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

1. State Of The Art **4**
1.1. Cognitive Architecture 4
1.2. ACT-R 5
1.2.1. Declarative memory and procedural memory 5
1.2.2. Chunks and productions 6
1.2.3. Architecture 6
1.2.4. Applications 8
1.3. OpenCV 8

2. Requirements **10**
2.1. Functional Requirements 10
2.2. Non Functional Requirements 10

3. Development Process **11**
3.1. SCRUM 11
3.2. Working Instruments 11

4. Design **12**
4.1. The Whole Software 12
4.2. The Class Hierarchy 12

5. Implementation **i**

References **ii**

List of Figures

1.1. *Structure of ACT-R.* 7

4.1. *Class diagram of the whole software* 12
4.2. *Class hierarchy of the recognized objects* 13

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 softwares of a theory about human cognition. The theory generally relies on a wide selection of human experimental data. The design of these architectures tries to simulate human intelligence in a humanlike way.

controllare correttezza di softwares

"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 specific task is called *model*. It is possible to create many models able to solve the same task. The specific knowledge is determined by the *modeler* [Sea02].

Another important feature of cognitive architectures is that they give the possibility to make comparison between the sequence of actions produced by the model and the ones produced by human beings when they try to solve the same task. In addition, the measured quantities can be not only qualitative ones, like the correctness of the goal, but also quantitative, as for example, the time necessary to complete a task. This gives the models the possibility to produce execution times, error rates and learning curves. Comparisons with human performances can be useful to evaluate the quality of model [Sea02].

As the term *infrastructure* suggests, cognitive architectures usually are built aggregating many software modules, most of which represent functions of the human brain. Anyway, there exist other modules, which coordinate the overall functioning and without which the whole architecture could not work. In the following section, which describes *ACT-R*, the

procedural module is an example of such modules. In fact, it does not represent a function of the human being, though it is necessary to coordinate the communication between all the other modules.

Some architectures, can, in addition, include some *learning mechanisms*. This can be the attempt to simulate the human memory system, thanks to which the behaviour of a human can be different after having experienced facts or consequences of a specific choice [Sea02].

1.2. ACT-R

ACT-R, that stands for *Adaptive Control of Thought-Rational*, is a cognitive architecture that implements the homonym theory developed by John Robert Anderson, professor of psychology and computer science at Carnegie Mellon University. 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. The current version of the software is the 6.0.

The following section describes the differences between declarative and procedural memory. This is important because it is the basic theory on which ACT-R is founded. Then, after the definition of chunk and productions, the building blocks of ACT-R structure, you can find an overview on the architecture of the framework and, after this, the most important applications in which the framework has been used.

1.2.1. Declarative memory and procedural memory

In psychology, *memory* is defined as the processes by which information is encoded, stored and retrieved [BEAA09].

ACT-R's most important assumption about knowledge is based on Anderson's theory about memory. Anderson divides memory into *declarative* and *procedural*.

Declarative memory refers to all the information that can be consciously recalled. This kind of knowledge comprehends facts and notions that human beings explicitly know. To call back this kind of information, there must be a conscious process by the human being. For this reason, this kind of memory is also called *explicit*.

In contrast, procedural memory refers to all that notions or skills that human beings have but which they learnt in an implicit way. Examples of this knowledge are, for example, driving, reading and writing. In this case, in order to call back this kind of information, the human being does not need a conscious process. That is why this kind

of memory is also called *implicit* [And76].

The following example is used to explain better how these two kinds of memory work. When a person starts learning typewriting, an attempt he can make in the beginning is trying to memorize the layout of the keyboard. The aware knowledge of all the positions of the keys is the declarative memory. After having become a skilled typewriter, the same person will write quickly putting his fingers on the right keys and pushing them in the correct order, without thinking anymore about the positions of the keys on the keyboard. Moreover, if we ask him where the position of a certain character is on the keyboard, he will probably answer that he can not say it without looking at it. This is because, now, for this task he is using his procedural memory [And93].

1.2.2. Chunks and productions

In ACT-R, declarative memory is represented by structures, called *chunks*, and procedural memory by rules, called *productions*. Chunks and productions are the basic building blocks of an ACT-R model [Bot].

The *chunks* are data structures which are defined by their *type* and their *attribute list*. This is a tuple of pairs, each of which is made up by a fixed part and a variable part. The fixed part is the *name* of the attribute and is called *slot*. The variable part is the *value* of the attribute. Each chunk has also a *name* but it is not considered to be a part of the chunk itself, as it does not exist in ACT-R theory. It is used only for convenience to reference the specific chunk when writing models. The chunk-types can be organized into hierarchies [Bot].

The *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. They can be represented as *if-then* rules, where the *if-part* is a set of conditions that must be true for the production to apply and the *then-part* is the action of the production and consists of the operations the model should perform when the production is selected and used. In general there could be some conflicts between productions. This happens when preconditions of two or more productions are satisfied at the same time. In these cases the production to be fired is the one with the highest *utility value*. This is a numeric quantity which gives a priority measure. It can be set a priori by the modeler or learnt while the model is running [Bot].

1.2.3. Architecture

All the activities carried out by the human brain, like talking or moving, are performed by neurons located close together in a well defined and limited area of the cortex. Trying

to imitate this "architecture", ACT-R's framework is structured in different *modules*, each of which represents one specific function of the human brain [Bot].

Figure 1.1 shows the modular structure of ACT-R. In the picture you can see two groups of modules, separated by the *procedural module* [Bot].

The first group comprehends *visual*, *aural*, *manual* and *vocal modules*. These let the model interact with the environment. The *visual module* is responsible for recognizing objects in the visual scene and shifting the focus to them. Similarly, the *aural module* identifies sounds and moves the attention to them. The *manual module* can move the virtual hands and perform actions like pressing the key on a keyboard or moving the mouse while the *vocal module* controls the virtual voice [Bot].

The other group comprehends *goal module*, *imaginal module* and *declarative module*. These represent the internal information of the model. The *goal module* provides the system with the structure of the goal of the task, defined as a chunk. The *imaginal module* has to contain and update the current context relevant to the current task. The *declarative module* provides the model with a declarative memory, thus it stores the declarative chunks generated by the model and provides a mechanism for retrieving them [Bot].

Finally, the *procedural module* is responsible of the communication and the coordination of all the other modules [Bot].

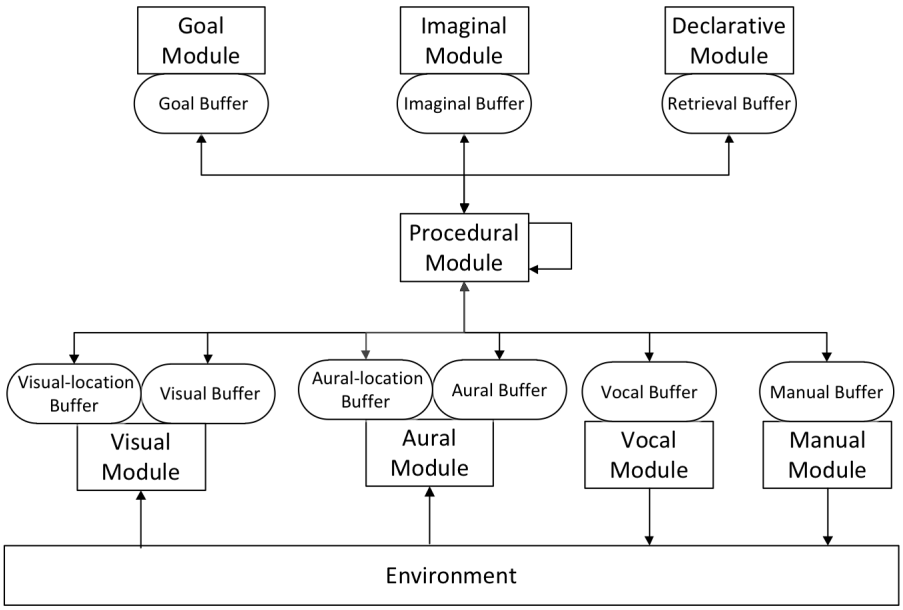

Figure 1.1.: Structure of ACT-R.

In fact, modules are independent of each other, they do not share variables or information. They can communicate with each other thanks to the *buffers*, which represent

the interfaces of a module towards the others. A module can have no buffers as well as one or more than one. The communication consists in exchanging chunks. Each module can read chunks from every buffer but it can make changes only to the chunks in its own buffers. Moreover each buffer can hold one chunk at a time [Bot].

Although modules usually work in a parallel way, their interactions can be only serial. There are two reasons for this limitation: the first one is that the structure of the buffers can hold only one chunk and the second one is that only one production can be fired at a time [Bot].

espandere e aggiungere fonti

1.2.4. Applications

Up to now, it has been used in many different areas, the most important of which are problem solving, communication and perception. . 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*.

aggiungere fonti

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 .

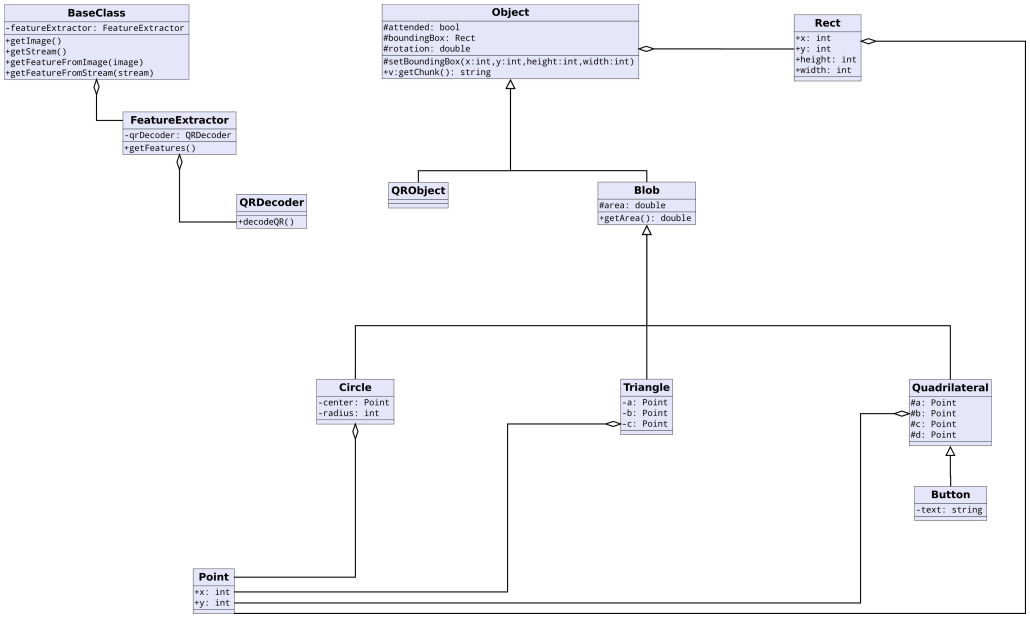

Figure 4.1.: Class diagram of the whole software

cambiare l immagine e aggiungere la descrizione...

4.2. The Class Hierarchy

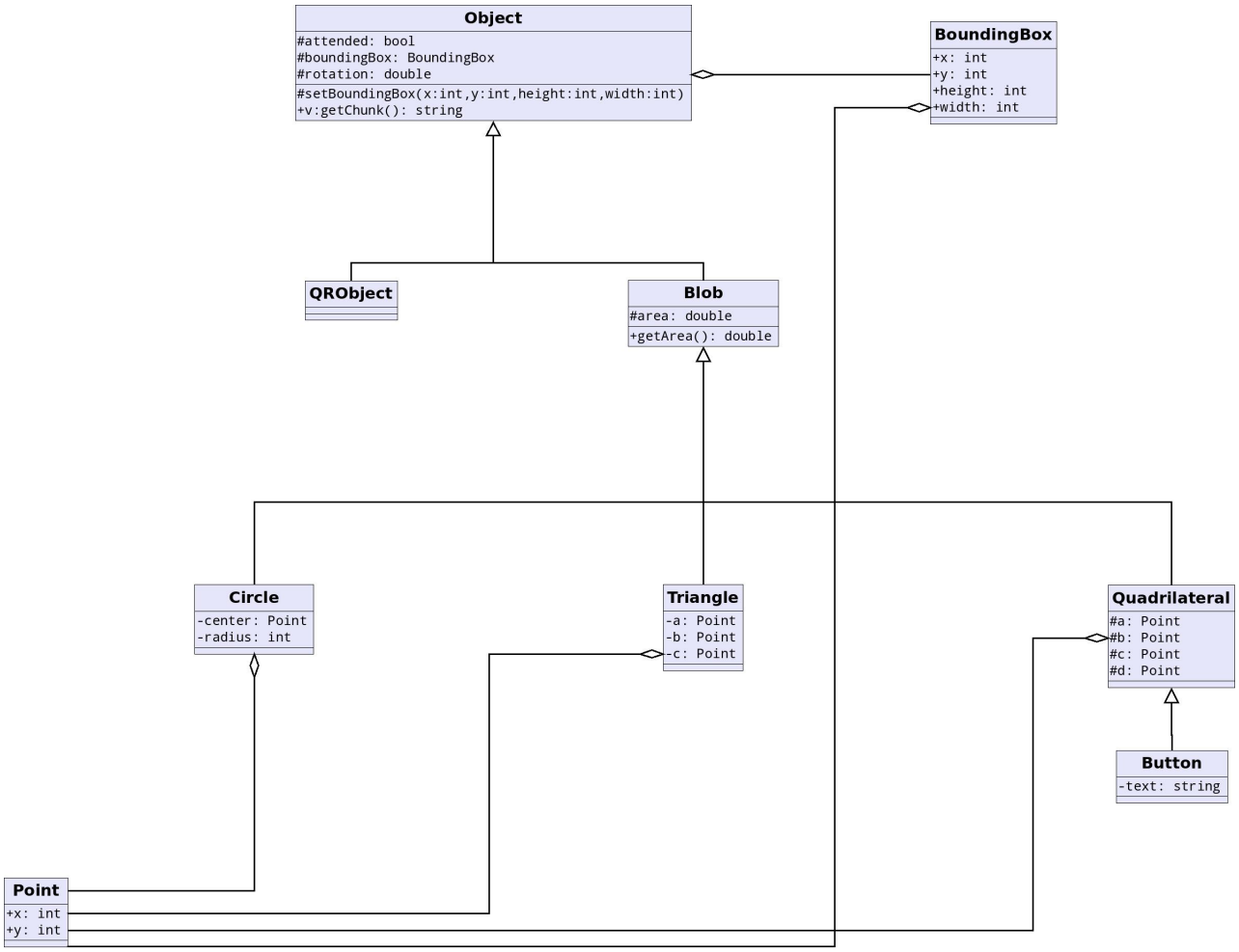

Figure 4.2.: Class hierarchy of the recognized objects

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.

[And76] J.R. Anderson. *Language Memory Thought*. The Experimental Psychology Series/ Arthur W. Melton consulting ed. Taylor & Francis Group, 1976.

[And93] J.R. Anderson. *Rules of the Mind with Mac Dis*. Taylor & Francis Group, 1993.

[BEAA09] A.D. Baddeley, M.W. Eysenck, M.C. Anderson, and M. Anderson. *Memory*. Taylor & Francis Group, 2009.

[Bot] Dan Bothell. *ACT-R 6 Reference Manual*.

[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.