mass Scale	100

Motor Writer (Script)

Script	MotorWriter
Max Velocity	0

2.1.8.5. *Sensors*

Proximity sensors

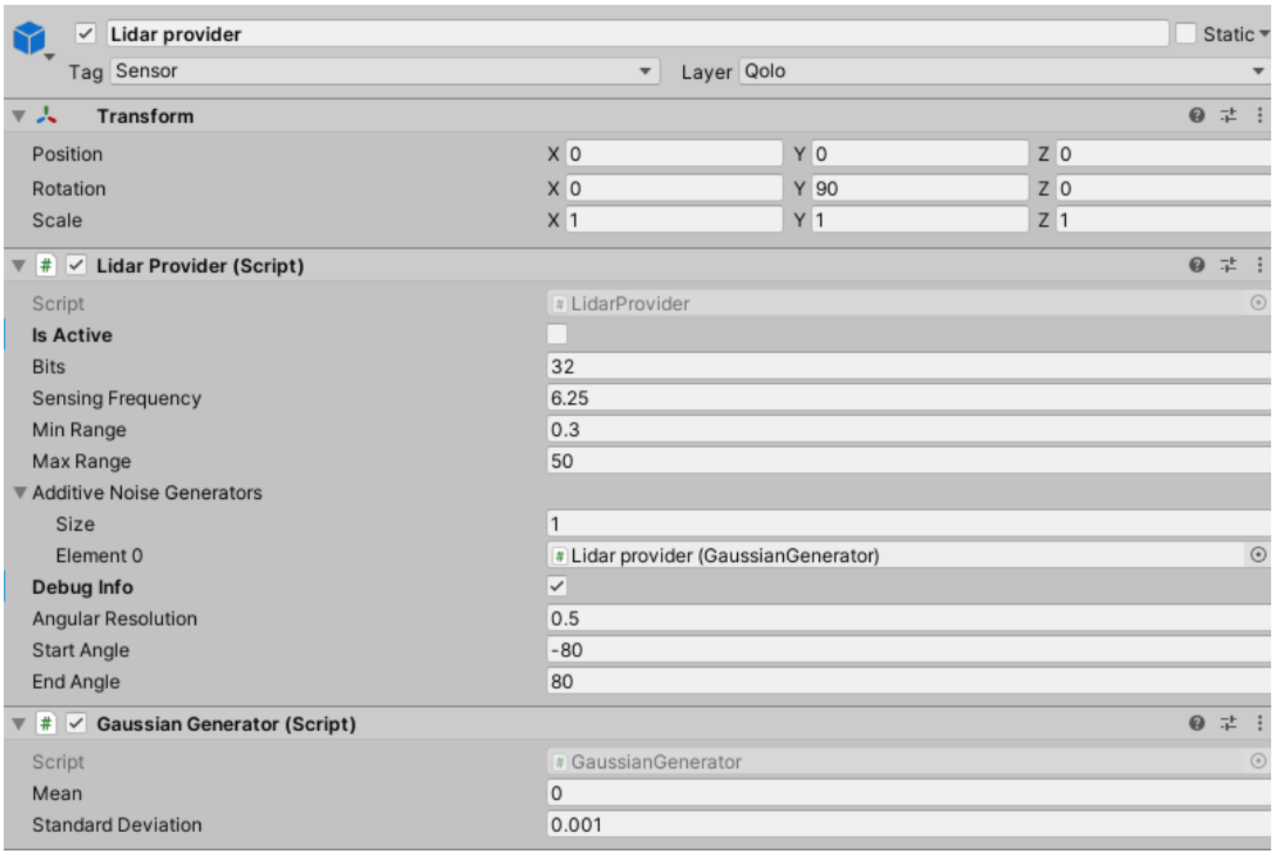
Our virtual sensors give the same output as real ones.

Our simulated sensors offer the possibility to tune the usual parameters according to the real sensor properties, such as maximum and minimum distance of detection, maximum incidence angle required for detection, field of view, update rate, bit-precision...

In order to provide more realistic sensors simulation, we have the possibility to add various types of noise to any type of sensors.

We implemented:

- a gaussian noise generator
- an offset generator
- a peak generator



Lidar provider — Static
Tag: Sensor — Layer: Qolo

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	90	0
Scale	1	1	1

Lidar Provider (Script)

Script	LidarProvider
Is Active	☐
Bits	32
Sensing Frequency	6.25
Min Range	0.3
Max Range	50
Additive Noise Generators	
Size	1
Element 0	Lidar provider (GaussianGenerator)
Debug Info	✓
Angular Resolution	0.5
Start Angle	-80
End Angle	80

Gaussian Generator (Script)

Script	GaussianGenerator
Mean	0
Standard Deviation	0.001

They are configurable and can be added to the output of any kind of sensors described below.

Robot sensor data are generated thanks to the 3D elements (their colliders and visual representation) of the simulation: the scene, and the human avatars.

About the sensors, we implemented

- Infrared (IR)
- ultrasound (US) proximity sensors
- 2D laser (LIDAR) sensors
- Camera (RGB)
- Depth sensor (RGB-D)
- Graphic based Lidar and depth (Work in progress)

For the US, IR, and Lidar, the simulated sensors cast a predefined number of rays in a given detection zone.

These rays are generated by the physics engine which returns a collision point when a ray hits a collider.

The rays are cast evenly in the detection zone, according to the predefined number of rays.

For a more realistic result, we propose to simulate the fact that such sensors fail to detect obstacles if the incidence angle is too high, by giving the possibility to specify a maximum incidence angle. If the ray hits an object in its path, and the hit angle is smaller than this value, then the object is detected and the distance between the sensor and the hitting point is recorded.

US sensors have specific detection zone, which we approximate by a teardrop function centered on the position and the angle of the sensor. The simulated US sensor returns the smallest observed value from the set of rays.

We approximate the detection zone of a IR sensor by a cone centered on the sensor reference in the simulated world. We can configure the maximum distance of detection, the angle of the cone and the level of sampling.

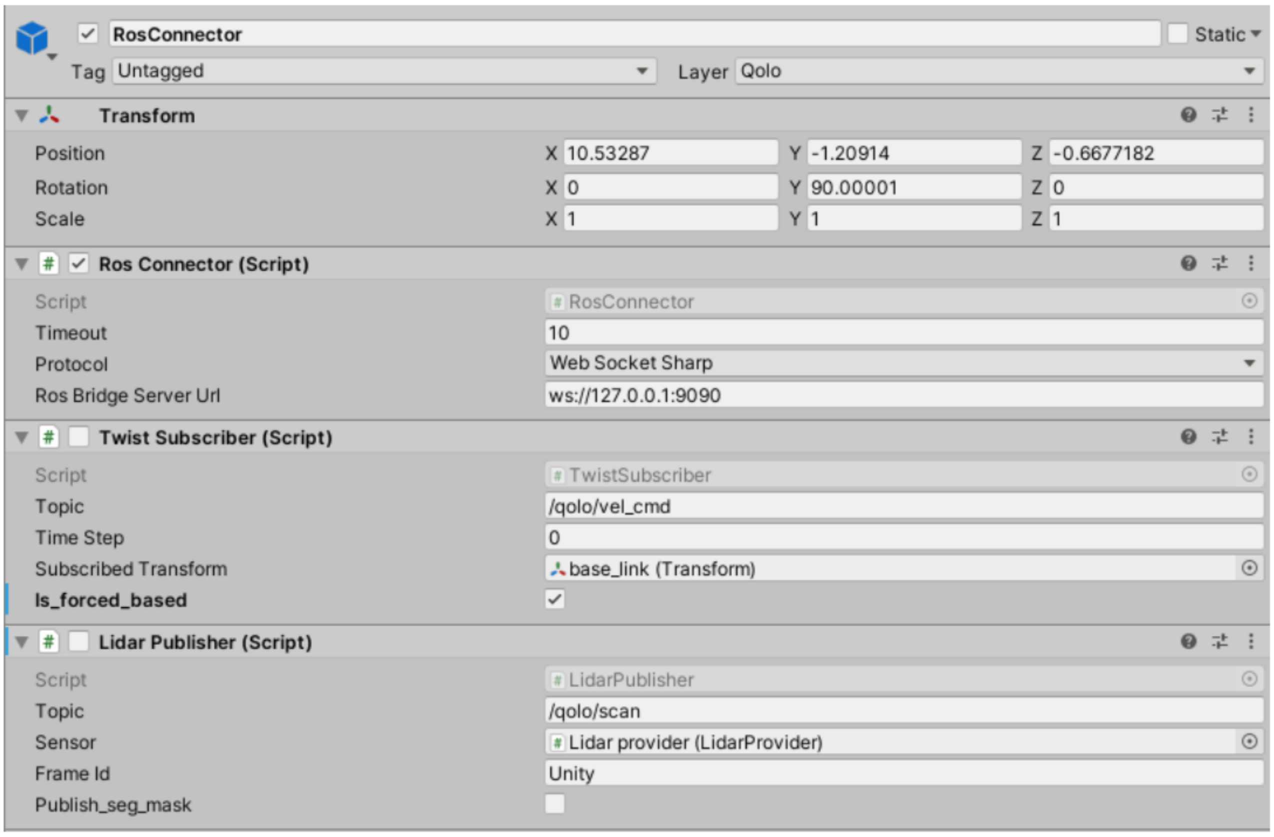
We simulate Lidar sensor by casting rays according to a partial sphere. You can configure the angular resolution, the starting and ending angles, as well as the minimum and maximum range of the LIDAR.

RGB cameras are simulated using simple images extracted from Unity.

The depth component of the RGB-D camera is simulated using Unity shaders, which are usually in charge of rendering textures. We use them to generate a depth texture, that gives us a grayscale image where each pixel value corresponds to the depth value. Also, the simulated camera is highly configurable (e.g. field of view, resolution...).

Generally speaking, a sensors is composed with 2 things: a sensor provider and and sensor publisher. The sensor provider is a script attached to a gameobject which is in the robot hierarchy, and which compute the synthetic data.

The sensor publisher is the script attached to the RosConnector which takes a sensor provider as input and communicates with ROS.


RosConnector — Static
Tag: Untagged — Layer: Qolo

Transform

	X	Y	Z
Position	10.53287	-1.20914	-0.6677182
Rotation	0	90.00001	0
Scale	1	1	1

Ros Connector (Script)

Script	RosConnector
Timeout	10
Protocol	Web Socket Sharp
Ros Bridge Server Url	ws://127.0.0.1:9090

Twist Subscriber (Script)

Script	TwistSubscriber
Topic	/qolo/vel_cmd
Time Step	0
Subscribed Transform	base_link (Transform)
Is_forced_based	✓

Lidar Publisher (Script)

Script	LidarPublisher
Topic	/qolo/scan
Sensor	Lidar provider (LidarProvider)
Frame Id	Unity
Publish_seg_mask	☐

Localization

We use odometry and JointState Reader for localization.

To initialize the JointState publisher, you should first click on the button "Generate" on the URDF robot script of the robot prefab.

Then, on the RosConnector you should add a "JointState patcher", fill the "Urdf robot" attribute by drag and dropping you robot from the hierachy, and click on "Enable" for publish joint state. Disable the script (the RosConnector script will enable it at startup), add a proper topic name (for instance /qolo/joint_state) and change the frameId from "Unity" to "base_link".

This maneuver allows to publish the joint_state on ROS which can tranform this information in "tf" messages and visualize it on rviz.

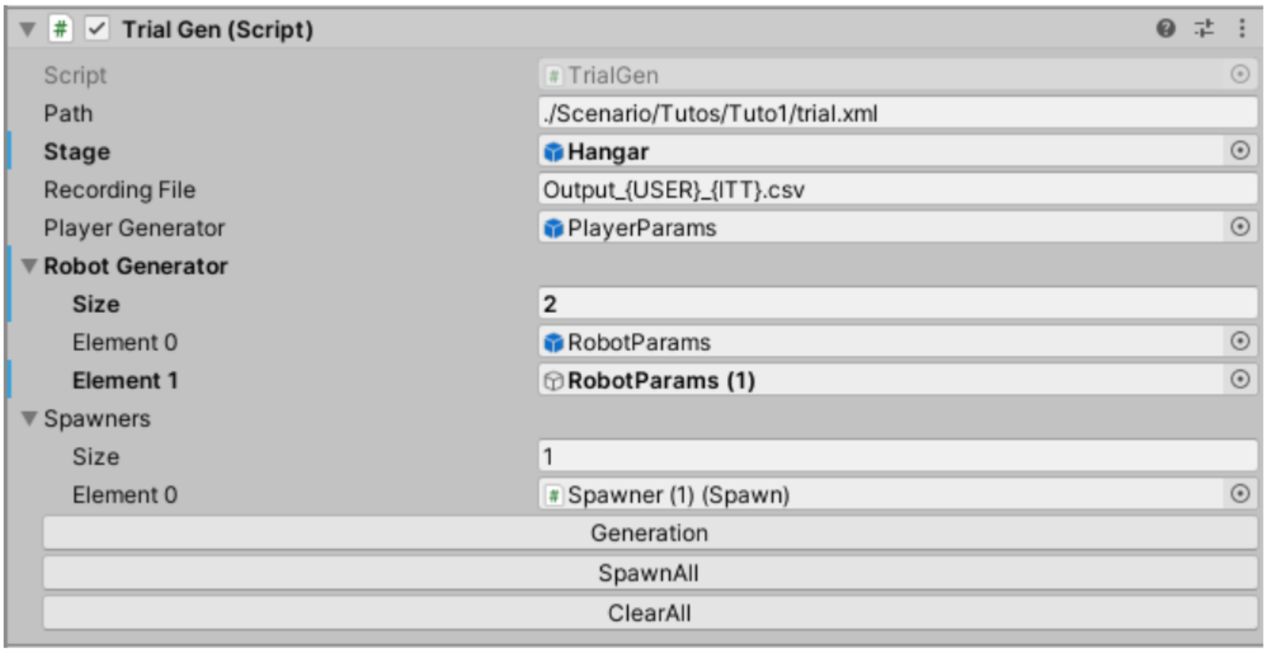
Now, we need to publish an odometry message. This is done with two components: an OdomPublisher on RosConnector and an OdomProvider script on the base_link of the robot.

The OdomProvider use 2 methods:

- (differential drive robots only) We can use Iverse Kinematic and the velocity of the motrised wheels to compute a linear and angular velocity. This method is subject to errors but is very similar to what is done in real life. This method returns 0 if the robot move with the Kinematic controller.
- (to fix) We read the velocity of the bae_link rigidbody directly. This method returns 0 if the robot move with the Kinematic controller.
- Note: The OdomProvider needs joints state readers on the wheels, so we recommand to use the Joint state patcher *

2.1.9. Trial Generator

The trial generator is a useful tool that allow you to generate trial configuration very easily. The picture show the different attributes that the trial generator needs to generate a trial configuration.



Trial Gen (Script)

Script	TrialGen
Path	./Scenario/Tutos/Tuto1/trial.xml
Stage	Hangar
Recording File	Output_{USER}_{ITT}.csv
Player Generator	PlayerParams
Robot Generator	
Size	2
Element 0	RobotParams
Element 1	RobotParams (1)
Spawners	
Size	1
Element 0	Spawner (1) (Spawn)

- Path: a relative path to the created file should be given. This file should also be written (by yourself) in the FileOrder reference in the main configuration file, if you want to use it in runtime. (cf configuration files)
- Stage: This attribute is the GameObject described in the section "Stage", which you want to use in the simulation.
- RecordingFile: an optional tool to record some data. By default, nothing is recorded. See tutorial 3. *Pro-tip: for more convenience, we suggest the use of rosbag for recording.*
- Player Generator: This attribute refers to a child of the trial generator GameObject. The "PlayerParams" have a PlayerGen script which have thet player GameObject as component, and details regarding it behavior in the crowd simulation. By default, the Player will use a camera controller and is not part of the simulation. You can give a control law to the player if you wish.
- Robot generator: it is an array of GameObjects, Each element of the array refers to a child of the TrialGenerator, which contain a "RobotGen" script. This script refers to a game object of a the robot you want in your simulation. You should specify a radius, a parameter used by the crowd simulation (local avoidance), and if you wish, control law (global) and control sim (local) generators. By default, we consider that the robot is being controlled through ROS messages.
- Spawners: it is an array of child of the TrialGenerator. A spawner is what allows you to generate a crowd. See the next section for an explanation on how to use it.

2.1.10. Spawners: Generate your crowd

The Spawn block contains all the classes used by the TrialGenerator object allowing spawning agents offline, visually modifying them in the editor and generating an XML trial file to run and XP with these agents.

- AgentGen.cs: Generate the agent part of the XML file.
- Spawn.cs: Spawn agents according to the linked AgentGen in a specific area.

- Goal.cs: Small script to handle sequence of goals/waypoints.
- SpawnEditorButtons.cs: Handle all the editor buttons for the spawns.
- ControlLawGen.cs: Interface to generate the control law part in the XML file.
    - ControlLawGen_LawFileData.cs: Generate XML for the law FileData.
    - ControlLawGen_LawGoals.cs: Generate XML for the law Goals.
    - …: One class for each existing control law.
- ControlSimGen.cs: Interface to generate the control sim part in the XML file.
    - ControlLawGen_RVO.cs: Generate XML for the sim RVO.
    - ControlLawGen_Helbing.cs: Generate XML for the sim Helbing.
    - …: One class for each existing simulator.

2.2. Tutorial 2 - The CrowdBot Challenge

This tutorial will be available in May 2020.

2.3. Tutorial 3 - Using plugins

This tutorial will be available in May 2020.

3. Use cases

The use cases per partner are not yet developed. They will be when the partners have taken ownership of the challenge.