LUND UNIVERSITY

# A Component-Based Approach to Localization and Collision Avoidance for Mobile Multi-Agent Systems

Alriksson, Peter; Nordh, Jerker; Årzén, Karl-Erik; Bicchi, Antonio; Danesi, Alberto; Sciadi, Riccardo; Pallottino, Lucia

2007

Link to publication

*Citation for published version (APA):*
Alriksson, P., Nordh, J., Årzén, K.-E., Bicchi, A., Danesi, A., Sciadi, R., & Pallottino, L. (2007). *A Component-Based Approach to Localization and Collision Avoidance for Mobile Multi-Agent Systems*. Paper presented at European Control Conference, 2007, Kos, Greece.

*Total number of authors:*
7

**General rights**
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

**Take down policy**
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Download date: 17. Dec. 2025

# A Component-Based Approach to Localization and Collision Avoidance for Mobile Multi-Agent Systems

P. Alriksson, J. Nordh, K.-E. Årzén
Department of Automatic Control, LTH
Lund University, Sweden

A. Bicchi, A. Danesi, R. Schiavi, L. Pallottino
Department of Electrical Systems and Automation
University of Pisa
Pisa, Italy

*Abstract*— **In the RUNES project a disaster relief tunnel scenario is being developed in which mobile robots are used to restore the radio network connectivity in a stationary sensor network. A component-based software development approach has been adopted. Two components are described in this paper. A localization component that uses ultrasound and dead reckoning to decide the robot positions and a collision avoidance component that ensures that the robots do not collide with each other.**

## I. Introduction

Within the EU/IST FP6 integrated project RUNES (Reconfigurable Ubiquitous Networked Embedded Systems) a disaster relief tunnel scenario has been defined. In the scenario mobile robots are used as mobile radio gateways that ensure the connectivity of the tunnel sensor network. In order to enable this accurate localization, collision detection, and collision avoidance mechanisms are required.

In RUNES a component-based approach to software development has been adopted. The RUNES tunnel scenario and the component model are described in more detail in the introductory paper [1]. This paper describes the localization method and the collision avoidance method used in the demonstrator. It also describes how these mechanisms have been implemented using the component framework.

### A. Outline of the Paper

Section II is devoted to the localization component. An ultrasound-based active localization scheme is used. Motivations for this approach are given, as well as details of how it has been implemented. Special attention is given to the Extended Kalman Filter that is used for data fusion.

Section III describes the collision avoidance component (CAC). The overall policy is described together with simulation and experimental results and implementation details.

## II. Localization Component

### A. Localization of Mobile Robots

In the disaster relief tunnel scenario autonomous mobile robots are used as radio gateways responsible for restoring the tunnel network connectivity. This implies that the robots are required to navigate inside the tunnel avoiding collisions with other robots and with unknown obstacles. A prerequisite for navigation is localization, i.e., the robots must know their current position and heading. Since the tunnel is assumed to be well-known, automatic map building is not considered. Instead it is assumed that the overall layout of the tunnel is known, with the exception of the position of a number of stationary obstacles, modeling, e.g., stalled vehicles.

Localization of mobile robots can be performed with a number of techniques. In laboratory experiments it is common to use vision, e.g., a ceiling-mounted camera combined with an image-processing system. In the tunnel scenario this is not a realistic approach due to, e.g., problems with light and smoke. Another possibility is to use dead-reckoning using a high-precision inertial measurement sensor unit onboard the robot. A problem with dead reckoning-based approaches, however, is that they are open loop and that unmeasurable disturbances will cause position errors that cannot be compensated for. In an outdoor environment GPS would have been another possibility, but inside a tunnel this is less realistic.

The localization approach chosen in the RUNES project is based on ultrasound. The basic idea is to transmit a wireless radio packet simultaneously with an ultrasound pulse from each sender node. The receiver nodes measure the difference in time of arrival between the radio packet and the ultrasound pulse and can in this way calculate their distance to the sender node. By combining, or fusing, several distance measurements an estimate of the position can be obtained.

Two main approaches exists, [2]. In an *active mobile* system the infrastructure, in this case the tunnel, has receivers at known locations, which estimate distances to a mobile device based on active transmissions from the device. Examples of this approach are the Active Badge [3], and the Ubisense [4] systems. In a *passive mobile* system, instead, the infrastructure has active beacons at known positions that periodically transmits signals to a passive mobile device. The most famous example of this is the Cricket system [5].

An advantage of the active approach is that it is more likely to perform accurate tracking than the passive approach. The passive approach, on the other hand, scales better with the number of mobile devices. Since in the tunnel scenario good tracking is important and the number of mobile robots is small, the active approach was chosen. The stationary sensor nodes in the tunnel are each equipped with an ultrasound receiver and each mobile robot is equipped with an ultrasound transmitter. The stationary sensor nodes are implemented as Tmote Sky sensor network "motes" together with a small ultrasound receiver circuit interfaced to the mote via the AD converter, see Figure 1. The mobile robots are equipped

ISBN: 978-960-89028-5-5

Fig. 1. Stationary sensor network nodes with ultrasound receiver circuit. The nodes are packaged in a plastic box to reduce wear.

with an ultrasound transmitter circuit. Both the ultrasound transmitters and receivers are designed to be isometric, i.e., to transmit and receive in the full $360^\circ$ degree plane.

A second reason for choosing ultrasound-based localization is that it involves the use of the sensor network in closed loop. One of the objectives of the RUNES project was to investigate the possibilities and problems associated with networked control over sensor networks. In wireless networks the lack of worst-case latency guarantees and risk of losing radio packets creates extra challenges for control. The ultra-sound based location method provides a possibility for evaluating this.

The non-determinism of wireless radio communication makes it necessary to combine the ultra-sound localization with dead reckoning. The latter is based on the measured, and in certain situations also the commanded, wheel movements. This combination is especially important when the robots approach the zone where the tunnel network is disconnected, i.e. where the stationary sensor nodes are malfunctioning. Here the robots can only use the dead reckoning. An additional reason for using dead reckoning is lack of information about the robot heading from the ultrasound. Although, this can be obtained also in ultrasound-based systems, e.g., in the Cricket system, it has not been included in the hardware setup for the tunnel scenario.

### B. Ultrasound Based Localization

The implemented localization method works according to the following principles. At the beginning of each measurement cycle, the robot first transmits a broadcast radio message to alert the nodes of the incoming ultrasound pulse. After a fixed time the robot then emits an ultrasound pulse. When the radio message reaches the sensor node, it starts sampling the ultrasound microphone signal. During this phase, the sensor node detects the beginning of the pulse using a moving median filter.

Next the sensor node reports the sample index of the moving median filter back to the robot over the radio channel. To avoid radio collisions the node waits a pre-specified time based on its IP address, before sending. The sample index provided by the nodes is proportional to the distance between the sensor node and the robot when the pulse was emitted. If the speed of sound is assumed constant and the sampling interval in the sensor nodes is known the actual distance can be computed.

By combining multiple distance measurements with the actual movements of the robot accurate position and heading estimates can be derived. Once each robot has updated its own position estimates these are broad-casted to all the other mobile robots.

The proposed scheme above does not work with multiple robots. If more than one robot send an ultrasound pulse at the same time or close in time, then their positions can become mixed up. To avoid this a CSMA (Carrier-Sense Multiple Access) approach is used. Before a robot sends an ultrasound pulse it listens so that no radio message has been sent from some other robot. If this is the case the robot aborts the ultrasound emission and inserts a small delay in its schedule. The delay will stretch the schedule and after, possibly, a few additional collisions the ultrasound send schedules of the two robots will be de-synchronized and no further collisions will occur. A prerequisite for the scheme to work is that all robots are sufficiently close together during initialization to avoid problems with hidden nodes.

In order to detect unknown obstacles the robots developed in Lund (the RBbots) are equipped with an IR proximity sensor that is mounted on a small RC-servo. The sensor acts as a radar that sweeps a $150^\circ$ degree sector in front of the robot. In addition to this this the robots are equipped with fixed touch sensors.

### C. Data Fusion

To make good use of the distance measurements a dynamical model of the robot is required. The Lund robot, see Figure 2, is a dual-drive unicycle robot.

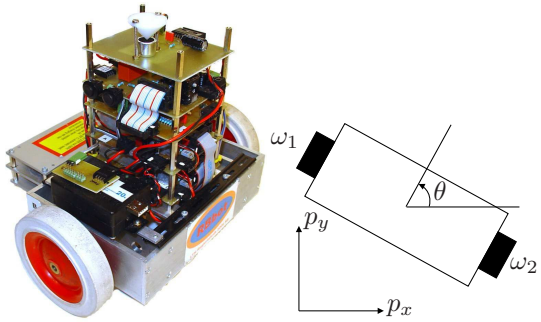


Fig. 2. Definition of coordinates with respect to the robot.

It is modeled as a third order system

$$\begin{aligned} \dot{p}_x &= \frac{1}{2}(R_1\omega_1 + R_2\omega_2)\cos(\theta) \\ \dot{p}_y &= \frac{1}{2}(R_1\omega_1 + R_2\omega_2)\sin(\theta) \\ \dot{\theta} &= \frac{1}{D}(R_2\omega_2 - R_1\omega_1) \end{aligned} \tag{1}$$

where the state consists of the $x$- and $y$-position together with the heading $\theta$. Input to the system are the angular velocities $\omega_1$ and $\omega_2$ of the two wheels, see Figure 2. The angular velocities are controlled by two PI-controllers and thus assumed known.

The model has three physical parameters; $R_1$, $R_2$ the radius of the two wheels and $D$ the distance between the

wheels. All these are easily obtained by physical measurements.

The relation between distance to node $i$ and the robot position is

$$d_i = h_i(\cdot) = \sqrt{(p_x - p_{xi})^2 + (p_y - p_{yi})^2 + (p_z - p_{zi})^2} \tag{2}$$

where $p_{xi}$, $p_{yi}$, and $p_{zi}$ are the position coordinates of node $i$. The locations of the stationary nodes are assumed to be known. Note that the elevation $p_z$ is not estimated and assumed known as the robot is only allowed to move in two dimensions. The available distance measurements $d_i$ are stacked in a vector $d$ with a corresponding stacked function $h(\cdot, k)$. Note that the time variation in $h(\cdot, k)$ is due to the varying number of available measurements.

To solve the state filtering problem, that is, to estimate the position and heading of the robot, a discrete-time extended Kalman filter [6] is used. Using a Tustin discretization scheme with sampling interval $T$ and adding stochastic disturbances a model on the following form was obtained.

$$x_{k+1} = f(x_k, u_k) + w_k \tag{3}$$
$$y_k = h(x_k, k) + e_k \tag{4}$$

Here $x_k$ denotes the state and $u_k$ the known angular velocities of the wheels at time $kT$. The white stochastic disturbances $w_k$ and $e_k$ are assumed independent of each other with covariance $Q$ and $R$.

Let us first briefly review the extended Kalman filter or EKF for short. The algorithm consists of two steps; the prediction step and the update step.

In the update step the measurements, that is $y_k$ in this case, are used to update the mean and covariance of the state estimate. As (4) is a nonlinear function of the state this is non trivial. In the EKF, (4) is linearized around the mean from the previous prediction step, which simplifies the update. The mean ($\hat{x}$) and covariance ($P$) are updated as

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \left(y_k - h(\hat{x}_{k|k-1})\right) \\ P_{k|k} &= (I - K_k C_k) P_{k|k-1} \end{aligned} \tag{5}$$

where

$$K_k = P_{k|k-1} C_k^T (C_k P_{k|k-1} C_k^T + R)^{-1} \tag{6}$$

and $C_k$ is the linearization of $h(\cdot)$ around $\hat{x}_{k|k-1}$.

In the prediction step (3) is used to make a prediction of the state. Also here, only the mean and covariance are propagated. The tool used to overcome the non-linearity is once again linearization, but this time around the mean after the update step. The mean and covariance are updated as

$$\begin{aligned} \hat{x}_{k+1|k} &= f(\hat{x}_{k|k}, u_k) \\ P_{k+1|k} &= A_k P_{k|k} A_k^T + Q \end{aligned} \tag{7}$$

where $A_k$ is the linearization of $f(\cdot)$ around $\hat{x}_{k|k}$.

In the actual implementation a number of problems arise. The update step in an EKF is computationally intensive and numerically sensitive, since it involves the inversion of a matrix of the same size as the number of measurements, i.e., the number of stationary nodes in range. To overcome this, a sequential update scheme where one update step is done for each available measurement was implemented.

In the scenario considered here more than one robot will use the network simultaneously, thus the update frequency is limited. However the prediction step does not use any shared resources, therefore two extra prediction steps are done between each update. In the current implementation the prediction step is executed every 400 ms and the update step every 1200 ms. The main execution schedule is shown in Fig. 3. The length of the schedule is $1,2$ ms. It consists of three repeating phases. In all the three phases the prediction step of the EKF and the collision avoidance component and the robot navigator/controller are executed. In the first phase only an ultrasound range request is sent out. As soon as a distance measurement is received a sequential update of the EKF is performed.

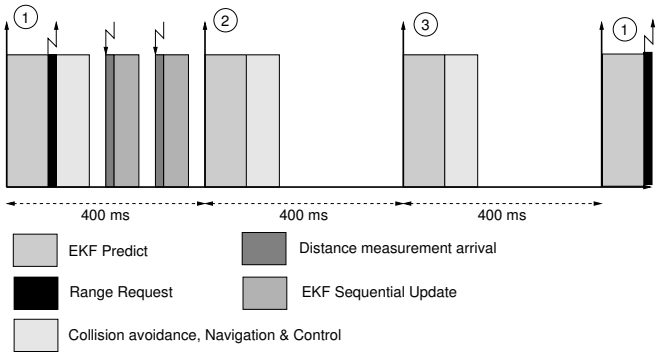


Fig. 3. Main execution schedule.

### D. Experimental Results

In Figures 4, 5 and 6 the estimated position and heading is shown together with measurements from a camera system. The accuracy of the position and heading measurements generated by the camera system is approximately 1 cm and $2^o$ respectively. Data was logged at the same frequency as the update step was run, that is every 1200 ms.

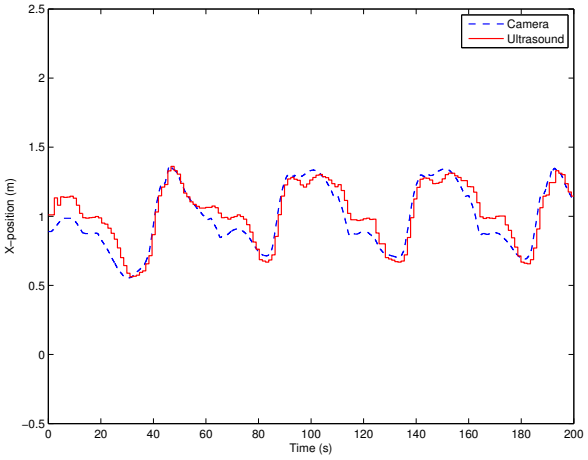


Fig. 4. X-position estimate generated by the ultrasound system (solid) together with measurements from a camera system (dashed). Data is logged every 1200 ms.

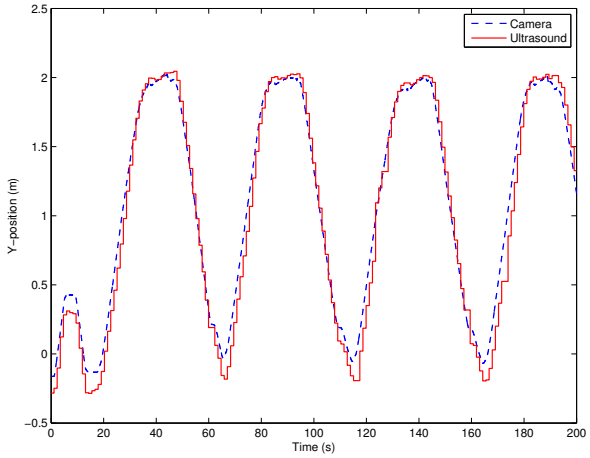


Fig. 5. Y-position estimate generated by the ultrasound system (solid) together with measurements from a camera system (dashed). Data is logged every 1200 ms.

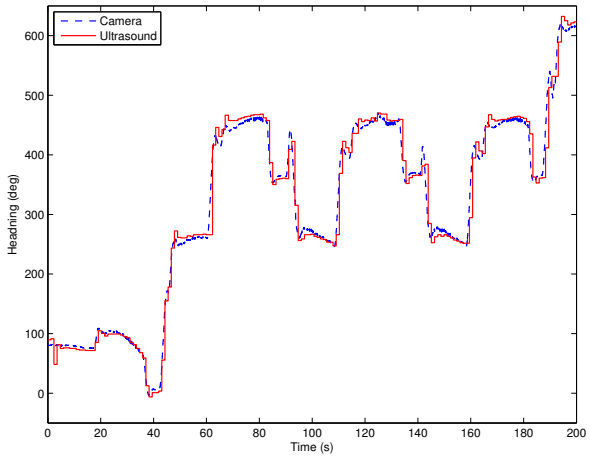


Fig. 6. Heading estimate generated by the ultrasound system (solid) together with measurements from a camera system (dashed). Data is logged every 1200 ms.

### E. Software Structure

The API of the RUNES component model is defined in terms of interfaces and receptacles. A component interface is a set of functions that are implemented by a component and offered to others. In order to access these functions, other components must define a corresponding receptacle and ask for a binding between its receptacle and the desired interface. A receptacle is a construct that the RUNES middleware uses to define a set of functionalities that a component expect to be implemented by other components. The RUNES middleware allows to bind a receptacle to different components declaring a proper interface.

The localization component consists of two parts. A distance sensor component that resides in the stationary sensor nodes and the data fusion component that resides in the mobile robots according to Figure 7.

The distance sensor component is more or less self-contained. Except for, possibly, an initializing setup function it only interacts with the environment through the radio

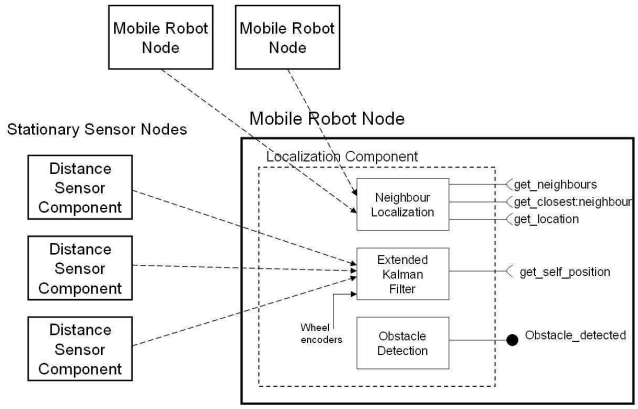


Fig. 7. The localization component.

interface. It is implemented as a Contiki proto-thread [7] that waits for an incoming radio message, listens for the ultrasound pulse, and sends a radio message back to the robot.

The interface of the localization component that resides in the mobile robots contains the following functions:

Syntax:

```
void get_self_position(int16& x, int16& y,
int& positionAccuracy, int16& orientation
 int& orientAccuracy)
```

This function returns the current position and orientation of the mobile agent together with an estimate of how accurate these values are.

Syntax:

```
get_neighbours(agents[] neighbours)
```

Returns all neighbours that currently are within radio range, i.e., have updated their position within a certain time window.

Syntax:

```
get_closest_neighbour(agent& neighbour)
```

Returns the closest mobile agent.

Syntax:

```
get_location(agent neighbour, int16& x,
int16& y, int& positionAccuracy,
int16& orientation, int& orientAccuracy)
```

Returns the position information for a certain neighbour agent

### F. Receptacles

The localization component requires the collision avoidance component to provide the following function:

Syntax:

```
obstacle_detected(agent ag, int16& distX)
```

Informs the collision avoidance algorithm that an obstacle has been detected at a certain distance in front of a mobile agent.

## III. CAC: Collision Avoidance Component

The collision avoidance component (CAC) is responsible for the motion coordination of the mobile agents in the environment. It prevents collisions and guarantees that each agent eventually accomplishes its individual task by reaching a desired destination.

### A. The collision avoidance policy

The component implementation is based on a decentralized collision avoidance protocol, called "generalized roundabout policy" (GRP), that has been recently proposed for mobile agents evolving on the plane [8], [9], [10]. The GRP is briefly described for the reader's convenience.

Consider a number of mobile agents moving in the plane at constant speed, along paths with bounded curvature. The state of each agent is represented by the coordinates $(x, y)$ and the heading angle $\theta$. According to the protocol a first circle is assigned to each agent, called the *safety disk*, being the circle centered at the agent position $(x, y)$ with a predefined radius $R_s$. A *collision* is said to occur whenever two or more safety disks overlap.

A collision avoidance policy has the main requirement of *safety* to be verified, i.e. to avoid collisions of the mobile agents while they attend to their tasks. Another important requirement is the *scalability* of the policy that imposes the absence of a centralized traffic supervisor dispatching detailed instructions to all agents. Rather, the policy should be completely decentralized and for example hinging upon a set of "traffic rules" shared among agents. These traffic rules, must enable each agent to autonomously make decisions about its own motion, based on information of its own state and the state of only a fixed, small number of "neighboring" agents.

The collision avoidance policy has been developed taking into account such requirements and vehicles with non-trivial kinematics, such as vehicles that are not able to stop their motion and have constraints in the angular velocities. For dealing with such a case, the policy defines a *reserved disk* for each agent as the circle that contains the path traveled by the safety disk, when its associated agent turns right at the minimum allowable curvature. The center of a reserved disk can easily be obtained from its agent state $((x^c, y^c) = (x + R_c \sin(\theta), y - R_c \cos(\theta))$ ; where $R_c$ is the minimum feasible curvature radius). In spite of the agent constraints, the motion of the reserved disk can be stopped at any time, by making the agent turn right at the minimum curvature rate. The reserved disk has radius $R_c + R_s$ and inherits the heading angle $\theta$ of the mobile agent.

Suppose that each agent has to reach a desired final position and heading to accomplish its task. The motion strategy followed by the agent is based on four distinct *modes of operation*, each assigning a suitable value to the curvature rate of the agent. Figure 8 shows these operation modes along with the corresponding switching conditions, a.k.a. *guards*. The policy has been defined in order to keep reserved disks disjoint. With reference to the figure, each agent enters the **`straight`** mode if the motion along the line directed as the agent heading is directed toward the goal configuration and is permitted, i.e. its reserved disk does not overlap with other reserved disks. During this mode, the agent's curvature rate is set to zero. Whenever its reserved disk becomes tangent to the one of another agent, a test is made based on the current motion heading $\theta$. If a further movement in the direction specified by $\theta$ causes an overlapping, then the agent enters the **`hold`** mode. Otherwise, the agent is able to proceed, and remains in the straight mode. When the hold mode is entered, the agent's curvature rate is set to the minimum allowable (turning on the right), and the motion of its reserved disk is stopped. As soon as the agent heading is permitted but not directed towards the target destination, the agent enters the **`roll`** mode, and tries to go around the other reserved disk. This is achieved by selecting a suitable value for the curvature rate of the agent such that the two disks never overlap. During the roll mode, the tangency of the two disks can unexpectedly be lost, then the agent enters the **`roll2`** mode, and the curvature rate is set to the maximum allowable in order to restore the contact (turning on the left). The roll2 mode can only be entered if the previous mode was roll. When the tangency is restored, the agent switches back to roll mode. A simple example of a possible trajectory of an agent that moves according to the GRP is depicted in Figure 9.

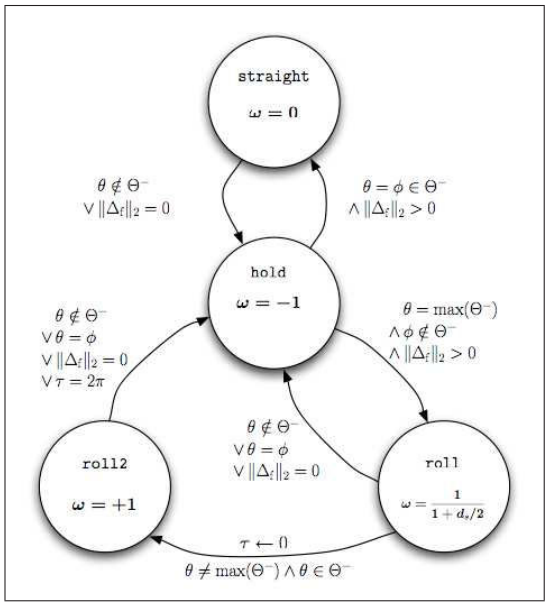


Fig. 8. Finite state automaton that summarizes the collision avoidance protocol GRP.

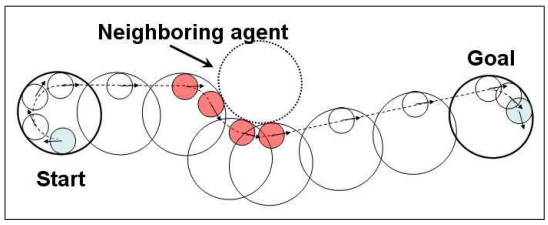


Fig. 9. Example of a possible trajectory of an agent applying the proposed collision avoidance protocol. Smaller circles are safety disks, larger circles are reserved disks.

From the description above, it is important to notice that the only information that agents need to exchange is the current configuration. Furthermore, each agent must know the configuration of neighbour agents that in case of identical agents are at most six. Hence, the policy is completely decentralized and the amount of information to be exchanged is limited and not based on intentions but only on positions. The decentralized characteristic of the protocol allows the CAC to be implemented on-board the agents. As a matter of fact, each agent is able to make a safe decision about its motion, based only on the locally available information. This information consists of the position and heading of agents that are within a certain sensing or communication radius. If agents are not homogeneous (different dimensions, safety radii or different curvature radii) the policy is still applicable. However, in this case the agents need to exchange information on the safety and curvature (or reserved) radii dimension.

In [8], under mild conditions on the initial configurations, the policy has been proved to be safe, i.e. to guarantee collision avoidance throughout the system evolution. Furthermore, conditions on the desired configurations of agents under which the ultimate convergence of all vehicles to their goals can also be guaranteed have been obtained. To show that such conditions are actually necessary and sufficient, which turns out to be a challenging liveness verification problem for a complex hybrid automaton, a probabilistic verification method has been used [8].

Even though the proposed collision avoidance policy has been designed based on mobile agent conflicts, it may also be applied in case of fixed obstacles. No details on the obstacle avoidance strategy based on the GR policy is herein presented for the sake of brevity. Please refer to Figure 9 for a possible obstacle avoidance in the simple case of obstacle with shape and dimension equal to the reserved disk.

### B. Simulations and experimental results

A large number of simulations have been conducted on the collision avoidance component. Simulations have shown that the CAC provides effective solutions for large-scale problems, such as e.g. the 70-agents conflict resolution illustrated in Figure 10.

In parallel, a preliminary platform has been designed for testing safe and secure decentralized traffic management policies for multi-agent mobile systems. In particular, several experiments have been conducted on the the GRP policy [11]. The architecture is based on wireless communication between agents. Furthermore, the architecture provides the agents with services such as the localization, which was implemented based on a centralized server that monitors the environment through a camera. In Figure 11 some screenshots of a three-vehicle case are reported with overprinted reserved disks. For technical details and experimental data please refer to [11]. Recently this preliminary platform has been revisited and updated to meet RUNES middleware requirements.

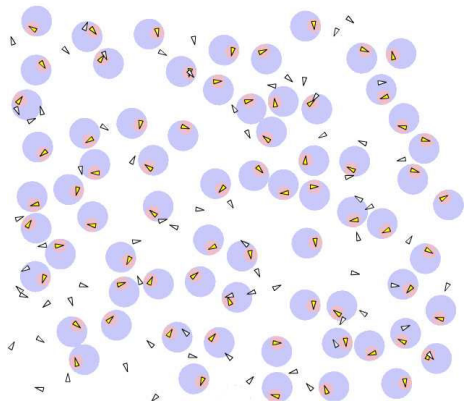

Fig. 10. A conflict resolution problem with 70 agents in narrow space, for which the proposed policy provides a correct solution. Initial configurations are identified by the presence of gray circles, indicating their reserved disks.

### C. CAC Interface Implementation

The implementation of interface for the CAC with other components (including the localization component described before) consists of the following three core functions (auxiliary interface functions are not described here for the case of brevity).

#### C.1 Parameter Initialization

Syntax:

```
setup_parameters(int16 Safety_Disk_Radius,
   int16 Reserved_Disk_Radius,
   int8 Vehicle_Speed, struct process
   *subscriber_process);
```

This function initializes parameters necessary for the correct execution of each CAC iteration. The safety of the policy can be compromised if these parameters are not properly initialized. The `Safety_Disk_Radius` and the `Reserved_Disk_Radius` are two numeric parameters related to the agent dimension and technical constraints and represent the radii of the safety disk and the reserved disk, respectively. The definition of these quantities is tightly related to the uncertainties of the environment and the maneuvering capabilities of the agent. The `Vehicle_Speed` is the desired forward velocity of our agent. The necessity of defining the speed of the robot is related to the possibility of the CAC to be directly bounded to the motion control component.

The `subscriber_process` is a pointer to the process that the CAC will activate after the vehicle reaches the goal position. The RUNES middleware for the Contiki operating system, even though very efficient for network stacks and multitasking, has a drawback in the scheduling process. Indeed, a process explicitly needs to block over an event to introduce a precise order in the scheduling. As programming paradigm, every time a callback parameter is needed, the code explicitly blocks on a PROCESS_WAIT_UNTIL procedure. The invoked function that comes to an end can awake the calling process with a PROCESS_POST

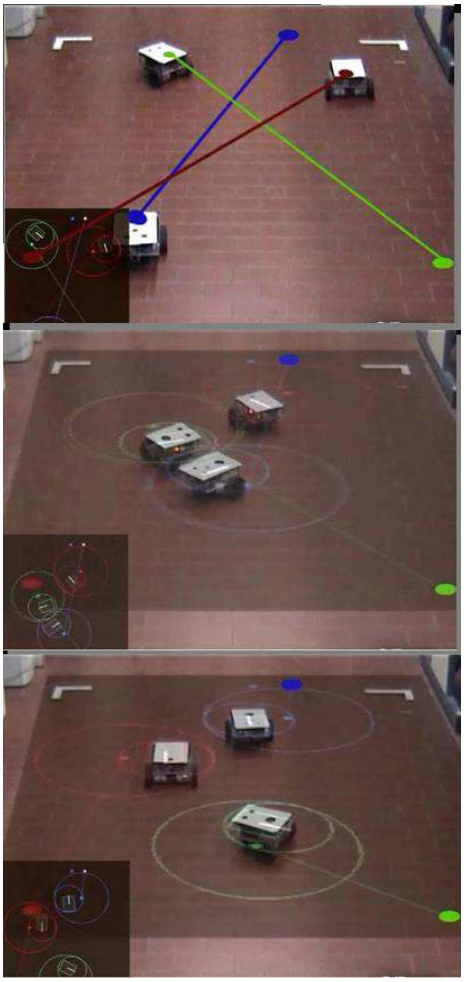

Fig. 11. Snapshots of GRP evolution in a three-vehicle scenario

procedure with `subscriber_process` as the parameter.

*C.2 Get Parameters*

Syntax:

```
get_parameters(int16 Safety_Disk_Radius,
   int16 Reserved_Disk_Radius,
   int8 Vehicle_Speed,
   strict process *subscriber_process);
```

This function returns the current value for each parameter that has been initialized. As mentioned in the previous section if agents have different dimensions or kinematic constraints, it is necessary to exchange information also on the radii used in the CA policy with the neighboring agents. Hence, each agent that wants to implement the CA policy must declare these parameters to its neighbours. As mentioned, the `subscriber_process` will receive an explicit PROCESS_POST as the current values are copied in the returned parameters.

*C.3 Set Goal Configuration*

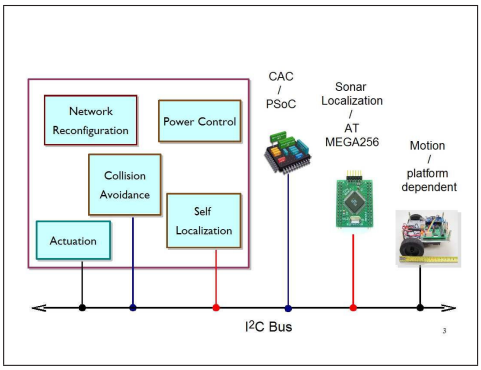


Fig. 12. Hardware Expansion Layout. All external hardware communication is performed by I2C bus sharing.

Syntax:

```
goal_specification(int16 X, int16 Y,
   int16 orientation);
```

This function sets the goal position of the vehicle. The position is specified in Cartesian `X,Y` coordinates and orientation angle. When the `goal_specification` is called, the CA algorithm begins to iterate. When the final position is achieved or some error occurs, the `subscriber_process` is awakened.

*D. Receptacles*

The receptacles of the CAC are strictly related to the localization component that gives information about the position of the neighbors and of the agent itself, and the motion control component, that provides the basic control that allows the agent to reach the desired position.

The *Localization Receptacle* defines functionalities for acquiring agents' position. The self localization receptacle is related to the agent position, while the neighbours localization receptacle asks for neighbours current configuration. The two services can be implemented by different technologies for different components. In our case, the described localization component provides both functionalities.

*D.1 Receptacle toward Localization Component*

The receptacles of the CAC component related to localization correspond to the localization interface functions defined in Section II (i.e., `get_self_position` etc)

*D.2 Receptacle for Actuation: Motion Control*

Syntax:

```
void set_steering(int8 heading, int8 speed);
```

The receptacle sets the desired heading and speed of the vehicle. These parameters are set-points for the motion controller of the robots.

### E. Hardware extensions

To overcome the limitations of the Tmote Sky board for complex control tasks, the RUNES partners have agreed to use a common paradigm for extending the capabilities of the experimental platform with dedicated hardware. As shown in Figure 12 all external computational resources and devices (such as sonar sensor boards, actuation control electronics) are connected over the $I^2C$ bus provided by the Tmote Sky board. The Tmote Sky board acts as the master of the bus according to requests that the component functionalities demand to external devices. The proposed CA policy has memory and computation needs that can hardly fit into Tmote Sky micro-controller. The same is true for the part of the localization component residing on the mobile robots.

The proposed solution is to implement the CA policy on a separate micro-controller of the PSoC 29 series. The micro-controller is connected over $I^2C$ and all functionalities are controlled by a CAC on the Tmote Sky. This solution allows to shrink the CAC footprint to 3KB. On the micro-controller, the code is about 15KB occupancy on ROM and 200 bytes on the RAM. Partners that want to implement the CA policy over an embedded systems may run the algorithm on the main CPU or use an external micro-controller. With the proposed implementation we have been able to achieve more than 50 iterations per second. A similar situation holds for the localization component. Although it does fit on the Tmote Sky processor it may be desirable to migrate it completely, or partly, to one or several micro-controllers. On the Lund robot an ATMEL AVR Mega128 processor is used as co-processor to the Tmote Sky. The extended Kalman filter can either execute on the Tmote Sky or on the AVR depending on memory and speed requirements.

## IV. Conclusions

Component-based techniques in software developments have many benefits: components are well-defined entities that can be replaced without affecting the rest of the systems, they can be developed and tested separately and easily integrated later, and they are reusable. These features are important for the development of large-scale complex systems. Component-based approaches for embedded applications such as sensor networks or mobile robots create special challenges.

Within the RUNES project a component-based approach has been adopted in a combined sensor network and mobile robot application. Two of the key components are the localization component and the collision avoidance component. The principles and implementation of these components have been described in this paper.

### A. Acknowledgment

The work has been done with partial support from EC project RUNES (Contract IST-2004-004536).

## References

[1] K.-E. Årzén, A. Bicchi, G. Dini, S. Hailes, K. Johansson, J. Lygeros, and A. Tzes, "A component-based approach to the design of networked control systems," in *Proceedings of the European Control Conference (ECC), Kos, Greece*, Kos, Greece, 2007.

[2] A. Smith, H. Balakrishnan, M. Goraczko, and N. B. Priyantha, "Tracking Moving Devices with the Cricket Location System," in *2nd International Conference on Mobile Systems, Applications and Services (Mobisys 2004)*, Boston, MA, June 2004.

[3] R. Want, A. Hopper, V. Falcao, and J. Gibbons, "The Active Badge Location System," *ACM Transactions on Information Systems*, vol. 10, no. 1, pp. 91–102, 1992.

[4] J. Cadman, "Deploying commercial location-aware systems," in *Proc. Fifth International Conference on Ubiquitous Computing*, October 2003.

[5] N. Priyantha, A. Chakraborty, and H. Balakrishnan, "The Cricket location-support system," in *Proc. Sixth ACM MOBICOM Conf.*, Boston, MA, August 2000.

[6] A. H. Jazwinski, *Stochastic Processes and Filtering Theory*. Academic Press, 1970.

[7] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: Simplifying event-driven programming of memory-constrained embedded systems," in *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, Colorado, USA, Nov. 2006. [Online]. Available: http://www.sics.se/~adam/dunkels06protothreads.pdf

[8] L. Pallottino, V. G. Scordio, E. Frazzoli, and A. Bicchi, "Decentralized cooperative policy for conflict resolution in multi-vehicle systems," *IEEE Transactions on Robotics*, May 2006, subm.

[9] L. Pallottino, V. Scordio, E. Frazzoli, and A. Bicchi, "Probabilistic verification of a decentralized policy for conflict resolution in multi-agent systems," in *ICRA06*, 2006.

[10] E. Frazzoli, L. Pallottino, V. G. Scordio, and A. Bicchi, "Decentralized cooperative conflict resolution for multiple nonholonomic vehicles," in *Proc. AIAA Guidance, Navigation and Control conference*, 2005.

[11] A. Danesi, A. Fagiolini, I. Savino, L. Pallottino, R. Schiavi, G. Dini, and A. Bicchi, "A scalable platform for safe and secure decentralized traffic management of multiagent mobile systems," in *ACM Workshop on Real-World Wireless Sensor Networks*, June 2006, Uppsala, Sweden.