Special Issue about Advances in Physical Agents

Miguel Cazorla and Vicente Matellan

***Abstract*—Nowadays, there are a lot of Spanish groups which are doing research in areas related with physical agents: they use agent-based technologies concepts, especially industrial applications, robotics and domotics (physical agents) and applications related to the information society, (software agents) highlighting the similarities and synergies among physical and software agents. In this special issue we will show several works from those groups, focusing on the recent advances in Physical Agents.**

***Index Terms*—Physical Agents.**

I. Advances in Physical Agents

THIS issue of the Journal of Physical Agents (JoPhA) contains a selection of papers related with Physical Agents from different Spanish groups. With this issue we start a series, which will provide a "big picture" of the current state of the art of this research at Spain. As a summary of this special issue, below you can find a brief description of the papers.

The first one *Control of Autonomous Mobile Robots with Automated Planning* presents an approach for the control of autonomous robots, based on Automated Planning (AP) techniques, where a control architecture was developed (ROPEM: RObot Plan Execution with Monitoring). The proposed architecture is composed of a set of modules that integrates deliberation with a standard planner, execution, monitoring and replanning.

The second one *Efficient Plane Detection in Multilevel Surface Maps* describes an automatic system aimed at producing a compact tridimensional description of indoor environments using a mobile3D laser scanner. The resulting description is made up of a Multi-Level Map (ML map) and a series of plane patches extracted from the MLSM. Authors propose a novel plane detection algorithm, based on the efficient RANSAC algorithm, that operates directly over the data structures of aML map and does not need to rely on the low level laser data cloud.

The third paper of this issue *Motion Planning for Omnidirectional Dynamic Gait in Humanoid Soccer Robots* deals with the problem of planning the Center of Mass (CoM) trajectory of a humanoid robot while its feet follow an omnidirectional walking pattern. This trajectory should satisfy the dynamic stability criterion to ensure analytically that the Zero Moment Point (ZMP) lies within the support polygon. The proposed approach provides flexibility and agility to humanoid robots, which is of special interest in highly dynamic environments, such as soccer robotics.

Miguel Cazorla is with the University of Alicante, Spain.
E-mail: miguel.cazorla@ua.es
Vicente Matellan is with the University of Leon, Spain.
E-mail: vicente.matellan@uleon.es

The fourth selected paper *Robust Behavior and Perception using Hierarchical State Machines: A Pallet Manipulation Experiment* presents the computational architecture and results obtained from a pallet manipulation experiment with a real robot. To achieve a good success rate in locating and picking the pallets a set of behaviors is assembled in a hierarchical state machine. The only sensory sources of information available to the robot are a binocular vision system and its internal odometry.

The fifth one *Combining invariant features and localization techniques for visual place classification: successful experiences in the robotVision@ImageCLEF competition* focus on the optional task of the RobotVision@ ImageCLEF competition, which consists of a visual place classification problem where images are not isolated pictures but a sequence of frames captured by a camera mounted on a mobile robot. This fact leads us to deal with this problem not as stand-alone classification problem, but as a problem of self localization in which the robots main sensor only captures visual information. Thus, authors base their proposal on a clever combination of Monte-Carlo-based self-localization methods with optimized versions of scale-invariant feature transformation algorithms for image representation and matching.

The last paper of this issue *Embedded Distributed Vision System for Humanoid Soccer Robot* presents a distributed architecture for humanoid visual control using specific nodes for vision processing cooperating with the main CPU to coordinate the movements of the exploring behaviours. This architecture provides additional computing resources in a reduced area, without disturbing tasks related with low level control (mainly kinematics) with the ones involving vision processing algorithms.

To conclude this introduction, we would like to thank the Spanish Network of Physical Agents (http://www.redaf.es) which provides support for all the events related with physical agents in Spain.

Control of Autonomous Mobile Robots with Automated Planning

Ezequiel Quintero Ángel García-Olaya Daniel Borrajo Fernando Fernández
Departamento de Informática. Universidad Carlos III de Madrid
Av. de la Universidad, 30. Leganés (Madrid). Spain
equinter@inf.uc3m.es

***Abstract*—In this paper we present an approach for the control of autonomous robots, based on Automated Planning (AP) techniques, where a control architecture was developed (ROPEM: RObot Plan Execution with Monitoring). The proposed architecture is composed of a set of modules that integrates deliberation with a standard planner, execution, monitoring and replanning. We avoid robotic-device and platform dependency by using a low level control layer, implemented in the Player framework, separated from the high level task execution that depends on the domain we are working on; that way we also ensure reusability of the high and low level layers. As robot task execution is non-deterministic, we can not predict the result of performing a given action and for that reason we also use a module that supervises the execution and detects when we have reached the goals or an unexpected state. Separated from the execution, we included a planning module in charge of determining the actions that will let the robot achieve its high level goals. In order to test the performance of our contribution we conducted a set of experiments on the International Planning Competition (IPC) domain *Rovers*, with a real robot (Pioneer P3DX). We tested the planning/replanning capabilities of the ROPEM architecture with different controlled sources of uncertainty.**

***Index Terms*—Automated Planning, Autonomous Robots, Robotic Architectures, Mobile Robots.**

I. Introduction

INTELLIGENT agents, such as mobile robots, are used in dynamic environments that entail sensor noise, in addition to the uncertainty of task execution on the real world and the difficulty of environment modeling. This represents a major challenge when applying any control technique to robotics.

Many approaches have been presented so far to coordinate sensing and acting in robot control. Most previous work implements reasoning in robotic tasks as reactive systems, with very little deliberation. In our work we propose the use of Automated Planning (AP) [1] to implement the deliberative step between observation and action execution. The field of AP has a remarkable activity but most of its research is done on theoretical domains of great scientific interest, that are implemented in real life only in the solution of problems within specific contracts, with private or public organizations. This is due to the complexity and cost involved on reproducing the problems of scientific interest in reality. Nevertheless, nowadays the AP field is receiving much attention from various production sectors such as logistics [2], satellites [3], [4], critical and control decision systems [5], [6], and even military operations and evacuation [7].

The difficulty of applying AP to robot control arises when we have to generate an accurate description of the control tasks, which is essential for the planning process. We overcome this challenge by including supervision to the task execution. This provides us with replanning capabilities. By including monitoring we can generate new plans to reach the final goals when the initial plan fails. So, with our work we are proposing a solution that will allow us to partially deal with the environment uncertainty (which is a common problem to all robotic control approaches) and will also allow us to apply planning to real world problems.

One of the weaknesses of most robotic control systems is that they are commonly linked to specific robotic devices. In our approach we solve this problem by separating the robot-platform control from the high level deliberation. AP techniques could also be applied to low level control but in this research we focus on high level task planning. Monitoring could also be done at the low level but for now we are just focusing on high level.

In the planning step we use an environment model - dynamically created from the sensors (low level) information - and manage high level task execution. By translating the high/low level information we achieve the mentioned control level independence. And by making use of AP techniques we benefit from its standard definition language; obtaining domain, problem and planner independence.

Our approach was tested on a real robot (Pioneer P3DX) using the International Planning Competition (IPC[1]) domain *Rovers*. This domain is inspired on the Mars exploration rover missions, and allows us to represent a set of mobile robots that can traverse waypoints on a planet, collecting samples and sending data to a lander. Problems involve task and path planning.

The rest of the paper is organized as follows. In Section II we introduce automated planning. In that Section we also introduce the Rovers domain. In Section III we describe the ROPEM architecture that allowed us to carry out our contribution. In Section IV, we discuss the experiments we conducted in order to test the proposed approach. In Section V, the related work is presented. And finally, the conclusions and future work are summarized in Section VI.

[1]IPC: http://ipc.icaps-conference.org/

II. Automated Planning

On this work, we focus on the classical AP approach (also known as STRIPS planning) with action costs. A STRIPS planning problem with action costs can be defined as a tuple $P = \{F, A, I, G, c\}$, where:

- F is a finite set of grounded predicates and functions
- A is a finite set of actions, being each $a_i \in A$ composed of preconditions establishing when the action can be applied, and effects, consisting of elements of F being added or deleted from the current state after a_i is applied
- $I \subseteq F$ is the initial state, i.e. a subset of F that represents the set of grounded literals that are true at the start of the planning process
- $G \subseteq F$ is the set of goals, i.e. a subset of F that must be true for the problem to be solved, and
- c is a function $c : A \mapsto \mathbb{R}_0^+$ that defines the cost of each action

A solution of the planning problem P is an ordered list of actions $\Pi = \{a_0, a_1 ... a_n\} | a_i \in A$, which applied to the initial set of facts I results in a state where all the elements of G are true. The cost of the plan Π is defined as $C(\Pi) = \sum_{a_i \in \Pi} c(a_i)$.

One of the main advantages of AP is the availability of a standard representation language, *Planning Domain Definition Language* (PDDL [8]). PDDL permits us to represent domains (objects of different types, predicates and actions) and problems (initial state and goals), providing us with planner/domain independence and nowadays also allowing us to take into account action costs, state preferences, and action durations. But specification of accurate action models for addressing AP tasks in the real world, like robotic control, is complex. Current technology does not allow us to extract all the information about the environment, so our vision of the world through sensors can not be fully informed. Even in traditionally easy-to-code planning domains, it is complex to specify the potential outcomes of actions when the environment is non-deterministic. So, most of the time the success of the AP systems fully depends on the skills of the experts that define the action model. In the real world defining these models is particularly difficult because of the action execution uncertainty, especially in the environments where autonomous robots are used. Furthermore, due to the above, generated plans can not always be successfully completed in reality.

Deterministic planning does not seem useful for control systems by itself, but it can be improved to take advantage of the benefits it provides (like domain/planner independence, long-term reasoning and explicit representation of states, goals and actions). Real world difficulties can be overcome by supervising the execution and planning process, and the planning community is currently developing systems for the acquisition, validation and maintenance of AP models to improve the knowledge representation issues (like for instance, integrating planning and learning to improve execution [9]).

For non-deterministic environments, probabilistic planning techniques can be used. And there is also a standard representation language *Probabilistic Planning Domain Definition Language* (PPDDL [10]) for this planning approach. This language allows us to include information about the probabilities of different effects of action execution, allowing a more realistic representation in some domains. For now, we focus on classical planning and we deal with the uncertainty by adding monitoring. The reason is that the use of probabilities complicates the planning process and generating the probability models is difficult.

We tested the performance of our approach using the IPC domain *Rovers*. The Rovers domain is inspired on the control of planetary rovers. A set of rovers navigate a planet surface, finding samples and communicating this information back to a lander spacecraft. The domain includes the following objects: rovers, their stores, cameras, waypoints, soil/rock sample waypoints, objectives, and a lander-waypoint where the lander spacecraft is located.

Each rover is situated at a given location, and it can carry a sample of a given waypoint or be empty. Taken samples may or may not have been communicated to the lander. Each rover will be able to:

- Take images of an objective with the calibrated camera.
- Traverse the path between two connected waypoints.
- Load soil/rock samples of a waypoint into the store.
- Transmit data for a sample or image.
- Empty the store.

To illustrate the domain definition with PDDL we are now going to describe *Rover* domain specification. In Figure 1 we can see how to define a PDDL domain. In line 1 we indicate the domain name, in 2 we specify that it is a typed domain, and in lines 3 and 4 the types of objects present at the domain are listed. In the case of this domain, the object types are: rover, waypoint, camera, mode (for the modes that the camera allows), lander (for the lander spacecrafts) and objective (points to be photographed).

```
1 (define (domain Rover)
2    (:requirements :typing)
3    (:types rover store waypoint lander
4            camera mode objective)
5 ...
```

Fig. 1. Initial part of the Rovers domain PDDL definition.

In Figure 2 the predicates are declared, so we can represent the world states. For example, predicate `at` (line 2) is used to indicate the current location of the rover, `at-lander` (line 3) indicates the lander spacecraft waypoint, the predicates from lines 5 to 7 describe the rovers instruments (`equipped-for-xxx`) and the predicates between lines 17 and 19 describe the data communication objectives (`communicated-xxx-data`).

Figure 3 shows the definition of the `take-image` action. The parameters involved of the actions are declared in lines 2 and 3. The rover `?r` takes an image of the objective `?o` from the waypoint `?p`, with the camera `?i` on mode `?m`. Between lines 4 and 9, the preconditions of the action are detailed. The rover must be equipped for imaging (line 4), the objective has to be visible from the current waypoint (line 5), the camera involved has to be calibrated (line 6), be on the rover (line

```
(:predicates
   (at ?x - rover ?y - waypoint)
   (at_lander ?x - lander ?y - waypoint)
   (can_traverse ?r - rover ?x - waypoint ?y - waypoint)
   (equipped_for_soil_analysis ?r - rover)
   (equipped_for_rock_analysis ?r - rover)
   (equipped_for_imaging ?r - rover)
   (empty ?s - store)
   (have_rock_analysis ?r - rover ?w - waypoint)
   (have_soil_analysis ?r - rover ?w - waypoint)
   (full ?s - store)
   (calibrated ?c - camera ?r - rover)
   (supports ?c - camera ?m - mode)
   (available ?r - rover)
   (visible ?w - waypoint ?p - waypoint)
   (have_image ?r - rover ?o - objective ?m - mode)
   (communicated_soil_data ?w - waypoint)
   (communicated_rock_data ?w - waypoint)
   (communicated_image_data ?o - objective ?m - mode)
   (at_soil_sample ?w - waypoint)
   (at_rock_sample ?w - waypoint)
   (visible_from ?o - objective ?w - waypoint)
   (store_of ?s - store ?r - rover)
   (calibration_target ?i - camera ?o - objective)
   (on_board ?i - camera ?r - rover)
   (channel_free ?l - lander)
)
```

Fig. 2. Rovers domain PDDL predicates.

7) and support the photography mode we want for the picture (line 8). Finally, the effects of the action are that we have the image (line 10) and that the camera is no longer calibrated (line 11).

```
(:action take_image
 :parameters (?r - rover ?p - waypoint ?o - objective
              ?i - camera ?m - mode)
 :precondition (and (equipped_for_imaging ?r)
                    (visible_from ?o ?p)
                    (calibrated ?i ?r)
                    (on_board ?i ?r)
                    (supports ?i ?m)
                    (at ?r ?p))
 :effect (and (have_image ?r ?o ?m)
              (not (calibrated ?i ?r))))
```

Fig. 3. Definition of the `take-image` action in PDDL language.

III. Architecture

In this section, the architecture that integrates planning, execution, monitoring and re-planning (ROPEM: RObot Plan Execution with Monitoring) is described. This is the first step of a long-term goal that consists on integrating other techniques for improving the autonomous robot control, refining the different modules presented in this section and adding new ones (such as several machine learning modules, that would merge learning reactive behaviours, with learning control and domain knowledge, for the deliberative planner).

In Figure 4 the system execution is described. ROPEM is initiated by loading a domain (line 3) and a problem (line 4). Right after that, we extract from the problem the information about the navigation map (line 5).

Then, we receive the low level information (line 7) and translate it (line 8) to the high level state. This process is also performed after each action is executed, but it is done at this point just to get the initial state. And we generate a plan with a planner (line 10).

```
ROPEM (problem, domain, planner)

  loadDomain(domain);
  loadProblem(problem);
  loadMap (problem);

  lowLevelState = receiveLowLevelState();
  highLevelState = lowToHigh(lowLevelState);

  plan = generatePlan(domain, problem, planner);

  while (!monitoringGoalsAchieved && executionCode != OK)

    while (!monitoringGoalsAchived && executionCode == OK)
      executionCode = executeAction (nextAction(plan));
      monitoringGoalsAchived = checkSampleGoals();
      lowLevelState = receiveLowLevelState();
      highLevelState = lowToHigh(lowLevelState);
    end-while

    if (executionCode == REPLAN)
      replanproblem = generateProblem(highLevelState);
      plan = generatePlan(domain, replanproblem, planner);
      executionCode = OK;
    end-if

    if (executionCode == ERROR)
      print("Unexpected Error. Execution Ended");
    end-if

  end-while

terminateExecution
```

Fig. 4. High level description of the execution algorithm.

Once we have the initial plan to be executed we enter on the *global execution loop* (lines 12-33), until the goals have been achieved, replanning is needed or there is a failure.

From line 14 to line 19 we have the *plan execution loop*, where we execute all the actions of the current plan. We execute each action (line 15) obtaining an execution code. After executing each action, goal achievement is monitored (line 16).

Once the algorithm exits the *plan execution loop* (lines 14-19), it analyzes the two exit conditions left (replanning and error) for the *global execution loop*. If there is an execution error (like a mapping mismatch or a device malfunction) it stops the execution (line 27). If the execution code indicates the need of replanning (line 21), then it generates a new PDDL problem from the current high level state (line 22), it executes the planner (line 23) with the same domain and the new problem and continues with the *global execution loop*.

Now, each module of the ROPEM architecture (Figure 5) is detailed.

A. Low Level Control

To achieve robotic-device independence, the actuators/sensors management is all done by the low level control layer that implements the basic control skills. This module receives low level action requests and sends the appropriate commands to the robot actuators, handling the corresponding communication with the control platform server. In other words, this module provides a set of basic skills that compose a low-level control server interface that is going to be used by the execution module (see Section III-B). This module is also in charge of obtaining the sensor readings from the robot after the execution of each actuator command.

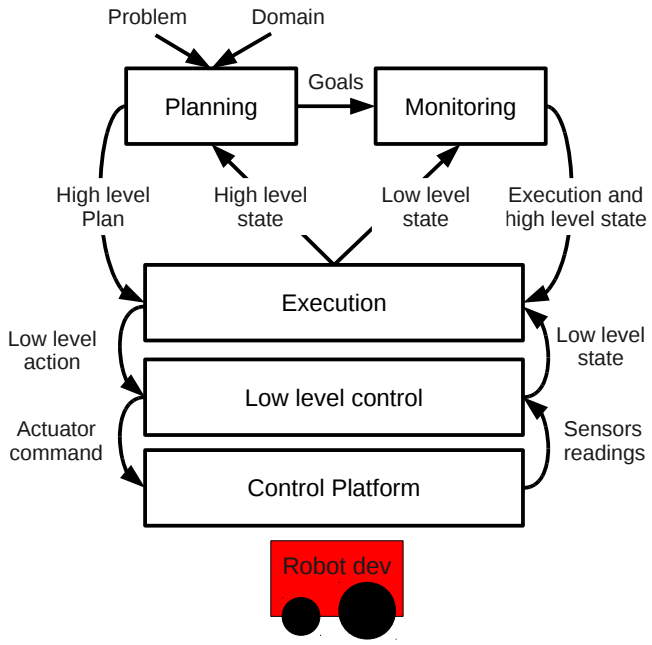


Fig. 5. ROPEM Overview.

The chosen control platform for commanding the robot is Player [11]. Player is a (TCP) network server that provides an interface for robot device (sensors/actuators) control, designed to be language and platform independent. The Player project includes simulation environments (2D, Player/Stage; and 3D, Player/Gazebo) and other useful tools such as the monitoring application *playerv*. This platform provides official support for several languages and has non-official libraries for many others, allowing ROPEM architecture to be as robot independent as possible.

Given that this module has been implemented for mobile robot bases in 2D, Player assures its reusability (with minor changes) for any planar mobile robot and has been tested with the real robot Pioneer P3DX.

B. Execution

To avoid control platform dependency, the execution is separated from the low level control. That way, our approach does not depend on control platforms or robotic devices.

The execution module receives the high level plan resulting from the execution of a deliberative planner (see Section III-D). Then, it executes one by one the high-level actions of the generated plan and receives the resulting state of the world after the execution of each action. The execution is done as follows: each high level action is decomposed into the corresponding low level actions (see Table I); then, each one of the low level actions is executed by the low level layer (see Section III-A); and once the execution of all these sub-actions is finished, the resulting low level state is translated to the corresponding high level state by the monitoring module (see Section III-C), which at the same time verifies the goal achievement.

High Level Actions	**Used Low Level Behaviours**
Navigate	`moveTowardX`, `moveTowardY`, `turnRight`, `turnLeft`
Calibrate and Sample Rock/Soil	`findBlob`, `gotoBlob`, `bumpCenter`
Communicate Data	`sendEmail`
Take Image	`saveFrame`
Drop	No low level behaviour (see section IV-C)

TABLE I
LOW LEVEL BEHAVIOURS CONFORMING EACH HIGH LEVEL ACTION.

C. Monitoring

It is difficult to predict the result of executing an action in non-deterministic planning domains such as the robot control ones, where the environment is dynamic. So, we need a module that supervises the execution and detects when it can not continue with the execution of the initial plan. Monitoring can be done at the low level or at the high level. For our work, we focus on high level supervision using a monitoring module.

This module carries out the supervision of the plan execution and the achievement of goals by receiving information from the execution module. To verify the execution state, the monitoring module receives (from the planning module, described in Section III-D) the goals of the problem to be solved. On each execution step, aside from checking whether the goals have been fulfilled, it also determines whether replanning is required.

The current re-planing policy verifies if the preconditions of the next action to be executed are satisfied. For example, in the action `navigate` (shown in Figure 6), where the rover `?x` navigates from `?y` waypoint to `?z`, it verifies the following three preconditions (`:precondition`) to determine if we need to re-plan or not: check that the rover is available (not performing another task), that it can go to the destination waypoint and that this waypoint is visible from the current one.

```
(:action navigate
 :parameters (?y - waypoint ?z - waypoint
              ?x - rover)
 :precondition (and (can_traverse ?x ?y ?z)
                    (available ?x) (at ?x ?y)
                    (visible ?y ?z))
 :effect (and (not (at ?x ?y)) (at ?x ?z)))
```

Fig. 6. PDDL `navigate` action description.

As mentioned in the previous section (Section III-B), every action (high level task) is decomposed into a set of low level behaviors. Our system supervises the sensor readings (low level information) after executing each low level action and if a failure is detected, the execution of the high level action being executed is considered erroneous. As the low level behaviours are simple and executed in a short period of time, state changes at both the low and high levels are going to be detected while executing low-level actions.

D. Planning

This module is in charge of executing the planner and generating the sequence of high level actions that are going to be executed. We use the SAYPHI planner [12]. SAYPHI is an automated planner built with the aim of integrating various machine learning techniques applied to planning. Actually SAYPHI consists of a forward search planner like Metric-FF [13], including many search algorithms. It also includes four learning sub-systems, all of them competitors in the first learning track of the International Planning Competition (IPC) [14].

Given that we are using the standard PDDL domain description from the IPC, any deliberative planner could have been used instead. The planner receives the PDDL domain and a problem specified in the same format, and returns the corresponding sequence of high level actions that have to be executed in order to achieve the goals.

The planning module is also in charge of sending the goals of the problem that is being solved to the monitoring (see section III-C) in order to supervise the execution of the proposed solution.

In case of replanning, the planning module (that already knows the domain) will receive the high level state from the execution module (see section III-B). Then the entire process will begin again using this state as the new initial state.

IV. Experiments

A first set of experiments (section IV-E) was conducted, focusing on the planning and replanning capabilities of our approach. The objective was to observe how our approach behaved with two different sources of controlled uncertainty: speed and obstacles. We executed two different problems in two maps, with different speed configurations and in presence and absence of obstacles. We wanted to test the planning/replanning capabilities and the global behaviour of the ROPEM architecture with these specific configurations. Each experiment was executed five times and the presented results are the average values obtained in all executions.

In the second set of experiments (section IV-F), a third source of simulated uncertainty was introduced to illustrate one of the issues that can be improved in the planning process. We included a temporal component in order to study the action durations, showing the importance of generating an accurate task model in AP. In particular, we added duration to the `navigate` action of the rovers domain and simulated different types of terrain affecting the real execution, illustrating the gap that can be found between the observed real world and our representation of it. Each experiment was executed once.

A. Experimental Setup

A Pioneer P3DX, equipped with sonar, bumpers and a motor-base, was used along with a Logitech Sphere cam (with PTZ capabilities). The control software was running in a PC connected via USB to the camera (usb-usb) and the robot (serial-usb).

The implemented low-level control module (see section III-A) provides an interface that allows controlling the following sensors: sonar, motor base, camera (PTZ and blob detection) and bumpers. For localization we used the odometry information provided by the motor base of the robot (x, y and yaw). This information was also used to locate the robot on the waypoint map (Section III-C).

The grid to which the waypoints were mapped was drawn on the floor, only for external monitoring during the experiments. The robot did not use this drawn grid to orient itself. Figure 7 shows the robot in the test environment.

Fig. 7. P3DX Robot (Our Rover), on the test environment (our Mars).

B. Rovers Domain Implementation

For the experiments we implemented the Rovers domain where, as stated in Section II, the objectives are to communicate a set of sample/image data to the lander, and problems involve task and path planning. For example, the PDDL formalization of a problem is detailed on Figure 8.

In line 1, the `Rover` domain is specified. Between lines 2 and 8, the objects are listed. In this problem, we use the rover (`p3dx`) and its store (`p3dxstore`), the camera (`logitechsph`), two waypoints (`wp0` and `wp1`) and one objective (`obj1`). From line 10 to 25, the initial state is described. And in lines 26 and 27 the goals (communicating an image of the only objective and the soil data of the only soil sample) are specified.

C. Domain Mapping

In order to map high-level actions and states into sensing/acting data and robot low-level behaviours, we use the following representation mapping.

Navigate (Figure 9) is mapped to turn and move behaviors, orienting the robot in the direction of the destination waypoint and moving it forward. First, we make the high level verifications (position, connection and visibility between waypoints). Then we determine the final orientation of the robot with respect to the destination waypoint and turn it into that direction, and move forward.

```
(define (problem wafexample) (:domain Rover)
  (:objects general - Lander
            colour high_res low_res - Mode
            p3dx - Rover
            p3dxstore - Store
            wp0 wp1 - Waypoint
            logitechsph - Camera
            obj1 - Objective)

  (:init (channel_free general)
         (available p3dx)
         (at p3dx wp0)
         (store_of p3dxstore p3dx)
         (empty p3dxstore)
         (equipped_for_imaging p3dx)
         (on_board logitechsph p3dx)
         (supports logitechsph high_res)
         (calibration_target logitechsph obj0)
         (visible wp0 wp1)
         (visible wp1 wp0)
         (can_traverse p3dx wp0 wp1)
         (can_traverse p3dx wp1 wp0)
         (at_lander general wp1)
         (visible_from obj1 wp1)
        (at_soil_sample wp0))
  (:goal (and (communicated_image_data obj1 high_res)
              (communicated_soil_data wp0))))
```

Fig. 8. PDDL Rovers problem description.

```
navigate (origin_wp, destination_wp)
     isAt(origin_wp)
     canTraverse(origin_wp, destination_wp)
     is_visible(origin_wp, destination_wp)
     orientateTO(destination_wp)
     moveTowards(destination_wp)
```

Fig. 9. Navigate action mapping.

Calibration and rock/soil sampling are represented by joined blob tracking and bumping actions. For instance, taking a rock sample (Figure 10) is represented as: locating a blue blob, reaching it and finally bumping that zone with the center front bumper of the robot. Specifically, blob-tracking is implemented as a simple blob-loop, with the camera panning and a blob detection proxy (provided by Player). Before executing any physical corresponding actions, the high level verifications (position and store emptiness) are performed. We used that representation for those actions, because we had no actuator (like a gripper or a robotic arm) for performing the actual actions.

```
sample_rock (sample_wp)
    isAt(sample_wp)
    isStoreEmpty()
    findblPanning(blue)
    aproachbl(blueBloob)
    sampledRock(sample_wp)
    fullStore()
```

Fig. 10. Sample-rock action mapping.

Taking an image is represented by capturing a real image with the camera at the moment of executing this action, and communication of (soil, image and rock) data is represented as sending an email. The rest of high level actions are not represented as any real physical action, because all the actuators are already in use. For example, the drop action was not included, because our robot does not have an arm that can take real samples, so there is nothing to drop.

Also, in order to use the Rovers domain, the waypoints are mapped to a grid. The mapping is done as follows: waypoints are defined as an (x, y) pair; the distance between waypoints, on the grid, is d; to differentiate between waypoints, a bounding box of d^2 is established for them; and to simplify, the bounding boxes are adjacent to each other.

More details on the high and low level states mapping is provided on the next section (Section IV-D).

D. Low/High Level States

The low level state consists of the following sensor readings: odometry information, x, y and yaw real values; bumper information, one binary value for each bumper, $b_1, \ldots, b_5$; the readings of the eight sonars, $s_1, \ldots, s_n \in \mathbb{R}$; and the information of the largest blob (x-y coordinates, top, bottom, area and color). The high level state is defined by the domain predicates: `at`, `calibrated`, `have-rock-analysis`, `have-soil-analysis`, `have-image`, etc.

The monitoring module translates the low level state to the high level state for the execution module. Robot sensor readings are translated to domain predicates after the execution of each high level action, as defined in Table II.

In the first column of Table II we mention the used low level values, in the second one we list the affected predicates and in the last one we summarize the mapping function.

The odometry information (x, y and yaw) of the low level is used to determine the current waypoint (`at` predicate) and orientation. Each waypoint has a cell on a high level grid (see Section IV-C). The current high level position is computed by checking in which cell the x, y pair is located.

To represent that a sample has been picked up and loaded in the robot store, we check that the blob of the corresponding color has been found and successfully reached (maximum size and sonar distance) and that the front center bumper has been bumped. In this case, we make true the `full` predicate for that store and the corresponding `have-xxx-analysis`. Notice that when a high level task is executed, the domain action preconditions are also checked (not mentioned on Table II).

E. Performance

We tested the ROPEM architecture performance with two different problems on the same map. In Figure 11 an example of the rock/soil samples placement and the waypoints from which the objectives are visible is shown. On that graphic representation, the gray waypoints are the external area, which is represented in order to consider the case of the rover going out of the map. Thus, rovers can be *in* that area (in case of a failure execution) but not go *into* them intentionally. The shown colors (yellow, blue and red) are the ones that were assigned to the blob-tracking behavior.

The first problem used for the experiments consists of nine goals, involving twenty waypoints (6 describing the map and 14 the exterior) and one rover (the P3DX robot). And the

Low Level	High Level	Mapping
Robot `r` in position x, y	`(at ?x - rover ?y - waypoint)`	**if** x,y in cell `w` **then** `(at r w)`
Blob, sonar and bumper data of robot `r` (with store `s`)	`(have-rock-analysis ?r - rover ?w - waypoint)` and `(full ?s - store)`. (Sample location: waypoint `w`)	**if** max blue blob reached and center bumper bumped **then** `(have-rock-analysis r w)` and `(full s)`
Blob, sonar and bumper data of robot `r` (with store `s`)	`(have-soil-analysis ?r - rover ?w - waypoint)` and `(full ?s - store)`. (Sample location: waypoint `w`)	**if** max yellow blob reached and center bumper bumped **then** `(have-soil-analysis r w)` and `(full s)`
Blob data of robot `r` (with camera `c`)	`(calibrated ?c - camera ?r - rover)`	**if** max red blob found **then** `(calibrated r c)`

TABLE II
LOW TO HIGH LEVEL STATES TRANSLATION.

second problem consists of communicating one sample of rock data, one sample of soil data and one image data.

Two kinds of tests were conducted: the *slow* experiments, that were done with a forward-speed of 0.2m/s and a turn speed of 0.2rads/s; and the *fast* experiments, that were performed with a forward-speed of 0.4m/s and a turn speed of 0.35rads/s. In the case of the experiments with obstacles, a second remotely-controlled robot was crossed in the middle of the rover trajectory five times, during the execution of different navigate actions. The moving obstacle momentarily crossed the path of the rover to simulate the case of another rover exploring a close zone, in order to test the replanning capabilities in that specific case.

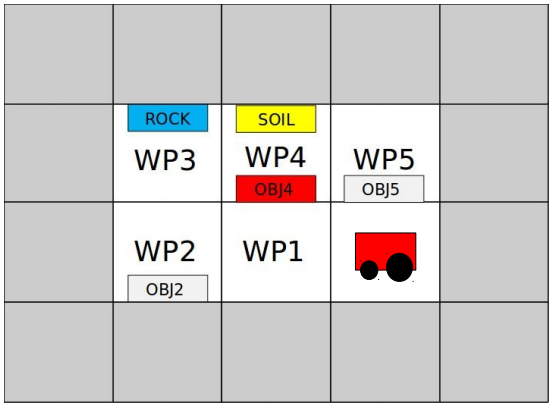


Fig. 11. Graphical representation of a problem.

1) Performance Results: Tables III and IV summarize the experimental results for the first problem. The following metrics (measure column) were established: number of initial actions, total executed actions; number of replanning steps performed; total execution time, in minutes; and total planning and replanning time, in seconds.

Without obstacles (see Table III), in both speed configurations, the total high-level executed actions is the same as in the original plan. Given that higher speeds introduce more failures in execution, in the fast executions, replanning was needed, while in the slow configuration it was not. Despite the fact that replanning was necessary because of the noise accumulation, caused by the turn and forward speedup, the execution time was still reduced.

Absence of Obstacles		
Measure	Slow	Fast
Initial actions	59±0	
Total actions	59±0	
Replanning	0±0	3±0.4
Total execution time	17±0.7m	14.7±0.5m
Planning/Replanning time	$<$ 0.5s	

TABLE III
SUMMARY OF EXPERIMENTS RESULTS FOR THE FIRST PROBLEM: ABSENCE OF OBSTACLES, WITH DIFFERENT SPEEDS.

With obstacles (see Table IV), in both speed configurations, the total high-level executed actions (71 in the slow configuration and 74 on the fast one) is significantly increased with respect to the number of initial planned actions (59). The number of replanning steps needed were the same on both configurations. Thus, the speedup did not affect significantly the total execution time. But, in the case of obstacles, we did not improve with the increase of speed.

Presence of Obstacles		
Measure	Slow	Fast
Initial actions	59±0	
Total actions	71±6.5	74±5.5
Replanning	20±9	20±5.5
Total execution time	15±0.3m	16.3±1.8m
Planning/Replanning time	$<$ 2s	

TABLE IV
SUMMARY OF EXPERIMENTS RESULTS FOR THE FIRST PROBLEM: PRESENCE OF OBSTACLES, WITH DIFFERENT SPEEDS.

In this problem, with our proposed configuration, the obstacles represented a stronger source of uncertainty, while the noise introduced by the speedup caused a smaller number of replanning episodes.

The second experiment to test the ROPEM architecture, unlike the first one, was executed only without obstacles because

the objective was to observe the behavior of the ROPEM architecture in smaller problems. Table V summarizes the experiments results for the executions of the second problem. The same metrics were used.

Speed Test		
Measure	Slow	Fast
Initial actions	26±0	
Total actions	26±0	22±2
Replanning	0±0	5±10
Total execution time	7.8±0.2m	4.8±1.5m
Planning/Replanning time	< 0.4s	

TABLE V
SUMMARY OF EXPERIMENTS RESULTS FOR THE SECOND PROBLEM. SIMPLE PROBLEM WITH DIFFERENT SPEEDS.

The solution of this problem consisted of 26 actions (initial plan), which is less than half of the previous problem solution. The difference between the total executed actions (second row) is because in some of the *fast* executions, the accumulation of noise in the odometry made the robot accidentally go into waypoints that shortened the path (i.e. moving forward two waypoints at a time). As it is a small problem, only 5 replanning steps were performed on the *fast* configuration; and no replanning was done on the *slow* one. In this problem the total execution time was reduced to half with the *fast* configuration; so the increase of speed was totally worth, taking into account the insignificant amount of time consumed by the replanning steps that the speedup noise caused.

In both problems the resulting replanning time was insignificant with respect to the tasks execution time, so there would be no need to include plan adaptation strategies at this point.

F. Navigation Time

Once the ROPEM architecture performance was tested, we decided to analyze the action durations. We improved the model by introducing a temporal component in the domain and analyzed how the task execution uncertainty and the sensor-noise caused by the environment can affect the action duration.

In order to introduce durations into the domain, for this part of the experimentation, the classical specification of the *Rovers* domain (used in the previous experiments) was modified. In Figure 12 we show the changes, with respect to the original domain, highlighted in red.

Two types of terrains (`sandy` and `rocky`) were added on lines 5 and 6. Also, a temporal component was incorporated to keep track of consumed navigation time (`navigationtime`, line 33). The time that our robot takes to traverse from one waypoint to another (with the slow speed configuration), in reality, is around 6 seconds, so the `navigationtime` fluent is increased 6 seconds every time a navigation action is executed (line 40). To our robot, that time was almost constant because we always worked in the same environment. The terrain of the hallway of our laboratory (where we performed all the experiments) does not present any irregularity. But that is not realistic. A rover navigating the surface of Mars will

```
1  (define (domain Rover)
2  (:requirements :typing)
3  (:types rover waypoint store camera mode lander objective)
4  (:predicates
5        (sandy ?x - waypoint)
6        (rocky ?x - waypoint)
7        (at ?x - rover ?y - waypoint)
8        (at_lander ?x - lander ?y - waypoint)
9        (can_traverse ?r - rover ?x - waypoint ?y - waypoint)
10       (equipped_for_soil_analysis ?r - rover)
11       (equipped_for_rock_analysis ?r - rover)
12       (equipped_for_imaging ?r - rover)
13       (empty ?s - store)
14       (have_rock_analysis ?r - rover ?w - waypoint)
15       (have_soil_analysis ?r - rover ?w - waypoint)
16       (full ?s - store)
17       (calibrated ?c - camera ?r - rover)
18       (supports ?c - camera ?m - mode)
19       (available ?r - rover)
20       (visible ?w - waypoint ?p - waypoint)
21       (have_image ?r - rover ?o - objective ?m - mode)
22       (communicated_soil_data ?w - waypoint)
23       (communicated_rock_data ?w - waypoint)
24       (communicated_image_data ?o - objective ?m - mode)
25       (at_soil_sample ?w - waypoint)
26       (at_rock_sample ?w - waypoint)
27       (visible_from ?o - objective ?w - waypoint)
28       (store_of ?s - store ?r - rover)
29       (calibration_target ?i - camera ?o - objective)
30       (on_board ?i - camera ?r - rover)
31       (channel_free ?l - lander)
32 )
33 (:functions (navigationtime))
34
35 (:action navigate
36 :parameters (?x - rover ?y - waypoint ?z - waypoint)
37 :precondition (and (can_traverse ?x ?y ?z) (available ?x)
38                   (visible ?y ?z) (at ?x ?y))
39 :effect (and (not (at ?x ?y)) (at ?x ?z)
40             (increase (navigationtime) 6)))
41 ...
```

Fig. 12. Rovers domain with navigation time.

traverse different types of terrains and that was one of the motivations for this second set of experiments.

Based on this modified Rovers domain with navigation time, we tested the effect of the terrain types on the navigation with a simulated navigation delay. To sum up the work, it was assumed that:

- a waypoint can only be of one type of terrain,
- the delay induced by each type of terrain is constant,
- the odometry is not affected,
- and the orientation tasks do not introduce any navigation delay.

The goal was to study the effect of another controlled source of uncertainty, as was the terrain types.

As stated before, the time spent by the real robot in navigating from one waypoint to another, ignoring the previous orientation process, is of around 6 seconds. We decided to simulate how the terrain types affected this time instead of working with different terrains in reality in order to avoid introducing other sources of uncertainty. For simulating the navigation delays, `sleep` commands were used during the execution of the high level task `navigate`. Specifically, the introduced delays were the following:

- 0 seconds, for the case where the origin and destination waypoints are both sandy;
- 4 seconds, when the origin or the destination waypoint are of type rocky;

- and 8 seconds when both the origin and the destination are rocky.

The resulting total navigation times to navigate from a given waypoint to a consecutive one, are as follows: 6s for the case of sandy-sandy, 10s for rocky-sandy and sandy-rocky, and 14s for the rocky-rocky combination.

1) Navigation Evaluation: In order to study the effect of simulated terrain types, we conducted a new set of experiments in bigger maps and analyzed the navigation time of the robot.

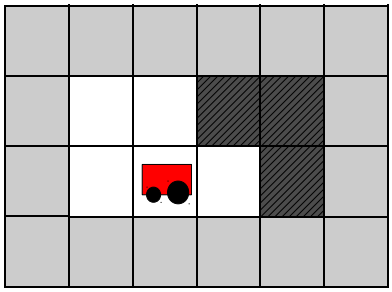


Fig. 13. Map used on the first problem for navigation experiments.

For the first experiment of this set we executed a problem on a 24 waypoints grid, where 8 waypoints represented the map where the rover is supposed to navigate and the rest (16 waypoints) represented the exterior of that map. In Figure 13 we can see the graphical representation of this map. The gray cells represent the exterior, the black cells represent the rocky waypoints and the white ones represent the sandy waypoints.

For the second and third tests we used two different problems in a new map. This time the execution was performed on a 36 waypoints map (Figure 14), with 22 waypoints representing the exterior and 14 representing the navigation map. Again, black cells are the rocky waypoints and white cells are the sandy ones.

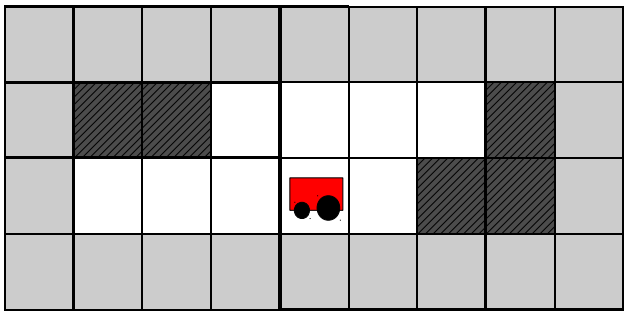


Fig. 14. Map used for the problems two and three during the navigation experiments.

We used the slow speed configuration for all the executions, and we executed one time each problem. In Table VI results are summarized; we show the initial planned actions versus the executed ones, the number of executed navigate actions and the navigation/execution time.

Navigation			
Measure	Ex1	Ex2	Ex3
Executed/Planed Actions	59/59	75/72	92/114
Navigate Actions	31	56	66
Navigate Time (Domain)	3.1m	5.6m	6.5m
Real Navigation Time	7.6m	13.9m	16.4m
Total Execution Time	8m	15m	26m
Replanning Episodes	0	2	5
Planning/Replanning Time	< 1s		

TABLE VI
RESULTS OF THE NAVIGATION EXPERIMENTS.

Execution of the first two problems (*Ex1* and *Ex2* on the Table VI) was completed, but the plan of the last problem was not successfully completed due to the accumulation of error in the odometry. With this last experiment (*Ex3* on the table) we noticed that the odometry stops being a reliable source of positioning information, to locate the robot on the map, during long runs, as expected. Thus, this first implementation approach is not scalable to big problems (as is). Although the monitoring and planning approach are useful to overcome other types of challenges, the localization is a problem that has to be solved apart.

Multiplying the `navigate` executed actions by the theoretical navigation time (6s), defined in the modified Rovers domain, we obtain the *Navigate Time (Domain)* row. And we can see that this time is far from the real navigation time in all problems (see the *Real Navigation Time* row on Table VI). This is due to the delays we introduced for terrain types. This information is not represented on the domain and this shows how the planning approach needs some help. The environment modification is not being taken into account on the planning process, so we need to include some technique that allows us to extract this knowledge during execution as, for instance, machine learning techniques that acquire domain models from execution.

V. RELATED WORK

One of the first applications of Automated Planning (AP) in robotics was for the generation and monitored execution of robot plans [15] using the classic planner STRIPS [16].

Reactive approaches have also been used for robot control, like architectures based on Reactive Actions Packages (RAPs) [17], [18]. RAP-based control systems, like other control systems, commonly suggest three levels of abstraction (execution, planning and hardware), similar to our structure. The main difference with our work is that, for these approaches, a plan is a set of RAP pre-defined tasks, so they are based on robot device-specific skills. In our approach we pull apart the low (hardware) control from the high level trying to gain platform independence. Reactive approaches usually focus on the combination of reactive behaviours, while at the moment, we focus on high level planning, using the PDDL standard combined with low level skills.

Another example of planning techniques applied to robotics, is the use of hierarchical planners [19] for robot control.

Architectures that combine hierarchical task planning with path planning, have been proposed [20]. Also hierarchical approaches have been implemented extending behavior-based architectures for robot control [21]. The disadvantage of using this type of planners is that a custom hierarchical task networks (HTN) have to be defined for each domain, complicating the domain definition.

AP techniques have also been applied to real planetary exploration, as with the Rovers that are currently on Mars [22]. This problem is particularly interesting, because it is a case in which having a planning system that provides autonomy is essential. In this case, AP techniques are not used on board the rover, but on ground. Also, they used a different planning paradigm, timeline-based planning, which lies closer to scheduling. A two-layer architecture has been used in real Rovers [4], focusing on interoperability of robot-control software, integrating a decision layer and taking into account high-level autonomy, as in our work.

T-REX [23] is an on-board system for planning and execution applied to the control of autonomous underwater vehicles in real oceanographic scientific missions. T-REX uses an specific language (NDDL) to describe the domain, and it is based on the notion of partition; planning is done at different abstraction levels by different hierarchical modules, each of one embedding a temporal constraints satisfaction-based planner. Instead, we propose to use a standard planner that reads a PDDL (Planning Domain Definition Language) domain description and is able to control the robot on-board. At the moment we are not focusing on domains where a temporal component is essential, so we can not benefit from T-REX advantages.

There have been recent developments in plan-based control of autonomous robots [24], where different approaches used plan-based high-level and Structured Reactive Controllers (SRCs), among others. Again, these approaches are closer to the hardware and therefore are tight to the robotic devices.

Control systems combining opportunistic planning and reactive techniques in the low level behaviors have been developed [25]. There have been contributions where high level actions have been defined as behaviors themselves, close to the current standard way of defining high level actions in PDDL. The advantage of our approach again is that we can benefit from any PDDL planner.

Autonomy systems for rovers control have been proposed using probabilistic planning technology [26]. Partially Observable Markov Decision Process (POMDP) approaches have been used for plan generation, taking into account action and sensing uncertainty. For our contribution we decided to focus on classical planning techniques and deal with uncertainty by using monitoring and replanning.

We are currently improving the architecture that supports our contribution in order to reach a system that fully supports planning, execution and monitoring, at low and high level, as more generic approaches that have been recently proposed [27].

VI. Conclusions

In this paper we have presented an approach for autonomous mobile robot control, that integrates automated planning (AP) techniques, execution and monitoring. We conducted a set of experiments to test the performance of our approach on a Pioneer P3DX, using the Rovers domain. We also evaluated the effect of the environment on actions durations.

Regarding the ROPEM architecture, as the chosen robotic-control platform (Player) provides support for different programming languages, our approach is language independent. Also, Player allows us to make the control code independent of the planar mobile robot bases in 2D (with minor changes). As we separated the execution from the control, we can reuse the low and high level skills.

By applying AP we benefit from the standard language PDDL that permits us to represent domains and problems, providing us with planner/domain independence taking into account action costs, state preferences, and action durations.

We are currently working on the integration of a navigation time learning module for automatic generation of navigate duration models, improving the planning process by taking into account terrain types. And including different learning modules to improve the planning process is also part of our future work.

Our medium-term goals include improving some aspects of the architecture, such as upgrading the localization system. This will allow us to test the architecture scalability. Moreover, we are going to study different sources of uncertainty to see how they affect our approach.

As part of our future work, we will try to learn probabilistic models from execution in order to apply probabilistic planning approaches.

Acknowledgments

This work has been partially supported by the Spanish MICINN under projects TIN2008-06701-C03-03, TRA-2009-008 and Comunidad de Madrid - UC3M (CCG10-UC3M/TIC-5597).

References

[1] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Amsterdam: Morgan Kaufmann, 2004.

[2] J. E. Flórez, Álvaro Torralba, J. García, C. L. López, Ángel García-Olaya, and D. Borrajo, "Timiplan: An application to solve multimodal transportation problems," in *Proceedings of "Scheduling and Planning Applications woRKshop" (SPARK). Workshop of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS'10)*, Toronto, Ontario (Canada), 2010.

[3] M. D. Rodríguez-Moreno, D. Borrajo, and D. Meziat, "An ai planning-based tool for scheduling satellite nominal operations," *AI Magazine*, vol. 25, no. 4, pp. 9–27, 2004.

[4] I. A. D. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and W. S. Kim, "Claraty: An architecture for reusable robotic software," in *in SPIE Aerosense Conference*, 2003.

[5] R. R. Penner and E. S. Steinmetz, "Automated support for human mixed initiative decision and control," in *42nd IEEE Conf. on Decision and Control*, 2003, pp. 3549–3554.

[6] M. de la Asunción, L. A. Castillo, J. Fernández-Olivares, Óscar García-Pérez, A. González, and F. Palao, "Siadex: An interactive knowledge-based planner for decision support in forest fire fighting." *AI Commun.*, vol. 18, no. 4, pp. 257–268, 2005.

[7] D. Wilkins and R. V. Desimone, "Applying an ai planner to military operations planning," in *Intelligent Scheduling*. Morgan Kaufmann, 1992, pp. 685–709.

[8] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, pp. 61–124, 2003.

[9] S. Jiménez, F. Fernández, and D. Borrajo, "The pela architecture: Integrating planning and learning to improve execution." in *AAAI*, D. Fox and C. P. Gomes, Eds. AAAI Press, 2008, pp. 1294–1299.

[10] H. L. S. Younes and M. L. Littman, "Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects," *Technical Report CMU-CS-04-167*, 2004.

[11] B. Gerkey, R. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *11th International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal, 2003.

[12] T. De la Rosa, A. García-Olaya, and D. Borrajo, "Using cases utility for heuristic planning improvement," in *Proceedings of the 7th International Conference on Case-Based Reasoning*. Belfast, Northern Ireland, UK: Springer Verlag, August 2007, pp. 137–148.

[13] J. Hoffmann, "The metric-ff planning system: Translating "ignoring delete lists" to numerical state variables," *Journal of Artificial Intelligence Research. Special Issue on the 3er International Planning Competition*, vol. 20, 2003.

[14] T. de la Rosa, R. García-Durán, S. Jiménez, F. Fernández, A. García-Olaya, and D. Borrajo, "Three relational learning approaches for lookahead heuristic search," in *Proceedings of the Workshop on Planning and Learning of ICAPS09*, Thessaloniki (Greece), September 2009.

[15] R. Fikes, "Monitored execution of robot plans produced by strips," in *Processing 71 IFIP Congress 1971*. Stanford Res. Inst., Menlo Park, CA, USA: IFIP, 1972, pp. 189–94.

[16] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[17] R. J. Firby, "Adaptive execution in complex dynamic worlds," Ph.D. dissertation, Yale University, New Haven, CT, USA, 1989.

[18] R. P. Bonasso, D. Kortenkamp, D. P. Miller, and M. Slack, "Experiences with an architecture for intelligent, reactive agents," *Journal of Experimental and Theoretical AI*, vol. 9, pp. 237–256, 1995.

[19] B. Morisset and M. Ghallab, "Learning how to combine sensory-motor modalities for a robust behavior," in *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents,*. London, UK: Springer-Verlag, 2002, pp. 157–178.

[20] J. Guitton, J.-L. Farges, and R. Chatila, "A planning architecture for mobile robotics," *AIP Conference Proceedings*, vol. 1019, no. 1, pp. 162–167, 2008.

[21] M. N. Nicolescu and M. J. Mataric, "A hierarchical architecture for behavior-based robots," in *In Proc., First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2002, pp. 227–233.

[22] J. L. Bresina, A. K. Jónsson, P. H. Morris, and K. Rajan, "Mixed-initiative activity planning for mars rovers," in *IJCAI*, 2005, pp. 1709–1710.

[23] C. McGann, F. Py, K. Rajan, J. Ryan, and R. Henthorn, "Adaptive control for autonomous underwater vehicles," in *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*. AAAI Press, 2008, pp. 1319–1324.

[24] M. Beetz, "Towards comprehensive computational models for plan-based control of autonomous robots." in *Mechanizing Mathematical Reasoning*, ser. Lecture Notes in Computer Science, D. Hutter and W. Stephan, Eds., vol. 2605. Springer, 2005, pp. 514–527.

[25] V. Matellán and D. Borrajo, "Abc2 an agenda based multi-agent model for robots control and cooperation," *J. Intell. Robotics Syst.*, vol. 32, pp. 93–114, September 2001.

[26] T. Smith, "Rover science autonomy: probabilistic planning for science-aware exploration doctoral consortium thesis summary," in *Proceedings of the 20th national conference on Artificial intelligence - Volume 4*. AAAI Press, 2005, pp. 1660–1661.

[27] V. Alcázar, C. Guzmán, G. Milla, D. Prior, D. Borrajo, L. Castillo, and E. Onaindía, "PELEA: Planning, learning and execution architecture," in *Proceedings of the 28th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'10)*, Brescia (Italia), December 2010.

Efficient Plane Detection in Multilevel Surface Maps

V. Prieto-Marañón, J. Cabrera-Gámez, A. C. Domínguez-Brito,
D. Hernández-Sosa, J. Isern-González, E. Fernández-Perdomo

***Abstract*—An automatic system aimed at producing a compact tridimensional description of indoor environments using a mobile 3D laser scanner is described in this paper. The resulting description is made up of a Multi-Level Map (ML map) and a series of planar patches extracted from the map. We propose a novel plane detection algorithm, based on the efficient RANSAC algorithm, that operates directly over the data structures of an ML map and does not need to rely on the low level laser data cloud. The mobile 3D scanner is built from a Hokuyo laser range sensor attached to a 2DOF pan-tilt, which is installed on top of a 3DX Pioneer mobile robot. The 3D spatial information acquired by the laser sensor from different poses is used to build a large single map of the environment using the SLAM 6D library. Experimental results demonstrate that the system described is capable of efficiently building compact and accurate 3D representations of complex large indoor environments at multiple semantic levels.**

***Index Terms*—3D Maps, plane detection, multilevel surface maps, laser scanner, SLAM6D**

I. Introduction

EFFICIENT use of robots in a tridimensional environment, be it indoor or outdoor, requires identification of structures and objects present in the world. These objects and structures can often be described in terms of simpler forms or primitives at different semantic levels. For example, indoor primitives based on planes can be used to characterize most of the elements conforming the environment. Their descriptions and relative locations can be used to define the internal representation or map that is to be used by the robots acting in the environment.

Maps are built from information acquired from one or more sensors. Numerous different sensors can be used to capture information in a 3D scenario from cameras (monocular, stereo or time-of-flight) or range sensors (sonars and lasers). Maps may be topological or metric, but metric maps are the preferred option when geometrical features from the environment, such as distances, volumes or surfaces, are needed.

The problem of building maps in large and unknown environments while the system orients itself, widely known as the SLAM problem, has been intensively studied by the robotics community over the last ten years [1]. Basically, the problem lies in incrementally adding new information to a map whilst estimating the relative displacements between observations and recognizing areas that have already been explored and are present in the map. This problem in two dimensions has been largely studied and —in general terms— is nowadays considered solved. The latest achievements in SLAM, together with the availability of faster sensors and processors, have fostered an interest for extending the SLAM problem to 3D scenarios with 6DoF observers, a context formerly termed as unfeasible due to its high computational demands.

In the present work, we describe an approach that allows us to construct 3D maps for large indoor scenarios using a mobile robotic system equipped with a laser sensor mounted on a pan-tilt unit. The resulting tridimensional representation of the environment comprises two levels of description. At the lowest level, a Multi-Level map (ML map) [2] is built integrating 3D scans acquired from different poses. To offset odometry errors, we address the inherent SLAM problem using the SLAM 6D software developed by Nüchter et al. [3]. From the ML map the system can detect planar patches using an algorithm that is an optimized adaptation for plane detection of the efficient RANSAC (eRANSAC) method [4]. These planar patches will be used to define a second level of description in the 3D map of the environment from which structures of higher semantic level such as walls, doors, tables, etc, could be detected.

This paper is set out as follows: after discussing related works in section II, the data acquisition system will be described in section III. Section IV shows how the ML maps are built. The plane detection algorithm in ML maps will be explained in section V. Finally, several experimental results and conclusions are discused in sections VI and VII respectively.

II. Related work

Different approaches have been adopted to allow autonomous mobile robots to build tridimensional maps of the environment. Several methods use a tridimensional grid that splits space into portions called voxels, whose values reflect the occupancy of the corresponding space volume [5]. Other authors have proposed using elevation maps, due to their much lower memory requirements. In elevation maps, the environment is represented by using a two dimensional grid where each cell represents the elevation, i.e. terrain height, at the corresponding point. These maps enable us to model large

This work has been partially supported by the Research Project *TIN2008-06068* funded by the Ministerio de Ciencia e Innovación, Gobierno de España, Spain.

V. Prieto-Marañón, J. Cabrera-Gámez, A. C. Domínguez-Brito, D. Hernández-Sosa, J. Isern-González and E. Fernández-Perdomo are with the Instituto Universitario SIANI, Universidad de Las Palmas de Gran Canaria, Spain.

J. Cabrera-Gámez, A. C. Domínguez-Brito and D. Hernández-Sosa are with the Instituto Universitario SIANI, and the Departamento de Informática y Sistemas, Universidad de Las Palmas de Gran Canaria, Spain.

Corresponding author: vprieto@ono.com

Fig. 1. An example of scenario with several surfaces crossing at different heights.

environments as shown in [6]. However, elevation maps are ill-suited to modeling scenarios containing structures crossing at different heights over the vertical of a point (see figure 1). Two examples are a table indoors or a bridge outdoors. In order to avoid this limitation, Triebel et al. [2] propose multilevel surface maps as an extension to elevation maps. These multilevel maps include, at every cell of a bidimensional grid, a list of the traversable surfaces that exist in the corresponding vertical. An improvement of the multilevel surface maps can be found in [7] where they are formally described using a probabilistic approach.

Detecting shapes in tridimensional data sets has been studied from different points of view. Starting from a 3D data point cloud, in [8], a 2 $\frac{1}{2}$ dimensional structure was built based on an incremental triangulation algorithm. Similarly, in [9], the authors developed a plane detection method using a more accurate range noise model for 3D sensors to derive from scratch the expressions for the optimum plane which best fits a point-cloud and for the combined covariance matrix of the plane's parameters. The parameters in question are the plane's normal and its distance from the origin. In other works, plane detection is addressed by using the information extracted from imaging sensors. A range imaging sensor is used in [10], with the aim of segmenting images of indoor environments in terms of horizontal and vertical planes by means of the Normalized-Cuts algorithm. An approach by Hähnel, Burgard and Thrun is presented in [11]. This work describes an algorithm for full 3D shape reconstruction of indoor and outdoor environments with mobile robots by approximating environments using flat surfaces. Other authors [12] present a method for obtaining the location, size and shape of main surfaces in an environment from points measured by a laser scanner onboard a mobile robot. The most likely orientation of the surface normal is first calculated at each point, from points in an adaptive-radius neighboring region. In other cases, stereo cameras are used, for example in [13], where an architecture for detection and estimation of planar surfaces in the scene from calibrated stereo images is presented.

III. Data acquisition

In this work, the data acquisition system is formed by a laser sensor coupled with a pan-tilt, both installed onboard a mobile robot. The laser sensor is a Hokuyo UTM-30LX with a scan width of $270°$ and $30\mathrm{m}$ detection range. The pan-tilt unit is a PTU 46-17.5 from Directed Perception. It has two degrees of freedom and it is used for scanning space in three dimensions. A Pioneer P3-DX has been used as a mobile platform and as the odometry data source.

The data acquisition system works in a move-and-stop way. The robot is moved to a new pose and then a 3D scan is taken. The pan-tilt is oriented with a pan angle α and, then, while the pan-tilt sweeps between the tilt start angle γ_s and the tilt end angle γ_e, the laser sensor takes range measurements from the environment. The laser sensor returns one scan every 25 msecs. The tilt angular speed is adjusted to obtain an angular separation between consecutive scans of ρ degrees at the maximal speed allowed by the hardware.

To integrate new laser measurements into the map we need to know the laser sensor orientation at all moment. The hardware used does not have a hardware synchronization system, so we have developed a software synchronization mechanism that allows us to acquire new laser measurements while the pan-tilt is moving between γ_s and γ_e. This synchronization system avoids having to stop the pan-tilt every time the laser initiates the acquisition of a new scan. The synchronization algorithm takes into account the pan-tilt's initial position when the laser scan starts and calculates the vertical elevation angle for every measurement returned by the sensor. The scan data timestamp t_s corresponds to the moment at which the laser sensor starts acquiring a new scan. The resolution of the Hokuyo UTM-30LX sensor is 1440 steps per revolution. The timestamp of range measurement m_p corresponding to step p is:

$$t_p = t_s + \frac{p}{1440f} \tag{1}$$

In this equation, f represents the laser beam rotation frequency. If t_0 is the instant when the pan-tilt began its tilt movement, then the time difference or delay l_p till the measurement m_p was taken is:

$$l_p = t_p - t_0 \tag{2}$$

The pan-tilt's tilt speed is adjusted so that tilt angle changes in ρ degrees while the laser beam completes a revolution. Thus:

$$\upsilon = \rho \cdot f \tag{3}$$

Thus, if l_p is known, then, by using the pan-tilt's trapezoidal acceleration scheme, we can calculate the tilt angle γ at which each measurement m_p was taken. The pan-tilt uses a trapezoidal acceleration scheme to achieve any velocity that is greater than the so called base speed v_b. The pan-tilt unit is considered to be able to accelerate instantaneously from zero to any speed up to v_b. Then γ is calculated by interpolation using this scheme.

The spatial coordinates $c = (c_x, c_y, c_z)$, corresponding to the 3D point where the laser beam impacts, must be calculated for each measurement returned by the laser sensor. Thus, it is necessary to look for a transformation function f such that:

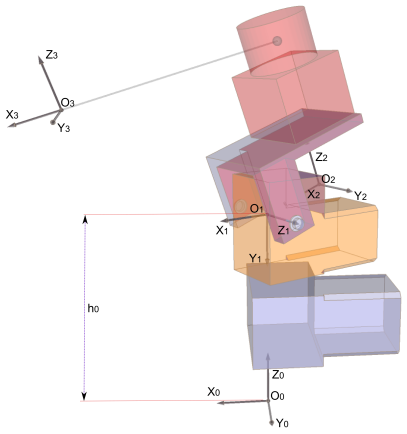


Fig. 2. Data acquisition system envisioned as a kinematic chain. The system is made up of a laser sensor (in red color, upper side) coupled over a two degrees of freedom pan-tilt. In the figure, the base reference system is showed.

$$c = f(\alpha, \gamma, \lambda, m_p, u) \tag{4}$$

If λ is the laser motor, i.e. beam, angle and $u = (u_x, u_y, u_z)$ are the coordinates from the pan-tilt and laser sensor localization. The transformation f can be easily found posing this problem as a direct kinematic problem. From this point of view, our system has three distinct joints. The first and second joints coincide with the pan-tilt's motors. The third joint corresponds to the laser sensor motor, considering the laser beam as another link of the chain (figure 2). The transformation f can then be determined by means of the Denavit-Hartenberg method [14].

A. SLAM

As equation 4 clearly shows, it is necessary to know the pan-tilt and laser sensor 3D orientation in order to merge the 3D scans taken from different places in a single map. The location of each pose is approximated by using the robot odometry. However, as is commonly accepted, odometry errors may grow without limit due to wheel sleepage or calibration errors. Specifically, one can expect odometry errors to increase rapidly with distance and turns. Hence, this error must be corrected in order to create a consistent map. To solve this issue, we first collected all the 3D scans and then the whole scan set was processed using the SLAM 6D package [3], [15]. The SLAM 6D project includes software to register 3D point clouds into a common coordinate frame. We used this registration software to correct the localization of the poses. This software matches 3D scans and it considers 6DoF for the robot pose: x, y and z coordinates and the roll, yaw and pitch angles. As a result, corrected poses are returned. We used the corrected poses to solve equation 4.

IV. ML MAP BUILDING

The first description level of the environment is based on Multilevel Surface Maps (MLSM) [2] and it is built using the 3D scans processed by the SLAM 6D package.

MLSM consists of a 2D grid where every cell $c_{i,j}$ stores a structure list. Each element of this list is represented as the mean $\mu^k_{i,j}$ and the variance $\sigma^k_{i,j}$ of the measured heights at the position of the cell in the map. Triebel et al.'s work [2] is aimed at obtaining an environmental representation that allows for robot navigation in tridimensional environments with several traversable surfaces at different overlapped heights. So, in that work, each list element (called surface patches) represents whether the space at the height indicated by the mean $\mu^k_{i,j}$, with an uncertainty equal to the variance $\sigma^k_{i,j}$, is traversable or not. Our objective, however, is to obtain a map that allows us to model and identify the objects present in the environment. Accordingly, in our map each list element, called block, represents a section of an object surface. This enables us to obtain a map that represents a compact discretization of the environment. This new approach introduces some differences during map building.

Within our ML maps, each cell $c_{i,j}$ stores a list of blocks $b^k_{i,j}$. Each measure $p = (p_x, p_y, p_z)$ returned by laser sensor is incorporated in a block so $p_x \geq j \cdot \text{cell\_size}$ and $p_x < (j+1) \cdot \text{cell\_size}$ and $p_y \geq i \cdot \text{cell\_size}$ and $p_y < (i+1) \cdot \text{cell\_size}$. The cell_size parameter expresses the map resolution. Each block is represented by a tuple (h, σ, d, π), where h is the height, σ the variance, d the depth and π the plane containing it (this last parameter will be explained in the next section). There are two block types:

1) Horizontal blocks represent a section of the external upper or lower surface of an object, for example: a floor section or a ceil part, a table board, etc. This kind of blocks has a depth equal to zero.
2) Vertical blocks, in turn, represent sections of vertical surfaces of objects like walls or wardrobes.

When new measures are acquired, the height and variance of horizontal blocks are updated using the Kalman update rule. In vertical blocks, in turn, the height and the variance are the height and variance of the highest measurement assigned to the block. The depth of a vertical block is the difference between the upper and lower measurements which fit in the block. When new measures are acquired, the map is updated as follows (see figure 3):

- Every time a new measurement (p, σ_m) is added, where $p = (p_x, p_y, p_z)$ are the coordinates and σ_m is the variance corresponding to the measurement, the cell $c_{i,j}$ where the measurement fits is selected.
- In the block list of the cell $c_{i,j}$ we look for a block (h, σ, d, π) that collects the new measurement. A block collects a measurement if $|p_z - h| < cell\_size$ and $|(h-d) - p_z| < cell\_size$.
- If there is a block that collects the measurement and this block is horizontal and $|p_z - h| < 3 \cdot \sigma$, then the height and variance of the block is updated using Kalman's update rule. In this case, the block remains horizontal. If, in turn, the block is horizontal but we obtain that $|p_z - h| \geq 3 \cdot \sigma$, then the block becomes a vertical one with $h = max(p_z, h)$ and $d = |p_z - h|$
- If the block that collects the measurement is vertical then we simply update the block height or depth as needed.

- If the new measurement is simultaneously collected by two blocks (h_1, σ_1, d_1) and (h_2, σ_2, d_2), then both blocks will be joined into a single vertical block and the old blocks are removed.
- If the measurement is not collected by any block, or the block list of the cell is empty, then a new horizontal block will be created with $h = p_z$ and $\sigma = \sigma_m$, and added to the list of cell $c_{i,j}$.

V. Plane detection

In this project, we have developed an algorithm called efficient RANSAC in Multilevel Surface Maps (eRMSM), as a modification of the efficient RANSAC (eRANSAC) algorithm [4]. While eRANSAC works in point clouds, eRMSM works directly over the block structures of an ML map and it focuses on detecting just planes.

If M is an ML map that collects a set of blocks $b^k_{i,j}$, (i, j) is the cell index pair where the block falls in and k is the block index in the cell's list, then the eRMSM algorithm detects and returns a set of planes $\Pi = \{\Pi_1, \cdots, \Pi_n\}$ in the map. Furthermore, each block is labeled with an index i which indicates that the block matches plane Π_i. Matching between a block and a plane implies that the block is close enough to the plane and that the block is part of a block setting with a similar orientation to the plane. When the algorithm stops, each block $b^k_{i,j}$ will be represented as (h, σ, d, π) where π is the index of the matching plane. A block that does not match any plane will have $\pi = 0$.

Iteratively, the algorithm produces candidate planes that are hypothesis of real planes. Each candidate plane (CP) obtains a score that is defined as a function of the blocks matching the plane. As in eRANSAC, at the end of each iteration the CP with the highest score is accepted as a valid plane only if the probability of not overlooking a better candidate is high enough. However, in the eRMSM algorithm we have changed the estimation of this probability. In our algorithm, the number of CP needed to accept a plane as valid is significantly reduced as we will demonstrate in the sequel. When a CP Π is accepted as a valid plane, each block that matches the plane is labeled with the index i of the plane. Once a CP has been accepted, any other CP that matches the accepted plane is removed from the CP list.

Before the algorithm begins, each block $b^k_{i,j}$ receives a direction vector ν. This direction vector will be used so that only blocks with a similar direction vector will produce a new CP. This vector is the normal vector to a hypothetic surface formed by the block $b^k_{i,j}$ and all the same kind of vertical or horizontal blocks in a r radious neighborhood of the block (see figure 4). To speed the process up, in eRMSM we use the Chebyshev distance as the selected distance because it does not change the result. Vector ν is calculated by using the principal component analysis (PCA) [16]. As eRMSM does not work over spatial coordinates, but over map blocks, we must supply, from each block, some coordinates that allow to obtain a vector $\nu \in \mathbb{R}^3$. Two cases must be differentiated:

- Case 1: the block $b^k_{i,j}$ is horizontal. The horizontal blocks are part of the upper or lower surface of an object such as the board in a table, or even the oblique surface of an object like a ramp. So, the direction vector that we are looking for can have any orientation in space. In this case, from the vertical blocks set $B_V = \{b^{k_1}_{i_1,j_1}, \cdots b^{k_n}_{i_n,j_n}\}$ that exist in a setting with radius r of $b^k_{i,j}$ we can obtain a point set $P_V = \{p_1, \cdots, p_n\}$ where $p_i \in \mathbb{R}^3$. Let $b^{k_l}_{i_l,j_l} = (h_l, \sigma_l, d_l, \pi_l)$, then the corresponding point p_l is $(i_l \cdot \text{cell\_size}, j_l \cdot \text{cell\_size}, h_l)$. PCA is applied to P_V to compute the normal vector to the surface that has the P_V elements.
- Case 2: the block $b^k_{i,j}$ is vertical. This block must be part of a vertical object: a wall, a chair back, etc. Hence, the direction vector in this block must be a vector parallel to the ground then. In this case, from the horizontal blocks set $B_H = \{b^{k_1}_{i_1,j_1}, \cdots b^{k_n}_{i_n,j_n}\}$ that exist in a setting with radius r of $b^k_{i,j}$ we can obtain a point set $P_H = \{p_1, \cdots, p_n\}$ where $p_i \in \mathbb{R}^2$. If $b^{k_l}_{i_l,j_l} = (h_l, \sigma_l, d_l, \pi_l)$, then the corresponding point $p_l = (i_l \cdot \text{cell\_size}, j_l \cdot \text{cell\_size})$. PCA is applied to P_H to compute the normal vector (vn_x, vn_y) to the surface that has the P_V elements. Using this two dimensional vector we get the vector $V_N = (vn_x, vn_y, 0)$ which is parallel to the ground.

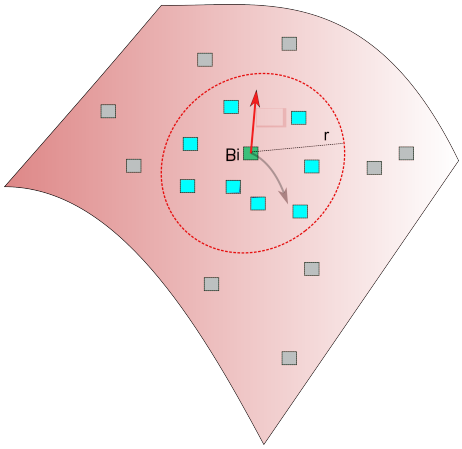


Fig. 4. The direction vector attached to block B_i is normal to a hypothetic surface formed by the block and all blocks of the same class (vertical or horizontal) in a neighbourhood radius r.

Once each block has a direction vector assigned, Algorithm 1 is executed. The candidate plane list obtains the hypotheses about planes in the environment. The detected plane list obtains the hypotheses that have been positively tested. The M variable represents the blocks list. Finally, p_t is the lowest probability considered valid to accept a hypothesis as a true plane.

The candidate planes are generated randomly selecting a block $b^{k_1}_{i_1,j_1}$ and two other blocks $b^{k_2}_{i_2,j_2}$ and $b^{k_3}_{i_3,j_3}$ close to the first that have not been matched to any other accepted plane. The neighborhood radius r is an algorithm parameter that affects the algorithm's behavior. If r is small, then the three blocks may be part of the same surface, but the plane's orientation will be affected by errors of measurement. On the other hand, if r is big, then the possibility of selecting blocks that do not match the same surface increases, but if

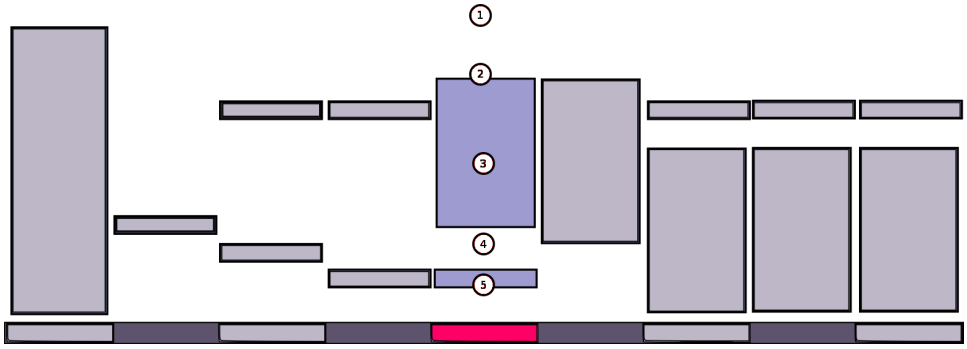


Fig. 3. Different cases when a new observation (numbered circles) is added to ML. For simplicity, the map is presented in 2D. (1) The new observation is far from registered blocks, so a new block is created. (2) The new is close to the top or bottom of a vertical block. Then, its height or depth is updated. (3) If the new observation falls inside a vertical block there are no changes. (4) The new observation is between and close enough to two registered blocks. In this case, both blocks are merged into a single one. (5) The new observation is close to a horizontal block. If the new measure is very close to the block, the height of the block is updated using the Kalman's filter updated rule. If the new observation is further than 3σ wrt. the block's, the block changes to vertical.

the blocks match the same surface, the increased distance will compensate the measurement error. The three selected blocks will generate a CP only if the angles between their direction vectors are lower than a threshold θ.

Algorithm 1 Plane detection in a ML map M

```
L_p ← ∅                                   ▷ detected plane list
L_c ← ∅                                   ▷ candidate plane list
for i = 0 to Max_cp − 1 do
    L_c ← L_c ∪ newCandidates(r, θ)
    b ← bestCandidate(L_c)
    s_c ← SimilarOrientationSurface(b)
    if P(surface(b), s_c) > p_t then
                                          ▷ matching blocks are removed:
        M ← M − M_b
        L_p ← L_p ∪ b                     ▷ CP that matches b are removed
        L_c ← L_c − C_m
    end if
end for
```

In earlier algorithms, the CP is determined as the plane that includes the three selected points (see figure 5 (a) and (b)). By contrast, to filter the surface localization error due to measurement errors, our method determines the CP in another way. The plane generated from the three blocks CP cp_i is determined as a point o and a normal vector to plane V_N. The point o is selected as the barycenter of the polygon with the three blocks as vertex and the normal vector V_N as the mean between the corresponding direction vectors. This CP represents a better hypothesis of a real plane (figure 5 c).

The way a score is assigned to each CP in eRMSM algorithm also varies in relation to previous works. Since our algorithm works with blocks, rather than point clouds, it is not possible to assign the number of matching points to CP as a score, so we propose a new score function. We will now give a definition of matching between a block and a plane. It is said that a block $b^k_{i,j}$ with a direction vector V_D matches a plane $\Pi = (o, V_N)$ if:

- The distance from the block to the plane is $d = dist(b^k_{i,j}, \Pi) < \varepsilon$.

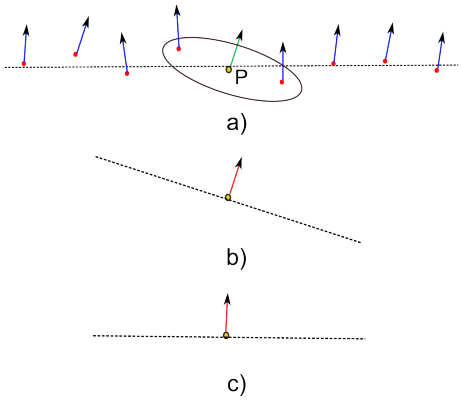


Fig. 5. Different ways to determine a candidate plane. For simplicity, in the figure the problem is depicted in two dimensions. a) Point observations obtained for a measures set of a straight line. b) Determining a candidate line as the line that best fit to three points. c) Candidate line defined as the line with a normal vector that is the mean of the direction vectors of the three points.

- The angle between the block's direction vector and the normal vector to the plane is $\beta = \arg(V_D, V_N) < \kappa$.

The thresholds ε and κ are system parameters that adjust the goodness of the accepted CP as valid planes.

Each CP receives a score depending of the area of the surface that is represented by the blocks that match that plane. To normalize the probabilistic computations, the area is measured in "surface units" s_u, where $s_u = \text{cell\_size} \times \text{cell\_size}\ \text{mm}^2$. Then the CP score is $S = \sum s_{b_i}$, where s_{b_i} is the area of the surface represented by the blocks matching the plane. The block surface depends on whether the blocks are vertical or horizontal. In a vertical block b_v the corresponding surface is $s_v = \frac{d}{\text{cell\_size}}$. When the block is horizontal it has a surface $s_h = 1$. This score method, instead of counting the number of matching blocks, as in the original method, has the advantage of being based on a real indicator of the importance of the plane in the real world. Hence, a CP that corresponds to a large surface has more possibilities of being found early on.

The CP generated in this way is likely to have some deviation in orientation with respect to the corresponding real plane. This deviation is explained by measurement errors

or due to the inclusion of neighboring blocks that in fact do not belong to that plane when computing the block's direction vector. Thus, we apply a refitness process to each CP. Whenever a new CP is created and scored we use the set of blocks that match the CP to adjust the CP's orientation using PCA. In this case, we apply PCA to the set of blocks matched by the CP to obtain the normal vector to this set. Subsequently, this vector is used as the new CP's normal vector. Finally, the modified CP is scored again. This process is recursively applied a predetermined number of iterations or when the score enhancement, from one iteration to the next, is lower than a given threshold (see algorithm 2).

Algorithm 2 Plane refitness procedure

Require: CP_i, the new plane candidate
Ensure: The optimized CP: CP_i
1: $score = score\_function(CP_i)$
2: **repeat**
3: $old\_score = score$
4: $normal = get\_normal(CP_i \rightarrow matched\_blocks)$
5: $modify\_normal\_CP(CP_i, normal)$
6: $score = score\_function(CP_i)$
7: **until** $score - old\_score < threshold$

A CP is accepted as valid only if the probability of not overlooking a better candidate is high enough. As we can see in [4], if $\wp$ is a cloud of N data points and Ψ a shape comprising n points, then the probability of detecting Ψ in a single iteration is

$$P(n) = \binom{n}{k} / \binom{N}{k} \approx \left(\frac{n}{N}\right)^k \tag{5}$$

If k is the minimum number of elements needed to define a shape —$k = 3$ for planes— thus, the probability $P(n, s)$ of successfully detecting a shape after s new candidates have been generated is

$$P(n, s) = 1 - (1 - P(n))^s \tag{6}$$

Finally, the number T of candidates needed to detect a shape of a size n with a probability $P(n, T) \geq p_t$, where p_t is the minimum desired probability, is

$$T \geq \frac{\ln(1 - p_t)}{\ln(1 - P(n))} \tag{7}$$

Applying equations 5, 6 and 7, and assuming that we have as an environment a corner formed by a ground section and two walls, if the number of points in the cloud is equally spread over the three planes, each plane has a third of the total points. Then, as 5 shows, the probability of detecting the ground in a single pass is:

$$P(n) \approx \left(\frac{1}{3}\right)^3 \approx 0.037 \tag{8}$$

Hence, according to equation 7, the number of CP that we need to detect the ground with a probability greater or equal to 0.99 is:

$$T \geq \frac{\ln(1 - 0.99)}{\ln(1 - 0.037)} > 122 \tag{9}$$

Clearly, with other shapes that represent less than a third part of the total information, the number of candidates increase significnatly as usually happens in realistic environments, where most surface planes represent a small portion of the total map. In the eRMSM algorithm, we have introduced changes to estimate the probability of not overlooking a better candidate. These changes considerably reduce the number of CP that is necessary to generate before a plane is accepted as valid.

In our approach, CP are not generated from any three blocks of the map. On the contrary, each CP is exclusively generated from three neighboring blocks with a similar orientation and therefore similar to the orientation of the plane itself. Exploiting that fact, in eRMSM algorithm, if Π is a CP where s_c is the surface of the blocks matching the plane and let s_o be the total surface of all blocks with a similar orientation to Π, we can calculate the probability of finding the plane in a single pass as

$$P(s_c) = \binom{s_c}{3} / \binom{s_o}{3} \approx \left(\frac{s_c}{s_o}\right)^3 \tag{10}$$

In the example of three planes forming a corner, the probability of finding the plane corresponding to the ground in a single pass is 1, since $s_c = s_o$ and then

$$P(s_c) \approx \left(\frac{s_c}{s_o}\right)^3 = (1)^3 = 1 \tag{11}$$

In this case, we have enough with only a single generated CP against the 123 candidates needed using the previous approach. This method can validate CP spurious planes or planes with little significance, i. e. with a small total surface if the blocks matching the plane represent a high percentage of all blocks with an orientation equal or similar to the generated CP. To avoid this, a threshold accepting candidate planes only with a score greater than a value s_m and hence with a minimal surface suffices.

The algorithm exit condition is reached when a given number of candidates is generated.

VI. Results

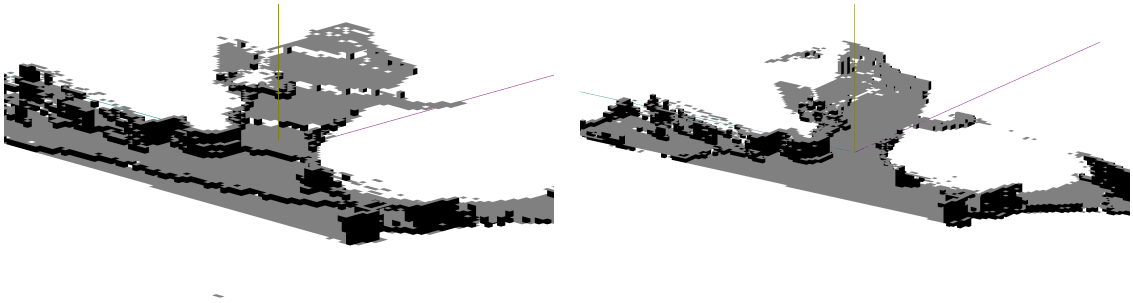
The system presented in this paper has been tested in several locations of the main building of the University of Las Palmas de Gran Canaria's Technological Park.

In the first test, we steered the robot through the basement and took 24 3D scans of the corridor (see figure 6(a)). The corridor's estimated dimensions are $40.4\mathrm{m}$ long and $4.75\mathrm{m}$ wide. The corridor has perpendicular subcorridors $11\mathrm{m}$ long.

No matter how carefully the acquisition system is placed on board the robot, it will not be parallel with respect to the floor. This inclination or deviation from horizontal causes a "step effect" in horizontal and vertical planes. The system solves this issue by self-calibration. At the beginning, a scan of the surrounding floor is adquired (see figure 7(a)). We then use the eRMSM algorithm to detect the floor plane. The normal vector to this plane is used to fix the system's inclination.

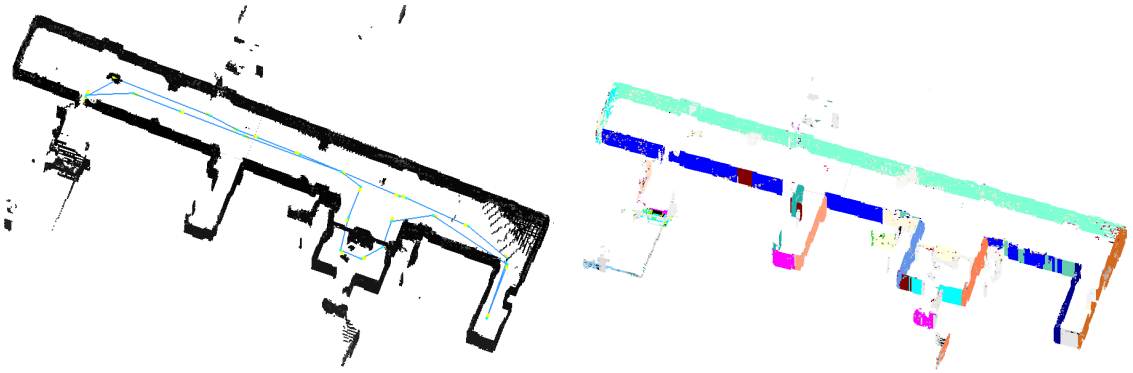
(a) (b)

Fig. 6. (a) Test scenario #1: Corridor of the Technological Park. ULPGC. (b) Test scenario #2: Robotics Laboratory of the Technological Park. ULPGC.



(a) (b)

Fig. 7. (a) "Step effect" in an horizontal plane (Robotics Laboratory's floor). (b) "Step effect" fixed using an estimate of the acquisition system's inclination.



(a) (b)

Fig. 8. (a) Upper oblique view of ML maps generated from 24 poses at the corridor. The line represents the robot path and the points over the line are the poses at which the scans were taken. (b) Planes detected in the corridor map. Grey zones represent blocks that do not match any plane. Each color represents different planes.

We can appreciate in figure 7(b) how the "step effect" has disappeared.

We used the Nüchter et al. SLAM 6D library [3] to correct the odometer location information returned by the robot regarding the robot's pose where the 24 3D scans were taken. Once the poses are corrected, we build a map from the set of measures taken in the 3D scans. A 3D visualization software was developed to make spatial zooms and rotations of the map. In figure 8(a) we can see an upper oblique view from a map of the corridor generated using a 100mm cell size. For improved visualization we have removed the floor and the ceiling from the map. In addition, we can see the poses where the 3D scans were taken from. This map allocates 44732 blocks. Once the map has been generated, the eRMSM algorithm is executed. With an implementation of the algorithm optimized for a 2.4GHz quad-core processor, it is possible to identify 12 planes in 7.8 seconds.

In figure 8(b) we can see the corridor map where the blocks that match any detected plane are depicted using different colors. The largest planes in this map match about 1650 blocks. Using the eRANSAC test it would be necessary to generate 1132 CP (see equation 7) to accept the first plane with a probability greater than 0.9. Using our probability estimation, the first plane hypothesis is confirmed after generating 50 CP in 600ms. At the same scenary, we can see how the different steps of a stair are intified by the system (see figure 9). It is important that the lower steps are detected better than the upper steps. The reason is that upper steps are parallel to laser

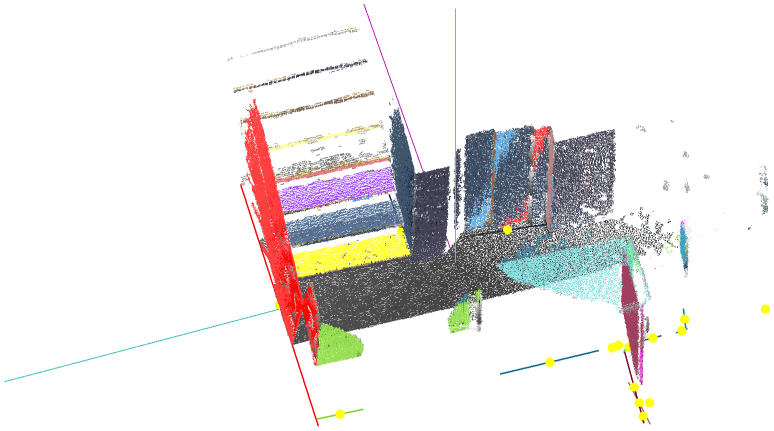
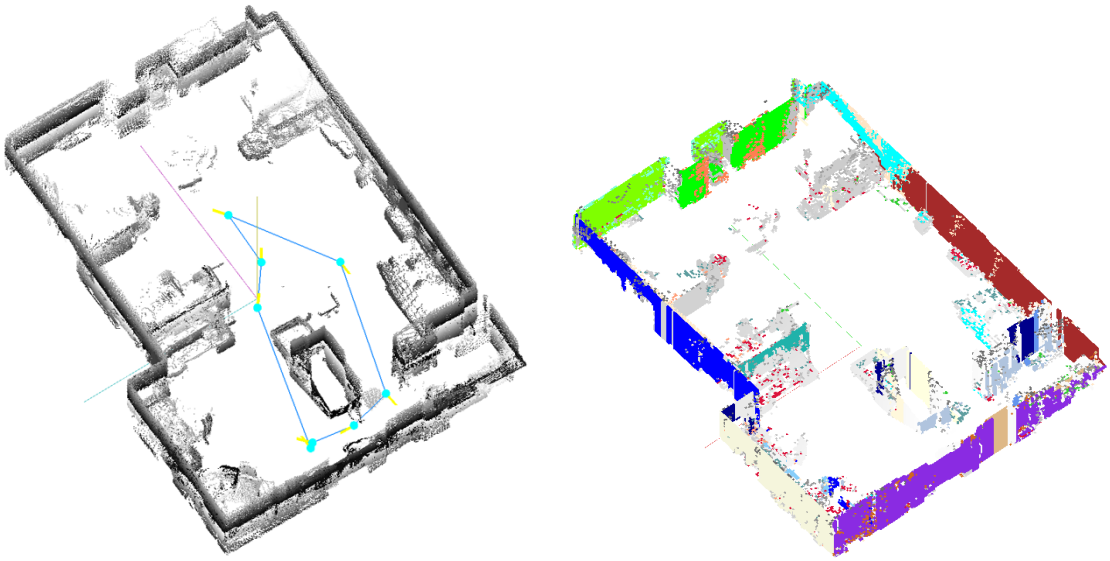


Fig. 9. An example of stairs detection in the middle of the corridor depicted in figure 6(a). The bottom steps are closer to perpendicular to the laser beams than the top steps, and thus they are detected better.



(a) (b)

Fig. 10. (a) Upper oblique view of ML maps generated from 8 3D scans taken at different poses in the Robotic's Laboratory. The line represents the robot path and the points over the line are the poses at which the scans were taken. (b) Planes detected in the Robotics Laboratory's map. Grey zones represent blocks that do not match any plane. Each color represents different planes.

beams, so the sensor does not receive an echo from these steps.

Figure 6(b) shows a new test scene. In this case, the scenario is a laboratory $8.3\mathrm{m}$ wide and $11.4\mathrm{m}$ long. Figure 10(a) shows a map generated using a cell size of $20\mathrm{mm}$. This map collects 196385 blocks. Fourteen different planes were detected in 4.6 seconds. As in the previous test, different colors in figure 10(b) correspond to blocks that match different detected planes.

ML maps, as we have generated them, easily allow the joining or fusion of different partial maps of adjacent spaces. The laboratory shown in figure 6(b) and the corridor of figure 6(a) are contiguous rooms in the same building. Both spaces were independently mapped using our approach, with the results depicted in figure 10(a) and 8(a). We have been able to generate a single map from the two data sets after the poses were corrected using the SLAM 6D software. We can see the resulting map in figure 11.

VII. Conclusions

This paper has described an approach to building compact 3D maps of indoor environments based on multilevel surface maps. This kind of space representation allows us to describe the scene with detail and balances spatial resolution and memory cost appropriately. These multilevel maps are easily scalable and versatile enough to provide sophisticated spatial information without having to rely on low level data, i.e. clouds of laser data points.

In addition, an efficient algorithm for detecting planes using the multilevel surface maps (eRSMS algorithm) has

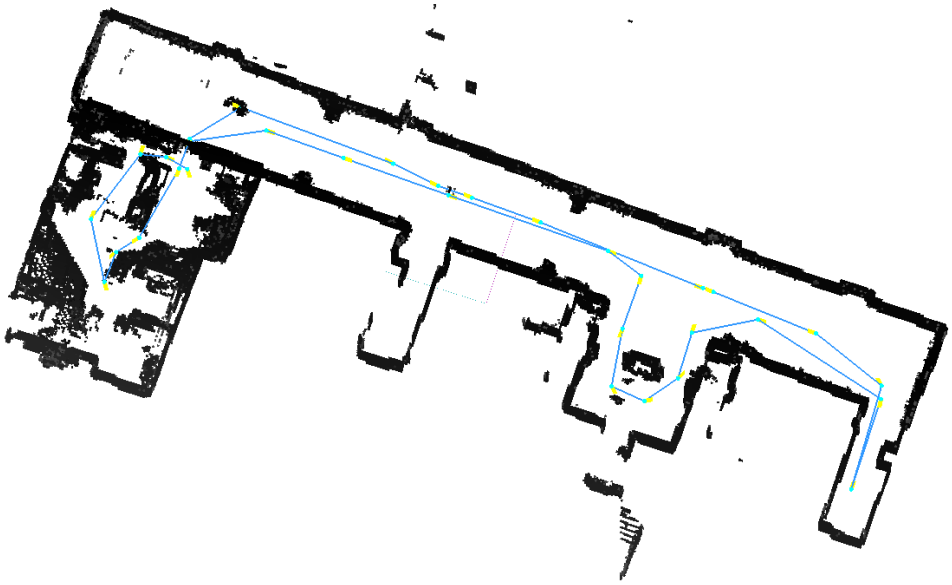


Fig. 11. Single map of the corridor and the Robotics Laboratory after merging the 3D scans independently taken at both scenarios.

been proposed. A key feature of the eRSMS algorithm that distinguishes it from the original eRANSAC algorithm is that it does not need to generate a high number of hypotheses in order to identify candidate planes with high probability. Moreover, eRSMS is easily parallelizable, an attractive feature that may be exploited on multicore processors.

While the system described in this paper has proved reliable, there is considerable room for improvement. Future work will be directed towards alleviating the off-line 6D SLAM pre-processing and the associated computational cost by using geometrical features instead of the scan point clouds. Moreover, the multilevel maps offer interesting possibilities to attempt to label an indoor space semantically.

VIII. Acknowledgements

The authors would like to thank Heather Adams, ULPGC, for checking the English version of this manuscript.

References

[1] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. The SLAM problem: a survey. In *Proceeding of the 2008 conference on Artificial Intelligence Research and Development*, pages 363–371, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[2] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[3] Andreas Nüchter, Kai Lingemann, and Joachim Hertzberg. 6D SLAM with cached K-D tree search. In *RA'07: Proceedings of the 13th IASTED International Conference on Robotics and Applications*, pages 101–106, Anaheim, CA, USA, 2007. ACTA Press.

[4] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, 26(2):214–226, June 2007.

[5] Julian Ryde and Michael Brünig. Non-cubic occupied voxel lists for robot maps. In *IROS'09: Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, pages 4771–4776, Piscataway, NJ, USA, 2009. IEEE Press.

[6] Shrihari Vasudevan, Fabio Ramos, Eric Nettleton, and Hugh Durrant-Whyte. Gaussian process modeling of large-scale terrain. *J. Field Robot.*, 26(10):812–840, 2009.

[7] Cesar Rivadeneyra, Isaac Miller, Jonathan R. Schoenberg, and Mark Campbell. Probabilistic estimation of multi-level terrain maps. In *ICRA'09: Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 3709–3714, Piscataway, NJ, USA, 2009. IEEE Press.

[8] Zoltan Csaba Marton, Radu Bogdan Rusu, and Michael Beetz. On fast surface reconstruction methods for large and noisy point clouds. In *ICRA'09: Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 2829–2834, Piscataway, NJ, USA, 2009. IEEE Press.

[9] Kaustubh Pathak, Narunas Vaskevicius, and Andreas Birk. Revisiting uncertainty analysis for optimum planes extracted from 3D range sensor point-clouds. In *ICRA'09: Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 2035–2040, Piscataway, NJ, USA, 2009. IEEE Press.

[10] GuruPrasad M. Hegde and Cang Ye. Extraction of planar features from swissranger sr-3000 range images by a clustering method using normalized cuts. In *IROS'09: Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, pages 4034–4039, Piscataway, NJ, USA, 2009. IEEE Press.

[11] D. Hähnel, W. Burgard, and S. Thrun. Learning compact 3d models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems*, 44(1):15–27, 2003.

[12] Mariano Martin Nevado, Jaime Gomez Garcia-Bermejo, and Eduardo Zalama Casanova. Obtaining 3d models of indoor environments with a mobile robot by estimating local surface directions. In *Robotics and Autonomous Systems*, pages 131–143, 2004.

[13] Martin Heracles, Bram Bolder, and Christian Goerick. Fast detection of arbitrary planar surfaces from unreliable 3D data. In *IROS'09: Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, pages 5717–5724, Piscataway, NJ, USA, 2009. IEEE Press.

[14] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Trans ASME J. Appl. Mech*, 23:215–221, 1955.

[15] Andreas Nüchter and K. Lingemann. 6D Simultaneous Localization and Mapping, 2010. http://slam6d.sourceforge.net.

[16] I.T. Jolliffe. *Principal Component Analysis*. Springer, New York, NY, 1986.

Motion Planning for Omnidirectional Dynamic Gait in Humanoid Soccer Robots

J.J. Alcaraz-Jiménez, D. Herrero-Pérez, and H. Martínez-Barberá

***Abstract*—This paper deals with the problem of planning the Center of Mass (CoM) trajectory of a humanoid robot while its feet follow an omnidirectional walking pattern. This trajectory should satisfy the dynamic stability criterion to ensure analytically that the Zero Moment Point (ZMP) lies within the support polygon. The proposed approach provides flexibility and agility to humanoid robots, which is of special interest in highly dynamic environments, such as soccer robotics. The experimental results show that the proposed method permits on-line calculation of omnidirectional stable trajectories in the commercial humanoid platform NAO, which has limited computational resources.**

***Index Terms*—Humanoid Soccer Robots, Omnidirectional Locomotion, Dynamic gait, RoboCup.**

I. Introduction

AN IMPORTANT feature that provides flexibility and simplifies drastically the control of mobile robots is omnidirectional locomotion, of special interest in dynamic environments. Omnidirectional locomotion provides the robot with the ability to modify its motion quickly, independently of its bearing, which is very useful when environmental conditions change. Moreover, it facilitates hardware abstraction, allowing us to reuse high level controllers and behaviors from other developments. When possible, omnidirectional drives have been employed in diverse platforms to facilitate robot control.

One interesting example is the Robocup competition. The different Robocup leagues have incorporated omnidirectional drives to facilitate the control of the robots and to provide the capabilities for kicking the ball accurately. The reason is that using an omnidirectional drive makes it easier to locate the robots in specific positions for kicking in the proper direction. Some examples in different leagues are omnidirectional drive for Small-Size League [1], past Sony Four-Legged League [2], legged-robots in Rescue League [3] and humanoid robots [4].

The most popular and simple schemes to control walking bipeds are *trajectory tracking* methods [5], which solve the motion dynamics equations to calculate offline trajectories for individual joints keeping the ZMP within the support polygon [6]. The main shortcomings of these approaches are as follows:

- There is only a finite set of gaits computed off-line.
- They require precise robot and environment models.
- Robustness under relative high disturbances is not ensured by tracking approaches.
- Dynamic equations with many degrees of freedom and trajectory tracking can be computationally expensive.

J.J. Alcaraz-Jiménez, D. Herrero-Pérez, and H. Martínez-Barberá are with the Department of Information and Communications Engineering, University of Murcia, 30100 Espinardo, Murcia, Spain.
E-mail: juanjoalcaraz@um.es, dherrero@um.es and humberto@um.es

The simplified models are used to calculate CoM trajectories and introduce feedback control. The most popular model is the *inverted pendulum* simplification [7], which represents humanoid dynamics by its Center of Mass (CoM) connected by a massless telescopic leg to the supporting foot. Some variants model sagittal and frontal components by means of more than one inverted pendulum [8], representing different linked parts of humanoid robots. On the other hand, central pattern generators [9] [10] need neither a robot dynamic model nor an environment model. They aim to imitate biological neural circuits that can produce rhythmic patterns without receiving rhythmic inputs.

In spite of capabilities that can incorporate omnidirectional locomotion to bipeds, some advanced developments are still controlled by pre-computed trajectories, with the feedback focused on tracking these routes and ensuring stability under slight disturbances. These trajectories cannot be coupled because they usually do not match up and smoothness is not ensured. Consequently, the robot has to stop in order to track a new trajectory.

The online CoM trajectory generation for omnidirectional dynamic gaits has also been studied in several ways. Recent developments [4] aim to address this problem by ignoring the ZMP position and employing empirical sinusoidal equations instead. When CoM trajectory generation is based on ZMP a precise model of the robot is required. For example, the problem can be formulated as a ZMP tracking servo controller [11]. Although this approach has been successfully tested in the Robocup environment [12], it assumes some error in CoM trajectory generation, which is decreased by adding future information about ZMP position. Finally, an analytical solution for CoM trajectory generation problem based on ZMP is proposed in [13]. However, effectiveness of the approach is compromised by fast changes in the expected position of the feet.

The flexibility provided by omnidirectional locomotion and the accuracy of online analytical approaches are the motivation for this paper, which proposes an analytical method for planning the CoM trajectories of an omnidirectional dynamic gait for biped robots. This problem was previously addressed by [13]. The main difference with the proposed approach is that a different set of parameters and a tuning criterion are proposed in order to improve the precision of ZMP trajectories.

The paper is structured as follows. Section II introduces the problem and the proposed approach. Section III describes the walking pattern generation approach adopted to plan omnidirectional trajectories. Section IV presents the boundary conditions employed to trace trajectories in any direction. The experimental validation is presented in Section V, and finally,

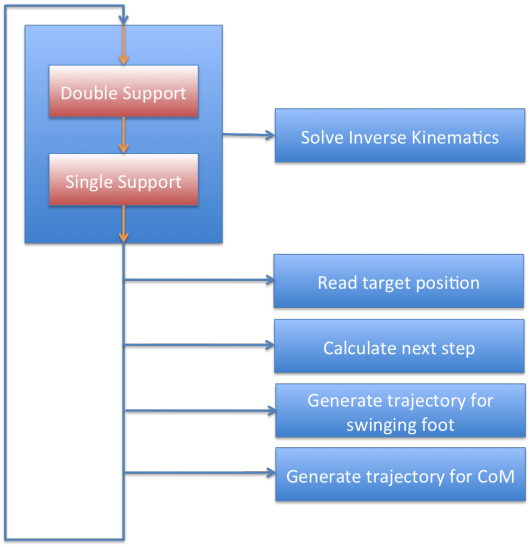


Fig. 1. Overview of the locomotion system.

some conclusions are presented in Section VI.

II. Omnidirectional Locomotion

The proposed locomotion system is based on position control, i.e. the humanoid robot is commanded by a target position and the locomotion system generates the sequence of joint positions to reach such a target location. Through a local coordinate system fixed to the hip of the robot, the target position consists of the relative target position, the relative target orientation, and the feet layout relative to the target reference.

The target position can be modified at any time in order to provide flexibility and reactivity to the robot control. When the target position is modified the sequence of joint positions is recalculated taking into account the stability criterion for humanoid robots. The locomotion problem can be divided into four steps:

1) Finding the target position of the swinging foot given the target position for the feet.
2) Planning the trajectory of the swinging foot for one step.
3) Planning the CoM trajectory to move the swinging foot one step considering single support time and target position.
4) Finding the sequence of joint positions to follow both CoM and swinging foot trajectories.

Fig. 1 shows the different stages of the locomotion system. We can observe that the inverse kinematics processing is the only stage executed at every cycle, which helps to maintain the CPU consumption at a low level. The trajectories for the swinging foot and the CoM are only calculated when a single support stage is finished, and thus robot reactivity is limited to time to walk one step. Reactivity, which is a key issue in robotic soccer, is increased by using short and fast steps.

III. Motion planning

The omnidirectional locomotion problem can be formulated as finding the CoM and feet trajectories to move the robot to any position. Moreover, the trajectories must satisfy the stability criterion for humanoid robots. Two approaches can be employed to determine the stability of the robot: static and dynamic balance criteria.

The static balance criterion assumes that only the gravitational force is acting on the robot, hence keeping the vertical CoM projection on the support polygon ensures stability. The support polygon is the paw area of the supporting foot contacting with the floor in single-support stage, while it is the convex hull including the paw areas of both feet in the double-support stage. However, the inertia forces should be negligible, in order to ensure stability. This can be achieved with static balance, but it gives rise to a slow gait.

On the other hand, dynamic balance takes into account both gravitational force and inertia. Normally, the *Zero Moment Point* (ZMP) is employed to determine the dynamic balance. The ZMP specifies the point with respect to which dynamic reaction forces at the contact of the foot with the ground do not produce any momentum. The dynamic balance condition consists of the ZMP projection on the ground lies within the convex hull. When ZMP projection on the ground is out of the convex hull, ZMP is called *Fictitious Zero Moment Point* (FZMP).

Humanoid gaits usually differentiate between single-support stage (robot standing on only one foot) and double-support stage (both feet on the ground). In order to walk, the robot has to move its legs from double-support stage to single-support stage, alternating between legs. The proposed approach aims to maintain the ZMP in the center of the supporting foot during the single-support stage and to avoid the ZMP leaving convex hull during the double-support stage.

A. Inverted Pendulum model

The 3D Linear Inverted Pendulum Model (3D-LIMP) [14] is used to plan the CoM motion given the ZMP position. Fig. 2 shows this simplified robot model as a single point located at the CoM where all the mass of the robot is concentrated. Such a point is connected to the ground by a massless support leg whose length can be modified. The entire model behaves as an inverted pendulum, turning freely around the supporting point, which is the place where the combination of inertia and gravity forces projects on the floor.

The CoM trajectories are planned considering that the supporting point is located at the ZMP of the humanoid robot (stability criterion). The 3D-LIMP model only considers the propulsion force and the gravity. The former force, applied to the point representing the mass of the robot, can be decomposed into,

$$f_x = (\frac{x}{r})f \quad (1)$$

$$f_y = (\frac{y}{r})f \quad (2)$$

$$f_z = (\frac{z}{r})f \quad (3)$$

where f_x, f_y and f_z are the Cartesian components of propulsion force and r is the distance between the supporting point

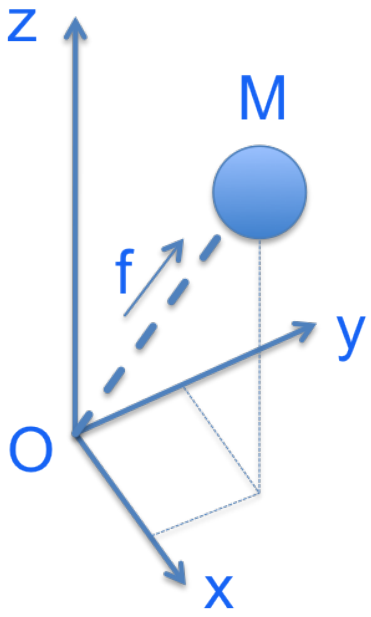


Fig. 2. Inverted Pendulum model.

and the CoM. By adding the gravity to the system, the motion equations defining the movement of the CoM are as follows,

$$M\ddot{x} = (\frac{x}{r})f \quad (4)$$

$$M\ddot{y} = (\frac{y}{r})f \quad (5)$$

$$M\ddot{z} = (\frac{z}{r})f - Mg \quad (6)$$

These motion equations are simplified by constraining the CoM movement to the plane XY at a height as follows,

$$z = k_x x + k_y y + z_c \quad (7)$$

where z_c is the point where the plane XY intersects the z axis, and k_x and k_y are the slopes of the trajectory constrained to such an XY plane. The forces applied to the model should be orthogonal in order to ensure that the CoM remains in this plane. This fact is described as follows,

$$\begin{bmatrix} f(\frac{x}{r}) & f(\frac{y}{r}) & f(\frac{z}{r}) - Mg \end{bmatrix} \times \begin{bmatrix} -k_x \\ -k_y \\ 1 \end{bmatrix} = 0 \quad (8)$$

By replacing z using the expression (7), we obtain the following equation for the propulsion force applied to the CoM,

$$f = \frac{Mgr}{z_c} \quad (9)$$

which should be proportional to the leg length. In addition, it decreases with the height of the CoM. This force can be replaced in (4) and (5) in order to obtain the relationship between the acceleration that should be applied to the CoM and the distance to the supporting point.

$$\ddot{x} = x\frac{g}{z_h} \quad (10)$$

$$\ddot{y} = y\frac{g}{z_h} \quad (11)$$

which can be expressed from the system reference centered at the support point (p_x, p_y) as follows,

$$\ddot{x} = (x - p_x)\frac{g}{z_h} \quad (12)$$

$$\ddot{y} = (y - p_y)\frac{g}{z_h} \quad (13)$$

This is the simplified robot model used for planning the trajectories of the CoM to ensure stability. The following sections describe the strategy adopted to plan omnidirectional trajectories for biped dynamic gait using these simplifications.

B. Trajectory planning

In order to satisfy the stability criterion during the single-support stage, the ZMP position should be approximately constant. In particular, the ZMP should be located in the convex hull defined by the sole of the supporting leg that is in contact with the floor. Thus, p_x and p_y are considered time independent and the previous differential equations can be solved by using the following expression for the CoM movement,

$$x(t) = c_{1x}e^{\alpha t} + c_{2x}e^{-\alpha t} + p_x \quad (14)$$

$$y(t) = c_{1y}e^{\alpha t} + c_{2y}e^{-\alpha t} + p_y \quad (15)$$

where $\alpha = \sqrt{\frac{g}{z_h}}$ is a constant defined for simplification.

These expressions can be used to plan the CoM trajectory that locates the ZMP at some target position. In order to generate a walking pattern, we only have to link these trajectories by moving the ZMP alternatively between feet. However, these kind of CoM trajectories induce large modifications of the ZMP due to the acceleration of the CoM. In other words, strong accelerations make the effect of body link deformation of humanoid robots stronger. Besides, the simplified robot model becomes less accurate when high accelerations exist, which is why the CoM acceleration should be minimized.

The CoM accelerations following the walking pattern can be minimized by using both planning and control based techniques. We have adopted the simple approach of introducing a double-support stage between single-support stages, which decreases the CoM accelerations induced by single-support stages. The double-support stage is introduced by defining the expression that moves the CoM from the position where it finished the last single-support stage to the position where it starts the next single-support stage. In order to ensure speed continuity an expression of four variables, two variables for speed and two variables for position boundary conditions, is chosen to represent the double-support trajectory as follows,

$$x(t) = c_{3x}t^3 + c_{4x}t^2 + c_{5x}t + c_{6x} \quad (16)$$

$$y(t) = c_{3y}t^3 + c_{4y}t^2 + c_{5y}t + c_{6y} \quad (17)$$

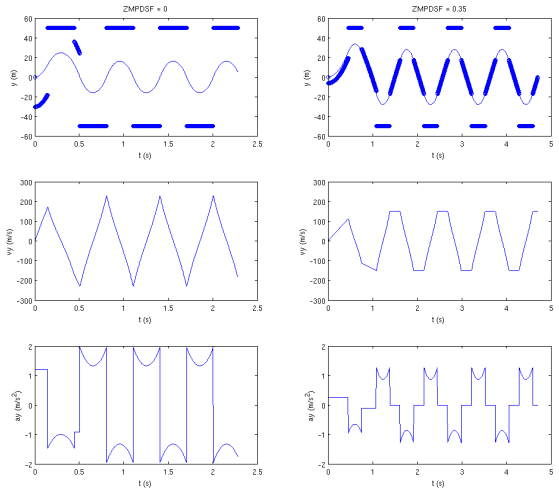


Fig. 3. Comparison between a pure single-support gait (left) and a typical gait with a double-support stage (right). The first row shows a curve with the lateral evolution of the CoM together with the position of the ZMP (thick points). Second and third rows show the lateral speed and acceleration of the CoM respectively.

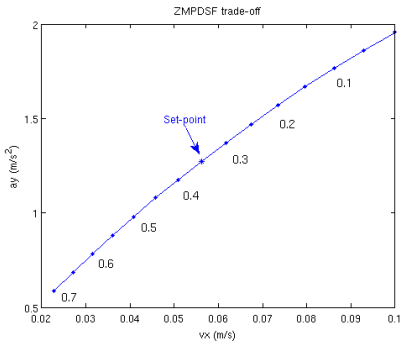


Fig. 4. Relation between average forward speed and maximum lateral acceleration of the CoM while the $ZMPDSF$ parameter is modified. The values of this parameter are shown next to the curve. The value 0.35 for the $ZMPDSF$ is chosen as the set-point for the experiments.

where CoM speed and acceleration profiles can be obtained as first and second derivative of CoM trajectory.

The double-support stage decreases the CoM acceleration between single-support stages and fixes a top for the maximum speed, removing the peaks that would occur in the transition from one support-foot to the next one in a pure single-support gait. The larger this double support area, the lower the CoM acceleration during the transition from one support-foot to the next. However, a larger double-support stage means the robot takes a longer time for each step, resulting in a slower global speed. In order to tune the behavior of our gait in this aspect, the $ZMPDSF$ parameter can be used.

C. Discussion

During single-support stages, the ZMP stays still at one point of the supporting-foot, while in the double-support stage, the ZMP has to travel from one foot to the other. The amount of space between the feet that will be used for the ZMP during its trip from one support-foot to the following one can be specified. In our approach, this amount of space is specified as a fraction of the total amount of space between the feet, and this fraction is the $ZMPDSF$.

Fig. 3 shows the influence of $ZMPDSF$ parameter for pure single-support walking pattern, $ZMPDSF = 0$ (left), and for double-support stage, $ZMPDSF = 0.35$ (right), while maintaining the other parameters of walking pattern generation constant; in particular, step length ($60mm$), feet separation distance ($100mm$), single support time ($0.3s$) and CoM height ($250mm$).

In the first row of this figure, the lateral evolution of the CoM is displayed. The thicker points show the position of the ZMP. It can be noticed that in the pure single-support gait, the ZMP jumps from one foot to the other, while in the version that includes a double-support stage there are samples of the ZMP between the feet. In the second and third rows, which display the lateral evolution of the speed and acceleration of the CoM, the consequences of the incorporation of a double-support stage can be appreciated. The second row illustrates the evolution of the lateral speed of the CoM. In the second column of this row, the peaks of speed are substituted by a flat top, reducing the maximum speed. Likewise, in the third row, the maximum acceleration for CoM in the double-support gait is reduced by about 50%, resulting in a much more stable gait.

However, as can be seen on the time axis, increasing the $ZMPDSF$ parameter makes the robot take longer to step, and thus, it decreases the CoM accelerations at the cost of robot speed. Therefore, this parameter should be determined as a trade-off between robot's speed and dynamic gait stability (including environmental factors, such as floor).

Focusing on average forward speed, and maximum lateral acceleration (which we consider are the main data to evaluate the trade-off between speed of the robot and stability), we have displayed the evolution of their values according to the parameter $ZMPDSF$ in Fig. 4. Given a minimum speed and maximum acceleration constraints, this figure can help choose a proper value for the $ZMPDSF$ parameter. We have obtained experimentally a value $ZMPDSF = 0.35$ for NAO platform, although this value should be tuned for specific surfaces in order to obtain a reasonable speed.

IV. Boundary Conditions

This section presents the constraints used to determine the constants defined in equations (14), (15), (16) and (17). For the sake of clarity, we only present the calculation in the frontal plane of the robot, i.e. lateral balancing. The calculation in the Sagittal plane is similar.

The calculation starts at the transition from the end of the single-support stage of the right foot to the double-support stage (Fig. 5). Accordingly, the time reference starts ($t_0 = 0$) at the beginning of the double-support stage that will move the ZMP from the right foot to the left one. The end of the double-support stage, t_{dl}, will lead to another stage of single-support. During this new stage of single-support, the ZMP will stay on the left foot, while the right foot will move to its target position. When this stage finishes (t_{ld}), the target position for

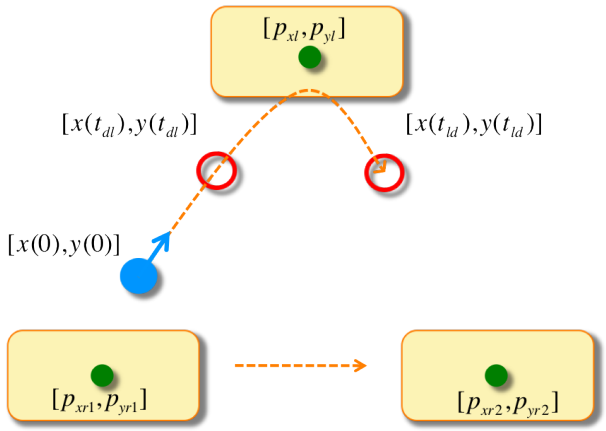


Fig. 5. Positions of the ZMP and the parameter ZMPDSF are employed to obtain the position of the CoM at the extremes of the single-support stage of the current step.

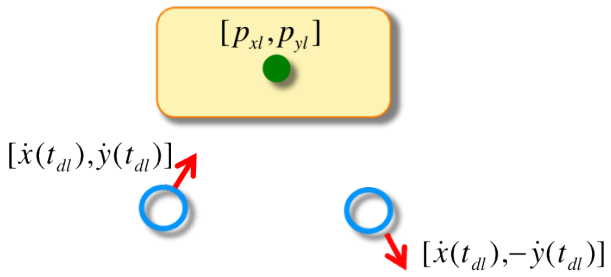


Fig. 6. The initial speed of the CoM in the simple-support stage is defined by the positions of the CoM at the beginning and at the end of this stage and by the duration of the simple-support stage (t_l)

the left foot will be read, and the process will start again. The available information at t_0 is summarized as follows:

- Position of the ZMP on the right foot before t_0, named p_{yr1}.
- Position of the ZMP on the left foot, p_{yl}, during the single-support stage (between t_{dl} and t_{ld}), which will have the left foot as support .
- Duration t_l of this single-support stage.
- Position of the ZMP on the right foot during the next single support stage, p_{yr2}.
- Position and speed of the CoM in t_0, named respectively $y(0)$ and $\dot{y}(0)$.

In equation (15), the data t_{dl}, t_{ld}, $y(t_{dl})$ and $y(t_{ld})$ will be employed to obtain the boundary conditions that will make it possible to find the values of the constants c_{1y} and c_{2y}. Since t_l is already known, it is possible to find $t_{ld} = t_{dl} + t_l$.

As for equation (17), t_0, t_{dl}, $y(t_0)$, $\dot{y}(t_0)$, $y(t_{dl})$ and $\dot{y}(t_{dl})$ will be used for the boundary conditions. Considering t_0, $y(t_0)$ and $\dot{y}(t_0)$ are known, only the values of t_{dl}, $y(t_{dl})$ and $\dot{y}(t_{dl})$ need to be found.

To sum up, in order to solve equations (15) and (17), it is necessary to obtain the values of t_{dl}, $y(t_{dl})$, $\dot{y}(t_{dl})$ and $y(t_{ld})$. This calculation will be detailed in the following lines. Once obtained, it will be possible to calculate the position of the CoM at any time during the single-support or double-support stages of the step.

A. Calculating $y(t_{dl})$ and $y(t_{ld})$

The point $y(t_{dl})$ is the geometric place of the CoM where the transition from the double-support stage to the single-support stage takes place. By employing the parameter ZMPDSF, it will be possible to define the portion of distance between the feet that will be covered by the CoM in double support mode. In this manner, a hint for $y(t_{dl})$ and $y(t_{ld})$ will be obtained.

$$y(t_{dl}^{aux}) = p_{yr1} + (p_{yl} - p_{yr1})(0.5 + \frac{ZMPDSF}{2}) \quad (18)$$

$$y(t_{ld}^{aux}) = p_{yl} + (p_{yr2} - p_{yl})(0.5 - \frac{ZMPDSF}{2}) \quad (19)$$

If there is a strong change in the lateral distance between the right foot and the left foot from one step to the next, there will be a large difference between $y(t_{dl})$ and $y(t_{ld})$. This large difference will lead to speed peaks in one of the extremes that will cause strong accelerations of the CoM in the double support stage.

Since the position of the right foot during the next step is available, it is possible to modify the hint position for $y(t_{dl})$ so that the position and the speed of the CoM at the end of the single support stage are adequate for the next step.

This is why the space in the single support stage has been distributed in a symmetric way, and the final position at the end of this stage is the same as the initial position of the next single support stage: $y(t_{ld}) = y(t_{dl})$. To force this equality, the hint values calculated in (18) and (19) have been averaged.

$$y(t_{dl}) = y(t_{ld}) = \frac{y(t_{dl}^{aux}) + y(t_{ld}^{aux})}{2} \quad (20)$$

B. Calculating $\dot{y}(t_{dl})$

The next value to be obtained is the speed of the CoM at the beginning of the single support stage, $\dot{y}(t_{dl})$, as can be observed in Fig. 6. The duration of the single support stage, t_l, and the recently calculated values of $y(t_{dl})$ and $y(t_{ld})$ will be employed to operate in (15).

$$y(t_{dl}) = c_{1y}e^{\alpha t_{dl}} + c_{2y}e^{-\alpha t_{dl}} + p_y \quad (21)$$

Since $t_{ld} = t_{dl} + t_l$,

$$y(t_{ld}) = c_{1y}e^{\alpha t_{dl}}e^{\alpha t_l} + c_{2y}e^{-\alpha t_{dl}}e^{-\alpha t_l} + p_y \quad (22)$$

If c_{7y} and c_{8y} are defined as follows:

$$c_{7y} = c_{1y}e^{\alpha t_{dl}} \quad (23)$$

$$c_{8y} = c_{2y}e^{-\alpha t_{dl}} \quad (24)$$

Equations (21) and (21) can be rewritten in the following way:

$$y(t_{dl}) = c_{7y} + c_{8y} + p_y \quad (25)$$

$$y(t_{ld}) = c_{7y}e^{\alpha t_l} + c_{8y}e^{-\alpha t_l} + p_y \quad (26)$$

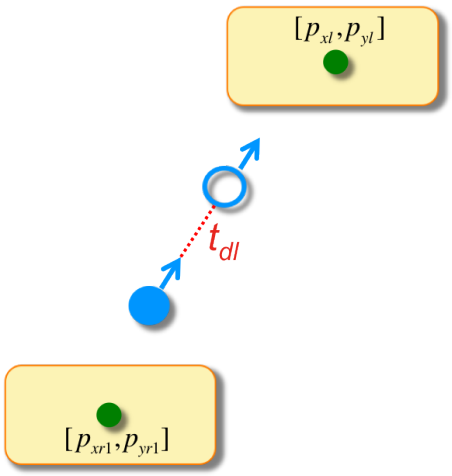


Fig. 7. Position and speed of the CoM at the extremes of the double support stage are employed to generate a value for the duration of the double support stage.

It is possible, then, to find the values of $c7_y$ and $c8_y$ by making use of (20). Finally, by deriving (15), the expression for $\dot{y}(t_{dl})$ can be obtained:

$$\dot{y}(t_{dl}) = \alpha c_{7y} - \alpha c_{8y} \tag{27}$$

C. Calculating t_{dl}

The last value to be found is the duration of the double support stage. This situation is illustrated in Fig. 7. The speeds at the beginning and at the end of the single support stage, $\dot{y}(t_0)$ and $\dot{y}(t_{dl})$ respectively, will be averaged to obtain a guiding value for the CoM speed during the double support stage. In this way, the value of t_{dl} can be found with equation (28).

$$t_{dl} = \frac{y(t_{dl}) - y(t_0)}{\frac{\dot{y}(t_{dl}) + \dot{y}(t_0)}{2}} \tag{28}$$

V. Experimental validation

This section evaluates the proposed locomotion approach. To begin with, the platform employed will be presented together with the environment where the experiments take place. The following subsection is devoted to a brief explanation of the inverse kinematics algorithm for the platform Nao. Later, several experiments will be analyzed: a first group consisting of pure unidirectional movements, and a second where omnidirectional capabilities are demonstrated.

A. Experimental setup

The platform employed to validate the proposed omnidirectional motion planning approach experimentally is the commercial humanoid robot Nao, developed by the French company Aldebaran Robotics. This is the platform used in the *Standard Platform* League of the international *Robocup* Competition event.

The real world experiments are performed on a similar carpet to the official soccer field of *Standard Platform* League,

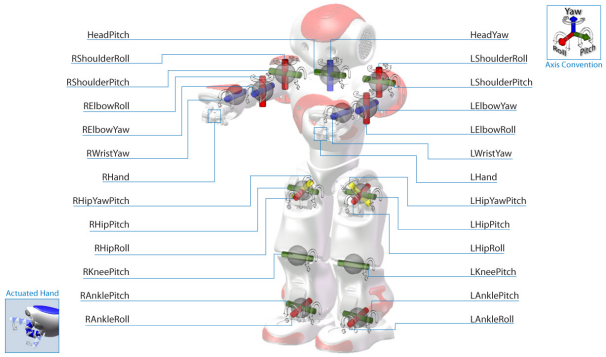


Fig. 8. Nao platform from Aldebaran Robotics Company.

2010 Edition, which is detailed in the official rules of such a league. The simulated experiments are performed in the Webots simulator, from Cyberbotics company. This simulator is convenient, since Aldebaran Robotics, the robot's manufacturer, provides a precise model of Nao platform for it. Moreover, this simulator provides dynamics simulation by making use of the Open Dynamic Engine (ODE), which permits the effect of gravitational and reaction forces to be evaluated.

The humanoid robot Nao has 21 degrees of freedom (DoF) depicted in Fig. 8: two DoF for the head, four DoF for each arm and six DoF for each leg. The number of DoF is twenty-one because both legs share one joint, named $HipYawPitch$ joint, which supposes a constraint in order to solve the inverse kinematics problem. The head joints are controlled by an active vision system depending on perceptual needs. The proposed method does not make use of any sensor, gyroscope and accelerometers available in the platform as feedback to improve the stability of the dynamic gait, i.e. the proposed method is an open-loop approach.

B. Inverse Kinematics

The inverse kinematics problem must be solved in order to calculate the joint positions for a CoM trajectory. This section describes the implementation details for this humanoid platform. This is the information used to calculate the sequence of joints:

- Reference system (located at the supporting leg).
- Position of the CoM (x,y,z).
- Orientation of the torso (Yaw,Pitch,Roll).
- Position of the swinging foot (x,y,z).
- Plane orientation of the swinging foot (Pitch,Roll).

The inverse kinematics problem can be divided into two stages:

- Finding the pelvis position that places the CoM at its target position.
- Finding the joint values for both legs constrained to pelvis position, torso orientation and swinging sole orientation.

The approach employed to find the support side position of the pelvis assumes that the only *DoF* are the joints of the supporting leg. Besides, the modifications of the CoM position are negligible. The aim is to find the position of the supporting

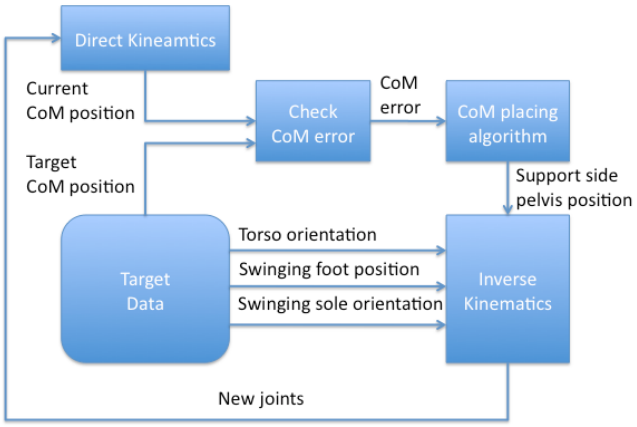


Fig. 9. The iterative method to find the joints values given the CoM target position.

side of the pelvis that places the CoM at the desired position. The solution is obtained by using an iterative method, shown in Fig. 9, in order to decrease the position error of the inverse kinematics problem.

The analytical solution of the proposed method is described next. The method assumes that the position of the supporting side of the pelvis is known, taking the one previously obtained. For the sake of simplification, we will describe the procedure for the right leg of the robot. The reference system is placed on the right sole. The rotation axes of the joints of both legs are numerated from one (right foot) to eleven (left foot). The chain of matrices to be multiplied in order to obtain the orientation of the torso is as follows,

$$R_{OT} = R_x(-\alpha_1)R_y(-\alpha_2)R_y(-\alpha_3)R_y(-\alpha_4) \\ R_x(-\alpha_5)R_y(-\alpha_6)R_x(-\tfrac{\pi}{4}) \quad (29)$$

where α represents the joints of the legs. Besides, the orientation of the left foot can be obtained as follows,

$$R_{OL} = R_{OT}R_x(-\tfrac{\pi}{4})R_y(\alpha_6)R_x(\alpha_7)R_y(\alpha_8) \\ R_y(\alpha_9)R_y(\alpha_{10})R_x(\alpha_{11}) \quad (30)$$

According to the sign criteria adopted, the turning sense is considered positive when the moving part of the body is farther from the torso. Since the reference system moves from one foot to the other, the sign criterion for the turning sense of the joints must be changed. Therefore, the sign of the joints is negative when the reference system is in the right leg. The α values corresponding to the *HipRoll* joints have been merged with an extra rotation in order to simplify the calculation. This simplification consists of orientating the y axis of this joint parallel to the *HipYawPitch* joint as follows,

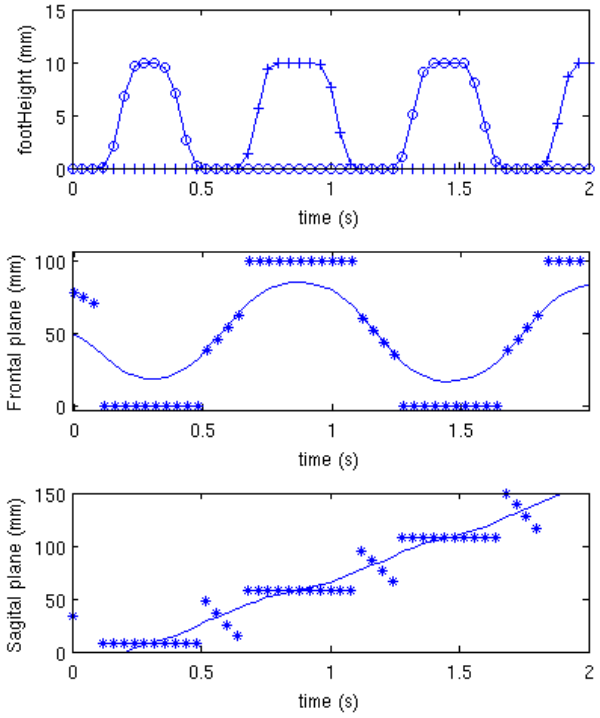


Fig. 10. Forward walking experiment. In the upper row, the height of the left foot (+) and right foot (o) is shown. The second and third rows show the evolution of the CoM (continuous curve) and the ZMP (*) in the frontal and sagittal planes respectively.

RAnkleRoll	α_1
RAnklePitch	α_2
RKneePitch	α_3
RHipPitch	α_4
RHipRoll	$\alpha_5 + \frac{\pi}{4}$
RHipYawPitch	α_6
LHipYawPitch	α_6
LHipRoll	$\alpha_7 - \frac{\pi}{4}$
LHipPitch	α_8
LKneePitch	α_9
LAnklePitch	α_{10}
LAnkleRoll	α_{11}

The first angles to be obtained are those of the right leg of the robot (ankle and knee). These three angles enable the robot to place the support side of the hip at its target position. Once we have obtained the joint values α_1, α_2 and α_3, we can use (29) to calculate the joint angles α_4, α_5 and α_6 by identifying the terms in the resulting matrix from the left members of (33),

$$R_{RThigh} = R_x(-\alpha_1)R_y(-\alpha_2)R_y(-\alpha_3) \quad (31)$$

$$R_{OT} = R_{RThigh}R_y(-\alpha_4)R_x(-\alpha_5)R_y(-\alpha_6)R_x(-\frac{\pi}{4}) \quad (32)$$

$$R^t_{RThigh}R_{OT}R_x(-\frac{\pi}{4})^t = R_y(-\alpha_4)R_x(-\alpha_5)R_y(-\alpha_6) \quad (33)$$

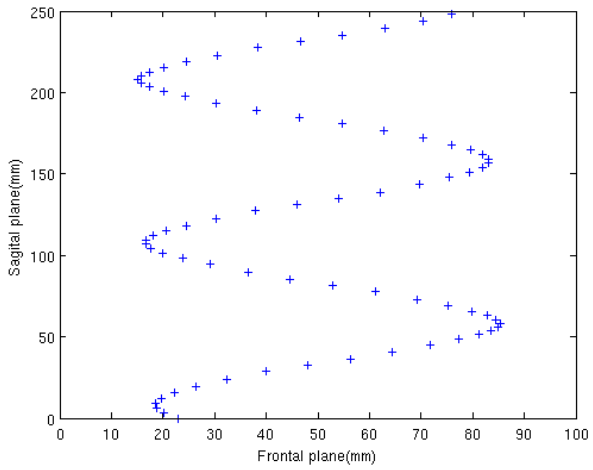


Fig. 11. Horizontal plane of CoM trajectory in a forward walking trajectory.

where we have made use of the property of rotation matrices in which inverse matrix is similar to its transpose.

Once the position and orientation of the swinging side of the hip and the position of the swinging ankle are known, it is possible to make use of the initial target information to find the joint values of α_7, α_8 and α_9. Finally, the α_{10} and α_{11} joint angles are obtained by updating the orientation of the reference frame and comparing it to the required orientation of the sole plane of the left foot.

C. Walking pattern experiments

This section shows the trajectories of two different walking patterns that make use of the proposed motion planning algorithm. These walking patterns are pure forward and lateral straight movements of the robot, since they allow us to illustrate clearly the behavior of the ZMP.

The height of the CoM will be fixed at 235mm, and a sampling time of $40ms$ is employed to generate the trajectories. In this case, the experiments are only tested on the Webots simulator.

In the first experiment, the robot walks straight in the forward direction, while the behavior of the CoM and the ZMP is analyzed. This is illustrated in Fig. 10. The first row indicates the height of the feet. When the circle points have a value above zero, it means that the right foot is in the air, therefore the left foot is the support-foot. In this case, the ZMP should stay still on the left foot. The opposite happens when the cross points are the ones which have a positive value. On the other hand, if no foot is in the air, the ZMP is constrained to stay at any place between both of the feet, that is, within the convex hull.

In the second row of Fig. 10, the lateral component of the evolution of the CoM is shown. The ZMP position is marked with star points. Since the lateral coordinates of the feet are constant (0 for the left foot and 100 for the right one), we note that the ZMP stays on the support foot during the single-support stages and that it moves between the feet during double-support stages.

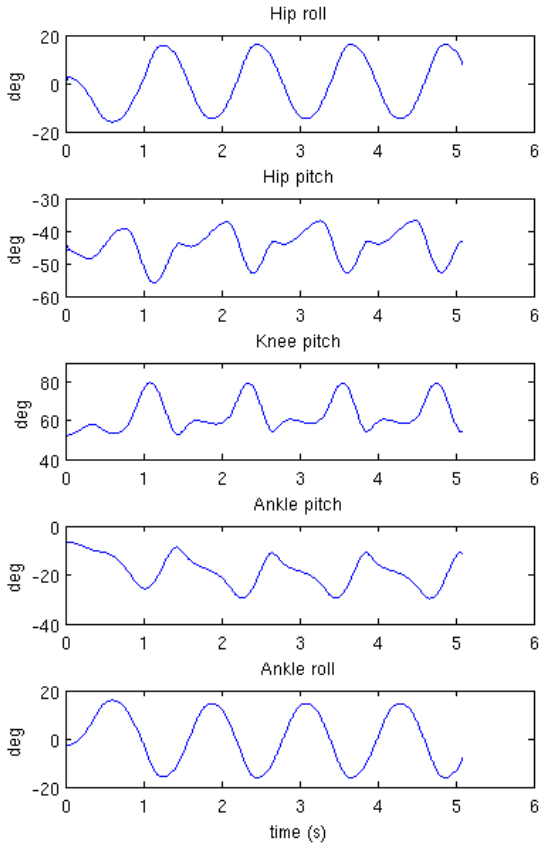


Fig. 12. Position of joints of left leg during forward walking experiment.

The bottom row of Fig. 10 shows this time the evolution of the CoM and the ZMP in the sagittal plane. Once again, the ZMP stays still during the single support stage and moves along the convex hull during the double-support stage. The slope of the curve plotted here is the forward speed of the robot.

Fig. 11 is a combination of the sagittal and frontal movement of the CoM plotted in Fig. 10. This time, the trajectory of the CoM in the horizontal plane is given. In order to get a timing reference, the distance between samples must be observed, since both axes refer to spatial information. The fact that the samples which are around the lateral extremes of the trajectory are closer than the ones in the middle, indicates that the CoM movement is slower during single support stages and faster during double support ones.

Fig. 12 and Fig. 13 show the joint position and the joint speed respectively when the robot is following a forward trajectory. We can observe in Fig. 13 that the joint speed is bounded at 100 $degrees/sec$, which can help to visualize the pace of the movements. Additionally, it is important to note that we do not appreciate any discontinuity in the joint speeds. This is one of the goals of the design of the locomotion system in order to provide smooth movements to the gait which increase stability.

The second pure straight movement to be evaluated is the

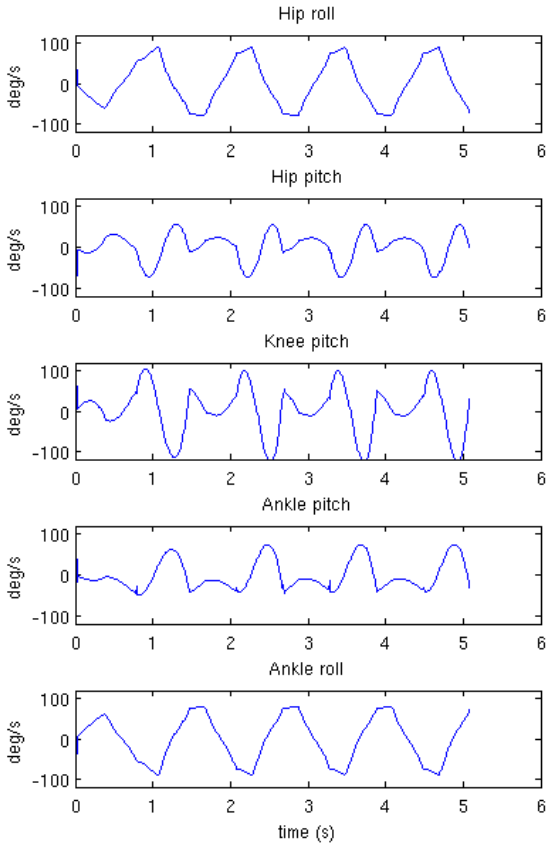


Fig. 13. Speed of joints of left leg during forward walking experiment.

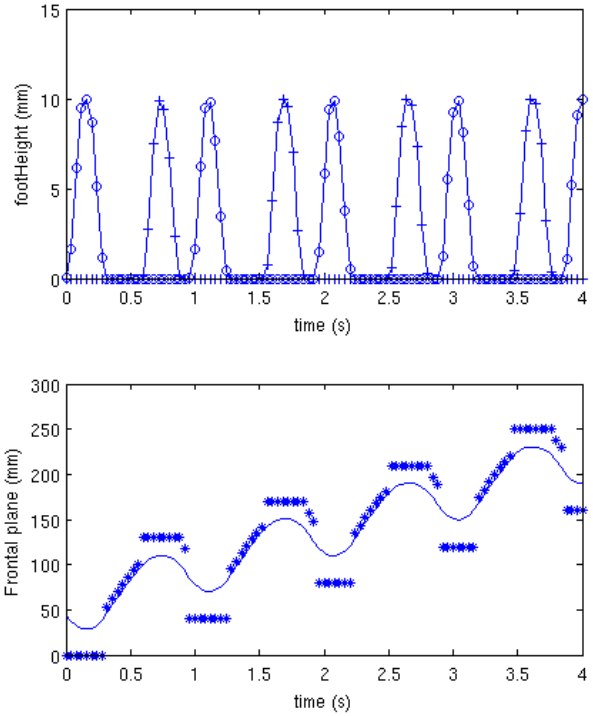


Fig. 14. Lateral walking experiment. In the upper graph, the height of the left foot (+) and right foot (o) is shown. The lower graph shows the evolution of the CoM (continuous curve) and the ZMP (*) in the frontal and plane.

lateral one. Analogously to Fig. 10, the evolution of CoM and ZMP trajectories are displayed (below) together with the height of the feet (above). In this case, a graph for the sagittal plane component of the movement is not shown, since its value is constant. We can also observe that the ZMP is kept between both feet during the double support stage, so ensuring stability. The average slope of the curve displayed in the lower graph is the average lateral speed of the robot.

D. Omnidirectional experiment

The above experiments have demonstrated the stability of the robot while performing pure forward and lateral movements. The next step in order to get an omnidirectional locomotion is to incorporate turning capabilities and simultaneous combination of the previous walking patterns. That will be the target of the next experiment: the circular walking.

Fig. 15 shows the trajectory of the CoM and the ZMP in the horizontal plane while the robot walks describing a circle. The CoM trajectory is the smooth curve described by the star points. The ZMP trajectory (circle points) can be divided in three groups of samples: an inner ring, an outer ring and a regular pattern of samples enclosed within this two rings. The inner and outer rings are formed by the ZMP positions on the left and the right foot respectively during the single-support stages. The samples enclosed within these rings belong to the double-support stages of the walking. It is important to notice that the double support samples are far from the region delimited by the single-support samples, so avoiding instability risks.

The temporal evolution of the trajectories is difficult to extract from the figure, since it is not explicitly shown. For instance, during the single-support stages, the ZMP samples overlap, resulting in a single point in the plot for all the samples of every single-support stage.

The final feature to be evaluated in the proposed method is the capability of linking different trajectories dynamically, which will be tested in the last experiment. In this test, the real robot is follows an online generated walking pattern during 10 seconds. At that time, the gait is forced to follow a new walking pattern abruptly. The different stages of the experiments are: forward walking, smooth curve to the left (5 degrees per step), stronger curve to the right (20 degrees per step), forward walking again and left sense turning (30 degrees per step). Fig. 16 shows the trajectories followed by the robot in the transverse plane. We can observe the transitions between walking patterns, and how the proposed approach solves the problem by generating smooth and continuous trajectories.

VI. Conclusion and Future Work

This paper has presented the motion planning approach used to generate online dynamical stable trajectories of the CoM of a humanoid robot that adapts to the omnidirectional walking

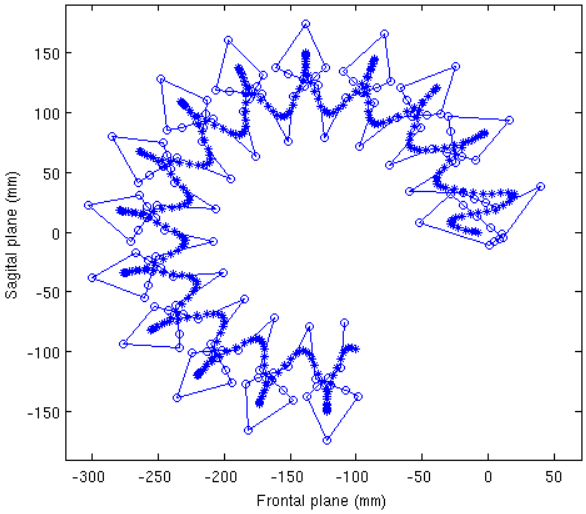


Fig. 15. Circular walking experiment. Trajectory of the CoM (star points) and the ZMP (circle points) in the transverse plane.

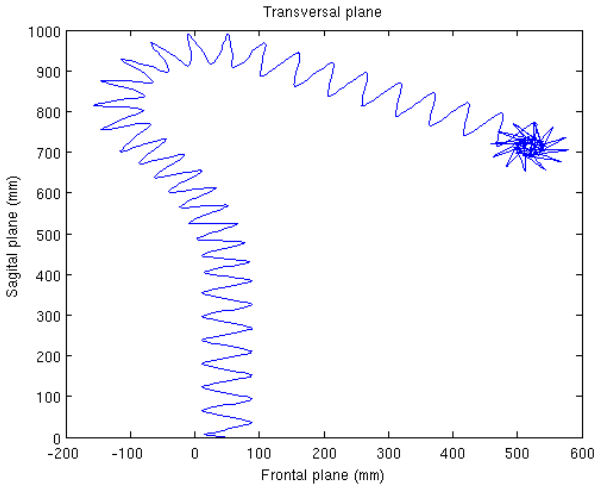


Fig. 16. Omnidirectional walking experiment. Different walking patterns are linked ensuring analytically stability.

patterns followed by its feet. The method has been designed to provide humanoids with the capability of fast reaction when environment changes rapidly, which is of special interest in dynamic and/or uncertain environments. The implementation of the method is focused on efficiency to meet the hard real-time constraints in this kind of applications. Currently, the omnidirectional gait is not optimized for velocity, but it can be used for specific tasks, such as approaching an object or positioning humanoids for accurate operations. The experimental results have shown that this method allows modification of the walking direction and orientation without stopping.

Future efforts will be focused on closing the loop control by making use of the gyroscope and the accelerometers sensors that incorporate the platform. Besides, stability can be improved using rhythmic patterns for arms, depending on the CoM trajectories.

ACKNOWLEDGEMENT

This work has been supported by Spanish Ministry of Science and Innovation under DPI-2007-66556-C03-02 CICYT project and by Spanish Ministry of Education through its FPU program.

REFERENCES

[1] T. Kalmár-Nagya, R. D'Andrea, and P. Ganguly, *Near-Optimal Dynamic Trajectory Generation and Control of an Omnidirectional Vehicle*, Robotics and Autonomous Systems, 46(1):47–64, 2004.

[2] B. Hengst, S.B. Pham, D. Ibottson, and C. Sammut, *Omnidirectional Locomotion for Quadruped Robots*, RoboCup 2001: Robot Soccer World Cup V, Lecture Notes in Computer Science, vol. 2377, pp. 73–94, 2002.

[3] K. Kamikawa, T. Arai, K. Inoue, and Y. Mae, *Omni-directional Gait of Multi-Legged Rescue Robot*, in Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 2171–2176, New Orleans, LA, USA, 2004.

[4] S. Behnke, *Online Trajectory Generation for Omnidirectional Biped Walking*, in Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 1597–1603, Orlando, Florida, USA 2006.

[5] Qiang Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Arai, N. Koyachi, and K. Tanie, *Planning Walking Patterns for a Biped Robot*, in IEEE Transactions on Robotics and Automation, 17(3):280–289, 2001.

[6] M. Vukobratović, A.A. Frank, and D. Juricic, *On the Stability of Biped Locomotion*, in IEEE Transactions on Biomedical Engineering, 17(1):25–36, 1970.

[7] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa, *The 3D Linear Inverted Pendulum Model: A Simple Modeling for a Biped Walking Pattern Generation*, in Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 239–246, Maui, Hawaii, USA, 2001.

[8] Y. Breniere, and C. Ribreau, *A Double-Inverted Pendulum Approach of the Human Gait*, Journal of Biomechanics, 31(1):86–86, 1998.

[9] A.J. Ijspeert, "Central Pattern Generators for Locomotion Control in Animals and Robots: A Review", *Neural Networks*, 21(4):642–653, 2008.

[10] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng, *Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot*, Int. Journal of Robotics Research, 27(2):213228, 2008.

[11] S. Kajitak, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi and H. Hirukawa, *Biped Walking Pattern Generation by using Preview Control of Zero-Moment Point*, in Proc. of IEEE Int. Conf. on Robotics and Automation, Taipei, Taiwan, 2003.

[12] S. Czarnetzki, S. Kerner, O. Urbann, *Observer-based dynamic walking control for biped robots*, in Robotics and Autonomous Systems 57(8):839–845, 2009.

[13] K. Harada, S. Kajitak, K. Kaneko and H. Hirukawa, *An analytical Method on Real-time Gait Planning for a Humanoid Robot*, in Proc. of IEEE/RAS Int. Conf. on Humanoids Robots 2, pp. 640–655, Los Angeles, CA, USA, 2004.

[14] S. Kajitak, F. Knaehiro, K. Kaneko, K.Yokoi and H. Hirukawa, *The 3D Linear Inverted Pendulum Mode: A simple modeling for a biped walking pattern generation*, in Proc. of 2001 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 239–246, Maui, Hawai, USA, 2001.

Robust Behavior and Perception using Hierarchical State Machines: A Pallet Manipulation Experiment

R. Cintas, L. J. Manso, L. Pinero, P. Bachiller and P. Bustos

***Abstract*—Interacting with simple objects in semi-controlled environments is a rich source of challenging situations for mobile robots, particularly when performing sequential tasks. In this paper we present the computational architecture and results obtained from a pallet manipulation experiment with a real robot. To achieve a good success rate in locating and picking the pallets a set of behaviors is assembled in a hierarchical state machine. The behaviors are arranged in such a way that the global uncertainty of the task is progressively reduced when approaching the goal. To do so, actions are generated in each stage that increase the confidence of the robot of being in that particular relation to the world. In order to set up this experiment, it is required a non-trivial set of working senso-motor behaviors. We build on this set to design and test a pallet moving task in which the robot has to locate, approach, obtain the pose, pick up and, finally move the pallet to its target position. The only sensory sources of information available to the robot are a binocular vision system and its internal odometry. To carry out this task we have equipped a RobEx robot with a 1 DOF forklift and a 4 DOF binocular head. We present the conceptual and computational models and the results of the experiments in a real setup.**

***Index Terms*—autonomous robots, mobile manipulators, active perception**

I. Introduction

DESIGNING the computational structures to be used for the execution of complex sequential plans involving manipulation is an important problem in mobile manipulators. [24]. Current state of research in this area is moving from the initial mapping and navigation skills towards smarter plan execution capabilities. However, building up new skills on top of previous ones is not an easy task. New algorithms of very different nature (plan executives) have to coexist with well known, but not yet fully understood, solutions to supporting abilities such as calibration, local navigation, localization, mapping or object recognition. Derived from sensor noise or from the ever increasing number of software lines of code (running on always limited computational resources), new complexities arise when dealing with real robots. Furthermore, teams of many developers and the need for code reuse, settle even more demanding requirements on today's technology. A promising approach is to use component-oriented specialized middlewares[2], [19], [11], [3] that provide a means to divide, reuse and organize large amounts of sophisticated and changing code, typical in robotic research environments. In this work we use RoboComp [1], [11], [20], an open-software robotics framework entirely developed in our laboratory. It provides, among other features, a wide variety of components and a set of tools specifically designed to facilitate software development.

Complex sequential tasks involve different abilities such as active visual searching, detection, recognition, pose estimation, maneuvering, picking and delivering. Building on the infrastructure provided by RoboComp we can more easily focus on the actual problem. As a simple but realistic example of these sort of tasks, we have selected the problem of manipulating a pallet by a mobile robot. To this end, we use the RobEx platform [10][14] equipped with a 1 DOF frontal forklift. All the sensor information available to the robot comes from a 4 DOF stereo head and the odometry of the robot platform. The most interesting aspect of this experiment and the result we want to stress here, is that each transition that takes the robot closer to the target is also designed to reduce the uncertainty in the robot-pallet spatial relation. We thus interleave actions to reach the goal with actions to perceive it, building specific representations in each stage. When the task begins and the robot is searching for something that resembles a pallet, many remote objects can satisfy the initial detection criteria. The representations used to maintain these initial hypothesis are simple and inaccurate. However, as the robot proceeds toward the target, more complex representations are used and more computation time is spent in order to refine these representations. During the approaching stage the robot keeps itself focused on the target by performing attentional eye movements. Thus, as the robot gets closer and new tests are performed, the confidence on the target being a pallet increases.

The rest of the paper is structured as follows: Section II provides an overview on previous works related with pallet manipulation and task execution. Section III details the overall design of the experiment and the development process. Section IV presents the main features of RoboComp and RobEx Section V provides a list of the software components used in the experiment. Section VI describes the states the robot enters during task execution and their purpose. Section VII covers the results obtained from the experiments. Finally, section VIII provides the conclusions extracted from the work and details the future works that will be carried out.

II. Related Work

Nowadays, industrialization and automation in storages is a resource increasingly on demand. Technological advances have allowed the development of sophisticated and automated equipments to give support in storage tasks. However, the control and supervision of this kind of equipment can become a complex task that may require an expert hand [9].

R. Cintas, L. Pinero, L. J. Manso, P. Bachiller and P. Bustos are with the University of Extremadura.

E-mail: rcintas@unex.es

There are many robotic devices specifically designed to manipulate pallets which can be used to optimize the space in warehouses and to improve the safety and speed conditions. The problem can be decomposed in two separate tasks: point to point navigation and pallet manipulation. The first one is typically divided into *fixed path navigation* and *open path navigation*. In fixed path navigation systems a magnetic or reflective element is fit to the floor, physically defining the actual paths used by the AGV's. In open path navigation systems the AGV uses a method to localize itself in the workspace and a planner to compute free paths reaching the current goal. Usual localizations methods are based on laser, inertial sensors, odometry and/or a fixed detectable pattern (optical or magnetic) covering the whole workspace [12].

In the pallet manipulation task there are many possibilities, ranging from knowing the absolute position of all the pallets in the workspace at any time, to a much more flexible markless visual detection and servoing scheme, such as the one we present here. In the works presented in [17][5][21] the authors use a color-based segmentation method combined with a priori knowledge about the geometry of the pallet. With this information the algorithm recovers its pose and generates a trajectory to pick it up. In [8] the AGV uses a 3D laser to detect the pallet, avoiding this way the problems associated with changes of illumination. In other works by the same authors, a laser scanner is also used for localization and generation of free-obstacles trajectories in factory buildings [25]. There are also some proposals that employ different sensory devices to provide better performance or even include the information of additional devices such as sonars [22].

In this work, we explore a different approach which is based on an active detection process using a sequential plan. Developing a passive detection algorithm that provides good performance in all different real situations seems infeasible. Instead, in our approach the problem is solved using a perceptive loop where the robot can hypothesize about what is being perceived, make decisions to reduce the uncertainty of its perceptions and act according to the correctness of its predictions.

III. Sequential task design

Robust manipulation by mobile robots requires a careful design of a sequence of states and transitions in order to act properly in the different task stages. Safe error recovery is a very desirable feature, whether when performing actions, or when perceiving the environment. This is even more important when using only odometric and visual information. In case of errors, a good option is to start over from a previous stage, even going back to the main plan if it is necessary[21]. Thus, errors make iterative the sequential task design, leading in some cases to a control logic of considerable complexity.

The formalism of state machines (e.g. as developed by Harel[6]) is a widely known tool that can be used to solve this problem. Statecharts provide a graphical means of modelling how a system reacts to stimuli. This is achieved by defining the possible states of the system, and how the system can switch from one state to another (transitions between states). A key characteristic of event-driven systems is that behavior often depends not only on the last or current event, but also on preceding ones. With statecharts, this information is easy to express. Qt Software has recently released a state machine framework based on Harel's Statecharts[18]. This framework provides an API and execution model that can be used to effectively embed the elements and semantics of statecharts. It provides us with concurrent and hierarchical structures that can be used as executive engines for robust plan execution. When combined with a component-oriented architecture, the concurrent dimension of the state machines can be easily extended to a fast growing network of these machines, keeping a reasonable bound in the complexity that needs to be managed by developers and researchers. We use this framework embedded in RoboComp.

Before deeply describing the state machine used in the experiments and the restrictions imposed to the environment inhabited by the robot (see section VI), this section provides an overall description. The task of pallet delivering is decomposed in a list of subtasks. This list is generic enough to be useful for different other proposes:

1) Gather context information.
2) Search for a target object candidate.
3) Approach to gain a favourable point of view.
4) Verify the target and gather initial information.
5) Refine object information.
6) Approach and pick/grasp the object.
7) Manipulate the object.

Note that this sequence of tasks is quite generic and can be applied to a wide variety of robots, applications and environments. Each of these subtasks represents an intermediate state towards reaching the final goal. To do so, each state should be associated with the corresponding algorithms that solve the specific problems, whether locally or through calls to remote components. Also and no less importantly, there are different failure conditions, local and remote, that can occur during the execution of each subtask. In order to avoid major problems, these error conditions have to be managed by transitions to former or halt exception states. These states are not denoted in the former list of subtasks but appear in the graph shown in figure 4.

The goal of this experiment is to analyse the advantages of using a state machine framework inside of a component-based robotics middleware in order to run a complex sequential task. To provide a complete description of the whole system, two description levels will be given. The first one, shown in the next section, describes the network of components that controls the robot, providing a coarse description of the system. The second one, in section VI, describes the state machine designed for the experiment.

IV. RoboComp and RobEx

RoboComp

RoboComp is a component-oriented robotics framework. It was created in 2005 by the Robotics and Artificial Vision Laboratory of the University of Extremadura. Since then, it has been widely used by many students and researchers of

the laboratory. Now, it can be considered a mature project which integrates many components with different functionalities: hardware interfacing (e.g. cameraComp, differentialRobotComp, laserComp, forkliftComp), data processing (e.g. visionComp and roimantComp, for visual features detection, and cubafeaturesComp, for laser features detection), robot behaviors (e.g. gotopointComp, wanderComp) and many others. Apart from its wide set of components, it provides other useful features such as: **a)** a flexible organization, easing the addition of new components; **b)** utility scripts for creating and modifying components; **c)** a graphical component manager that allows setting up component networks and monitoring of their behavior dynamically; **d)** transparent connection to open source simulators (i.e. Gazebo and Stage); **e)** an automated installation script; **f)** logging facilities; **g)** recording and playback of component data structures for off-line development and debugging; **h)** rapid Python prototype development support. Beside all these features, RoboComp can seamlessly use two different communication middlewares: a) Ice, a industrial grade middleware created by ZeroC and b) DDS, a high-performance publish/subscribe middleware that has been incorporated so that it can also support RMI-alike calls[15].

RoboComp is also equipped with a numerous set of classes comprising different issues related to robotics and computer vision such as matrix computation, hardware access, Kalman filtering, graphical widgets, fuzzy logic or robot proprioception. Among the different available classes, the robot proprioception class, which we call *InnerModel*, plays an important role in this work. It deals with robot body representation and geometric transformations between different reference frames, lightening the handling of many questions related to analytical and projective geometry. For instance, figure 1 shows the different reference frames that take part in the problem of pallet manipulation. These reference frames are associated with all the mobile elements of the robot, but also with the floor, objects of the environment and virtual elements. The class *InnerModel* provides the mathematical support to represent and manage all this elements. It is based on an XML description file where all the transformations nodes are identified and described. Using this description, *InnerModel* creates an internal representation of the kinematics of the robot and its environment through which it provides many methods to estimate projections and frame transformations.

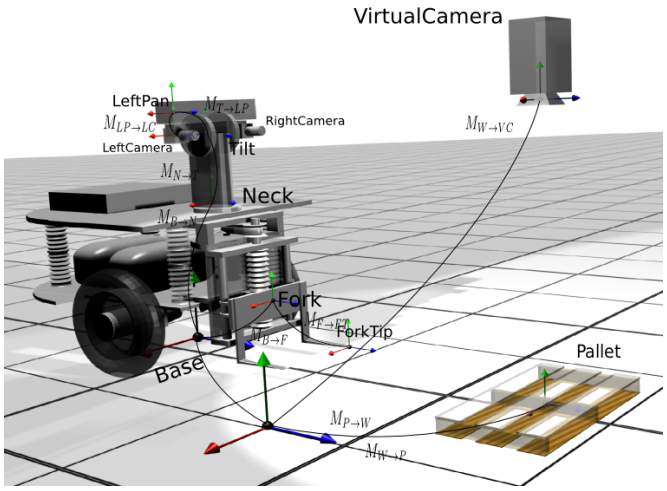


Fig. 1: Reference frames taking part in the pallet manipulation problem.

RobEx

RobEx is an open-hardware robotics platform that incorporates different accessories forming a totally equipped robot. It presents all the necessary features to conduct real experiments in computer vision, robot manipulation and mobile robotics.

For the pallet manipulation problem, RobEx has been equipped with a jointed stereo vision head and a 1 DOF forklift (see figure 2). The head provides 4 movements: a neck movement followed by a common tilt and two camera-specific pan movements. The neck allows cameras to point to objects on the sides of the robots without moving the robot platform. The tilt allows the robot to point to low or high positions. The pan movements can be used for vergence fixation of objects lying approximately in front of the camera pair. This is particularly useful in order to increase the binocular space and to reduce the 3D triangulation error. The forklift is similar to its industrial counterparts. It is capable of supporting loads of up to 5kg safely and has a span of 150mm. The gap between the forks separation can be manually adjusted.

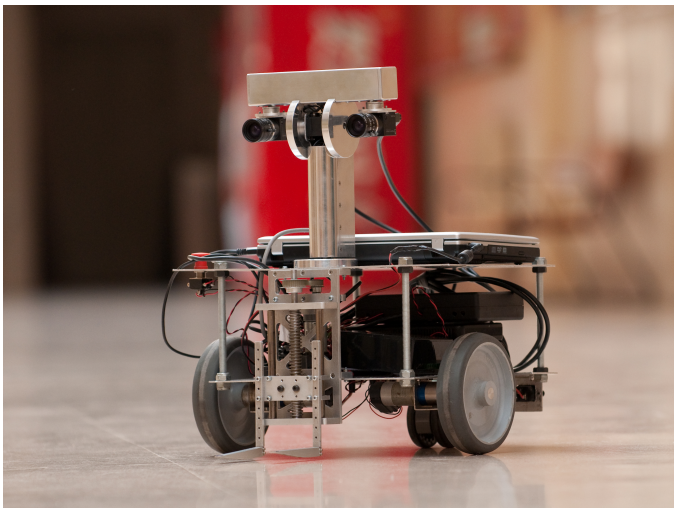


Fig. 2: The RobEx platform.

V. Basic Components

Each node of the component network contributes with a particular functionality to the whole system. Besides *Forlift*, the component that holds the state machine that sequences the behaviors presented in this paper, there are several other components. Some of them are goal-oriented and are associated with behaviors (e.g. Tracking or Trajectory), others play a passive role. A list of the components with a brief description of their function is detailed below.

- DifferentialRobot: Provides an API to control a differential mobile robot. It currently supports the RobEx, Scitos

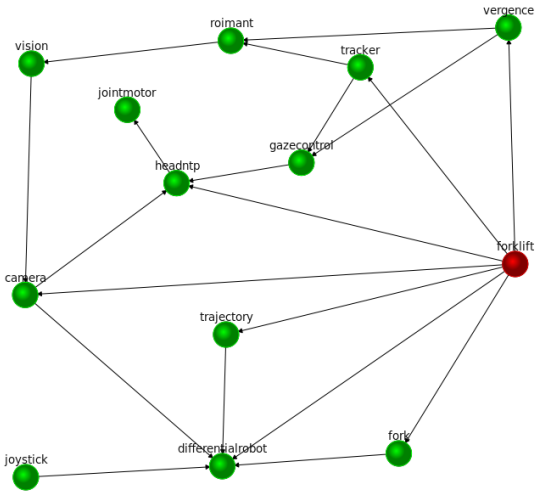


Fig. 3: Component network.

and Morlaco robots, as well as the open source Gazebo simulator and for the Player hardware abstraction layer.

- JointMotorArray: It is used to control motor arrays sharing a communication bus. The component provides configuration parameters for the bus and for each motor. The provided API can be used to command motors individually or synchronously. Besides new bus drivers can be added easily, it currently supports a wide variety of motors.
- Fork: Provides an API to access forklifts. Currently it supports the RobEx forklift manipulator[14].
- JoyStick: It is used to manually send motor control commands to robot platforms.
- HeadNT2P: Provides an API to control a stereo head with four degrees of freedom: neck (common pan), common tilt movement affecting both cameras, and two separate camera-specific pan movements (see figure 1). Its API makes available commands to trigger single or coordinate saccadics. In order to perform movements this component relies on the JointMotor component.
- CameraArray: Accesses arrays of cameras that use the same communication bus. Provides configuration parameters for the bus and for the image retrieval process. Its API allows single or synchronized multiple image retrieval. Currently, the component supports Firewire, V4L2, Gazebo and the privative SDK's from Prosilica and Point Grey. New camera drivers can be added by subclassing and abstract "Camera" class.
- Vision: Computes regions of interest as local extrema in Harris-Laplace pyramid. It provides the list of regions along with the image pyramids. If a suitable GPU is available, the component can compute SIFT descriptors at video rate on the detected regions using SiftGPU.
- Roimant: This component stabilizes the ROI's computed by Vision. In stereo configurations it also maintains in memory a locally updated copy of the regions visible in the world around the robot. It computes the 3D coordinates of regions using a standard correlation measure and the epipolar geometry as reported by HeadNT2P.
- Tracker: Controls a camera to provide a tracking behavior on a certain ROI or initial angular coordinates. It can apply correlation over the whole pyramid to recover from failure situations.
- RobotTrajectory: Computes and follows local trajectories using odometric information. It can compute Bézier curves to fit initial and final orientation conditions for the robot.

The resulting network of components can be seen on figure 4. The following section covers *Forklift*, the component specifically designed for the experiment.

VI. A State Machine for Pallet Manipulation

In this section, it is described the design of *Forklift*, the most relevant component of the experiment and the one that holds the task-specific state machine. The statechart illustrating the behavior of the component is shown in figure 4.

Some of the algorithms used in this experiment use top-down mechanisms in which the representation of the world is compared with the actual inputs of the cameras. In order to maintain such a representation, components use a class named *InnerModel*. Instances of this class are not synchronized but updated by remote calls to DifferentialRobot and HeadNT2P components (which maintain the odometry and the positions of the joints, respectively). Using this object, Forklift builds a basic 3D representation of its environment using the OpenSceneGraph engine (OSG)[16].

The remaining of this section chronologically describes the states that the component (the state machine describing its behavior) would go through assuming absence of any kind of error. Error-triggered transitions are specified within each state description.

A. Getting floor color

The component initially assumes that it is initialized with a flat colored floor underneath the robot. Thus, when in this state, the floor color is obtained from the central region in the left camera image. In order to accomplish this step, the robot points down directly to the closest area in front of its body. The color is further used in order to specify how the floor should look like in the 3D representation of the environment mentioned at the beginning of the section.

Figure 5 shows the initial 3D world representation after extracting the color of the floor.

B. Search for a target object candidate

The next step deals with the acquisition of a target candidate that can become a certain goal after a few selected actions are taken. As can be seen in 11, depending on the distance and relative orientation to the pallet, the visibility conditions may vary drastically. We have developed an algorithm for this stage that reliably detects close pallets and suggests good candidates when the distance to the target increases. The algorithm processes the images in several steps beginning with a superpixel segmentation as reported in [4]. This step performs a color

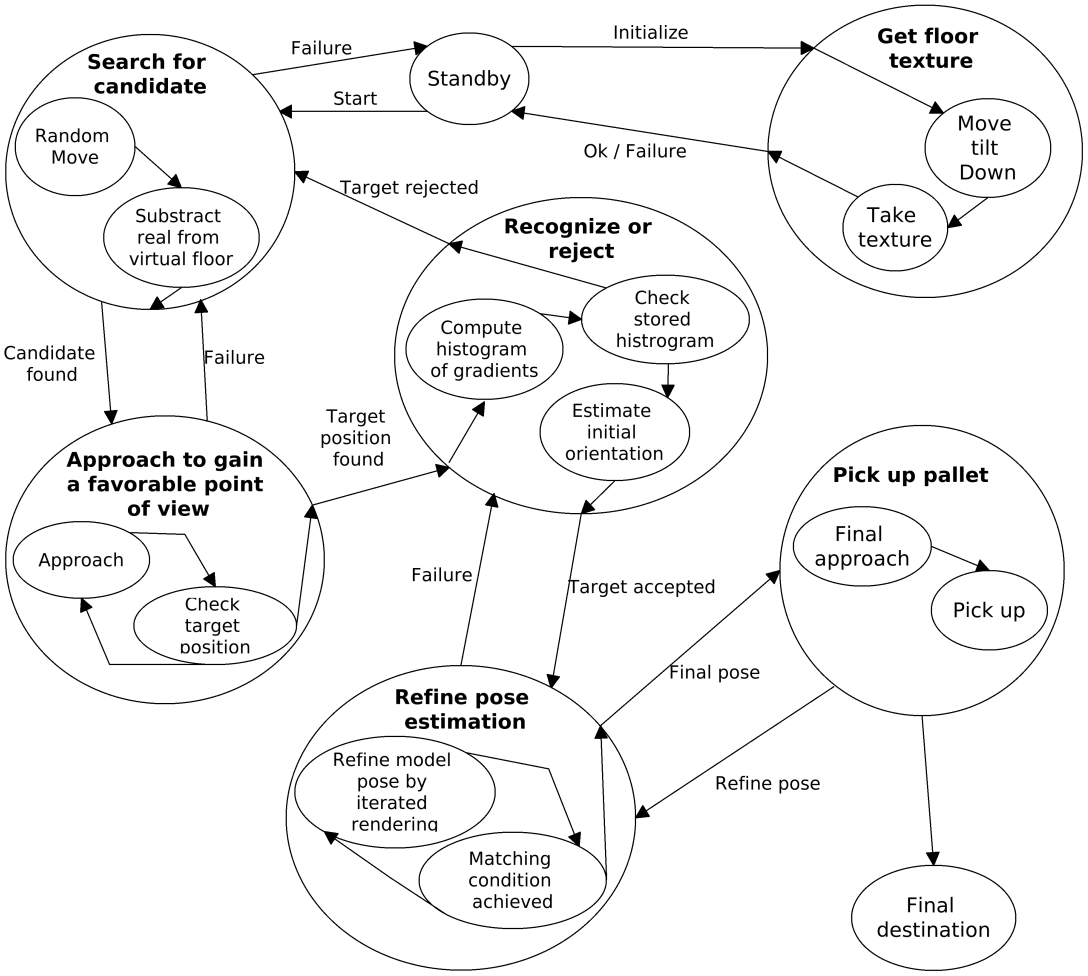


Fig. 4: Hierarchical state machine used in the experiment. There are six macro states and the *Standby* and *Final destination*. Each macro state includes inner states representing with finer detail the structure of each stage. Note the arrows pointing back to former states, signalling failure situations that prevent the normal working of the plan

based partition of the image using graph techniques. The result of processing an image with this technique can be seen in figure 6.

The superpixel segmentation outputs a list of regions, each one pointing to a list of pixels in the image. We describe now the remaining steps carried out by the algorithm and depicted in figure 7.

- Gray scale transformation and range reduction down to 100 bins using the following expression:

$$g_i = 100 * (R_i + G_i + B_i)/755 \tag{1}$$

  where g is the final gray level assigned to pixel i. The goal of this step is to reduce the sensibility of the segmentation algorithm to small variations in color.
- Apply a flood fill algorithm placing seeds at the center of each gray level region. The output is a list with the position and size of the rectangles surrounding the regions and the total number of pixels inside each one. The result is shown in figure 7 under the subtitle *Detected regions*.
- Grouping of compatible overlapping regions to further reduce the number of separate regions belonging to the same object. The criterion for compatibility is expressed in the following conditions:

$$merge(r_i, r_k) \Leftrightarrow \begin{cases} \|color(r_i) - color(r_k)\| < T_c \\ \wedge \\ size(r_i) \cap size(r_k) > T_a \end{cases} \tag{2}$$

  where *merge()* is a predicate that merges regions r_i and r_k if both conditions are satisfied, being $color(r_i)$ the mean rgb color and $size(r_i)$ the size in pixels of i. T_c is an empirical threshold for color absolute difference and T_a a threshold for size difference. The set S of current regions is updated correspondingly:

$$\begin{cases} \mathbb{S}_{t+1} \leftarrow \mathbb{S}_t - \{r_i, r_k : r_i, r_k \in \mathbb{S}_t \wedge merge(r_i, r_k) = true\} \\ \mathbb{S}_{t+1} \leftarrow \mathbb{S}_t + \{merge(r_i, r_k)\} \end{cases} \tag{3}$$

  The output of this step is shown in figure 7, over the subtitle *Overlapping*.
- In this step most of the regions belonging to the floor are eliminated from the current list and the mean color of the floor is reestimated. To do so, regions are sorted by

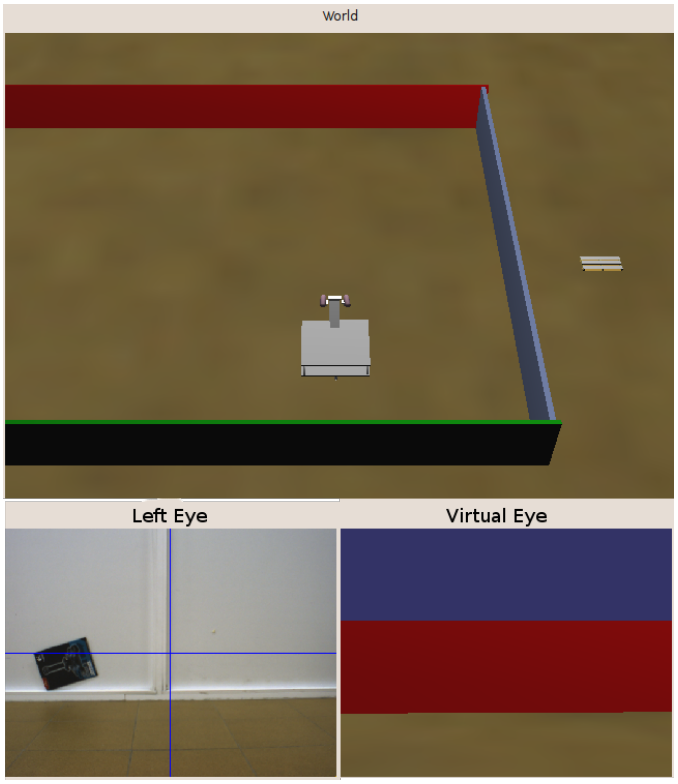


Fig. 5: Top: the 3D representation of the modelled world. Bottom: an image fetched from the real camera (left), and the predicted image using the 3D engine (right).

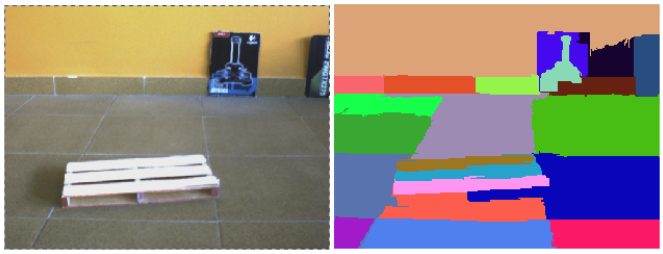


Fig. 6: Left, original image. Right, superpixels obtained with the Felzenswalb and Huttenlotcher segmentation algorithm

the value of the Y coordinate (vertical axis in image) in ascending order. The list arranged this way holds, in the initial positions, the regions situated lower in the image and, therefore, closer to the robot. for each one of them, its mean rgb color is compared to the current floor color, as obtained in the *Get floor color* subsection. If the region color is close enough to the floor color, the region is removed form the current list S:

$$\mathbb{S}_{t+1} \leftarrow \mathbb{S}_t - \{r_i : r_i \in \mathbb{S}_t \wedge rgb(r_i) < T_c\} \tag{4}$$

and the mean floor color is updated using the following expression:

$$rgb(F) = rgb(F) * (1-\lambda) + rgb(r_i) * \lambda \tag{5}$$

where $color(F)$ is the RGB current mean color of the floor and $color(r_i)$ is the mean color of region i. The output of this step is shown in figure 7, over the subtitle *Floor substraction*.

- Now the shape of the regions is analysed to eliminate those with a clear elongated shape. To obtain a better estimate of the region shape than the provided as an enclosing square by the flood fill algorithm, we compute the auto-correlation matrix of the points belonging to the region:

$$M = \begin{pmatrix} \frac{\sum(\bar{x}-x_i)^2}{N} & \frac{\sum(y_i-\bar{x})(x_i-\bar{y})}{N} \\ \frac{\sum(y_i-\bar{x})(x_i-\bar{y})}{N} & \frac{\sum(\bar{y}-y_i)^2}{N} \end{pmatrix} \tag{6}$$

A simple check on the ratio between the eigenvalues of M gives us a criterion to eliminate elongated regions, such as those corresponding to the junctions among the floor tiles. Then set of admitted regions S gets updated as:

$$\mathbb{S}_{t+1} \leftarrow \mathbb{S}_t - \left\{ r_i : r_i \in \mathbb{S}_t \wedge \frac{\lambda_1}{\lambda_2} < T_\lambda \right\} \tag{7}$$

being λ_1 and λ_2 the two eigenvalues of M. The output of this step is shown in figure 7, over the subtitle *Shape analysis*.

- The last feature analysed is the size of the region in the world reference system. To estimate it we assume that the object is on the floor. Knowing the geometry of the robot and of its cameras it is straightforward to backproject the optic rays passing through any pixels of the region, and calculate the point of intersection with the floor plane. From these 3D coordinate an overall size can be easily computed. Those regions too big or too small are removed form S:

$$\mathbb{S}_{t+1} \leftarrow \mathbb{S}_t - \{r_i : r_i \in \mathbb{S}_t \wedge size(r_i) < T_s\}. \tag{8}$$

being T_s a threshold on admitted sizes derived from knowledge of the pallet real size. The output of this step is shown in figure 7, over the subtitle *Size restriction*.

- The final candidate is selected comparing all the remaining regions in the list to a model pallet P stored in memory. Empirically, the most reliable feature to select a final candidate is its RGB color, so a direct check using the euclidean RGB distance to the model pallet color is performed and the best region selected:

$$\mathbb{S}_{t+1} \leftarrow \left\{ r_i : \underset{i}{argmin}(\|rgb(r_i) - rgb(P)\|) \right\} \tag{9}$$

C. Approach the candidate object to gain a favorable point of view

Once a candidate object has been detected, the robot starts an approaching behavior that should take it to a close and favorable point of view. We define here *favorable* as a combination of the distance from the robot to the object and the percentage of image it occupies. To accomplish this subtask several concurrent behaviors must be active. In order to move the robot towards the target position, the world coordinates of the target are given to the previously mentioned

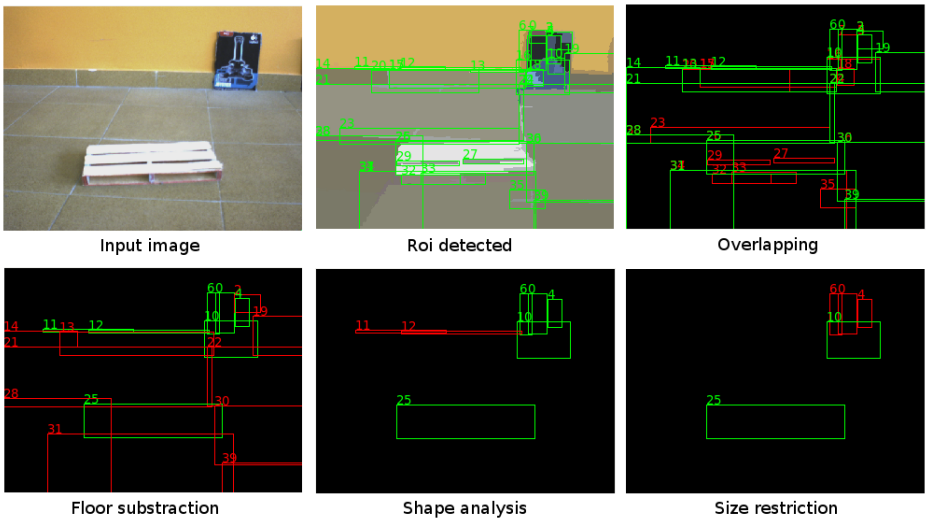


Fig. 7: Segmentation and classification process for extraction of candidate regions. See text for details.

"RobotTrajectory" component. It computes the path to be followed and drives the robot according to it. Trajectories are computed using Bézier curves so, not just the final position can be provided, but also a specific final orientation. Third degree Bézier polynomials are simple and easy to use curves, as long as the initial and final orientations can be specified. Regarding the cameras, the tracking component fixates the candidate object triggering correcting saccades on the left camera when needed. It is worth mentioning that, despite the robot has a stereo vision system, only the left camera is used.

D. Recognize or reject the candidate object and estimate its orientation on the floor

When the robot enters the "recognize or reject" state, a rapid test to accept or discard the candidate object is run. At the same time, its orientation on the floor is estimated. The process has two different stages. First, a set of texture descriptors computed on the regions of interest is requested to the VisionComp component (see graph of components in Figure 3). Then, we match the obtained set of descriptors against a collection of templates using a simple voting scheme[1]. If the classifier returns a positive answer, the object is recognized as a pallet and the subtask proceeds.

The second stage consists on computing the main orientation of the object. This is achieved by calculating the histogram of gradients of the bounding box surrounding the candidate object. The orientation of the pallet on the floor is computed as the main mode of the histogram. This completes the estimation of the initial pose of the pallet and triggers the beginning of the next state.

E. Refine object pose estimation

When entering this state, the robot *believes* that it is taking a close look at a pallet. However, its pose estimation being still imprecise, he decides to refine the estimated pose of the pallet. A known 3D wire-frame model of the pallet with its real dimensions is used to achieve this task. Using the OSG 3D engine, the pallet model and the already mentioned InnerModel class (a continuously updated representation of the state of the robot), it is easy to render the virtual pallet. This way, the scene is rendered with the pallet in the estimated pose, as it should be seen by the real left camera of the robot. This virtual image is subtracted from the real image using a euclidean metric in RGB space:

$$I_{Diff} = \left\| I_r - R_{x,y,\lambda})) \right\| \tag{10}$$

The result is converted to grayscale and binarized using an adaptive Otsu threshold. Finally, all white pixels are counted to obtain a score. This value must be greater than a predefined threshold. If it is not, the pallet is also rejected. If this test succeeds the procedure is iterated varying the position and orientation of the pallet in a small range to obtain the pose that minimizes the sum of white pixels:

$$P = \underset{x,y,\lambda}{argmin} \left\| I_r - R_{x,y,\lambda})) \right\| \tag{11}$$

P is the final pose, I_r is the current image as taken by the left camera and $R_{x,y,\lambda}$ is the image synthesized by the OSG rendering engine with the model pallet set at x, y, λ coordinates.

The pose P is selected as the new estimated pose. This loop is repeated twice, reducing the second time to half the search range in the x, y, λ dimensions. Figure 8 shows how the wire-frame model looks when it is drawn in the image from the real camera and the corresponding image from the virtual camera.

F. Final approach and pick up operation

Once a good estimate of the pose is obtained, this last state moves the robot towards the pallet by a remote call to the "RobotTrajectory" component. Before exiting this state, the forklift should have entered smoothly through the pallet openings. Four infrared sensors placed in the forklift arms, two in each one, send a signal to the component when they

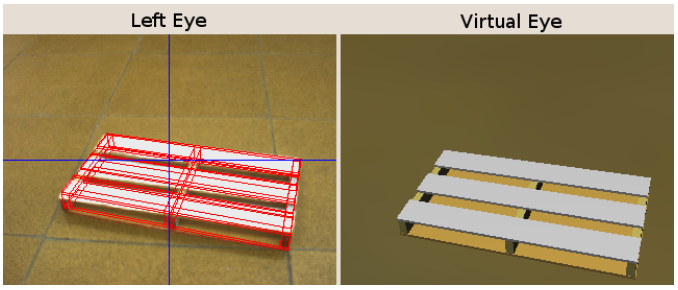


Fig. 8: Final estimation of pallet pose after iterating over x, y and α computing de difference between the real and the synthetic images

are occluded by the pallet. This information triggers the lifting behavior that is performed by the Fork component.

VII. Experimental results

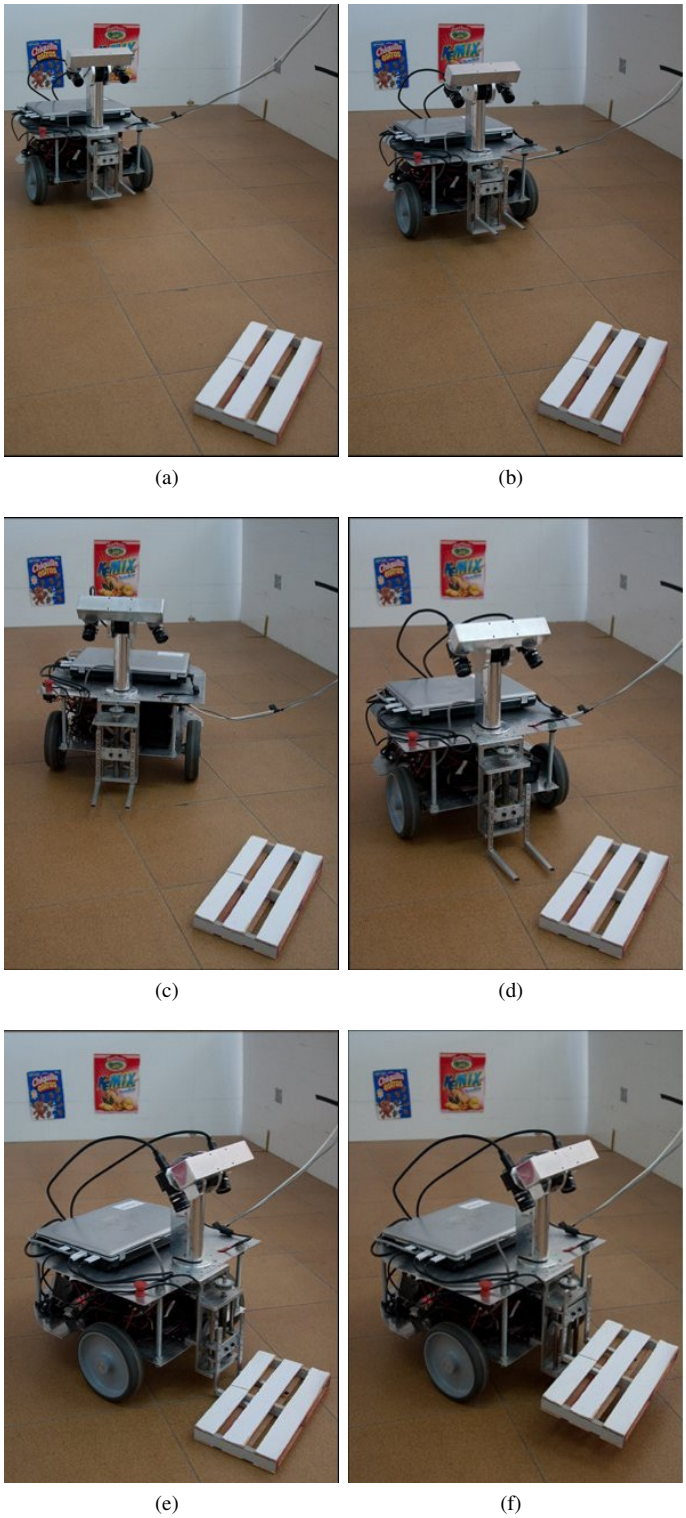
The experiments have been conducted using the RobEx platform [10], [14] (see figure 2). In the experiments the floor does not have a totally homogeneous texture, but its dominant color is different from the surrounding walls. Figure 9 shows a sequence of six pictures in which the robot detects, approaches, recognizes and estimates the pose, maneuvers and, finally, picks and manipulate the the pallet. Figure 10 provides an overhead perspective of the environment.

The experimental procedure used to evaluate the robustness of the system (both the algorithms and the state machine used) is the execution of the task with different values for the variables defining the pallet pose: pallet distance d and pallet orientation a. Distance has been tested for three different values: 100, 150 , 200 and 250 centimeters. Pallet orientation takes the following values: $-\pi/2$, $-\pi/3$, $-\pi/4$, 0, $\pi/4$, $\pi/3$ and $\pi/2$ radians. For each of the possible combinations the task is performed three times. Thus, a total of 84 tests were run. Results are shown in table I. Cells containing the character ✓ express a 100% of success for the whole subset of experiments. By contrast, those cells containing an additional X indicates a certain percentage of failure for the corresponding pallet pose and those marked with a single X express a 100% of failure.

TABLE I: Experimental results

	$-\pi/2$	$-\pi/3$	$-\pi/4$	0	$\pi/4$	$\pi/3$	$\pi/2$
100	✓	✓	✓	✓	✓	✓	✓
150	✓	✓	✓	✓	✓	✓	✓
200	✓	✓	✓ X	X	✓ X	✓	✓
250	X	X	X	X	X	X	X

As expressed in the table, all failures are related to the robot-pallet distance. Thus, when it is far enough the robot can not recognize the pallet. Figure 11 shows the pallet at 200 meters with an orientation of $-\pi/2$ and 0 radians respectively. It can be observed that the size of the projection of the pallet varies depending on the orientation. This problem becomes permanent when the pallet distance increases in such a way that the robot can not recognize it at any orientation. In these



(a) (b) (c) (d) (e) (f)

Fig. 9: Sequence of images showing different states of the pallet manipulation experiment: (a) detection of the pallet; (b) approach the target position; (c) visual tracking of the target; (d) refine pallet pose estimation; (e) final approach; (d) pick up operation.

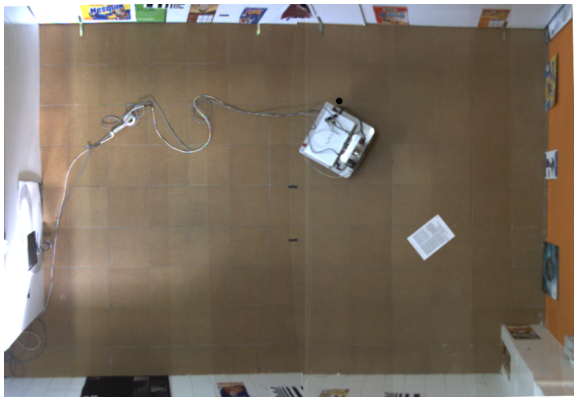


Fig. 10: Overhead perspective of the environment used for the experiments.

situations, any detector would fail since the visual information is not enough to obtain reliable results. We think that this question can only be solved using an active approach that actively drives the robot in search of the pallet. In our detection scheme, it would translate into new states and transitions that would endow the robot with the necessary behaviors to affront and solve the aforementioned situations.

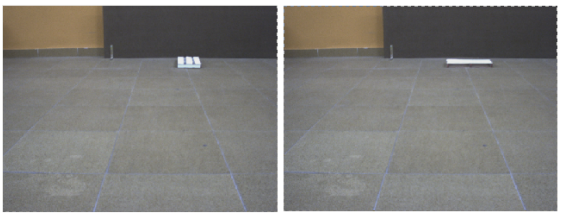


Fig. 11: Perspective of a pallet, with different orientations, situated two meters away from the robot

VIII. Conclusions and Further Work

In this paper we have described an experiment designed to study the problem of sequential integration of behaviors in a real manipulation task conducted by a mobile robot. Instead of using complex and time-consuming algorithms, robust behavior is achieved by iterating motor and perceptual states. In these states, the robot selects, approaches and verifies its target, in such a way that uncertainty is reduced by an iterative global behavior mediated by active perception. Embedding plans in state machines has proved an efficient technique for achieving complex sequential goal in mobile manipulators.

In order to build complex behaviors for the robots, we need to handle complex software systems using state of the art software engineering technologies. These new tools must provide us with the necessary means to ensemble many different concurrent processes, each one contributing to a piece of the overall robot behavior. We have shown how one of these tools, RoboComp, can be further extended to include hierarchical and concurrent state machines providing a necessary level of sequential control. Further work needs to be done to achieve higher levels of robustness and repeatability. Each vision algorithm can be improved individually and the whole state machine can be augmented with new states representing active relations between the robot and its environment. An interesting direction of research would be to apply machine learning techniques to modify on-line some internal parameters of the algorithms and parts of the structure of the state machine.

Acknowledgments

This work has been supported by grant PRI09A037 and GRU09064, from the Ministry of Economy, Trade and Innovation of the Government of Extremadura; by grant TSI-020301-2009-27, for the ACROSS project, funded by the Ministry of Industry, Turism and Commerce of the Spanish government (AVANZA2) and the European FEDER program; and by grant PDT9A044 for the project “Escáner móvil robotizado” funded by the Ministry of Economy, Trade and Innovation of the Extremaduran Government.

References

[1] Bachiller P. *Percepción dinámica del entorno en un robot móvil*. PhD thesis. 2008.

[2] Brooks A., Kaupp T., Makarenko A., Williams S. and Oreback A. *Orca: A Component Model and Repository*. Software Engineering for Experimental Robotics. Springer. 2007.

[3] Brugali, D., and Shakhimardanov A. *Component-Based Robotic Engineering (Part II)*. Robotics and Automation Magazine, IEEE 17, no. 1: 100–112. 2010.

[4] Felzenswalb P. F., Huttenlocher D. P. *Efficient Graph-Based Image Segmentation* International Journal of Computer Vision, Volume 59, Number 2, September 2004

[5] Garibotto G., Masciangelo S., Bassino P. and Ilic M. *Computer vision control of an intelligent forklift truck* In: Proceedings of Conference on Intelligent Transportation Systems. Ieee: 589-594, 1997

[6] Harel D. *Statecharts in the Making: A Personal Account*. Communications of the ACM. Vol 52, issue 3. 2009

[7] Henning M. *A new approach to object-oriented middleware.* IEEE Internet Computing 8, no. 1: 66-75. 2004.

[8] Lecking D., Wulf O. and Wagner B. *Variable pallet pick-up for automatic guided vehicles in industrial environments*. In: IEEE Conference on Emerging chnologies and Factory Automation, 2006. ETFA'06. Citeseer; 2006:11691174.

[9] Maldonado F. J. and Vega J. R. *Diseño y control del sistema de manipulación en un almacén automático.*

[10] Manso L. J., Bustos P. and Bachiller P. *Multi-cue Visual Obstacle Detection for Mobile Robots*. Journal of Physical Agents. Vol 4, issue 1. 2010.

[11] Manso L. J., Bustos P., Bachiller P., Cintas R., Calderita L. and Núñez P. *RoboComp: a Tool-based Robotics Framework*. In Proc. of Int. Conference on Simulation Modeling and Programming for Autonomous Robots. 2010.

[12] Martínez H., Cánovas J.P., Izquierdo M.A., and Skarmeta A.G *I-Fork: a flexible AGV system using topological and grid maps*. In Robotics and Automation, Proceedings. ICRA'03. IEEE International Conference on, 2:2147–2152. 2003.

[13] Martínez J., Romero-Garcés A., Manso L., and Bustos P. *Improving a Robotics Framework with Real-Time and High-Performance Features.* In SIMPAR, Second International COnference on Simulation, Modelling and Programming for Autonomous Robots, 2010.

[14] Mateos J., Sánchez A., Manso L. J., Bachiller P. and Bustos P. *RobEx: an Open-hardware Robotics Platform*. In Proc. of Workshop de Agentes Físicos. 2010.

[15] *Object Management Group: Data Distribution Service for Real-time Systems (DDS), version 1.2* 2007.

[16] OpenSceneGraph home page. *http://http://www.openscenegraph.org*.

[17] Pages J., Armangue X., Salvi J., Freixenet J. and Marti J.. *A computer vision system for autonomous forklift vehicles in industrial environments*. in In Proc. of The 9th Mediterranean Conference on Control and Automation (MEDS). 2001.

[18] Qt Software. *Qt State Machine Framework*. "http://doc.trolltech.com/solutions/4/qtstatemachine/". Last visited 2010.

[19] Quigley M., Gerkey B., Conley K., Faust J., Foote T., Leibs J., Berger E., Wheeler R. and Ng A. *ROS: an open-source Robot Operating System*. In Proc. of Int. Conference ICRA Workshop on Open Source Software. 2009.

[20] RoboLab. *RoboComp Project*. "http://robocomp.sourceforge.net". 2009.

[21] Seelinger M. and Yoder J-D.*Automatic Pallet Engagment by a Vision Guided Forklift*. Proceedings of the 2005 IEEE International Conference on Robotics and Automation. 4068-4073, 2005.

[22] Tamba TA, Hong B, and Hong K. S. *A path following control of an unmanned autonomous forklift,* Intl J. of Control, Automation and Systems. vol:7no1pp113-122, 2009.

[23] Teller S., Walter M.R., Antone M., Correa A., Davis R. and Fletcher L. *A Voice-Commandable Robotic Forklift Working Alongside Humans in Minimally-Prepared Outdoor Environments.* IEEE International Conference on Robotics and Automation, Anchorage, Alaska, USA. 2010

[24] Wasik Z. and Saffiotti A. *A hierarchical behavior-based approach to manipulation tasks*. In Proc. of Int. Conference on Robotics and Automation. 2003.

[25] Wulf O., Lecking D. and Wagner B. *Robust Self-Localization in Industrial Environments based on 3D Ceiling Maps*, International Conference on Intelligent Robots and Systems (IROS), Beijing, China, October 2006

Combining invariant features and localization techniques for visual place classification: successful experiences in the robotVision@ImageCLEF competition

Jesus Martinez-Gomez, Alejando Jimenez-Picazo, Jose A. Gomez and Ismael Garcia-Varea

***Abstract*—In the last decade competitions proved to be a very efficient way of encouraging researchers to advance the state of the art in different research fields in artificial intelligence. In this paper we focus on the optional task of the RobotVision@ImageCLEF competition, which consists of a visual place classification problem where images are not isolated pictures but a sequence of frames captured by a camera mounted on a mobile robot. This fact leads us to deal with this problem not as stand-alone classification problem, but as a problem of self localization in which the robot's main sensor only captures visual information. Thus, we base our proposal on a clever combination of Monte-Carlo-based self-localization methods with optimized versions of scale-invariant feature transformation algorithms for image representation and matching. The goodness of our approach has been validated by being the winners of this task in the 2009 RobotVision@ImageCLEF and 2010 RobotVision ImageCLEF@ICPR competitions.**

***Index Terms*—computer vision, robot localization, place recognition, semantic place representation.**

I. Introduction

In the last decade competitions proved to be a very efficient way of encouraging researchers to advance the state of the art in different research fields in artificial intelligence: KDDCUP (data mining), RoboCup (robotic soccer), Image processing and retrieval (at CLEF or ICPR), Computational Intelligence in Games, DARPA gand and urban challenge (autonomous driving), time series forecasting (at IJCNN), etc.

Image classification is one of the most difficult problems in computer vision research. This problem becomes highly complex when images are captured by a robot's camera within dynamic environments with occlusions and illumination changes. One of the main applications of visual classification is robot localization, but this adds several constraints to the process. The most important one is the processing time, as images need to be processed in real-time.

Visual classification techniques with applications in indoor robot localization (visual place classification) typically use the information retrieved from sequences of training images.

Jesus Martinez-Gomez, Alejando Jimenez-Picazo, Jose A. Gomez and Ismael Garcia-Varea are with the I3A Research Institute, Campus Universitario s/n, 02071, Albacete, Spain. E-mail: jesus_martinez@dsi.uclm.es

E-mail: jesus_martinez@dsi.uclm.es, ajimenez@dsi.uclm.es, Jose.Gamez@uclm.es, Ismael.Garcia@uclm.es

This work was supported by the Spanish "Junta de Comunidades de Castilla-La Mancha" under PCI08-0048-8577 and PBI-0210-7127 Projects and FEDER funds. The support is gratefully acknowledged.

The approaches presented here carry out classification by using techniques such as Scale-Invariant Feature Transform (SIFT) [10], RANSAC [4], and the well-known Monte Carlo localization method [3].

SIFT is used to extract invariant features from images and also to perform a preliminary matching. RANSAC provides us with a useful technique to improve the first matching obtained with SIFT by discarding invalid correspondences. The Monte Carlo localization method was used to take advantage of the similarity between consecutive frames and its relationship with the robot's location.

All experiments were carried out following the proposed procedure, using the appropriate training sequences and a final test sequence. Our proposals were evaluated for the two proposed tasks: obligatory (classification must be performed separately for each test image) and optional (the algorithm can exploit the continuity of the sequences), which is a more realistic localization task.

The rest of the article is organized as follows: a brief introduction of the task is given in Section II; Section III provides an overview of the main proposals from other participant research groups; all the techniques that have been used to develop our proposals are explained in Section IV; Section V presents a full description of the proposals for 2009 and 2010; Section VI offers official results to show the performance of the proposed solutions and conclusions and future work are discussed in Section VII.

II. Robot Vision Task

The RobotVision@ImageCLEF task [18], [16] addresses the problem of visual place classification. Participants are asked to classify rooms and semantic areas on the basis of image sequences captured by a camera. This camera is mounted on a mobile robot that moves within an office environment under varying illumination conditions. Participant systems should be able to answer the question "where are you?" when presented with test sequences containing images acquired in the previously observed part of the environment (using different viewpoints and acquired under different conditions) or in additional rooms that were not imaged in the training stage (these rooms should be labelled with the new room category).

There are two subtasks: obligatory and optional. For the first subtask, the classification has to be performed without taking

into account the order and the relationship between frames. The second subtask is optional but is more related to robot localization. Within this subtask, the classification algorithms can take advantage of the continuity of the sequence of test frames.

The competition starts with the release of annotated training and validation image sequences from a database (*COLD-Stockholm* [15] in 2010 and *KHL-IDOL2* [12] in 2009). These data sets could be used to train different systems and to evaluate their goodness by using the validation data set.

The final test image sequence is released later and contains images acquired in the same environment but including additional rooms not imaged previously. In addition, lighting conditions vary from the training to the validation and test sequences.

A. Data Set

There are three main data sets: training, validation and test. Training sequences are used to generate the classification systems and validation sets can be used to evaluate the performance of preliminary proposals. Training and validation frames are labelled with the room and, therefore, some preliminary scores can be obtained. Test frames are not labelled and the final score for a participant's proposal can only be obtained by submitting the results to the organizer's website.

The number of frames in the different data sets can be observed in Table I.

Sequence	Frames	Rooms	Lighting Conditions
2009 Edition			
Training 1	1034	5	Night
Training 2	915	5	Cloudy
Training 3	950	5	Sunny
Validation 1	952	5	Night
Validation 2	928	5	Cloudy
Validation 3	909	5	Sunny
Test	1600	6	Cloudy
2010 Edition			
Training Easy	8149	9	Cloudy
Training Hard	4535	9	Cloudy
Validation	4783	9	Night
Test	5102	12	Night

TABLE I
ROBOTVISION DATA SET INFORMATION

The main difference between the *KHL-IDOL2* (2009) and the *COLD-Stockholm* (2010) database is the information provided with the training image sequences. Images from the 2009 database were labelled with the room where the frame was taken and the complete $< x, y, \theta >$ of the robot's pose. The map of the environment was also provided. With respect to the 2010 database, the information related to the robot's pose was removed and therefore only the room in which the frame was captured was provided.

All classes used to label training and validation frames are semantic terms extracted from indoor environments such as "corridor" or "printer area". Figure 1 shows several exemple images from the cloudy training sequence of the 2009 CLEF

Fig. 1. Examples of images from the CLEF 2009 database.

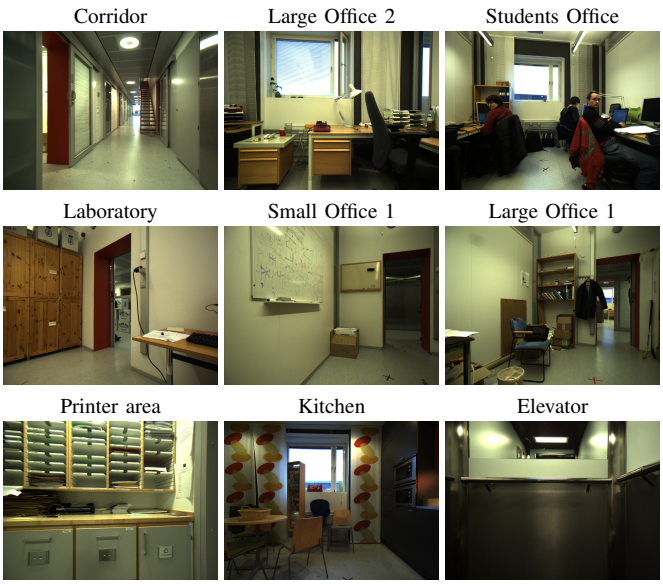


Fig. 2. Examples of images from the CLEF 2010@ICPR database.

edition and an image from the test sequence (bottom-right). For the first edition, only 5 different semantic categories were used for training and just one new room not imaged previously (toilet) was added for the test sequence.

The number of semantic categories used for the task was drastically increased for the 2010@ICPR edition (from 5 to 9). Three categories were kept from the 2009@ImageCLEF edition (corridor, printer area and kitchen), and six new categories were added (elevator, laboratory) or modified from the previous edition (student office, large office 1, small office and large office 2).

B. Performance Evaluation

Robot Vision task organizers use several rules to calculate the final score for each submitted run:

- $+1.0$ points for each correctly classified image (correct detection of an unknown room is treated in the same way as correct classification).
- -0.5 points for each misclassified image.
- $+0.0$ points for each image that was not classified (the algorithm refrained from the decision).

The final score will be a sum of points obtained for the test sequence. The test sequence consists of a single set for the 2009 edition and two sets (easy and hard) for the 2010@ICPR one. This score is used as a measure of the overall performance of the user proposals.

III. Other Group's Proposals

Seven groups registered and submitted at least one run for the 2009 edition of the Robot Vision task [18], with a total of 27 submitted runs. The number of participant groups increased to 8 for the 2010@ICPR edition [17], and the number of submitted runs also rose(29).

A. Glasgow - 2009

The multimedia Information Retrieval Group from the University of Glasgow achieved second position for the optional task using point matching technologies with rule-based decision techniques. Point extraction was performed by employing the Harris corner detector [8] and an illumination filter called Retinex [21] was applied to increase the number of interest points that can be extracted. The main drawback of this proposal is the hard assumption that all frames will contain similar content and geometric information. This assumption will be wrong with hard variations for the viewpoint the frames were taken from. The Harris corner detector will fail when facing changing environments and anoother invariant point extractor should be selected.

B. LSIS - 2009

Another interesting solution for the 2009 task was that proposed by the *Laboratoire des Sciences de l'Information et des Systemes (LSIS), La Garde-France* [7]. This proposal constructs a Support Vector Machine (SVM) using two different types of features: the new Profile Entropy Features (PEF) and the generic Descriptor of Fourier (DF) [20]. PEF is proposed by the LSIS group and combines RGB colour and texture, yielding one hundred dimensions. The SVM classifier is generated by using 19 sub-classes created from the 5 original rooms. A total of 19 different binary SVM classifiers were generated and their outputs were combined to get the final decision. The SVM model generated on PEF achieved 4th position for the obligatory task but needed much less training time than the other systems (around 50 times less). The SVM model using only the Descriptor of Fourier obtained a negative score, indicating the importance of the features extracted (even using an efficient and robust classifier such as SVM).

C. CVG - 2010@ICPR

The winner of the obligatory task for the 2010@ICPR edition was the Computer Vision Group, from the ETH Zurich. This group presented a novel and interesting proposal [6] based on the use of visual words [5] computed from SIFT features. Each test frame is classified using a similarity ranking performed using the visual words, already extracted from the training frames. An additional and useful constraint is considered, based on a geometric verification. This verification uses depth information and is computed using the planar 3-pt algorithm [14], which assumes that the robot is moving in a plane.

IV. Techniques

This section gives a brief introduction to all different techniques that were used in our proposal for the Robot Vision challenge. They all are well-known techniques that have proved their goodness in different scenarios. Monte Carlo was the selected localization method, using SIFT for the image processing (needed for the update phase). RANSAC was used to improve SIFT matching by discarding outliers.

A. SIFT

In order to perform a correct comparison between (at least) two frames representing the same scenario, appropriate features should be used. These features have to be invariant to changes in the position where the frames were captured. Frames to compare can be acquired over different lighting conditions but such changes should not affect the quality of the comparison. In addition to this, some elements in a dynamic environment can be removed, added or replaced.

All these factors mean that it makes no sense to use classical computer vision techniques based on the Hough transform [22](such as line or square recognition) to solve the problem of robot localization using visual information. These techniques rely on specific scenario elements which (in the problem we are dealing with) are liable to be removed or replaced.

The most popular technique for extracting relevant features from images is the Scale-Invariant Feature Transform (SIFT) [11]. The main idea of this algorithm is to apply different transformations and study the points of the image which are invariant under these transformations. These extracted points can be used to perform object recognition by carrying out a matching between images representing the same object or scenario.

Features extracted are invariant to image scale and rotation, and they are also robust to noise and changes in viewpoint. An important characteristic of systems developed to perform object recognition using SIFT is that they are robust to partial object occlusion. On the other hand, the algorithm presents a high computational cost that is an important drawback for real-time systems.

In order to deal with the considerable processing time of the algorithm, an implementation of the algorithm over the graphics processor unit (GPU) was considered. The selected implementation (named "SiftGPU"[1]) speeds up the process and allows us to perform a higher number of experiments.

B. RANSAC

Random Sample Consensus [4] (RANSAC) is a non-deterministic iterative method for estimating a mathematical model from a dataset. The idea behind RANSAC is to find a significant group of points which are all consistent with a specific model and reject the remaining points as outliers.

[1]http://www.cs.unc.edu/ ccwu/siftgpu/

In order to achieve this goal, RANSAC iteratively estimates models using a random subset of points and evaluates these models with the complete dataset. The algorithm ends when certain constraints (such as the obtained model accurately fitting the data) are overcome or after a maximum number of iterations is reached. The RANSAC paradigm is formally stated as follows:

- Given a model that requires a maximum of n data points to instantiate its free parameters, and a set of data points P such that the number of P is greater than n $[\sharp(P) \geq n]$, randomly select a subset $S1$ of n data points from P and instantiate the model. Use the instantiated model $M1$ to determine the subset $S1^*$ of points in P that are within some error tolerance of $M1$. The set $S1^*$ is called the consensus set of $S1$.
  - If $\sharp(S1^*)$ is greater than some threshold t, which is a function of the estimate of the number of gross errors in P, use $S1^*$ to compute a new model $M1^*$.
  - If $(S1^*)$ is less than t, randomly select a new subset $S2$ and repeat the above process. If, after some predetermined number of trials, no consensus set with t or more members has been found, either solve the model with the largest consensus set found, or terminate in failure.

There are three free parameters in RANSAC:

- The error tolerance that defines whether or not a point is compatible with a model;
- The number of subsets that defines the number of iterations;
- The threshold t that defines the number of compatible points used to decide if the right model has been found.

Our proposal for the 2010@ICPR edition of the Robot Vision task uses RANSAC to improve the initial matching obtained with SIFT. Such initial matching obtains a high number of outliers that do not fit the real correspondence between two candidate images. Fig.3 illustrates the result of a matching between the invariant points extracted from two images and how the outliers (red lines) are discarded using RANSAC.

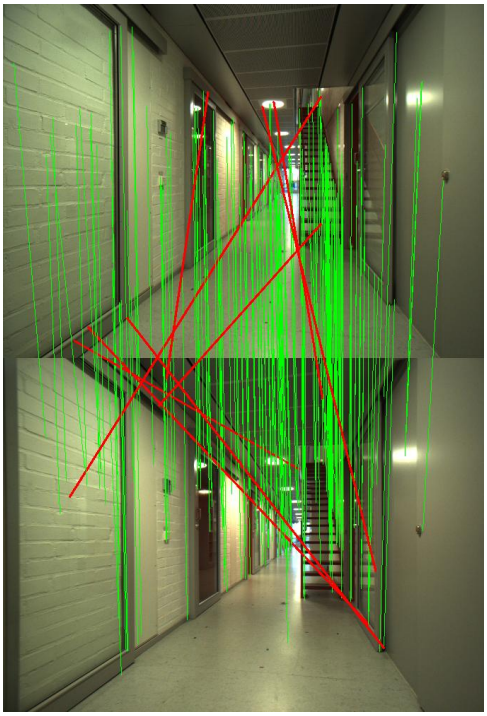


Fig. 3. SIFT matching between two images where outliers (red lines) are discarded.

C. Monte Carlo Localization Method

The Monte Carlo Localization method [3] (denoted as MCL-method) is a method to determine the position of a robot given a map of its environment. This method is basically an implementation of the particle filter applied to robot localization and has been proposed for use within indoor dynamic environments. The MCL-method keeps information about different environment locations and allows us to represent uncertainty about the robot's localization.

The main alternative to the use of the MCL-method for developing a robot localization algorithm are (Extended) Kalman filters [13]. This alternative method was not considered for our 2009 edition proposal because it only keeps information about a single robot's location and it is necessary to know the robot's initial pose. Moreover, this method presents problems when the uncertainty about the pose increases significantly.

Using a particle filter approach, the density of the localization is represented by a set of N random samples drawn from it. Each one of these samples or particles represents a robot's pose $< x, y, \theta >$. Using the set of samples ($S_k = \{s_k^i; i = 1..N\}$) we can approximately reconstruct the density and estimate the robot's localization.

At each time-step k the MCL-method recursively computes the set of particles that is drawn from the information sensed at k. In the scope of ImageCLEF RobotVision, the information sensed from the environment is that retrieved from the image captured at time-step k.

The iterative MCL-method proceeds in two phases (where $\mathbf{u}_k$ denotes the movement order sent to the robot at time-step k and $\mathbf{z}_k$ represents the information sensed from the environment at k):

Prediction Phase This phase starts from a set of particles S_{k-1} computed in the previous iteration. The method applies the motion model to each particle s_{k-1}^i by sampling from the density $p(\mathbf{x}_k|s_{k-1}^i, \mathbf{u}_{k-1})$:

(i) for each particle s_{k-1}^i: draw one sample $s_k'^i$ from $p(\mathbf{x}_k|s_{k-1}^i, \mathbf{u}_{k-1})$

We obtain a preliminary set of particles at time k S'_k, when we have not yet incorporated any sensor measurement.

Update Phase The second phase takes into account the measurement $\mathbf{z}_k$, and weights each of the samples in S'_k by the weight $m_k^i = p(\mathbf{z}_k|s_k'^i)$, i.e. the likelihood of $s_k'^i$ given $\mathbf{z}_k$.

We then obtain S_k by resampling from this weighted set:

(ii) for j=1..N: draw one S_k sample s_k^j from $\{s_k'^i, m_k^i\}$

The resampling selects, with higher probability, samples $s_k'^i$ that have a high likelihood associated with them, and in doing so a new set S_k is obtained that approximates a random sample from $p(\mathbf{x}_k|Z^k)$.

After the update phase, steps (i) and (ii) are repeated iteratively. The filter is initialized at time $k = 0$ with a random sample $S_0 = \{s_0^i\}$ from the prior $p(x_0)$.

While the method is being applied, the best particles should be duplicated and the worst particles should disappear from the set of samples. After some iterations, the algorithm should converge with all particles around the real robot's pose. This situation is shown in Fig. 4, where blue circles represent the particles for three time-steps. The size of the circumferences represents the weight of the particles $m_k^i = p(\mathbf{z}_k|s_k'^i)$. It can be observed how most of the low-weight particles from the first iteration disappear in the third one. On the other hand, high-weighted particles are duplicated.

V. Proposal

All our work is focused on the optional task because we want to develop robotic systems that can feasibly be used in the real world. For the obligatory task (mandatory) we presented the result of applying a subset of our processing. This subset was used to define the image processing necessary to perform the optional task. The temporal and spatial relationship between consecutive frames was not considered for the obligatory task, where the sequence of test frames could differ from the original one.

A. 2009 Proposal

Our proposal for the 2009 RobotVision@ImageCLEF competition was to develop a localization method based on the use of particle filters. We used an MCL method and (as was mentioned in Section IV-C) we needed a motion model (for the prediction phase) and an image processing algorithm to sense the information from the environment (and use this information within the update phase).

1) Image Processing: A preliminary pre-processing step was applied to extract all the available information from the training sequences. This process stored (for each training frame) the complete pose of the robot $< x, y, \theta >$, labelled with the correctly classified room C_i and the set of m SIFT points extracted from that frame $[P_{1...}P_m]$. This feature extraction can be performed offline.

Once we had extracted all training information, we defined the way to classify each test frame. This classification was performed based on the similarity computed between each test frame and all training frames. For each test frame, we obtained a similarity ranking and classified that test frame with the label of the k most similar training frames. This process is similar to the k-nearest neighbor algorithm [2].

The main problem of this approach is the processing time necessary to perform a SIFT matching ($\cong 0.18sec$ per image). Another important drawback is the problem of invariant feature extraction (by using SIFT) when facing lighting changes. Extracted SIFT features heavily depend on lighting conditions under the frame was taken. Therefore, matchings between two frames taken from the same viewpoint (and room) but acquired under different lighting conditions will not obtain a high similarity score.

In addition to this slow processing based on similarity, we added an extra step based on the detection of natural environment landmarks. Robot localization by using environment landmark detection is commonly used in the scope of the RoboCup [1], [19]. We applied a Hough transform[22] to study the distribution of the lines and squares obtained. Environment landmarks such as the corridor ceiling or the external door located in the printer area were selected as unchanging sections. Some examples of these detections can be observed in Fig. 5, where the corridor class is detected. A corridor situation is defined as a frame with a large number of vertical lines (higher than 20) and two symmetric lines starting from the top of the image and representing the ceiling. Vertical lines will correspond with to doors, wardrobe and other elements likely to appear within a corridor. The execution time of this extra step on a current computer was lower than 0.005 seconds.

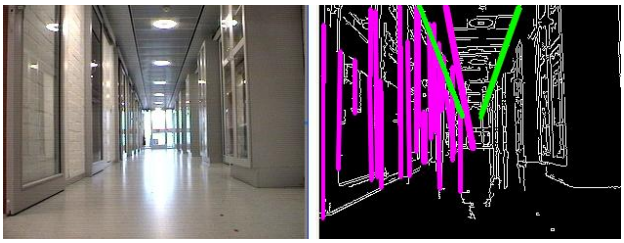


Fig. 5. Corridor detection. Purple lines are candidates and green lines are the correct ones.

Natural landmarks should be selected carefully, taking into account that these landmarks should be highly discriminative elements that are easy to detect under changing lighting conditions. All selected landmarks should have specific characteristics that cannot appear in new unknown rooms. We only used the corridor ceiling (larger than all the other room ceilings) and a window combination for the Printer Area.

Fig. 6. Printer Area detection. Purple squares are candidates and green squares are the correct ones.

2) Localization Algorithm: In order to reduce the number of comparisons and to take advantage of the similarity between consecutive test frames, we added a particle filter. It

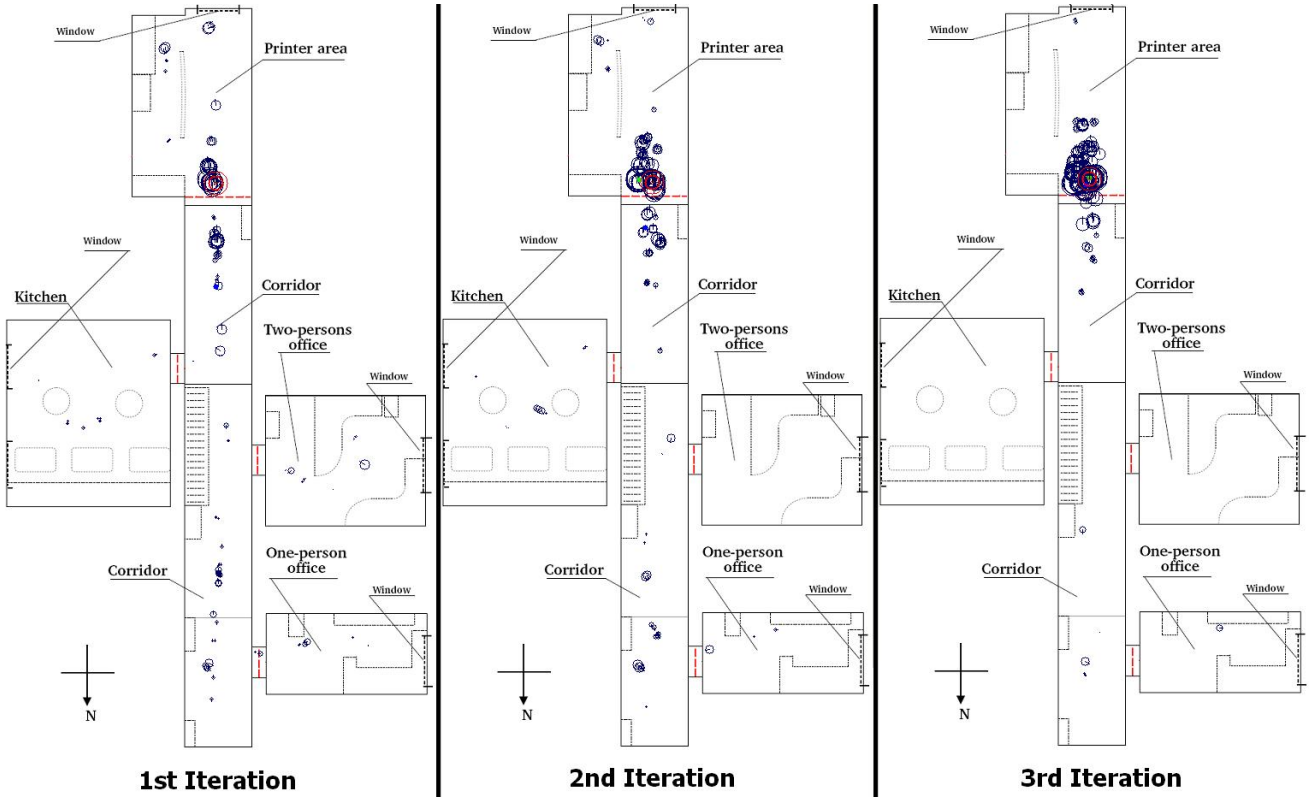


Fig. 4. Particle convergence after three iterations of the MCL method, where circles represent particles and the size of circles is used for a particle's weight.

was necessary to define a particle representation and how the prediction and update phases are iteratively applied.

Particle Representation: Each particle is represented by three numerical values (representing the pose of the robot): x and y are used for position and θ for orientation. These parameters have continuous numeric values. The limits for x and y values are the boundary of the environment. In this case, values for the x component are between 0 and 1500 centimetres. Bounds for the y component are 0 and 2500 centimetres.

Prediction Phase: This step applies a movement model without knowing the movement order sent to the robot between test frames. We assume that the robot movement will be similar to that performed during the training (training frames were acquired while the robot was moving). The average robot velocity was estimated from the difference between the poses of the training sequence. We obtain the average linear and angular velocity, which are defined in centimetres (and degrees) per frame.

We add white noise in this step to represent some uncertainty in the obtained movement. This noise applies a random variation based on the particle's weight, obtaining higher variations for worst particles. Using this approach, best candidates will obtain minor changes.

Update Phase: This phase obtains the weight for each particle using the information extracted from the test frame to classify and from the training sequence. Each particle is weighted by matching the SIFT points of the test frame with those extracted from the training frame taken from the pose the particle represents. We only have SIFT points extracted from a discrete number of poses in the environment and so we have to select the nearest training frame to the particle's pose.

Once we have weighted all particles, the robot's position is estimated using the particle information: x_i, y_i and weight w_i. The average robot position is obtained as a weighted sum, taking into account the particle weight:

$$\bar{x} = \sum_{i=0} x_i \times w_i, \quad \bar{y} = \sum_{i=0} y_i \times w_i$$

Each test frame will be labelled with the room belonging to the average robot position, or not labelled if this position is between two rooms.

We noticed that some test frames were not represented by any training frame, failing all the available SIFT matchings. The main consequence of this situation is that the weight of all the particles decreases continuously and they are spread over the environment. When facing this situation and finding new false positives, particles will converge to wrong areas and the robot's location will not be correctly obtained.

In order to avoid this situation and to escape from wrong convergences, we added an extra step to the original process (basic MCL method). This step performs a population initialization when the best particle's weight is below a certain threshold (similar to approaches to escape from local optima).

This first modification improved the system behaviour but the process became unstable and the algorithm presented problems to achieve convergence.

At this point, we decided to restrict the negative effect of population initializations, performing them over restricted environment areas. We assumed that it is possible to detect when our algorithm converges (all particles are close together) and that this event represents a perfect pose estimation (particles represent robot's current pose).

Future population initializations will be carried out over a restricted area. This area is defined as a circumference with radius r and centred at point $< x_a, y_a >$. This radius r is a function of the instability level (obtained with the best particle's weight for the last n iterations) and the value of point $< x_a, y_a >$ that was obtained under the last stable situation representing the robot's most reliable pose.

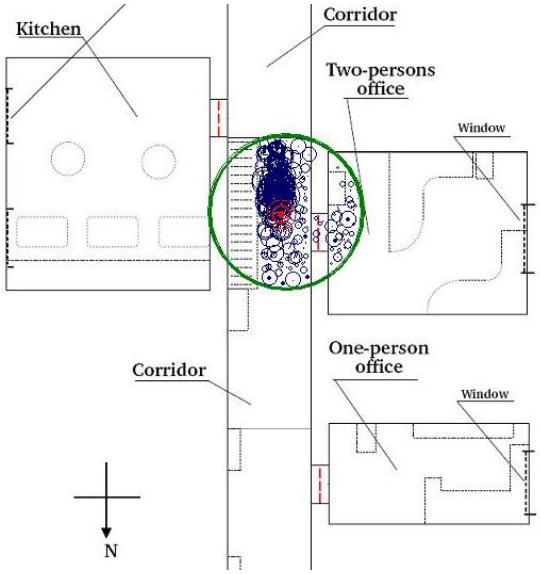


Fig. 7. Population re-initialization over a restricted area represented by the green circle.

A re-initialization is shown in fig 7, where a new population is created by spreading particles over the area covered by the green circle.

The stability of the algorithm could be estimated by studying the variation in the pose of the particle. The process will be stable when the variation obtained for the x and y components of the last n pose estimations is sufficiently small (all particles are spread over a small portion of the environment). The next step was to define a state for the algorithm based on its stability: When the algorithm is stable(i), no population initializations are performed. Otherwise, the initialization depends on the instability level. If the algorithm has been stable for the last few frames and it becomes unstable(ii), initialization is performed over a restricted area (a circumference centred at the most reliable robot position, obtained from previous iterations with a stable process). The size of this area depends on the instability level. If the algorithm has been unstable for the last few frames(iii) and a new initialization has to be applied, all the particles will be spread over the whole environment.

The whole process is shown in Fig. 8, where these three situations (and the action associated to each case) are shown.

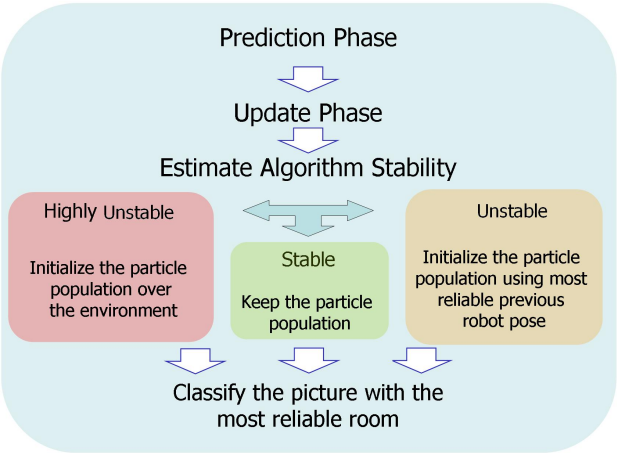


Fig. 8. General Processing Scheme.

B. 2010 Proposal

In this edition, the training information provided by the RobotVision@ICPR organizers prevented us from applying localization principles. Training frames were just labelled with the room where they were taken and the map of the environment was not provided. The task was reduced to a visual place classification problem and we decided to improve the classification techniques used for the 2009 edition.

First of all we adopted the GPU implementation of the SIFT algorithm (known as SiftGPU). This implementation speeded up the process for the system training and allowed us to perform a higher number of experiments and tests.

Training frame pre-processing. In order to reduce the amount of information to work with and to discard redundant frames, we added training sequence pre-processing. All training frames were converted to greyscale. After this conversion, we computed the difference between two frames as the absolute difference for the colour, pixel by pixel (an example of the difference is shown in Fig. 9). Each frame whose difference between it and the last non-removed frame was lower than a certain threshold (0.05% in all our experiments) was removed from the training sequence.

Once we had rejected all redundant information, we selected a subset of the most representative training frames. This process was performed as a k-medoids [9] clustering algorithm and the similarity between two frames was computed by matching the SIFT points extracted from them. An example of the training sequence pre-processing is illustrated in Fig. 10, where the first row of frames represents the complete training sequence, the second row is used to show which frames are removed (because they are redundant). Finally, the most representative frames are shown in the third row in Fig. 10.

Frame Classification. The complete process for classifying a test frame consists of three steps. First, SIFT points are extracted from the current frame. Second, we compute the

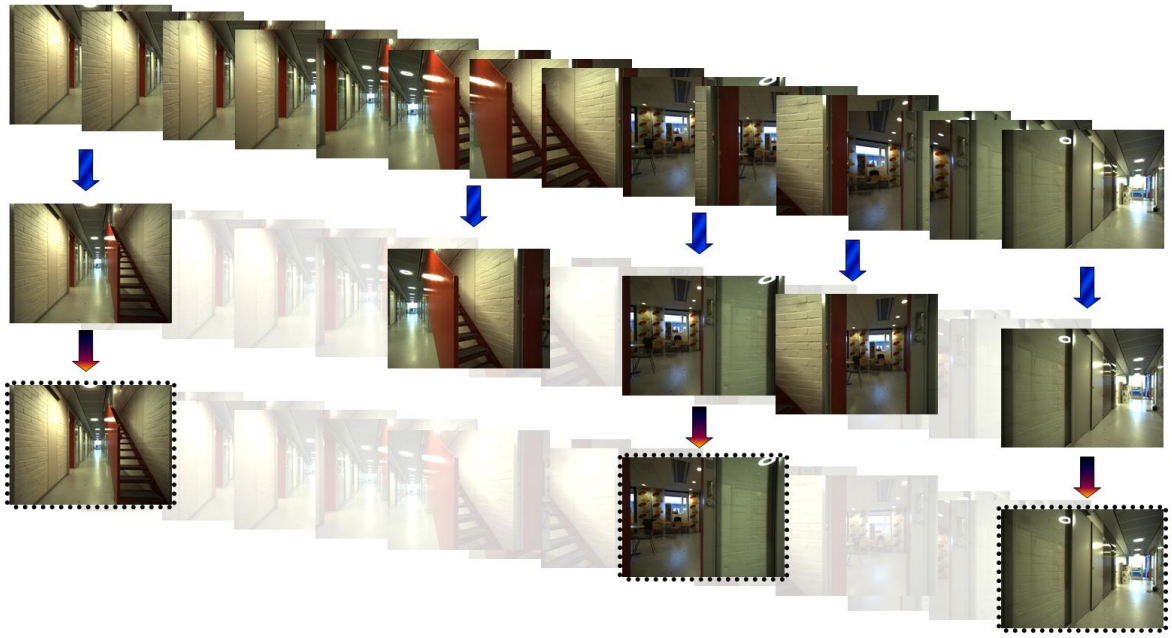


Fig. 10. An example of sequence pre-processing where redundant frames are discarded and best candidates are selected as representative.

Fig. 9. Difference computed between two frames

similarity between the current frame and the training frames by means of the percentage of matching points (SIFT + RANSAC). Finally, these similarities are used to classify the test frame.

The similarity value between a test frame and a training frame is obtained using SIFT matching and RANSAC. After all the SIFT matching points are obtained, RANSAC is applied to discard the outliers. The percentage of common points between both frames is stored as the similarity value.

Each test frame can be classified as a room, marked as unknown or not classified. A ranking with the best n similarity values and its associated rooms is obtained. We compute the sum of the similarity values separately for the different rooms in the ranking. The test frame will be classified as the room with the highest value when this value clearly exceeds all the other ranking rooms, otherwise it will not be classified. Unknown is used to denote a test frame acquired in a room not included in the training rooms and will be used when the maximum similarity value is below a certain threshold (0.05 in the experiments). A complete classification process where the test frame is matched with all the selected training frames can be observed in Fig. 11. In this case the test frame should be labelled as a corridor.

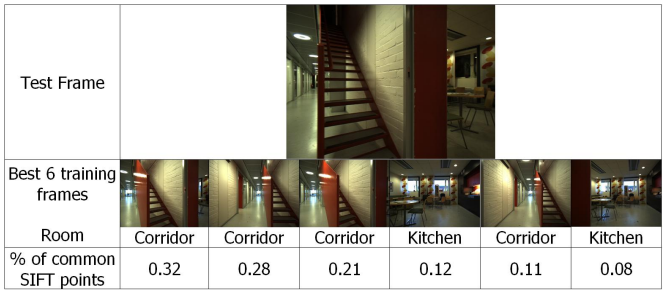


Fig. 11. Test frame classification using a similarity ranking of 6 training frames

For the optional task (where the algorithm can exploit the continuity of the sequence) we took into account the test frame we are going to classify and the last 4 test frames already classified. Test frames initially labelled as not classified were labelled as the room used to classify the last 4 frames when this room was the same. In addition to these considerations, we avoided passing from one room to another without using the corridor (the last test frame was labelled as not classified)

VI. Results

Our proposals were evaluated by the RobotVision@ImageCLEF organizers. The performance of each algorithm was evaluated using a score that is computed as follows: for each correctly classified frame the score is updated by $+1.0$, for each misclassified one the score is

reduced by -0.5. A non-classified frame does not alter the score.

Each run consisted of submitting the results for the test sequence, using the information provided by the training sequence.

Participants in 2009 were asked to use a specific training sequence acquired under night lighting conditions, but in 2010, two training sequences had to be used: easy and hard. The amount of information to work with was higher when using the easy training sequence. For the easy training set, the robot was driven through the environment following a similar path as for the test sequence. The direction followed by the robot in the hard training set was the opposite to that used for the easy one.

A. 2009 Preliminary Experiments

Before submitting any result file to the website, we performed a set of preliminary experiments using all the available training sequences. We generated three different classifiers, which were trained using the training sequences acquired under the three lighting conditions (cloudy, night and sunny). Each one of these classifiers was tested using three validation sequences (one for each lighting condition). These three validation sequences were selected from the set of validation sequences provided by the organization (*KHL-IDOL2* database). For each combination of training and validation, we applied our processing for the obligatory and optional task, storing the number of correctly classified, not classified and misclassified frames. A detailed view of the results obtained is presented in Table II.

From data collected in Table II (Obligatory task), we can conclude that there is a high dependency of the system on lighting changes. This dependency makes the algorithm obtain worse results when training the classifier with different lighting conditions than those used for testing. This situation is clearly revealed for the obligatory task, where the best score was always obtained using the same lighting conditions for training and testing.

This dependency is not so important when using the additional processing included in the optional task. For all the test sequences, (at least) 60% of frames were correctly classified. It can be noticed that scores obtained for the different test sequences are not so strongly dependent on training illumination conditions as those obtained for the obligatory task. By performing a comparison between these results and those obtained for the obligatory track, we can state that the MCL-method notably improved the results for the optional task.

B. 2009 Competition

All internal thresholds were tuned using the proposed validation sequences: the process was estimated as unstable when the best particle's weight was below 0.05 or when average variation for the x (or y) component was above 2 (or 3) meters. The maximum radius of the circumference used to initialize populations was 3 meters, and its value was calculated with $3 \cdot \bar{m}$ (where $\bar{m}$ denotes the average weight for all the population particles).

Different runs were submitted to the website and the best score obtained was 916.5 for the optional task and 511.0 for the obligatory one. Table III shows the complete results obtained for the 2009 competition, with the number of correctly classified (C), misclassified (M) and not classified frames (NC). We achieved 5th position for the obligatory task (the best score of 793.0 was obtained by the Idiap Research Institute from Switzerland).

Obligatory task - 5th position	
Score	**511.0**
Correctly Classified	676
Missclasified	330
Not Classified	684
Optional task - 1st position	
Score	**916.5**
Correctly Classified	1072
Missclasified	311
Not Classified	217

TABLE III
RESULTS FOR THE 2009 COMPETITION

Thanks to the use of a particle-filter algorithm, the final score for the optional task improved upon the results obtained for the obligatory one (from 511.0 to 916.5). The final score for the optional task was the highest one of all submitted runs from all participants, and the SIMD group of the University of Castilla-La Mancha was the winner for the optional task.

C. 2010 Competition

For the 2010 competition, each run consisted of submitting the results for the two training sets: easy and hard. The complete results are shown in Table IV, and as can be observed, our run achieved 3rd place for the obligatory task, for which 8 different research groups submitted results.

Obligatory task - 3rd position			
	Total	Easy	Hard
Score	**3372.5**	2000.0	1372.5
Correctly Classified	3886	2180	1706.0
Missclasified	1027	360	667
Not Classified	189	11	178
Optional task - 1st position			
	Total	Easy	Hard
Score	**3881.0**	2230.5	1650.5
Correctly Classified	4224	2332	1892
Missclasified	686	203	483
Not Classified	192	16	176

TABLE IV
RESULTS FOR THE 2010 COMPETITION

Our proposal again obtained 1st position for the optional task, for which 4 different groups submitted results.

VII. CONCLUSIONS AND FUTURE WORK

The current article presents all the proposals of our group (SIMD) for the first two editions of the RobotVision task,

TRAINING SEQUENCE	VALIDATION SEQUENCE											
	Obligatory task											
	Night (952 frames)				Cloudy (928 frames)				Sunny (909 frames)			
	Score	C	M	NC	Score	C	M	NC	Score	C	M	NC
Night (1034 fr.)	**531**	643	224	85	285	433	265	230	265.5	421	311	177
Cloudy (915 fr.)	270.5	426	311	215	**404.5**	538	267	123	420.5	534	227	148
Sunny (950 fr.)	285.5	435	299	218	358.5	457	197	247	**509**	615	212	82
	Optional task											
	Night (952 frames)				Cloudy (928 frames)				Sunny (909 frames)			
	Score	C	M	NC	Score	C	M	NC	Score	C	M	NC
Night (1034 fr.)	**837.5**	861	47	44	534.0	635	202	91	476.5	560	167	182
Cloudy (915 fr.)	600.5	695	189	68	680.5	748	135	45	733.5	774	81	54
Sunny (950 fr.)	725.0	791	132	29	**701.0**	769	136	23	**798.5**	823	49	37

TABLE II

SCORE, NUMBER OF CORRECT, INCORRECT AND NON CLASSIFIED FRAMES FOR DIFFERENT COMBINATIONS OF TRAINING AND VALIDATION FRAME SEQUENCES. PRELIMINARY RESULTS OBTAINED USING SEQUENCES FROM THE 2009 COMPETITION.

within the CLEF campaign. According to the results obtained for the optional task in both editions, visual place classification is clearly helped by the aplication of robotic localization principles.

Two well-known techniques, namely SIFT and RANSAC, were used to perform the matching or comparison between images. The disadvantage of SIFT's high processing time was solved by using a new implementation over the GPU. RANSAC has been properly used to discard the outliers and to improve the matching between SIFT points.

The percentage of common points between the test frame and the best training frame can be used to weight the classification. This information could be highly useful for estimating classification quality or performance.

For future work, we aim to develop an indoor localization system capable of being trained automatically. This system will use the information extracted from vision cameras and that obtained from robot actuators. This future line of research is related to the integration of both sources of information: visual and odometrical.

ACKNOWLEDGEMENTS

The authors acknowledge the financial support provided by the Spanish "Junta de Comunidades de Castilla-La Mancha (Consejería de Educación y Ciencia)" under PCI08-0048-8577 and PBI-0210-7127 Projects and FEDER funds.

REFERENCES

[1] G. Adorni, S. Cagnoni, and M. Mordonini. Landmark-based robot self-localization: a case study for the robocup goal-keeper. In *Information Intelligence and Systems, 1999. Proceedings. 1999 International Conference on*, pages 164–171, 1999.

[2] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 2002.

[3] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proc. of the IEEE InternationalConference on Robotics*, 1999.

[4] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. 1981.

[5] F. Fraundorfer, C. Wu, J. Frahm, and M. Pollefeys. Visual word based location recognition in 3d models using distance augmented weighting. In *Fourth International Symposium on 3D Data Processing, Visualization and Transmission*, volume 2, 2008.

[6] F. Fraundorfer, C. Wu, and M. Pollefeys. Combining monocular and stereo cues for mobile robot localization using visual words. In *International Conference on Pattern Recognition (in press)*, 2010.

[7] H. GLOTIN, Z. ZHAO, and E. DUMONT. Fast LSIS Profile Entropy Features for Robot Visual Self-Localization. *Working Notes of CLEF*, 2009.

[8] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[9] L. Kaufman, P. ROUSSEEUW, D. D. o. M. Technische Hogeschool, and Informatics. Clustering by means of medoids. Technical report, Technische Hogeschool, Delft(Netherlands). Dept. of Mathematics and Informatics., 1987.

[10] D. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, volume 2, pages 1150–1157. Corfu, Greece, 1999.

[11] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[12] J. Luo, A. Pronobis, B. Caputo, and P. Jensfelt. The KTH-IDOL2 database. Technical Report CVAP304, Kungliga Tekniska Hoegskolan, CVAP/CAS, October 2006.

[13] R. Negenborn. Robot localization and kalman filters. 2003.

[14] D. Ortín and J. Montiel. Indoor robot motion based on monocular images. *Robotica*, 19(03):331–342, 2001.

[15] A. Pronobis and B. Caputo. COLD: The CoSy Localization Database. *The International Journal of Robotics Research*, 28(5):588, 2009.

[16] A. Pronobis and B. Caputo. The robot vision task. In W. B. Croft, H. Müller, P. Clough, T. Deselaers, and B. Caputo, editors, *ImageCLEF*, volume 32 of *The Information Retrieval Series*, pages 185–198. Springer Berlin Heidelberg, 2010.

[17] A. Pronobis, M. Fornoni, H. Christensen, and B. Caputo. The Robot Vision Track at ImageCLEF 2010. 2010.

[18] A. Pronobis, L. Xing, and B. Caputo. Overview of the clef 2009 robot vision track. In C. Peters, B. Caputo, J. Gonzalo, G. Jones, J. Kalpathy-Cramer, H. Müller, and T. Tsikrika, editors, *Multilingual Information Access Evaluation II. Multimedia Experiments*, volume 6242 of *Lecture Notes in Computer Science*, pages 110–119. Springer Berlin / Heidelberg, 2010.

[19] T. Röfer, T. Laue, and D. Thomas. Particle-filter-based self-localization using landmarks and directed lines. In A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 608–615. Springer Berlin / Heidelberg, 2006.

[20] F. Smach, C. Lemaître, J.-P. Gauthier, J. Miteran, and M. Atri. Generalized fourier descriptors with applications to objects recognition in svm context. *Journal of Mathematical Imaging and Vision*, 30:43–71, 2008.

[21] Z. ur Rahman, D. J. Jobson, and G. A. Woodell. Retinex processing for automatic image enhancement. *Journal of Electronic Imaging*, 13(1):100–110, 2004.

[22] H. V and P. C. Method and means for recognizing complex patterns, 1962. US Patent 3,069,654.

Embedded Distributed Vision System for Humanoid Soccer Robot

F. Blanes, P. Muñoz, M. Muñoz, J.E. Simó, J.O. Coronel, M. Albero

Abstract—**Computer vision is one of the most challenging applications in sensor systems since the signal is complex from spatial and logical point of view. Due to these characteristics vision applications require high computing resources, which makes them especially difficult to use in embedded systems, like mobile robots with reduced amount memory and computing power. In this work a distributed architecture for humanoid visual control is presented using specific nodes for vision processing cooperating with the main CPU to coordinate the movements of the exploring behaviours. This architecture provides additional computing resources in a reduced area, without disturbing tasks related with low level control (mainly kinematics) with the ones involving vision processing algorithms. The information is exchanged allowing to linking control loops between both nodes.**

Index Terms— **Humanoid robots, visual servoing control, distributed system, embedded computing.**

I. INTRODUCTION

THE firsts experiments in artificial vision systems dates back to 1950 and many of the concepts currently considered essentials have been stated in 1980. Since then computer vision system is one of the most growing topics in the last years with applications in many fields like supervision in industry process, health, security, classification and recognizing, agriculture and robotics.

In all of these applications the final goal is to understand an image obtained using vision sensors (image-information transformation) going beyond the image treatment (image-image transformation).

In the case of robotics, vision systems are one of the most used sensor information sources and in the case of humanoid robots, by definition, is the main source for environment interpretation. In these robots there are many other problems to solve, mainly those related with locomotion stability issues that increase computing requirements. At the same time many other topics like human-machine interaction, robot cooperation, mission and behaviour control give to humanoid robot a higher level of complexity like no any other robots.

In this paper an active vision system is presented under the paradigm of distributed control architecture. This vision system provides high level information after image processing, and shares this information with other nodes using the Ethernet network.

Nowadays, many robot competitions are being used as benchmark test in robotics. This is the case of Robocup which encourages the developments of researchers in all the topics in humanoid robotics, applying the results in a robot soccer competition.

Robocup organization promotes, with the change of rules every year, to imitate in robots the human mechanism for perception. In this sense vision sensors are basic in this competition where active perception is more necessary every time due to limitations in the field of view to 180º.

II. DEVELOPMENTS IN HUMANOID VISION SYSTEMS

One of the most productive ways to analyze the behavior of the engineering designs are competitions because developments can be compared with those of other research groups and draw conclusions.

The annual RoboCup competition is a meeting point of many research groups coming from different part of the world. Participants test the new developed algorithms and designs against adversary teams focusing in a well know high level goal: "win the match". Since 1997, the development of humanoid robots, in particular, vision systems and their integration into control architectures has been a driving line to performance improvements.

In recent years one of the most active areas in the integration of interdisciplinary research efforts has been the humanoid robotics. The humanoid robots development provides a new shape of classical embedded systems challenges (such as stability, sensor network and integration of vision systems) while inspiring new challenges related to the natural goal of human imitation such as artificial intelligence, human-machine interaction and cooperation between robots and other cyber-physical systems. This is a very rich platform inspiring research in many cutting-edge technologies.

Vision systems are becoming the main source of sensory information in robotic systems. Talking about image processing, there are available developments reached in many application fields like medicine, astronomy, transportation, entertainment and, of course, robotics. This state-of-the-art in image processing should be the starting point of RoboCup image sensors.

The RoboCup Soccer League, which we intend to participate with the development proposed here, consists of a series of technical tests and soccer matches between robots. In order to make the environment images processing easier, the objects of interest (goals, ball, field and lines) are indicated by colours that can be clearly distinguished in the colour

F. Blanes, P. Muñoz, M. Muñoz, J.E. Simó, J.O Coronel and M. Albero are with the Universidad Politécnica de Valencia, Instituto de Automática e Informática Industrial.
E-mail: pblanes@ai2.upv.es

spectrum. The promoters claim the RoboCup humanoid robots get increasingly resemble a human, so they include restrictions on the design of the robot. In the field of vision, for example, they have restricted the field of view of cameras and limited the number of cameras to two.

For RoboCup 2010 edition, the proposals of video processing subsystems can be organized into three main categories attending to the attached hardware and the architecture integration capabilities.

- webcam attached to a general purpose Pocket Computer [1],[13],[22].
- webcam [4],[5],[10],[16],[17],[19],[21], CMOS [3],[6],[7],[9],[11],[12],[15],[20] or CCD [2],[18] cameras directly attached to the main board.
- CMOS or CCD cameras attached to a dedicated processor or micro-controller acting as a kind of intelligent camera [8],[14].

The vast majority of the groups present proposals using monoscopic vision (single camera), except for [11],[15] that present a stereo vision system (two cameras). Stereo vision systems intend to look like human vision and allow much more precise estimates of lengths and positions in 3D environments at the cost of increased use of resources.

Mainly inherited from general purpose vision systems, the basic steps in video analysis followed by all participating teams are depicted in Fig 1.

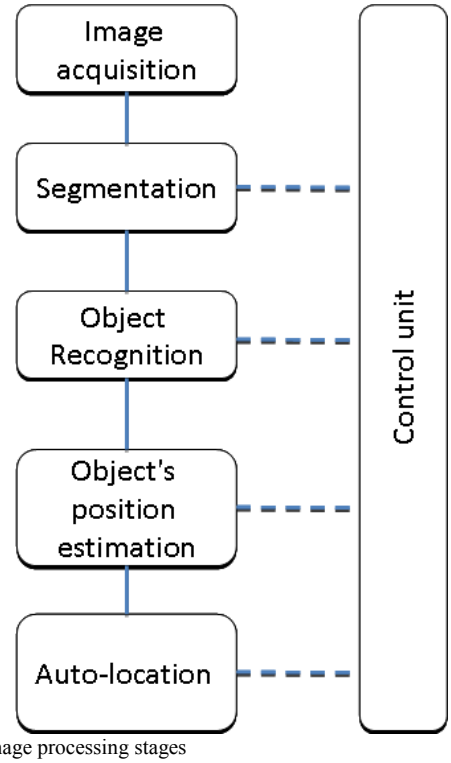


Fig. 1. 5 Image processing stages

All the groups use to follow this strategy (with different image resolutions and different colour spaces), but final implementation can differ in:

- Segmentation "image scoring" instead of the standard for colour thresholds [17]
- Adding some features as detecting extra lines [15], [16]
- Path planning of robots from the characterization of the environment [11]
- Identification of robots [13] and exchange of messages between robots for coordination and information fusion [19].

This paper presents a proposal for an embedded active vision system integrated into the distributed control architecture of the robot. The vision subsystem is an intelligent sensor capable of interpreting the environment and generate information to share with the rest of the control subsystems.

The used scenarios in this paper is a kind of RoboCup environment, but the proposed hardware and control architecture is designed to be applied to any changing environment and ensures the characterization of any real-time dynamic environment.

III. ROBOT PLATFORM

A primary issue in robotics field is the environment perception and robot interaction. The configuration on their mechanical design, accuracy, low weight, etc., is a critical issue for biped robots where stability must be ensured in every moment. This goal must be present during the design process to obtain reliable robot architecture.

Taking on mind these goals, the final dimensions and mechanical proportions for MicroBIRO-II robot are based on a Robocup mid-size league rules. That means that MicroBIRO-II has a total height of 54 cm, and a total weight of 4 Kg. The system is self-contained, holding aboard all the necessary systems to achieve autonomous movements, control, and real world interaction.

MicroBIRO-II could operate up to 21 DOF. This number of DOF guarantees high mobility for the robot platform, and ensures a versatile locomotion. The DOF distribution that we can observe are six DOF per leg, one for the trunk motion, three in each arm, and two for the vision subsystem. This DOF distribution, enables a large number of different movements, like walking, kicking to a ball, getting up from floor, etc.

One of the main rules for robot design is the use of mechanical material that makes their structure lightweight, but strong, and without mechanical deformations. For these reasons some of the MicroBIRO-II mechanical pieces have been designed and mechanized under several constraints to obtain a 3D structure, with low weight and high rigid constraints. As Fig 2 shows, the main legs pieces structure have been designed in only one mechanized piece, without screws or unions. This design ensures us a high rigid structure, with a very low weight.

Humanoids design involves the need to obtain a mechanical structure with a human appearance, in order to operate into a human real world. MicroBIRO-II mechanical structure provides a final design with structural proportions similar to humans, achieving a human appearance, as illustrated in Fig 3.

Fig 3

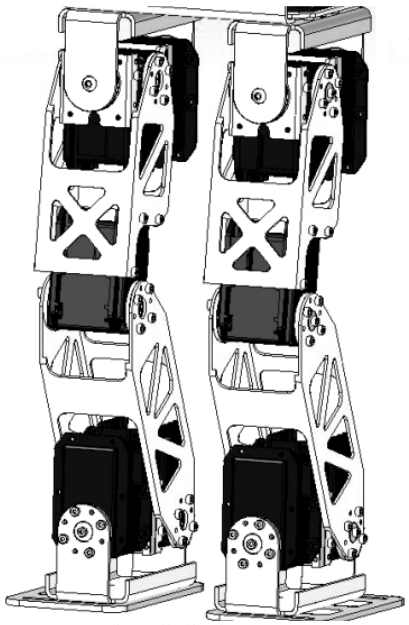


Fig. 2. MicroBIRO-II DoF legs distibution.

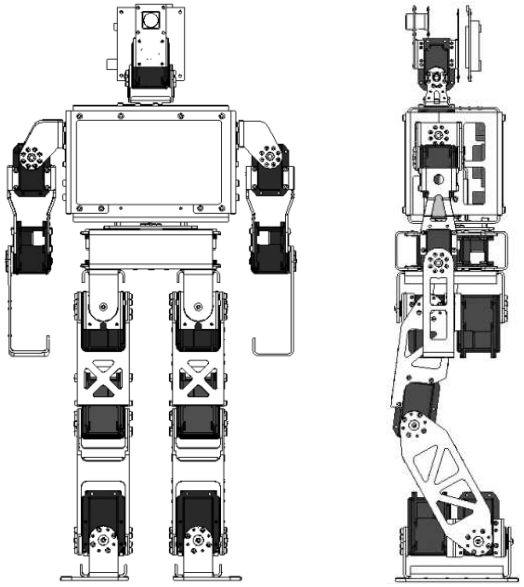


Fig. 3. MicroBIRO-II mechanical views.

Another important point of the design process is the correct selection of the actuators. The weight of the actuators is quite significant, as they add up to 60% of the total weight, but they should move without error all the joints in the robot, for this reason we need to use a high ratio weight/torque actuators. Another important issue on actuators features is their communication topology and bus expandability. This election can determine the main control period on the robot. In this way, Robotis servomotors with RS-485 bus topology communication have been used like actuators on the robot, achieving up to 36 Kg/cm of torque in some of them, and up to 1Mbps. Actually the internal MicroBIRO-II motor control network its capable to obtain a periodicity of 10msec, based on a write/read processes on all nodes.

MicroBIRO-II includes all the power systems in its platform in order to achieve autonomous operation. The power system is composed by the Li-Po batteries and the power supply control board. This system gives to MicroBIRO-II a total energy storage of 40 Watts-hour. This power system is divided into different voltage system: digital supply system (5V), high voltage drive supply (18V), middle voltage drive supply (12V) and low voltage drive supply (8V). Each one of this power supplies are designed for different servomotor models running on robot. To achieve each one of the voltage output subsystem and minimize power losses, the power control board has been designed using DC-DC converters, with high efficiency and power output.

IV. HARDWARE

A. Vision System

The vision system is supported on an embedded module. This autonomous device contains the necessary peripherals for communications with the robot through an RJ45-Ethernet connector, and the connection to the CMOS camera. It also has the power system in order to adjusting the different tension levels that this board needs. The computing capability of this embedded module is given by the core module CM-BF537E, that is part of the Blackfin family from Analog Devices. Concretely the processor DSP Blackfin 537. Their main capabilities are:

- Up to 600 MHz clock speed,
- 132 Kbytes of L1 memory peripherals access,
- 32 MB SDRAM,
- 4 MB Flash
- 25 Mhz crystal.

The sensor used in this module is an Omnivision OV7660 CMOS camera. The camera is connected to the embedded system through the SCCB (Serial Camera Control Bus).This sensor support different image resolution modes and formats.

The vision system is situated in the robot's head (Fig 3). It can adjust its orientation relative to the axis pitch and yaw. The vision system must be able to handle these actuators to set the camera in the correct orientation to track the objects.

B. Main Control Unit

The main control unit of the robot is supported on another embedded module. The processor module integrates an XScale microcontroller, concretely the PXA320 processor type that outstands for its high performance and low power consumption. This board allows to the system to manage the SSP-SPI bus, USB, Ethernet or WiFi connection among others.

A novel approach is currently under development designing a new board that contains an FPGA mapped in the processor memory. This FPGA will multiplex the actuator bis RS-485 in 6 branches in order to manage each one of these in parallel. It is expected that this device will allow real-time management of communication buses, freeing up the main microprocessor from this difficult task. In this way to get the information, the processor reads directly as if it were an external memory. This

new configuration allows to dispose all the information contained in the actuators in each one of the control cycles.

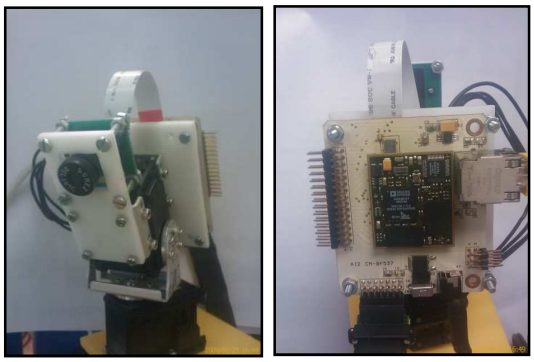


Fig. 4. Vision system Hadrware.

C. Communication

In order to allow the information exchange between the vision system module and the main control unit, a messaging protocol based on TCP / IP has been implemented. The characterization of the environment of the robot detained after the image processing is sent to the control module, which is responsible for taking appropriate decisions according to the state of the system and the sensory input.

V. SYSTEM ARCHITECTURE

The proposal is based on the integration of an embedded vision system into the distributed control architecture of a humanoid robot. A message protocol has been defined for the communication between the main control unit and the vision system.

The main control unit selects the task to be performed by the vision system among a predefined set of task. The information extracted by the vision system is shared with the main control unit and the rest of the distributed control system via Ethernet.

A. Active vision system

The vision system is mounted with two servomotors, allowing movements with two degrees of freedom. The motors are connected directly to the DSP through its UART port using serial TTL communication. So, the vision system has the capability of deciding its own movements, becoming then an active vision system and receiving from the main control unit just higher level instructions.

The entire computation referred to the visual perception of the environment is carried out in the DSP. The application in the DSP has been implemented as a multithreading process.

The main thread takes over the camera and communication initialization and runs the acquisition and processing threads, which are synchronized using semaphore mechanisms.

The acquisition thread reads the image delivered by the camera sensor through the driver and stores it in a shared buffer. The processing thread picks the stored image and applies the algorithms to extract the desired information. This information is classified in a matrix, according to the messages protocol, and sent to the main control unit through the Ethernet communication.

Threads are implemented using the VisualDSP Kernel, a very light and robust operative system, provided with the development environment of the Blackfin DSP, which allows, among other things, to define threads, priorities, critical regions, semaphores and messages.

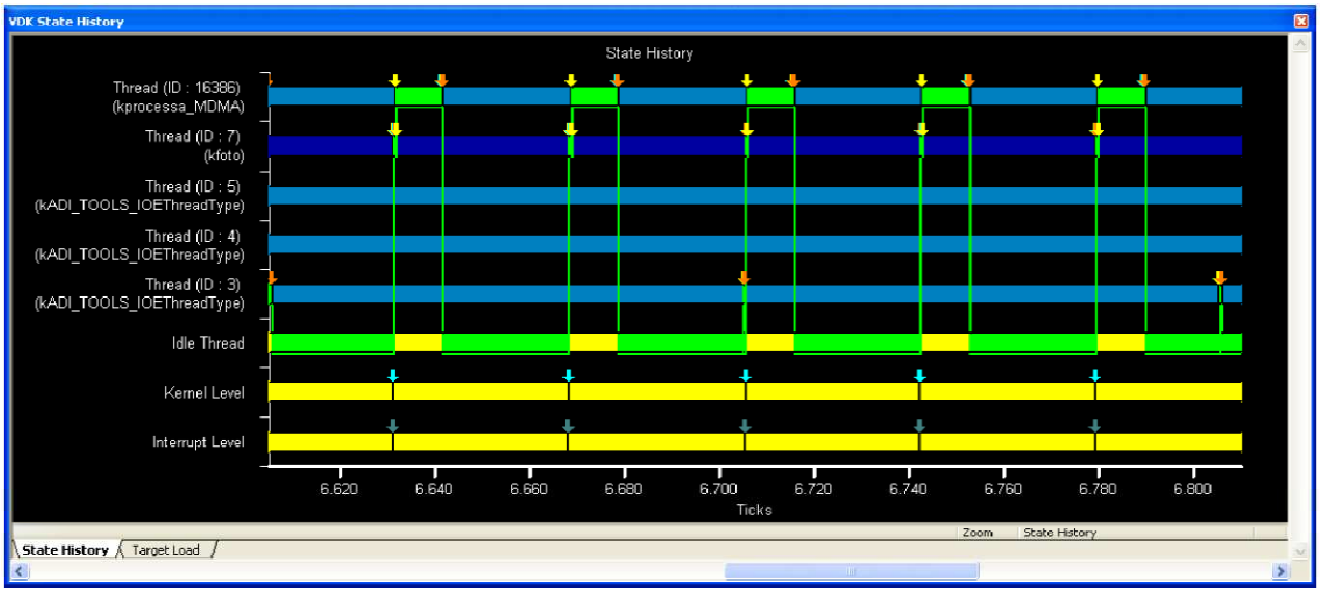


Fig. 5. Execution chronogram of the DSP application.

The computation time (CT) of the acquisition thread is minimal, since the image storage is carried out using DMA (*Direct Memory Access*) without any processor usage, while the CT of the processing thread depends on the environment characterization needs and the algorithms used.

The execution period of both threads is related to the CT of the processing thread and must be multiple of the camera

frame rate.

Working on QVGA resolution (320 x 240 pixels), the frame rate is 30 fps (one image each 33 ms) and the CT processing thread is 10 ms, with the algorithms developed until the moment. Thus, the execution period would be 33 ms. If the CT processing thread would any time overcome the 33 ms, the execution period would increase to 66 ms (two times the frame rate).

So now, with QVGA resolution, one image is processed each 33 ms with a processor usage of 30 %. The execution chronogram is shown in Fig 5.

The vision system extracts the position of the objects of interest from each image, stores it in a structure and sends it to the main control unit each execution period. This shared structure, shown in Table 1, is composed of:

a) Position of the servomotors in the moment of taking the image and objects recognized in the image
b) Position of objects respect to the robot's centre of mass.

- The mission layer is defined by behaviours, consisting of various basic skills, for example, look for the ball and go to it.
- The tactic layer is defined by basic skills, consisting of various trajectories, for example, walking.
- The reactive layer defines trajectories or articulations positions.

The vision system task can be selected from any of the layers, but it is intended to be done from the highest layers (mission or tactic). On the other hand, the vision system will maintain the last behaviour defined until a new one is received from the main control unit.

The main control unit decides, according to the state of execution and the information knowledge, the behaviour to be performed by the vision system.

The actions defined until now are:

- Main control unit access to the servomotors.
- Adopt a predefined behaviour (scan, tracking, idle).

The position and orientation of the vision system

TABLE I
MESSAGE WITH POSITION AND ORIENTATION OF THE VISION MODULE

struct DSP_VISION								
struct OBJETCS [NO_OBJETCS]								
SERVO_H_POS [0,1024]	SERVO_V_POS [0,1024]	update1	ρ1	φ1	…	upn	ρn	φn
2 bytes	2 bytes	1 byte	2 bytes	2 bytes		1 byte	2 bytes	2 bytes

TABLE II

struct COLIBRI_VISION						
ACTION	**DATA**					
0x00: servomotors access 0x01: Behaviours	SERVO_H_POS [0,1024]	SERVO_V_POS [0,1024]	BEHAVIOURS 0x01: BALL 0x02: SCAN 0x03: IDLE	Z_HEAD [0,1024]	OY_HEAD [0,1024]	OZ_HEAD [0,1024]
1 byte	2 bytes	2 bytes	1 byte	2 bytes	2 bytes	2 bytes

One of the behaviours performed by the vision system is the image-based object tracking. Based on the error between current and desired object coordinates in the image plane, the servomotors move the camera until the object is in the centre of the image.

B. Main control unit

The control architecture of the robot is divided in 3 basic layers: mission, tactic and reactive.

coordinates origin (under the two servomotor axes origin) respect to the centre of mass of the robot, solved by direct kinematics, is also sent to the vision system to calculate the distance and orientation of the objects.

All this information is classified into a structure, shown in Table 2, containing: action to be performed, servomotors desired position, behaviour, and position and orientation of the vision system coordinates origin.

It is intended that in a future, the main control unit, according to the needs and the available resources, will decide

the type of processing to apply in terms of time and reliability.

VI. IMAGE PROCESSING

The work developed until now in image processing covers the steps of image segmentation and object recognition (ball and goals).

A. Colour space and resolution

The selected colour space of the image delivered by the sensor is YUV, which splits the pixels' information into components of illuminance (Y) and chrominance (U, V). The resolution adopted is QVGA (320 pixels width and 240 pixels height)

B. Image segmentation and pixel connectivity

The goal of image segmentation techniques is to isolate pixels of the image following predefined criteria in order to simplify the later processing and analysis.

In our case, the segmentation criteria are based on colour, so the isolated pixels correspond to objects known in advance. From the original image, pixels whose colour is between previously defined thresholds are isolated. At the same time, contiguous pixels of the same colour are grouped. These pixels clusters are characterized with colour, number of pixels, width, height and centroid).

Five different connectivity algorithms were compared, being the *Run Length Encoding (RLE)* algorithm the one selected. The processing times of the algorithms, implemented in Matlab, are shown in Table 3.

The *RLE* algorithm implemented represents the segmented image as sequences of 1s or *runs*, characterized by the centre

TABLE III
SEGMENTATION AND CONNECTIVITY ALGORITHMS COMPARISON

Algorithm	Execution time (s)
Adjacency matrixes	1.32
Flood fill approach	1.96
Equivalence class resolution	0.98
Union find	1.04
Run length encoding	0.90

pixel coordinates and the number of pixels. The adjacent *runs* are grouped in object candidates.

```
candidates run_lenght_encoding()
for each row in image

    sequence of same colour pixels
    run = {colour,no_pixels,row,centre}

    if (run is adjacent to candidate and
        has the same colour)
              update candidate
    else
              create candidate = {colour,
              centroid, height, width}
    end
end
return candidates;
```

Algorithm 1. Run length encoding algorithm

C. Objects recognition

Once characterized the grouping pixels with colour, width, height and centroid, shape algorithms are applied in order to identify the predefined objects (ball and goals)

```
centroid object_recog(candidates, object)

for each object candidate

  filtered_candidates =
   shape_filter (candidate)

  decision(filtered_candidates)

  if (decision == ok)
    return candidate.centroid
  end

end
```

Algorithm 2. Objects recognition

D. Distance and orientation estimation

To have a good estimation, the object must be in the centre of the image, i.e. it must be tracked.

Once there, the distance and orientation are calculated, according to the neck's origin position, the current neck's servomotors position and the position of the camera respect to the origin resulting of the design.

E. Information for the main control unit

The recognized objects, with their distance and orientation estimation, are organized in a matrix and sent to the main control unit.

VII. CALIBRATION TOOL

During the normal operation, it is very useful to have a tool that allows the user to modify the vision system performance, without modify and reload the robot software.

In this way, the system must allow to define the different objects that the robot has to identify in the environment, as well as the image features.

The need of this tool becomes evident during the RoboCup. In this robot competition, before each match, an adjustment of the vision system is required to prevent the changes, mainly in lighting, that suffers the play field. The fact of having this application allows the user to adapt quickly to changing conditions. Otherwise the robot may not be able to distinguish some of the objects (ball, goal, lines), with behaviour problems that these issues entail.

Attending to this, a PC application that can connect directly to the robot vision system to use the following capabilities:

- Adjust the settings of the camera to the lighting and environment conditions.
- Calibrate the colours of objects depending on lighting conditions and object types.
- Allow processing of images captured or not by the robot in order to show in the screen the extracted information.
- Test and Debug the new image processing algorithms.

The procedure to be followed when introducing the robot in a new environment, for instance a new field, would be:

- First, you should adjust the camera settings. For this, the application allows to select auto exposure mode, auto white balance mode, and all those image parameters that offers the camera. Otherwise, user can set these manually based on the experience and expertise of the programmer. For this task, the GUI offers a friendly interface with scroll bars, allowing to save and reload sets of parameters.
- After this image adjustment has been done, ranges corresponding to the colours of different objects are defined in order to be used in segmentation algorithms. This process is made easily by clicking on the objects of interest in the image taken by the robot. The application automatically extracts the characteristics of the clicked pixel, and uses these as a seed to extrapolate these features to similar objects.

To manage these definitions, a procedure that allow you to take a picture through the robot camera and directly assign each interest pixel to the corresponding object has been dedined. After that, the application extracts the properties of these pixels and redefines the object colour ranges. To perform the pixel selection properly a set of tools as such as zoom have been developed and implemented.

The graphical interface is shown in Fig 6, in witch can be appreciated the picture before and after the information extraction process. In the second one, the detected objects have been highlighted the detected objects. In the bottom one, the set of controls to parameterize the image is shown.

VIII. RESULTS

A. Image segmentation and connectivity of pixels

Fig 7 shows several segmented images. The pixels whose colour matches the colour of objects to identify are the segmented pixels. The objects to identify are the ball and the goal.

The group of pixels to identify the objects are characterized by means of the connectivity. Additionally, it also removes groups that do not contain a minimum number of pixels.

B. Identifying objects

The objects are identified from the characterized group of pixels. When the object is identified, a message is built which is transmitted to the control module. This message contains the position of all the objects identified. The position is transmitted in image coordinates. The results are shown in figure 7.

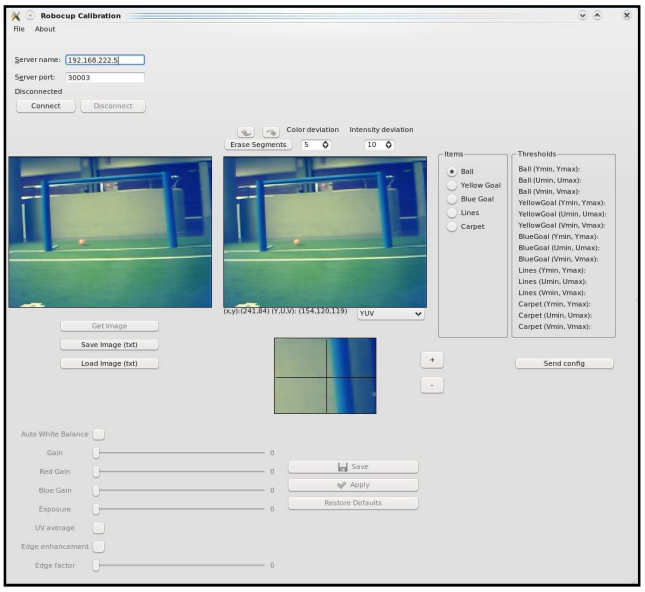


Fig. 6. Graphical Calibration Tool.

IX. CONCLUSION AND FUTURE WORK

In this paper a humanoid robot design and the developed work has been presented focusing on the embedded vision system for environment recognition. This is a long term project where many engineering areas converge (mainly mechanical engineering, electronics, and computer architecture) in a complex system which has been designed for Robocup competitions. This application oriented approach do not subtract to developments applicability for other scenarios.

In the next future we will focus on completing the basic steps shown in figure 1, i.e., the estimation of the location of other objects (apart from ball and goal) with respect to the robot and the self-localization of the robot in the environment.

Furthermore, we will work on developing of line detection algorithms to combine with the location information of the fixed objects such as goals. This will improve the robot self-location ability.

In addition, we will study the possibility of incorporating stereoscopic vision on the robots. This approach has not been use in small humanoid robots, but currently image processing dedicated embedded microcontrollers allow to expand computing power with reduced size. The effort must be done in the architecture design following distributed control principles. Finally the information fusion and sharing among robots will also be considered using communication capabilities.

ACKNOWLEDGMENT

This work was supported from the Spanish MICINN project SIDIRELI DPI2008-06737-C02-01/02 and FEDER founds.

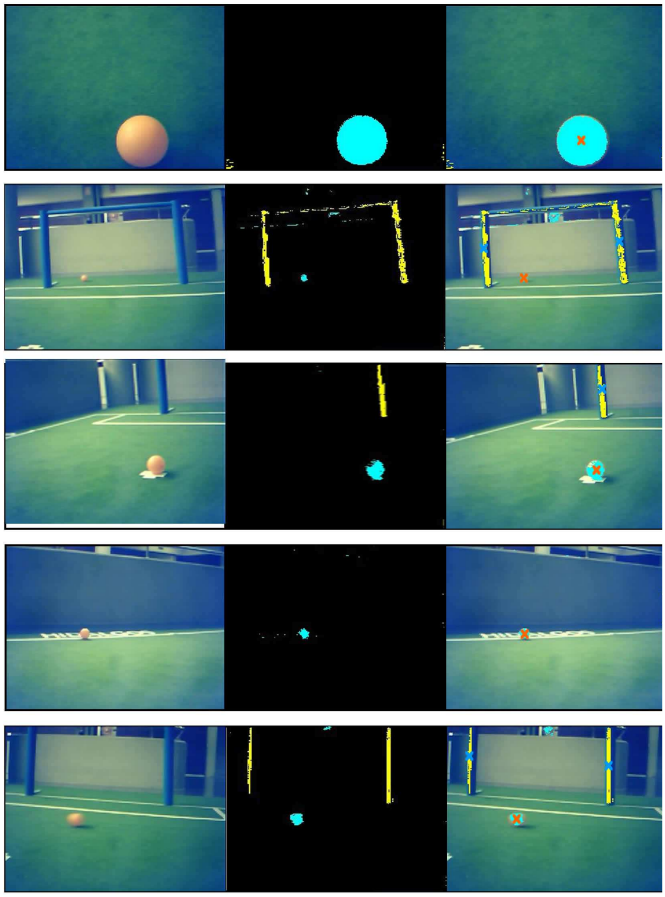


Fig. 7. Segmentation and identification algorithms of objects.

REFERENCES

[1] Tzuu-Hseng S. Li et al "aiRobots: Team Description for Humanoid KidSize League of RoboCup 2010". Workshop Robocup Singapore 2010.

[2] Chokchai Pengyasa et al. "Team BSRU-I: Team Description Paper". Workshop Robocup Singapore 2010.

[3] Aphilux Buathong et al. "Chibi Dragon Team Description Paper". Workshop Robocup Singapore 2010.

[4] Luis F Lupián et al. "Cyberlords RoboCup 2010 Humanoid KidSize Team Description Paper". Workshop Robocup Singapore 2010.

[5] M. Friedmann et al. "Darmstadt Dribblers. Team Description Paper for Humanoid KidSize League of RoboCup 2010". Workshop Robocup Singapore 2010.

[6] Bennet Fischer et al. "FUmanoid Team Description Paper 2010. Workshop Robocup Singapore 2010.

[7] Lim Sock Lip et al. "Team NYP Lions: Team Description Paper". Workshop Robocup Singapore 2010.

[8] Roberto Carlos Ramírez Márquez et al. "PIONEROS MEXICO Team Description Paper ROBOCUP 2010 Singapore". Workshop Robocup Singapore 2010.

[9] Guangnan Ye et al. "PKU-SHRC Team Description for RoboCup 2010". Workshop Robocup Singapore 2010.

[10] Buck Sin Ng et al. "Robo-Erectus Jr-2010 KidSize Team Description Paper". Workshop Robocup Singapore 2010.

[11] Keith Sullivan et al. "RoboPatriots: George Mason University 2010 RoboCup Team". Workshop Robocup Singapore 2010.

[12] Shohei Takesako et al. "SitiK KIT. Team Description for the Humanoid KidSize League of RoboCup 2010". Workshop Robocup Singapore 2010.

[13] S. Hamidreza Mohades Kasaei et al. "Persia Humanoid Robot. Team Description Paper 2010". Workshop Robocup Singapore 2010.

[14] Guillermo Villarreal-Pulido et al. "Bogobots-TecMTY humanoid kid-size team 2010". Workshop Robocup Singapore 2010.

[15] R. Gerndt et al. "WF Wolves KidSize Team Description RoboCup 2010". Workshop Robocup Singapore 2010.

[16] Jaekweon Han et al. "Team DARwIn. Team Description for Humanoid KidSize League of RoboCup 2010". Workshop Robocup Singapore 2010.

[17] Soo Theng Koay et al. "Team Description 2010 for Team RO-PE". Workshop Robocup Singapore 2010.

[18] Thavida Maneewarn et al. "Team KMUTT: Team Description Paper". Workshop Robocup Singapore 2010.

[19] Tang Qing "ZJUDancer Team Description Paper". Workshop Robocup Singapore 2010.

[20] Ching-Chang Wong et al. "Humanoid Soccer Robot Design by TKU Team for Humanoid League of RoboCup 2010". Workshop Robocup Singapore 2010.

[21] Chung-Hsien Kuo et al. "Team Description Paper: HuroEvolution Humanoid Robot for RoboCup 2010 Humanoid League". Workshop Robocup Singapore 2010.

[22] Javier Testart et al. "UChile RoadRunners 2010 Team Description Paper". Workshop Robocup Singapore 2010.