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. Introduction

1.1. Organization of the document

1.2. The Objective

2. 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.

2.1. 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 *chunks* and *productions*, the building blocks of ACT-R structure, you can find an overview on the architecture of the framework.

2.1.1. 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].

o to better explain?

2.1.2. 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 [Bota].

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 [Bota].

o in?

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 [Bota].

2.1.3. 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 [Bota].

Figure 2.1 shows the modular structure of ACT-R. In the picture you can see two groups of modules, separated by the *procedural module* [Bota].

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 [Bota].

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 [Bota].

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

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 [Bota].

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 at a time and the second one is that only one production can be fired at a time [Bota].

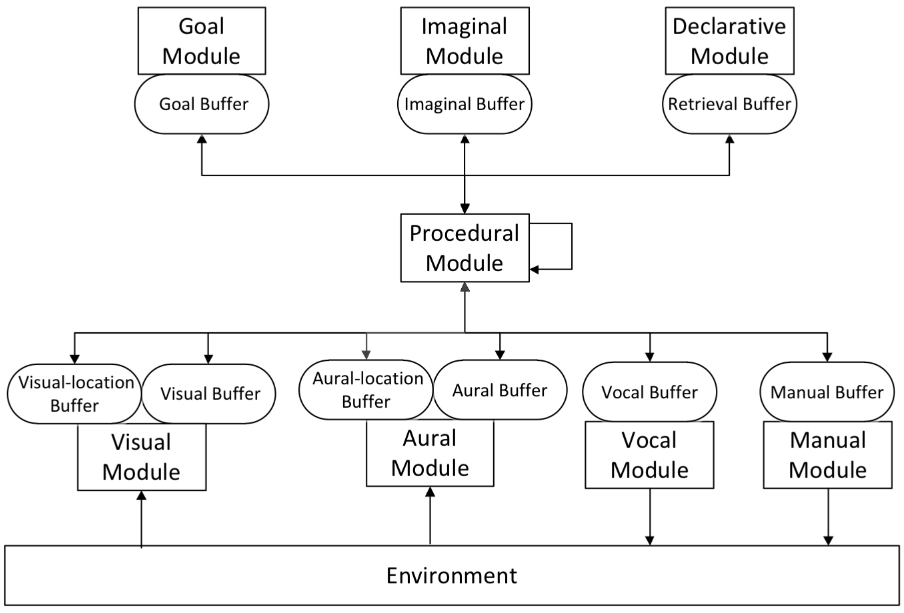

Figure 2.1.: Structure of ACT-R.

2.2. 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].

perchè non va bene?

3. Objective

3.1. How psycologists define the experiments with ACT-R

3.2. The requirements

4. Development Process

This chapter, after having introduced the agile development framework SCRUM, describes the development process adopted by the team in order to complete the work.

4.1. SCRUM

SCRUM is a framework for the agile management of the project development. As such, it does not define the technical way in which the developers must do the job but it follows the development process. The framework has many dimensions: roles, events, rules and artifacts within the framework serve specific purposes and contribute the Scrum's success.

The following sections describe one by one these components. The first section describes the main roles of people within the framework, the second focuses on the events and the meetings necessary for the implementation of the methodology and the third one puts in evidence the artifacts needed.

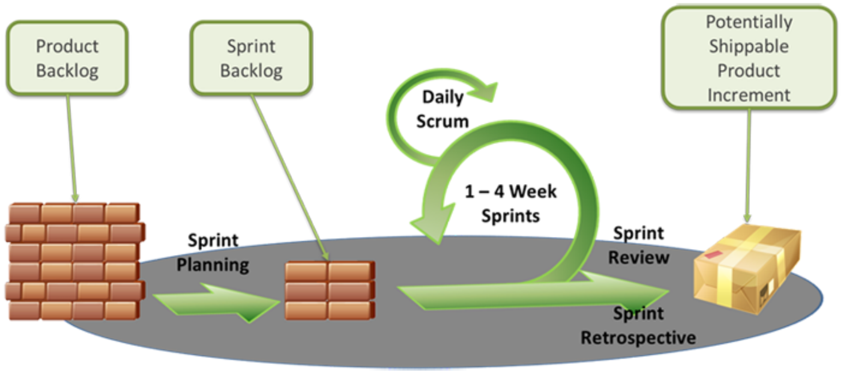

Figure 4.1.: Overview of the Scrum Process

4.1.1. The Scrum Team

The *Scrum Team* is composed of a *Product Owner*, the *Development Team*, and a *Scrum Master*. These figures have different roles but all of them have to reach the same goal, deliver an increment part of usable product at constant time intervals. Scrum Teams are self-organizing and cross-functional. Self-organization gives the members the possibility to choose how best to accomplish their work, without being directed by others outside the team. Cross-functionality gives the team all the competencies needed to accomplish the work without depending on others not part of the team. The delivery of the products is iterative and incremental. This fact guarantees a constant feedback on the correctness of the job and ensures that a potentially useful version of the working product is always available [Botb].

The following paragraphs will describe each of the three roles.

Figure 4.2.: The Scrum Team

Product Owner

The *Product Owner* is the person who represents the stakeholders of the product. The Product Owner can represent the will of a committee but must be one person. His role is to define the requirements of the new versions of the product, assign them the priorities, explain them to the team in detail and is responsible for the performance of the team. The requirements are called *Backlog Items* and are included in the *Product Backlog*. This document is described in more details in section 4.1.3 [Botb].

Development Team

The *Development Team* consists of professionals who have to add the new functionalities to the product. The team is usually composed of a number of member which varies from three to nine. One team must be self-organizing, this means that only the members can decide the step by step tasks to be accomplished in order to add functionalities to the product. Every team is also cross-functional, i.e. is composed by people who have different skills. In this way it can be autonomous and it does not have to depend on other people outside the team to accomplish its job. More the Development Team's synergy is, more optimized its overall efficiency and effectiveness are [Botb].

Scrum Master

The *Scrum Master* has the role to verify that Scrum is understood and put in place. He or she does this checking that everyone in the team follows Scrum theory, practices, and rules. The Scrum Master is the enforcer of the rules and interacts with all the people inside and outside the Scrum Team in order to teach which interactions are useful and which are not. On one side, he helps the Product Owner finding techniques for managing the *Product Backlog*, teaching him how to communicate in clear way with the Development Team and in understanding and practicing agility. On the other side, he coaches the Development Team in self-organization and cross-functionality, he protects it from unhelpful interruptions and keeps it focused on the tasks. For this, often this role is referred as a servant-leader for the Scrum Team [Botb].

4.1.2. Events

Prefixed meetings in Scrum have the purpose to create regularity and hence to minimize the need for meetings not defined in Scrum, which can distract the members of the team from their tasks. In this events, different roles in the Scrum Team can interact with regularity, without the need for continuous interruptions. All the events are time-boxed, i.e. every event has a maximum duration. This ensures that only the appropriate amount of time is spent planning avoiding time wasting [Botb].

The following sections describe the Sprint, the Sprint Planning Meeting, the Daily Scrum, the Sprint Review and the Sprint Retrospective.

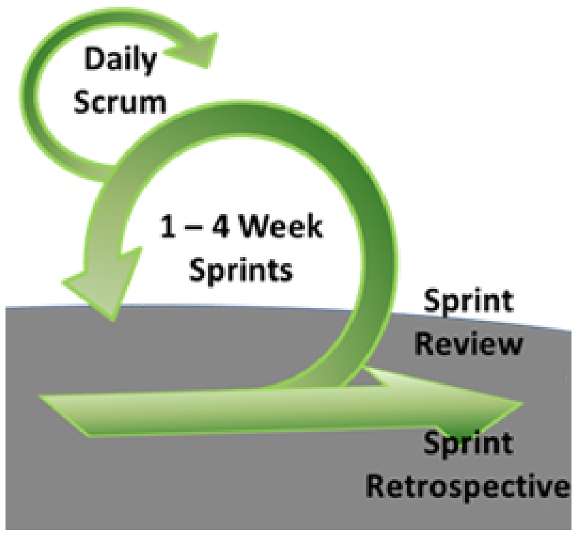

Figure 4.3.: The Scrum Events

Sprint

The *Sprint* is the primary unit of development in Scrum. It is a time-box with a fixed in advance duration. The duration can last from one week to one month. During this period of time the team creates finished portions of a product [Botb].

Each Sprint is preceded by a *Sprint Planning Meeting*, where the Scrum Team defines the tasks for the Sprint and gives an estimation for the Sprint goal. It ends with the *Sprint Review Meeting* and *Sprint Retrospective Meeting*, where the Scrum Team respectively reviews the progress and analyses the mistakes in the Scrum method and proposes solutions in order to avoid it in the next sprint. All these events will be described with more details in the next sections [Botb].

Sprint Planning Meeting

The *Sprint Planning Meeting* is a meeting at which all the member of the Scrum Team participate. It precedes every Sprint. In this meeting the team decides which feature add to the product by the end of the Sprint [Botb].

At first the Product Owner informs the team of the features that he wants to be completely added to the product. Every feature is an item in the *product backlog*, which is described in the section 4.1.3. Then, the Development Team estimates how long it takes to add every new feature to the product and, consequently, how many of them will be added by the end of the Sprint. Often this part of the meeting is signed by a clarification about the requirements between the Product Owner and the Development Team that leads to new time estimation for each goal. The goals are split into tasks, each of which takes no more than two days to be accomplished by the development team. Every task and the plan for delivery them are collected in the *Sprint Backlog*, described in the section 4.1.3 [Botb].

Every Sprint must have a *Sprint Goal*. The Sprint Goal acts like a sort of motivation which reminds the Development Team during the whole sprint which is in a wider context the goal of their activities. For this the sprint goals should not be changed during the sprint [Botb].

Daily Scrum

The *Daily Scrum* is a fifteen minutes event for the Development Team, that allows the team to synchronize the work and plan the next day activities. This purpose is achieved analyzing the work done in the last day and forecasting the work for the next one. During the meeting, each Development Team member explains what he did since the last meeting, what he is going to do before the next meeting and the problems he has to front. The meeting has the purpose to evaluate the progress towards the Sprint Goal and to analyze the trend of the progress in comparison with the Sprint Backlog [Botb].

Sprint Review

The *Sprint Review* is a time-boxed meeting which closes every Sprint. In this meeting the increment of the work is analyzed and, if needed, the Product Backlog is updated. During the meeting, the Product Owner analyzes what has been completed and what has not been completed. The Development Team describes the problem it encountered, how it solved them and shows the new functionalities added to the product. Then the whole team discusses of new opportunities and, more generally, collaborates on what to do next, updating consequently the Product Backlog [Botb].

Sprint Retrospective

The *Sprint Retrospective* is a time-boxed meeting which is fixed after ever Sprint Review. It is an opportunity for the Scrum Team to analyze its scrum implementation and create a plan for the improvements of the next Sprint. During the meeting the focus is set on people, relationships, process and tools. The most important successes are shown and a plan for implementing potential improvements is created. The purpose of this meeting is to make the implementation of the method more effective optimizing the development process and introducing techniques that make the method more productive and enjoyable [Botb].

4.1.3. Artifacts

Scrum's *artifacts* represent the work in many different ways. The basic artifacts required by the framework are the Product Backlog and the Sprint Backlog.

Product Backlog

The *Product Backlog* is an artifact which contains the list of the requirements for a product. The elements of the list, called *Backlog Items*, are sorted by priority.

Every Item must have a name, a description, an estimate and a priority. For each Item, the Product Owner, representing the stakeholder will, sets the priority and the Development Team defines the estimate.

The Product Backlog is dynamic. The earliest versions of it only contain the initially and best understood requirements of the product. Then, as long as the product evolves and new requirements are introduced, it is continuously updated. In this way, it always contains all the features, enhancements and fixes that must be added in the future at the product.

As the higher ordered Items are more important than lower ordered ones, they are clearer and more detailed. The team makes more precise estimates basing on the greater clarity and increased detail. The lower the order of the items, the less detailed are the description and the time estimation.

The activity of adding detail, estimates, and order to the items in the Product Backlog is called *grooming*. The Scrum Team decides how and when doing it. Anyway the grooming activity must not consume more than the 10% of the capacity of the team.

Sprint Backlog

The *Sprint Backlog* contains the *Product Backlog Items* selected for the current Sprint and a plan for completing all the modification to the product and realizing the Sprint Goal. The Sprint Backlog is a forecast about which functionalities should be added to the product by the end of the Sprint and, at the same time, monitors the state of the work during every day of the Sprint [Botb].

The level of detail of the work is such that it can be measured during the *Daily Scrum*. Usually every Product Backlog Item is split into tasks, each of which must be accomplished

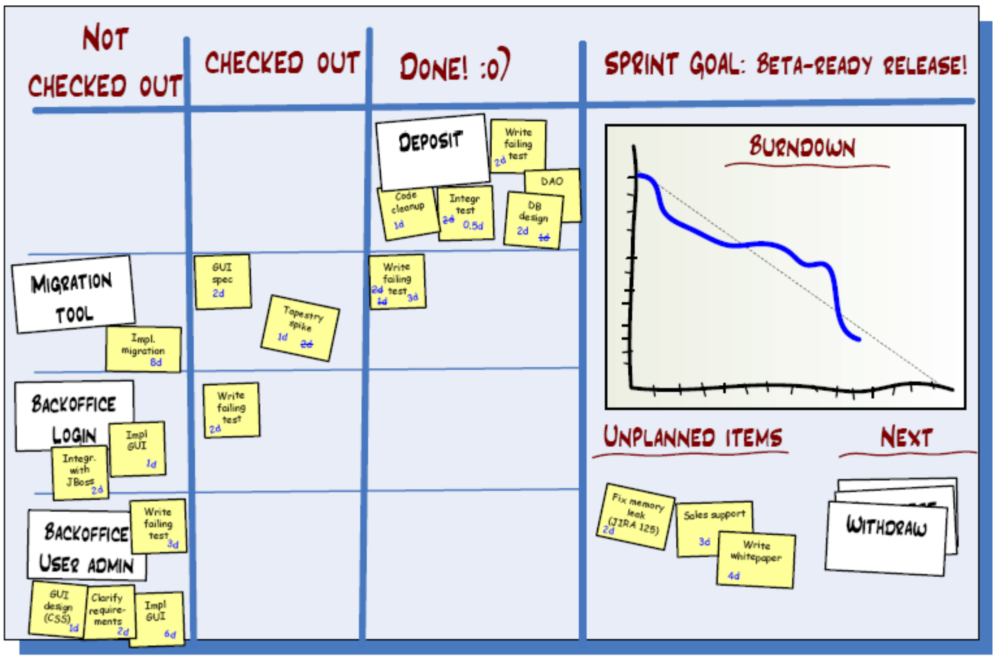

Figure 4.4.: Sprint Backlog and Burn Down Chart

in not more than than sixteen hours of work. The tasks are never assigned; rather, every member of the team takes charge of one of it during the daily scrum, according to the set priority and his own skills.

The Sprint Backlog is updated every day during the *Daily Scrum*. If some new work emerges to be necessary in order to complete some requirements, the Development Team adds it to the Sprint Backlog. If there are some elements which are deemed unnecessary, they are removed.

The state of the ongoing activities can be monitored by a *Burn Down Chart*, which measures the number of completed tasks per day and compares it with an ideal trend, which represents the estimates of the tasks. This chart allows to analyze the accuracy of the estimates and to have a visual representation of the ongoing work [Botb].

4.2. The Adopted Development Process

1) Processo di sviluppo incementale e iterativo 2) No pair programmin nè test driven development (dirlo ?) 3) Sviluppo agile 4) Come abbiamo usato noi SCRUM -) sprint: 2 settimane -) non c'era un committente ma c'era il project manager -) 1 team

5. Design

5.1. Overview of the design

5.2. Class Hierarchy

5.3. Feature Extractors

5.4. Communication with ACT-R

6. Implementation And Testing

6.1. The actual implementation of the software

6.2. COmmunication with ACT-R

7. Conclusions

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.

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

[Botb] Dan Bothell. *Scrum Guide 2011.*

[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.