UNIVERSITY OF PADOVA

Department of Information Engineering

Master Degree in Computer Engineering

Master Thesis

A distributed approach for safe kinodynamic motion planning in multi-robot systems, based on a Social Force Model

Student:
Nicolò Boscolo
matr. 626611

Supervisor:
Prof. Enrico Pagello
Co-supervisor:
Ing. Matteo Munaro

Academic year 2011/2012

“Considerate la vostra semenza: fatti non foste a viver come bruti ma per seguir virtute e canoscenza.”

Divina Commedia

Introduction

This work deals with the design of a real-time system able to solve the problem of planning collision-free motions for multiple robot vehicles that operate in the same, partially-observable environment. In robotics this problem has a wide literature and a lot of solutions were found. One of these systems uses the coordination of robots in a distributed manner even considering the kinodynamic time constraints. The system proposed in this work uses an hybrid architecture using the Social Force Model for dealing with human presence. This kind of systems permits scalability and a low computational load for the single robot. These document describe the design of a distributed system able to work with robot with low computational resources. All the systems proposed in the literature were developed with the main objective of using particular software frameworks oriented to use it in a mainframe system. Our proposed implementation goes in the opposite direction. We used a general purpose robotic operating system developed by Willow Garage, called ROS. This policy gives the chance to this framework to be potentially used on every robot using the ROS middleware.

After a brief overview (Chapter 1) where the problem is described, following, we introduce a general system architecture for a moving autonomous system (Chapter 2). In this part we describe the navigation system, the ICS state and the collision avoidance problem. After that we explain how to add intelligence to the system with different features. In Chapter 4 a people motion prediction is described. Moreover, in Chapter 3, we explain how to add the collaboration between robots. After the main theoretical topics we describe the ROS implementation (Chapter 5) of the system proposed in Chapter 4.3. We conclude (Chapter 6) reporting experimental results.

Contents

Chapter 1

Overview

1.1 Brief state of art

The ability to safely navigate in crowded environments is a key element for most applications involving mobile robots, and collision avoidance is a crucial component of any navigation systems. Here we focus on coordinating robots' maneuvers to achieve collision avoidance for a group of mobile platforms with limited computational capabilities.

The *collision avoidance problem* in a known static environment with multiple robots has a wide literature, but we can identify two approaches: the *reactive (myopic)* and the *predictive* one. The former is a class of methods that permits robots to avoid collisions on a dynamic environment without explicit communication. Such methods include the *Dynamic Window Approach* [3] and *Velocity Obstacles* [9]. The latter has his most recent extension on the *Optimal Reciprocal Collision Avoidance* (ORCA) [21]. This can be used to simulate thousands of moving agents without collisions and achieve this objective without communication. Among myopic methods, path deformation techniques compute a flexible path that is adapted on-line so as to avoid moving obstacles [26]. These approaches are very efficient in simulations with a high number of agents as shown with *Reciprocal Velocity Obstacles* (RVO) [22]. However, that approach only works well if the only moving obstacles are other robots with the same behavior, furthermore some deadlocks can arise, e.g. the *dancing behavior*, and it can not deal with the *Inevitable Collision State* (ICS)[1] issue [11].

The predictive approaches can be addressed either with coupled or decoupled approaches. *Coupled* approaches guarantee completeness but generate an exponential dependence on the number of robots and use a centralized computation [6]. Decoupled approaches allow robots to compute their own paths and then resolve conflicts, so that feasible solutions are usually incomplete, but computed faster and in a decentralized way. For instance, in prioritized planners, where low priority robots have to adapt their path plans upon the decisions of high

[1]A state is an ICS if every next state involves a collision.

priority robots, this decoupled approach could have a heavy impact on finding a feasible real-time solution.

The solution to the path-planning problem for robots with second-order dynamics, i.e. the kind of robots proposed in this work, can be achieved by using a sampling-based tree planner [16, 2] and, even if all robots decide a feasible plan, maybe its end state is an ICS [11]. The literature on contingency planning to avoid ICS in static environments shows that braking maneuvers are sufficient to provide safety if used within a control-based scheme [25] or in sampling-based re-planning [4], as well as with learning-based approximations of ICS sets [14] or approximations for computing space×time obstacles [5].

This work uses a not prioritized, decoupled approach and is inspired by the safety rules proposed in [2] about how to avoid ICS and collisions between robots. This system is based on the computation of a set of plans concatenated with the robot's contingency plan and the exchange of this information among robots. This permits the choice of a safe trajectory for every robot. Here, we propose the same kind of decoupled approach but with some key differences: the *factor graph* [15] as communication network, the *max-sum algorithm* [1] for the distributed coordination. The system still provide safety and good performance even without real-time design communication protocols such as in ROS.

1.2 Problem Setup

We are going to propose our formal representation of the problem. Stressing, the system faces with an partially-known environment. Here a crew of moving robots has to reach different goals where moving unknown obstacles are present.

Let R be a set of n independent robots, i.e. $R = \{R_1, R_2, \dots, R_n\}$ and let each robot R_i, $1 \leq i \leq n$ have second order dynamics ruled by the time constraint

$$\dot{x}_i(t) = f(x_i(t), u_i),\ g(x_i(t), \dot{x}_i(t)) \leq 0,\ \forall t \in \mathbb{R}, \tag{1.2.1}$$

where $x_i(t)$ represents a system state, u_i a robot control and function f and g are both smooth. Let $E \subseteq \mathbb{R}^2$ be the *environment* where the robot operates and $FE \subseteq E$ the *free environment*, the free space of that environment. Given a point $p \in \mathbb{R}^2$, for each robot R_i, $f_P(R_i, p)$ is called the *footprint* on the point p, i.e. the subset of FE occupied by the robot, while $c(R_i) \subseteq \mathbb{R}^2$ is the center of that footprint. The 2D local subspace of FE where robot R_i can perceive and move, i.e. every $c(R_i)$ such that $f_P(R_i, c(R_i)) \subseteq FE$, is called the *safe environment* and is represented by SE_i.

Let robot R_i be the owner of a *local goal* list GL_i filled with 2D space points $(x, y) \in SE_i$, the problem to be solved is the following: every robot R_i have to reach its *global goal* G_i passing through a sequence of local goals, where a local goal can be reached by choosing a linear trajectory and maintaining a speed $\mathbf{v}_i$ selected from V_i, a *discrete and finite set of velocities.* In particular, when a local goal is reached, R_i has to compute a new GL_i and select both a new local goal $gl_i \in GL_i$ and a velocity $\mathbf{v}_i \in V_i$ such that, until R_i does not reach gl_i, $\forall t$

$$\begin{cases} f_P\left(R_i, \mathbf{v}_i \cdot t\right) \in SE_i \\ f_P\left(R_i, \mathbf{v}_i \cdot t\right) \cap f_P\left(R_i, \mathbf{v}_j \cdot t\right) \neq \emptyset \quad \forall i \neq j \end{cases} . \tag{1.2.2}$$

Definition 1. Given a generic robot R_i, we will call *Trajectory* tr_i a tuple $(sp_i, \mathbf{v}_i, dt_i)$ where

- $sp_i = f_p\left(R_i, \mathbf{v_i} \cdot t\right), t = 0$ is the starting point of R_i,
- $\mathbf{v}_i \in V_i$ is the selected velocity for reaching the selected local goal gl_i,
- dt_i is the period where to apply $\mathbf{v}_i$.

Then, we evince that $\mathbf{v}_i \cdot dt = gl_i$, so we can even write the tuple tr_i as (sp_i, gl_i) if needed.

Remark. We are not placing limits about the *shape* of the robot, just on kinodynamic.

Chapter 2

System architecture

A basic architecture is needed by an autonomous system moving in the world. Here it is described an implementation of that architecture. As annotation, the modules meaning below described will not change on the next chapters.

2.1 Main modules

The robot executes a sense-planning-act cycle. For accomplish it, the automata performs the following steps using the modules on the system represented in Figure 2.0.1:

1. The *Environment Model Builder* retrieves sensor and odometry data and computes a *costmap*, a discrete grid inflated with costs (e.g. distance from an obstacle) obtained from the environment sensor data;

2. Given the global goal and the SE, the *Local Goals Generator* computes a set of feasible[1] goals around the robot position.

[1]Feasible means that for sure there is a path between robot position and the goal.

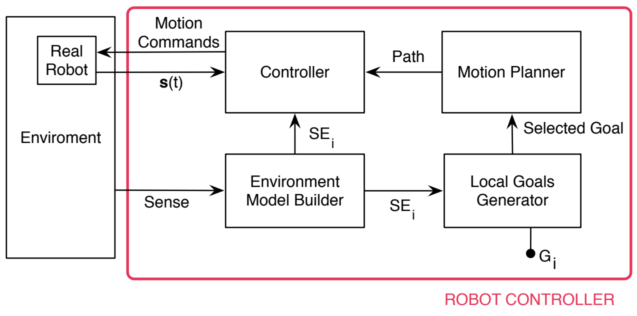


Figure 2.0.1: Block schema of the collision avoidance system.

3. Given a local goal, the *Motion Planne*r computes the shortest path to reach it.
4. The *Controller* receives the path and starts to send the motion commands to the real robot using a reactive approach(see 2.3.1) until it does not reach the local goal.

Unlike other similar works on distributed planning, this one does not use a single approach for computing the path to reach the goal pursued by the robot. Moreover, even if the RRT algorithm or another probabilistic approach has been demonstrated powerful[19], in the environment used in this work could not be effective. Every probabilistic approach checks, before any movement of the machine, if all the robot's states reached along the path towards the waypoint will not collide with the static obstacles. In this case the environment can change rapidly and it is very common that the agent cannot respect perfectly the roadmap. The reasons are many: because it has to avoid moving obstacles, for its physical constraints (e.g. different wheels friction) or the software loop control that does not respect the deadline (see 6). Then, our policy prefers that only the local goal is in a position without collisions whereas the computed path has to be just far enough from the static obstacles. We use this approach because, along the way, we will use a reactive approach for avoiding unknown threats. Respecting that policy we use two algorithms:

- **Dijkstra's algorithm** for the global navigation, that is used for compute the path from the current position and the local goal. The local goal is selected using a greedy approach described below, see 2.2.1.
- **Dynamic Window approach**[10] for the local/reactive navigation. It follows the global path ensuring the obstacle avoidance from static or moving and unknown obstacles.

2.2 Global Navigation: Dijkstra's algorithm

As described in 2.1, the Environment Model Builder create a costmap, which is a grid where each cell has a cost $c \in EnvValues$ $(\subset \mathbb{Z})$. The values in $EnvValues$ can be interpreted as danger levels. When a cell has a danger level $c \geq thresh$, where $tresh \in EnvValues$ is a danger threshold, the cell is representing an obstacle. The computation of the path for robot R is given by the Dijkstra's algorithm [7] with some adaptations listed below:

- Let be the initial node, the cell of the cost-map where the robot center $c(R)$ is placed;
- All the cells are considered as nodes connected by edges with the near cells. When a cell has a cost higher than the danger threshold, it will be not connected with its neighbors;
- The edges have a unitary cost.

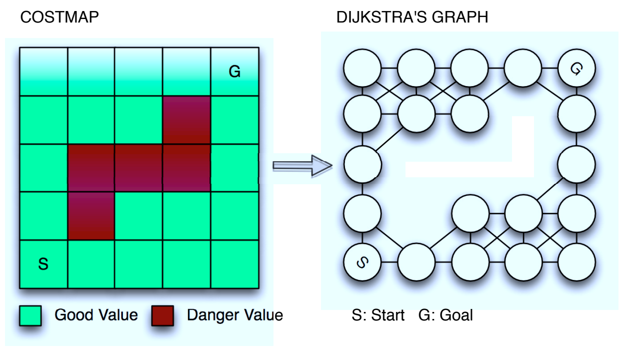


Figure 2.2.1: Transformation of costmap into a graph

The weakness of the algorithm is that it makes no attempt to direct "exploration" towards the destination as one might expect. Rather, the only consideration in determining the next "current" node is its distance from the starting point. In some sense, this algorithm "expands outward" from the starting point, iteratively considering every node that is closer in terms of shortest path distance until it reaches the destination. So the algorithm has a relative slowness in some topologies.

2.2.1 The greedy approach

The greedy approach directly computes the set of reachable goals around the robot. We will consider a goal directly reachable if it is in the line of sight of the robot. With the purpose of build the set, the robot tests around itself every n $(n \in \mathbb{Z}^+, n \leq 360)$ degrees if there are at least $\frac{360}{n}$ local goals. The local goal must be placed at safety distance (e.g. 1.5 meters in our tests) even if the robot will move for a minor distance for the reasons explained below, see 2.2.3. There is no upper limit to the new goal distance, the bound is only related with the sensing capabilities of the robot. Given the set of the all feasible local goals, the approach selects the one which minimizes the distance from the global goal. If no goals are available, the robot starts the recovery plan as explained in 2.2.3.

2.2.2 Avoid the patrolling behavior

The simple greedy approach could arise a problem. For instance, consider the scenario where the robot reaches the end of a close corridor and the global goal is just after the corridor wall, so that the final goal is not directly reachable. Using the logic of the previous greedy solution all the reachable goals will be behind the robot. So it will reach one of these goals and for sure it will increase the distance form the global goal. Then, the following greedy step oblige the robot to return again next to the wall, because it is the waypoint that minimize the distance from the global goal: the robot enters in a “looping behavior” where

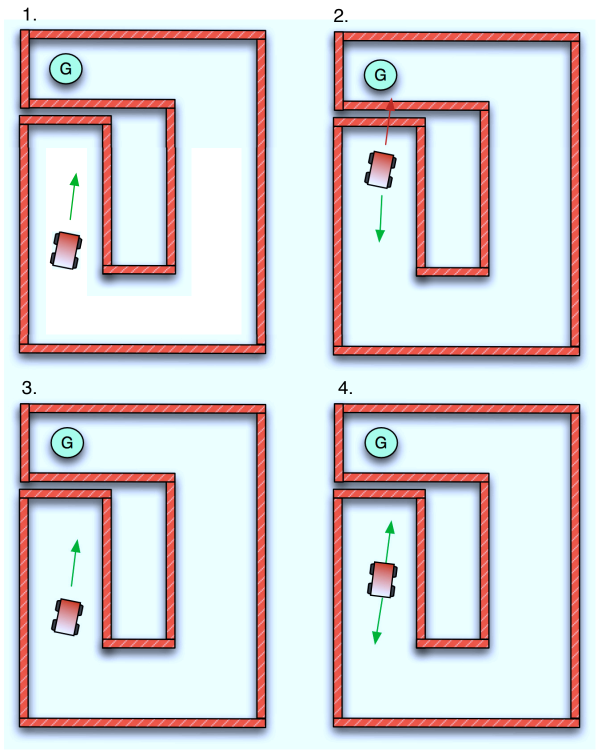


Figure 2.2.2: The looping behavior.

it looks like patrolling, as shown in Figure 2.2.2.

With the purpose of bypassing this critical problem, a navigation algorithm has been developed (see Algorithm2.1). Specifically, we use a set structure called geostructure, where we keep all the computed goals. This is a discrete set with a given precision degree. Then, when a goal is going to be added in the geostructure, but in the set there is a goal which is far less than the precision degree (i.e $0.5\,m$) from the new goal, the last one will not be added to the set. Geostructure is equipped with a method that return the goals directly reachable from the current robot position (neighborhood). Next, when we call *compute_ goals_ from* function, we select the safe goals around the robot position that are not near to the useless goals in geostructure. Finally, if at least a goal is found, we select the local goal that minimizes the distance to the global goal, otherwise recovery actions are taken. So, this algorithm add information about the global environment. Doing that, the robot knowledge increases.

Algoritmo 2.1 High level navigation procedure

```
geostructure gs;
gs.precision = 0.5;
position global_goal;
while(global_goal is not reached){
  curr_pos = get_current_position();
  gs.add(curr_pos);
  gs.find(curr_pos).type = GOOD;
  local_goals = compute_goals_from(curr_pos);
  if(local_goals.size() > 0){
    new_local_goal = select_best(local_goals);
    gs.add(new_local_goal);
  }else{
    gs.find(curr_pos).type = USELESS;
    recovery_goal = gs.find_good_neighbour(curr_pos);
    if(not set recovery_goal)
      contingency_plan();
    else
      move_to(recovery_goal);
  }
}
```

2.2.3 ICS and Recovery Action

An ICS is a state from which the robot can only transfer to a collision state. For instance, a collision state could be a position too near to a wall or a state with high velocity. Let *SD* be the *safe distance* (e.g. 0.5 meters) equals to the space needed by the robot to safely carry out one of its ICS escape maneuverers. The ICS is avoided using a simple policy: each robot has to cover at least a distance of SD, such that if the path has a length $L \geq SD$ the robot will move for that length L minus the safe distance SD, i.e. $L - SD$.

Our escape action is divided into two steps: first the robot tries to rotate slowly with the purpose of updating the costmap with some moving obstacles, then, if after two rotations obstacles still block its path the robot stops and looks for another path.

2.3 Local Navigation

2.3.1 Dynamic Window Approach (DWA)

The key advantage of local techniques over global ones lies in their low computational complexity, which is particularly important when the world model is updated frequently and based on sensor information. Other methods, like potential fields methods, often fail to find trajectories between closely spaced

1. **Search space**: The search space of the possible velocities is reduced in three steps:

   (a) **Circular trajectories**: The dyamic window approach considers only circular trajectories (curvatures) uniquely determined by pairs (v, ω) of translational and rotational velocities. This results in a two-dimensional velocity search space.

   (b) **Admissible velocities**: The restriction to admissible velocities ensures that only safe trajectories are considered. A pair (v, ω) is considered admissible, if the robot is able to stop before it reaches the closest obstacle on the corresponding curvature.

   (c) **Dynamic window**: The dynamic window restricts the admissible velocities to those that can be reached within a short time interval given the limited accelerations of the robot.

2. **Optimization**: The objective function

$$G(v, \omega) - \sigma\left(\alpha \cdot angle(v, \omega) + \beta \cdot dist(v, \omega) + \gamma \cdot vel(v, \omega)\right) \qquad (2.3.1)$$

   is maximized. With respect to the current position and orientation of the robot this function trades off the following aspects:

   (a) **Target heading**: *angle* is a measure of progress towards the goal location. It is maximal if the robot moves directly towards the target.

   (b) **Clearance**: *dist* is the distance to the closest obstacle on the trajectory. The smaller the distance of the robot and supports fast movements.

   (c) **Velocity:** *vel* is the forward velocity of the robot and supports fast movements.

The function σ smoothes the weighted sum of the three components and results in more side-clearance from obstacles.

Figure 2.3.1: Different parts of the dynamic window approach

obstacles. The approach here described is preferred to the more recents in literature because it's efficient even with low-cost robots, simple. Moreover the newer approaches were developed mainly for the simulation of thousand of agents. Therefore, those systems only work with unknown dynamic objects that use the same behavior of the agent, like RVO and its evolutions [23, 22, 21, 24, 20]. This approach differs from others in

- It is derived directly from the motion dynamics of the mobile robot;
- It takes the inertia of the robot into account which is particularly important if a robot with torque limits travels at high speed;
- the method has proven to avoid collisions reliably with speeds of up to $95\,cm/sec$ on several robots in several indoor and cluttered environments.

It uses a simple but elegant way to incorporates the dynamics of the robot: it reduces the velocity (rotational and translational) search space to the *dynamic window* which consists of the velocities reachable within a short time interval. Within the dynamic window, the approach only considers admissible velocities yielding a trajectory on which the robot is able to stop safely. The candidate velocity is selected using an objective function that includes a measure of progress towards a goal location, the forward velocity of the robot, and the distance to the next obstacle on the trajectory, see Figure 2.3.1. This permit a reactive approach to the collision avoidance with unknown obstacles.

In a nutshell, the approach considers periodically (every $0.25\,sec$) only the velocities reachable from the robot according with its dynamics, this set of rotational trajectories[2] is called V_s. Let be $V_a \subseteq V_s$ the set of (v,ω) couples that permits a safety robot break given $\dot{v}_b$ the breaking translational acceleration and $\dot{\omega}_b$, the breaking rotational acceleration (see Figure 2.3.2) . This set is computed with the following formula:

$$V_a = \left\{ (v,\omega)\,|v \leq \sqrt{2 \cdot dist\,(v,\omega) \cdot \dot{v}_b} \,\wedge\, \omega \leq \sqrt{2 \cdot dist\,(v,\omega) \cdot \dot{\omega}_b} \right\} \qquad (2.3.2)$$

where $dist\,(v,\omega)$ is the distance to nearest obstacle given the rotational trajectory.

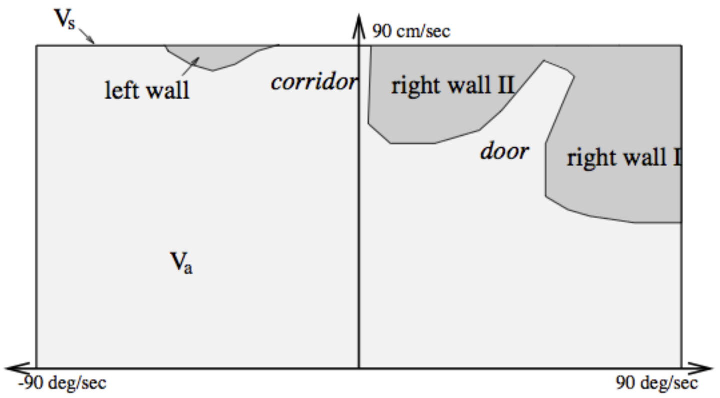
Now the set V_a is reduced again to the set V_d, that includes the velocities reachable in a short time t using feasible accelerations v_a and ω_a and is computed as below:

$$V_d = \{(v,\omega)\,|v \in [v_a - \dot{v} \cdot t, v_a + \dot{v} \cdot t] \,\wedge\, \omega \in [\omega_a - \dot{\omega} \cdot t, \omega_a + \dot{\omega} \cdot t]\} \qquad (2.3.3)$$

these velocities form the dynamic window V_r $(= V_s \cap V_a \cap V_d)$ (see Figure 2.3.3) which is centered around the current velocities of the robot in the velocity space.

[2]set of trajectories that make a translation and a rotation as well.

Figure 2.3.2: Velocity Space



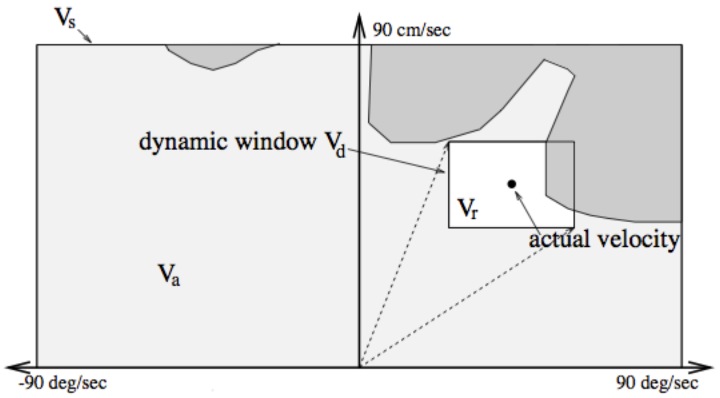


Figure 2.3.3: Dynamic Window

Algoritmo 2.2 Algorithm used in the Random Model.

```
Begin

 while(true)

   if global_goal not reached then
     local_goals_set = computeLocalGoals()
   else
     exit
   neighbours = getNeighbours()
   var waypoint new_goal
   if(neighbours > 0) then
     new_goal = select_random_goal(local_goals_set)
   else
     //the waypoint that mimimize the distance towards
         the global gloal
     new_goal = select_best_waypoint()
   move_to(new_goal)

 end while

End
```

2.4 Collision avoidance among robots

The approach proposed in 2.2.1-2.2.3 works well with one robot but the aim of this work is to provide safety motion planning for more robots playing on the same environment with heterogeneous goals. When using the previous policy it could happen that some robots collide because their trajectories intersect. It has been shown before that we cannot use a reactive and greedy approach alone. It can lies to a dangerous suboptimal solution where the robot is involved in a patrolling behavior.

2.4.1 Random Model

A naive solution is that when we are near to another robot and given the local goals set, the robots select the new waypoint randomly and hope that "everything it's gonna be right", see Section2.2. This approach could work if the robots are in a wide environment and the number of robots is really low.

The model does not make valid considerations about robot future states. Then, the solution is not considering the possibility to reach an ICS region (e.g. see Figure 2.4.1).

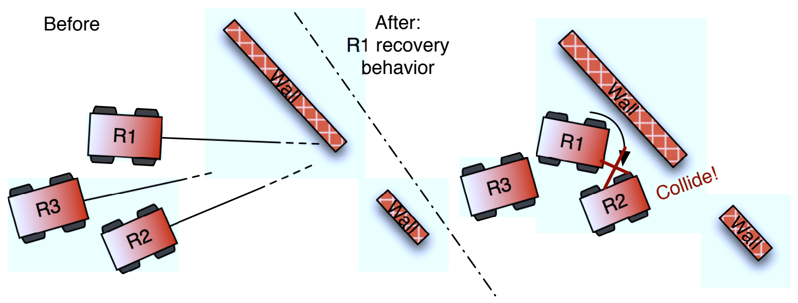


Figure 2.4.1: The robot 1 can't perform the recovery even if its current position is safe.

Chapter 3

The Distributed Approach

We can use another multi-robot collision avoidance method where the robots share their own trajectories with the neighbors and after that, each one decides which its own trajectory is the best safe solution. Formalizing, given a robot R_j and its robot neighbors NB_j[1], let be:

- INB_j , a vector where the indexes of the robots in NB_j are stored . $INB_j\,[0 \ldots |NB_j| - 1]$(i.e. looking at 3.0.4, $NB_4 = [4, 1, 3]$);
- $INB_j[0]$, for convention, is the index of the robot R_j ;
- given R_i, TR_i is the set of trajectories for reaching each local goal in GL_i ;

Each robot R_j has to find the tuple of trajectories

$$(tr_{INB_j[0]} \in TR_{INB_j[0]} \ldots \; tr_{INB_j[|NB_j|-1]} \in TR_{INB_j[|NB_j|-1]}) \tag{3.0.1}$$

[1]Remember that $R_j \in NB_j$ because each robot is neighbor of itself

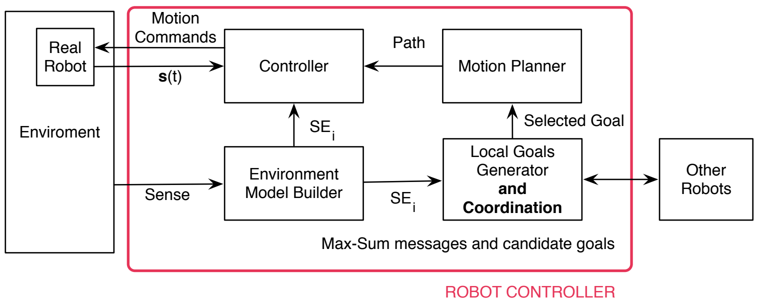


Figure 3.0.1: The Distributed Multi Robot System (DMRS).

Listing 3.1: Payoff function

```
//for convenience $tr_0$ is the trajectory of the robot $R_j$
    calling the function
begin $f_j\left(G, tr_0, \ldots, tr_{n-1}\right)$
  distance = 0
  safety_distance = 2.5
  goal_distance = get_euclidean_goal_distance($tr_0$,G)
  $\forall\left(tr_0, tr_v\right)$ with $u = 1 \ldots n-1$
    distance += min_euclidean_distance($tr_0, tr_u$)
  mean_distance = distance/ $(n-1)$
  if mean_distance < safe_distance then
    return log(mean_distance/safety_distance)
  else
    return log(safety_distance/goal_distance)
end
```

that maximizes a $\mathbb{R} - \textit{valued}$ function, like the following one:

$$f_j\left(G_i, tr_{INB_j[0]} \ldots tr_{INB_j[|NB_j|-1]}\right) \tag{3.0.2}$$

the function payoff will consider the mean minimum distance between trajectories and the distance from the end of the trajectory $tr_{INB_j[0]}$ to G_j, the global goal of R_j.

The higher is the minimum distance and the shorter is the distance with G_j, the higher is the value returned by f_j see Listing 3.1.

The *min_euclidean_distance* (tr_u, tr_v) function calculates the minimum distance among the two robots if they choose those trajectories. This function uses the velocity of the trajectory, adding the time dimension to the problem. Using this policy it could happen that two trajectories are good even if they are geometrically in collision, they are even good because by considering the time variable the robot don't collide.

The best tuple solution computed by the robot R_j probably is not the best one for its neighborhood. For this reason our approach will use a distributed method using the factorgraph and the max-sum algorithm to decide given some common rules or policies, which trajectory is the best safe solution for the neighborhood.

3.0.2 The Trajectory Window

When the number of robots increases (e.g. more than 5 robots) it is not rare that each robot R_i can count only on a reduced SE_i. In this case we cannot just consider the problem from the geometric side but maybe we should consider the time as variable. When the situation start to be chaotic it could be useful not to plan long paths but to use short-time paths and communicate with the neighbors whose *fast path* is more safe. Stressing, we want use the heuristic *"less free space, use more frequent collaborations".*

This heuristic appears on the system as time parameter called *Trajectory Window* T_{window}, that is the maximum time that a trajectory can be used by the robot. Then, we need to modify the Definition 1.

Definition 2. Given a generic robot R_i, we will call *Trajectory* tr_i a tuple $(sp_i, \mathbf{v}_i, dt_i)$ where

- $sp_i = f_p\left(R_i, \mathbf{v_i} \cdot t\right), t = 0$ is the starting point of R_i,
- $\mathbf{v}_i \in V_i$ is the selected velocity for reaching the selected local goal gl_i,
- dt_i is the period where to apply $\mathbf{v}_i$.
- $dt \leq T_{window}$

3.0.3 The Factor Graph

We use the *factor graph* [15] framework to solve the coordination in the distributed motion planning problem.

Indeed, let $x = \{x_1, x_2, \ldots, x_n\}$ be a collection of variables, where each variable x_i takes values in a finite alphabet A_i and let $g\left(x_1, x_2, \ldots, x_n\right)$ be a $R-valued$ function of these variables, i.e. a function whose domain is

$$S = A_1 \times A_2 \times \cdots \times A_n \tag{3.0.3}$$

and codomain R. The domain S is called *configuration space* for this collection of variables, while each element of S is a particular configuration of the variables, i.e. an assignment of a value to each variable. The codomain R may generally be any *semiring*, so that we can also assume R is the set of real numbers. We recall that a *commutative semiring* is a set K, with two binary operations called $" + "$ and $" \cdot "$, which satisfy these axioms:

1. The operation "+" is associative and commutative and there is an additive identity element "0" such that $k + 0 = k, \forall k \in K$;
2. The operation $" \cdot "$ is associative and commutative and there is a multiplicative identity element "1" such that $k \cdot 1 = k, \forall k \in K$;
3. For all triples (a, b, c), $a, b, c \in K$ $(a \cdot b) + (a \cdot c) = a \cdot (b + c)$, that is to say the distributive law holds.

As stated before, the set of real or complex numbers, with ordinary addition and multiplication, forms a commutative semiring. However there are many other commutative semirings, some of which are summarized in 3.1. For example, consider the Max-sum semiring (entry 5), where:

1. $K$is the set of real numbers, plus the symbol ∞;

Number	K	$"(+,0)"$	$"(\cdot,1)"$	Short name
1	$[0,\infty)$	$(+,0)$	$(\cdot,1)$	Sum-product
2	$(0,\infty]$	$(\min,\infty)$	$(\cdot,1)$	Min-product
3	$[0,\infty)$	$(\max,0)$	$(\cdot,1)$	Max-product
4	$(-\infty,\infty]$	$(\min,\infty)$	$(+,0)$	Min-sum
5	$[-\infty,\infty)$	$(\max,-\infty)$	$(+,0)$	Max-sum
6	$\{0,1\}$	$(OR,0)$	$(AND,1)$	Boolean

Table 3.1: Some commutative semirings.

2. The operation $" + "$ is defined as the operation of taking the maximum with $-\infty$ as identity element (i.e. $\max(k,-\infty) = k, \forall k \in K$);
3. The operation $" \cdot "$ is defined as the ordinary addition with 0 as identity element and $k + \infty = \infty, \forall k \in K$;
4. The *distributive law* $\max(a+b, a+c) = a + \max(b,c)$ is always true.

Moreover, suppose that function $g(x)$ can be decomposed into a summation of different functions, that is

$$g(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{r} (\mathbf{x}_i), \mathbf{x_i} \subseteq \mathbf{x} \tag{3.0.4}$$

a *factor graph* is defined as a *bipartite graph*[2] that shows the structure of this summation. In particular a factor graph $FG = \mathbf{F}, \mathbf{x}$ is made up of *variable nodes*, one for each x_i, i.e. $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ and *function nodes*, one for each $F_i(\cdot)$, that is $\mathbf{F} = \{F_1, F_2, \ldots, F_n\}$, where a variable node x_i is connected to the function node F_j if and only if that variable is one of the arguments of that function, i.e. $x_i \in \mathbf{x}_j$.

Consider for example the function $g(x_1, x_2, x_3) = F_1(x_1) + F_2(x_2)$, where $\mathbf{x_1} = \{x_1, x_2, x_3\}$ and $\mathbf{x}_2 = \{x_1, x_2\}$. This structure is shown by the graph of 3.0.2 with *function nodes* $\mathbf{F} = \{F_1, F_2\}$ and *variable nodes* $\mathbf{x} = \{x_1, x_2, x_3\}$ and edge connections represented by set

$$E = \{(F_1, x_1), (F_1, x_2), (F1, x_3), (F_2, x_1), (F_2, x_2)\}. \tag{3.0.5}$$

In the proposed approach, given the set of robots R_n, we suppose each robot $R_i \in R_n$ locally possesses and can control only a function $F_i(x_i)$ and a variable x_i and has knowledge of, and can directly communicate only with its neighboring[3] robots. Our supposition is that $x_i = \{x_1, x_2, \ldots, x_n\}, \forall i$, i.e. *the*

[2]In the mathematical field of graph theory, a bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V , that is U and V are independent sets.

[3]Two robots are neighbors if there is a relationship connecting variables and functions that robots control.

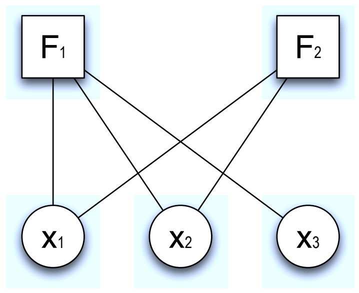


Figure 3.0.2: Factor graph example

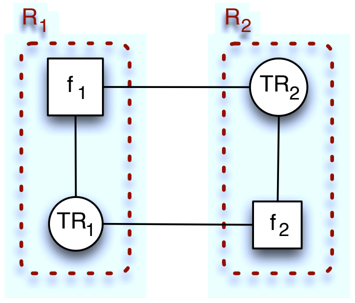


Figure 3.0.3: Robot-controlled factor graph.

factor graph is a *complete bipartite graph*[4], which also means that all robots belong to the same neighboring. A complete factorgraph exchanges a number of messages that is exponential with the number of nodes. For instance in 3.0.3, there are two robots, R_1 and R_2 which respectively possess their pair function-variable (F_1, x_1) and (F_2, x_2) but are neighbors and can communicate each other, because both $\mathbf{x_1}$ and $\mathbf{x_2}$ equal $\{x_1, x_2\}$.

3.0.4 The Max-Sum

The Max-Sum algorithm belongs to the family of iterative *message passing* algorithms called *Generalized Distributive Law* (GDL) [1], which can be combined with factor graphs to efficiently compute functions like $f_j(\cdot)$. Given a set of robots, i.e. $R_1, R_2, \ldots, R_n$, and a factor graph $FG = (\mathbf{x}, \mathbf{F})$ (see an example in Figure 3.0.4) where each robot R_j owns a function f_j and a subset $\mathbf{x_j} \subseteq \mathbf{x}$ of

[4]In the mathematical field of graph theory, a complete bipartite graph is a special kind of bipartite graph where every vertex of the first set is connected to every vertex of the second set.

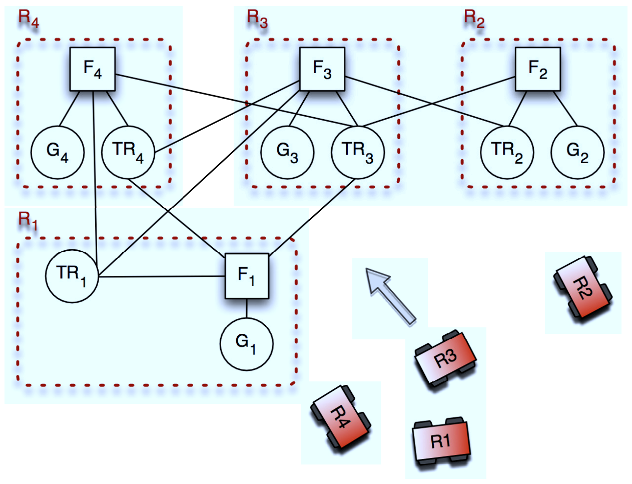


Figure 3.0.4: Example of a simple factor graph, each robot $R_i(i = 1 \dots 4)$, R_2 has the function nodes F_i and the variables nodes G_i , the global goal and TR_i, the list of trajectories.

variables, the max-sum algorithm computes

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \sum_{j=1}^{n} f_j\left(\mathbf{x_j}\right) \tag{3.0.6}$$

by repeatedly passing *variable-to-function* q-messages and *function-to-variable* r-messages.

Let $\mathbf{x}_j$ be a set of trajectories (one for each robot) (i.e $f_4\left(x_4\right)$, $x_4 = \{tr_4, tr_1, tr_3\}$), $f_j\left(\mathbf{x}_j\right)$ calculate a payoff that considers the minimum distance between all possible local goals, logarithmically weighted with the distance to the global goal G_i (see Listing 3.1). In case $\mathbf{x}_j$ would lead to a collision, $f_j\left(\mathbf{x}_j\right)$ is set to an arbitrarily big negative quantity representing $-\infty$. Hence $\mathbf{x}^*$ represents the sequence of robots trajectories that maximize the system utility $\sum_{j=1}^{n} f_j\left(\mathbf{x_j}\right)$, i.e. the local goals whose relative trajectories allow each robot to avoid collisions and to get closer to its global goal at the same time.

After this table filling, the optimized $\mathbf{x}^*$ of Equation (1.4) is achieved by repeatedly passing messages within the *factor graph* (Figure 1.13 shows the messages exchanged between robots R_1 and R_2 in the factor graph Figure 1.11):

1. From variable nodes to function nodes, called *q-messages*;

2. From function nodes to variable nodes, called *r-messages*.

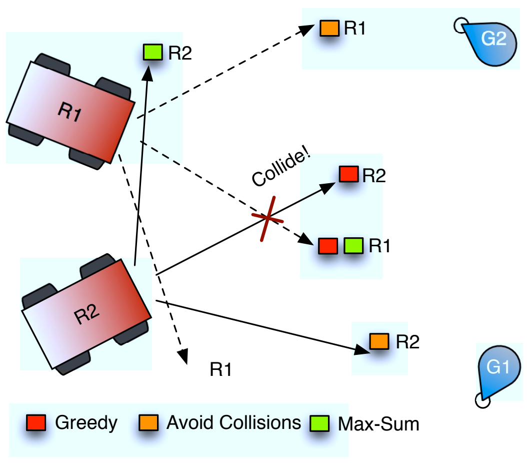


Figure 3.0.5: Comparison between 3 motion planning methods

Formally, a *r-message is*

$$r_{r\to i}\left(x_i\right) = \max_{\mathbf{x_j}\setminus\mathbf{x_i}} \left[F_j\left(\mathbf{x}_j\right) + \sum_{k\in\mathcal{N}_j\setminus x_i} q_{k\to j}\left(x_k\right) \right] \tag{3.0.7}$$

where $\mathcal{N}_j$ is the set of variable indexes, indicating which variable nodes in the *factor graph* are connected to function node F_j and $\mathbf{x}_j\setminus x_i \equiv \{x_k : k \in \mathcal{N}_j\setminus i\}$.

On the other hand a *q-message*

$$q_{i\to j}\left(x_i\right) = \alpha_{ij} + \sum_{k\in\mathcal{M}_i\setminus j} r_{k\to i}\left(x_i\right) \tag{3.0.8}$$

where $\mathcal{M}_i$ is the set of function indexes, indicating which function nodes in the *factor graph* are connected to variable node x_i and α_{ij} is the *message normalization factor* such that $\sum_{x_i} q_{i\to j}\left(x_i\right) = 0$[5].

When the factor graph is cycle free, the algorithm is guaranteed to converge to the global optimal solution $\mathbf{x}^*$, thereby optimally solving the proposed optimization problem. Moreover, this convergence can be achieved by first calculating

$$z\left(x_i\right) = \sum_{j\in\mathcal{M}_i} r_{j\to i}\left(x_i\right) \tag{3.0.9}$$

[5]In cyclic factor graphs such normalization prevents messages to increase endlessly, on condition that there are not negative infinity utilities, which are usually taken into account to represent hard constraints on the solution.

we have trivially called *z-message*[6], and then computing

$$\arg\max_{x_i} z_i\left(x_i\right). \tag{3.0.10}$$

3.0.5 The sense-plan-act cycle

The robot compute the main cycle described as follows:

START

1. SENSE
2. PLAN
    (a) Compute possible local goals and relative trajectories
    (b) Understand if there are neighbors which can collide with myself if not go to ACT entry .
    (c) Ask to all neighbors if the want to perform Max-Sum. If not, go to entry (b).
    (d) Create the local variables and the local function
    (e) Create the factorgraph with the neighborhood.
    (f) Perform Max-Sum and choose the best trajectory.
3. ACT
    (a) Bring the robot to the trajectory velocity.
    (b) Keep moving until the local or global goal is reached.

END

In the (2.b) entry we consider neighbors just the robots which are far from the current robot, less than a d_{max} distance, where d_{max} is the distance to travel by the robot if it should go to the maximum velocity for all the time of the prediction window :

$$d_{max} = v_{max} \cdot T_{window}. \tag{3.0.11}$$

Actually we don't communicate directly with all the neighbors in that distance but only with those that are directly visible by the robot as showed in Figure 3.0.6.

Using this policy we reduce the factorgraph complexity and the number of messages exchanged in the max-sum. When we build the factor graph we

[6]The *z-message* is considered a message because it is made up of the sum of other messages, but at the same time it is judge trivial because it is calculated locally and is not exchanged between robots.

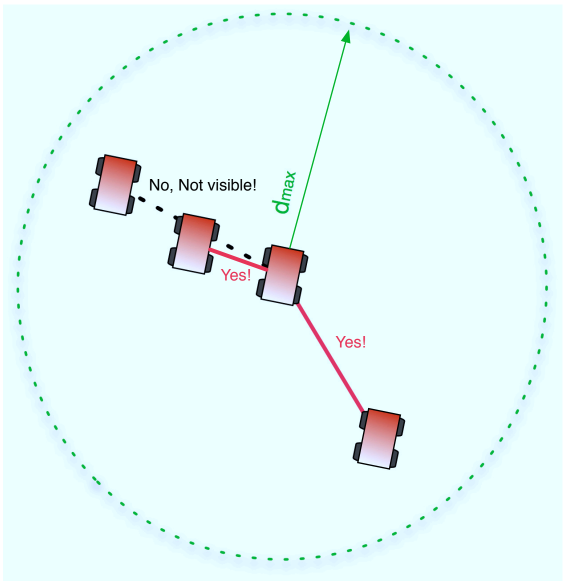


Figure 3.0.6: Line of sight neighbor selection.

exchange the variable values with the neighbors in order to build the local function. Let's see and example of max-sum iteration[7]:

$$
\begin{aligned}
R_1 : \quad & G_1 = (1.0; 5.0)\,, \; sp_1 = (7.0; 1.0) \\
& V_1 = \{0.2\} \rightarrow v_1 = 0.2, \\
& TR_1 = \{\{v_1, sp_1, (5.0; 3.0)\}, \{v_1, sp_1, (7.0; 3.0)\}, \{v_1, sp_1, (9.0; 3.0)\}\}\,. \\
R_2 : \quad & G_2 = (7.0; 5.0)\,, \; sp_2 = (3.0; 1.0) \\
& V_2 = \{0.2\} \rightarrow v_2 = 0.2, \\
& TR_2 = \{\{sp_2, v_2, (0.0; 4.0)\}, \{sp_2, v_2, (3.0; 3.0)\}, \{sp_2, v_2, (5.0; 3.0)\}\}\,.
\end{aligned}
\tag{3.0.12}
$$

The Table Functions will be:

Since at the first iteration[8] *q-messages* $q^0_{i\rightarrow j}(x_i)$are set to 0, consequently *r-messages* become $r^0_{j\rightarrow i}(x_i) = \max\limits_{\mathbf{x_j}/x_i} f_j(\mathbf{x}_j)$, in our example we get:

- $q^0_{i\rightarrow j}(TR_i) = 0, \; i \in \{1, 2\}\,, j \in \{1, 2\}$
- $r^0_{1\rightarrow 1}(TR_1) = \max\limits_{TR_2} f_j(G_1, TR_1, TR_2) = \{-0.5816, -1.1935, -0.9281\}$
- $r^0_{1\rightarrow 2}(TR_2) = \max\limits_{TR_1} f_j(G_1, TR_1, TR_2) = \{-0.5816, -0.9281, -0.5816\}$
- $r^0_{2\rightarrow 1}(TR_1) = \max\limits_{TR_2} f_j(G_1, TR_1, TR_2) = \{-0.5816, -0.1234, -0.1234\}$

[7]For sake of semplicity we consider a velocity set with only one entry.
[8]The messages indicated with the superscript 0.

G_1	TR_1	TR_2	$f_1(G_1, TR_1, TR_2)$
	$\{v_1, sp_1, (5.0; 3.0)\}$	$\{v_2, sp_2, (0.0; 4.0)\}$	-0.5816
	$\{v_1, sp_1, (5.0; 3.0)\}$	$\{v_2, sp_2, (3.0; 3.0)\}$	-0.5816
	$\{v_1, sp_1, (5.0; 3.0)\}$	$\{v_2, sp_2, (5.0; 3.0)\}$	-36.9600
	$\{v_1, sp_1, (7.0; 3.0)\}$	$\{v_2, sp_2, (0.0; 4.0)\}$	-0.9281
	$\{v_1, sp_1, (7.0; 3.0)\}$	$\{v_2, sp_2, (3.0; 3.0)\}$	-0.9281
	$\{v_1, sp_1, (7.0; 3.0)\}$	$\{v_2, sp_2, (5.0; 3.0)\}$	-0.9281
	$\{v_1, sp_1, (9.0; 3.0)\}$	$\{v_2, sp_2, (0.0; 4.0)\}$	-1.1935
	$\{v_1, sp_1, (9.0; 3.0)\}$	$\{v_2, sp_2, (3.0; 3.0)\}$	-1.1935
	$\{v_1, sp_1, (9.0; 3.0)\}$	$\{v_2, sp_2, (5.0; 3.0)\}$	-1.1935

Table 3.2: Function for R_1.

G_2	TR_2	TR_1	$f_2(G_2, TR_2, TR_1)$
	$\{v_2, sp_2, (0.0; 4.0)\}$	$\{v_1, sp_1, (5.0; 3.0)\}$	-1.0397
	$\{v_2, sp_2, (0.0; 4.0)\}$	$\{v_1, sp_1, (7.0; 3.0)\}$	-0.5816
	$\{v_2, sp_2, (0.0; 4.0)\}$	$\{v_1, sp_1, (9.0; 3.0)\}$	-36.9600
	$\{v_2, sp_2, (3.0; 3.0)\}$	$\{v_1, sp_1, (5.0; 3.0)\}$	-1.0397
	$\{v_2, sp_2, (3.0; 3.0)\}$	$\{v_1, sp_1, (7.0; 3.0)\}$	-0.5816
	$\{v_2, sp_2, (3.0; 3.0)\}$	$\{v_1, sp_1, (9.0; 3.0)\}$	-0.1234
	$\{v_2, sp_2, (5.0; 3.0)\}$	$\{v_1, sp_1, (5.0; 3.0)\}$	-1.0397
	$\{v_2, sp_2, (5.0; 3.0)\}$	$\{v_1, sp_1, (7.0; 3.0)\}$	-0.5816
	$\{v_2, sp_2, (5.0; 3.0)\}$	$\{v_1, sp_1, (9.0; 3.0)\}$	-0.1234

Table 3.3: Function for R_2.

	f_1	f_2
TR_1	0.2761	0.9011
TR_2	0.5816	0.6971

Table 3.4: Example of message normalization factors

- $r^0_{2\to 2}(TR_2) = \max_{TR_1} f_j(G_1, TR_1, TR_2) = \{-1.0397, -0.1234, -0.5816\}$

Next we apply Equation 3.0.10 , and we obtain:

- $TR_1 = \arg\max_{TR_1} [\{-0.5816, -1.1935, -0.9281\} + \{-0.5816, -0.1234, -0.1234\}]$

- $TR_2 = \arg\max_{TR_2} [\{-0.5816, -0.9281, -0.5816\} + \{-1.0397, -0.1234, -0.5816\}]$

The algorithm will choose tr_2 for variable TR_1 and the trajectory tr_1 for TR_2.

However the algorithm does not end at the first iteration and immediately computes the second iteration q-messages, which are normalized by setting each α_{ij} according to Algorithm 3.1. For instance we get $\alpha_{21} = 0.5816$ and consequently $q^1_{2\to 1}(TR_2)$ following these steps:

1. Component-wise sum all vectors (lines 2-5);

2. Sum the resulted components (lines 6-8) (0.21 + 0.25 = 0.46);

3. Change sign to the obtained value (line 9) (−0.46);

4. Divide the previous result by the vector cardinality (− 0.46 = −0.23) (line 10);

Then if we add α_{21} to each $r_{2\to 2}(TR_2)$ component we get $\{-0.4581, 0.4581, \epsilon\}$. Therefore if the same procedure (Table 3.4 summarize the computed message normalization factors) is applied to other messages we also obtain:

- $q^1_{1\to 1}(TR_1) = \{-0.3054, 0.1527, 0.1527\}$

- $q^1_{1\to 2}(TR_2) = \{0.3195, -0.2924, -0.0271\}$

- $q^1_{2\to 2}(TR_2) = \{0.1155, -0.2310, 0.1155\}$

At this point the algorithm has the required messages to computed the relative *r-messages*, whose evaluation is left to the reader. The most important fact is that the *z-messages* obtained using the $r^1_{j\to i}(TR_i)$ messages make the system evaluate $TR_1 = tr_2$ and $TR_2 = tr_1$ again, which means the algorithm has converged to a solution, ending its execution.

Algoritmo 3.1 Message normalization factor evaluation

```
computeAlpha(RMessages)
  messageQ messQ
  foreach message in RMessages
    messQ .+ message
  foreach value in messQ
    alpha += value
  alpha *= −1
  alpha /= messQ.size()
  return alpha
```

3.1 Bounded Max-sum algorithm

The *Max-Sum* algorithm is extremely attractive for the decentralized coordination of computationally and communication constrained devices for the following reasons:

1. The messages are small and scale with the domain of the variables;
2. The number of messages exchanged typically varies linearly with the number of agents within the system;
3. The computational complexity of the algorithm scales exponential with just the number of variables on which each function depends.

However, the lack of guaranteed convergence and guaranteed solution quality, limits the use of the standard *Max-sum* algorithm in many application domains, and such exponential computation behavior is undesirable in systems composed of devices with constrained computational resources. The *Bounded Max-sum*, a variation of the *Max-sum algorithm*, whose approach removing cycles from the *factor graph*, by ignoring some of the dependencies between functions and variables, ensures the convergence of the algorithm to a bounded approximate solution. Since our proposed approach could work with complete factor graphs, the bounded version should be the candidate approach. Nevertheless, due to the characteristics of the proposed environment and optimization problem, we choose to adopt and implement a system which uses the simple version of the algorithm. Our system has real time constraints so we are not interested to the best optimal solution, but to a optimized solution. We prefer to spend time to compute max-sum iterations instead of using the time to reduce the factorgraph and start the computation after that even if the instance does not converge.

Chapter 4

The Social Force Model and the DSFM architecture

In the previous chapters we discussed about how to avoid collisions among connected robots with the same behavior. What about the human aware policy? Our final objective is an environment where humans and robots can move together. Below we present the Social Force Model and how to use it in our robot controller.

4.1 The Social Force Model

The social force model by Helbing [12, 13] is a computational model in which the interactions between pedestrians are described by using the concept of a social force. It is based on the idea that changes in behavior can be explained in terms of social fields or forces. Applied to pedestrians, the social force model accounts for the influence of the environment and other people and describes how the intended direction of motion changes as a function of these influences. The model does not cover cases of multiple options, when people have to actively decide. Game theoretic approaches can be applied in such situations.

Formally, the models assumes that a pedestrian p_i with mass m_i likes to move with a certain intended velocity $\widehat{\mathbf{v}}_i$ in an intended direction $\widehat{\mathbf{e}}_i$ and therefore tends to adapt his or her velocity v_i within a so called relaxation time τ_i. This change of velocity is modeled by the personal motivation

$$\mathbf{F}_i^{pers} = m_i \frac{\widehat{\mathbf{v}}_i \widehat{\mathbf{e}}_i - v_i}{\tau_i}. \tag{4.1.1}$$

The relaxation time is the time interval needed to reach the intended velocity and the intended direction. In the presence of other people or objects in the environment, a pedestrian might not be able to keep the intended direction and velocity. In the social force model, repulsive effects from these influences are described by an interaction force $\mathbf{F}_i^{soc}$. This force prevents humans from walking along their intended direction and is modeled as a sum of forces either introduced by other individuals p_j or by static obstacles denoted by subscript o

$$\mathbf{F}_i^{soc} \sum_{j \in \mathcal{P} \setminus \{i\}} \mathbf{f}_{i,j}^{soc} + \sum_{o \in \mathcal{O}} \mathbf{f}_{i,o}^{soc} \tag{4.1.2}$$

with $\mathcal{P} = \{p_i\}_{i=1}^{N_\mathcal{P}}$ being the set of all people and $\mathcal{O}$ the static objects of the environment. These forces decrease proportional to the distance of their sources and are modeled as

$$\mathbf{f}_{i,k}^{soc} = a_k e^{\left(\frac{r_{i,k} - d_{i,k}}{b_k}\right)} \mathbf{n}_{i,k} \tag{4.1.3}$$

where $k \in \mathcal{P} \cup \mathcal{O}$ is either a person or an object of the environment, a_k specifies the magnitude and b_k the range of the force. In order to calculate the Euclidean distance between p_i and entity k, pedestrians and objects are assumed to be of circular shape with radii r_i and r_k,respectively. Then, distance $d_{i,k}$ is given by the Euclidian distance between the centers, and $r_{i,k}$ is the sum of their radii. The term $n_{i,k}$ is the normalized vector pointing from k to p_i which describes the direction of the force. Given the limited field of view of humans, influences might not be isotropic. This is formally expressed by scaling the forces with an anisotropic factor

$$\mathbf{f}_{i,k}^{soc} = a_k e^{\left(\frac{r_{i,k} - d_{i,k}}{b_k}\right)} \mathbf{n}_{i,k} \left(\lambda + (1 - \lambda) \frac{1 + \cos(\varphi_{i,k})}{2} \right) \tag{4.1.4}$$

where λ define the strength of the anisotropic factor and

$$\cos(\varphi_{i,k}) = -\mathbf{n}_{i,k} \cdot \widehat{\mathbf{e}_i}. \tag{4.1.5}$$

Human motion is not only influenced by personal motivation and reactive behavior towards obstacles or other people but is physically constrained by the environment [15]. Hard constraints restrict the motion and thereby define the walkable area of the environment. Therefore, the social force model introduces a physical force

$$\mathbf{F}_i^{phys} = \sum_{j \in \mathcal{P} \setminus \{i\}} \mathbf{f}_{i,j}^{phys} + \sum_{o \in \mathcal{O}} \mathbf{f}_{i,o}^{phys} \tag{4.1.6}$$

$$\mathbf{f}_{i,k}^{phys} = c_k g\left(r_{i,k} - d_{i,k}\right) \mathbf{n}_{i,k}, \tag{4.1.7}$$

where c_k represents the magnitude of the exerted force. To make the physical forces a real contact force where the circular shapes of p_i and k overlap, the function g is defined as $g(x) = x$ if $x \geq 0$ and 0 otherwise. Finally, human motion is explained by the superposition of all exerted forces. Accordingly, the force $\mathbf{F}_i$ changing the motion of individual p_i is

$$\mathbf{F}_i = \mathbf{F}_i^{pers} + \mathbf{F}_i^{soc} + \mathbf{F}_i^{phys} \tag{4.1.8}$$

Using $\mathbf{F}_i$, the basic equation of motion for a pedestrian is then of the general form

$$\frac{d}{dt} v_i = \frac{\mathbf{F}_i}{m_i} \tag{4.1.9}$$

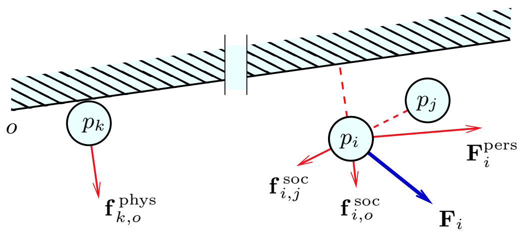

o
p_k
$\mathbf{f}_{k,o}^{\text{phys}}$
p_j
p_i
$\mathbf{F}_i^{\text{pers}}$
$\mathbf{f}_{i,j}^{\text{soc}}$
$\mathbf{f}_{i,o}^{\text{soc}}$
$\mathbf{F}_i$

Figure 4.1.1: Forces in the social force model shown as red arrows. Pedestrian p_k is affected by the physical force of the static part of the environment. Pedestrian p_i is both affected by the wall and an other person p_j. The superposition of forces exerted to p_i is shown as the blue arrow $\mathbf{F}_i$

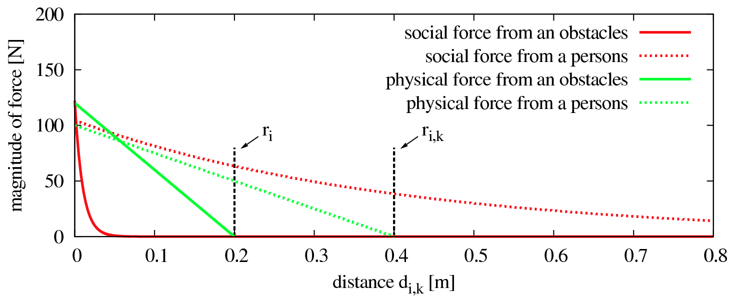


Figure 4.1.2: Typical functions for the exerted forces. The x-axis shows the distance from person p_i to an object o or a person p_k and the y-axis the magnitude of the forces in Newton. The radius of p_i is $r_i = 0.2m$ and the sum of the radii of p_i and p_k is $r_{i,k} = 0.4m$.

and describes the movements of p_i over time. An illustration of all forces is shown in Fig. 4.1.1. The physical force $\mathbf{f}_{k,o}^{phys}$ that the wall exerts onto person p_k is shown. This avoids motion predictions through walls. A superposition of different forces onto pedestrian p_i is also shown. The person wants to keep his or her intended velocity through the motivation $\mathbf{F}_i^{pers}$ but also is influenced by $\mathbf{f}_{i,j}^{soc}$ from person p_j and by $\mathbf{f}_{i,o}^{soc}$ from the wall. This result in the superimposed force $\mathbf{F}_i$ used to adapt the velocity of p_i .

4.2 Social Force Model as reactive approach

The DWA approach explained in Section 2.3.1 can easily plan the robot motions. It has a low computational load as well. The weakness of that method is the inability to distinguish between the type of obstacles. It computes the next velocity just considering the current robot and environment state: no *intelligent* data are used by the system.

In this work we apply the social forces directly to the robot enhancing the Dynamic Approach. The SFM has been introduced in computer vision algorithms for people tracking in order to predict the trajectories of the tracked people. The system here proposed uses the people tracker presented in [8]. In our model the local goal is represented by the personal force $\mathbf{F}^{pers}$, where the intended velocity

$$\widehat{\mathbf{v}}\left(t\right)=\begin{cases}\min\left\{v,\frac{\|\mathbf{g}(t)-\mathbf{x}(t)\|}{T_{max}-dt}\right\} & dt<T_{max}\\ v & otherwise\end{cases}, \tag{4.2.1}$$

and the intended direction is

$$\widehat{\mathbf{e}}\left(\mathbf{t}\right)=\frac{\mathbf{g}\left(t\right)-\mathbf{x}\left(t\right)}{\|\mathbf{g}\left(t\right)-\mathbf{x}\left(t\right)\|}. \tag{4.2.2}$$

In 4.2.1 and 4.2.2, $\mathbf{g}\left(t\right)$ is the current local goal and $\mathbf{x}\left(t\right)$ the current robot position. We introduce the dt parameter which is the duration between the goal choice and when the velocity is computed. That interval has an upper bound T_{max} that represent the maximum time patience of the robot. When this time is reached the personal force will reach the maximum of the magnitude.

The new waypoint $\mathbf{x}\left(t+1\right)$ used by the DWA approach will be

$$\mathbf{x}\left(t+1\right)=\mathbf{x}_t+\mathbf{v}_tT_{cycle}+\frac{1}{2}\frac{\mathbf{F}}{m}T_{cycle}^2 \tag{4.2.3}$$

where T_{cycle} is the robot sense-plan-act cycle time. The waypoint processed by the SFM will be probably more safe.

This model has another key feature: the estimation of future people motions. Information about the people around the robot, will be added to the obstacles region of the DWA as explained below.

4.2.1 Motion prediction using social forces

The SFM can be combined with a Kalman filter based tracker to result in a more precise prediction model of human motion.

Let $\mathbf{x}_t=\left(x_t,y_t,\dot{x}_t,\dot{y}_t\right)^T=\left(\mathbf{x}_t,\mathbf{y}_t\right)^T$ be the state of the pedestrian p_i at time t and Σ_t its 4×4 covariance matrix estimate. The term $\mathbf{x}_t$ represents the position and $\mathbf{v}_t$ the velocity of the pedestrian in Cartesian space. We will skip the subscript i for sake of simplicity. The constant velocity motion model is defined as

$$p\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right)=\mathcal{N}\left(\mathbf{x}_t;A\mathbf{x}_{t-1},A\Sigma_{t-1}A^T+Q\right), \tag{4.2.4}$$

with A being the state of the transition matrix. The entries of Q represent the acceleration capabilities of a human. Using a discrete time approximation of Eq.

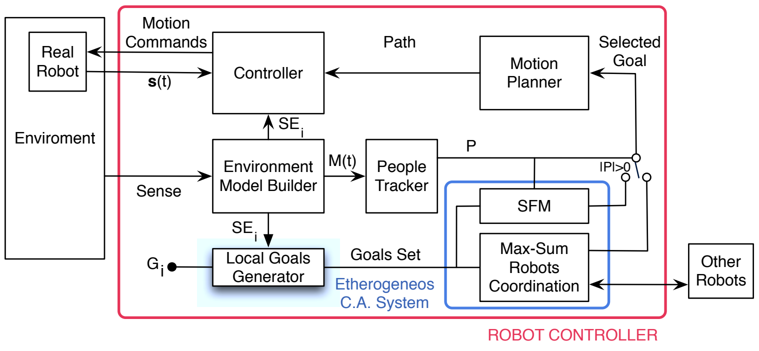


Figure 4.3.1: The Distributed Multi Robot System with the Social Force Model (DSFM). In the blue square the hybrid collision avoidance structure.

4.1.9 within a fixed interval of time $\triangle t = T_{cycle}$ to obtain $\mathbf{x}_t = \xi\left(\mathbf{x}_{t-1}, \mathcal{P}, \mathcal{O}\right)$, where

$$\xi\left(\mathbf{x}_{t-1}, \mathcal{P}, \mathcal{O}\right) = \begin{bmatrix} \mathbf{x}_{t-1} + \mathbf{v}_{t-1}\triangle t + \frac{1}{2}\frac{\mathbf{F}}{m}\triangle t^2 \\ \mathbf{v}_{t-1} + \frac{\mathbf{F}}{m}\triangle t \end{bmatrix} \tag{4.2.5}$$

describes how the motion of a pedestrian evolves over time. The change in motion is calculated according to the pedestrian's intended velocity, reactive behavior from interaction forces and physical constraints from the environment, according to Eq. 4.1.8. Assuming that the motion is affected by Gaussian noise with zero mean and covariance matrix Q we have

$$p\left(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathcal{P}, \mathcal{O}\right) = \mathcal{N}\left(\mathbf{x}_t; \xi\left(\mathbf{x}_{t-1}, \mathcal{P}, \mathcal{O}\right), J_\xi \Sigma_{t-1} J_\xi^T + Q\right), \tag{4.2.6}$$

where $J_\xi = \frac{\partial \xi(\cdot)}{\partial \mathbf{X}}$ is the Jacobian of $\xi\left(\cdot\right)$ evaluated at $\mathbf{x}_{t-1}$.

Without external stimuli and deviation from the intended direction and preferred velocity, **F** is zero, no social forces are applied to the pedestrian and the social force model falls back onto the constant velocity motion model.

4.3 DSFM: the hybrid architecture

As stressed, the final collision avoidance system has to be able to compute a kinodynamic motion plan in an environment populated with humans and robots.

Obviously, the nature of the actors is totally different. Robots cannot exchange complex information with people. The human aware policy requires a reactive approach able to predict human behavior. On the other hand, we cannot just use a reactive method as explained in the Subsection 2.2.3. The robot could not be able to avoid an ICS state. Our proposed hybrid architecture is represented in Figure 4.3.1. The collision module uses different approaches when the robot perceives people, robot, or both. The robot controller mechanism has been changed(with respect to the one in Figure 2.0.1) as follow:

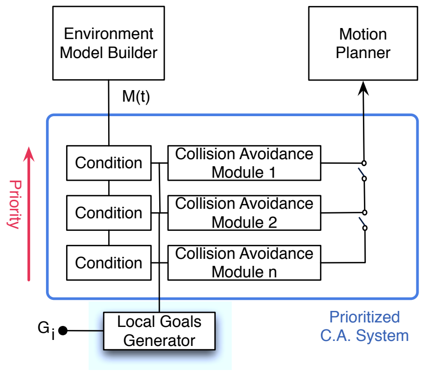

Environment Model Builder
Motion Planner
M(t)
Priority
Condition
Collision Avoidance Module 1
Condition
Collision Avoidance Module 2
Condition
Collision Avoidance Module n
G_i
Local Goals Generator
Prioritized C.A. System

Figure 4.3.2: The prioritized collision avoidance module.

1. The *Environment Model Builder* retrieves sensor and odometry data and computes a *costmap*, a discrete grid inflated with costs (e.g. distance from an obstacle) obtained from the environment sensor data;

2. Given the global goal and the SE, the *Local Goals Generator* computes a set of feasible[1] goals around the robot position.

3. The local goals set is sent to the Collision Avoidance Module (CAM) which route this set to the various sub modules.

4. At the same time the people tracker recognize the presence or not of humans. If there is any, a trigger blocks the robots coordination because unknown obstacles are coming. In this case, the Social Force Model Planner will compute the next waypoint for the robot.

5. The *Motion Planne*r computes the shortest path to reach it using the Dynamic Window Approach.

6. The *Controller* receives the path and starts to send the motion commands to the real robot using a reactive approach(see 2.3.1) until it does not reach the local goal

The double collision avoidance architecture takes the name of DSFM (Distributed + SFM).

The advantage of an heterogeneous collision avoidance module is that we can add other sub-modules which can be triggered when an event is reached. Each CA policy has a condition block that decide if to activate the CAM. The design is represented in Fig. 4.3.2 and use an interrupt-like system. When multiple CAMs are called, the one with the highest priority disarms its own trigger. This action will exclude the plans proposed by lower priority CAMs.

[1]Feasible means that for sure there is a path between robot position and the goal.

Chapter 5

System implementation in ROS

The middleware ROS (*Robot Operating System*) makes available libraries and tools to help software developers to create robot applications, e.g. hardware abstraction, device drivers, visualizers, message-passing and package management. ROS is organized in software packages with binary nodes, which are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control package will usually comprise many nodes. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, one node provides a graphical view of the system, and so on. A ROS node is written with the use of the ROS c++ or python client libraries.

Nodes communicate with each other by passing messages, ,even if they are in different packages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily

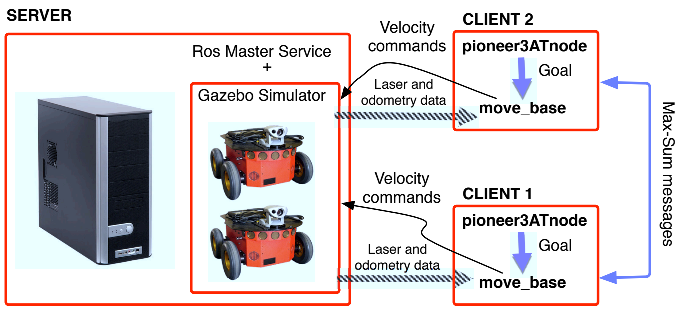


Figure 5.0.1: System setup on our ROS implementation.

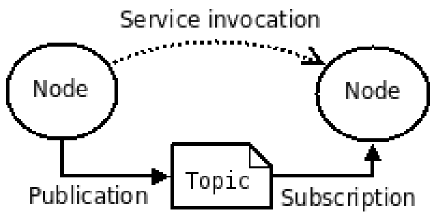


Figure 5.0.2: Communication modes in ROS

nested structures and arrays (much like C structs). These messages can be exchanged thanks to asynchronous modes (see Figure 5.0.2):

- *Topics*: Messages are routed via a transport system with publish / subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each others' existence. The idea is to decouple the production of information from its consumption. Logically, one can think of a topic as a strongly typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages as long as they are the right type.

- *Services*: The publish / subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.

Nodes use topics and services by a peer-to-peer connection, but all the nodes have to communicate with the *Master service* to enable all the connection. It provides name registration and lookup, without the Master, nodes would not be able to find each other, exchange messages, or invoke services.

5.1 ROS System Schema

The ROS network architecture highly favors a distributed arrangement. In the our developed package the executable node, called *pioneer3AT* node, has three main tasks:

- communicate with other "max-sum" robots;
- compute the new node goal;
- strictly collaborate with *move_base*, the other ROS node used in this system.

Given the footprint of the robot, environmental data from on-board sensors (e.g. laser, stereoscopic camera) and the odometry, the *move_base* node has the objective of computing the velocities and the steering angles of the robot in order to reach a goal communicated by the pioneer3AT node. *Move_base* is tuned for avoid the static obstacles by using the Dynamic Window Algorithm (see 2.3.1). This node also publishes a costmap of the local environment built from the sensor data. This is used to compute the path toward the local goal by using the Dijkstra's algorithm. Moreover, we do our test using virtual worlds builded for the *gazebo simulator* [1]. The gazebo APIs permit the modeling and the developing of a virtual pioneer3AT, which can read the velocity command computed by the move_base node.

The strength of this modular approach is the reusability of the code and its portability. In fact, the pioneer3AT node source code can be easily adapted and executed on a group of real robots even different from the pioneer3AT.

5.2 Gazebo overview

Gazebo is a multi-robot simulator able to simulate a population of robots, sensors and objects, it does so in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid-body physics).

Gazebo implementation is monolithic that glues together a physics engine (ODE), a rendering engine (OGRE), and a GUI (wxWidgets). User access to sensors and controllers are provided via a shared memory interface. Two threads run in Gazebo. The first manages the GUI and rendering engine, and the second thread manages the physics engine.

The worlds rendered in gazebo are described by a XML document where you can use external 3D objects (e.g. meshes). The ROS version of gazebo provides some key-features:

- ROS service for loading robot models defined using the URDF language;
- Services whose permit to move directly joints and links resident in the gazebo simulating.

When we load into gazebo a URDF model using ROS there are 2 ways to interact with it: the gazebo node and the gazebo dynamic plugin.

[1]This tool is directly available in ROS.

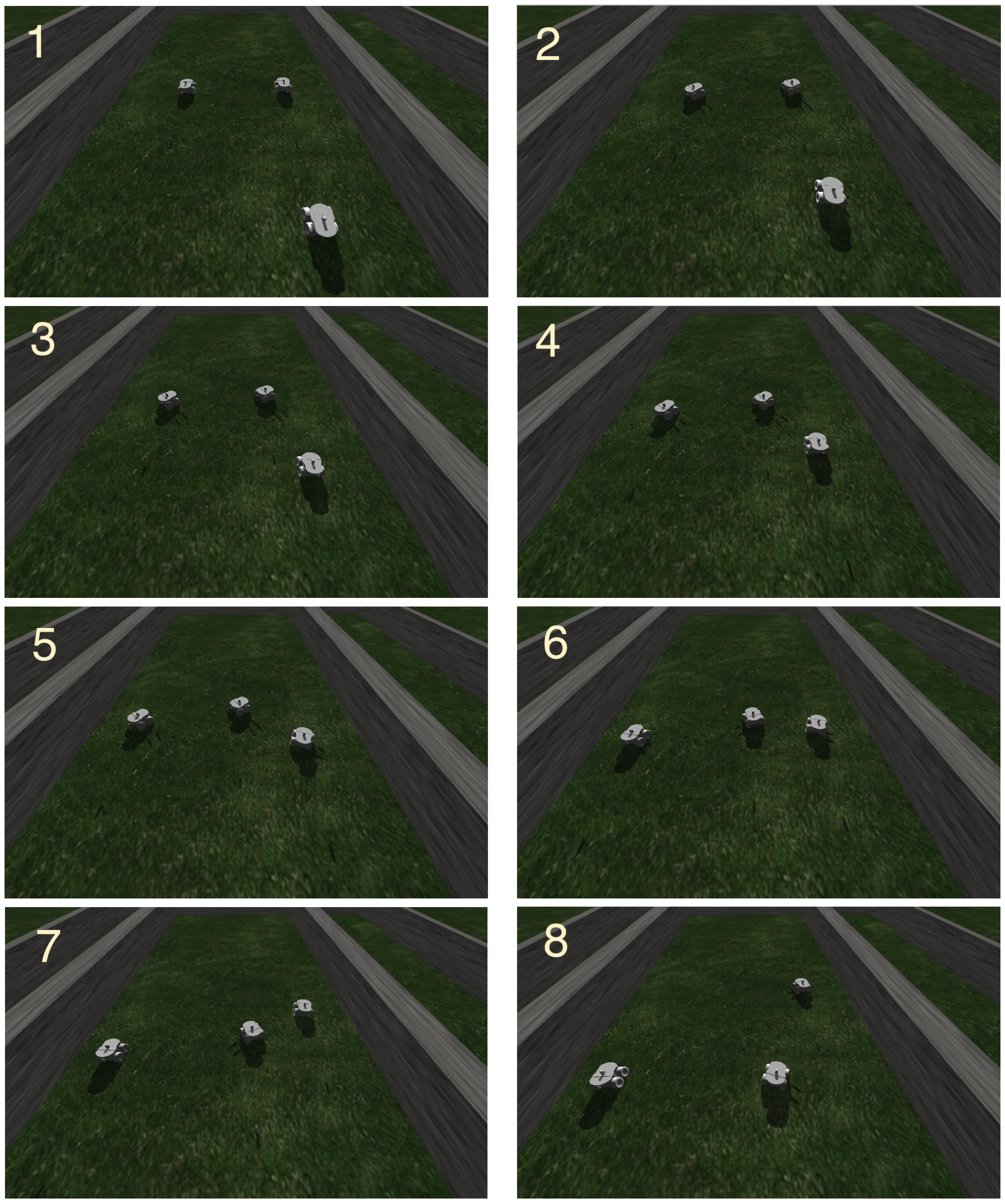


Figure 5.2.1: A three robots Max-sum trial on the Gazebo simulator.

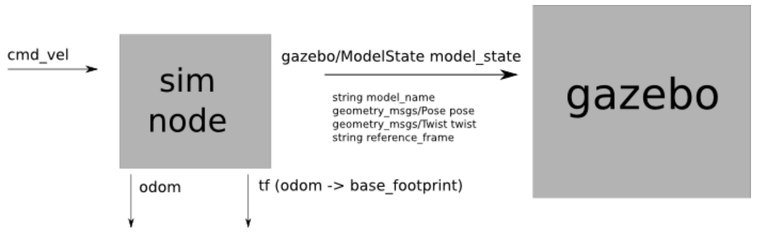


Figure 5.2.2: Standalone node schema

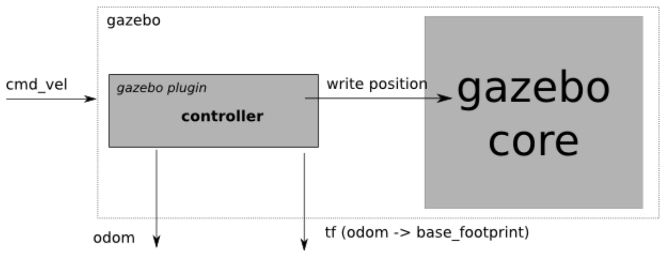


Figure 5.2.3: Standalone node schema

5.2.1 Standalone simulator node outside GAZEBO

This approach consists on writing a generic ROS node which will be able to communicate with GAZEBO using the gazebo/model_state topic/services. The node developed will receive the twist velocity message, simulate odometry (publishing it via ROS topic) and finally generate and send the proper ModelState message to GAZEBO in order to make the robot move.

5.2.2 GAZEBO dynamic plugin

This is a different approach to the robot simulator that gives better results when using this method than the standalone simulator node. The better performance derives from the fact that more data exchanges (e.g. joint position) are done inside gazebo. So the data are moved within the same program space, these are more rapid than the external communications.

This method consists on writing a GAZEBO plugin, a controller that can be linked into the URDF robot file to offer the simulator the control capability. The plugin will use the GAZEBO and ROS code to perform the simulation actions.

5.3 The Pioneer 3AT model

Here it's briefly described the Pioneer 3AT model developed in [17], after it's given a high vision of the gazebo implementation of the robot model.

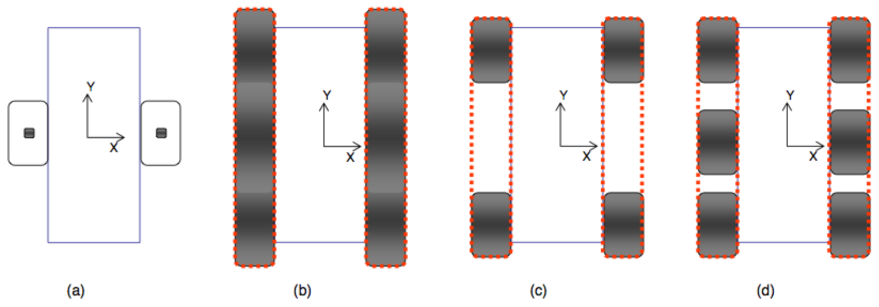


Figure 5.3.1: Left and right treads on the plane for differential drive (a), tracked vehicle (b), four-wheel skid-steer vehicle (c), and six-wheel skid- steer vehicle (d).

5.3.1 Pioneer 3AT physical model

The Pioneer 3AT is a four-wheeled skid-steering, this kind of traction is based on controlling the relative velocities of the left and right side drives, similarly to differential drive wheeled vehicles. In fact, each side's tread consists of several contact patches that correspond to mechanically linked wheels, as those within the dashed lines in Figure 5.3.1 c-d. However, since all wheels are aligned with the longitudinal axis of the vehicle, turning requires wheel slippage. This locomotion system functions like that of a tracked vehicle.

Skid-steering kinematics is not straightforward, since it is not possible to predict the exact motion of the vehicle only from its control inputs. Thus, pure rolling and no-slip assumptions considered in kinematic models for non-holonomic wheeled vehicles do not apply in this modeling. The local frame of the vehicle is assumed to have its origin on the center of the area defined by the left and right contact surfaces on the plane, and its Y axis is aligned with the forward motion direction. Skid-steer vehicles are governed by two control inputs: the linear velocity of its left and right treads with respect to the robot frame (V_l, V_r). Then, direct kinematics on the plane can be stated as follows

$$\begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix} = f_d \begin{pmatrix} V_l \\ V_r \end{pmatrix} \tag{5.3.1}$$

where $v = (v_x, v_y)$, is the vehicle's translational velocity with respect to its local frame, and ω_z is its angular velocity.

When turning, the Instantaneous Center of Rotation (ICR) of the vehicle on the motion plane is expressed in local coordinates as $ICR_v = (x_{ICR_v}, y_{ICR_v})$, as shown in Fig. 5.3.2. Besides, the ICRs for the left and right treads can be defined in the local frame as $ICR_l = (x_{ICR_l}, y_{ICR_l})$ and $ICRr = (x_{ICR_r}, y_{ICR_r})$, respectively. These result from the composition of the motion of the vehicle and that of linear tread velocity (i.e., V_l or V_r). It is known [18] that ICR_l and ICR_r lie on a line parallel to the local X axis that also contains ICR_v. Note

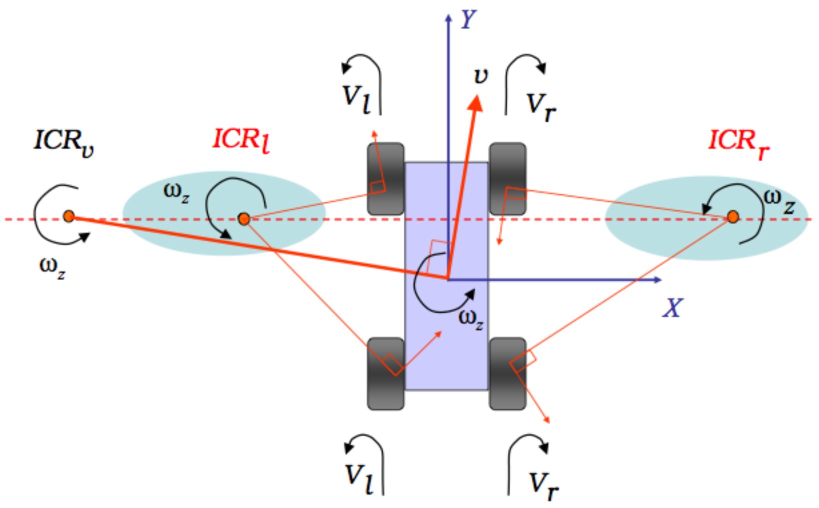


Figure 5.3.2: Vehicle and thread ICRs on the plane.

that treads have the same angular velocity ω_z as the whole vehicle.

Thus, the geometrical relation between ICR positions and the vehicle's translational and rotational velocities is expressed by:

$$x_{ICR_v} = \frac{-v_y}{\omega_z} \tag{5.3.2}$$

$$x_{ICR_l} = \frac{\alpha_l \cdot V_l - v_y}{\omega_z} \tag{5.3.3}$$

$$x_{ICR_l} = \frac{\alpha_r \cdot V_r - v_y}{\omega_z} \tag{5.3.4}$$

$$y_{ICR_v} = y_{ICR_l} = y_{ICR_r} = \frac{v_x}{\omega_z} \tag{5.3.5}$$

where the nominal tread speeds have been affected by correction factors (α_l, α_r) to account for a number of fuzzy mechanical issues such as tire inflation conditions or the transmission belt tension.

From eq. 5.3.2-5.3.5 the kinematic relation can be obtained as

$$\begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix} = A \cdot \begin{pmatrix} V_l \\ V_r \end{pmatrix} \tag{5.3.6}$$

where the elements of matrix A only depend on tread ICR coordinates and correction factors:

$$A = \frac{1}{x_{ICR_r} - x_{ICR_l}} \cdot \begin{bmatrix} -y_{ICR_v} \cdot \alpha_l & y_{ICR_v} \cdot \alpha_r \\ x_{ICR_r} \cdot \alpha_l & -x_{ICR_l} \cdot \alpha_r \\ -\alpha_l & \alpha_r \end{bmatrix} \tag{5.3.7}$$

Control inputs can be obtained from 5.3.6 and 5.3.7 as:

$$V_l = \frac{v_y + x_{ICR_l} \cdot \omega_z}{\alpha_l} \tag{5.3.8}$$

$$V_r = \frac{v_y + x_{ICR_r} \cdot \omega_z}{\alpha_r} \tag{5.3.9}$$

where v_x references cannot be directly addressed due to the non-holonomic restriction of the locomotion system.

5.3.2 Pioneer 3AT implementation in gazebo and its ROS interfacing

When the robot is simulated there's no engine inside, so the center of gravity is in the same position of the geometric center of the robot. This is the case of an ideal symmetrical kinematic model (i.e., ICRs lie symmetrically on the local X axis and $y_{ICR_v} = 0$), matrix A takes the following form

$$A = \frac{\alpha}{2x_{ICR}} \cdot \begin{bmatrix} 0 & 0 \\ x_{ICR} & x_{ICR} \\ -1 & -1 \end{bmatrix} \tag{5.3.10}$$

where $\alpha = \alpha_l = \alpha_r$ and $x_{ICR} = -x_{ICR_l} = x_{ICR_r}$.

The implementation of pioneer 3AT gazebo model consists, besides the graphical and physical design, on developing three plugins that permit to the simulated pioneer to communicate directly with ROS without passing by the common gazebo node as explained in 5.2.2. The plugins are :

- *gazebo_ros_pioneer3AT_p3d*: it publishes the robot current position on the map, the robot odometry is calculated as the transform from the robot frame (positioned and fixed in center of the robot body) to a common frame positioned on the origin of the gazebo simulated map;
- *gazebo_ros_pioneer3AT_laser*: using the robot odometry, the plugin publishes the laser data of the environment around it in a topic;
- *gazebo_ros_pioneer3AT_diffdrive*: this plugin subscribes to a topic where are posted the commands (v_x, v_y, ω_z) and, using the mathematical model illustrated above (5.3.1) , it computes the linear velocities V_l, V_r and transform it in the angular velocity for the joints connected to the wheels.

The publishing of the robot frame is done by the *robot_state_publisher* node, which takes in input the URDF model and gives out the position of all the frames of the robot joints. The positions published are updated with a frequency of 10 Hz but in our work it is not necessary to know the position of each joint(i.e. we don't care about the position of the wheel). Thus, a custom *robot_state_publisher* has been designed that publishes only the positions useful for the odometry and the localization, reducing the system load.

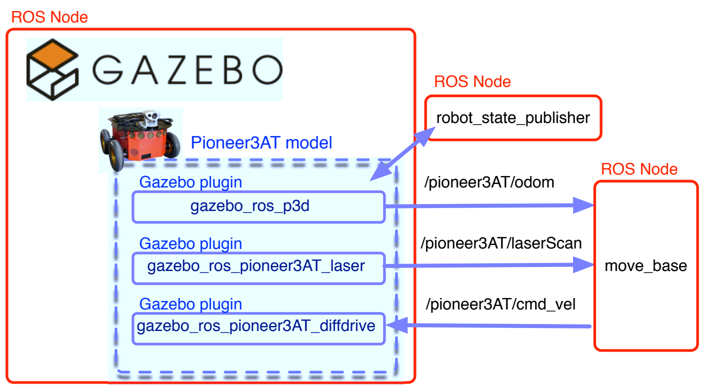


Figure 5.3.3: The Pioneer 3AT ROS node architecture.

5.4 Max-Sum communication implementation

The Max-Sum module is divided in two levels represented mainly by two classes: MaxSum and DMailMan (see Figure 5.4.1). The first one has to accomplish to the high-level operations: compute, build the $\{q, r, z\} - messages$ and evaluate the optimal solution. The robots performing the Max-Sum need to exchange with others:

- *q-messages*: these messages are sent from the variable nodes to the function nodes;
- *r-messages*: these messages are sent from the function nodes to variable once;
- *z-messages*: messages exchanged among robots to understand if everyone is arrived to same solution;
- the values of the variable nodes used in the part of factorgraph that interests the robot.

this exchanges are managed by the DMailMan class. The class will publish for each local variable[2] a topic where the their values of those are published. Thus, each robot can benefit of the variables managed by the other robots without an explicit requests. It has just to subscribe to the topic of the variable and when the values has changed, a callback is launched that will store the new values locally. Using this policy we can avoid the problem of asking data when probably the net is stressed by the message exchanging. The messages are exchanged using a ROS service (one for each robot) called Mailbox. It stores in buffers the messages before they are sent and after they have been received, improving the network performances.

[2]The variables created, managed and updated by the robots are called locals.

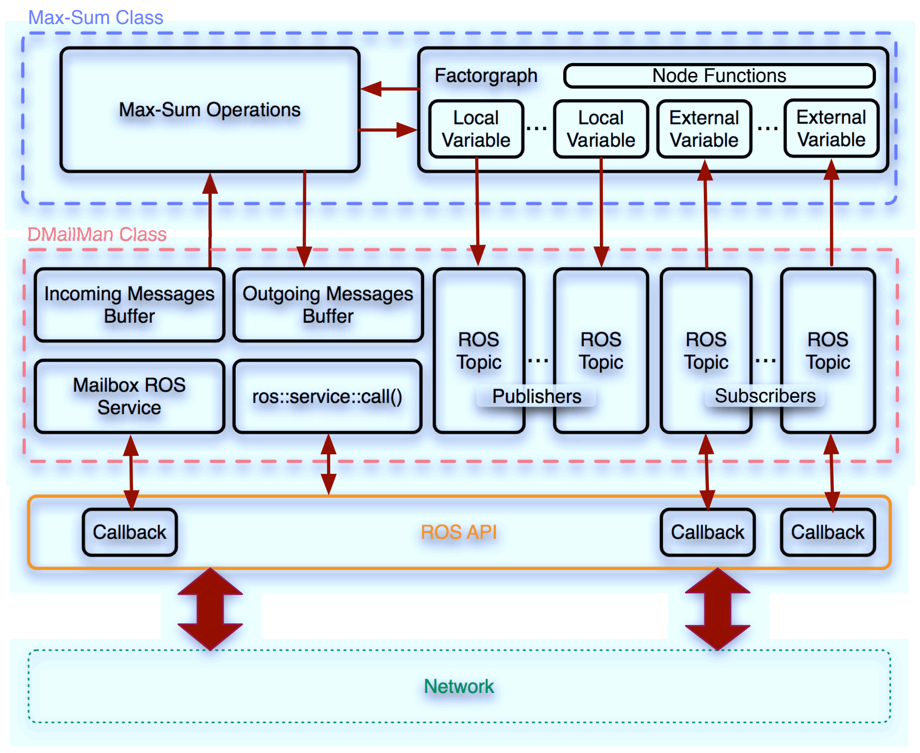


Figure 5.4.1: The Max-Sum communication software architecture

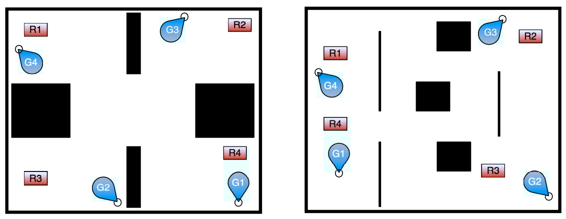
Chapter 6

Experiments and results

6.1 Stress tests: find the minimal requirements

This first series of tests on our collision avoidance approach has been focused on the system payload of the planning cycle times. The workbench was a workstation[1] where we simulated the robots as different ROS nodes. We used two different scenarios created for the Gazebo environment. In the first one (see Figure 6.1.1a), robots have to reach their global goals respectively placed on their opposite corner. In the second scenario (see Figure 6.1.1b) the placement of walls, blocks and goals has been changed for creating a labyrinth. All the simulations involved on the tests have a key topic: all the nodes share the machine memory and its processors, hence increasing the number of robots results in a reduction of resources that they can own. In the last column of Table 6.1 we estimate how much RAM and number of CPUs are available for each node: we can consider these resources like a virtual on board robot computer with low resources. Moreover, we consider also the worst communication case, where

[1]This machine is equipped with an Xeon 3.10 GHz quad-core processor and 3.8 GiB DDR3 RAM memory.



(a) Crossroads. (b) Labyrinth.

Figure 6.1.1: Scenarios with robots R_1, R_2, R_3, R_4 and global goals G_1, G_2, G_3, G_4.

Table 6.1: Total algorithm execution times (s) in the crossroads scenario with different number of robots and different $|GL_i|$, $*$: outlier data due to RAM memory swapping.

Robots	**Number of local goals**				**Resources per robot**
Units	**4**	**6**	**8**	**12**	**RAM, CPU #**
2	0.15	0.28	0.37	0.43	1.5 GiB, 2
3	0.48	1.32	107.49*	65.89	1.0 GiB, 1
4	2.02	29.52	-	-	0.7 GiB, 1

factor graphs are complete[2] so every robot has to exchange messages with all other robots. This case could happen for example, in the first scenario, when all the robots are closed on the center of the environment and they need to share their own paths with all other players. Tests have shown the important role played by the number of candidate goals computed in each planning cycle: the statistics show that if this number decreases under the 6 units, collisions happen on almost the 100% of the tests. As shown on Table 6.1, we can deduce that a real-time onboard robot computer needs 1 CPU with 1 GiB RAM at least.

6.2 Models performances

In this section we will report an analysis varying some parameters of the proposed models: Random Model, Social Force Model, Distributed Model. All the tests will be performed in two main environments:

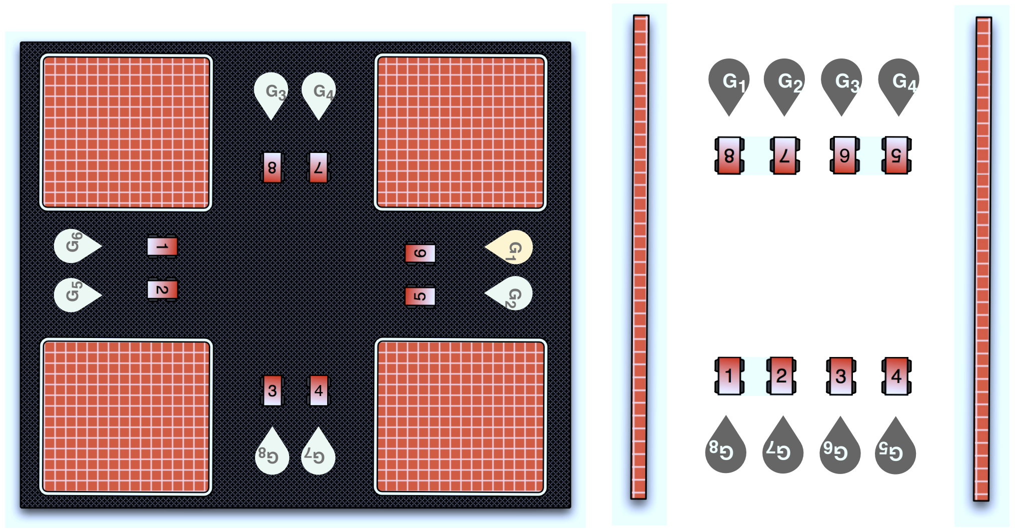
1. A corridor where the same number of robots start from opposite sides and have the goal in the other corridor end (see Figure 6.2.1b).

2. An intersection where each robot has to reach the street in front of it (see Figure 6.2.1a).

Performances have been evaluated using two criterions: the time before the first collision, and the number of collisions in a finite period. The considered term is 2 minutes long, it corresponds to the time needed by a robot to reach the selected global goal when it is going at the minimum speed, $0,2\ m/s$ in the pioneer3AT case.

In the first experiment we compare the *Random model* with the *Distributed* one. Each robot has a possible set of trajectories with $\{6, 12, 24\}$ elements, each trajectory computed correspond to a 1 meter long path that the robot performs with a fixed velocity of 0.2 m/s in the next second given a velocity. We use a short path and a low velocity with the purpose of minimizing the motion

[2]In a complete factor graph every node function has all the node variables as neighbors, in other words the functions have all the variables as arguments.



(a) Intersection. (b) Corridor.

Figure 6.2.1: Scenarios considering for testing model performances.

planning errors which are *time-growing* (e.g odometry). The results are shown in Tables 6.2 and 6.3.

Predictably, the best performance is obtained using the *Distributed Model*, stating and the random approach become not useful even when the number of robots is higher than two. Moreover, the random decisions could move the robot far from the global goal. As shown in the Table, no robot reaches the global goal before the two minutes.

With the Distributed model robots do not collide in any case but, we notice that they reach the goal in time when the goal set has a low cardinality. This fact is caused by the initial positions of the robots: all of them are placed closer to each other. In these conditions the probability of building a complete factor-graph between the robots is high and as explained in Section 3.0.3 it increases exponentially the number of messages exchanged. In Table 6.4the communication times for some complete factor graphs with low cardinality are illustrated.

We evince that the best cardinality for the trajectory set is 6 and with a complete factorgraph of 3 nodes at most.

In the second experiment we want to test how long a robot can perform a path without communicating again with other neighbors if needed. Then, we want to evaluate how far in the future we can plan the trajectory of the robot without encountering collisions. Even in this case we use a fixed velocity of 0.2 m/s and we test various time windows(see Section 3.0.2): 5, 7.5, 10, 15 seconds.

Trajectories	Goal reached before 2 mins	Robots	Model	Collisions before 2 mins	Time of the first collision
12	No	2	RAND	0-1	1’ 20”- 1’ 50”
12	No	4	RAND	1-3	1’-1’ 15”
12	No	6	RAND	3-5	2” - 15”
12	No	8	RAND	4-8	2”-11”
24	No	2	RAND	0-1	No coll.
24	No	4	RAND	0-1	No coll.
24	No	6	RAND	3-6	18”-30”
24	No	8	RAND	4-7	2”-11”

Table 6.2: Experiment 1 results with the *Random Model*.

Trajectories	Goal reached before 2 mins	Robots	Model	Collisions before 2 mins	Time of the first collision
6	Yes	2	DMRS	No Coll.	No Coll.
6	Yes	4	DMRS	No Coll.	No Coll.
6	No	6	DMRS	No Coll.	No Coll.
6	No	8	DMRS	No Coll.	No Coll.
12	Yes	2	DMRS	No Coll.	No Coll.
12	No	4	DMRS	No Coll.	No Coll.
12	No	6	DMRS	No Coll.	No Coll.
12	No	8	DMRS	No Coll.	No Coll.

Table 6.3: Experiment 1 results with the *DMRS Model*.

Variable nodes	Function nodes	Communication Times (secs)
2	2	0.3
3	3	1.7
4	4	30

Table 6.4: Complete factorgraph Times.

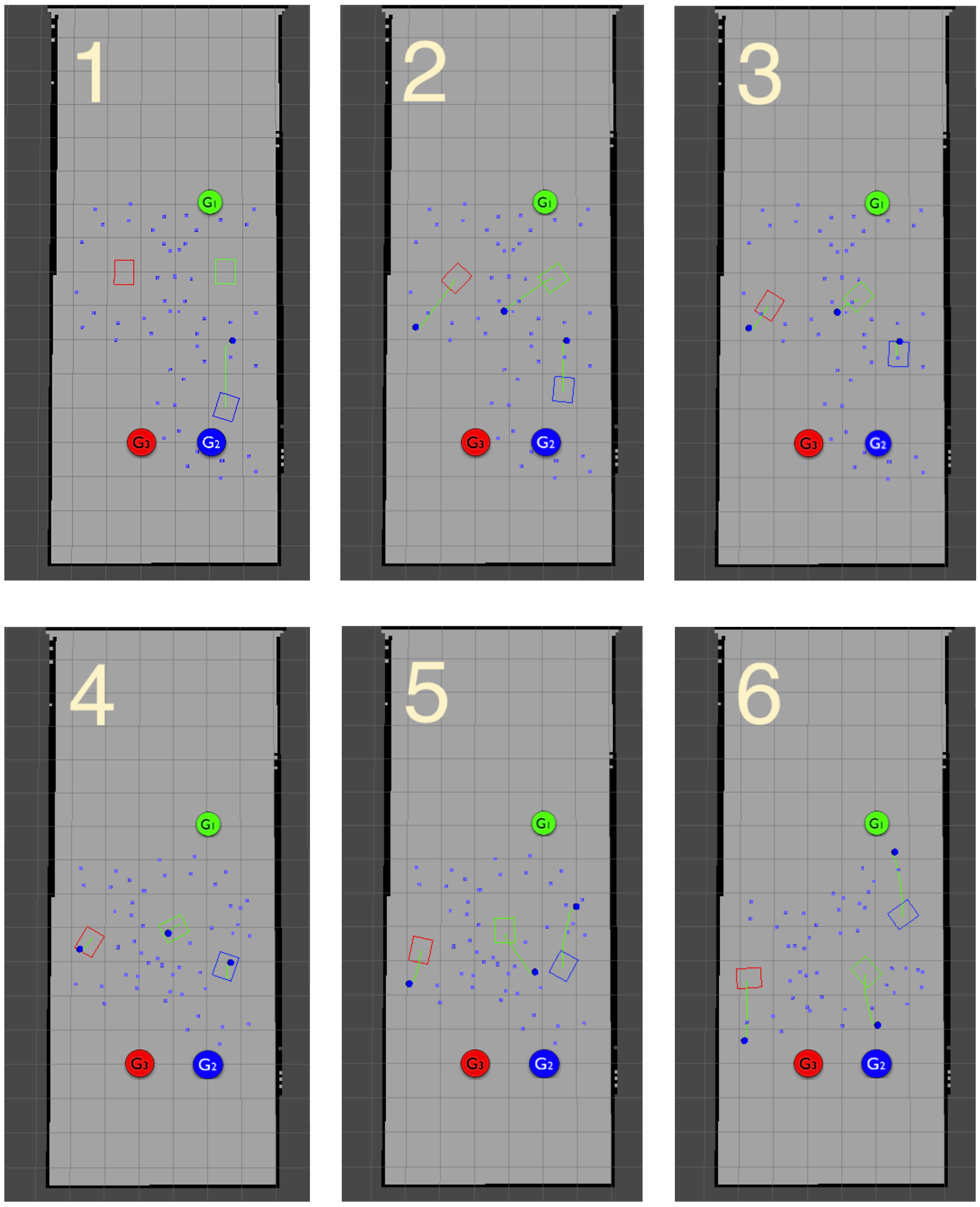


Figure 6.2.2: Three robots corridor test. The green lines are the robot trajectories and the blu dots are the candidate goals. This screens coming from the ROS Rviz visualization tool.

Robots	Trajectory time window (secs)	Time to goal	Collisions before 2 mins	First collision time (secs)
2	5	1.20	No Coll.	No Coll.
4	5	>2	No Coll.	No Coll.
6	5	>2	No Coll.	No Coll.
2	7.5	1.30	No Coll.	No Coll.
4	7.5	>2	No Coll.	No Coll.
6	7.5	>2	No Coll.	No Coll.
2	10	1.20	No Coll.	No Coll.
4	10	>2	No Coll.	No Coll.
6	10	>2	0-1	27
2	15	1.15	0-2	27
4	15	>2	3-4	17
6	15	>2	3-5	15

Table 6.5: Experiment 2. Results with various time windows.

	People	Other Obstacles
a (Force Magnitude)	700 N	23 N
b (Force Range)	2 m	0.01 m
c (Exerted Force Magnitude)	250 N/m	600 N/m

Table 6.6: Social Force Model parameters.

The Table 6.5 shows that we cannot plan the robot trajectory over the 10 seconds. Given the fixed velocity, it corresponds to a 2 meters path. This is coherent with the choice to communicate only with robots under a 3 meters distance.

We even test the SFM, changing what is the maximum time that permits a reactive approach in a environment. In other words we increase the sense-plan-act time varying the number of robots. We use the parameters in Table 6.6 for the social forces and a human mass of $70Kg$.

The SFM is an approach similar to the potential fields method and it does not work well in cluttered environments. Then we decide to test it in a scenario composed by some little rooms, where the task was to exit from these rooms by passing through narrow doors. The door size was different for each door. The objective was to evaluate how the SFM is good in little environments.

In the last experiment we test how long we can set the cycle time on the SFM without collisions. In this case the velocity is not constant because it will lost the meaning of the Social Forces use. The results in Graphic 6.2.3 shows that we cannot permit a cycle time that exceed the 2 seconds. Moreover it is clear that we cannot use only this reactive approach for the navigation because the times to reach the goal rises rapidly. The explanation is that the SFM approach does not give enough priority to reach the local priority and it does not take in account the global goal. Too much reactivity sometimes brings the robot too far from the local goal before reaching it.

Narrow Pass size (meters)	Exit time	Average velocity (m/s)
1	-	<0.1
1.4	34	0.2
1.8	35	0.2
2.0	26	0.3

Table 6.7: Time to pass through various narrow doors.

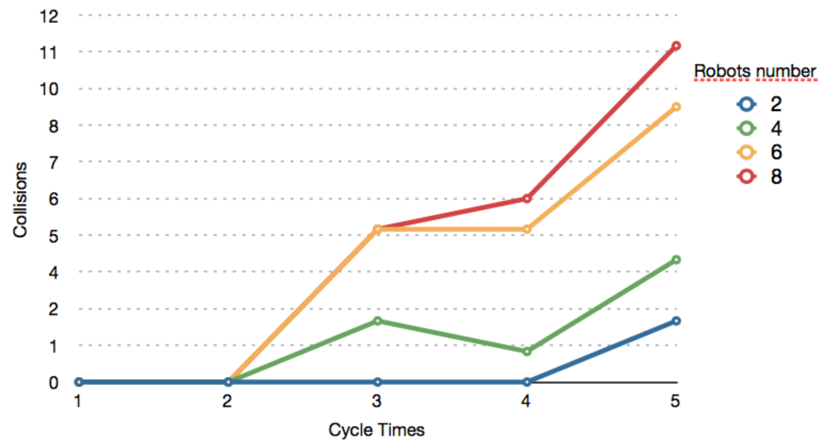


Figure 6.2.3: SFM cycle times and collisions.

		Cycle Time (Secs)				
		1	2	3	4	5
Robots	2	Yes	Yes	Yes	Yes	Yes
	4	No	No	Yes	Yes	No
	6	No	No	No	No	No
	8	No	No	No	No	No

Table 6.8: Goals reached in two minutes changing the cycle time .

Conclusions

In this work, we present a hybrid approach for safe robot motion planning where the environment is populated with humans and other robots. We have proposed a distributed collision avoidance system that has been enforced with a two-level navigation approach. The results reported in Table 6.1 report that this architecture is able to keep the system computational load low. The network bandwidth used by the Max-Sum messages is less than $5KB/s$ for each robot.

The architecture has been expanded with more collision avoidance modules. The proposed hybrid system is based on an heterogeneous collision avoidance system organized in two levels:

- Distributed kinodynamic motion planner using the concept of ICS state,
- Dynamic Window Approach enhanced with the Social Force Model with the aim of predicting people motions.

The test in Section 6.2 shows that the system can work without collisions with affordable times. When the robot plans using the Social Force Model, the sense-plan-act cycle can reach the 2 seconds. Despite, the SFM can't be used for navigate in cluttered environments as explained in the doors experiment. For this reason this reactive method is used only when people are perceived by the robot. The model based on social forces has been chosen because it can be used not only as reactive approach. The Social Force Model key feature is the estimation of next human positions. These informations can be used for increasing the Dynamic Window performances.

Thanks to the trajectory window, we added the time dimension to the distributed approach. This increases the set of possible solutions to the multi-robot planning problem. When the autonomous system is using the distributed approach the maximum trajectory window is 10 seconds. If we use a half of the time window we minimize the time needed to reach the goal. Despite, when we are using the Max-Sum we cannot use a complete factorgraph with more than 3 function and variable nodes. If the complete graph is bigger, the system lose the real-time capability.

The navigation algorithm proposed in 2.1 permits to reach a goal even if the robot do not know the entire map. This approach frees the robot by the task of building a representation of all the environment before starting the navigation. Obviously, this approach works if there is for sure a path that connects the robot position to the final goal (e.g. the path from a section of a warehouse to another).

Concluding, as a future experiment we suggest to try the proposed system on real robots. The Max-Sum permits better performances if implemented in a parallel manner. Then, another way to extend this work is to try to use multiple network interfaces for increasing the Max-Sum network parallelization. The used Max-Sum utility function sometimes drives the robots in bad trajectories. When it happens the robot chooses a trajectory that drives itself far from goal even if it is not near to an obstacle. A candidate future work could be the develop of a utility function that uses a heuristic approach.

Bibliography

[1] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, March 2000.

[2] K. E. Bekris, K. Tsianos, and L. E. Kavraki. Safe and distributed kinodynamic replanning for vehicular networks. *Mobile Networks and Applications*, 14(3), February 2009 2009.

[3] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 341–346. IEEE, 1999.

[4] James Bruce and Manuela Veloso. Safe multi-robot navigation within dynamics constraints. *Proceedings of the IEEE, Special Issue on Multi-Robot Systems*, 2006.

[5] Nicholas Chan, James Kuffner, and Matthew Zucker. Improved motion planning speed and safety using regions of inevitable collision.

[6] Christopher Clark, Stephen M. Rock, and Jean-Claude Latombe. Motion planning for multiple mobile robot systems using dynamic networks. In *IEEE Int. Conference on Robotics and Automation*, pages 4222–4227, 2003.

[7] A. Dijkstra. Dijkstra's algorithm.

[8] Stefano Michieletto Filippo Basso, Matteo Munaro and Emanuele Menegatti. Fast and robust multi-people tracking from rgb-d data for a mobile robot. In *In Proceedings of the 12th Intelligent Autonomous Systems (IAS) Conference, Jeju Island (Korea)*, July 2012.

[9] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[10] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *Robotics & Automation Magazine, IEEE*, 4(1):23–33, 1997.

[11] T. Fraichard and H. Asama. Inevitable collision states. a step towards safer robots? In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 388 – 393 vol.1, oct. 2003.

[12] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature, Vol. 407, pp. 487-490, 2000*, 2000.

[13] D. Helbing, I.J. Farkas, P. Molnar, and T. Vicsek. Simulation of pedestrian crowds in normal and evacuation situations. *Pedestrian and evacuation dynamics*, 21, 2002.

[14] Maciej Kalisiak. Faster motion planning using learned local viability models.

[15] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 47:498–519, 1998.

[16] Steven M. LaValle, James J. Kuffner, and Jr. Randomized kinodynamic planning, 1999.

[17] A. Mandow, JL Martínez, J. Morales, J.L. Blanco, A. Garcia-Cerezo, and J. Gonzalez. Experimental kinematics for wheeled skid-steer mobile robots. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1222–1227. IEEE, 2007.

[18] JL Martínez, A. Mandow, J. Morales, S. Pedraza, and A. García-Cerezo. Approximating kinematics for tracked mobile robots. *The International Journal of Robotics Research*, 24(10):867, 2005.

[19] E. Plaku, K.E. Bekris, B.Y. Chen, A.M. Ladd, and L.E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *Robotics, IEEE Transactions on*, 21(4):597–608, 2005.

[20] J. Snape, J. Berg, S.J. Guy, and D. Manocha. The hybrid reciprocal velocity obstacle. *Robotics, IEEE Transactions on*, (99):1–11, 2011.

[21] J. Van Den Berg, S. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. *Robotics Research*, pages 3–19, 2011.

[22] J. Van den Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1928–1935. IEEE, 2008.

[23] J. van den Berg, S. Patil, J. Sewall, D. Manocha, and M. Lin. Interactive navigation of multiple agents in crowded environments. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 139–147. ACM, 2008.

[24] J. van den Berg, J. Snape, S.J. Guy, and D. Manocha. Reciprocal collision avoidance with acceleration-velocity obstacles. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3475–3482. IEEE, 2011.

[25] Thomas S. Wikman, Michael S. Branicky, and Wyatt S. Newman. Reflexive collision avoidance: A generalized approach. In *ICRA (3)*, pages 31–36, 1993.

[26] Yuandong Yang and Oliver Brock. Elastic roadmaps–motion generation for autonomous mobile manipulation. *Auton. Robots*, 28:113–130, January 2010.

Acknowledgments

I thank the IAS Lab at Padova University, especially Matteo Munaro and Stefano Michieletto for the help on developing the proposed system. Thanks to Riccardo De Battisti for the help on implementing the Max Sum algorithm and on writing the paper derived by from this thesis.

I say thank you to my advisor Enrico Pagello, who gives me the opportunity of working in the marvelous world of robotics.

I show gratitute to Antonella, Maurizio, Jacopo, Simone, Giulia and my friends for supporting me in these not always simple academic years.