Computers and Electrical Engineering 95 (2021) 107408

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

A membrane computing framework for social navigation in robotics[☆]

Ignacio Pérez-Hurtado [*], David Orellana-Martín, Miguel Á. Martínez-del-Amor, Luis Valencia-Cabrera

Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Universidad de Sevilla, Seville, Spain

ARTICLE INFO

Keywords:
Membrane computing
Enzymatic numerical P systems
Robotics
Social force model
Navigation

ABSTRACT

A mobile robot acting in a human environment should follow social conventions, keeping safety distances and navigating at moderate speeds, in order to respect people in its surroundings and avoid obstacles in real-time. The problem is more complex in differential-drive wheeled robots, with trajectories constrained by nonholonomic and kinematics restrictions. It is an NP-hard problem widely studied in the literature, combining disciplines such as Psychology, Mathematics, Computer Science and Engineering. In this work, we propose a novel solution based on Membrane Computing, Social Force Model and Dynamic Window Approach Algorithm. The resulting model is able to compute, in logarithmic time, the best motion command for the robot, given its current state, considering the surrounding people and obstacles. The model is compatible with other membrane computing models for robotics and suitable for an implementation on parallel hardware. Finally, a visual simulator was implemented in ROS and C++ for validation and testing.

1. Introduction

The problem of motion planning in robotics can be described in the configuration space of a robot as follows: *Given an initial robot configuration state and a goal configuration state, find a sequence of motion commands to move the robot towards the goal state while avoiding static and dynamic obstacles*. In general, the motion planning problem is crucial in robotics and other areas such as verification, computational biology, and computer animation [1]. In particular, a critical case is given when a wheeled or legged robot is walking in a human environment populated with people, for example a pedestrian street, an office, a hospital or a marketplace. In this situation, the robot must not only avoid obstacles while trying to reach its goal, but also generate socially acceptable trajectories. An extensive survey on human-aware navigation can be found in [2]. Moreover, it implies several philosophical questions concerning the challenges of the application of these technologies to real-life scenarios. In the case a robot has to choose irretrievably between injuring one out of two persons, what should it do? And who would be the one condemned for such an act? [3]. On the other hand, several applications for these technologies can be found in the literature [4]. The problem is complex and involves multidisciplinary fields including Psychology, Mathematics, Computer Science and Engineering. Furthermore, even if the robot's optimal velocity vector could be computed for each step of time, there are several physical constraints to be considered, such as acceleration limits and nonholonomic restrictions.

☆ This paper is for special section VSI-bioc. Reviews processed and recommended for publication by Guest Editor Dr. Xiaochun Cheng.
* Corresponding author.
E-mail addresses: perezh@us.es (I. Pérez-Hurtado), dorellana@us.es (D. Orellana-Martín), mdelamor@us.es (M.Á. Martínez-del-Amor), lvalencia@us.es (L. Valencia-Cabrera).

https://doi.org/10.1016/j.compeleceng.2021.107408
Received 22 October 2020; Received in revised form 4 May 2021; Accepted 27 August 2021
Available online 20 September 2021

In this paper, we present the first solution based on membrane computing to the motion planning problem in human environments. Membrane computing [5] is a branch of natural computing inspired in the structure and functions of the living cells. The computational devices in membrane computing are called membrane systems or P systems, and they have been successfully applied to a large variety of scientific areas: the modeling of microscopic and macroscopic biological systems [6,7]; and the study of neural models [8,9] incorporating fuzzy reasoning [10], among others. In particular, a variant of P systems called Enzymatic Numerical P systems (ENPS) has been used to simulate robot controllers [11,12] and to simulate robot navigation algorithms [13,14].

One of the most important features of any robot navigation system is the need to provide real-time solutions to complex situations. Therefore, most of the classical algorithms can only provide approximate solutions given a fixed time of computation. Several authors have studied how to accelerate such algorithms by using parallel hardware [15]. On the other hand, an intermediate computing paradigm can be used. In this sense, the membrane computing paradigm provides an inherently parallel computing framework with a large variety of simulators that can emulate computations over parallel hardware [16,17]. Thus, ENPS can be used to model solutions to robot navigation problems, and well-studied hardware simulators can apply the solutions to real case studies. The main advantage of this approximation is to use the computational power of membrane computing, which is inherently parallel, as well as widely-studied and robust simulation algorithms for parallel hardware. On the other hand, the main disadvantage is the complication of adapting such algorithms, requiring expert knowledge of both robotics and membrane computing.

The problem of motion planning can be divided in two sub-problems [18]: the global planning problem, in which only static obstacles are considered, and the local planning problem, in which the robot must follow a path while avoiding static and dynamic obstacles. If the dynamic obstacles are given by moving people, then it is a social local planning problem, since the robot should execute socially acceptable trajectories. In this paper, we provide a novel ENPS model for social local planning with a computational complexity of $\mathcal{O}(log(n))$ based on the Social Force Model (SFM) [19,20] and the Dynamic Window Approach Algorithm (DWA) [21]. The model is compatible with global planning ENPS models such as [14] and can be simulated in parallel hardware. A software simulator has been implemented for validation and testing purposes by using the Robot Operating System (ROS)[1] and C++. The simulator introduces in a virtual environment a number of human agents walking, following the SFM and a dual-wheeled non-holonomic robot, navigating according to the proposed ENPS model. Average velocities and distances to obstacles and other agents are measured for all the simulated agents to provide some results and discussion.

The rest of this work is structured as follows: Section 2 introduces the necessary preliminaries to understand the paper. Section 3 presents the proposed social local planning algorithm. Section 4 explains the corresponding ENPS model. Section 5 presents the simulation tool. Section 6 is devoted to analysis and discussion. Finally, Section 7 enumerates some conclusions and future work.

2. Preliminaries

This Section provides the reader with the basic concepts and notations used throughout this paper, from the description of the problem addressed in this study to the methods involved in the proposed solution.

2.1. Global and local planning

The problem of motion planning in robotics can be partially solved by a software architecture in three levels [18]: global planner, local planner and PID controller. Firstly, the global planner is the module in charge of solving the motion planning problem without considering dynamic obstacles (usually surrounding people). The output of the global planner is a trajectory or plan representing the path from the starting point to the goal area. This process is usually carried out by using a pre-computed map of static obstacles, *i.e.*, walls, furniture, and other fixed obstacles that have been previously tracked. The computation is done before the robot starts to move by applying algorithms such as variants of RRT [22,23], and RRT* [24], among others. After global planning has been performed, the local planner module generates motion commands to follow the trajectory in a safe manner, by considering the information given by the robot sensors in real time. The Dynamic Window Approach for Obstacle Avoidance Algorithm [21] and the Pure Pursuit Algorithm [25] are well-known algorithms for local planning. Finally, a PID controller module [26] manages the power of the motors to fit each motion command and maintain a constant velocity until the next command is processed.

2.2. Differential wheeled robots

Depending on the motion constraints imposed, two types of mobile robots can be distinguished: holonomic and nonholonomic. On the one hand, holonomic robots are those that can move in any direction in the configuration space. On the other hand, nonholonomic robots are systems where the velocities (magnitude and/or direction) and other derivatives of the position are constrained. In this paper, we consider the type of differential wheeled robots in which each wheel is driven independently; in particular, we focus on two-wheeled robots. A differential wheeled robot can be characterized in a 2D space by using a tuple (x, y, θ), known as its 2D pose, where x, y are the Cartesian coordinates of its center and θ is its yaw angle, *i.e.*, the orientation of its head. A differential wheeled robot is a nonholonomic robot, since the possible velocities are constrained for each value of θ; *e.g.*, it cannot move laterally along its axle. It should be noted that in this paper we also use poses in order to characterize the position and orientation of people surrounding the robot.

[1] https://www.ros.org/.

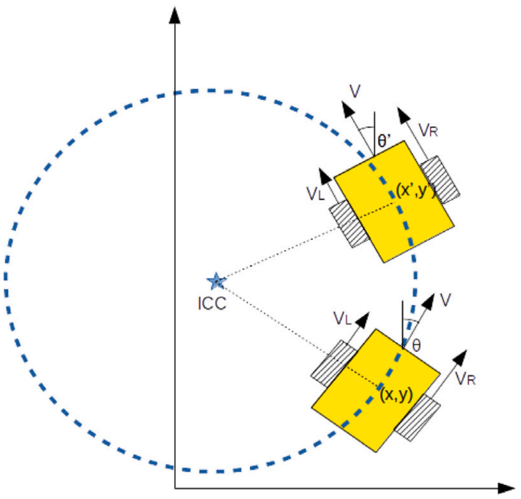

Fig. 1. A differential wheeled robot. The movement of the non-holonomic robot with speed V and angle θ is depicted. In this case, $V_R > V_L$ makes the robot rotate to its left side.

The possible trajectories for a differential wheeled robot can be expressed as circular trajectories passing through the center of the robot, as is shown in Fig. 1. Each trajectory is centered in a point called Instantaneous Center of Curvature (ICC). If the ICC is located at the center of the robot, the corresponding motion produces a contrary rotation of the wheels, which is an *in-place* rotation. On the other hand, if the ICC is located at the infinity along the robot perpendicular axis, then the wheels move at the same velocity, resulting in a straight-line motion.

The motion commands used for differential wheeled robots, *i.e.*, the commands generated by the local planner, are pairs of linear and angular velocities (v, ω), where v is the desired linear velocity of the robot (*i.e.*, the magnitude of its instant velocity vector) and ω is the desired angular velocity of the robot over the ICC. Therefore, the PID controller computes the individual linear velocities v_l and v_r for each wheel in order to fit v and ω by applying the kinematic equations for differential drive [27]. Finally, the PID powers the motors in an open loop to maintain a constant velocity until the next motion command is applied.

In this work, we make use of the kinematic equations to compute the future pose (x', y', θ') after applying a motion command (v, ω) from a starting pose (x, y, θ) for a time δ, based on the following calculations: $x' = x + v \cdot \delta \cdot cos(\theta + \frac{\omega \cdot \delta}{2})$, $y' = y + v \cdot \delta \cdot sin(\theta + \frac{\omega \cdot \delta}{2})$ and $\theta' = \theta + \omega \cdot \delta$.

Additionally, the resulting instant velocity vector $\mathbf{v}' = (v'_x, v'_y)$ is given by: $v'_x = v \cdot cos(\theta')$ and $v'_y = v \cdot sin(\theta')$

2.3. The dynamic window approach for obstacle avoidance algorithm

Given the pose of a differential wheeled robot, its current instant velocity vector and the map of surrounding obstacles, there are three types of possible circular trajectories:

1. The set of circular trajectories that cannot be executed in a time δ due to the robot velocity vector and its hardware acceleration limits. For example: A robot moving forward cannot begin to move backwards in a very short period of time. We consider these trajectories as *non-reachable*.
2. The set of reachable trajectories that can provoke a collision with surrounding obstacles, *i.e.*, when there are one or more obstacles in the trajectory curve and the robot cannot stop in a safe manner within time δ due to its velocity vector and hardware acceleration limits. We consider these trajectories as *non-admissible*.
3. The remaining circular trajectories are called *safe* trajectories. They can be evaluated according to some fitness function related to the robot's goal.

In [21], the problem of computing and selecting safe circular trajectories is addressed and the authors propose the *Dynamic Window Approach for Obstacle Avoidance Algorithm* (DWA), which is shown in Algorithm 1.

The algorithm uses a set of predefined circular trajectories C, *a.k.a.* motion commands. It should be noted that a motion command $(v, \omega) \equiv (0, 0)$ to stop the robot must be included in the set. Then, the algorithm assigns a fitness value for each safe circular trajectory and selects a trajectory with the minimum fitness value. The fitness function is user-defined; in [21], the authors use a weighted sum of features, such as the distance to obstacles in the trajectory, the velocity of the robot and the distance to the goal position.

Furthermore, in [21], the authors give the following definition for the subset of reachable trajectories in time δ:

$$C_r \equiv \{(v, \omega) \in C | v \in [v_a - \dot{v} \cdot \delta, v_a + \dot{v} \cdot \delta] \wedge \omega \in [\omega_a - \dot{\omega} \cdot \delta, \omega_a + \dot{\omega} \cdot \delta]\}$$

where (v_a, ω_a) are the current robot linear and angular velocities and $(\dot{v}, \dot{\omega})$ are the robot linear and angular acceleration limits.

Algorithm 1 DWA Algorithm

```
Require: Set of predefined circular trajectories $C \equiv \{(v_i, \omega_i) : 1 \le i \le M\}$; maximum robot linear and angular accelerations $(\dot{v}, \dot{\omega})$;
  trajectory execution time $\delta$; map of obstacles $\mathcal{O}$; robot position $\mathbf{r}$, yaw angle $\theta$, velocity vector $\mathbf{v}$.
  for $i$ from 1 to $M$ do
    Check if $(v_i, \omega_i)$ is reachable given $\mathbf{v}$, $\delta$, $\dot{v}$, $\dot{\omega}$.
    Check if $(v_i, \omega_i)$ is admissible given $\mathbf{r}$, $\theta$, $\dot{v}$, $\dot{\omega}$, $\mathcal{O}$.
    if $(v_i, \omega_i)$ is not reachable or $(v_i, \omega_i)$ is not admissible then
      Set the fitness value $f_i \leftarrow \infty$
    else
      Set the fitness value $f_i \leftarrow fitness(v_i, \omega_i, \delta, \mathbf{r}, \theta)$
    end if
  end for
  Execute a circular trajectory $(v, \omega) \in C$ with minimum fitness value.
```

Additionally, the authors provide the following definition for the subset of admissible trajectories:

$$C_a \equiv \{(v, \omega) \in C_r | v \le \sqrt{2 \cdot dist(v, \omega) \cdot \dot{v}} \wedge \omega \le \sqrt{2 \cdot dist(v, \omega) \cdot \dot{\omega}}\}$$

where $dist(v, \omega)$ is the Euclidean distance to the closest obstacle in the circular trajectory.

2.4. *The Social Force Model*

The Social Force Model (SFM) was introduced in [19] and widely extended in [28,29], among other works. The SFM studies the motion of pedestrians as particles acting under the influence of forces in the environment. These forces can be attractive or repulsive, being produced by people's intentions, static obstacles (walls, furniture, etc.) and dynamic obstacles (people moving in the environment). Attractive forces usually represent people's intentions to reach specific goals. On the other hand, repulsive forces model comfort distances between people in movement, as well as safety distances to obstacles during the execution of trajectories.

In this paper, we consider a Social Force Model based on [28,29], which can be summarized as follows: The instant velocity $\mathbf{v}(t)$ and position $\mathbf{r}(t)$ at time t for a pedestrian (*the subject* hereafter) trying to reach a desired goal $\mathbf{r}_{goal}$ as socially as possible are given by: $\mathbf{v}(t) = \mathbf{v}(t-\delta) + \delta \cdot \mathbf{F}(t)$ and $\mathbf{r}(t) = \mathbf{r}(t-\delta) + \delta \cdot \mathbf{v}(t)$, where $\mathbf{F}(t)$ is the instant force for the subject at time t. Considering unitary mass, such a force is equal to the acceleration. Following the SFM, $\mathbf{F}(t)$ is:

$$\mathbf{F}(t) = K_1 \cdot \mathbf{F}_{goal}(t) + K_2 \cdot \mathbf{F}_{social}(t) + K_3 \cdot \mathbf{F}_{obstacle}(t)$$

where K_1, K_2 and K_3 are the weights for the corresponding forces, and:

- $\mathbf{F}_{goal}(t) = \frac{v}{\tau}[\mathbf{e}_{goal}(t-\delta) - \mathbf{v}(t-\delta)]$ is the subject's attractive force to $\mathbf{r}_{goal}$ at time t, where v is a parameter called the *comfort subject's velocity*, τ is a parameter known as *relaxation time* and $\mathbf{e}_{goal}$ is the unitary vector from $\mathbf{r}(t-\delta)$ to $\mathbf{r}_{goal}$.
- $\mathbf{F}_{social}(t) = \sum_i^N \mathbf{f}_i^{social}(t)$ is the subject's repulsion force at time t produced by the surrounding pedestrians, where N is the number of pedestrians and $\mathbf{f}_i^{social}(t)$ is the repulsion force from pedestrian i at time t. The latter is defined as $\mathbf{f}_i^{social}(t) = -Ae^{-\frac{d_i}{B|\mathbf{d_i}|}}\left[e^{-(n'B|\mathbf{d_i}|\gamma_i)^2}\mathbf{t}_i + \eta_i e^{-(nB|\mathbf{d_i}|\gamma_i)^2}\mathbf{n}_i\right]$, where: A, B, n' and n are parameters; d_i is the distance between the subject and pedestrian i at time $t-\delta$; $\mathbf{t}_i$ is the interaction direction given by $\frac{\mathbf{d}_i}{|\mathbf{d}_i|}$, with $\mathbf{d}_i = \lambda[\mathbf{v}(t-\delta) - \mathbf{v}_i(t-\delta)] + \mathbf{e}_i$, where λ is a parameter, $\mathbf{v}_i(t-\delta)$ is the instant velocity of pedestrian i at time $t-\delta$ and $\mathbf{e}_i$ is the unitary vector from $\mathbf{r}(t-\delta)$ to $\mathbf{r}_i(t-\delta)$, with $\mathbf{r}_i(t-\delta)$ being the position of pedestrian i at time $t-\delta$; $\mathbf{n}_i$ is an orthonormal vector to $\mathbf{t}_i$; γ_i is the angle from $\mathbf{t}_i$ to $\mathbf{e}_i$; and η_i is the sign of γ_i (-1 if negative, 0 if null or 1 if positive).
- $\mathbf{F}_{obstacle}(t) = \frac{1}{M}\sum_i^M \mathbf{f}_i^{obstacle}(t)$ is the subject's repulsion force at time t from static obstacles (not people), where M is the number of surrounding obstacles and $\mathbf{f}_i^{obstacle}(t)$ is the repulsion force from obstacle i at time t, given by: $\mathbf{f}_i^{obstacle}(t) = ae^{-\frac{d_i}{b}}\mathbf{e}_i$, where a and b are parameters, d_i is the distance from the subject to obstacle i at time $t-\delta$ and $\mathbf{e}_i$ is the unitary vector from $\mathbf{r}(t-\delta)$ to the obstacle position.

2.5. *Enzymatic numerical P systems*

Membrane systems or, simply, *P systems* are computing models studied in Membrane Computing. They have an associated structure (given by a rooted tree or a directed graph), whose nodes (unit processors) are called *compartments* (*membranes*, *cells* or *neurons*). Compartments basically work with two types of elements: (a) multisets of symbols of a given (working) alphabet (called *objects*) and (b) some kinds of rules, generally inspired by nature (chemical reactions, neurons, dynamics of real ecosystems, etc.). Starting from an initial situation, the multisets of objects evolve according to fixed semantics for the rules. Membrane systems basically work with natural numbers through the multiplicity of objects in different compartments. Numerical P systems (NPS) are a special type of membrane systems where the concept of multisets of objects is replaced by *numerical variables* (real numbers), while the evolution rules are, in this case, *programs*. Numerical variables and programs are associated with compartments in a way that, from an initial situation (initial values associated with variables), they evolve dynamically, according to fixed semantics for programs.

There are membrane systems where the application of the evolution rules is controlled by certain elements, such as P systems with promoters and inhibitors. Inspired by this fact, *Enzymatic Numerical P systems* (ENPS) were introduced in [11] as an extension of numerical P systems in which some variables, named *enzymes*, control the execution of the programs. This computing model is being applied in different areas, especially in the simulation of control mechanisms of mobile and autonomous robots.

In this work, we have extended the original definition of ENPS introduced in [11] as follows:

Definition 2. An enzymatic numerical P system of degree $q \geq 1$ is a tuple $\Pi = (H, \mu, \{(Var_h, E_h, Pr_h, Var_h(0)) \mid h \in H\})$, where:

1. H is an alphabet of labels, containing q symbols;
2. μ is a rooted tree with q nodes bijectively labeled by elements from H;
3. Var_h, $h \in H$, is a finite set of numerical variables $x_{j,h}$ associated with the membrane labeled by h, with $1 \leq j \leq k_h$, $k_h = |Var_h|$;
4. E_h, $h \in H$, is a subset of variables $e_{t,h}$ from Var_h, called *enzymes*, with $1 \leq t \leq r_h$, $r_h = |E_h|$, $r_h < k_h$;
5. Pr_h, $h \in H$, is a finite set of programs associated with the membrane labeled by h, where each program $p \in Pr_h$ has the following form

$$F_p(x_{1,h}, \dots, x_{k_h,h})|_{Cond_p(e_{1,h},\dots,e_{r_h,h})} \longrightarrow c_{p,1}|v_{p,1}, \dots, c_{p,n_p}|v_{p,n_p}$$

where

- the left-hand side $F_p(x_{1,h}, \dots, x_{k_h,h})$ is a computable function, with $Var_h = \{x_{1,h}, \dots, x_{k_h,h}\}$;
- $Cond_p(e_{1,h}, \dots, e_{r_h,h})$ is a condition function $\mathbb{R}^{r_h} \to \{true, false\}$ using the set of enzymes $E_h = \{e_{1,h}, \dots, e_{r_h,h}\}$, in such a way that it disables the program when the output is $false$.
- the right hand side $c_{p,1}|v_{p,h}, \dots, c_{p,n_p}|v_{p,n_p}$ is an expression, with: $c_{p,1}, \dots, c_{p,n_p}$ being natural numbers, and $v_{p,1}, \dots, v_{p,n_p}$ numerical variables from membrane h, the parent membrane of h and all children of h according to μ.

6. $Var_h(0)$, $h \in H$, represents the initial values of the variables in Var_h.

An enzymatic numerical P system, $\Pi = (H, \mu, \{(Var_h, E_h, Pr_h, Var_h(0)) \mid h \in H\})$, of degree $q \geq 1$, can be viewed as a set of q membranes, labeled by elements of H, arranged in a hierarchical structure μ given by a rooted tree, whose root is called the skin membrane, thus the membrane labeled by h has: (a) a finite set Var_h of numerical variables $x_{j,h}$ (their initial values are defined by the set $Var_h(0)$), (b) a subset E_h of variables from Var_h, called enzymes, and (c) a finite set Pr_h of programs, where each program p has the following form $F_p(x_{1,h}, \dots, x_{k_h,h})|_{Cond_p(e_{1,h},\dots,e_{r_h,h})} \longrightarrow c_{p,1}|v_{p,1}, \dots, c_{p,n_p}|v_{p,n_p}$, where:

- $F_p(x_{1,h}, \dots, x_{k_h,h})$ is a computable function (called *production function* of the program), with $Var_h = \{x_{1,h}, \dots, x_{k_h,h}\}$;
- $Cond_p(e_{1,h}, \dots, e_{r_h,h})$ is a Boolean function $\mathbb{R}^{r_h} \to \{true, false\}$ (named *condition function*), with $E_h = \{e_{1,h}, \dots, e_{r_h,h}\}$.
- $c_{p,1}|v_{p,1}, \dots, c_{p,n_p}|v_{p,n_p}$ is the so-called *repartition protocol*, associated with the program, with $c_{p,1}, \dots, c_{p,n_p}$ being natural numbers specifying the proportion of the current production distributed to variables $v_{p,1}, \dots, v_{p,n_p} \in \left(Var_h \cup Var_{par(h)} \bigcup_{b \in ch(h)} Var_b\right)$, with $par(h)$ being the parent of h and $ch(h)$ the set of children of h in μ;

A *configuration* or *instantaneous description* of an ENPS at time $t \in \mathbb{N}$ is a tuple $(Var_1(t), \dots, Var_q(t))$, where $Var_h(t)$, for $h \in H$, provides the values of all numerical variables $x_{j,h}$ from the set Var_h at time t (the value of $x_{j,h}$ at time $t \in \mathbf{N}$ is denoted by $x_{j,h}(t)$). The *initial configuration* of Π is the tuple $(Var_1(0), \dots, Var_q(0))$.

A program $p \equiv F_p(x_{1,h}, \dots, x_{k_h,h})|_{Cond_p(e_{1,h},\dots,e_{r_h,h})} \longrightarrow c_{p,1}|v_{p,1}, \dots, c_{p,n_p}|v_{p,n_p}$ is executable (applicable) to a membrane labeled by h from configuration C_t if $Cond_p(e_{1,h}(t), \dots, e_{r_h,h}(t))$ is $true$. When such a program is executed to a membrane labeled by h from configuration C_t: (a) the value of function F_p at time t is computed: $F_p(x_{1,h}(t), \dots, x_{k_h,h}(t))$; (b) if we denote $C_p = \sum_{s=1}^{n_p} c_{p,s}$ then the contribution to the value of variable $v_{p,s}$, $1 \leq s \leq n_p$, at time $t+1$, is $q \cdot c_{p,s}$ value $q = \frac{F_p(x_{1,h}(t),\dots,x_{k_h,h}(t))}{C_p}$. Then the value $v_{p,s}(t+1)$ of the variable $v_{p,s}$ at time $t+1$ will be the sum of total contributions that $v_{p,s}$ receives from the neighboring membranes at time t. It is worth pointing out that the enzymes $e_{t,h}$ with $1 \leq t \leq r_h$, which enables the execution of programs, are variables from Var_h. Therefore, their values can change due to the contribution they receive from other programs and compartments.

Let us assume that there exists a global clock marking the time. In our semantics, all the programs that can be executed are selected in each time unit. The concept of *transition step* from a configuration C_t to C_{t+1} and the concept of *computation* for ENPS are defined in [11].

3. A social local planning algorithm for differential wheeled robots

In this section, we propose a social local planning algorithm for differential wheeled robots, using the SFM, presented in Section 2.4, to compute the robot's optimal instant velocity vector for each time step.

In an actual environment, not all the velocity vectors can be executed by a robot due to acceleration limits and collision threats, as explained in Section 2.3. Moreover, some velocity vectors cannot be executed due to nonholonomic constraints, as explained in Section 2.2. Therefore, the problem is to find a circular trajectory, *i.e.*, a motion command, approaching the optimal instant velocity vector. In our approximation, we apply the DWA algorithm to solve it. Thus, for each time step, we use a fixed set of motion commands and filter admissible and reachable commands with respect to acceleration limits and surrounding static/dynamic obstacles. Finally, a fitness function is computed for each filtered command, and one with the best fitness value is executed by the PID controller. The fitness function evaluates the difference between the corresponding circular trajectory and the optimal instant velocity vector, as explained below in this section. The computation is repeated in a loop until the robot's goal is reached or an error is produced, as shown in Algorithm 2.

Algorithm 2 A social local planning algorithm

```
Require: Set of predefined motion commands $C \equiv \{(\upsilon_i, \omega_i) : 1 \le i \le M\}$; maximum robot linear and angular accelerations $(\dot{\upsilon}, \dot{\omega})$;
  goal position $\mathbf{g} = (g_x, g_y)$; average loop period $\delta$.
  while $\mathbf{g}$ has not been reached by the robot and there is no error do
    Get the current robot position $\mathbf{r} \equiv (r_x, r_y)$.
    Get the current robot yaw angle $\theta$.
    Get the current robot velocity vector $\mathbf{v} \equiv (\upsilon_x, \upsilon_y)$.
    Get the current set of obstacle positions $\mathcal{O} \equiv \{(o_x, o_y)_i : 1 \le i \le N\}$.
    Get the current set of people positions $\mathcal{P} \equiv \{(p_x, p_y)_i : 1 \le i \le Q\}$.
    Get the current set of people velocities $\mathcal{V} \equiv \{(\upsilon_x, \upsilon_y)_i : 1 \le i \le Q\}$.
    Get the optimal robot velocity vector $\mathbf{v}' \leftarrow SFM(\mathbf{r}, \mathbf{v}, \mathbf{g}, \delta, \mathcal{O}, \mathcal{P}, \mathcal{V})$
    for $i$ from 1 to $M$ do
      Check if $(\upsilon_i, \omega_i)$ is reachable given $\mathbf{v}$, $\delta$, $\dot{\upsilon}$, $\dot{\omega}$.
      Check if $(\upsilon_i, \omega_i)$ is admissible given $\mathbf{r}$, $\theta$, $\dot{\upsilon}$, $\dot{\omega}$, $\mathcal{O}$, $\mathcal{P}$.
      Set the fitness value $f_i \leftarrow fitness(\upsilon_i, \omega_i, \delta, \mathbf{r}, \theta, \mathbf{v}')$
    end for
    Execute a motion command $(\upsilon, \omega) \in C$ with minimum fitness value.
  end while
```

M is the number of pre-defined motion commands. N is the number of static obstacles surrounding the robot and Q is the number of surrounding pedestrians. Function $SFM(\mathbf{r}, \mathbf{v}, \mathbf{g}, \delta, \mathcal{O}, \mathcal{P}, \mathcal{V})$ applies the calculation described in Section 2.4 to compute the robot's optimal instant velocity vector by applying the Social Force Model. Function $fitness(\upsilon_i, \omega_i, \delta, \mathbf{r}, \theta, \mathbf{v}')$ computes a fitness value for a motion command (υ_i, ω_i) given the robot pose $(\mathbf{r}, \theta)$, as well as time δ and the optimal instant velocity vector $\mathbf{v}'$ as reference. $fitness(\upsilon_i, \omega_i, \delta, \mathbf{r}, \theta, \mathbf{v}') = \infty$ if (υ_i, ω_i) is not reachable or (υ_i, ω_i) is not admissible. This fitness value is obtained in three steps:

1. Compute the theoretical pose of the robot (x', y', θ') after executing the circular trajectory described by (υ_i, ω_i) for time δ by applying the equations in Section 2.2.
2. Compute the theoretical pose of the robot (x'', y'', θ'') after following the optimal instant velocity vector $\mathbf{v}'$ as if the robot were holonomic, i.e., in a straight line for time δ.
3. Compute and return the fitness value as follows: $k_1[(x' - x'')^2 + (y' - y'')^2] + k_2|\theta' - \theta''|$

where k_1 and k_2 are parameters. More precisely: k_1 is the weight for the error in the robot's position and k_2 is the weight for the error in the robot's yaw angle. Such values can be set respectively to 1.0 and 0.1 by default and optimized or tuned by simulation or experimentation. Finally, it should be noted that δ must be low enough for a safe navigation in real time, e.g., 0.01 or 0.02 s.

4. An ENPS model for social local planning

In this section, a novel ENPS model $\Pi = (H, \mu, \{(Var_h, E_h, Pr_h, Var_h(0)) \mid h \in H\})$ for social local planning is presented, where $H \equiv \{1\}$ and, therefore $\mu \equiv [\ ]_1$. For the sake of simplicity, since there is only one membrane, we are going to omit the membrane label h in the numerical variables. Furthermore, a simplified syntax will be used for the programs in Pr_1:

$$F(x_1, \dots, x_n)|_{cond} \longrightarrow y$$

where $F(x_1, \dots, x_n)$ is a computable function; $x_1, \dots, x_n$ and y are numerical variables from Var_1; and, $cond$ is a Boolean condition using enzymes from E_1, which must be *true* in order to execute the program.

The set of numerical variables Var_1 is defined as follows:

$$\begin{aligned}
&Var_1 \equiv \{\theta, \delta, \upsilon, \tau, \lambda, A, B, n, n', a, b, e, K_1, K_2, K_3, \alpha, \alpha_0\} \cup \\
&\{r_i, \upsilon_i, \upsilon'_i, g_i, F_i, k_i : 1 \le i \le 2\} \cup \\
&\{\hat{\upsilon}_i, \hat{\omega}_i : 1 \le i \le M\} \cup \{o_{i,1}, o_{i,2}, d'_i, e'_{i,1}, e'_{i,2}, f'_{i,1}, f'_{i,2} : 1 \le i \le N\} \cup \\
&\{p_{i,1}, p_{i,2}, \upsilon_{i,1}, \upsilon_{i,2}, d_i, d_{i,1}, d_{i,2}, t_{i,1}, t_{i,2}, e_{i,1}, e_{i,2}, f_{i,1}, f_{i,2}, \gamma_i : 1 \le i \le Q\} \qquad (1)
\end{aligned}$$

where M is the number of pre-defined motion commands; N is the number of static obstacles surrounding the robot and Q is the number of surrounding pedestrians; $\upsilon, \tau, \lambda, A, B, n, n', a, b, K_1, K_2, K_3$ are the SFM parameters explained in Section 2.4; α and α_0 are enzymes for synchronization purposes; (r_1, r_2, θ) is the current robot's pose; (v_1, v_2) is the current robot's velocity vector; (v'_1, v'_2) is the optimal robot's velocity vector; (g_1, g_2) is the robot's goal position; (F_1, F_2) is the robot's force vector to the goal position; (k_1, k_2) are the weights for an user-defined fitness function (see Section 3); δ is the average period of execution; e is the Euler's number; $\{(\hat{v}_i, \hat{\omega}_i) : 1 \le i \le M\}$ is the set of pre-defined motion commands; $\{f_i : 1 \le i \le M\}$ is the fitness value for the pre-defined motion commands; $\{(o_{i,1}, o_{i,2}) : 1 \le i \le N\}$ is the set of positions of static obstacles; $\{d'_i : 1 \le i \le N\}$ is the set of distances from the robot to each static obstacle position; $\{(e'_{i,1}, e'_{i,2}) : 1 \le i \le N\}$ is the set of unitary vectors from the robot to each static obstacle position; $\{(f'_{i,1}, f'_{i,2}) : 1 \le i \le N\}$ is the set of forces from each obstacle position to the robot; $\{(p_{i,1}, p_{i,2}) : 1 \le i \le Q\}$ is the set of pedestrian positions; $\{(v_{i,1}, v_{i,2}) : 1 \le i \le Q\}$ is the set of velocity vectors of pedestrians; $\{d_i : 1 \le i \le Q\}$ is the set of distances from the robot to each pedestrian; $\{(d_{i,1}, d_{i,2}) : 1 \le i \le Q\}$ is the set of interaction vectors from the robot to each pedestrian; $\{(t_{i,1}, t_{i,2}) : 1 \le i \le Q\}$ is the set of interaction direction vectors from the robot to each pedestrian; $\{(e_{i,1}, e_{i,2}) : 1 \le i \le Q\}$ is the set of unitary vectors from the robot's position to each pedestrian position; $\{(f_{i,1}, f_{i,2}) : 1 \le i \le Q\}$ is the set of social forces from each pedestrian to the robot; $\{\gamma_i : 1 \le i \le Q\}$ is the set of interaction angles between the robot and the pedestrians.

The set of initial values is as follows:

$$\begin{aligned}
&Var_1(0) \equiv \{\theta(input), \delta(input), v(0.6), \tau(0.5), \lambda(2.0), A(1.5), B(0.35), n(2.0),\\
&n'(3.0), a(1.0), b(0.2), e(2.71828), K_1(2.0), K_2(2.1), K_3(10.0), \alpha(1), \alpha_0(0)\} \cup\\
&\{r_i(input), v_i(input), v'_i(0), g_i(user), F_i(0), k_i(user) : 1 \le i \le 2\} \cup\\
&\{\hat{v}_i(user), \hat{\omega}_i(user), f_i(0) : 1 \le i \le M\} \cup\\
&\{o_{i,1}(input), o_{i,2}(input), d'_i(0), e'_{i,1}(0), e'_{i,2}(0), f'_{i,1}(0), f'_{i,2}(0) : 1 \le i \le N\} \cup\\
&\{p_{i,1}(input), p_{i,2}(input), v_{i,1}(input), v_{i,2}(input), d_i(0), d_{i,1}(0), d_{i,2}(0),\\
&t_{i,1}(0), t_{i,2}(0), e_{i,1}(0), e_{i,2}(0), f_{i,1}(0), f_{i,2}(0), \gamma_i(0) : 1 \le i \le Q\}
\end{aligned} \tag{2}$$

where the word *input* means a value obtained by the robot's on-board sensors and the word *user* means an user-defined value.

The set of Enzymes is $E_1 \equiv \{\alpha, \alpha_0\}$ and the set of programs Pr_1 is as follows:

$$\begin{aligned}
p_1 &\equiv \tfrac{v}{\tau} \cdot \Big(\tfrac{g_1 - r_1}{\sqrt{(g_1-r_1)^2+(g_2-r_2)^2}} - v_1\Big)|_{\alpha=1} \longrightarrow F_1\\
p_2 &\equiv \tfrac{v}{\tau} \cdot \Big(\tfrac{g_2 - r_2}{\sqrt{(g_1-r_1)^2+(g_2-r_2)^2}} - v_2\Big)|_{\alpha=1} \longrightarrow F_2\\
p_{3,i} &\equiv \sqrt{(p_{i,1} - r_1)^2 + (p_{i,2} - r_2)^2}|_{\alpha=1} \longrightarrow d_i : 1 \le i \le Q\\
p_{4,i} &\equiv \sqrt{(o_{i,1} - r_1)^2 + (o_{i,2} - r_2)^2}|_{\alpha=1} \longrightarrow d'_i : 1 \le i \le N
\end{aligned}$$

In the first step of the computation, programs p_1 and p_2 are executed to generate the robot's force to the goal position. Programs $p_{3,i} : 1 \le i \le Q$ and $p_{4,i} : 1 \le i \le N$ are executed to generate, respectively, the Euclidean distances from the robot to each pedestrian position and from the robot to each obstacle position.

$$\begin{aligned}
p_{5,i} &\equiv \tfrac{r_1 - p_{i,1}}{d_i}|_{\alpha=2} \longrightarrow e_{i,1} : 1 \le i \le Q\\
p_{6,i} &\equiv \tfrac{r_2 - p_{i,2}}{d_i}|_{\alpha=2} \longrightarrow e_{i,2} : 1 \le i \le Q\\
p_{7,i} &\equiv \tfrac{r_1 - o_{i,1}}{d'_i}|_{\alpha=2} \longrightarrow e'_{i,1} : 1 \le i \le N\\
p_{8,i} &\equiv \tfrac{r_2 - o_{i,2}}{d'_i}|_{\alpha=2} \longrightarrow e'_{i,2} : 1 \le i \le N
\end{aligned}$$

In the second step of computation, programs $p_{5,i}, p_{6,i} : 1 \le i \le Q$ generate the unitary vectors from the robot's position to each pedestrian position. Programs $p_{7,i}, p_{8,i} : 1 \le i \le N$ produce the corresponding unitary vectors from the robot's position to each obstacle position.

$$\begin{aligned}
p_{9,i} &\equiv \lambda(v_1 - v_{i,1}) + e_{i,1}|_{\alpha=3} \longrightarrow d_{i,1} : 1 \le i \le Q\\
p_{10,i} &\equiv \lambda(v_2 - v_{i,2}) + e_{i,2}|_{\alpha=3} \longrightarrow d_{i,2} : 1 \le i \le Q\\
p_{11,i} &\equiv \tfrac{d_{i,1}}{\sqrt{d_{i,1}^2 + d_{i,2}^2}}|_{\alpha=4} \longrightarrow t_{i,1} : 1 \le i \le Q\\
p_{12,i} &\equiv \tfrac{d_{i,2}}{\sqrt{d_{i,1}^2 + d_{i,2}^2}}|_{\alpha=4} \longrightarrow t_{i,2} : 1 \le i \le Q
\end{aligned}$$

In the third and fourth steps of computation, programs $p_{9,i}, p_{10,i}, p_{11,i}, p_{12,i} : 1 \le i \le Q$ compute the interaction directions from the robot to each pedestrian.

$$p_{13,i} \equiv \operatorname{atan2}(e_{i,2}, e_{i,1}) - \operatorname{atan2}(t_{i,2}, t_{i,1})|_{\alpha=5} \longrightarrow \gamma_i : 1 \le i \le Q$$

In the fifth step of computation, programs $p_{13,i} : 1 \le i \le Q$ are executed to obtain the angles between the interaction directions and the unitary vectors from the robot to each pedestrian, by using the 2-argument arctangent (*atan2*).

$$p_{14,i} \equiv -Ae^{-\frac{d_i}{B'_i}}\left(e^{-(n'B'_i\gamma_i)^2} t_{i,1} - \eta_i e^{-(nB'_i\gamma_i)^2} t_{i,2}\right)|_{\alpha=6} \rightarrow f_{i,1} : 1 \le i \le Q$$

$$\begin{aligned}
&p_{15,i} \equiv -Ae^{-\frac{d_i}{B'_i}}(e^{-(n'B'_i\gamma_i)^2}t_{i,2} + \eta_i e^{-(nB'_i\gamma_i)^2}t_{i,1})|_{\alpha=6} \rightarrow f_{i,2} \, : \, 1 \le i \le Q \\
&p_{16,i} \equiv ae^{-\frac{d'_i}{b}}e'_{i,1}|_{\alpha=6} \longrightarrow f'_{i,1} \, : \, 1 \le i \le N \\
&p_{17,i} \equiv ae^{-\frac{d'_i}{b}}e'_{i,2}|_{\alpha=6} \longrightarrow f'_{i,2} \, : \, 1 \le i \le N \\
&p_{18} \equiv \max(\lfloor\log_2(N-1)\rfloor, \lfloor\log_2(Q-1)\rfloor)|_{\alpha=6} \longrightarrow \alpha_0
\end{aligned}$$

In the sixth step of computation, programs $p_{14,i}, p_{15,i} \, : \, 1 \le i \le Q$ compute the robot's social forces for each pedestrian, being $B'_i = B\sqrt{d^2_{i,1} + d^2_{i,2}}$ and η_i the sign of γ_i. Programs $p_{16,i}, p_{17,i} \, : \, 1 \le i \le N$ generate the robot's obstacle forces for each static obstacle. Finally, the mission of program p_{18} is to compute a temporal variable α_0 for synchronization purposes. Let $\hat{n} = \lfloor\log_2(N-1)\rfloor$ and $\hat{q} = \lfloor\log_2(Q-1)\rfloor$.

$$\begin{aligned}
&p_{19,i,j} \equiv f'_{i,1} + f'_{i+2^j,1}|_{\alpha=\hat{n}-j+7} \longrightarrow f'_{i,1} \, : \, 1 \le i \le 2^j, 0 \le j \le \hat{n} \\
&p_{20,i,j} \equiv f_{i,1} + f_{i+2^j,1}|_{\alpha=\hat{q}-j+7} \longrightarrow f_{i,1} \, : \, 1 \le i \le 2^j, 0 \le j \le \hat{q} \\
&p_{21,i,j} \equiv f'_{i,2} + f'_{i+2^j,2}|_{\alpha=\hat{n}-j+7} \longrightarrow f'_{i,2} \, : \, 1 \le i \le 2^j, 0 \le j \le \hat{n} \\
&p_{22,i,j} \equiv f_{i,2} + f_{i+2^j,2}|_{\alpha=\hat{q}-j+7} \longrightarrow f_{i,2} \, : \, 1 \le i \le 2^j, 0 \le j \le \hat{q}
\end{aligned}$$

Programs $p_{19,i,j}, p_{21,i,j} \, : \, 1 \le i \le 2^j, \lfloor\log_2(N-1)\rfloor \ge j \ge 0$ and programs $p_{20,i,j}, p_{22,i,j} \, : \, 1 \le i \le 2^j, \lfloor\log_2(Q-1)\rfloor \ge j \ge 0$ use a reduction procedure[2] to compute, respectively, the sum of social and obstacle forces in logarithmic time.

$$\begin{aligned}
&p_{23} \equiv v_1 + \delta(K_1F_1 + K_2f_{1,1} + \tfrac{K_3}{N}f'_{1,1})|_{\alpha=\alpha_0+8} \longrightarrow v'_1 \\
&p_{24} \equiv v_2 + \delta(K_1F_2 + K_2f_{1,2} + \tfrac{K_3}{N}f'_{1,2})|_{\alpha=\alpha_0+8} \longrightarrow v'_2
\end{aligned}$$

In the step of computation $\max(\lfloor\log_2(N-1)\rfloor, \lfloor\log_2(Q-1)\rfloor)+8$, the optimal velocity vector for the robot is computed by applying programs p_{23} and p_{24}

$$p_{25,i} \equiv \text{fitness}(\hat{v}_i, \hat{w}_i, \delta, k_1, k_2, r_1, r_2, \theta, v'_1, v'_2)|_{\alpha=\alpha_0+9} \longrightarrow f_i \, : \, 1 \le i \le M$$

In the step of computation $\max(\lfloor\log_2(N-1)\rfloor, \lfloor\log_2(Q-1)\rfloor) + 9$, a user-defined fitness function is computed for each motion command. Such a function should assign an infinite value to non-reachable and/or non-admissible commands, and other values otherwise.

$$\begin{aligned}
&p_{26,i,j} \equiv \min{}^*(\hat{v}_i, \hat{v}_{i+2^j}, f_i, f_{i+2^j})|_{\alpha=\lfloor\log_2(M-1)\rfloor-j+\alpha_0+10} \longrightarrow \hat{v}_i \, : \, 1 \le i \le 2^j, \lfloor\log_2(M-1)\rfloor \ge j \ge 0 \\
&p_{27,i,j} \equiv \min{}^*(\hat{\omega}_i, \hat{\omega}_{i+2^j}, f_i, f_{i+2^j})|_{\alpha=\lfloor\log_2(M-1)\rfloor-j+\alpha_0+10} \longrightarrow \hat{\omega}_i \, : \, 1 \le i \le 2^j, \lfloor\log_2(M-1)\rfloor \ge j \ge 0
\end{aligned}$$

The programs enumerated above are devoted to obtain a motion command with the minimum fitness value by applying a reduction procedure in logarithmic time by means of the function:

$$\min{}^*(a,b,c,d) = \begin{cases} a, & \text{if } c < d \\ b, & \text{if } c \ge d \end{cases}$$

$$p_{28} \equiv \alpha + 1|_{true} \longrightarrow \alpha$$

Finally, program p_{28} is included for synchronization purposes.

Therefore, after $\lfloor\log_2(M-1)\rfloor + \max(\lfloor\log_2(N-1)\rfloor, \lfloor\log_2(Q-1)\rfloor) + 10$ steps of computation, the best motion command is stored in $(\hat{v}_1, \hat{\omega}_1)$.

5. Simulator

A simulation tool was implemented in this work for validation and testing purposes. The Robot Operating System (ROS) framework was selected to develop the simulator. ROS is a flexible framework for writing robot software. It contains a wide variety of tools and libraries, which simplifies the task of creating robust software. ROS can be used together with Python and/or C++. In this work, the C++ programming language was selected. We adapted the BSD-licensed libraries LightSFM[3] and Pedlab[4] to simulate the human agents. The RVIZ tool[5] was used to generate a real-time 3D visualization. The simulator is GNU GPLv3 licensed and can be downloaded from https://github.com/RGNC/enps_dwa.

[2] A reduction procedure applies an associative binary operator to reduce the elements of a collection into a single value. This procedure can be parallelized in logarithmic scale.

[3] https://github.com/robotics-upo/lightsfm.

[4] https://github.com/robotics-upo/pedlab.

[5] http://wiki.ros.org/rviz.

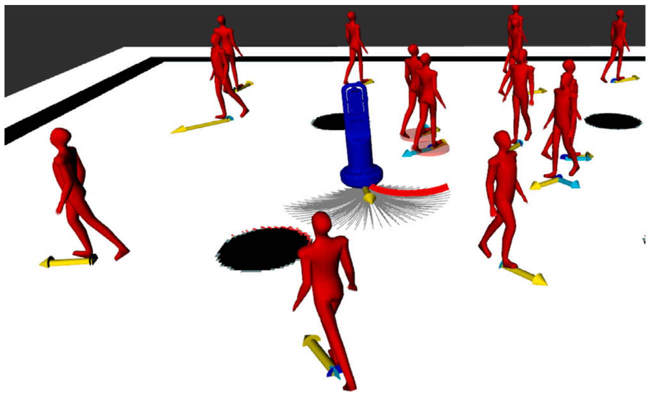

Fig. 2. Real-time 3D visualization showing each agent in the indoor environment. Yellow, dark-blue and light-blue arrows represent the global, obstacle and social forces for all the agents. The robot can only detect nearby humans which are marked with a semi-transparent red circle on the floor. The robot uses a 360-degree simulated LIDAR scanner to detect obstacles which are marked with small red dots. All possible motion commands, i.e., circular trajectories, are represented in front of the robot by using gray thin lines. The selected motion command for each time-instant is highlighted with a thick red line.

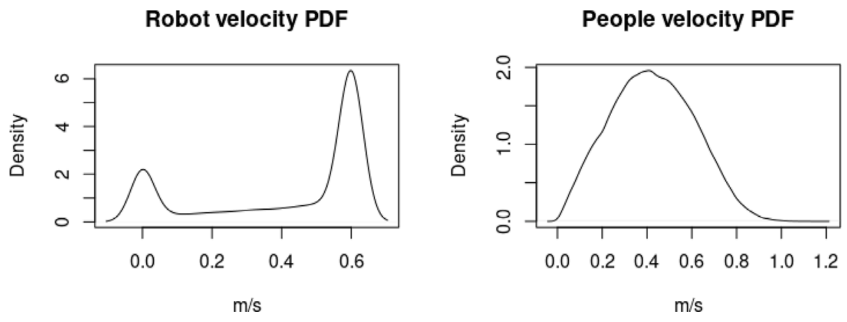

Fig. 3. Probabilistic Distribution Functions (PDF) of the robot and people velocities during the simulation. The robot tries to navigate at maximum velocity (0.6 m/s) most of the time. Several in-place rotations were produced by applying a linear velocity of 0 m/s along with an angular velocity different from 0 rad/s. The rest of the motion commands produce the remaining circular trajectories. Human agents have no velocity constraints and, therefore, they can follow a desired velocity vector regardless of its direction.

The software simulates an indoor environment with obstacles containing a configurable number of human agents walking according to a *social random walk*, i.e., each agent selects a nearby goal and walks approaching it by following the SFM explained in Section 2.4, when the goal is reached, another goal is selected. A dual-wheeled non-holonomic robot agent is also included, navigating according to the ENPS model proposed in Section 4. The robot's goals are randomly selected in the same way.

The ENPS model was simulated ad-hoc, i.e., the model definition was developed inside the simulator's source code. More precisely, the file simulator.hpp implements the behavior of the model.

A 3D visualization of the virtual world is generated in real-time by using RVIZ. A screenshot of the visualization is shown in Fig. 2. The visualization includes the environment, the agents, the force vectors, the simulated LIDAR scan, the people detections, the possible robot's motion commands, i.e., circular trajectories and the selected robot's trajectory at each time-instant.

The robot model is based on the TERESA robot, whose parameters are: *radius* of 0.3 m, *linear velocity* from 0 m/s to 0.6 m/s, *angular velocity* from −0.8 rad/s to 0.8 rad/s, *people detection range* (360°) of 2.5 m and *obstacle detection range* (360°) of 10 m. A more thorough explanation can be found in [30].

Regarding the computational complexity of the simulator, it is worth noting that it is greater than $\mathcal{O}(log(n))$, since the implementation is purely sequential for the sake of simplicity and according to the purposes of validation and testing.

6. Analysis

We run an experiment of 160 min by using the software presented in Section 5. Twenty human agents and 1 robot agent were included in the simulation. The robot and SFM parameters were set to the common parameters of the TERESA robot. A set of 33 possible angular velocities from -0.8 rad/s to 0.8 rad/s was combined with a set of 13 possible linear velocities from 0 m/s to 0.6 m/s, producing a whole set of 429 possible motion commands for the robot. The initial velocity of each human agent is randomly set by using a Gaussian distribution with $\mu = 1.2$ m/s and $\sigma = 0.001$.

For each agent and for each second of simulation, the software records in a file the instant velocity and the Euclidean distances to the corresponding nearest obstacle and nearest agent. Despite the natural differences between people and robot velocities shown

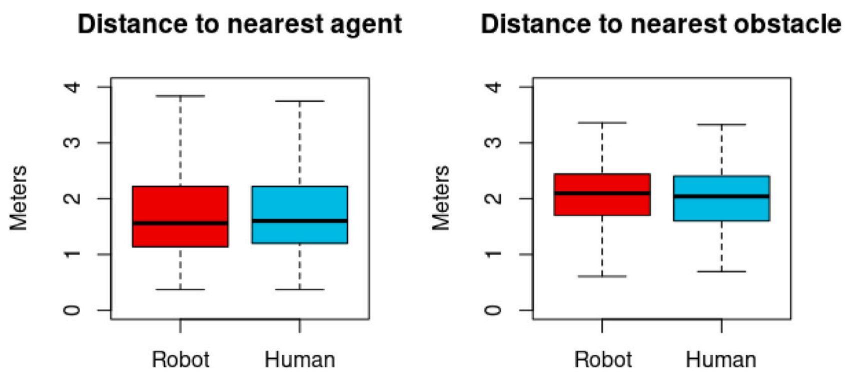

Fig. 4. The distances from the robot and from each human to their corresponding nearest obstacle and nearest agent were measured at each second of simulation. Such distances can be used as metric to compare the social behavior of the agents. Despite the differences in the velocity PDFs, a high similarity is observed in the average distances.

in Fig. 3, we observed a similar social behavior when considering the average distances to obstacles and other agents during the simulation, as shown in Fig. 4.

7. Conclusions and future work

This work presents the first model based on membrane computing for the social navigation of robots in human environments. Such a model is based on the ENPS framework, the Social Force Model and the Dynamic Window Approach Algorithm. It is carefully designed for differential-drive wheeled robots where nonholonomic and kinematic constraints are taken into account. The computational complexity of the model is $\mathcal{O}(log(n))$ and it is compatible with other ENPS models for the navigation of robots. The main advantage of using the ENPS framework is the natural way to expose parallelism in the algorithm and, therefore, enabling the possibility of using parallel hardware simulators. A simulation tool was implemented in this work for validation and testing purposes. The simulation was conducted in a 3D environment and metrics about velocities and distances were recorded. We run experiments with the software in order to compare the social behavior of the simulated human agents and the simulated robot. Despite the differences related to non-holonomic constraints and velocity limits, we observed a similar social behavior in the simulation with respect to average distances to nearby obstacles and agents. The main research line for future work is the simulation of the model by using parallel architectures in order to approach the $\mathcal{O}(log(n))$ computational complexity, as well as the integration of this model in the navigation workflow of actual robots.

CRediT authorship contribution statement

Ignacio Pérez-Hurtado: Software, Investigation, Writing - Original Draft. **David Orellana-Martín:** Conceptualization, Writing - Review & Editing. **Miguel Á. Martínez-del-Amor:** Validation, Writing - Review & Editing. **Luis Valencia-Cabrera:** Methodology, Writing - Review & Editing.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.compeleceng.2021.107408.

Acknowledgments

This work was supported by “FEDER/Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación/ _Proyecto (TIN2017-89842-P)” - MABICAP.

The software developed in this work adapts the LightSFM and Pedlab libraries developed by the Service Robotics Lab (Pablo de Olavide University) under the terms of the BSD free-software license. The authors acknowledge the Service Robotics Lab (Pablo de Olavide University) for this software.

References

[1] Latombe J. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. Int J Robot Res 1999;18(11):1119–28.
[2] Ríos-Martínez J, Spalanzani A, Laugier C. From proxemics theory to socially-aware navigation: A survey. Int J Soc Robot 2014;7:137–53.
[3] Perc M, Ozer M, Hojnik J. Social and juristic challenges of artificial intelligence. Palgrave Commun 2019;2019(61).
[4] Helbing D, Brockmann D, Chadefaux T, Donnay K, Blanke U, Woolley-Meza O, et al. Saving human lives: What complexity science and information systems can contribute. J Stat Phys 2015;158:735–81.
[5] Păun G. Computing with membranes. J Comput System Sci 2000;61(1):108–43.
[6] Gheorghe M, Krasnogor N, Camara M. P systems applications to systems biology. Biosystems 2008;91(3):435–7.
[7] Colomer MA, Margalida A, Pérez-Jiménez MJ. Population dynamics P system (PDP) models: A standardized protocol for describing and applying novel bio-inspired computing tools. PLoS One 2013;8(5):e60698.
[8] Peng H, Li B, Wang J, Song X, Wang T, Valencia-Cabrera L, et al. Spiking neural P systems with inhibitory rules. Knowl-Based Syst 2020;188:105064.
[9] Orellana-Martín D, Martínez-del-Amor MA, Valencia-Cabrera L, Pérez-Hurtado I, nez AR-N, Pérez-Jiménez MJ. Dendrite P systems toolbox: Representation, algorithms and simulators. Int J Neural Syst 2021;31(1).
[10] Wang T, Zhang G, Zhao J, He Z, Wang J, Pérez-Jiménez MJ. Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. IEEE Trans Power Syst 2015;30(3):1182–94.
[11] Pavel A, Arsene O, Buiu C. Enzymatic numerical P Systems - a new class of membrane computing systems. In: 2010 IEEE fifth international conference on bio-inspired computing: theories and applications. 2010, p. 1331–6.
[12] Florea A, Buiu C. Membrane Computing for Distributed Control of Robotic Swarms: Emerging Research and Opportunities. IGI Global; 2017, p. 1–119.
[13] Pérez-Hurtado I, Pérez-Jiménez MJ, Zhang G, Orellana-Martín D. Simulation of rapidly-exploring random trees in membrane computing with P-lingua and automatic programming. Int J Comput Commun Control 2019;13(6):1007–31.
[14] Pérez-Hurtado I, Martínez-del-Amor MA, Zhang G, Neri F, Pérez-Jiménez MJ. A membrane parallel rapidly-exploring random tree algorithm for robotic motion planning. Integr Comput-Aided Eng 2020;27:1–18.
[15] Bialkowski J, Karaman S, Frazzoli E. Massively parallelizing the RRT and the RRT*. In: Proceedings of the 2011 IEEE/RSJ international conference on intelligent robots and systems. 2011, p. 3513–8.
[16] Martínez-del-Amor MA, Pérez-Hurtado I, Orellana-Martín D, Pérez-Jiménez MJ. Adaptative parallel simulators for bioinspired computing models. Future Gener Comput Syst 2020;107:469–84.
[17] Zhang G, Shang Z, Verlan S, Martínez-del-Amor MA, Yuan C, Valencia-Cabrera L, et al. An overview of hardware implementation of membrane computing models. ACM Comput Surv 2020;53(4).
[18] Mahtani A, Sánchez L, Fernández E, Martínez A, Joseph L. ROS programming: building powerful robots. Packt Publishing; 2018.
[19] Helbing D, Molnar P. Social force model for pedestrian dynamics. Phys Rev E 1998;51. http://dx.doi.org/10.1103/PhysRevE.51.4282.
[20] Ferrer G, Sanfeliu A. Proactive kinodynamic planning using the extended social force model and human motion prediction in urban environments. In: IEEE international conference on intelligent robots and systems. In: 2014 IEEE/RSJ international conference on intelligent robots and systems. 2014, p. 1730–5.
[21] Fox D, Burgard W, Thrun S. The dynamic window approach to collision avoidance. IEEE Robot Autom Mag 1997;4(1):23–33.
[22] LaValle SM. Rapidly-exploring random trees: a new tool for path planning. Tech. rep., Iowa State University; 1998.
[23] LaValle SM, Kuffner JJ. Randomized kinodynamic planning. Int J Robot Res 2001;20(5):378–400.
[24] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning. Int J Robot Res 2011;30(7):846–94.
[25] Amidi O. Integrated mobile robot control. Tech. rep., Carnegie Mellon Univ. Robotics Institute; 1990.
[26] Astrom K, Hagglund T. PID controllers: theory, design and tuning. ISA; 1995.
[27] Dudek G, Jenkin M. Computational principles of mobile robotics. 2nd ed. USA: Cambridge University Press; 2010.
[28] Moussaïd M, Helbing D, Garnier S, Johansson A, Combe M, Theraulaz G. Experimental study of the behavioural mechanisms underlying self-organization in human crowds. Proc Biol Sci R Soc 2009;276:2755–62. http://dx.doi.org/10.1098/rspb.2009.0405.
[29] Moussaïd M, Perozo N, Garnier S, Helbing D, Theraulaz G. The walking behaviour of pedestrian social groups and its impact on crowd dynamics. PLoS One 2010;5:e10047. http://dx.doi.org/10.1371/journal.pone.0010047.
[30] Shiarlis K, et al. TERESA: A socially intelligent semi-autonomous telepresence system. In: Workshop on machine learning for social robotics, IEEE international conference on robotics and automation. 2015, p. 1–6.

Ignacio Pérez-Hurtado received his B.Sc. in 2003, and afterwards he worked for an IT company. From 2007 to 2013 he worked at the RGNC (University of Seville) receiving his Ph.D. in 2010. From 2014 to 2017 he was a postdoctoral fellow at Pablo Olavide University (Spain), researching in Social Robotics. He is currently an Assistant Professor at the University of Seville.

David Orellana-Martín received his B.Sc. in 2014, his M.Sc. in 2016 and his PhD in 2019 at the University of Seville. He received the best PhD thesis of 2019 Award from the International Membrane Computing Society. He was an ERCIM fellow at NTNU (Norway) from 2019 to 2021, and nowadays he is a postdoctoral fellow at the University of Seville.

Miguel A. Martínez-del-Amor obtained his M.Sc. in Computer Engineering at the University of Murcia (2008), and his Ph.D. in Computer Science and Artificial Intelligence at the University of Seville (2013). He was a research associate at Fraunhofer IIS (Germany) from 2014 to 2017. He is an Assistant Professor at the University of Seville since 2017, and an NVIDIA DLI ambassador since 2018.

Luis Valencia-Cabrera received his B.Sc. (2005), worked for an IT company (2005–2010), got SCJP, SCWCD, SCBCD (Oracle), CFPS (IFPUG), professional M.Sc. (2007), academic MSc (2013), and Ph.D. (2015, Ph.D. Extraordinary award). He is an Assistant Professor (University of Seville), supervised many M.Sc./B.Sc., 2 doctoral theses, and has > 100 scientific contributions (~ 40 in ISI-JCR journals).