Design and modeling of the multi-agent robotic system: SMART

Cecilia García , Pedro F. Cárdenas , Lisdandro J. Puglisi , Roque Saltaren

A B S T R A C T

This article presents the design, kinematic model and communication architecture for the multi-agent robotic system called SMART. The philosophy behind this kind of system requires the communication architecture to contemplate the concurrence of the whole system. The proposed architecture combines different communication technologies (TCP/IP and Bluetooth) under one protocol designed for the cooperation among agents and other elements of the system such as IP-Cameras, image processing library, path planner, user Interface, control block and data block. The high level control is modeled by Work-Flow Petri nets and implemented in C++ and C♯. Experimental results show the performance of the designed architecture.

Keywords:
Communication architecture
Multi-agent robotic systems
Cooperative robots
Concurrent system

1. Introduction

Scientific interest in multi-agent robotic systems (MARS) has increased considerably during the last decades [1–5]. This interest is based on the diversity of disciplines that are related to the robotics field involved in giving solutions to great problems of MARS like perception, cognition, behavior, coordination and configuration among others.

Distributed Artificial Intelligence (DAI), integrates two fields of knowledge: artificial intelligence and distributed systems. Therefore, DAI arises as a field of knowledge that attempts to build intelligent and autonomous agent sets that cooperate in developing their tasks and communicating among them through mechanisms based on the messages sent and received [6]. Basically, DAI is based on these three main concepts:

- Multi-agent systems (MAS), the level at which the behavior of intelligent agents that cooperate for solving problems is studied;
- Distributed Solving Problems (DSP), where a task is divided conveniently into subtasks and then assigned to a set of independent entities in order to find together the solution, and
- Parallel Artificial Intelligence (PAI), seeks the development of languages and parallel algorithms for concurrent systems.

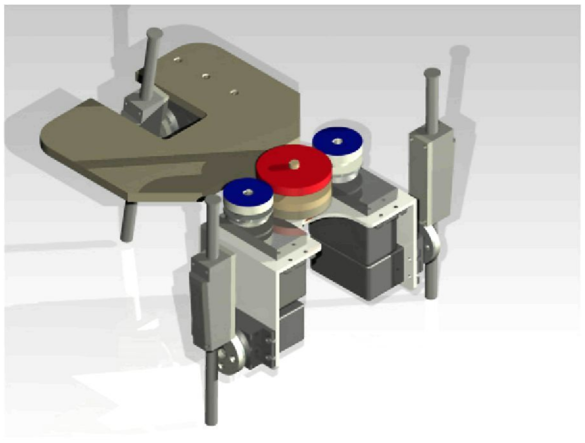
The integration of a heterogeneous set of agents into MAS requires three important concepts: communication, cooperation and coordination [7,8]. The objective of coordination and cooperation is to achieve a harmonious adjustment of individual jobs for the benefit of a common goal.

The kinematic and dynamic characteristics of any robotic agent are substantially different to a computational agent, so the coordination and cooperation techniques applied in MAS are not the most suitable for the treatment of uncertainty in a robotics system [1]. Some of the requirements imposed on agents for operating in real environments are the following [9]:

- To know how to behave under different circumstances (situated behavior), and react to unknown events.
- Tasks should be done with efficiency and in real time.
- The presence of other agents in the environment must be considered.

The SMART multi-agent system is composed of three-leg and four-leg walking robotic modules with 8 and 14 degrees of freedom respectively, (see Fig. 1).

This article presents the SMART System, a multi-agent system designed for studding cooperative tasks between robots (see Fig. 2). The systems have robots with three and four legs. The capabilities of each system is different and therefore we cannot develop the same kind of task. The complete kinematics model of the leg is presented in this article. This analysis allows designing a “walking-protocol”. The cooperation strategies are modeled by using Work-flow Petri Nets and the real implementation is performed in C++ and C#. The hardware and software architecture


(a) 3-leg walking robot (Robot3L).

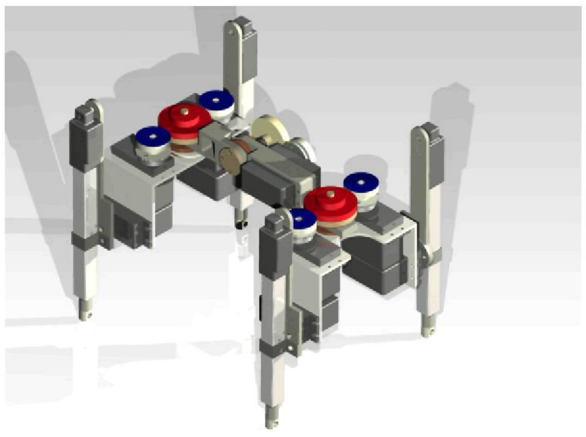

(b) 4-leg walking robot (Robot4L).

Fig. 1. CAD representation of the robotic modules.

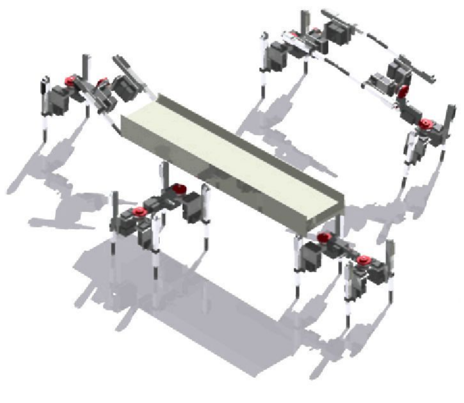


Fig. 2. Example of cooperative behavior.

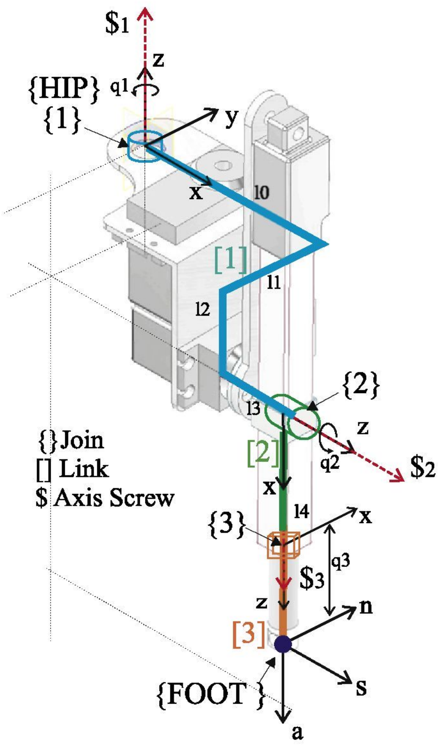


Fig. 3. Leg forward kinematics.

are developed in detail. This work is organized as follows. Section 2 presents the inverse and direct kinematic model of the leg. The direct kinematic model is found via the Successive Screw Displacement Method. Then the workspace of the leg is found and finally instantaneous kinematics is introduced. In Section 3 the high level control architecture and the multi-task model are presented. In Section 4 the communication architecture and the user interface are developed and the experimental results are shown in Section 5. Finally conclusions and future works are presented.

2. Kinematic model of the leg

One of the most relevant topics of a walking robot is the design of its legs. It must be focused not only on their individual behavior but also on the overall behavior of the robot, considering the desired task and posture that the robot has to achieve.

The new SMART robotic walking agents propose a four-legged mechanism, with 3 D.o.F. for each leg composed of two rotational joints (named **hip** and **knee**), and a prismatic joint for extending the leg, named **ext** (see Fig. 3).

2.1. Direct kinematics

The direct kinematics model of the leg is obtained by applying a successive screw displacement method [10]. This method is based on the identification of the screw axes parameters ($), the reference position (P_o) and the target position (P_{ef}).

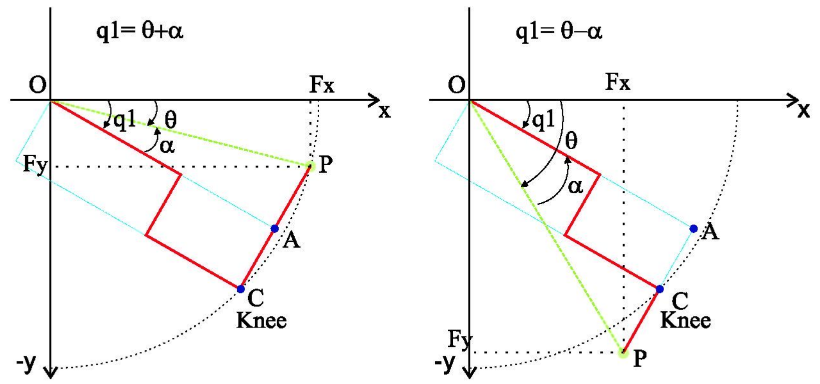
Let us consider as the reference position of the mechanism the one presented in Fig. 3. Let us attach a fixed frame O_{xyz} placed on the **hip** joint, and place one screw axis in each joint with the same direction of the joint axis. Then, the targeted position can be found according to (1).

Table 1
Screw axis parameters.

Joint	s_i	θ_i	s_{0i}	d_i
1	$(0, 0, 1)$	q_1	$(0, 0, 0)$	0
2	$(1, 0, 0)$	q_2	$(0, -l_1, -l_2)$	0
3	$(0, 0, -1)$	0	$(l_0 + l_3, -l_1, 0)$	q_3

$$\mathbf{p_{ef}} = A_1 A_2 A_3 \, \mathbf{p_o}, \tag{1}$$

where A_i is the ith screw transformation matrix associated to the ith-screw axis, and $\mathbf{p_o} = [l_o + l_3, -l_1, -l_2 - l_4, 1]$ are the homogeneous coordinates of the reference position. The parameters of each screw axis are presented in Table 1.



(a) Backward-Knee. (b) Forward-Knee.

Fig. 4. Possible configurations.

2.2. *Inverse kinematics*

Given the nature of the kinematic chain of the leg, two possible configurations may result from the inverse kinematic problem. These two possible configurations named: forward-knee and backward knee, depend on the selection of the state of q_1.

Let us consider an arbitrary position of the foot given by $F = [F_x, F_y, F_z]$, and let us take a closer look at the projection of the leg over the XY plane, as shown in Fig. 4, where P is the projection of F over the plane XY.

As it can be seen,

$$d_{AP} = \sqrt{d_{OP}^2 - d_{OA}^2} \tag{2}$$

$$\alpha = \arctan(d_{AP}, d_{OA}) \tag{3}$$

$$\theta = \arctan(F_y, F_z), \tag{4}$$

where $d_{OP} = \sqrt{{F_x}^2 + {F_y}^2}$, and $d_{OA} = l_0 + l_3$.

Thus, the state of q_1 for a backward-knee or forward-knee configuration is given by (5)

$$\begin{cases} q_1 = \theta + \alpha, & \textit{Knee front} \\ q_1 = \theta - \alpha, & \textit{Knee back.} \end{cases} \tag{5}$$

However, there still exists a blind gap where the selection of configuration is uncertain.

With q_1 known, the position of the knee joint can be found according to (6).

$$B = [\cos q_1 d_{OC},\ \sin q_1 d_{OC},\ -l_2], \tag{6}$$

where $d_{OC} = \sqrt{(l_0 + l_3)^2 + {l_1}^2}$.

The distance from the knee joint to the foot, defines the state of q_3, as expressed below

$$q_3 = \|\overrightarrow{BF}\| - l_4, \tag{7}$$

where $\overrightarrow{BF} = \overrightarrow{OF} - \overrightarrow{OB}$.

The state of q_2 is defined as follows,

$$\begin{cases} q_2 = \arccos \dfrac{\overrightarrow{BF}\hat{z}}{\|\overrightarrow{BF}\|}, & \textit{Backward-Knee}, \\ q_2 = -\arccos \dfrac{\overrightarrow{BF}\hat{z}}{\|\overrightarrow{BF}\|}, & \textit{Forward-Knee}, \end{cases} \tag{8}$$

where $\hat{z} = [0,\ 0,\ -1]$.

2.3. *Workspace of the leg*

The workspace of the kinematic chain is made up of all those possible positions where the foot can reach without exceeding the physical capabilities of the mechanism.

Therefore, several positions for the foot ($F = [F_x,\ F_y,\ F_z,\ ]$) are proposed, and using the inverse kinematic model of the kinematic chain, it is verified if the state of the joints resides in their work range. If the proposed position passes the verification procedure, then the position belongs to the workspace of the kinematic chain, otherwise, it is discarded.

After the evaluation of several positions taken from a rectangular box of 1 cm × 1 cm × 1 cm, and considering the real amplitude of work of each joint (see Table 2), the workspace of the kinematic chain is generated and presented in Fig. 5.

2.4. *Instantaneous kinematics*

In order to make a synchronized movement of the leg along a desired path with a prescribed speed, the motion of the individual joint has to be carefully coordinated. This coordination is achieved by relating the joint velocity space and the foot velocity space (end effector velocity space).

According to [10], the first-order instantaneous kinematics of a serial robot can be written as (9)

$$\$_n = \sum_{i=1}^{n} \dot{q}_i \hat{\$}_i, \tag{9}$$

where $\$_n = [\omega_n,\ v_0]^T$ is the resultant twist that describes the infinitesimal displacement of the end effector, $\hat{\$}_i$ is the unit twist associated to the ith joint and $\dot{q}_i$ is the intensity of the ith twist.

Expressing (9) in a matrix form, and defining $\mathbf{J} = [\hat{\$}_1, \hat{\$}_2, \dots, \hat{\$}_n]$, the instantaneous kinematic equation can be written as,

$$\dot{\mathbf{x}} = \begin{bmatrix} \omega_n \\ v_0 \end{bmatrix} = \mathbf{J} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix}. \tag{10}$$

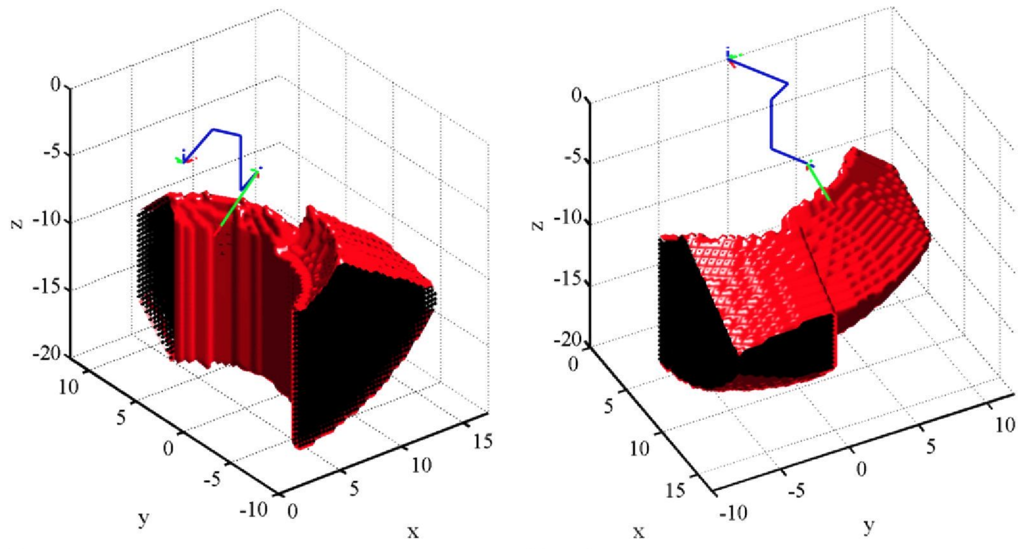
Therefore, the columns of J (from the Jacobian matrix of the kinematic chain), corresponds to the twists associated to each joint.

Considering that the general expressions of an unit twist for a revolute joint and a prismatic joint is given by (11) and (12) respectively,

$$\hat{\$}_0 = \begin{bmatrix} \mathbf{s}_n \\ \mathbf{s}_o \times \mathbf{s} \end{bmatrix} \tag{11}$$

$$\hat{\$}_\infty = \begin{bmatrix} \mathbf{0} \\ \mathbf{s} \end{bmatrix}, \tag{12}$$

where $\mathbf{s}$ and $\mathbf{s}_o$ are given in Table 1, the Jacobian matrix of the leg can be expressed as follows,



(a) WS a. (b) WS b.

Fig. 5. Leg's Workspace.

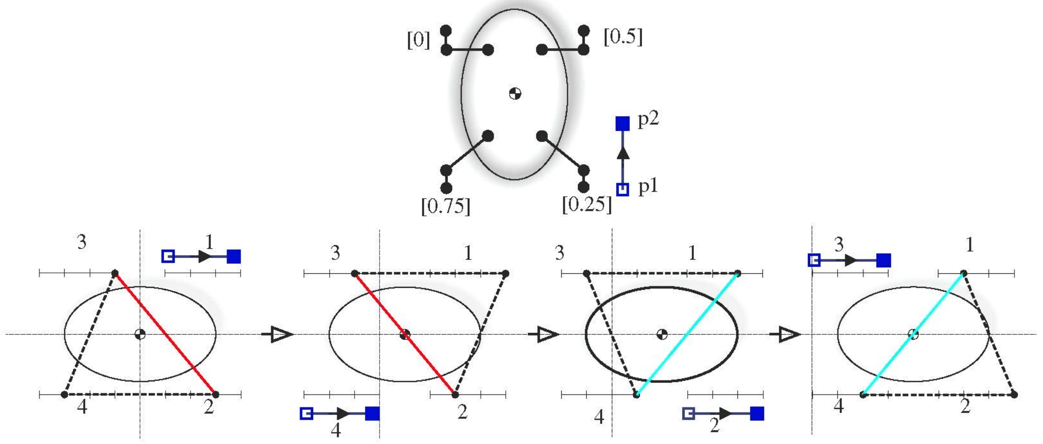


Fig. 6. Walking gait of the SMART robotic agent.

Table 2
Amplitude of work for each joint.

Joint	Min	Max	Unit
q_1	$-\pi/3$	$\pi/3$	rad
q_2	$-\pi/3$	$\pi/3$	rad
q_3	0	100	mm

$$J_0 = \begin{bmatrix} \omega_f \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 & \cos(q_1) & 0 \\ 0 & \sin(q_1) & 0 \\ 1 & 0 & 0 \\ 0 & l_2 \sin(q_1) & -\sin(q_1)\sin(q_2) \\ 0 & -l_2\cos(q_1) & \cos(q_1)\sin(q_2) \\ 0 & l_1 & -\cos(q_2) \end{bmatrix}. \tag{13}$$

However, v_0 is the linear velocity of a point p_0 in the end effector that is instantaneously coincident with the origin of a reference frame in which the twists are expressed [10]. Therefore, the velocity at any point ${}^0p_f = [p_x,\ p_y,\ p_z]$ will be given according to the following expressions, (15).

$$\begin{bmatrix} w_f \\ v_f \end{bmatrix} = \begin{cases} {}^0\omega_f = {}^0\omega_f \\ {}^0\mathbf{v}_f = {}^0\mathbf{v}_0 + {}^j\omega_n \times {}^0\mathbf{p}_f \end{cases}. \tag{14}$$

Taking into consideration (14), the Jacobian matrix can be rearranged, and expressed as,

$$J_f = \begin{bmatrix} 0 & \cos(q_1) & 0 \\ 0 & \sin(q_1) & 0 \\ 1 & 0 & 0 \\ p_y & \sin(q_1)(-p_z + l_2) & -\sin(q_1)\sin(q_2) \\ -p_x & (p_z - l_2)\cos(q_1) & \cos(q_1)\sin(q_2) \\ 0 & -p_y\cos(q_1) + p_x\sin(q_1) + l_1 & -\cos(q_2) \end{bmatrix}. \tag{15}$$

And the instantaneous kinematic equations can be rewritten as,

$$\begin{bmatrix} {}^0\omega_f \\ {}^0\mathbf{v}_f \end{bmatrix} = J_f \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix}. \tag{16}$$

2.5. Walking pattern for each leg

Fig. 6 shows the robot's movements while performing a walking cycle. Additionally, the initial position of the robot agent and the movement of each leg can be observed. The transfer phase is represented by the initial point ■ and the final point (□). Both points are linked by a straight line and the direction of the movement is shown by the arrow.

In order to study the timing of the leg's trajectory, it is necessary to define the path to be followed by the foot as a function of time. The movement of the foot should be done smoothly and

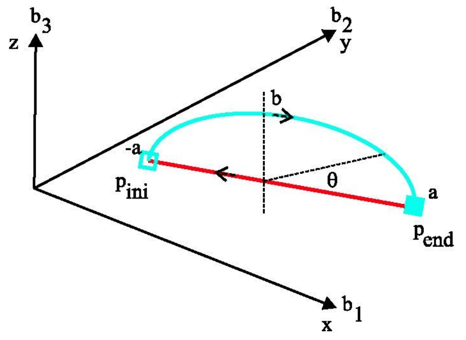


Fig. 7. Elliptical movement of robot's foot.

continuously and at the second-order to be at least differentiable. The smooth movement of the foot implies that the movement of the robot's trunk is also smooth. To ensure the smooth foot trajectory it is required to observe spatial and temporal restrictions imposed on the kickstand. Some criteria to select the path are listed below.

- The orientation of the curve has to be normal to the ground during the upward or downward movement of the supporting foot.
- The second derivative of the curve should be continuous.

The walking mode is solved by fixing the paths that make the robot's feet; i.e. using the same path for all legs and setting an identical relative gap. In the Smart system it is proposed the legs will follow an elliptical path. This path fulfills both restrictions imposed on the motion curve. Fig. 7 shows this trajectory where the marked *b* point is limited by the maximum and minimum length of the third actuator.

In order to complete the leg's movement, it is necessary that the leg comes back to the initial position after the relative gap. Since the leg is in contact with the ground, it is assumed that the path described by the leg is a straight line, which should join the starting and ending points (p_{ini} and p_{end}) of the gap (see Fig. 8).

3. Description of the multi-task architecture

Fig. 9 shows the control scheme implemented in the SMART project. The task planning and control are centralized in a PC that acts as the master and sends out commands in order to control the modules simultaneously. Closing the control loop, a commercial IP camera is placed at the top of the scene where the robots move. This camera sends images (in bmp format) to the controlling software in the master computer.

The free library OpenCV (special functions for matrix processes and position recognition) for image processing was used. However, a especial software was developed for the orientation of the robot. With this visual information the control software can recognize each module, its position and orientation and, also, the obstacles present in the scenario. After that, it is possible to plan the path for each robot. Path calculation is based on computer vision techniques. The images that the system receives from the colored camera are the input to the computer vision system. Using a binary matrix, created in the computer vision part using OpenCV libraries, the system can search for the shortest path between two coordinates.

The matrix gives a binary view of the image. All obstacles are set to 1 and all other coordinates are set to 0. Objects and other SMART robots should be avoided. When all objects and robots are detected, the objects are expanded, in order to build a safe zone. The path planner only uses the free spaces to find a path. When moving a single robot, the other SMART robots can be seen as obstacles too. When calling the path planner function, a vector with the coordinates to move the robot to a certain position is returned.

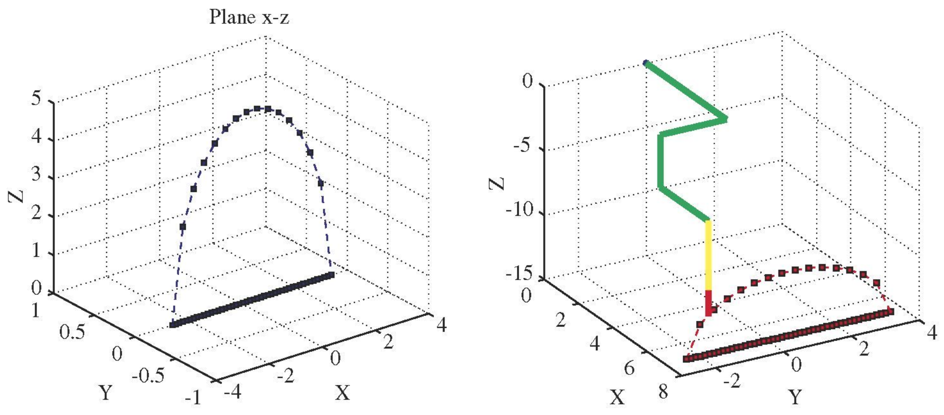
Finding all obstacles and robots in a System is not only important but they should also be identified. Fig. 10 shows a processed imaged after being taken by the camera. The trajectory generated by the path planner in order to avoid obstacles can be observed here. Based on the information given by the path planning function, the controlling block will send the commands to the robots so they can move in the correct direction. The commands can be forward, turn left or right movements. These basic movements for the robots are in text-files. Going to another coordinate can be a movement containing multiple basic movements.

3.1. Communication structure

The communication part is created in C++ and a UML scheme is shown in Fig. 11. The Socket class includes all functionality to use sockets in a client and server mode. The socket can be bound to the address of the server using the **connect()** function. Sending and receiving data can be done by using the **read()** and **write()** system calls.

The typical server does not initiate the connection. Instead, it waits for a client to call and request for services. The server can be established by using the **listen()** and **accept()** functions.

The Camera class uses the functionality of this socket to send, order the web cam and receive information from the socket. The communication block in the engine also uses the server side functionality of the Socket class to listen for incoming commands from the graphical user interface.



(a) Elliptical Trajectory designed for the leg.

(b) Trajectory for a walking gait.

Fig. 8. Trajectory of the leg in the task space.

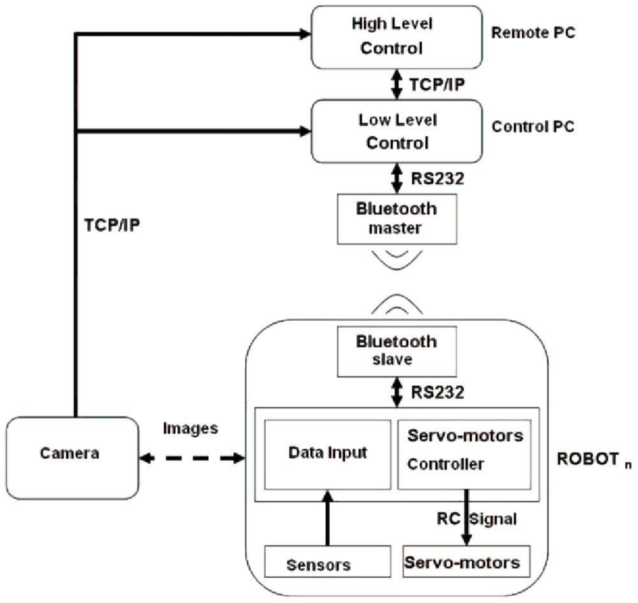


Fig. 9. Hardware and software architecture.

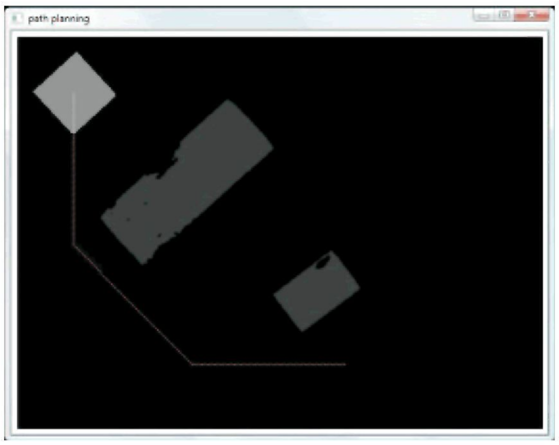


Fig. 10. Hardware and software architecture.

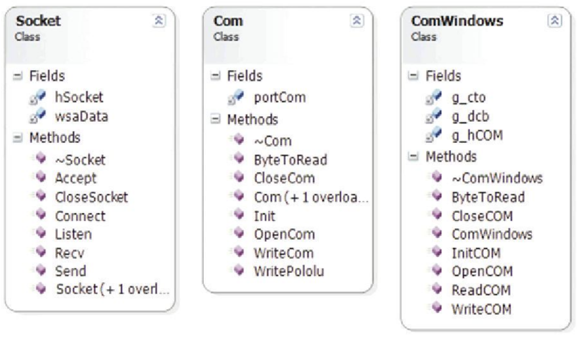


Fig. 11. UML diagram for the communication architecture.

Bluetooth communication is established in the Com and ComWindows classes. These classes contain the C++ functionality in order to bind a Bluetooth device to a certain COM port. Data can then be sent to the Bluetooth device using the **WriteCom()** function.

The SMART control software, called the *engine*, is a C++ programmed Win32 console program. It contains all the parts needed to control the SMART robots. The most important parts of

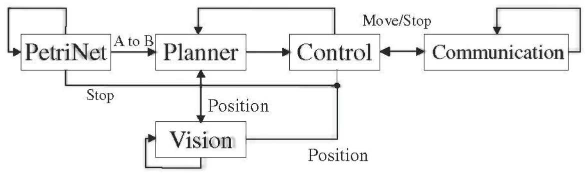


Fig. 12. Scheme of the ENGINE architecture.

Table 3
OEMSPA 312i technical data.

Power class	1, +7 dBm (50m)
Processor	BGB203 Philips chipset
Supply voltage	3–6 Vdc
Power consumption	1 mA (min), 17 mA (avg), 70 mA (max)
External dimensions	23 × 36 × 5 mm
Environmental conditions	Max operating T: −30 to +85 °C

the engine could be the communication net, the computer vision, the path planner and the control. The engine has a multi-threaded architecture. This makes a good amount of task-level parallelism (TLP) available. After modeling tasks by Petri Net [7] they can be implemented and executed thought a multi-threaded architecture. This advantage of a multi-threaded program allows it to operate faster. The engine architecture is given in Fig. 12. The interaction between the main parts in the engine is based on the control of threads and call function. The interaction between the main parts included in the control software can be seen here. In a modular robotic system, agents should be able to communicate through messages. More complex modular robot structures are made of a high amount of individual units. A robust communication protocol is crucial. The communication architecture has to take into account possible mechanical or electronic failures of a module.

Bluetooth communication. The SMART environment uses Bluetooth with GFSK modulation to communicate among SMART robot and control software. By using Bluetooth-based communication the SMART robot developers are able to separate the communication from the mechanical connectors. Wireless communication is suitable for self-reconfiguring modular robots. Socket communication is used between the client user interface and the control software (kernel) and they receive images from the IP webcam. The PC running the engine is equipped with a Belkin Bluetooth USB adapter. The Belkin F8T009 Bluetooth adapter supports Bluetooth version 1.2. SMART robots are equipped with the OEMSPA 312i adapter by ConnectBlue. Once it is connected to its host system and configured, the Serial Port Adapter can communicate, using Bluetooth, with a wider range of other Bluetooth devices such as other Serial Port Adapters, mobile phones, handheld computers and laptops. The OEM Serial Port Adapter Electrical & Mechanical Datasheet contains important information about the OEM Serial Port Adapter. A list of the characteristics of the adapter is given in Table 3.

For the communication with the serial port adapter, the baud rate is changed to 9600 since the Pololu mini servo controller used for controlling the SMART robot servos, works at 2400 or 9600 baud. This can be done, using the OEMSPA 312i Serial Port Adapter in the AT mode.

Thread synchronization. The ability in executing more than one process at a time is known as multi-processing. A thread is another mechanism for splitting the workload into separate execution streams. A thread is lighter in weight than a process. The engine in the SMART environment manages the different threads used in the environment. Synchronization of threads is considered, so that they do not break while executing on resources and memory.

Threads can be easily created and destroyed. They can be given different attributes. While starting up, the main thread starts two other threads: an image processing thread (called the vision thread) and a communication thread. The processing thread includes the computer vision part. It gets an image from the webcam, using a socket and does the processing using OpenCV. The thread runs in a never-ending loop. In order to reach synchronization, mutual exclusion is used. The communication thread acts as a socket server and can receive commands from a user interface. While receiving a command in the communication thread, some other threads can be started such as the controlling block. The controlling block can call the path planning function to update path calculation. The program creates a new thread every time a new action is sent by the user interface. When the previous action for a specific robot is not yet finished, the thread is killed and a new thread is started. The previous command will never be finished.

3.2. *Socket communication*

The communication that occurs between the client and the server must be reliable. That is, no data can be dropped and have to arrive on the side of the client in the same order as sent by the server. TCP provides a reliable, point-to-point communication channel for those client–server applications on the Internet.

The structure designed for the SMART environment, supports command communication from client to server using a socket on TCP protocol. The commands have a defined structure including the action identifier, some parameters and a character indicate the end of a command. The protocol designed for this communication can be extended when new commands are included in the system. The action identifier is a single number that is unique for each service the server fulfills. The command associated with a given identifier can take some parameters divided by a comma. While the protocol is a sequence of seven bytes, the % sign is used for dividing different commands [8].

The first byte is the command identifier. Different Petri Net tasks can have different command identifiers. The ID of the SMART robot is given in the second byte, followed by a comma to separate the data. Bytes four and six are the parameters for the command i.e. when the command is for a certain robot to move to a point, the parameters will be an X and Y coordinate. Socket communication is also used for receiving images from the webcam as an input for the computer vision delivered by the controlling software. The webcam acts as a server, sending images over the socket to the kernel reacting on HTTP GET/POST requests from the controlling software. The images are saved as bitmap images with the maximum resolution.

3.3. *Petri net in SMART system*

The behavior modeling for both the individual agent and the group are implemented with Petri Nets (PN). SMART is a heterogeneous system since the robots, the IP-camera, the image processing software and the control software are considered as agents of the system. As a consequence,

Definition. In the SMART System, an agent is a collection of software and hardware elements that are able to cooperate in order to reach a common objective.

The classical Petri net allows modeling of states, events, conditions, synchronization, parallelism, choice, and iteration. For this reason, the PN is widely used in multi-agents modeling [11–13, 5]. Even though, the agents might be treated as software systems, the modeling concept can be taken to robot agents [14,15]. On the other hand, the formal theory of PN allows evaluating the systems behavioral properties, which is a way for its application to be much more generalized.

In a strict sense, a PN is defined as an n-tuple $N = \{P, T, F, W\}$, where P is a set of states, T is a set of finite transitions. Both sets satisfy $P \bigcap T = \emptyset$ and $P \bigcup T \neq \emptyset$. The weight of the arch f_{ik}, that bounds the place p_i with the transition t_k is defined as $w \in W \longrightarrow Na+$ where $Na+ = \{1, 2, \dots.\}$ is the set of positive integers. A mark M over an N net, is a mapping $P \longrightarrow Na$. $M(p_i)$ defines the number of marks at $p_i \in P$. A marked PN, is defined as $N_M = \{N, M\}$ and the initial mark is noted as M^0.

A place p is called an input place of a transition t if there exists a directed arc from p to t. Place p is called an output place of transition t if there exists a directed arc from t to p. We use $\bullet t$ to denote the set of input places for a transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g., $p\bullet$ is the set of transitions sharing p as an input place.

Most workflow systems support a separate model of the workflow process from the modeling of the structure of the organization and the resources within the organization [16]. The reasons for decoupling these two dimensions when specifying a workflow are: the complexity is reduced, the reuse is stimulated, and a process without changing the organizational model (and vice-versa) can be modified. In the process dimension, the tasks needed to be executed and the order is specified. Modeling a workflow process definition in terms of a Petri net is rather straightforward: tasks are modeled by transitions, conditions are modeled by places, and cases are modeled by tokens.

Definition WF–PN. A Petri net $P_N = (P, T, F)$ is a WF-net (WorkFlow net) if and only if:

- P_N has two special places: an input place, p_{in}, and an output place p_{out}. Place p_{in} is a source place: $\bullet p_{in} = \emptyset$. Place p_{out} is a sink place: $p_{out}\bullet = \emptyset$.
- If a transition t^* to PN is added which connects place p_{out} with p_{in} (i.e. $\bullet t^* = \{p_{out}\}$ and $t^*\bullet = \{p_{in}\}$), then the resulting Petri net is strongly connected.

As it was referred before, WF–PN are implemented as a multi-threaded architecture in C++, and the synchronism is managed by the kernel. The reader is referred to [17] for details. In Section 4, the authors present two examples of behavior that show the manner of modeling using workflow Petri Nets.

3.4. *User interface*

In order to improve system control and viewing, a user interface was developed. This interface is a graphical program that allows the user to send commands to the robots and it gives a view of the system environment.

The commands given in the user interface are sent to the engine which takes control over the robots. Communication between user interface and engine is based on socket communication. Using sockets, it is possible to create the user interface in any programming language that support socket communication like Java, C++ or C#. For programming the client, a C# Windows application is chosen.

When creating a user interface, usability is one of the main problems. In the SMART environment, the user can interact with different SMART robots at the same time. In order to implement the different functions in a single user interface, a multiple document interface (MDI) is used. The disadvantage of MDI is the lack of information about the current opened windows. In order to view a list of windows open in MDI applications, the user typically has to select a specific menu. For this reason a window organizer was implemented. The interface is shown in Fig. 13. The user interface offers all functionalities for a user to run the robots based on Petri net methodology. The user has the option to move a robot from the current position to a new coordinate. This command uses

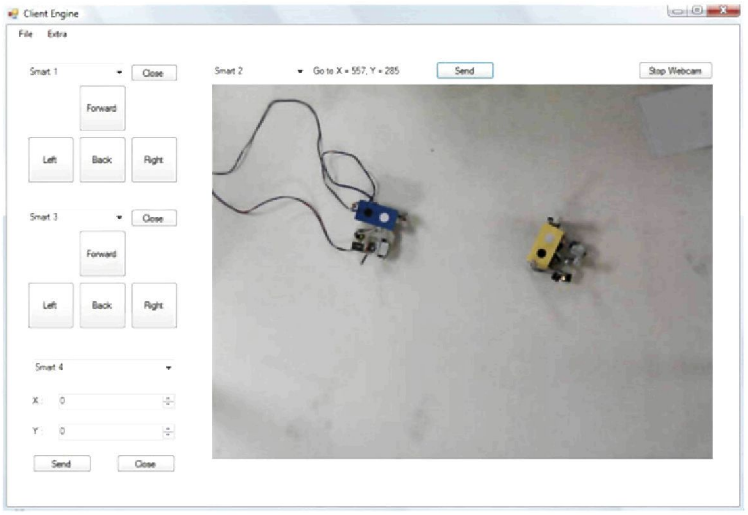


Fig. 13. View of the user interface developer in C#.

the path planner available in the engine. Another functionality offered by the client is 'free ride'. This option allows the user to send basic movement commands to the robot. These actions send the correct command to the controlling software using a socket communication and client communication protocol as discussed in the following section.

4. Experimental result

In this section two complete examples about modeling behavior of SMART's agents are developed. One of the objectives of this project is to move a set of SMART robots in a coordinated way to complete a predefined task. SMART agents move inside a metallic structure that has a colored IP-Camera placed 2 meters on its top. The camera takes the actual pictures in the host computer where the engine software runs and carries out recognition and proper decisions based on the computer vision results. This architecture is shown in Fig. 14.

The time to process an image, taken by the camera, and to give a control order to a SMART robotic agent takes 150 ms. Moreover the robot has to complete the work, such as receiving commands, and moving the servos at the same time. If not, a buffer will be filled with unfinished commands.

4.1. Example 1: Cooperation among SMART agents for helping a robotic agent in avoiding an obstacle

The tasks that every agent of the SMART system realize is modeled with a WF–PN. As a consequence, a more complex net that models the system in its totality would exist. As an example, a WF–PN where an agent must change its trajectory since the detection of an obstacle is schematized in Fig. 15. It can be seen that it is composed of 5 sub-WF–PN, detailed above.

- $N_M^{IP\_C}$ models the software of the image which is captured.
- $N_M^{ID\_S}$ models the image processing software, in order to get the localization of the agents and obstacles. The capture and image processing is done 11 times per second.
- $N_M^{r_i}$ models the movement of any robotic agent. When this sub-net is active, it implies that one or more agents are moving following a free path reference.
- N_M^{NCol} models the avoiding collision algorithm. If two agents are too close, they are ordered to stop and reprogram their trajectories.
- N_M^{CTRL} models the kernel of the application.

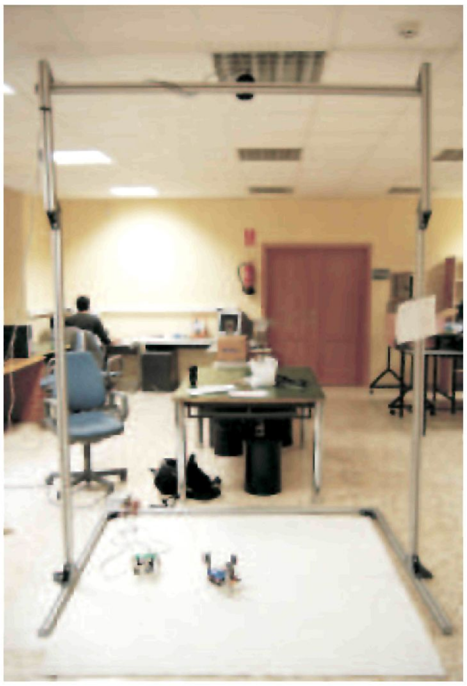


Fig. 14. Testbed for Smart project.

The sub-net that models the camera $N_M^{IP\_C}$ is integrated by the followings elements:

P_1	IP camera takes a bmp image
P_2	Waiting time for taking a new image
T_1	Send bmp image to processing block
T_2	Take a new picture
W	weighing of the arc: $W \in R^{2x1} = \{1, 1\}$
$M(0)$	initial marking $M(0) = \{0\}$.

The sub-net that model the image processing $N_M^{ID\_S}$, is composed by the elements detailed bellow:

P_3	Get an image from a kernel message
P_4	Send Information of the scene to the kernel program
P_5	Ready to process a new image
T_3	Processing image package
W	weighing of the arc: $W \in R^{2x1} = \{1, 1\}$
$M(0)$	initial marking $M(0) = \{0\}$.

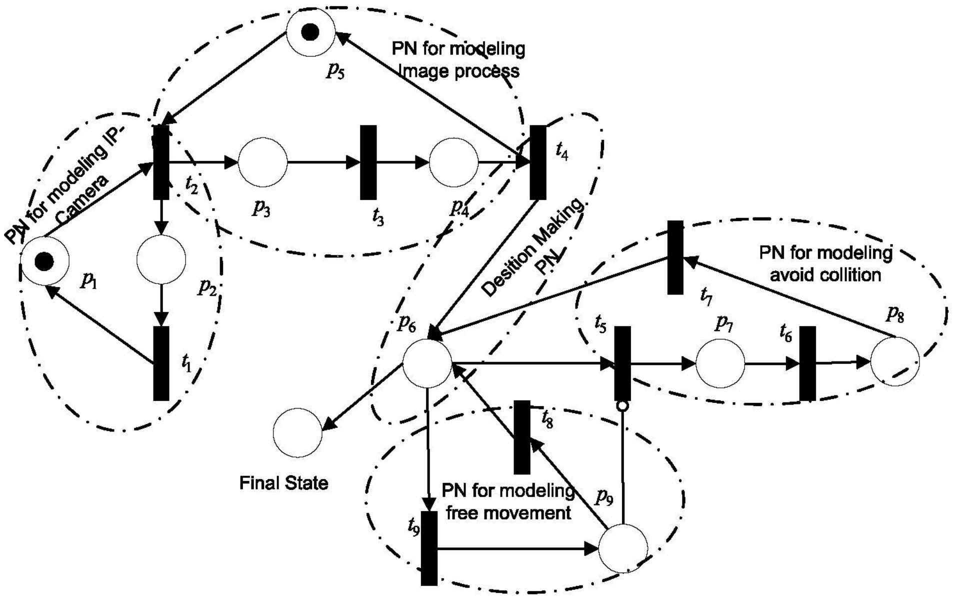


Fig. 15. Petri Net to model a cooperative task among SMART Agents. A robotic agent is helped to avoid an obstacle.

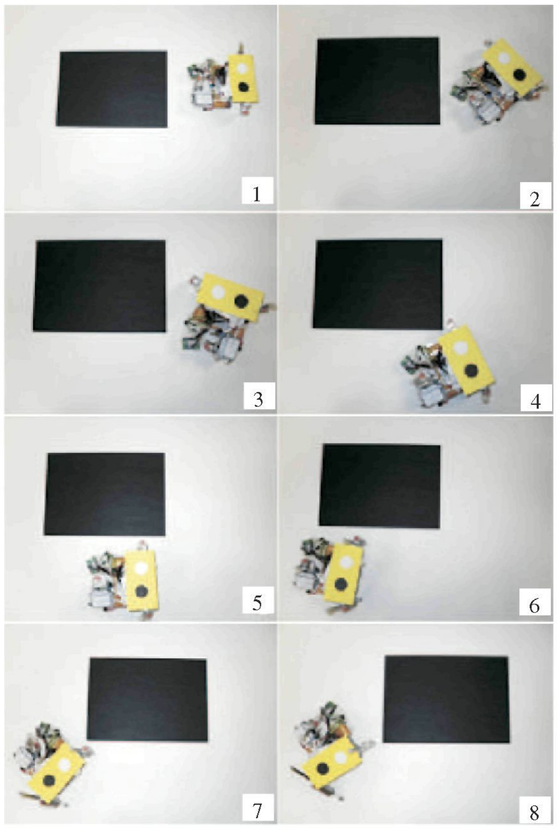


Fig. 16. Example 1: Cooperative Task among SMART Agents.

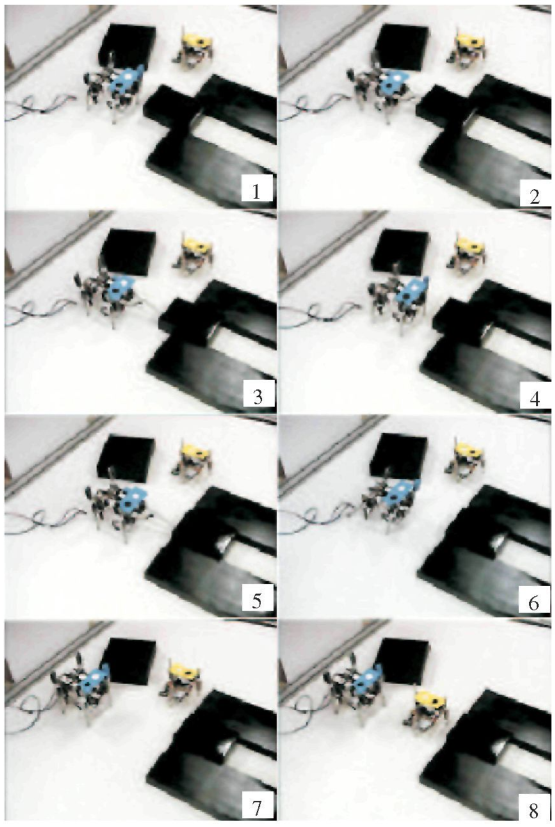


Fig. 17. Example 2: Cooperative Task among SMART Agents.

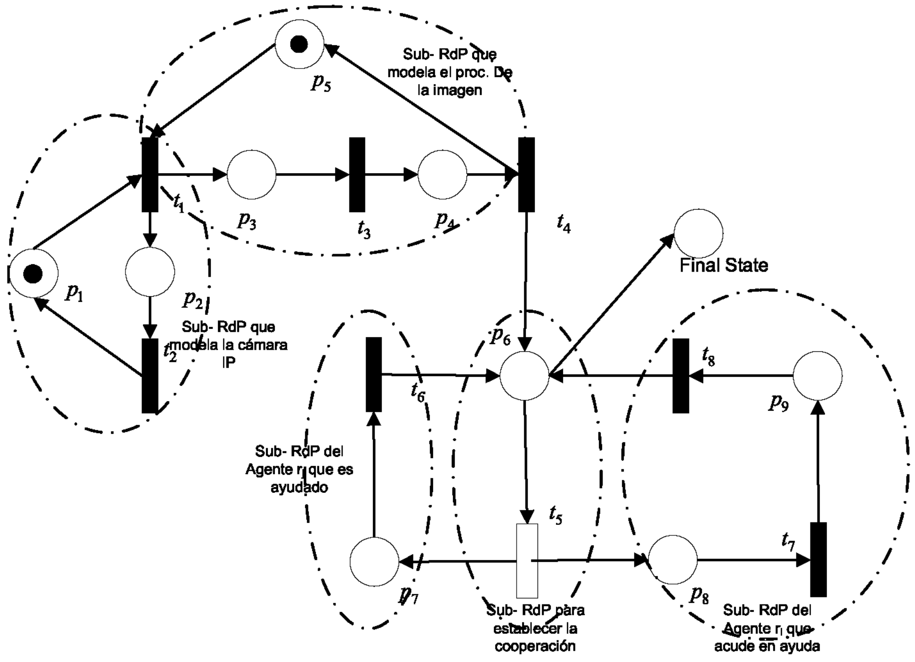
The elements that integrates the sub-net that models the control software, N_M^{CTRL} are:

P_6	Start the decision process
T_4	XOR-Split transition. Information of robot and obstacle localization is in the control block
W	weighing of the arc: $W \in R^{2x1} = \{1, 1\}$
$M(0)$	initial marking $M(0) = \{0\}$.

Transitions T_1 and T_3 make the synchronization of the visual process, that means if the process function does not finish its job, the IP camera is not going to take a new photograph.

The sub-net that models the communication interface $N_M^{r_i}$ is composed of,

T_9	Resource Fire Transition: Send an order to a robot by Bluetooth
P_8	Execute the movement order
T_9	Send a finish movement message to a decision block



Sub- RdP que modela el proc. De la imagen
p_5
t_1
p_3
t_3
p_4
t_4
p_1
p_2
Sub- RdP que modela la cámara IP
t_2
Final State
p_6
t_8
p_9
t_6
Sub- RdP del Agente r que es ayudado
t_5
t_7
p_7
p_8
Sub- RdP para establecer la cooperación
Sub- RdP del Agente r que acude en ayuda

Fig. 18. Model of Cooperative task between two SMART robotic agents using Petri Nets.

W	weighing of the arc: $W \in R^{2x1} = \{1, 1\}$
$M(0)$	initial marking $M(0) = \{0\}$.

Finally, the path planning (A^*, N_M^{NCol}) that is integrated as a routine in the kernel software is modeled with the following elements,

P_7	Start the Planning function
P_8	Place for math consistency
T_5	Resource Fire Transition. Send message to calculate a new position
T_6	Calculus of the new position
T_7	Send new position to the control block
W	weighing of the arc: $W \in R^{2x1} = \{1, 1\}$
$M(0)$	initial marking $M(0) = \{0\}$.

The above example is shown in Fig. 16. The robot agent has to move to a new point, it can detect obstacles by using vision and can consequently dodge them. The SMART robot detects the black obstacle and moves around it.

4.2. Example 2: Cooperation among SMART agents to help robotic agents to complete a task

Another task for a SMART robot, can be changing the configuration of robots and other objects. Fig. 17 shows a possible situation, where the yellow SMART with three legs has to move to a new point, but there are obstacles that prevent it from carrying out such a task. In this case, a 4-legged robot helps the yellow robot by moving the obstacles and consequently finding a path to its final point. This is another example of cooperation between the robots in a modular robotic system.

This cooperative behavior is modeled by the WorkFlow Petri Net shown in Fig. 18. In this model two new sub-nets appear to shape the cooperation between both robotic agents.

5. Conclusions

This article presents a modular robotic system called SMART. This system consists of different types of software and hardware agents. In the hardware agents there are two kinds of robots with three and four legs. In any multi-agent system, its success depends largely on its communications architecture. Therefore this article broadly describes the model and protocol used in this system. The software developed to control the system includes all functionality that was planned for the SMART robots in this first phase of the project. The software can easily be expanded in the future when new tasks will be added to the modular robotic system. The developed software has a typical modular robotics architecture. The multi-agent system guarantees good performance of cooperation tasks among agents *robots*, *camera*, *userinterface* and the communication protocol.

In the future an auto-connected architecture will be developed, that is using reciprocal communication, and robots will thereby be able to help each other. When a single robot cannot finish the task, other robots can help accomplish the task without the controlling software having to interact. In the same manner, process calculus will be added on board, so each module can take decisions by itself.

References

[1] A. Collinot, A. Drogoul, P. Benhamou, Agent oriented design of a soccer robot team, in: Proceeding of International Conference on Multi Agents Systems, 1996.

[2] S. Deloach, E. Matson, Y. Li, Applying agent oriented software engineering to cooperative robotics, in: Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference, 2002.

[3] H. Fiorino, C. Tessier, Agent cooperation: a petri net based model, Proceeding of International Conference on Multi Agent Systems 3 (1998).

[4] D. Franklin, T. Gresser, Is it an agent, or just a program? a taxonomy for autonomous agents, in: Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

[5] H. Lund, R.L. Larsen, O.s.E. Hallundbæk, Distributed Control in Self-reconfigurable Robots, Springer, 2003.

[6] L. Zhiwu, X. Shuwen, On Modeling a soccer robot system using Petri nets, in: Proceeding of the IEEE International Conference on Automation Science and Engineering, Shanghai, China.

[7] J.-S. Lee, A Petri net design of command filters for semiautonomous mobile sensor networks, IEEE Transactions on Industrial Electronics 55 (2008).

[8] M. Sims, D. Corkill, R.V. Lesse, Automated organization design for multi-agent systems, pp. 151–185.

[9] M. Veloso, D. Nardi, Special issue on multirobot systems, Proceedings of the IEEE 97 (2006).

[10] L. Tsai, Robot Analysis: The Mechanics of Serial and Parallel Manipulators, Wiley-Interscience, 1999.
[11] K. Hiraishi, An elementary model for design and analysis of multi-agent systems, proc. on coordination models and languages, in: Proceedings of the 5th International Conference on Coordination Models and Languages, 2002, pp. 220–235.
[12] K. Hiraishi, Performance evaluation of workflows using continuous petri nets with interval firing speeds, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 11 (2008) 3219–3228.
[13] Y.T. Kotb, S.S. Beauchemin, J.L. Barron, Petri net-based cooperation in multi-agent systems, in: Proc. of Fourth Canadian Conference on Computer and Robot Vision, 2007, CRV'07, pp. 123–130.
[14] D. Corkill, S.E. Lander, Modelling, analysis and execution of multi-robot tasks using petri nets agent organizations, Object Magazine 8 (4) (2008) 41–47.
[15] L. Montano, J.F. García, J.L. Villarroel, Using the time petri net formalism for specification, validation, and code generation in robot-control applications, The International Journal of Robotics Research 19 (1) (2000) 59–76.
[16] W. van der Aalst, Three good reasons for using a petri-net-based workflow management system, in: [16] pp. 179–201.
[17] C. García, R. Saltarén, J. López Blázquez, R. Aracil, Development of the user interface for the robotic multiagent system Smart, Revista Iberoamericana de Automatica e Informatica 7 (2010) 17–27.