Abstract—Methods based on Rapidly-exploring Random Trees (RRTs) have been in use in robotics to solve motion planning problems for nearly two decades. On the other hand, in the membrane computing framework, models based on Enzymatic Numerical P systems (ENPS) have been applied to robot controllers. These controllers handle the power of motors according to motion commands usually generated by planning algorithms, but today there is a lack of planning algorithms based on membrane computing for robotics. With this motivation, we provide a new variant of ENPS called Random Enzymatic Numerical P systems with Proteins and Shared Memory (RENPSM) addressed to implement RRT algorithms and we illustrate it by presenting a model for path planning of mobile robots based on the bidirectional RRT algorithm. A software for RENPSM has been developed within the Robot Operating System (ROS) and simulation experiments have been conducted by means of the Pioneer 3-DX robot simulation platform.

I. INTRODUCTION

Robots are machines oriented to objectives equipped with actuators, sensors and computation units acting under physical constraints. Regardless of their morphology, they should accomplish tasks by acting in the real world. This is one of the main reasons by which robot motion planning [8] is an eminent research area in robotics. In general terms, the problem of motion planning can be defined in the configuration space of a robot as follows: Given a start configuration state, a goal configuration state, a geometric description of the robot, and a geometric description of the environment, find a path that moves the robot gradually from start to goal.

A configuration state is a specification of the positions of all robot points relative to a fixed coordinate system. This is usually expressed as a vector of positions and orientations, for example, a rigid-body robot in a 2D world can be expressed as a vector \((x, y, \theta)\) representing the center \((x, y)\) of the robot in a fixed coordinate system and its yaw angle \(\theta\), i.e., the heading angle of the robot. Since the shape of the robot is described, all of its points are then known.

Several constraints can be added to this problem, the most common is to reach the goal while never touching any obstacle in the environment. Others can also be added, for example, a social robot could restrict configuration states in order to guarantee the human comfort.

The configuration space of a robot can also be constrained by the type of movements the robot can perform. In this sense, nonholonomic robots are those that cannot instantly modify its direction without employing rotation in-place. On the other hand, holonomic robots can do it (assuming zero mass). For example, a holonomic robot in a 2D world can move along the \(x\) axis and the \(y\) axis, as well as modify its yaw angle if needed. But a nonholonomic robot can only move forward/backward and/or modify its yaw angle. This is the typical case of dual-wheeled mobile robots and cars.

Classical path planning algorithms have been widely adapted and applied to the problem of motion planning with constraints in robots, for example, in [21], an application of the Dijkstra algorithm for robot path-planning was presented. In such solutions, the general problem is usually divided into two smaller problems: the global path planning problem, as described above; and the local path planning, where the robot tries to connect two consecutive states in real-time considering features not included in the global plan as, for example, dynamic obstacles. The accumulated error during the local planning conducts to periodically recompute the global plan. For this reason, the computational complexity of global planners is a critical point regarding to real-time constraints. Many efforts have been made to provide good global planners. For example: in [20], a search algorithm, called \(D^*\), was presented for path planning in real-time environments. In [11], a variant of the classical search algorithm \(A^*\) is applied to grids with blocked and unblocked cells. In [9], a tool for global path planning, called Rapidly-Exploring Random Trees (RRT), was presented.

The class of RRT algorithms for global path planning is based on the randomized exploration of the configuration space before moving the robot by building a tree in memory where nodes represent states that can be reached by the state of the corresponding parent in a fixed amount of time, furthermore each edge contains a velocity reference to reach the state in the child node from the state in the parent node. It is currently one popular method in robot motion planning due to its good properties. The computed RRT can be used together with search algorithms or, as presented in [10], the
RRT generation algorithm can be used by itself as a path planning algorithm, where two RRTs are built simultaneously, one beginning from the initial configuration and another one beginning from the ending configuration (bidirectional RRT).

In order to follow the path in a safe manner, a local planner module should be executed considering dynamic obstacles. Finally, each motor of the robot must be able to reach and maintain velocity references for fixed periods of time. This is the function of a type of software called controller on-board of the robot. Thus, robot control [1] is the branch of robotics dedicated to the study and practice of controlling robots.

Robot controllers are usually based on common silicon microprocessors, but in the recent years, some classes of membrane systems [12] have been in use for modelling them [14] [15] [16] [23]. Membrane systems are models of computation based on the structure and functions of the living cells. In a membrane system, there are objects being evolved inside compartments according to rules applied in a non-deterministic, maximally parallel way. They have been used as a new technique to attack the P versus NP problem [17], and several applications have been also studied: stochastic P systems for modelling biological phenomena [19], probabilistic P systems for modelling real ecosystems [2], spiking neural P systems incorporating fuzzy reasoning, for fault diagnosis and learning [22], and others.

With respect to robot control, numerical P systems (NPS) were used for modelling and simulating robot controllers [16], although the initial application of NPS was related to economical processes [13]. A variant called enzymatic numerical P systems (ENPS) [14] was introduced and applied to the distributed control of a swarm of mobile robots. Indeed, reactive and proportional-integral-derivative (PID) dual-wheeled robot controllers have been successfully designed and simulated by means of ENPS, as well as software simulation tools [23]. This variant has been also used [15] to address robot localization problem [5], where the robot must know its position in the environment by using sensors.

In this paper, following [18], a new variant of ENPS called random enzymatic numerical P systems with proteins and shared memory (RENPSM, for short) is introduced in order to get a new implementation of RRT algorithms. Since the current applications of membrane systems to robotics are focused on controllers, we propose an approximation from “the other side”: global path planning with physical constraints. The idea is to provide a complementary approach walking towards a mobile dual-wheeled robot system based completely on membrane systems. The new syntactical ingredients are related to fit the requirements of the RRT algorithm:

- **Random numbers**: The algorithm uses a randomized method to explore the physical space, therefore random numbers must be generated.
- **Shared memory**: The algorithm is parallelized using processes sharing common variables, and a distinguished membrane, called shared memory, is included. At any instant, each membrane can read from or write to it.
- **Proteins**: In order to synchronize the sequential execution of the algorithm, proteins are used.

This paper is structured as follows. In the next section, some notions about robot path planning are introduced. In Section III the rapidly-exploring random trees (RRTs) are described with some details. Section IV is devoted to introduce random enzymatic numerical P systems with proteins and shared memory. In Section V, a RENPSM model for the bidirectional RRT algorithm is described. In Section VI, a software for simulating the model within the ROS framework is presented with some experimental results in a simulation environment. Finally, conclusions and future work are drawn.

II. ROBOT PATH PLANNING

In general terms, robot path planning can be solved by applying a solution based on three modules:

- **Global planner**: It receives the desired ending configuration of the robot, its safety radius and current localization, as well as the precomputed position of the static obstacles in the environment and current information of sensors and odometry. The current dynamic obstacles detected by the sensors are added to the static ones in order to generate a more descriptive information of the environment. The odometry is used to obtain the current velocity of the robot when kinodynamic constraints are considered. Then, the global planner computes a plan from the starting configuration $x_{init}$ to the desired final configuration $x_{end}$ of the robot. The plan is represented as a sequence of local goals $\{g_i|1 \leq i \leq n\}$, where $g_i = x_{init}$ and $g_n = x_{end}$. Each goal can be reachable from the previous goal considering the constraints of the problem, i.e., avoiding static obstacles, nonholonomic and kinodynamic constraints, etc. RRT algorithms and other similar algorithms can be used for this task.
- **Local planner**: It receives the sequence of local goals generated by the global planner, as well as the current information of sensors, localization and odometry, then it sends velocity references to the controller in order to command the robot along the path. Several algorithms such as the dynamic window approach [4], pure pursuit [3], and potential fields [7] algorithms, among others and variants, can be used.
- **Controller**: It receives velocity references from the local planner and manages the power of the motors to fit each reference and maintain it constant until the next one.

In Figure 1, it is represented the general robot path planning cycle. First the robot computes a global plan from its current pose to the desired ending pose; if the plan can be generated, i.e., the robot could reach the destination considering all the constraints, then the local planner receives a sequence of intermediate goals and sends velocity references to the PID controller in order to follow the path in a safe manner until reaching the destination; if some error occurs, for example, a dynamic obstacle is too close to the next local goal or the robot is too far from the next local goal (considering a predefined
threshold), then the global plan is recomputed from the current robot position. Notice that if there is a dynamic obstacle too close to the ending configuration, then the global plan cannot be found.

III. RAPIDLY-EXPLORING RANDOM TREES

An RRT [9] is a randomized tree structure for rapidly exploring in memory a state space $X$ from an initial state $x_{\text{init}}$. It can be successfully used for nonholonomic and kinodynamic global path planning in robotics [10].

Nodes in an RRT represent possible reachable states, for mobile robots in a 2D world this is given by $(x, y, \theta)$ where $(x, y)$ are the Cartesian coordinates of the robot position and $\theta$ is the heading angle. However, the heading angle can be omitted in order to reduce the size of the tree.

It is assumed that a fixed obstacle region $X_{\text{obs}} \subseteq X$ must be avoided, so the nodes of the RRT are states in $X_{\text{free}}$, the complement of $X_{\text{obs}}$ in $X$.

Edges in an RRT represent transitions between reachable states, each one is labelled with the velocity reference $u$ that the robot should execute for a fixed period of time $\Delta t$ in order to change the corresponding states. For a mobile robot in a 2D world, the velocity reference can be represented by the pair of linear and angular velocities $(v, \omega)$ to be sent to the controller. On the other hand, if $\theta$ has been omitted, the edges in the RRT are labelled with instant linear velocities. Thus, a holonomic robot can reach a state $x_1$ from another state $x_0$ connected by an edge labelled with $u$ by applying $x_1 = x_0 + u \cdot \Delta t$. In the case of nonholonomic robots, the local planner should select the best sequence of motions in order to approximate $x_1$, for example, if the robot can rotate in-place, a naive solution is to perform a rotation in-place before developing the motion in a straight line.

Algorithm 1 GENERATE_RRT

Require: $x_{\text{init}}, K, \rho, \Delta t, X_{\text{obs}}, d_{\text{min}}$

$V_\tau \leftarrow \{x_{\text{init}}\}; E_\tau \leftarrow \emptyset$

for $k = 1$ to $K$ do
  $x_{\text{rand}} \leftarrow \text{RANDOM_STATE}(X)$;
  $\text{EXTEND}(\tau, x_{\text{rand}}, \rho, \Delta t, X_{\text{obs}}, d_{\text{min}})$;
end for

return $\tau = (V_\tau, E_\tau)$;

Algorithm 2 EXTEND

Require: $\tau, x, \rho, \Delta t, X_{\text{obs}}, d_{\text{min}}$

$x_{\text{near}} \leftarrow \text{NEAREST_NEIGHBOR}(x, \tau)$;

if $\text{DISTANCE}(x, x_{\text{near}}) \geq d_{\text{min}}$ then
  $u \leftarrow \text{SELECT_INPUT}(x, x_{\text{near}})$;
  if $\neg \text{COLLISION}(x_{\text{near}}, u, \Delta t, X_{\text{obs}})$ then
    $x_{\text{new}} \leftarrow \text{NEW_STATE}(x_{\text{near}}, u, \Delta t)$;
    $V_\tau \leftarrow V_\tau \cup \{x_{\text{new}}\}$
    $E_\tau \leftarrow E_\tau \cup \{[x_{\text{near}}, x_{\text{new}}]\}$
  if $\text{DISTANCE}(x, x_{\text{new}}) < d_{\text{min}}$ then
    return Reached;
  else
    return Advanced;
  end if
else
  if $\text{COLLISION}(x_{\text{near}}, u, \Delta t, X_{\text{obs}})$ then
    return Trapped;
  elseif
    return Trapped;
  else
    return Trapped;
  end if
end if

Algorithm 1 is an iterative method to generate an RRT using the function EXTEND defined in Algorithm 2, where:

- $x_{\text{init}}$ is the initial state.
- $K$ is the number of iterations to build the RRT.
- $\rho$ is a prefixed distance metric.
- $\Delta t$ is a fixed amount of time for transitions.
- $X$ is the state space.
- $X_{\text{obs}}$ is the obstacle state space.
- $d_{\text{min}}$ is the minimum distance threshold according to $\rho$ in order to include a new node in the RRT.
- $\tau = (V_\tau, E_\tau)$ is the RRT generated.
- RANDOM_STATE$(X)$ is a function to get a random state from $X$.
- NEAREST_NEIGHBOR$(x, \tau)$ is a function to get the closest node to $x$ in $\tau$ according to $\rho$.
- DISTANCE$(x, x_{\text{near}})$ is a function to get the distance of $x$ to $x_{\text{near}}$ according to $\rho$.
- SELECT_INPUT$(x, x_{\text{near}})$ is a function to get the velocity input that should be commanded to the robot in order to achieve state $x$ from $x_{\text{near}}$. 

![Fig. 1. Robot path planning cycle](image-url)
• COLLISION($x_{near}$, $u$, $\Delta t$, $X_{obs}$) is a function returning true if a collision could be produced moving the robot from state $x_{near}$ by applying the input $u$ for $\Delta t$ time considering the obstacles in $X_{obs}$.

• NEW_STATE($x_{near}$, $u$, $\Delta t$) is a function to get a new state $x_{new}$ by applying the input $u$ to the robot for $\Delta t$ time starting at state $x_{near}$.

The function EXTEND tries to add a new node to the RRT considering a reference $x$. If the function fails, then it returns Trapped; if the new node is closer than $d_{min}$ to $x$, then it returns Reached; and if the new node is far from $x$ considering $d_{min}$, then the function returns Advanced. Figure 2 describes a RRT generated after 5000 iterations by using Algorithm 1 with the Euclidean distance and omitting the heading angle.

In [10], a bidirectional RRT algorithm is introduced for path planning. The main idea of this algorithm is to create two RRTs: $\tau_a$ starting at $x_{init}$ and $\tau_b$ starting at $x_{end}$. If $\tau_a$ and $\tau_b$ are connected in a prefixed number $K$ of iterations, then a path is returned; otherwise the function returns failure. Algorithm 3 is the bidirectional RRT algorithm presented in [10], where:

• $x_{init}$ is the initial state.
• $x_{end}$ is the ending state.
• $\tau_a = (V_{\tau_a}, E_{\tau_a})$ is an RRT starting at $x_{init}$.
• $\tau_b = (V_{\tau_b}, E_{\tau_b})$ is an RRT starting at $x_{end}$.
• PATH($\tau_a$, $\tau_b$) is a function to compute a path from the initial node of $\tau_a$ to the initial node of $\tau_b$. Both RRTs must be connected.
• SWAP($\tau_a$, $\tau_b$) is a procedure to interchange the values of $\tau_a$ and $\tau_b$.

The rest of variables have the same meaning than the variables used in Algorithms 1 and 2.

In this paper we propose the Algorithm 4 as a parallel version of the bidirectional RRT algorithm, where $\tau_a$ and $\tau_b$ are built at the same time.

### Algorithm 3 GENERATE_PATH

**Require:** $x_{init}$, $x_{end}$, $K$, $\rho$, $\Delta t$, $X$, $X_{obs}$, $d_{min}$

```plaintext
V_{\tau_a} \leftarrow \{x_{init}\}; E_{\tau_a} \leftarrow \emptyset;
V_{\tau_b} \leftarrow \{x_{end}\}; E_{\tau_b} \leftarrow \emptyset;

for k = 1 to K do
  x_{rand} \leftarrow RANDOM_STATE(X);
  if EXTEND($\tau_a$, $x_{rand}$, $\rho$, $\Delta t$, $X_{obs}$, $d_{min}$) \neq Trapped then
    if EXTEND($\tau_b$, $x_{new}$, $\rho$, $\Delta t$, $X_{obs}$, $d_{min}$) = Reached then
      return PATH($\tau_a$, $\tau_b$);
    end if
  end if
  SWAP($\tau_a$, $\tau_b$);
end for
return Failure
```

### Algorithm 4 GENERATE_PATH_PARALLEL

**Require:** ($x_{init}$, $x_{end}$, $K$, $\rho$, $\Delta t$, $X$, $X_{obs}$, $d_{min}$)

```plaintext
V_{\tau_a} \leftarrow \{x_{init}\}; E_{\tau_a} \leftarrow \emptyset;
V_{\tau_b} \leftarrow \{x_{end}\}; E_{\tau_b} \leftarrow \emptyset;

for k = 1 to K do
  x_{rand,a} \leftarrow RANDOM_STATE(X);
  x_{rand,b} \leftarrow RANDOM_STATE(X);
  begin parallel block
    result_a = EXTEND($\tau_a$, $x_{rand,a}$, $\rho$, $\Delta t$, $X_{obs}$, $d_{min}$);
    result_b = EXTEND($\tau_b$, $x_{rand,b}$, $\rho$, $\Delta t$, $X_{obs}$, $d_{min}$);
  end parallel block
  if result_a \neq Trapped then
    if EXTEND($\tau_b$, $x_{new}$, $\rho$, $\Delta t$, $X_{obs}$, $d_{min}$) = Reached then
      return PATH($\tau_a$, $\tau_b$);
    end if
  end if
  if result_b \neq Trapped then
    if EXTEND($\tau_a$, $x_{new}$, $\rho$, $\Delta t$, $X_{obs}$, $d_{min}$) = Reached then
      return PATH($\tau_a$, $\tau_b$);
    end if
  end if
end for
return Failure
```

### IV. RANDOM ENZYMATIC NUMERICAL P SYSTEMS WITH PROTEINS AND SHARED MEMORY

In this section a variant of enzymatic numerical P systems incorporating new features is presented, in order to simulate RRT algorithms.

A random enzymatic numerical P systems with proteins and shared memory (RENPSM, for short) of degree $(p,q)$, $p, q \geq 1$ is a tuple $(H, \mu, P, E_{mem}, E_{mem}(0), \{P_h(0), Var_h, Var_h(0), Pr_h\} | h \in H), R, h_a, h_b)$, where:

1. $H = \{1, \ldots, p \cdot q\} \cup \{v, mem\}$, $mem \notin \{1, \ldots, p \cdot q\}$, $v \notin \{mem, 1, \ldots, p \cdot q\}$, is the set of labels of the system;
2. \( \mu \) is a dynamic membrane structure (rooted tree) initially consisting of one skin membrane with label \( v \) including two inner membranes labelled respectively with \( h_a \in \{1, \ldots, p \cdot q \} \) and \( h_b \in \{1, \ldots, p \cdot q \} \), \( h_a \neq h_b \), in such manner that along the computation only child membranes of \( h_a \) and \( h_b \) will be created with labels in \( \{1, \ldots, p \cdot q \} \). In Figure 3, it is represented the initial membrane structure;

3. \( \text{mem} \) is the label of a distinguished component (the shared memory of the system);

4. \( P \) is a finite set of objects, called catalyser proteins, and \( P_h(0) \) is the protein initially associated with region labelled by \( h \);

5. \( E_{\text{mem}} \) is a finite set of variables, called enzymes, disjoint with \( \text{Var}_{\text{mem}} \), and \( E_{\text{mem}}(0) \) is the initial values of the enzymes;

6. \( \text{Var}_h, h \in H \), is a finite set of variables \( x_{j,h} \) associated with region labelled by \( h \) (a membrane or the shared memory), its values are natural numbers and the value of \( x_{j,i} \) at time \( t \in \mathbb{N} \) is denoted by \( x_{j,i}(t) \);

7. \( \text{Var}_h(0) \) is a vector that represents the initial values for variables in \( \text{Var}_h \);

8. \( P_r_h, h \in H \), is a finite set of programs associated with region labelled by \( h \), having the following syntactical format \( F(x_1, \ldots, x_{k_F,h}) \) \( \varepsilon(F) \alpha(F) \rightarrow c_1[v_1, \ldots, c_{n_F}|v_{n_F}] \), where:
   - \( F(x_1, \ldots, x_{k_F,h}) \) is a computable function (the production function), being \( x_{j,h}, \ldots, x_{k_F,h} \in \text{Var}_h \);
   - \( c_1[v_1, \ldots, c_{n_F}|v_{n_F}] \) is the repartition protocol associated with the program, being \( c_1, \ldots, c_{n_F} \) natural numbers specifying the proportion of the current production distributed to variables \( v_1, \ldots, v_{n_F} \in \text{Var}_h \cup \text{Var}_{\text{par}(h)} \cup \text{Var}_{\text{ch}(h)} \), where \( \text{par}(h) \) the parent of \( h \) and \( \text{ch}(h) \) the set of child of \( h \) in \( \mu \);
   - \( e(F) \in E_h \) is an enzyme and \( \alpha(F) \in P \) is a protein, both of them associated with program \( F \), if no enzyme or protein is used in a program then it will be omitted;

9. \( R \) is a finite set of rules of the following form:
   - \textbf{Protein evolution rules:} \( [\alpha \rightarrow \alpha']_h \), where \( h \in H, \alpha \in P \) and \( \alpha' \in P \).
   - \textbf{Writing-only communication rules between the shared memory and the membranes:}
     \( [h, X_h/Y_{h,\text{mem}}, \text{mem}]_\alpha^W \)
     where \( X_h \in \text{Var}_h, Y_{h,\text{mem}} \in \text{Var}_{\text{mem}}, \alpha \in P \) in such manner that there is, at most, one rule for each membrane \( h \in \{1, \ldots, p \cdot q \} \). Variables \( Y_{h,\text{mem}}, Y'_{h,\text{mem}} \) should be different for two membranes \( h, h' \).
   - \textbf{Reading-only communication rules between the shared memory and the membranes:}
     \( [h, X_h/Y_{\text{mem}}, \text{mem}]_\alpha^R \)
     where \( X_h \in \text{Var}_h, Y_{\text{mem}} \in \text{Var}_{\text{mem}}, \alpha \in P \). Variable \( Y_{\text{mem}} \) is the same for each \( h \in \{1, \ldots, p \cdot q \} \).

\begin{itemize}
    
    \item \textbf{Membrane creation rules:}
    \[ [ \begin{array}{c}
    X_{1,h} \quad X_{2,h} \quad \ldots \quad X_{n,h} \\
    h \end{array} ]_{h'} : \alpha \]
    where \( h, h' \in \{1, \ldots, p \cdot q \} \) are different, \( \alpha \in P \) and \( X_{1,h}, \ldots, X_{n,h} \subseteq \text{Var}_h \).
\end{itemize}

Fig. 3. The initial membrane structure with a representation of the shared memory.

The term \textit{region} \( h (h \in H) \) is used to refer to membrane \( h \) in the case \( h \in \{1, \ldots, p \cdot q \} \cup \{v\} \), as well as to refer to the shared memory in the case \( h = \text{mem} \).

Next, we describe the semantics of RENPSHs. A configuration of a RENPSH at any instant \( t \) is described by the current membrane structure \( \mu \), together with proteins and all values of the variables and enzymes associated with all regions. The initial configuration is \( [\mu, \text{Var}_h(0), \{P_h(0), \text{Var}_h(0) | h \in H\}] \), where \( \mu = \{[\{]_{h_a}][h_b]\} \). We will call \( \mu_a \) (resp. \( \mu_b \)) to the membrane structure rooted in membrane \( h_a \) (resp. \( h_b \)).

A program \( F(x_1, \ldots, x_{k_F,h}) \) \( \varepsilon(F) \alpha(F) \rightarrow c_1[v_1, \ldots, c_{n_F}|v_{n_F}] \) associated with a region is applicable to a configuration \( C_t \), at moment \( t \), if the value of \( e(F) \) at that instant is greater than \( \min\{x_{1,h}(t), \ldots, x_{k_F,h}(t)\} \) and protein \( \alpha(F) \) is inside the region \( h \) of \( C_t \). When applying such a program, variables associated with \( C_t \) are processed as follows: first, the value \( F(x_{1,h}(t), \ldots, x_{k_F,h}(t)) \) is computed as well as the value
\[ q(t) = \frac{F(x_{1,h}(t), \ldots, x_{k_F,h}(t))}{c_1 + \ldots + c_{n_F}} \]
This value represents the \textit{unary portion} at instant \( t \) to be distributed among variables \( v_1, \ldots, v_{n_F} \) according to the repartition expression. Thus, \( q(t) \cdot c_s \) is the contribution added to the current value of \( v_s (1 \leq s \leq n_F) \), at step \( t+1 \). So, \( v_s(t+1) = v_s(t) + q(t) \cdot c_s \) and \( v_s(t) \) become zero, i.e., it is assumed that variable \( v_s \) is “consumed” when the production function is used and other variables retain their values. Each program in each membrane can only be used once in every computation step, and all the programs are executed in parallel.

A protein evolution rule \( [\alpha \rightarrow \alpha']_h \) is applicable to a configuration \( C_t \) at moment \( t \) if protein \( \alpha \) is in membrane \( h \) of \( C_t \). When applying such a rule the protein \( \alpha \) in \( h \) evolves to protein \( \alpha' \) in \( h \). These rules are applied in a maximal manner.

A writing-only communication rule between the shared memory and the membranes, \( [h, X_h/Y_{h,\text{mem}}, \text{mem}]_\alpha^W \), is applicable to a configuration \( C_t \) at moment \( t \) if protein \( \alpha \) is in membrane \( h \) of \( C_t \). When applying such a rule the value \( X_h(t) \) is assigned to the variable \( Y_{h,\text{mem}}(t+1) \) of the shared
memory, that is \( Y_{h,\text{mem}}(t+1) \leftarrow X_h(t) \). These rules are applied in a maximal manner.

A reading-only communication rule between the shared memory and the membranes, \((h, X_h/Y_{\text{mem}}, \text{mem})^k\) is applicable to a configuration \( C_t \) at moment \( t \) if protein \( \alpha \) is in membrane \( h \) of \( C_t \). When applying such a rule the value \( Y_{\text{mem}}(t) \) is assigned to the variable \( X_h(t+1) \) of membrane \( h \), that is \( X_h(t+1) \leftarrow Y_{\text{mem}}(t) \). These rules are applied in a maximal manner.

A membrane creation rule \([(X_{1,h}, \ldots, X_{n,h})_h]_{h'}\); \( \alpha \) is applicable to a configuration \( C_t \) at moment \( t \) if protein \( \alpha \) is in membrane \( h' \) of \( C_t \). When applying such a rule, a new membrane labelled by \( h \) is created in such manner that \( h' \) is the parent of \( h \) and the set of its variables is \( Var_h = \{X_{1,h}, \ldots, X_{n,h}\} \).

Given a random enzymatic numerical P system with proteins and shared memory \( \Pi \), we say that configuration \( C_t \) at time \( t \) yields configuration \( C_{t+1} \) in one transition step if we can pass from \( C_t \) to \( C_{t+1} \) by applying in parallel each program in each membrane only once, and by applying the rules in a maximal parallel way following the previous remarks. A computation of \( \Pi \) is a (finite or infinite) sequence of configurations such that: (a) the first term is the initial configuration of the system; (b) for each \( n \geq 2 \), the \( n \)-th configuration of the sequence is obtained from the previous configuration in one transition step; and (c) if the sequence is finite (called halting computation) then the last term is a halting configuration (a configuration where no rule of the system is applicable to it). All the computations start from an initial configuration and proceed as stated above; only halting computations give a result, which is encoded by the objects present in the output region \( i_{\text{out}} \) associated with the halting configuration. If \( C = \{C_i\}_{i \leq r+1} \) of \( \Pi \) \((r \in \mathbb{N}) \) is a halting computation, then the length of \( C \), denoted by \( |C| \), is \( r \). For each \( i \) \((1 \leq i \leq q) \), we denote by \( C_i(i) \) the finite multisets of objects over \( I^* \) contained in all membranes labelled by \( i \) at configuration \( C_i \).

V. SIMULATION OF ONE ITERATION OF THE BIDIRECTIONAL RRT ALGORITHM FOR PATH PLANNING

The input of the bidirectional RRT algorithm generating a global path for a robot trajectory consists of the following parameters \((x_{\text{init}}, x_{\text{end}}, K, \rho, \Delta t, X, X_{\text{obs}}, d_{\text{min}})\), where:

- \( x_{\text{init}} \) is the initial state.
- \( x_{\text{end}} \) is the ending state.
- \( K \) is the number of iterations to find the path.
- \( \Delta t \) is a fixed amount of time for transitions.
- \( X \) is the state space.
- \( X_{\text{obs}} \subseteq X \) is the obstacle state space.
- \( d_{\text{min}} \) is the minimum distance threshold according to some distance metric \( \rho \) in order to include a new node in an RRT.

For mobile robots in a 2D environment, the state space is given by \((x, y, \theta)\), i.e. the Cartesian coordinates \((x, y)\) and the heading angle \( \theta \) of all the possible robot poses. However, the angle \( \theta \) has been omitted in this solution to reduce the size of the problem and the state space is given by \((x, y)\)

considering the Euclidean distance as distance metric. In this case, a holonomic robot can follow the RRT by performing motions in straight line, otherwise a nonholonomic robot can include rotations in-place. Moreover, any state or position \((i, j) \in \{1, \ldots, p\} \times \{1, \ldots, q\}\) can be encoded by the natural number \((i-1) \cdot q + j\). In such a manner that, given a natural number \( n \) encoding a state \((i, j)\), the following holds: \( i = 1 + qt(n, q) \) and \( j = rm(n, q) \).

One iteration of the parallel bidirectional RRT algorithm defined in Algorithm 4 will be simulated by a RENPSM of degree \((p, q)\)

\[
\Pi = (H, \mu, P, E_{\text{mem}}, E_{\text{mem}}(0)), \quad \{(P_h(0), Var_h, Var_h(0), Pr_h) \mid h \in H\}, \mathcal{R}, h_a, h_b \}
\]

defined as follows:

- \( H = \{1, \ldots, p \cdot q\} \cup \{v, \text{mem}\}, v \notin \{1, \ldots, p \cdot q\}, \text{mem} \notin \{1, \ldots, p \cdot q\} \}
- \( \mu = \{h_a[h_b]v \mid h_a \in \{1, \ldots, p \cdot q\}, h_b \in \{1, \ldots, p \cdot q\} \} \) and \( h_a \neq h_b \). We call \( \mu \) to the membrane structure rooted on \( h_a \) and \( \mu \) to the one rooted on \( h_b \).
- \( P = \{\alpha_i \mid 1 \leq i \leq 18\} \), and \( P_h(0) = \{\alpha_1\} \), for each \( h \in H \).
- \( E_{\text{mem}} = \{\text{Flag}_{A,\text{mem}}, \text{Flag}_{B,\text{mem}}, p \cdot q + 1\} \) and \( E_{\text{mem}}(0) = \{p \cdot q + 1\} \).

The set of variables is:

- \( Var_h = \{X_{1,h}, X_{2,h}, Y_{1,h}, Y_{2,h}, Z_{1,h}, Z_{2,h}, D_h\} \), for each \( h, 1 \leq h \leq p \cdot q \).
- \( Var_{\text{mem}} = \{X_{1,\text{mem}}, X_{2,\text{mem}}, X_{3,\text{mem}}, X_{4,\text{mem}}\} \cup \{Y_{1,\text{mem}}, Y_{2,\text{mem}}, Y_{3,\text{mem}}, Y_{4,\text{mem}}\} \cup \{Z_{1,\text{mem}}, Z_{2,\text{mem}}, Z_{3,\text{mem}}, Z_{4,\text{mem}}\} \cup \{U_{1,\text{mem}}, U_{2,\text{mem}}, U_{3,\text{mem}}, U_{4,\text{mem}}\} \cup \{A_{\text{mem}}, B_{\text{mem}}, N_{A_{\text{mem}}, N_{B_{\text{mem}}, N_{H_{\text{mem}}}}}} \cup \{A_{\text{mem}}, B_{\text{mem}}, N_{A_{\text{mem}}, N_{B_{\text{mem}}, N_{H_{\text{mem}}}}}} \cup \{A_{\text{mem}}, B_{\text{mem}}, N_{A_{\text{mem}}, N_{B_{\text{mem}}, N_{H_{\text{mem}}}}}\}

- Initially, all variables in \( Var_h(h \neq h_a \land h \neq h_b) \) and all variables in \( Var_{\text{mem}} \) different to \( Y_{1,h}, Y_{2,h} \), are equal to zero. Besides, initially the tuple \((Y_{1,h}, Y_{2,h})\) (resp. \((Y_{1,h}, Y_{2,h})\)) provides the position of the initial state of the robot \( h_a \) (resp. the position of the final state of the robot \( h_b \)).

If variable \( Halt_{\text{mem}} \) is equals to 1, then the computation stops.

- Next, the finite set of programs \( P_{\text{end}} \) and the set of rules \( \mathcal{R} \) of the system are defined according to the requirements to simulate the bidirectional RRT algorithm.

- In order to synchronize the sequence of an iteration, for each \( h \in H \) the protein evolution rules \( \{\alpha_i \rightarrow \alpha_{i+1}\}_h \), for \( 1 \leq i \leq 17 \), and \( [\alpha_{18} \rightarrow \alpha_1]_h \) are considered.

Four random numbers are generated in the shared memory; for \( 1 \leq k \leq 4 \):

\[
\begin{cases}
\text{Production function} : F(X_{k,\text{mem}}) = \\
\text{Random}(i, 1 \leq i \leq p) \\
\text{Repartition protocol} : 1 \mid X_{k,\text{mem}} \\
\text{Protein} : \alpha_3
\end{cases}
\]
• Each membrane $h \in \mu_a$ will read the random numbers $X_{1,mem}, X_{2,mem}$. Each membrane $h \in \mu_b$ will read the random numbers $X_{3,mem}, X_{4,mem}$.

$$\begin{align*}
\{ (h, X_{1,h} / X_{1,mem}, mem)_{a_2}^R : h \in \mu_a \} \\
\{ (h, X_{2,h} / X_{2,mem}, mem)_{a_2}^R : h \in \mu_a \} \\
\{ (h, X_{1,h} / X_{3,mem}, mem)_{a_2}^R : h \in \mu_b \} \\
\{ (h, X_{2,h} / X_{4,mem}, mem)_{a_2}^R : h \in \mu_b \}
\end{align*}$$

• For each membrane $h \in \{\mu_a, \mu_b\}$, the distance $D_h$ between its position $(Y_{1,h}, Y_{2,h})$ and the position given by the generated random natural numbers $(X_{1,h}, X_{2,h})$ is computed. For the remaining membranes, $D_h = p \cdot q + 1$.

$$\begin{align*}
\text{Production function: } & F(X_{1,h}, X_{2,h}, Y_{1,h}, Y_{2,h}) = \\
& \begin{cases} \\
\sqrt{\sum_{j=1}^{2}(X_{j,h} - Y_{j,h})^2} & \text{if } h \in \{\mu_a, \mu_b\} \\
p \cdot q + 1 & \text{if } h \notin \{\mu_a, \mu_b\} \\
\end{cases} \\
\text{Repartition protocol: } & 1|D_h \\
\text{Protein: } & \alpha_3
\end{align*}$$

• Each membrane $h$ writes its value $D_h$ to the shared memory.

$$\begin{align*}
\{ (h, D_h / A_{h,mem}, mem)_{a_4}^W : h \in \mu_a \} \\
\{ (h, D_h / B_{h,mem}, mem)_{a_4}^W : h \in \mu_b \}
\end{align*}$$

• The minimum of all distances $A_{h,mem}$ is computed in the shared memory.

- Production function: $F(A_{1,mem}, \ldots, A_{p,q,mem}) = \min\{A_{1,mem}, \ldots, A_{p,q,mem}\}$
- Repartition protocol: 1|A
- Protein: $\alpha_5$

• The minimum of all distances $B_{h,mem}$ is computed in the shared memory.

- Production function: $F(B_{1,mem}, \ldots, B_{p,q,mem}) = \min\{B_{1,mem}, \ldots, B_{p,q,mem}\}$
- Repartition protocol: 1|B
- Protein: $\alpha_5$

• Variable (enzyme) $FlagA_{mem}$ is set to zero if $A_{mem} \leq \text{Threshold}$.

- Production function: $F(A_{mem}) = \\
\begin{cases} \\
0 & \text{if } A_{mem} \leq \text{Threshold} \\
p \cdot q + 1 & \text{otherwise} \\
\end{cases}$
- Repartition protocol: 1|FlagA
- Protein: $\alpha_6$

• Variable (enzyme) $FlagB_{mem}$ is set to zero if $B_{mem} \leq \text{Threshold}$.

- Production function: $F(B_{mem}) = \\
\begin{cases} \\
0 & \text{if } B_{mem} \leq \text{Threshold} \\
p \cdot q + 1 & \text{otherwise} \\
\end{cases}$
- Repartition protocol: 1|FlagB
- Protein: $\alpha_6$

• The label $near_a$, corresponding to the closer membrane to the randomly generated pose for $\mu_a$, is obtained.

- Production function: $F(A_{1,mem}, \ldots, A_{p,q,mem}) = \arg\min\{A_{1,mem}, \ldots, A_{p,q,mem}\}$
- Repartition protocol: 1|NA
- Protein: $\alpha_7$
- Enzyme: $FlagA_{mem}$

• The label $near_b$, corresponding to the closer membrane to the randomly generated position for $\mu_b$, is obtained.

- Production function: $F(B_{1,mem}, \ldots, B_{p,q,mem}) = \arg\min\{B_{1,mem}, \ldots, B_{p,q,mem}\}$
- Repartition protocol: 1|NB
- Protein: $\alpha_7$
- Enzyme: $FlagB_{mem}$

• The position of membrane $near_a$ is computed.

$$\begin{align*}
\text{Production function: } & F(NA_{mem}) = 1 + qt(NA_{mem}, q) \\
\text{Repartition protocol: } & 1|Y_{1,mem} \\
\text{Protein: } & \alpha_8 \\
\text{Enzyme: } & FlagA_{mem}
\end{align*}$$

• The position of membrane $near_b$ is computed.

$$\begin{align*}
\text{Production function: } & F(NB_{mem}) = 1 + qt(NB_{mem}, q) \\
\text{Repartition protocol: } & 1|Y_{2,mem} \\
\text{Protein: } & \alpha_8 \\
\text{Enzyme: } & FlagB_{mem}
\end{align*}$$

• The unitary vectors are created in the shared memory.

$$\begin{align*}
\text{Production function: } & F(X_{1,mem}, X_{2,mem}, Y_{1,mem}, Y_{2,mem}) = \\
& \frac{1}{X_{1,mem} - Y_{1,mem}} \\
& \sqrt{\sum_{j=1}^{2}(X_{j,mem} - Y_{j,mem})^2} \\
\text{Repartition protocol: } & 1|U_{1,mem} \\
\text{Protein: } & \alpha_9 \\
\text{Enzyme: } & Flag1_{mem}
\end{align*}$$

$$\begin{align*}
\text{Production function: } & F(X_{1,mem}, X_{2,mem}, Y_{1,mem}, Y_{2,mem}) = \\
& \frac{1}{X_{2,mem} - Y_{2,mem}} \\
& \sqrt{\sum_{j=1}^{2}(X_{j,mem} - Y_{j,mem})^2} \\
\text{Repartition protocol: } & 1|U_{2,mem} \\
\text{Protein: } & \alpha_9 \\
\text{Enzyme: } & Flag1_{mem}
\end{align*}$$
Variable (enzyme) $FlagA_{mem}$ is set to zero if there is collision for $\mu_a$.

- Production function:
  \[
  F(Y_{1,mem}, Y_{2,mem}, U_{1,mem}, U_{2,mem}) = \begin{cases} 
  0 & \text{if } COLLISION(Y_{1,mem}, Y_{2,mem}, \\
  p \cdot q + 1 & \text{otherwise}
  \end{cases}
  \]
  Repartition protocol: 1 | $FlagA_{mem}$
  Protein: $\alpha_9$
  Enzyme: $Flag2_{mem}$

Variable (enzyme) $FlagB_{mem}$ is set to zero if there is collision for $\mu_b$.

- Production function:
  \[
  F(Y_{3,mem}, Y_{4,mem}, U_{3,mem}, U_{4,mem}) = \begin{cases} 
  0 & \text{if } COLLISION(Y_{3,mem}, Y_{4,mem}, \\
  p \cdot q + 1 & \text{otherwise}
  \end{cases}
  \]
  Repartition protocol: 1 | $FlagB_{mem}$
  Protein: $\alpha_{10}$
  Enzyme: $Flag2_{mem}$

Positions of new membranes are computed in the shared memory.

- Production function:
  \[
  F(Y_{1,mem}, U_{1,mem}) = Y_{1,mem} + U_{1,mem} \cdot \Delta t
  \]
  Repartition protocol: 1 | $Z_{1,mem}$
  Protein: $\alpha_{11}$
  Enzyme: $FlagA_{mem}$

- Production function:
  \[
  F(Y_{2,mem}, U_{2,mem}) = Y_{2,mem} + U_{2,mem} \cdot \Delta t
  \]
  Repartition protocol: 1 | $Z_{2,mem}$
  Protein: $\alpha_{11}$
  Enzyme: $FlagA_{mem}$

The membranes labelled by $NA_{mem}$ and $NB_{mem}$ will read the positions corresponding to the new membranes from the shared memory.

- A child membrane with position $(Z_{1,NA_{mem}}, Z_{2,NA_{mem}})$ is created in $\mu_q$.
  \[
  \begin{bmatrix}
  X_{1,h} \\
  Y_{1,h} \\
  Z_{1,h} \\
  D_h
  \end{bmatrix}
  \]
  Being $h = (Z_{1,NA_{mem}} - 1) \cdot q + Z_{2,NA_{mem}}$.
  This rule is mediated by protein $\alpha_{12}$.

- A child membrane with position $(Z_{1,NB_{mem}}, Z_{2,NB_{mem}})$ is created in $\mu_b$.
  \[
  \begin{bmatrix}
  X_{1,h} \\
  Y_{1,h} \\
  Z_{1,h} \\
  D_h
  \end{bmatrix}
  \]
  Being $h = (Z_{1,NB_{mem}} - 1) \cdot q + Z_{2,NB_{mem}}$.
  This rule is mediated by protein $\alpha_{13}$.

Each membrane in $\mu_a$ reads the pose of the new membrane created in $\mu_b$.

- Each membrane in $\mu_a$ reads the pose of the new membrane created in $\mu_b$.
  \[
  \begin{bmatrix}
  h, X_{1,h} / Z_{3,mem}, mem \end{bmatrix}^R_{\alpha_{14}} : h \in \mu_a
  \]
  \[
  \begin{bmatrix}
  h, X_{2,h} / Z_{4,mem}, mem \end{bmatrix}^R_{\alpha_{14}} : h \in \mu_a
  \]
For each membrane $h \in \{\mu_a, \mu_b\}$, the distance $D_h$ between its position $(Y_{1,h}, Y_{2,h})$ and the position given by the new membrane in the other membrane structure is computed. For the remaining membranes, $D_h = p \cdot q + 1$.

\[
\begin{align*}
\text{Production function: } & F(X_{1,h}, X_{2,h}, Y_{1,h}, Y_{2,h}) = \\
& \sqrt{\sum_{j=1}^{2}(X_{j,h} - Y_{j,h})^2} & \text{if } h \in \{\mu_a, \mu_b\} \\
& p \cdot q + 1 & \text{if } h \notin \{\mu_a, \mu_b\}
\end{align*}
\]

Repartition protocol: $1|D_h$

\[
\text{Protein: } \alpha_{15}
\]

Each membrane $h$ writes its value $D_h$ to the shared memory.

\[
\begin{align*}
(h, D_h / A_{h,\text{mem}}, \text{mem})_{\alpha_{16}} : h \in \mu_a \\
(h, D_h / B_{h,\text{mem}}, \text{mem})_{\alpha_{16}} : h \in \mu_b
\end{align*}
\]

The minimum of all distances $A_{h,\text{mem}}$ is computed in the shared memory.

- Production function: $F(A_{1,\text{mem}}, \ldots, A_{p,\text{mem}}) = \min\{A_{1,\text{mem}}, \ldots, A_{p,\text{mem}}\}$
- Repartition protocol: $1|A_{\text{mem}}$
- Protein: $\alpha_{17}$

The minimum of all distances $B_{h,\text{mem}}$ is computed in the shared memory.

- Production function: $F(B_{1,\text{mem}}, \ldots, B_{p,\text{mem}}) = \min\{B_{1,\text{mem}}, \ldots, B_{p,\text{mem}}\}$
- Repartition protocol: $1|B_{\text{mem}}$
- Protein: $\alpha_{17}$

If $A_{\text{mem}} \leq d_{\text{min}}$ or $B_{\text{mem}} \leq d_{\text{min}}$ then the RRTs have been connected and the computation must halt.

- Production function: $F(A_{\text{mem}}, B_{\text{mem}}) = \\
\begin{cases} 
1 & \text{if } A_{\text{mem}} \leq \text{Threshold} \vee B_{\text{mem}} \leq \text{Threshold} \\
0 & \text{otherwise}
\end{cases}$
- Repartition protocol: $1|\text{Halt}_{\text{mem}}$
- Protein: $\alpha_{18}$

VI. Simulation experiments

A C++ simulator has been developed within the ROS [24] framework. It can be downloaded from https://github.com/RGNC/renpsm. The experiments have been conducted by using a dual-wheeled nonholonomic robot (the Pioneer 3-DX) in two virtual environments. The software is composed by three modules:

- MobileSim module [25]: It receives the static information about the map, as well as motion commands $(v, \omega)$ and generates the wheels odometry and information related to sensors (laser rangefinder for obstacle detection). It moves the simulated robot in the virtual environment.
- RENPSM module: It receives the information about the map, as well as the information about odometry and sensors and the goal of the robot. It computes a bidirectional RRT by using a RENPSM simulator and finally it sends a sequence of motion commands to the MobileSim module.
- RVIZ module [26]: This module is used for visualization. It receives the static information about the map, as well as all the information generated by the MobileSim module and several visual markers generated by the RENPSM module. It shows to the user all the information in real-time by using a 3D representation of the environment and the robot.

We have used two virtual environments, in Figures 4 and 6, it can be seen the corresponding RVIZ visualization for each one before starting the robot motion, i.e, after generating the bidirectional RRT by using the RENPSM module.

The first environment has been used for experimental validation of the RENPSM model by generating several simulations and comparing the resulting RRT visualizations with the ones generated with a conventional RRT software. The second environment has been used for benchmarking, generating 1435 simulations by fixing the starting point and the goal of the robot and measuring the cost in distance of the generated path. The results are shown in Table I.

We have measured the cost of an optimal path generated by hand (about 10m) and, as expected, the cost of the best path generated by the bidirectional RRT is larger than the optimal cost, since the algorithm generates the first feasible path that can be found.

![Fig. 4. RVIZ visualization of a simulation in environment 1](image1)

![Fig. 5. RVIZ visualization of a simulation in environment 2](image2)
VII. CONCLUSIONS

This paper deals with an algorithm belonging to a family widely used to solve the problem of motion planning in robots, e.g., the RRT algorithms. Such class of algorithms are based on the randomized exploration of the configuration space. The main contribution of this paper is to study the bidirectional RRT algorithm within the framework of Membrane Computing.

In this work, a variant of Enzymatic Numerical P systems, called random enzymatic numerical P systems with proteins and shared memory (RENPSM, for short) is introduced. Besides, a simplified version of the standard bidirectional RRT algorithm is described by a RENPSM system capturing the semantics of the new variant, where maximal parallelism is used.

A software platform has been provided in order to simulate RENPSM systems and validate the computations in comparison with the usual implementation of the bidirectional RRT algorithm. Some benchmarking results are also included.

Three challenges are planned as future work. First, to provide a formal verification of such RENPSM systems, in the sense that they in fact simulate the RRT generation algorithm. The second challenge is to move to the RRT* algorithm [6], a variant of the initial algorithm that is able to approximate optimal motion planning with enough iterations. Finally, to provide real-life robot path planning experiments, by using a nonholonomic robot with kinodynamic and environment constraints.

ACKNOWLEDGEMENTS

This work was supported by National Natural Science Foundation of China (61672437 and 61702428) and by Sichuan Science and Technology Program (18ZDYF2877, 18ZDYF1985).

Authors from the University of Seville also acknowledge the support of the Spanish Ministry of Economy, Industry and Competitiveness by means of the project Máquinas bioinspiradas sobre plataformas de altas prestaciones: Un enfoque multidisciplinar (TIN2017-89842-P).

REFERENCES