Visually-guided Object Manipulation by a Mobile Robot

L. Pinero, Escuela Politécnica, Univ. de Extremadura, luispineror@gmail.com
R. Cintas, Escuela Politécnica, Univ. de Extremadura, rcintas@gmail.com
L. J. Manso, Escuela Politécnica, Univ. de Extremadura, lmanso@unex.es
P. Bachiller, Escuela Politécnica, Univ. de Extremadura, pilarb@unex.es
P. Bustos, Escuela Politécnica, Univ. de Extremadura, pbustos@unex.es

Abstract

For mobile robots, the problem of interaction with simple objects in a semi-controlled environment is a rich source of challenging situations. In this paper we present a real experiment dealing with the design of a sequential task and its implications in the active nature of the perceptual process involved. In order to set up this experiment, it is required a non trivial set of functioning senso-motor behaviours. We build on this set to design and test a pallet picking task in which the robot has to locate, approach, obtain the pose and, finally, pick up the target. The only sensorial information available to the robot is its 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's binocular head. We present the conceptual and computational models and the results of the initial experiments in a real setup.

1 Introduction

Designing computational structures that can represent and be used in the execution of complex sequential plans involving manipulation is an important problem in the field of mobile robotics[10]. 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 a 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. When dealing with real robots and real software, new complexities arise derived from sensor noise and from the ever increasing number of lines of code, running on always limited computational resources. Furthermore, teams of many developers and the need for code reuse settle even more demanding requirements on feasible solutions. A promising approach is to use component-oriented specialized middlewares[2, 7] that provides a means of dividing, reusing and organizing large amounts of sophisticated and changing code, typical of robotic research environments.

In this work we use RoboComp [4, 1, 8] and the RobEx robot[5]. RoboComp is a component-oriented robotics framework. It is based on Ice, a industrial grade communications middleware from ZeroC. RoboComp provides several useful features such as: **a)** a component model; **b)** an automatic release installer; **c)** a flexible directory structure; **d)** several utility scripts for creating and modifying components; **e)** a graphical component manager for static configuration of component networks and dynamic monitoring of their behaviour; **f)** seamless connection to the open source Gazebo simulator; **g)** logging facilities; **h)** recording and playback of component data structures for off line development and program debugging; **i)** rapid Python-based prototype development scripting.

Building on this infrastructure, we can more easily focus on the problem of designing sequential tasks involving active visual searching, detection, recognition, pose estimation, manoeuvring, picking and delivering. As a simple but realistic example of these sort of tasks, we have selected the problem of manipulating a pallet by a robot equiped with a 1 dof frontal forklift. All sensor information it can use comes from its 4 dof stereo head and the base internal odometry. 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 designed to also reduce the uncertainty in the robot-pallet spatial relation. We thus interleave actions to reach the goal with actions to perceive the target, 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 inner representations built to maintain these initial hypothesis are simple and undetailed. But as the robot proceeds toward the target, more complex representations are used and more computation time is spent in refining these representations. During the approaching stage the robot keeps itself focused on the target, by performing attentional eye movements. So, the closer the robot is, the bigger the confidence on the target being a pallet and on its size and orientation.

2 Sequential task design

Robust pallet manipulation by mobile robots using only odometric and visual information requires a careful design of a sequence of states and transitions, so the robots can recover safely from perceptual and action misreadings or incorrect model hypothesis, back to the main plan[9]. In this work we decompose this task in the following generic subtasks:

1. Gather context information
2. Search for a target object candidate
3. Approach to gain a favourable point of view
4. Recognize or reject the candidate as the target object and estimate initial orientation
5. Refine object pose estimation by minimizing reprojection error of a synthetic parametrized internal 3D model
6. Approach and pick the target object
7. Start final destination approach procedure

Note that this sequence of tasks is quite generic and can be applied to a wide variety of robot and context configurations. Each of these subtaks represents an intermediate state towards reaching the final goal. To do so, each state is implemented with algorithms that solve the specific problems locally and/or through remote calls to other components. The possibility of doing remote procedure calls to other components is what gives behaviour allocation its real meaning. Also and no less importantly, there are different failure conditions, local and remote, that can occur during the execution of each subtask. These error conditions have to be managed by transitions to former states or to halt exception states. They are not denoted in the former list of subtasks but appear in the graph shown in figure 2.

A standard approach to this problem is the formalism of state machines (e.g. as developed by Harel[3]). 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 behaviour often depends not only on the last or current event, but also the events that preceded it. With statecharts, this information is easy to express. Qt Software has recently released a state machine framework based on Harel's StateCharts[6]. This framework provides an API and execution model that can be used to effectively embed the elements and semantics of statecharts in Qt applications. This tool 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 in these experiments and will show corresponding state diagrams modelling the visual handling task described before. In this section we describe the state machine used in the experiment and the restrictions imposed to the environment inhabited by the robot. The goal of this experiment is to analyse the use of a state machine framework inside our component-based robotics middleware, RoboComp, in order to run a complex sequential behaviour allocation task. To show the complete architecture of the system we need to describe it in two levels. The first one is shown in the next sec-

tion and contains the specific configuration of components that drives the robot. The second one is described afterwards and shows the state machine designed for the experiment.

3 Basic Components

Each component in the graph contributes with a function to the whole system. Some of them are called behaviours when they are goal oriented, such as Tracking, Vergence or Trajectory. A list of the components and a brief description of its function are detailed below.

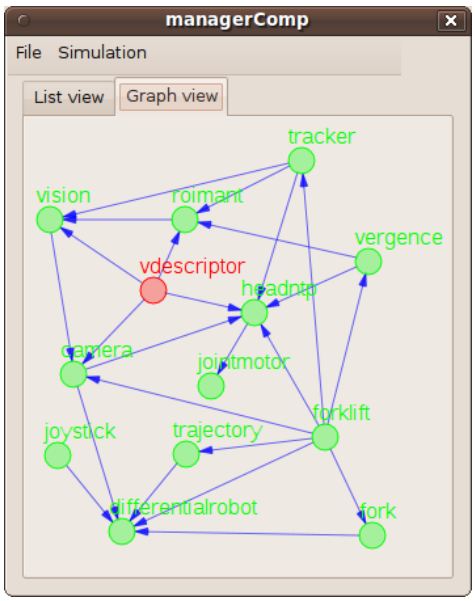

Figure 1: Component graph

- DifferentialRobot: Provides a functional API to control a differential mobile robot and maintains a probabilistic odometric state that can be zeroed at any time. It can also run in a energy saving mode deactivating the power amplifiers if no command is received within a given period of time. It currently supports drivers for a custom made microcontroller based PID controller, for the open source Gazebo simulator and Player hardware abstraction layer.
- JointMotorArray: An array of servomotors sharing a communication bus. The component provides configuration parameters for the bus and for each motor, and an API to command the servos individually or synchronously. New bus drivers can be added by subclassing an abstract "Handler" class.
- Fork: Provides basic functional access to the RobEx forklift manipulator.
- JoyStick: Provides an API for a generic Linux joystick used to control a differential mobile robot.
- HeadNT2P: Models a stereo head with four degrees of freedom: neck (common pan) and common tilt movement affecting both cameras, and two separate camera-specific pan movements. Its API offers commands to trigger coordinate saccadic and smooth pursuit movements.
- CameraArray: An array of cameras sharing a common communication bus. Provides configuration parameters for the bus and for each motor, and an API to access each camera individually or synchronously. New camera drivers can be added by subclassing and abstract "Camera" class. Currently, the component supports Firewire, V4L2, Gazebo and the specific sdk's of Prosilica and Point Grey cameras.
- Vision: Computes regions of interest as local extrema in Harris-Laplace pyramid. The list of regions can be recovered along with the pyramid. 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. It maintains in memory a local updated copy of the regions visible in the world around de robot. In stereo configurations it computes the 3D coordinates of regions using a standard correlation measure and the epipolar geometry as reported by HeadNT2P.
- Tracker: Controls the dominant camera to provide a tracking behavior on a certain roi or initial angular coordinates. It can apply correla-

tion over the whole pyramid to recover from failure situations.

- Vergence: Controls the non-dominant camera so its principal ray converges with the dominant camera's principal ray at the currently attended 3D point. It uses the whole pyramid to avoid local minima when finding the best matching angular position. Correct vergence maximizes the shared field of view in a stereo configuration.
- RobotTrajectory: Computes and follows local trajectories using odometric information. It can compute Bézier curves to fit initial and final orientation conditions for the robot.
- Forklift: This components holds the main state machine that sequences the behaviours in this experiment.
- Vdescriptor: Computes texture descriptors on the regions of interest maintained by Roimant. It can be activated on demand due to the heavy computations involved. Currently implemented descriptors are: Rift, Spin y Sift[1].

4 A State Machine for Pallet Manipulation

The second level is the specific state machine used in the experiment that is embedded in ForkliftComp and is shown in figure 2.

We go now through each of the relevant states:

4.1 Get floor texture

The robot maintains an inner model of itself in a class that is replicated in almost each component. Copies of this object are not synchronized but get updated by remote access to DifferentialRobot and HeadNT2P components state. Using this object, Forklift extends the internal model to build a basic 3D representation of its environment using the OpenSceneGraph engine. It initially assumes a flat floor underneath the robot. When in this state, the floor is filled up with the texture obtained from the central region in the left camera image. In order to acomplish this step the robot points down directly to the closest floor area. This rapid reflex allows us to use a very simple object detection algorithm based

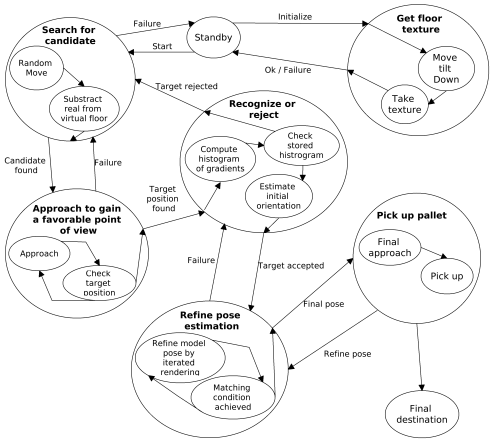

Figure 2: State machine used in the experiment

on color differences that is explained for the next state. Figure 3 shows the initial 3D world representation and the real(left) and virtual(right) images after copying the floor texture.

4.2 Search for a target object candidate

Taking advantage of the realistic texture for the synthetic floor obtained in the previous state, and assuming that the real floor texture is homogeneous in the local space extending in front of the robot, we derive a very simple object detection algorithm. Using the rendering capabilities of the OSG engine, we obtain a video stream from a virtual camera placed exactly where the real left camera is located in the robot head. This virtual camera is configured to have the same intrinsic parameters as the real one. The algorithm follows these steps: substract virtual and real images in RGB space using euclidean metric, convert difference image to gray level, decimate twice the gray level image, binarize, search columns in the binary image from lower to upper rows searching for a white pixel, flood fill from that seed to obtain the closed contour and bounding box, check that the bounding box has the correct size, check that the bounding box is surrounded by floor pixels and compute the 3D coordinates of the center of the box and establish them as current target hypothesis. The result of these steps is shown in Figure 4.

3D coordinates are computed using inverse pro-

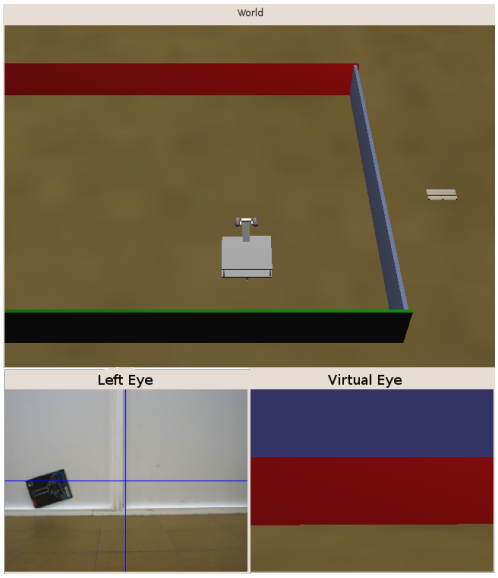

Figure 3: 3D world and virtual and real images of the floor

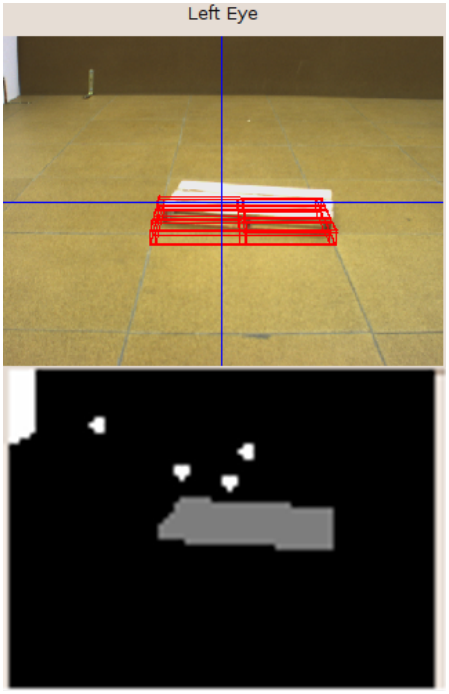

Figure 4: Target object segmented. No estimated orientation computed yet

jection from the camera under the assumption that the object is on the floor. Note that a rather flat object floating on the floor can meet this condition leading to a wrong depth measure, but it would be a very unlikely object. Nevertheless, even this situation could be easily detected with additional tests on the position of the object. As a matter of fact, 3D coordinates of salient points in the pallet are being computed by the RoimantComp component using the updated epipolar geometry of the head provided by HeadNT2PComp. On the other side, an object adjacent to a wall cannot be detected. Again, weaker hypothesis on the surrounding floor texture can be accepted at the cost of additional 3D tests to discard off the floor objects. Floor texture homogeneity or shadows are harder restrictions to overcome and more sophisticated texture modelling algorithms would be needed.

4.3 Approach the candidate object to gain a favourable point of view

Once a candidate object has been detected, the robot starts an approaching behaviour that should take it to a close and favourable viewpoint over the object. We define here *favourable* as a combination of the distance of the object to the robot and the percentage of image occupied by the bounding box. To accomplish this subtask several concurrent behaviours must be active. These behaviours are executed by other components running in the robot computer. For the cameras, the tracking component fixates the candidate object triggering correcting saccades on the dominant camera (left camera by default) when needed. At the same time, the vergence component triggers correcting saccades on the slave camera (right camera by default) in order to maintain proper alignment with the attended position. To drive the robot towards the target position, its world coordinates are passed to a third component that computes and follows planar trajectories. This trajectories are computed using Bézier curves to match the initial and a final orientation of the robot. Third degree Bézier polynomes are very simple curves and easy to use when an initial and final orientation have to be specified.

This state is iterated with the previous one to obtain new visual information on the 3D position of the candidate object.

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

We run here a rapid test to accept or discard the candidate object and, at the same time, estimate its orientation on the floor. We do this in two stages. First, we activate the rather computationally heavy component, Vdetector, that computes texture descriptors on the region of interest extracted by the VisionComp component (see graph of components in Figure 1). By selectively activating this component we save computational resources that may be needed in other stages of the task. Then, we match this 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. To achieve this we calculate the histogram of gradients of the bounding box surrounding the candidate object. The main mode of the histogram gives us the orientation of the pallet on the floor. This completes the estimation of the initial pose of the pallet and triggers the beginning of the next state.

4.5 Refine object pose estimation

On entering this state, the robot believes that he is taking a close look at a pallet of which he approximately knows its pose. The next step is a new validation test on the "palletness" of the object placed right ahead and, in case it succeeds, a finer estimation of its pose. We start with a previously stored 3D model of the pallet, a wire frame drawing of real dimensions. Using the OSG graphics engine that runs inside the ForkliftComp component and the already mentioned InnerModel object (continuously updated representation of the state of the robot), it is easy to render the virtual pallet placed at the estimated pose and through a dominant virtual camera situated in its corresponding place in the virtual robot. This virtual image is subtracted from the real image using a RGB euclidean metric. These result is converted to grayscale and binarized using an adaptive threshold. Finally, all white pixels are counted to obtain a score. This value must be greater than a predefined threshold in order to accept the object as a real pallet. If this test succeeds the procedure is iterated varying the x, y and alfa coordinates of the pose of the pallet in a small range. The pose that minimizes the sum of white pixels is selected as the new estimated pose, if this value is smaller than the original one. This loop is repeated several times with increasingly smaller ranges in pose dimensions, until no further improvement is obtained or a maximum amount of time is elapsed. En example of a final estimate pose is shown in Figure 5

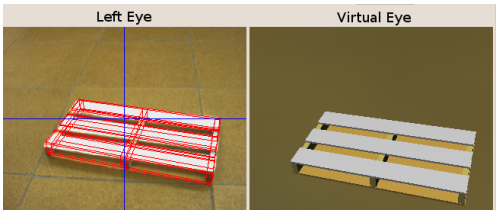

Figure 5: Final estimated pose of the pallet

4.6 Final approach and pick up operation

Once a good estimate of the pose is obtained, this last state drives the robot towards the pallet along the appropriated trajectory. At the end, the forklift should enter smoothly through the pallet openings. Four infrared sensors placed in the forklift arms and pointing upwards, two in each one, send a signal to the component when they are occluded by the pallet. This information triggers the lifting behaviour that is performed by the Fork component.

5 Experimental results

The experiment has been conducted using the RobEx platform[5]. In the experiment the floor is assumed to have a rather homogeneous texture and different color from the surrounding walls. Figure 6 shows a sequence of six pictures in which the robot detects, approaches, recognizes and estimates the pose, manoeuvres and, finally, picks the pallet. An accompanying video shows a run of the experiment in which the relative orientation of the pallet with respect to the robot is quite significant. This situation forces a final manoeuvre to gain a correct final position.

6 Conclusions and further work

In this paper we have described an experiment designed to study the problem of sequential integration of behaviours in a real manipulation task conducted by a mobile robot. One difficult aspect has an architectural nature and deals with the hierarchical and distributed structure of the resulting software system. In order to build complex behaviours 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 behaviour. 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. With this enhancement, we have instantiated a generic manipulation for a mobile robot using a forklift.

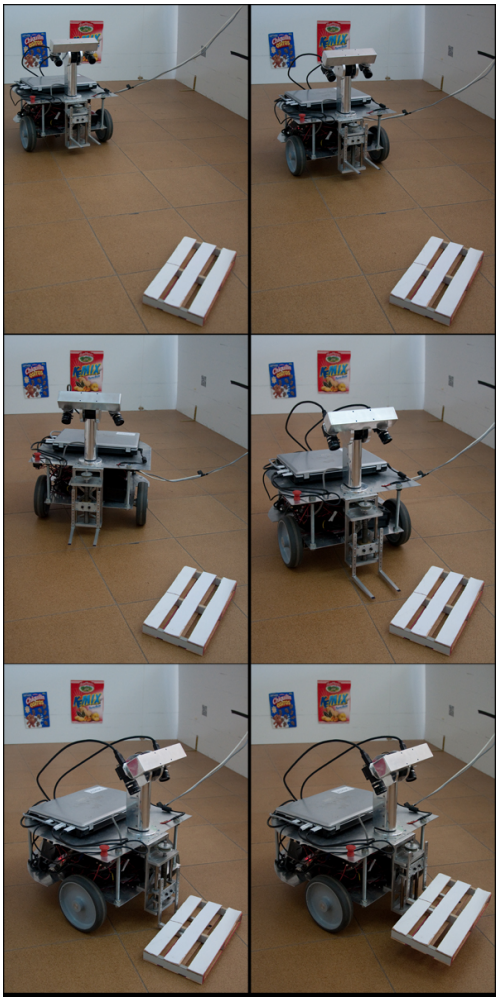

Figure 6: "RobEx picking a pallet"

Acknowledgments

This work has been supported by grant PRI09A037 and GRU09064, from the Ministry of Economy, Trade and Innovation of the Extremaduran Government, by grant TSI-020301-2009-27, for the ACROSS project, funded by the Ministerio de Industria, Turismo y Comercio (AVANZA2) and the European FEDER program and by grant PDT9A044 for the project Escáner móvil robotizado funded by the Consejería of Economía, Comercio e Innovación de la Junta de Extremadura.

References

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

[2] Brooks A, Kaupp T, Makarenko A, Williams S, Oreback A. *Orca: A Component Model and Repository*. Software Engineering for Experimental Robotics, Springer, pp. 231-251, 2007.

[3] Harel D. *Statecharts in the Making: A Personal Account* Communications of the ACM Vol 52 Nº 3, March 2009

[4] Manso L, Bustos P, Bachiller P, Cintas R, Calderita L, Núñez P. *RoboComp, not another robotics middelware* Submitted to IROS 2010, Taiwan

[5] Manso L, Bustos P, Bachiller P. *Multi-cue Visual Obstacle Detection for Mobile Robots*. Journal of Physical Agents. 2010;4(1):3-10.

[6] Qt State Machine Framework *http://doc.trolltech.com/solutions/4/qtstatemachine/*

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

[8] RoboComp, *http://robocomp.sourceforge.net*, 2009.

[9] Seelinger M, Yoder J. *Automatic Pallet Engagment by a Vision Guided Forklift*. Proceedings of the 2005 IEEE International Conference on Robotics and Automation. 2005;(April):4068-4073.

[10] Wasik Z, Saffiotti A. *A hierarchical behavior-based approach to manipulation tasks*. In Proceedings of the IEEE International conference on robotics and automation.Vol 2. Pag: 2780–2785. 2003