Evaluation of CMPack'03*

Patrick de Oude
poude@science.uva.nl

Tim van Erven
talerven@science.uva.nl

Jochem Liem
jliem@science.uva.nl

Tim van Kasteren
tlmkaste@science.uva.nl

January 29, 2004

Abstract

In this paper the CMU CMPack'03 architecture for playing in the Sony four legged Robocup league is evaluated. Its advantages and disadvantages are discussed and improvements of the software are suggested. Finally a comparison is made between the CMPack'03 architecture and the architecture of the software of the German Team. This report should give the reader a better understanding of the CMPack'03 architecture. For technical details about the CMPack'03 architecture refer to [1].

Contents

*We would like to thank our project supervisor Peter van Lith and teacher Arnoud Visser for their support. We also want to thank the team evaluating GT2003 for their information about the German architecture (see [5]).

1 Introduction

The Robot World Cup Soccer and Conferences (RoboCup) is an international research and education initiative to do research on a wide variety of methods in artificial intelligence. The task of a team that participates in RoboCup is to let fast-moving robots play a game of soccer in a dynamic environment. This is a good way to explore technologies that are needed to create autonomous robots.

RoboCup is divided into three different domains: RoboCupSoccer, RoboCupRescue and RoboCupJunior. Every domain is subdivided into a number of leagues. The RoboCupSoccer domain contains the Sony four legged robot league where Sony Aibo robots compete against each other. Many teams already participate in the RoboCupSoccer four legged league, but there is no Dutch team competing. In the following RocoCupSoccer event the Dutch team, a collaboration of five institutes in the Netherlands, will participate with the Aibo ERS-7 robot. Currently the Dutch team doesn't have its own architecture. As developing a new architecture from scratch is very time-consuming, they want to start from an existing code base. The Dutch team is currently considering both the German Team and the CMU CMPack'03 code bases. The available architectures are written for the older ERS-210 Aibo robot and must be ported to the Aibo ERS-7 robot. This document should give the Dutch team and other interested an overview of the American CMPack architecture and explore the possibility to port the CMPack software to the Aibo model ERS-7.

The Carnegie Mellon architecture was first developed in 1998 under the name CMTrio and renamed to CMPack in 2000. The Carnegie Mellon Aibo team won the RoboCupSoccer in 1998 with CMTrio and in 2002 with CMPack software. In 2003 they ended up fourth with CMPack.

The main modules of the CMPack'03 architecture are motion, vision, behavior, localization and the world model. This paper should provide the reader with an overview of these modules and how they interact with each other. After a description is given the modules of the CMPack'03 architecture will be compared to German Team architecture (GT2003). Another group of students is researching the architecture of GT2003 [5].

This document should help the Dutch team to make a final decision about the architecture to use in the upcoming RoboCupSoccer event.

2 Architecture

The software architecture of CMPack'03 consists of a number of objects. The two most important objects are *motion*, in which the robot's movement is controlled, and *main*, which bundles a couple of modules that determine the robot's behavior. The communication between these objects and modules is depicted in Figure 1. The other objects in the program are responsible for outputting debug information and managing shared memory.

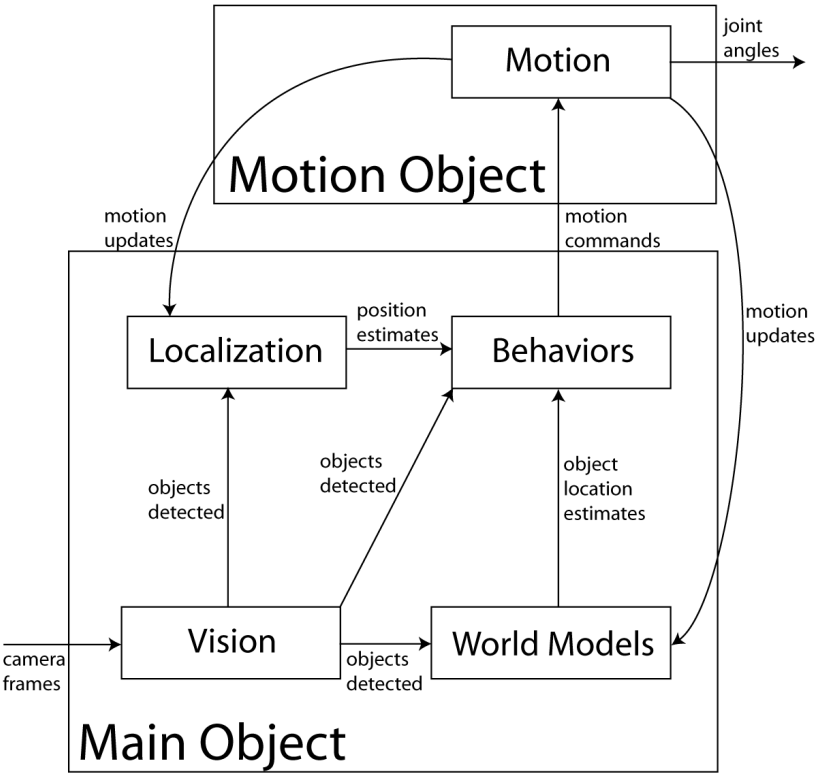

Figure 1: The main and motion object of the CMPack'03 software

The iterative process shown in the figure, is initiated by an incoming camera frame. After this frame is processed and distributed to the various modules, the next appropriate movement is determined in the behavior module. After this movement has taken place, the main object is updated with the motion results. The cycle continues once the next camera frame is received.

3 Modules

This section describes the various modules contained in the CMPack'03 architecture:

3.1 Vision

The vision module consists of three major components. The component for high-level vision searches for objects of interest in an image. The visual sonar component identifies regions of freespace and obstacles surrounding the robot. Both depend on preprocessing of the image by the low-level vision component, but not on each other.

3.1.1 Low-Level

The low-level vision component is responsible for summarizing the important features of the image. It consists of three main stages. First it segments the image into symbolic colors using a lookuptable that is calibrated off-line. Then it Run Length Encodes the image.[1] And finally nearby runs are joined into regions.

3.1.2 High-Level

The high-level component searches for the ball, markers, goals and other robots in the image by comparing candidate regions to a set of models that predict features the object should have when seen through the camera. The best match for each object is reported to the behaviors module together with a confidence measure and location relative to the robot for the object.

3.1.3 Visual Sonar

The visual sonar vision component vertically scans the image at fixed horizontal increments. It identifies regions of freespace and objects in each scanline. Under the assumtion those objects are on a common ground plane a distance map of the surroundings of the robot is created. For each point a certainty measure is stored. This certainty decreases as time progresses. If the robot moves, the distance map is updated to reflect the new positions of the objects relative to the robot. Unseen objects are *forgotten* after four seconds.

[1] In Run Length Encoding horizontal repetitions of the same color (runs) are stored as pairs of starting locations and number of repetitions.

Based on the information from the visual sonar an *occupancy grid* is constructed. The occupancy grid consists of a grid overlayed on the distance map with each grid cell maintaining a probability of occupancy based on the number of obstacles relative to the amount of freespace it contains.

3.1.4 Remarks

The vision code in CMPack'03 contains much practical knowledge gained from experience. This includes work-arounds for weaknesses in the vision algorithms and details of the implementation that are too small to mention in the theoretical descriptions of the vision algorithms. We believe these relatively minor tweaks have a significant impact on the performance of the CMPack'03 software.

3.2 Localization

Localization is the procedure of determining the location of the robot in the environment. The CMPack architecture uses a probabilistic approach to localization of the AIBO. Probabilistic localization represents the location of the robot as a probability density over possible robot locations or poses (position and orientation of the robot). This probability density function is known as the *belief state* of the robot and is updated for movements of the robot and information from the sensors. The belief state is only updated using the previous robot location (*Markov assumption*). The update equation for movements:

$$B^-(L^k) = \int_{l^{k-1}} P(L^k|u^{k-1}, L^{k-1})B(L^{k-1}), \tag{1}$$

where $B^-(L^k)$ is the belief state without incorporating the sensor information at location L on time k, $P(L^k|u^{k-1}, L^{k-1})$ is the likelihood of location L at time k given the previous motion command u and the previous location and $B(L^{k-1})$ is the previous belief state. The update equation for sensors:

$$B(L^k) \propto P(o^k|L^k)B^-(L^k), \tag{2}$$

where $P(o^k|L^k)$ is the likelihood of sensor reading o at time k given the location at time k.

The belief state is updated using probability density functions $P(o^k|L^k)$ and $P(L^k|u^{k-1}, L^{k-1})$. These probability densities can not be expressed in closed form, because of the high complexity of the these functions. To solve this problem the pose probability density function (*belief state*) is approximated. This approximation is done by using pose samples of the robot. This procedure is called a *particle filter* (also known as *Monte Carlo Localization*). There are different types of particle filter based localization methods. The CMPack'03 uses *Sensor Resetting Localization (SRL)* which is an extention of the Monte Carlo Localization.

A drawback of using this procedure for localization is the difficulty involved in choosing the right number of samples to use. The number of samples that are used to approximate the pose probability density is proportional to the computation time required. Too many samples used for approximation could result in

excessive CPU usage and too few samples could result in bad approximation of the pose probability density. Other drawbacks for the motion model are biased movement estimates, collisions, movement of the robot by humans, slippage of the robot on the floor and effects of partially drained batteries. For the sensor model biased sensor estimates, effects of changes in the environment (like lighting conditions etc), failure or degradation of the sensors or interference by humans could result in bad estimates of the location of the robot.

Because localization is not always successfull a failure recovery procedure is needed. Detecting a failure can easily be done by using the pose probability density to estimate the sensor information. Comparing the estimated sensor information with the measured sensor information results in a good indicator if localization is failing. Resetting localization by telling the robot it is lost is a good way to recover from a failure.

3.2.1 Remarks

The *SRL* method for localization is very suitable for handling relatively few samples and large errors. However the procedure needs a threshold value to determine whether to keep samples. This threshold value must be empirically determined. A better choice would be the emphadaptive Monte Carlo Localization localization method. This method outperforms *SRL* with respect to robustness and failure recovery. In addition there is no need to set a threshold value. For more details about these localization methods refer to [2], [3] and [4].

3.3 Motion

CMPack uses linear interpolation between keyframes to produce motions. Each keyframe specifies either positions or angles for the legs, head and mouth and a position for the body. These keyframes are stored in .mot files, which can be generated in various ways.

This separation between motion descriptions and the rest of the code allows the team to easily switch to a different set of motions or to try out motions from other teams. CMPack includes its own utilities to generate motions, but also includes code to import movements from other teams like e.g. the German and UNSW team. Another benefit is that the code to interpolate between keyframes can easily be changed. Code to use spline interpolation instead of linear interpolation is available in the codebase.

3.3.1 Remarks

To port the motion module from the ERS-210 to the ERS-7, both the code and the .mot files would need to be adapted to the changes in the degrees of freedom between the two models. However, we anticipitate this would pose no major difficulties.

3.4 World Model

The world model is used to track objects on the field, even when they are out of the robot's view. The position of objects and their uncertainties are represented with two-dimensional Gaussians. Sensor and motion information is used to update the means and standard deviations of these Gaussians. In addition to these position estimates, the world model also provides the robot with the ball's velocity and how close the ball's path passes the robot. This information is calculated by applying linear regression on the history of the ball position.

Besides getting information for the world model from its sensors the robot can also obtain information from the world model it shares with its teammates. In fact, the robot always uses the information from the shared world model to obtain the location of its teammates. This is because the people from CMU trust their localization function better than their vision function. Another example in which the shared world model is used is ball tracking. If the robot has lost sight of the ball for a period longer than three seconds it consults the shared world model to find out whether any of the other robots see the ball.

3.5 Behavior

The behavior module in CMPack combines multiple low-level reactive behaviors using sequencing. The main idea of sequencing is to run only one reactive behavior at a time and switch the active behavior from time to time. Sequencing is implemented using a finite state machine (FSM) with each state of the FSM corresponding to a reactive behavior. The low-level reactive behaviors in turn may be implemented using a FSM as well. Transitions between states of the finite state machines are based on transition rules. In addition *hysteresis* is used to prevent oscillation between states. Hysteresis involves using a buffer zone between two states. In this buffer zone the robot uses the state it was using when it entered the buffer zone

The behavior module acquires the information it needs about the world from the other modules. There is one large object in which all the information is collected, which is updated after each camera frame.

The CMPack software uses four roles. The goalie role is assigned in advance and doesn't change during the game. The other three roles are those of primary attacker, supportive attacker and supportive defender. These last three are assigned dynamically at run-time. Different actions are avaible to the robots depending on their assigned role.

Token passing is used to ensure mutual exclusion when assigning the role of primary attacker. The primary attacker becomes the team leader and assigns the supportive roles to the two remaining robots.

3.5.1 Remarks

Using sequencing has many advantages. The behavior is very responsive to the environment, different actions are possible from the same perceptual state, and it is quite easy to chain together actions into larger behaviors. The latter enables easy implementation of dynamic role assignment.

4 Comparison of CMPack'03 to GT2003

In this section the codebase of the Carnegie Mellon University (CMPack'03) is compared to the codebase of the German Team (GT2003). We will specifically look at the criteria modularity, functionality, documentation and quality of utilities.

4.1 Modularity

Both the CMPack'03 and the GT2003 code base are quite modular. The the German software, however, has two things going for it. Firstly, modules can be enabled, disabled or switched at run-time in the GT2003 software. Secondly, GT2003 has been composed by selecting the best modules from multiple competing implementations developed by various German institutes. This proves the German architecture is very modular and the interfaces are well defined.

In contrast, the modularity of the CMU software has not been subjected to any such challenges. While CMPack'03 contains multiple implementations of some modules, these have been developed by the same people, so they provide no proof for the functional separation of the implementation. In our experience the module interfaces of CMPack'03 are not immediately clear at first sight, but can be figured out with some work.

In both codebases modules appear to be self-contained. We have found no indication of functions being distributed over multiple modules. Because of the modular approach both teams take, we estimate it would take an equal amount of work to replace or add functionality to modules.

4.2 Functionality

4.2.1 Vision

CMU and the German team have taken very different approaches to vision. CMPack'03 combines the object localization provided by the high-level part of their vision module with the occupancy grid extracted by their visual sonar. In contrast the German team uses an approach based on scanlines parallel and vertical to the horizon [7]. An assessment of the advantages of both approaches would require a separate research project measuring their performance under game-like conditions.

4.2.2 Calibration

The documentation provided for the German software [6] claims that they do not need any calibration of the camera. However its codebase does include a file in which the colors the agent needs to know are defined. The values of the colors do not need to be changed to incorporate lighting changes, because this is taken care of at run-time. In the CMPack'03 software the camera has to be calibrated before the match begins, by using a special behavior in which pictures are taken of the environment. The resulting pictures must be segmented by hand and the colors put in a color.txt file.

4.2.3 Motion

CMPack'03 and GT2003 [7] both use linear interpolation between keyframes stored in motion files to produce motions. The CMU software has the minor advantage that it includes several scripts to import keyframes from other teams.

4.2.4 World Model

CMPack'03 and GT2003 store more or less the same information on the world. While in the CMU software all this information is combined into a single world model, in GT2003 it is distributed over multiple separate data structures. It is not immediately clear, however, if either approach provides any real advantages over the other.

This approach is also used when storing information received from other robots. Each robot running the CMU software stores the information it has received from its teammates in a single shared model of the world. Robots running the GT2003 software store this information in multiple separate data structures.

4.2.5 Behavior

Both teams model different behaviors as finite state machines (FSMs). The American team implements behaviors in a C++ file, while the German team describes behaviors in a declarative manner in XML-files. The German behaviors can be parsed by a script which outputs the FSM as a .png-file. Both CMPack'03 and GT2003 use dynamic role assignment.

To participate in Robocup challenges both teams have created specific behaviors. Low-level changes which might be needed for the challenges are equally difficult to incorporate in the software. It is impossible for us to evaluate the performance of the software on different challenges, because we lack the equipment (field, markers, ball, Aibo's) for evaluation and neither porting team has finished a complete port of the software for the ERS-7.

4.3 Utilities

The German team has a very large advantage when porting or adapting their software. They have a utility which lets them switch modules and behaviors at run-time, show debug information, move the robot and visualize the playing

field with all robots. The program can even show the results of the work the modules do, so they can easily be evaluated.

A simulator is also present in the German software, which they use to test their code. This is particularly useful to test new or adapted moves. It can be risky to try new moves for the CMPack'03 software, as they cannot be tested in advance and might damage the robot.

To get the CMPack'03 software running on the Aibo files have to be copied by hand to the memory stick. It is easy to make a mistake when doing this, which causes the robot to fail. The German team provides a small script which copies the necessary files to the memory stick.

Another utility the German team has developed can be used to record robot movements by moving the joints by hand. These movements can later be optimized, either by hand or using a learning algorithm.

The CMPack'03 codebase includes a lot of small scripts to do calibration and convert movements made by other teams. These are by far not as advanced as the German utilities.

4.4 Documentation

The documentation about the CMPack'03 software consists of a few papers (see references) and several series of slides which can be found online [8]. The slides are sufficient to get a general insight into the software and the papers discuss various modules more in depth. The German team has made significantly more documentation available, which is also more neatly bundled. Only a few of the utilities are not documented.

If we look at the codebases of both teams we see large differences. The CMPack'03 software has very few comments, but still is quite clear. Another thing we noticed when reading the CMPack'03 software is the amount of tweaks which are in the code. The CMU team obviously has a lot of experience and this can very clearly be seen when looking at the code. The German software is very well documented and there even is a utility to export the comments to html-files. Both code bases are quite easy to get into if you have read the general overview of the architecture.

5 Porting results

We were able to get the software to run on the ERS-210 and made a partial port to the ERS-7. For details of this port, please see [1].

6 Conclusion

Most important in comparing the CMPack'03 architecture with the GT2003 architecture are differences in modularity, functionality, utilities and documentation.

6.1 Advantages of the German Team GT2003 software

The German team has a proven modular approach and their extensive and well-structured documentation is to be preferred over the limited and distributed documentation of the Americans. The development of behaviors in XML is much easier and foolproof than in pure C++. This is another advantage of the German software. In addition the provided utilities of the German team are far more advanced and useful compared to the American utilities. If the calibration is as straightforward as the German documentation claims, obviously the German method is to be preferred.

6.2 Advantages of the CMU CMPack'03 software

The CMU software has one clear advantage over the German software: it incorporates the practical experience from years of participating in the four-legged Robocup league. This is most apparent in the vision module.

All in all we feel the advantages of the German software outlined in the comparison of the two software architectures outweigh the benefits of the CMU software. In our opinion the Dutch Robocup team should choose the German architecture to base their Robocup agents on.

Unfortunately there isn't a complete port of the CMPack'03 architecture to the ERS-7 available to evaluate the performance of the various modules present in the CMPack'03 architecture. Interesting future work is to evaluate these modules and their algorithms in comparison with the modules used in the GT2003 architecture. Especially the vision module of the CMPack'03 is interesting to evaluate, because this module is far more advanced than the GT2003 vision module.

References

[1] Patrick de Oude, Tim van Erven, Tim van Kasteren, Jochem Liem: Technical Report CMPack'03, 2004

[2] Jens-Steffen Gutmann e.a.: An Experimental Comparison of Localization Methods, 1998

[3] Jens-Steffen Gutmann e.a.: An Experimental Comparison of Localization Methods Continued, 1998

[4] Scott Lenser, Manuela Veloso: Sensor Resetting Localization for Poorly Modelled Mobile Robots, 2000

[5] B. Ottens, A. Abbo, P.J. van der Meer, M. Stienstra: Aibo Project 2004 - German Team Report, 2004

[6] M. Jüngel, J. Hoffmann, M. Lötzsch: A real-time auto-adjusting vision system for robotic soccer, 2004
In: 7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences), Lecture Notes in Artificial Intelligence, Padova, Italy, 2004. Springer (to appear).

[7] Thomas Röfer, Hans-Dieter Burkhard, Oskar von Stryk, Ingo Dahm et al.: GermanTeam RoboCup 2003
Available from: http://www.robocup.de/germanteam/

[8] Scott Lenser, Manuela Veloso: CMRoboBits http://www-2.cs.cmu.edu/ robosoccer/cmrobobits/html/schedule.html