A distributed kinodynamic collision avoidance system under ROS

Nicoló Boscolo*, Riccardo De Battisti*, Matteo Munaro, Alessandro Farinelli, and Enrico Pagello

Abstract This paper focuses on decentralized coordination for small or medium groups of heterogeneous mobile robots with relatively low computational resources. Specifically, we consider coordinated obstacle avoidance techniques for mobile platforms performing high level tasks, such as patrolling or exploration. In more details, we propose the use of a greedy kinodynamic collision avoidance approach for the single robots and the use of the the Max-sum algorithm for multi-robot coordination. The system implementation and its testing are based on the popular robot middleware ROS and the gazebo simulation environment. Obtained results show that our distributed collision avoidance approach is able to achieve safe navigation in real-time with a very low overhead in terms of computation and communication.

1 Introduction

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

Nicoló Boscolo, Riccardo De Battisti, Matteo Munaro and Enrico Pagello
Department of Information Engineering
University of Padova, via Gradenigo, 6B 35131 Padova, Italy.
e-mail: {debattis,boscolo2,munaro,epv}@dei.unipd.it

Alessandro Farinelli
Department of Computer Science
University of Verona, Strada Le Grazie, 15 37134 Verona, Italy.
e-mail: alessandro.farinelli@univr.it

* Nicoló Boscolo and Riccardo De Battisti contributed equally to this work

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* [4] and *Velocity Obstacles* [8]. The latter has his most recent extension on the *Optimal Reciprocal Collision Avoidance* (ORCA) [16]. 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 [18]. These approaches are very efficient in simulations with a high number of agents as shown with *Reciprocal Velocity Obstacles* (RVO) [3]. 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)[2] issue [10]. 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 [7]. 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 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 [15, 2] and, even if all robots decide a feasible plan, maybe its end state is an ICS [10]. 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 [17] or in sampling-based replanning [5], as well as with learning-based approximations of ICS sets [13] or approximations for computing space×time obstacles [6].

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* [14] 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. The paper is organized as follows. Section 2 outlines the problem statement. Section 3 presents the description of the collision

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

avoidance system, first focusing on its single components, then explaining some algorithm contributions to the problem and finally illustrating the implementation within the ROS system. Section 4 explains the experiments conducted and the results obtained. Finally, conclusions and future works are discussed in Section 5.

2 Problem statement

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

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

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 v_i selected from V_i, a discrete 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 $v_i \in V_i$ such that, until R_i does not reach gl_i, $\forall t$

$$\begin{cases} f_P(R_i, v_i \cdot t) \in SE_i \\ f_P(R_i, v_i \cdot t) \cap f_P(R_i, v_j \cdot t) \neq \emptyset \quad \forall i \neq j \end{cases}.$$

3 System description

The planning cycle is executed by performing the following steps on the system represented in Figure 1:

1. The *Environment Model Builder* retrieves sensor and odometry data and computes a *costmap*, i.e. a discrete grid inflated with costs obtained from the environment sensor data;

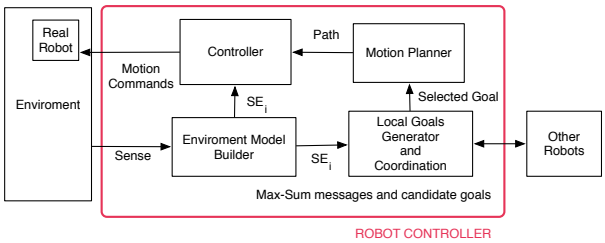


Fig. 1 Block schema of the collision avoidance system.

2. Given the global goal and the SE, the *Local Goals Generator* computes a set of feasible[3] goals around the robot position. Next the robots start the coordination step which ends with the selection of such goals that maximize the distances between all robots.
3. Given a local goal and by using the A^* algorithm [12], the *Motion Planner* computes a path whose distance is covered with the dynamic window approach[9].
4. The *Controller* receives the path and starts to send the motion commands to the real robot in sense-react loop until it does not reach the local goal.

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 blocks its path the robot stops and looks for another path.

3.1 Local goal generator and coordination

In this section we outline the theoretical and mathematical frameworks used for developing our collision avoidance algorithm *(Local Goal Generator and Coordination* on Figure *1*).

3.1.1 Navigation algorithm

Our navigation algorithm is detailed in Figure 2. Specifically we use a set structure called *geostructure,* where we keep all the computed goals. Geostruc-

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

ture is able to return the goals directly reachable from the current robot position (*neighbourhood*). Next, when we call c*ompute_goals_from* function, we select the safe[4] 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.

```
geostructure gs;
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);
    move_to(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);
  }
}
```

Fig. 2 High level navigation procedure

3.1.2 Factor graph

We use the *factor graph* [14] framework to perform our coordination problem. Specifically, given a real valued function $g(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{m} f_i(\mathbf{x_i})$, where $\mathbf{x_i} \subseteq \{x_1, x_2, \ldots x_r\}$, $r \leq n$, a factor graph is defined as a bipartite graph that shows the structure of this summation. In particular the factor graph $FG = \{\mathbf{x}, \mathbf{f}\}$ consists of *variable nodes* $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ and *function nodes* $\mathbf{f} = \{f_1, f_2, \ldots f_m\}$ where a variable node x_i is connected to the function node f_j if and only if the variable is an argument of the function, i.e. $x_i \in \mathbf{x}_j$.

[4] A goal is defined as *safe* when the trajectory towards it does not make robot collide.

3.1.3 Max-sum algorithm

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 $g(\cdot)$. 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 4) where each robot R_i owns a function F_i and a subset $\mathbf{x_i} \subseteq \mathbf{x}$ of variables, the max-sum algorithm computes $\mathbf{x}^* = \arg\max_{\mathbf{x}} \sum_{i=1}^{n} F_i(\mathbf{x_i})$ by repeatedly passing *variable-to-function* q-messages and *function-to-variable* r-messages. Let $\mathbf{x}_i$ be a set of paths to candidate local goals, $F_i(\mathbf{x}_i)$ represents the minimum distance between all possible local goals, logarithmically weighted with the distance to the global goal G_i (see Figure 3). In case $\mathbf{x}_i$ would lead to a collision, $F_i(\mathbf{x}_i)$ is set to an arbitrarily small positive quantity ϵ. Hence $\mathbf{x}^*$ represents the local goals that maximize the system utility $\sum_{i=1}^{n} F_i(\mathbf{x_i})$, 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. A similar

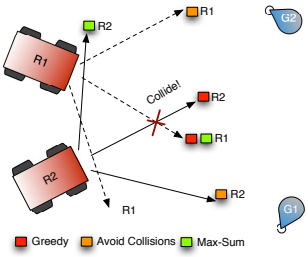


Fig. 3 Comparison between 3 motion planning methods in choosing the local goals with $|GL_i| = 3$

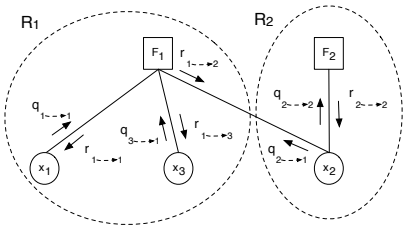


Fig. 4 Example of a simple factor graph, the robots R_1, R_2, the function nodes F_1, F_2, the variables nodes x_1, x_2, x_3

coordination problem in [2] is modeled with coordination graphs [11] and it is

solved using the message passing algorithm max-plus [11], but that method does not consider *n-ary* functions. In our approach such functions are very important because they allow us to simultaneously consider, if necessary, not only collisions between couples of robots, but collision between a group of n robots. In particular in our approach each robot owns only its local part of

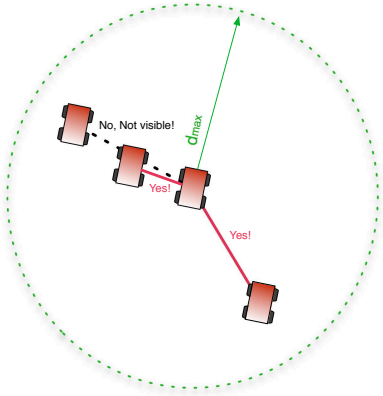


Fig. 5 Line of sight neighbor selection

the factor graph, hence a minimal factor graph can be computed even without a real distributed computation. Indeed as Table 1 shows, the Max-sum computational time increases exponentially with the number of graph connections, so in order to keep low the time to reach a solution we communicate with just the two nearest neighbors. As in Algorithm 1 the robot computes the main cycle, where in step 2b we consider neighbors just the robots which

Algorithm 1 Sense-plan-act cycle

1. *Sense*;
2. *Plan*:
   a. Compute some possible local goals and their relative trajectories;
   b. if there are not neighbors which can collide with myself go to 3, else continue;
   c. if all neighbors do not want to perform the Max-sum algorithm go to 2b, else continue;
   d. Create the local variables and the local function;
   e. Create the factor graph with its 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 the global goal is reached.

are less than a d_{max} far from the robot. Such distance is that a robot could cross if it moves at the maximum velocity for T_{act}, i.e. the time of the *act* step, and it is so defined

$$d_{max} = v_{max} \cdot T_{act}.$$

In 2e entry, the connection are created after the local graph is computed and it is not rare that two agents are not symmetrically situated in the neighbors set. Indeed the factor graph is a not oriented graph so we check locally the graph consistency. For instance, let the robot R_i be the owner of the variable TR_i *locally connected* with an external function F_j, whose owner is robot R_j, hence if for some reasons in the factor graph of R_j there is not connection between F_j and TR_i, the R_i local edge will be deleted. As stated before, when the Max-sum uses complete factor graphs the communication times grow exponentially (see Table 1) with the number of nodes. We use a *graph pruning* policy where an agent is connected with two nodes at most.

Table 1 Complete factor graph communication times

Variable nodes	Function nodes	Communication Times (secs)
2	2	0.3
3	3	2.7
4	4	23

3.2 System design under 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 that can communicate with other ones even if they are in different packages thanks to asynchronous recipients called *topic* and *service*. Nodes are connected to each other by a peer-to-peer connection but all the nodes have to communicate with the *Master service*[5] to enable the connection. This kind of network architecture highly favors a distributed arrangement. In 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.

[5] Master service only provides lookup information in some way like a DNS server.

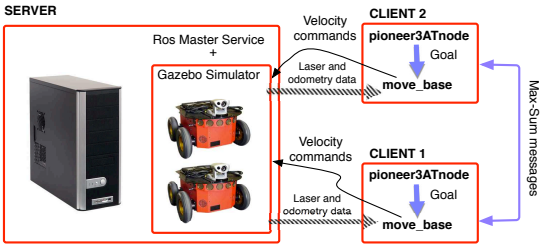


Fig. 6 System setup for our ROS implementation

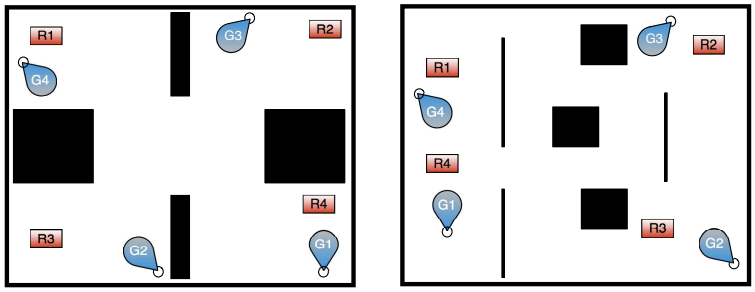
Given the footprint of the robot, environmental data from on-board sensors (e.g. laser, stereoscopic camera) and the odometry, the *move_base* node has the objective to compute the velocities and the steering angles of the robot in order to reach a goal communicated by the pioneer3AT node. *Move_base* is tuned for avoid the static obstacles by using the Dynamic Window Algorithm. This node also publishes a costmap of the local environment built from the sensor data, which is used to compute the path toward the local goal by using the A^* algorithm. Moreover, we do our test using virtual worlds builded for the *gazebo simulator* [6]. Its 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.

4 Experiments and results

The tests on our collision avoidance approach have been focused on the scalability of the planning cycle times. The workbench was a workstation[7] 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 7(a)), robots have to reach their global goals respectively placed on their opposite corner. In the second scenario (see Figure 7(b)) 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 Table 2 we report the mean execution times for a full round of our algorithm (message exchanges and convergence

[6] This tool is directly available in ROS.

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



(a) Crossroads (b) Labyrinth

Fig. 7 Scenarios with robots R_1 , R_2 , R_3 , R_4 and global goals G_1 , G_2 , G_3 , G_4

Table 2 Mean 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

to a local goal choice) and in the last column we show the estimated RAM and number of CPUs 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 factor graphs are complete[8] 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. Our tests showed the important role played by the number of candidate goals computed in each planning cycle: we noticed that if this number decreases under the 6 units, collisions happen on almost the 100% of the tests. As shown on Table 2, we can deduce that a real-time onboard robot computer needs 1 CPU with 1 GiB RAM at least.

5 Conclusions

We built a decoupled and distributed coordination approach with low computational overhead using the Max-Sum and a greedy algorithm with the aim of using it on robots with low computational resources. Since the first results

[8] 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.

look promising, we envision to try it on a real low cost robots group. As a future work, we also plan to implement a reactive collision avoidance system able to avoid unknown objects, like people.

References

1. Aji, S.M., McEliece, R.J.: The generalized distributive law. IEEE Transactions on Information Theory **46**(2), 325–343 (2000). DOI 10.1109/18.825794
2. Bekris, K.E., Tsianos, K., Kavraki, L.E.: Safe and distributed kinodynamic replanning for vehicular networks. Mobile Networks and Applications **14**(3) (2009)
3. Van den Berg, J., Lin, M., Manocha, D.: Reciprocal velocity obstacles for real-time multi-agent navigation. In: IEEE International Conference on Robotics and Automation, 2008, pp. 1928–1935. IEEE (2008)
4. Brock, O., Khatib, O.: High-speed navigation using the global dynamic window approach. In: IEEE International Conference on Robotics and Automation, 1999., vol. 1, pp. 341–346. IEEE (1999)
5. Bruce, J., Veloso, M.: Safe multi-robot navigation within dynamics constraints. Proceedings of the IEEE, Special Issue on Multi-Robot Systems (2006)
6. Chan, N., Kuffner, J., Zucker, M.: Improved motion planning speed and safety using regions of inevitable collision. In: 17th CISM-IFToMM Symposium on Robot Design, Dynamics, and Control (RoManSy'08) (2008)
7. Clark, C., Rock, S.M., Latombe, J.C.: Motion planning for multiple mobile robot systems using dynamic networks. In: IEEE Int. Conference on Robotics and Automation, pp. 4222–4227 (2003)
8. Fiorini, P., Shiller, Z.: Motion planning in dynamic environments using velocity obstacles. The International Journal of Robotics Research **17**(7), 760–772 (1998)
9. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. Robotics & Automation Magazine, IEEE **4**(1), 23–33 (1997)
10. Fraichard, T., Asama, H.: Inevitable collision states. a step towards safer robots? In: Intelligent Robots and Systems, 2003, vol. 1, pp. 388 – 393 vol.1 (2003). DOI 10.1109/IROS.2003.1250659
11. Guestrin, C., Koller, D., Parr, R.: Multiagent planning with factored MDPs (2001)
12. Hart, P., Nilsson, N., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100–107 (1968). DOI 10.1109/TSSC.1968.300136
13. Kalisiak, M., van de Panne, M.: Faster motion planning using learned local viability models. In: ICRA'07, pp. 2700–2705 (2007)
14. Kschischang, F.R., Frey, B.J., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE TRANSACTIONS ON INFORMATION THEORY **47**, 498–519 (1998)
15. LaValle, S.M., Kuffner, J.J., Jr.: Randomized kinodynamic planning (1999)
16. Van Den Berg, J., Guy, S., Lin, M., Manocha, D.: Reciprocal n-body collision avoidance. Robotics Research pp. 3–19 (2011)
17. Wikman, T.S., Branicky, M.S., Newman, W.S.: Reflexive collision avoidance: A generalized approach. In: ICRA (3), pp. 31–36 (1993)
18. Yang, Y., Brock, O.: Elastic roadmaps–motion generation for autonomous mobile manipulation. Auton. Robots **28**, 113–130 (2010). DOI http://dx.doi.org/10.1007/s10514-009-9151-x