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].

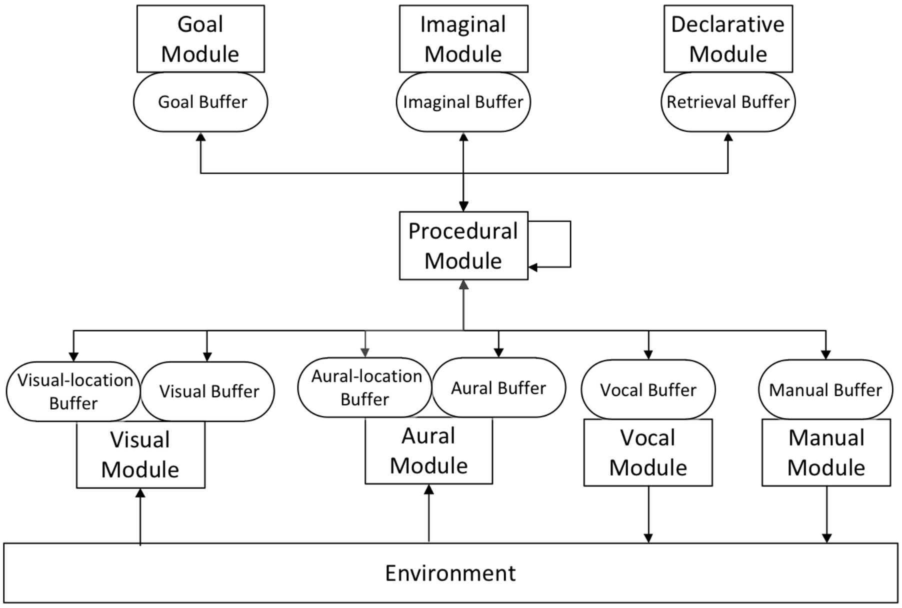

Figure 2.1.: Structure of ACT-R.

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. The Goals of the Software

The software to be designed and developed is a visual module for the cognitive architecture ACT-R, whose aim is to improve ACT-R visual perception in order to make it more similar to the human one.

After a model is developed in ACT-R, it needs to be validated. Model validations usually are realized with experiments in which human beings take part. For every experiment a specific program is created in order to measure some variables as, for example, the time needed to complete a task and the accuracy of the result. The parameters obtained by the model are then compared with the ones obtained in these experiments.

Image 3.1 shows a test case of an experiment in which the goal is to solve some levels of the game *Rush Hour*. As shown, the grid contains some colored rectangles, each of which represents a car. The player must free the red one making it go out of the grid through the exit on the right. The cars can be moved only in the horizontal and vertical directions. The goal of the game is to free the red car in the minimum number of moves and in the shortest possible time.

For each test of the experiment, the ad-hoc program shows the image with the initial configuration and the person has to give the solution of the game by clicking on the rectangles in the order he thinks to be the correct one and the best one to free the red car. As the program never updates the image, the player has to remember the moves he makes. The software manages to interpret the direction of the shift by analyzing the movement of the eye, thanks to a module of *eye tracking*. Moreover, it records the movements of the mouse, the clicks of the mouse, the time needed to give the solution and the correctness of the solution.

The models written in ACT-R that need to process images, like the one used to solve the rush hour game, at the moment skip the image processing step and start directly from a list of objects which is written directly in the input of the model. ACT-R can not accomplish this task because its visual module does not provide functions for the image processing. This fact represents a big limitation for the architecture and leads

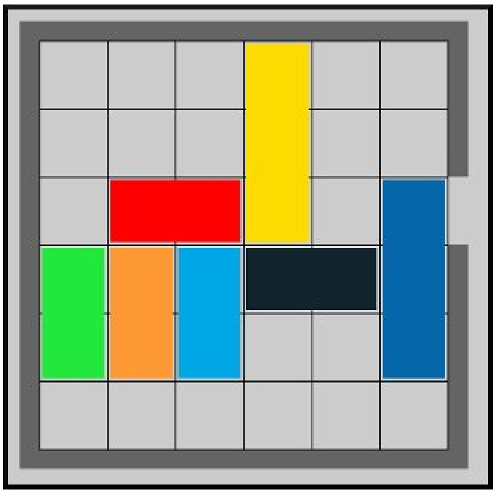

Figure 3.1.: Example of level of Rush Hour game

to a significant gap between the human cognitive system and the framework one. In addition, writing the objects in a model leads to other problems as, for example, the loss of time to write the object for every single test, the consequent delay of the work and the non-scalability of this approach.

The goal of the developed software is to reduce the gap between the human and ACT-R visual systems, being a starting point for a complete tool for the cognitive architecture to recognize objects. To achieve this, the software analyzes the image created for the user, processes it extracting the features needed by ACT-R and stores the information in dedicated data structures. Then, when requested by the cognitive architecture, it communicates to the artificial intelligent framework all the stored information.

and general purpose??

3.2. Requirements

The requirements are grouped in the following sections according to the functional/non-functional classification.

3.2.1. Functional Requirements

The objective of the work is to design and implement a standalone software module which receives as input an image and is able to analyze it and extract some features of it.

The input image is a color image which contains simple shapes. The shapes must not be overlapping and must have the same color hue.

The software must recognize simple shapes, in particular:

- triangles;
- rectangles;
- quadrilaterals;
- circles.

For each shape, it must calculate:

- area;
- perimeter;
- dimensions;
- rotation;
- a rectangular bounding box;
- center;

Moreover, the software has to:

- recognize the color of a single pixel;
- recognize the color of a shape;
- calculate distances between objects;
- make dimensional comparisons between objects;
- calculate the relative position of one object in respect with another one.

The software must be able to communicate with ACT-R. In particular, ACT-R must signal to it which is the image to process and ask for a the features to be extracted. The module must return all the information extracted by ACT-R.

3.2.2. Non-Functional Requirements

The non-functional requirements described in the following subsections are respectively the requirements on the product and the organizational ones.

Product Requirements

The software module must be multi-purpose, portable and must work in background. As multi-purpose is intended that, if possible, it should be easily adapted to be able to work with all the experiments that will be put in place in the future. Moreover, there should be the possibility to adapt the software in order to use it as a shape recognition tool in the navigation with robots. For this topic, see 3.3. As portability is intended that it has to work on more operative systems.

Organizational Requirements

The language of the implementation of the software must be C++. The computer vision library to be used must be OpenCV. The adopted version control system must be Git. A strict monitoring of the work is required.

3.3. Future Development

Another scope in which the software will be used is the robot navigation. The shapes identified by the software will be used like a starting point for an object recognition process. The information about these objects, then, will be used by ACT-R in order to take intelligent decisions during the robot navigation inside a building, without any other knowledge of the environment. The object recognition module has not been developed yet nor the requirements for achieving this second goal are defined.

Moreover, in order to scale with other kinds of experiments an *optical character recognition* tool will be added and other simple shapes will be recognized.

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 [Botb].

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.

All the information about Scrum are taken from [Botb]. See it for a more detailed description.

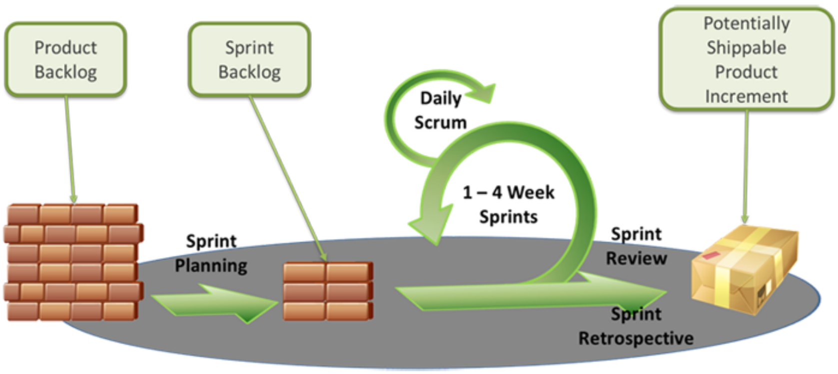

Figure 4.1.: Overview of the Scrum Process

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, add to the product a finite and usable set of new features at constant time intervals.

Scrum Teams are self-organized and cross-functional. Self-organization gives the members the possibility to choose the best way to accomplish their work, without being directed by others outside the team. This fact not only avoids the team to be directed by someone who has not the technical knowledge to solve specific problems and, consequently, can not find the best ways to achieve the goals, but also allows the Team to optimize its own development process. Moreover, the consequent improvement of the sense of ownership that the Team feels on the product increases the motivation of the members. Cross-functionality gives the team all the competencies needed to accomplish the work without depending on others not part of the team. In this way, all the tasks can be accomplished directly by the team, thus the team is never blocked waiting for the work of someone outside it.

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 deliverable version of the product is always available.

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 and is responsible for the performance of the team. His role is to define the requirements for each new version of the product, assign them the priorities and explain them to the team in detail. 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.

Scrum rules require a person to assume the role of Product Owner to represent the will of a committee. In this way, he assumes an intermediation role for all the communications between the developers and the stakeholders and becomes a reference point for both these categories. On one side, he has to give the stakeholders all the information about the progress of the development and the performances of the team. On the other hand, he has to decide the priorities of the tasks and deliver the team the messages of the stakeholders.

Development Team

The *Development Team* consists of professionals programmers, whose role is to implement the new functionalities to the product. The team is usually composed of a number of members which varies from three to nine. One team must be self-organizing, this means that only the members can decide step by step the tasks in which the items are split and how to accomplish each of them 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. The more synergy the Development Team has, the higher its overall efficiency and effectiveness are.

Scrum Master

The *Scrum Master* has the role to verify that the Scrum process is understood and put in place. He does this by 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 avoid useless interactions. 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 understanding and practicing the agile development. On the other hand, 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, p.6].

4.1.2. Events

Prefixed meetings in Scrum have the purpose of marking time and hence minimizing 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 wasting time.

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 prefixed duration and can last from one week to one month. During this period of time the team creates a self-contained portion of product.

Each Sprint starts with the *Sprint Planning Meeting* and ends with the *Sprint Review Meeting* and the *Sprint Retrospective Meeting*. All these events will be described with more details in the next sections.

Sprint Planning Meeting

The *Sprint Planning Meeting* is a meeting at which all the member of the Scrum Team participate. It is the beginning of every Sprint. In this meeting the team decides the new features the product will include by the end of the Sprint.

At first the Product Owner informs the team about the features that he wants to be added to the product, each one with its priority. Each one is an item in the *Product Backlog*, which is described in the section 4.1.3. Then, the Development Team estimates how long it would take to add every new feature to the product and, consequently, how many of them will be added by the end of the Sprint. Often in this part of the meeting the Product Owner and the Development Team have a talk in order to define better the requirements. This fact guarantees that the developed features adhere strictly to the requirements and leads to a new time estimation for most of the goals.

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 delivery plan are collected in the *Sprint Backlog*, described in the section 4.1.3.

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 reason the Sprint Goal should not be changed during the Sprint.

Daily Scrum

The *Daily Scrum* is a fifteen minutes event for the Development Team and the Scrum Master, 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 had to front. The meeting has the purpose of evaluating the progress towards the Sprint Goal and analyzing the trend of the progress in comparison with the Sprint Backlog. This allows the Scrum Team to measure its speed and, consequently, to make better time estimations in the next Sprints.

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. The Development Team describes the problems it encountered, how it solved them and shows the new functionalities added to the product. Then the whole team discusses of new features to be added, old ones to be deleted or how to improve other ones and, more generally, collaborates on what to do next, updating consequently the Product Backlog.

Sprint Retrospective

The *Sprint Retrospective* is a time-boxed meeting that takes place after ever Sprint Review. It is an opportunity for the Scrum Team to analyze its Scrum implementation and create a plan to improve the next Sprint. During the meeting the focus is set on people, relationships, processes 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 Scrum method more effective by optimizing the development process and introducing techniques that make the method more productive and enjoyable.

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, sets the priority and the Development Team defines the estimate.

The Product Backlog is dynamic. Its earliest versions 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 to the product.

As the higher ordered Items are more important than lower ordered ones, they are clearer and more detailed. The team can make more precise estimates thanks to the greater clarity and increased detail.

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 to do 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 modifications to the product and realizing the Sprint Goal. The Sprint Backlog contains all the functionalities that should be added to the product by the end of the Sprint, each of which with its estimates and priorities, and, at the same time, monitors the state of the work during every day of the Sprint.

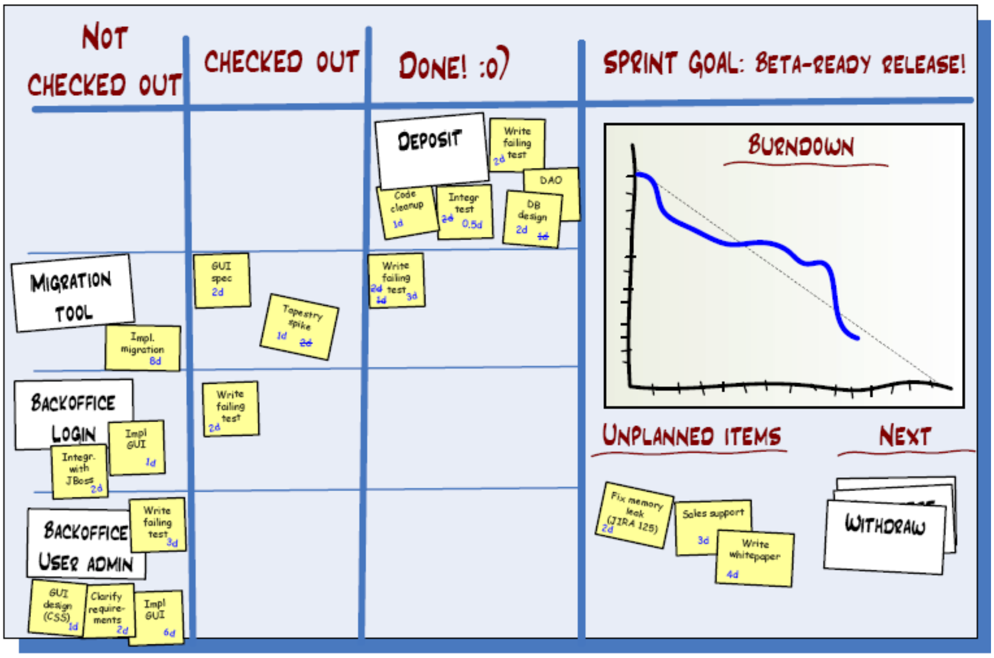

Figure 4.4.: Sprint Backlog and Burn Down Chart

The level of detail of the work is such that it can be measured during the Daily Scrum. In this meeting, in fact, the amount of work done in the day is calculated and the activities for the next day are planned. Usually every Product Backlog Item is split into tasks, each of which must be accomplished in not more than sixteen hours of work. The tasks are never assigned; rather, each member of the team takes charge of some of them 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. The knowledge of the speed of the team allows the team to make better estimations in the future Sprints.

4.2. The Adopted Development Process

The development team decided to adopt an incremental and iterative development process, implementing the *Scrum* framework.

The main motivation of this choice is the capacity of this method to front very well the modifications of the requirements. In most of cases, in fact, even if the requirements at the beginning seem to be well defined and clear, they change during the development. For example, it could happen that new requirements reveal to be more important than other ones or that after some clarifications between the client and the developers some features are redefined, some are deleted and some others are added. The possibility of an incremental and iterative development process to make continuous reviews of the product faces better such changes in the requirements.

Moreover, the introduction of meetings in which is possible to discuss the state of the work and the working method improves considerably the performances of the team itself. In this way, in fact, the supervisors of the development team can postpone all the not essential communications until the next meetings, without the need to interrupt his work nor the developers'. This leads to a more focused and less stressed work both for the developers and the supervisors.

The possibility to check the work at every Sprint increases the compliance between the system and the requirements. During the Sprint Reviews, the new increment of the software is shown. In this meetings, the clients give the Scrum Team a feedback about what they want and if the feature developed satisfy their need. So if there have been some misunderstandings in the comprehension of the requirements, they can be fixed in the next Sprints. This continuous product revision leads to a strict adherence between the system and the requirements.

The adopted development process allows the Product Owner and the Scrum Master to evaluate the performance of the team. As the features to be added and the estimates are decided during the Spring Planning Meeting, the Product Owner and all the members of the team have the plan of the whole Sprint. During the Sprint Review, the members can monitor how the actual work has been done in comparison with the plan. On a shorter period, the Daily Scrum is used to monitor the activity of the days, while the Product Backlog contains the history of all the modifications done on the software since it was born. For all there reasons, with Scrum, the performances of the team are constantly measured.

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

7.1. Sviluppi futuri

aggiungere: riconoscimento ellissi riconoscimento testo

introdurre il flusso video –¿ predisposto

migliorare le performance dell'algoritmo in modo tale che la shape detection sia utilizzata in tempo reale pnell'ambito della navigazione con robot.

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.