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.

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

E-mail: rcintas@unex.es

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].

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.

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.

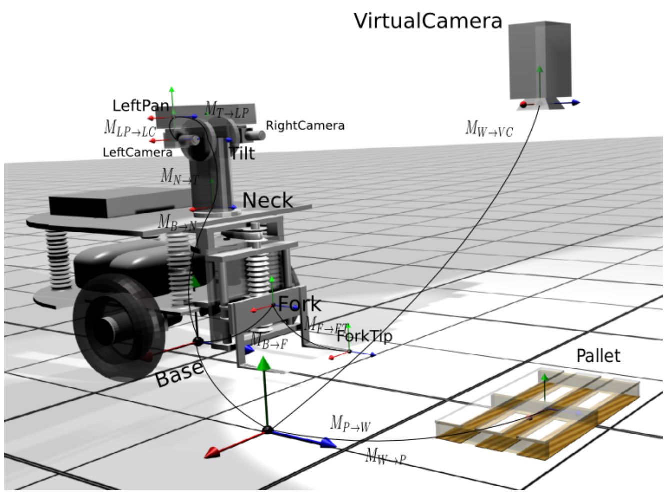

Fig. 1: Reference frames taking part in the pallet manipulation problem.

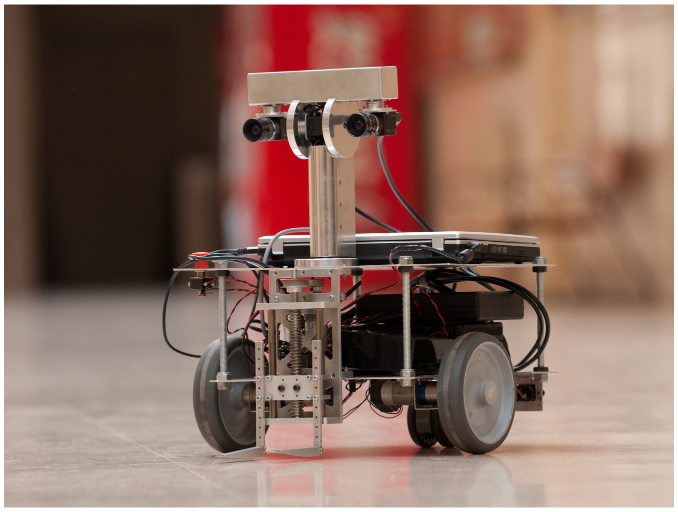

Fig. 2: The RobEx platform.

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

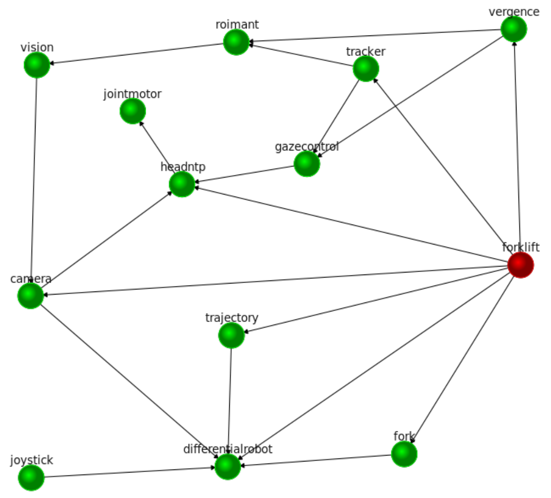

Fig. 3: Component network.

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

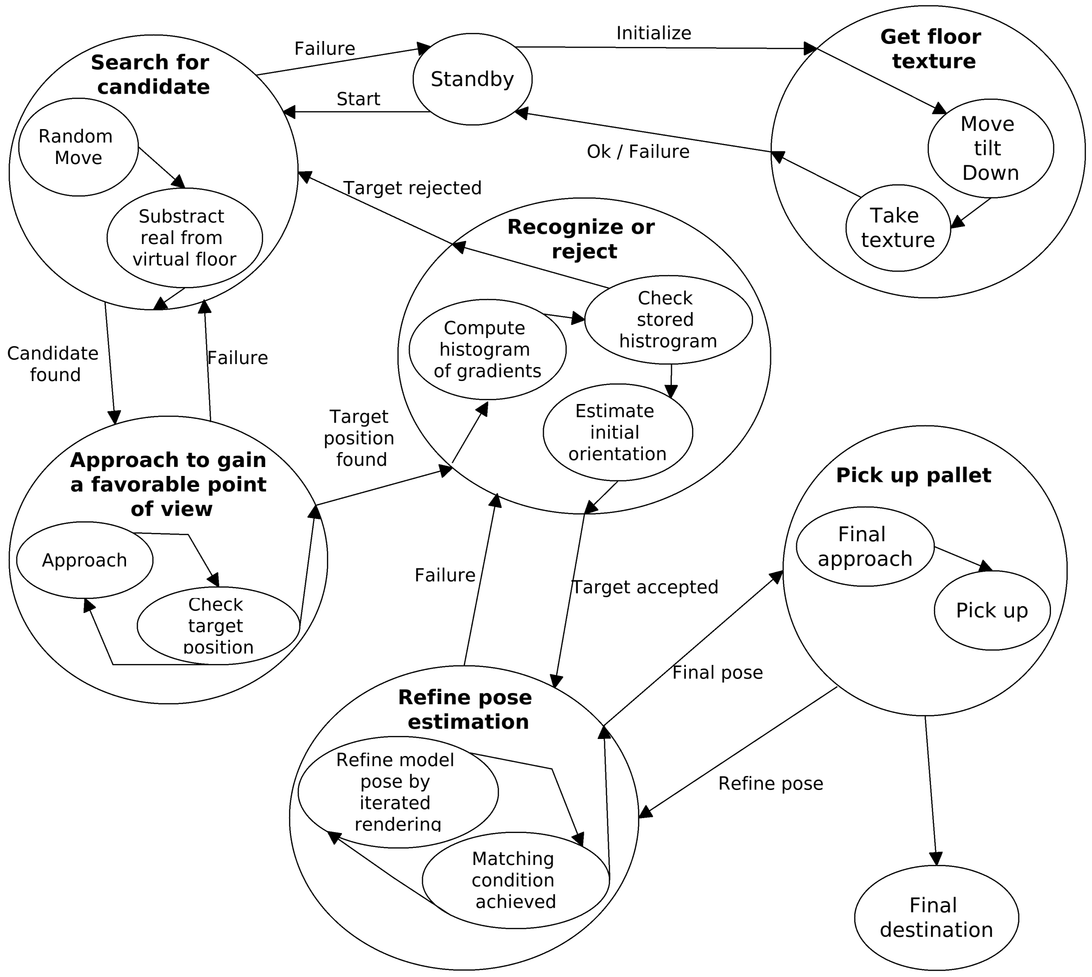

Fig. 4: Hierarchical state machine used in the experiment. There are six macro states and the *Standby* and *Final destination*. Each macro state includes inner states representing with finer detail the structure of each stage. Note the arrows pointing back to former states, signalling failure situations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

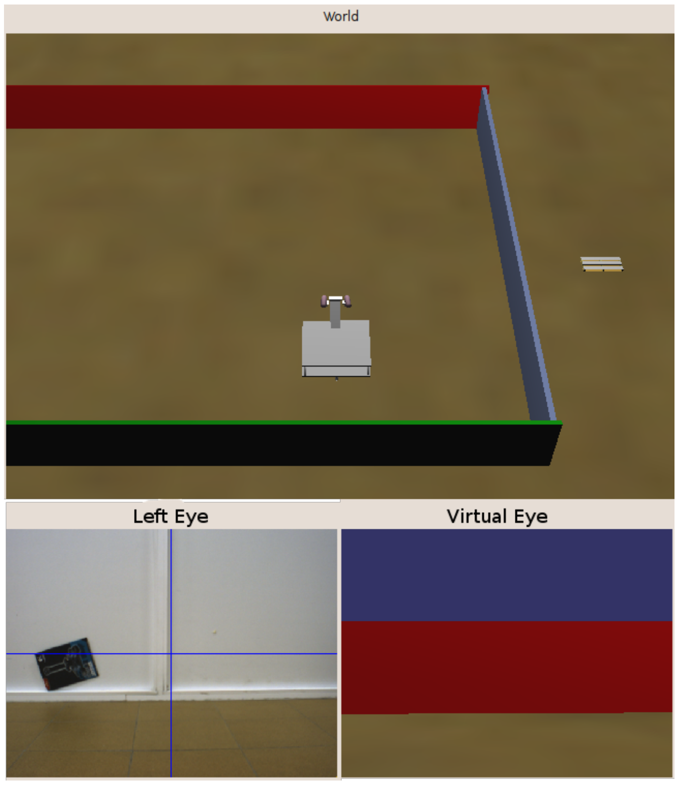

Fig. 5: Top: the 3D representation of the modelled world. Bottom: an image fetched from the real camera (left), and the predicted image using the 3D engine (right).

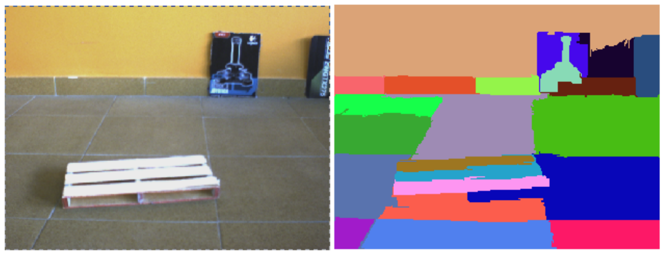

Fig. 6: Left, original image. Right, superpixels obtained with the Felzenswalb and Huttenlotcher segmentation algorithm

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

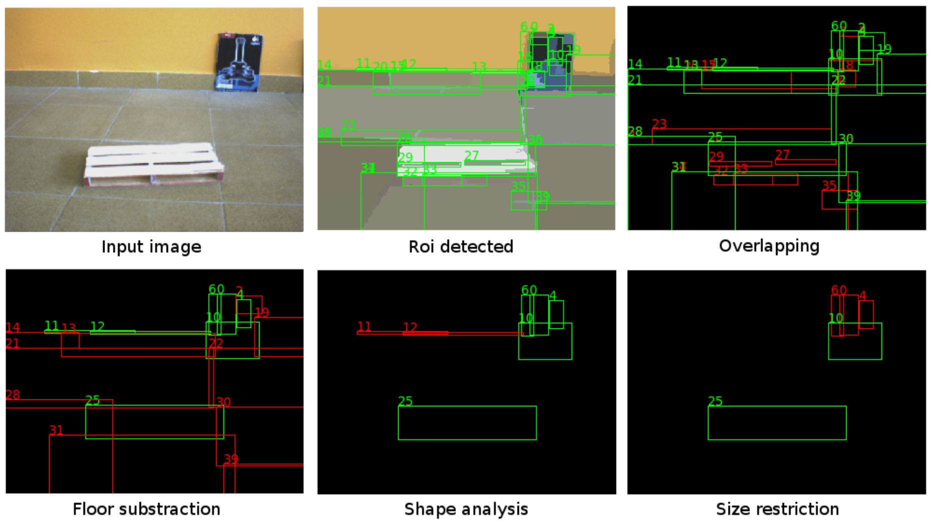

Fig. 7: Segmentation and classification process for extraction of candidate regions. See text for details.

"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:

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

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:

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

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

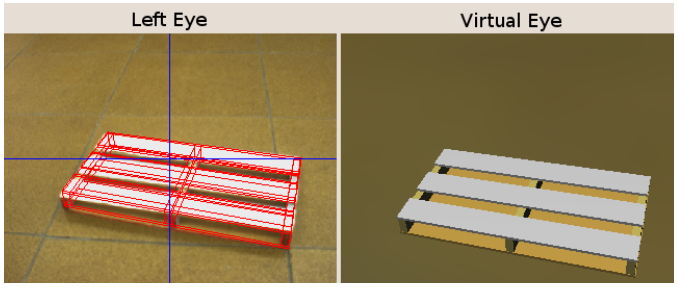

Fig. 8: Final estimation of pallet pose after iterating over x, y and α computing de difference between the real and the synthetic images

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

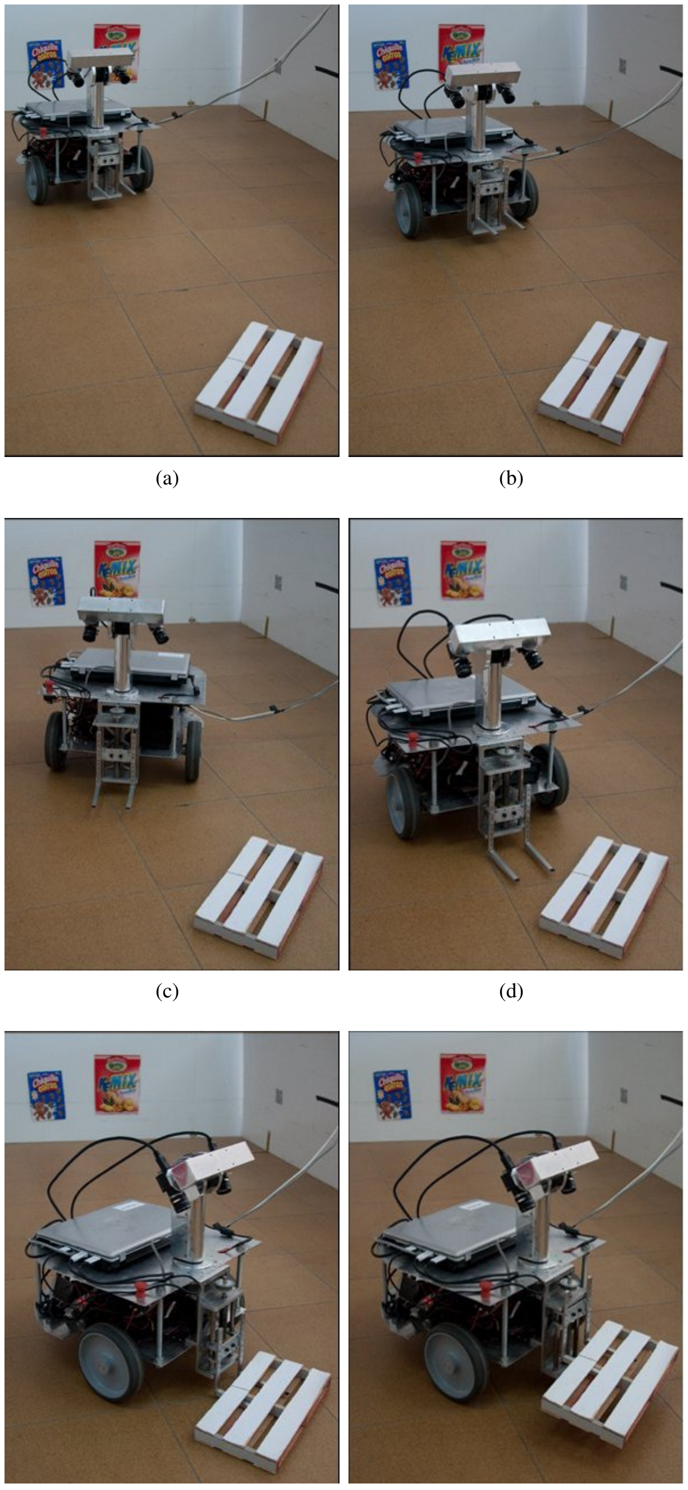

Fig. 9: Sequence of images showing different states of the pallet manipulation experiment: (a) detection of the pallet; (b) approach the target position; (c) visual tracking of the target; (d) refine pallet pose estimation; (e) final approach; (d) pick up operation.

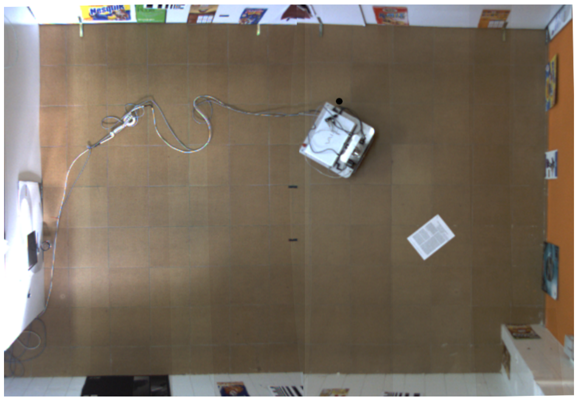

Fig. 10: Overhead perspective of the environment used for the experiments.

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.

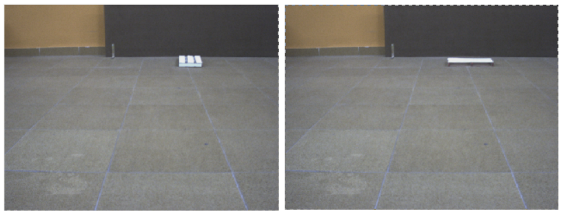

Fig. 11: Perspective of a pallet, with different orientations, situated two meters away from the robot

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