UNIVERSITÀ DEGLI STUDI DI BRESCIA
FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA
TESI DI LAUREA SPECIALISTICA

ANALISI DI FATTIBILITÀ ED IMPLEMENTAZIONE DI UN SISTEMA DI ROBOTICA COGNITIVA PER COMPITI DI NAVIGAZIONE

(Analysis of preconditions and implementation of a Cognitive Robotic System for navigation tasks)

Relatore:

Ch.mo Prof. Riccardo Cassinis

Correlatore:

Ch.mo Prof. Marco Ragni

Laureando:
Francesco Bonfadelli
Matricola 83174

Anno Accademico 2012-2013

Contents

List of Figures

1. State Of The Art

This chapter, after having introduced the concept of cognitive architecture, describes the main working instrument used for this work: ACT-R, the cognitive architecture and OpenCV, the computer vision library.

1.1. Cognitive Architecture

A cognitive architecture is the implementation through computer simulation programs of a theory about human cognition. The theory generally relies on a wide selection of human experimental data.

"A cognitive architecture is not a single algorithm or method for solving a problem; rather, it is the task-independent infrastructure that brings an agent's knowledge to bear on a problem in order to produce behavior" [JEL12]. A cognitive architecture alone can not solve any problem. In order to be able to do it, it needs to be supplied with the knowledge to perform it. The combination of an architecture and the knowledge necessary to solve a task is called a *model*. It is possible to create many models able to solve the same task. The specific knowledge is determined by the *modeler*. [Sea02]

In addition, they could include "learning mechanisms that populate the agent's memories based on experience" [JEL12].

built aggregating many software modules,

1.2. ACT-R

ACT-R, that stands for *Adaptive Control of Thought-Rational*, is a cognitive architecture, that is the implementation of a theory about how human cognition works.

The aim of such architecture is to model the behaviour of the human brain. From an external point of view, ACT-R can be intended as a programming language; however its internal structure is based on assumptions about human cognition. Psychologists who research and study the human cognition established the assumptions running numerous

experiments. [New94] Differently from other cognitive architectures, ACT-R allows the researchers to collect quantitative measures, as, for example, time needed to perform the task or accuracy of the task, and to compare them directly with the quantitative measures obtained from the human beings who partecipates to the experiments. For each task the researchers write a program in ACT-R, called *model*, adding their own assumptions related to that task. A model simulates the behaviour of a cognitive agent. The approach based on measuring and comparison leads to very effective models. For example, it has been shown that ACT-R can successfully predict *Blood Oxygen Level Dependence* of several parts of the brain, through experiments with *Functional Magnetic Resonance Imaging*. Up to now ACT-R has been used successfully to create models in many different areas, the most important of which are learning, problem solving, communication and perception. ACT-R's most important assumption is that human knowledge can be divided into two irreducible kinds of representations: declarative and procedural. Declarative memory refers to all the kind of memory that can be consciously recalled, like the shape of an object or its color. Procedural is referred to actions that can be done unconsciously. The most common way used to explain how these two kinds of memory work is the typewriter example. A skilled typewriter can write quickly without looking at the keyboard, putting his fingers in the right place and pushing them in the correct order. If we ask to the same typewriter where a certain character is positioned in the keyboard, he will probably answer that he is not sure about that without looking at it. In this example the procedural memory is stored as the action of "putting the right finger in the right place" when the typewriter has the goal to "write a certain character".

e la memoria dichiarativa?

    In ACT-R theory the declarative memory is represented by the so called *chunks*, data structures defined as a n-tuple (slots) of arbitrary size. The n-tuples and their values are specified in the *chunk-type*, which defines also (if any) the initial value of the slot. Every chunk has also a name, used to reference it, but that is not considered to be a part of the chunk itself. The chunks and their slots are used to implement the mechanisms of ACT-R's theory, even if the slots are only software construct and not part of the theory itself. The chunk-types can be organized into hierarchies.

    The ACT-R's counter part of the human procedural memory are the *productions*. Productions are the ACT-R equivalent of functions. They define sequences of actions and can be fired only if a set of preconditions is satisfied.

    All the activities carried out by the brain, i.e. talking or moving, are performed by neurons located close together, in a well defined and limited area of the cortex. Similarly each module of ACT-R represents a function of the human brain. In the ACT-R's architecture the task of the procedural module is to coordinate all the other modules. The exchange

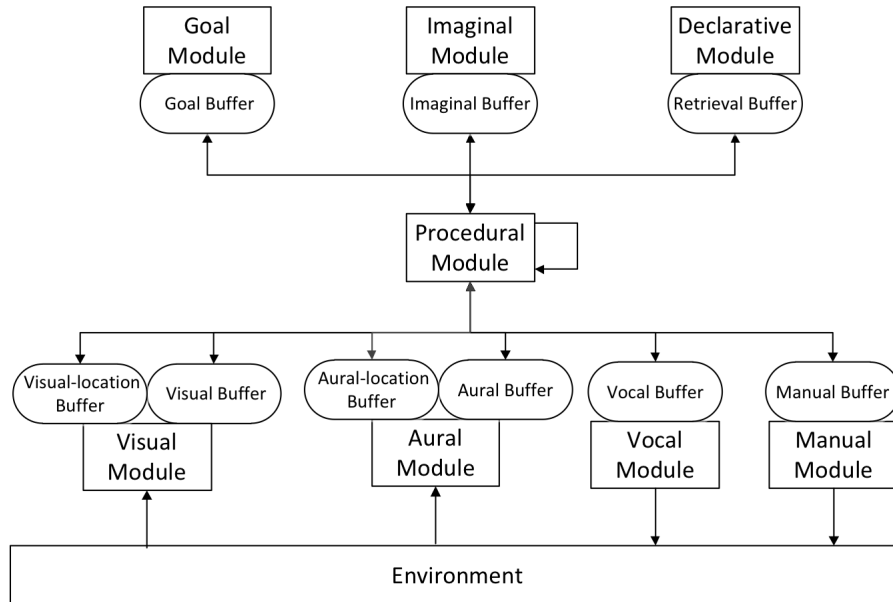of information between the modules is achieved through *buffers*.

Figure 1.1.: Structure of Act-r.

Each module is independent from the others. To communicate one to each other they need to pass through the procedural module. As shown in Figure 4.2, a module can have more than one buffer; for example, the aural module has two buffers: the first is the "Aural-location Buffer", whose function is to get the focus about changes happened in the environment; the second is the "Aural Buffer", to get the real information about the environment.

The buffers are the built in interface between modules, used to exchange chunks between them. Every buffer belongs to a module, while a module can also have no buffers. Each buffer can hold one chunk at a time, which is readable by the other modules, the guidelines say that a buffer can be written only by its owner.

The modules usually work in a parallel way, but their interactions, which are limited to a serial exchange of chunks, can be only serial. There are two reasons for this limitation: firstly the structure of the buffers can hold only one chunk; secondly only one production whose preconditions are satisfied can be fired. When the preconditions of more than one production are satisfied, the one to be fired is the one with the higher priority value, called *utility value*. This is a numeric quantity, it can be associated with each production in advance or it can be learned while the model runs.

From an informatics point of view ACT-R is a software written in Lisp and its models are written in a Lisp-like language. It is thought to have a modular structure so that

it can be easily extended. ACT-R 6.0 is the implementation of a state of the ars theory of human cognition, developed by John R. Anderson. Anderson credits Allen Newell as source of influence to his theory.

> mancano le fonti pratica-mente ovunque

1.3. OpenCV

OpenCV, an abbreviation that stands for *Open Source Computer Vision*, is a computer vision library that was originally developed by Intel and, later on, by Willow Garage. It is a cross-platform library, released under a BSD license, thus it is free and open source. In the beginning it was developed in C and C++ and afterwards it was expanded by the addition of interfaces for Java and Python. OpenCV is designed for computational efficiency and with a strong focus on real-time applications. The version 2.4 has more than 2500 algorithms. The library has been used in many applications as, for example, mine inspection and robotics [Ope12b]. The following sections contain a brief history of the library and a list of its main features.

History

The OpenCV Project started in 1999 as an Intel Reasearch initiative aimed to improve CPU intensive applications as a part of projects including real-time ray tracing and 3D display walls. The early goals of the project were developing optimized code for basic vision infrastructure, spreading this infrastructure to developers and making it portable and available for free, using a license that let the developers create both commercial and free applications.

The first alpha version was released to the public in 2000, followed by five beta versions between 2001 and 2005, which lead to version 1.0 in 2006. In 2008, the technology incubator Willow Garage begun supporting the project and, in the same year, version 1.1 was released. In October 2009, OpenCV 2.0 was released. It includes many improvements, such as a better C++ interface, more programming patterns, new functions and an optimization for multi-core architectures. According to the current OpenCV release plan, a new version of the library is delivered on a six-months basis. [Ope12a].

Main Features

OpenCV offers a wide range of possibilities. First of all, it provides an easy way to manage image and video data types. It also offers functions to load, copy, edit, convert and store images and a basic graphical user interface that lets the developers handle

keyboard and mouse and display images and videos. The library lets manipulate images even with matrix and vector algebra routines. It supports the most common dynamic data structures and offers many different basic image processing functions: filtering, edge and corner detection, color conversion, sampling and interpolation, morphological operations, histograms and image pyramids. Beyond this, it integrates many functions for structural analysis of the image, camera calibration, motion analysis and object recognition. [Aga06].

2. Requirements

2.1. Functional Requirements

The final objective of this work is to make easier the creation of an ACT-R test.
At the moment, a psychologist, in order to create a test, has to create two interfaces, one for the user and one for ACT-R. The user interface is quite simple, it is formed by simple shapes that can have different colours and words. The user has to read or watch the screen and then choose between a series of possibilities. In order to be analysed by ACT-R, this interface must have a corresponding one, which is simpler and contains only the most important features of the first one. This "simpler" interface is given as input to ACT-R and is written according to a fixed pattern. The figure below shows an example of the user interface of a test and of its corresponding one.

The purpose of this work is to create a module which is able to receive as input the user interface, recognize the main features in it and create the corresponding ACT-R interface, which must contain only such features. In order to do this, the software must be made by two parts. The first part will analyze the image. It will recognize and distinguish simple shapes, such as squares, rectangles, circles, stars, arrows; distinguish the colours of the objects; determine the position of each object in the image and recognize the text in the image. The second part will create the correspondent interface for ACT-R, thus...

The software is going to be tested with several ACT-R test.

2.2. Non Functional Requirements

3. Development Process

3.1. SCRUM

3.2. Working Instruments

In addition to ACT-R and OpenCV, many other tools have been used. Here follows a list of the most important ones.

- **Eclipse IDE for C/C++ Developers** as *integrated development environment* [1];

- **Cute** as *unit testing framework* [2];

- **Mylyn** as *task and application lifecycle management* [3];

- **Trac** as *bug tracking system* [4];

- **Git** as *version control system* [5];

- **Dia** and **cpp2dia** to create UML diagrams [6];

- **ZBar** to read QR codes [7].

---

[1]More informations at www.eclipse.org
[2]More informations at http://cute-test.com
[3]More informations at www.eclipse.org/mylyn
[4]More informations at trac.edgewall.org
[5]More informations at git-scm.com
[6]More informations at dia-installer.de and at cpp2dia.sourceforge.net
[7]More informations at zbar.sourceforge.net

4. Design

The following chapter describes the architecture of the software. The software design is supported by diagrams to better explain the contents and the structure of the project itself. The following sections, after having given a brief description of the whole architecture of the sofware, describe the class hierarchy, the . terminare...

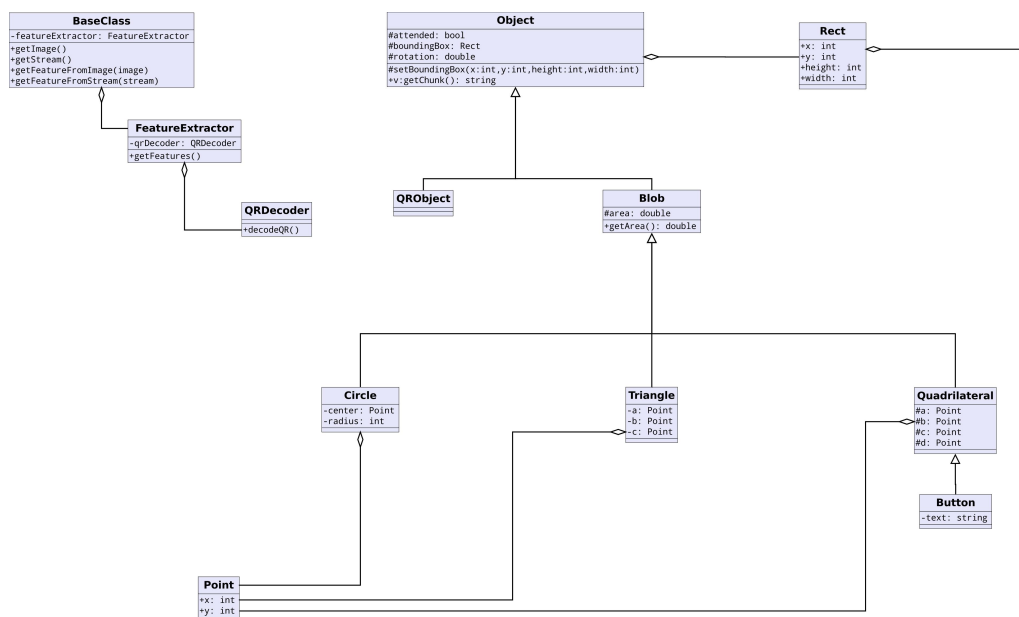## 4.1. The Whole Software

The aim of the .

Figure 4.1.: Class diagram of the whole software

cambiare l immagine e aggiungere la descrizione...
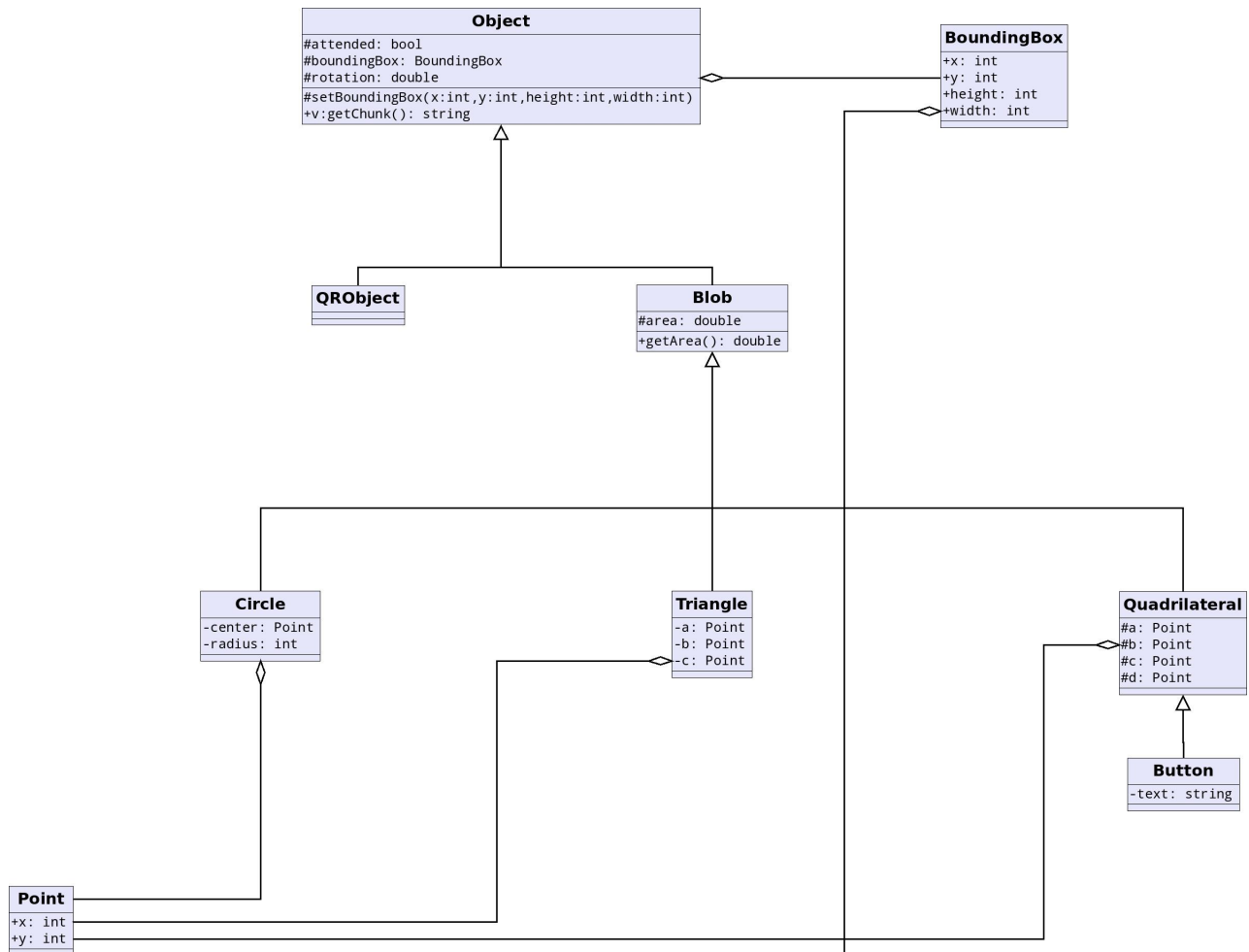
4.2. The Class Hierarchy

Figure 4.2.: Class hierarchy of the recognized objects

The picture above describes the hierarchy of the classes defined to contain the objects detected in the images. The *Object* class represents a generic object that can be found in an image. Thus, everything which is different from the background can be seen as an instance of the *Object* class. Every object must be bounded by a *bounding box*. This structure is represented by the *BoundingBox* class. All the bounding boxes have rectangular shape. The requirements were such that it was not necessary to define more complicate shapes . The *attended* attribute of the *Object* class is set to *true* if the object has been already returned to ACT-R, otherwise it is set to *false*. The *rotation* is a value that gives the amount of the rotation in the counterclockwise directions starting from the horizontal direction . . Besides the generic objects, there are two categories of object that can be found in the processed images, the *QR codes* and the *simples shapes*. A simple shape can be, for example, a circle, a square, a rectangle or a triangle. The QR codes is represented

migliorare questa frase, fa schifo

controllare la correttezza

vedere se il metodo getChunk esiste ancora

in the hierarchy with the *QRObject* class, the generic shape with the *Blob* class. This class has the *area* parameter, which stands for the area of the object in pixel, and the *getArea* method, which returns this value. The *Blob* class is realized by three classes, *Circle*, *Triangle* and *Quadrilateral*. The first one has, as parameters, the *center* and the *radius* of the circle itself. The parameters of the *Triangle* and the *Quadrilateral* classes are respectively three and four points, which represent the vertices of the shape. Notice that the *Quadrilateral* class defines every polygon with four sides and four vertices. Thus, rectangles and squares can be instances of this class. The *Button* class is a realization of the *Quadrilateral* class. This is because, as a requirement, buttons have always rectangular or squared shapes. The additional information they add is the *text*, that is a simple message that is always present in a button. It is represented by the *text* attribute. The *Point* class identifies a generic point in the bidimensional space. Its parameters, *x* and *y*, are the two coordinates in the plane.

... aggiungere, correggere o finire

5. Implementation

References

[Aga06] Gady Agam. Introduction to programming with OpenCV. Technical report, 2006.

[JEL12] John L. Tishman John E. Laird. The soar cognitive architecture. *AISB Quarterly*, 134:1, 2012.

[New94] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, 1994.

[Ope12a] Opencv change logs, 2012. Available on line.

[Ope12b] Opencv web page, 2012. Available on line.

[Sea02] Andrew Sears. *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*. Lawrence Erlbaum, 2002.