A Multi-Robot Architecture for Autonomous Cooperative Behaviours

by

Nils Ole Tippenhauer

Research project - Final report
presented to the University of Waterloo

Date of issue: May 10th 2005
Date of delivery: August 10th 2005

Supervisor:
Professor Mohamed Kamel
Canada Research Chair
PAMI Research Group Director
Electrical and Computer Engineering

Home supervisor:
Professor Mayer-Lindenberg
Distributed Digital Systems Group Director
Computer Science

Waterloo, Ontario, Canada, August 2005

I hereby declare that I am the sole author of this thesis.
I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Nils Ole Tippenhauer

I further authorize the University of Waterloo to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Nils Ole Tippenhauer

Abstract

This paper focuses on the design and implementation of an architecture for a multi-robot system with autonomous cooperative behaviour between the individual robots. Different software used to solve this problem is compared and two implementation efforts are made resulting in a working architecture based on the two programs PLAYER and JADE which is then used to realize a cooperative group behaviour example.

Contents

List of Figures

List of Tables

1 Introduction

1.1 About this project

This project is a research project to partially fulfill the requirements to obtain the diploma in my German study program, becoming a "Diplom Informatik-Ingenieur". The requirements for this projects are the following: a total work duration of about 400 hours, done in 3 months.

1.2 The lab

The PAMI research group was founded in 1980 and is a part of the Department of Electrical and Computer Engineering. The group works on a wide diversity of pertinent research areas. It also consists of several professors, researchers and graduate students, as well as adjunct members from other departments and universities. The lab has a range of different computers, servers and robots[14].

1.2.1 The robots

The PAMI lab has different robots, the ones used mostly are the three MagellanPro robots1. They were produced by Real World Interfaces,Inc.[1]. These cylindrical robots are about 50cm high and about 40cm in diameter and have a full set of 16 sonar, 16 infra-red and 16 bumper sensors arranged around the body. Those sensors are operated by two dedicated embedded controllers which are connected to a standard PC inside the robot. This PC is running the operating system RedHat Linux 6.2. It is also equipped with wireless LAN, an odometer and a video camera on a pan-tilt unit. No other harddrive than the harddisk is available. Direct connections to the robot can be made through a serial connection and a terminal emulator or by connecting a standard PC screen and keyboard.
Due to the linux installed the robot is also accessible via ssh or telnet. The robots names are mag1, mag2 and mag3, they have a DNS entry for mag1.uwaterloo.ca and mag2. and mag3. respectively.

The other kind of robot I used in my project was an ATRV mini by the same company, iRobot. The ATRVs in the lab are not as well equipped with sensors as the smaller Magellan Pros and are by design more outdoor robots with big wheels which should allow them to manoeuvre outside in rougher terrain. The default sensors of the ATRVs are 16 sonar sensors spread around the

[1]Real World Interfaces is now called iRobot and discontinued their series of research robots

Figure 1: iRobot Magellan Pro robot

Figure 2: iRobot ATRV mini robot

robot with a strong emphasis on the front section. The unit is also heavier than the Magellan Pro which weights about 20 Kg, it weights around 40 Kg. I used those robot to experiment with the Linux installation process and the PLAYER configuration. To test the PTZ and camera interface of PLAYER I temporarily installed the Sony camera of mag3 on the test ATRV, min1.
Their DNS names are min1, min2 and min3.

1.2.2 The mobility software

The original software by the producer of the robots is called mobility. It allows access to the last measurements taken by the sensors and provides an easy API for accessing informations like the current position of the robots or sensor values. It also provides a central naming service to

communicate with the robots. As it is a binary only software ("black box") its use to researchers is limited. To establish accurate sonar perception and calibrated infrared perception for example, considerable time and effort is required because information about the time at which each individual sonar transducer is activated is inaccessible through the binary-only software layer [13].

1.3 Overview over work done so far in the PAMI lab

This section will give a short overview of the preview work done with the Magellan Pro robots. It will concentrate on the projects which immediately preceded my work in the last year.
Every project has in common that the software was developed from scratch and specifically for the project's goals with a very limited use for the following projects. After the switch from the mobility software to C++ programming a big part of the programming work was spent for the message exchange and inter-robot communication in the projects with group behaviour. The original goal of my work, the combination of different already programmed behaviours would require a major extension of existing communication code and would allow later groups to concentrate more on higher level group behaviour work rather then low level networking.
The goals and design of my project are described in the corresponding sections 3 and 4

1.3.1 early 2004: 4th year project on Co-operative Robot Behaviour

Four students (Christpher Book, Jean-Paul Haddad, David Henderson and Andrew Sheppard) started with programming cooperative group commands like a triangle formation using the mobility software. They implemented algorithms to make cooperative decisions and to communicate between the robots. At the end of the project the robots were able to form a straight line with a given length and a triangle with a given side length.

1.3.2 RobotManager

Ben Miners wrote a replacement for the closed source mobility manager in C and C++ to have a more flexible and better maintained system. It enables direct access to the sensor data and other features. Communication through Corba is replaced by sending direct Messages and commands per telnet. This new softare layer allowed precise calibration of the robots sensors. Documentation is available in the form of a descriptive PDF file and automatic documentation generation from the source files by DOxygen. Behaviour of robots can be added by simple C++ programs which use

the robotmgr framework and are run directly on the robot. No naming service like in mobility is provided, the inter-robot communication is planned but not fully implemented. My initial work was done using RobotMgr, I will explain more about RobotMgr later when I introduce the software used in section 2.5.

1.3.3 late 2004: Work done by Slim and Raydan

Slim and Raydan ported the code from the first 4th year project to the new robotmgr and extended it to a greater variety of commands such as the implementation of an algorithm to shoot an orange football (basketball). The second program they tracks a moving basketball with the cameras of both robots. The last program located the previously unknown position of the robot by recognising landmarks at known positions and deducting the own position from those informations.

1.3.4 Geometric landmark discovery

Ben Miners and other wrote code to detect a special artificial landmark which is recognisable regardless of the angle of the viewer, in realtime. This is based on a paper from Briggs, Scharstein and Abbott [4]. The landmark consists of an p-similar pattern which can be detect reliable and fast from widely differing angles and distances. The landmark can have another barcode included which enables to identify this special landmark and give more data about it. After the p-similar pattern is recognised the additional barcode is located and analysed. This gives you the opportunity to use more than one landmark, for example to help the robot to navigate in a room.

1.3.5 Multiple agent architecture for a multi robot system

Bram Gruneir worked with the Magellan pros as part of his master's thesis about a multiple agent architecture for a multi robot system. He used JADE and Robotmgr to control and coordinate the robots in a group with a dynamical number of 1-3 robots. The report about his work is unpublished so far and I haven't got a copy of it, so all I could use of his work was his source code and what I got from a presentation from him about his work. There is a video of his robots building a formation around a special coloured can on the PAMI website[14]. JADE is introduced in section 2.5.

2 Theoretical background

This chapter will introduce basics about robotic software architectures, cooperation between the robots and possible problems in that area. To conclude a selection of software I found in literature is compared, among them the later used PLAYER and JADE.

2.1 Robotic software architectures

A software architecture in robotics defines the key components the software running to control the robot and the way they interact. It also suggests ways how to use this combination in its intended way for fulfilling tasks. It is common to differentiate distinct levels in the architecture. As stated in [16, p.932]

> Modern-day software architectures for robotics must decide how to combine reactive control and model-based deliberative control. In many ways, reactive and deliberate control have orthogonal strengths and weaknesses. Reactive control is sensor-driven and appropriate for making low-level decisions in real time. However, reactive control rarely yields a plausible solution at the global level, because global control decisions depend on information that cannot be sensed at the time of decision making. For such problems, deliberate control is more appropriate.
> Consequently, most robot architectures use reactive techniques at the lower levels of control with deliberate techniques at higher levels. [...] Architectures that combine reactive and deliberate techniques are ususally called hybrid architectures.

2.1.1 Control localization in behaviours

A third layer which is often added between the two mentioned layers is the executive layer which is the translator between the reactive and deliberate layer. As an example the executive layer can receive waypoints which are generated by a long term path planner in the deliberate layer and choose an appropriate way to reach this waypoint, then giving the associated commands to the executive layer.
Following [19, p. 291] wanted features of a well-designed architecture (in this case a navigation architecture) are:

1. Modularity for code reuse and sharing

> Basic software engineering principles embrace software modularity, and the same general motivations apply equally to mobile robot applications. But modularity is of even greater importance in mobile robotics because in the course of a single project the mobile robot hardware or its physical environmental characteristics can change dramatically , a challenge most traditional computers do not face. For example, one may introduce a Sick laser rangefinder to a robot that previously used only ultrasonic rangefinders.

2. Control localization

> Localization of robot control is an even more critical issue in mobile robot navigation. The basic reason is that a robot architecture includes multiple types of control functionality (e.g.,obstacle avoidance, path planning, path execution, etc.). By localizing each functionality to a specific unit in the architecture, we enable individual testinf as well as a principled strategy for control composition. [...] It is also valuable to localize such high-level decision-making software, enabling it to be tested exhaustively in simulation and thus verify even without a direct connection to the physical robot.

A general tool to implement control decomposition are behaviours, which incorporate a specific component of the architecture.

2.1.2 Temporal control decomposition

A possible axis along which we can discriminate the control architecture into distinct behaviours is the time constraints on the components. High level path planning components are only going to be called every minute or 10 minutes, whereas low level sensor handling and motor control will be updated in frequencies of 10 Hz and faster as depicted in figure 3[2]. Those low level process also have real-time demands to the scheduling while processing them is usually quite fast. for the path planning no real-time requirements are made, but the computation might take a while due to its complex nature depending on the algorithm used. An example for a more complex architecture to combine different behaviours and motivations is the ALLIANCE[2] architecture

[2]Based on [19, p.294], numbers changed slightly

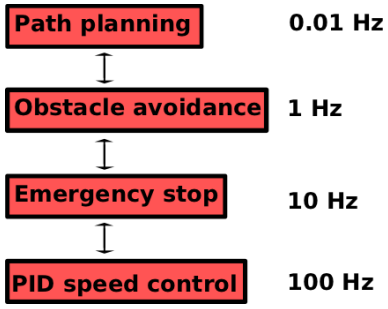

Figure 3: Schematic of four level temporal decomposition

2.2 Cooperation among a group of robots

Without going too much into notions of multibody agent planning some basics will be explained here. As stated in [16, p.450]

> Plans can be constructed that specify actions for both players on the team [in that example]; we will describe techniques for constructing such plans efficiently. Efficient plan construction is useful, but does not guarantee success; the agents have to agree to use the same plan! This requires some form of **coordination**, possibly achieved by **communication**.

The mechanisms to achieve coordination between the agents can vary. In the simplest case there are domain-specific conventions which force the agents to make the right decisions, also known as social laws if they are widely adopted by everyone.
More abstract mechanisms are domain-independent conventions such executing the same algorithm with same inputs on every robot as described in [16, p.452]

> For example, if each agent runs the same multibody planning algorithm with the same inputs, it can follow the convention of executing th efirst feasible joint plan found, confident that the other agents will come to the same choice A more robust but more expensive strategy would be to generate all joint plans and then pick the one, say, whose printed representation is alphabetically first.

More advanced coordination strategies often emerge through evolutionary processes which result in the flocking behaviour of birds, for example. Because every agent is following a simple set of rules an emergent behaviour can be recognised in the group.

2.3 Practical problems in group communications

A problem in the coordination of a group of autonomous robots is the communication between them. If there is a central infrastructure to provide data in a "black board" fashioned way, like the mobility software does, data exchange is relatively easy. If a more robust and decentralised approach is wanted the number of messages which have to be exchanged to get a common data knowledge in the group is increasing exponentially, or a broadcasting solution has to be chosen. The broadcasting solution would send the data to a certain address in the network which causes every robot to receive those messages. This makes sending the message easier but the amount of messages sent from the network switch to the robots remains the same, effectively only halving the amount of messages sent.
To further lower this amount more intelligent strategies have to be used such as sending messages only to certain robots, for example the immediate neighbors. This way the amount of messages sent can grow linearly with the amount of robots in the group.

2.4 Properties of the task environment

The properties of task environments can be categorized to determine the appropriate agent design and the application of techniques for the specific agent. These categories are taken from [16, pp.40] The world in which the mobile Agent, the robots, will act is assumed with the following informal statements:

- It is only partially observable
  Noisy sensors will prevent the robot from having a full exact overview over its surrounding at every moment, the tilt angle of the camera only allows the camera to view a certain range and the robot can't detect anything when its sight is blocked by obstacles such as other robots, for example.

- Its behaviour is deterministic
  We assume no outside Agents or actions change the environment after the robots have been started, the next stage of the environment is always completely determined by the actions

of the robot. An exception to this is the scenario where the robots should track a moving landmark.

- The task environment is episodic.
  The different tasks will be well separated from each other, so the agent's experience is divided into atomic episodes.

- The task is semidynamic. It is semidynamic in that aspect that the world around the robot will not change while time is passing. The performance score of the robot's behaviour will depend on the time though, so the task is semidynamic. In some applications the world is dynamic, specifically in those cases where the robots are supposed to follow a moving landmark.

- The time during the comletion of the task is continuous
  Both the movement of the robots and other objects are continuous in time, so the state of the environment is changing in a continuous time. The perceptions are continuous in the way they measure angles, get pictures from the PTZ camera and get sonar feedback.

- Finally the setup has multiple, cooperative agents.
  No competing agents try to maximise their performance measure, thus minimising our performance measure, communication is needed to interact with the other agents to reach a common goal, a shared performance measure for example.

2.5 Overview over software used in this area

The advantages of a platform independent multi-robot system which enables flexible reuse and interchangeability of code have been recognised by many groups, thus different software solutions were created. They all have their own distinct features which will be compared now. My goal was to find a software platform allowing the flexible reuse of existing solutions, even if they were originally written for a different platform. Such a software would allow an easy exchange of results between developers of different robotic systems. This software should support the robots in the PAMI lab, of course. Additional comparisons between single and multiple agent software along with a proposed set of comparative dimesions can be found in [1]

2.5.1 ARIA

ARIA[3] is a Object Oriented interface to ActivMedia mobile robots, this way of little direct use for our iRobot robots, but it uses basic ideas which can be considered for a framework for robotmgr as well. ARIA can be used to read sensors and drive the robot with custom actions or to merely send and receive commands. It does threading via its own wrapper around linux pthreads and has reference documentation generated via auto documentation in the code. To determine the action which is to be executed priorities are used. It is licenced under the GNU GPL, allowing us to view and reuse the sourcecode. Other interesting features:

- Built-in socket layer eases inter-program communication via network
- Built-in actions for obstacle avoidance

2.5.2 ALLIANCE

ALLIANCE[2] is a control architecture for fault tolerant, reliable and adaptive cooperation among small to medium sized teams of heterogeneous mobile robots, performing (in dynamic environments) missions composed of independent tasks that can have ordering dependencies. ALLIANCE uses a "intentional" cooperation approach as opposed to a swarm-type approach. This approach was chosen to enable smaller teams of heterogeneous robots to efficiently perform a given task. A key issue here is the allocation of a task to the optimal robot to solve this task. This is solved with a behaviour based approach without a central control. The robots use mathematically modelled motivations such as impatience and acquiescence to achieve adaptive action selection. Lower level behaviours

2.5.3 MARIE

MARIE[12] is a robotic development and integration environment focused on software re-usability and exploitation of already available APIs and middlewares frequently used in robotics. MARIE acts as the centralized control unit using the mediator design pattern for distributed system to coordinate global interactions between different, normally incompatible applications. By using a solid and generic communication framework Marie aims to create a very flexible system that will support a wide variety of applications. To realize these goals each program to be used with MARIE must have a clear method of interactions that MARIE can use for integration. An

implementation integrating different programs such as FlowDesigner/RobotFlow, ARIA, PLAYER/STAGE/GAZEBO, CARMEN and ACE is available for download on the projects website. This implementation is using a Magellan Pro like the ones in the PAMI lab as the hardware basis. The role of the components is:

- ACE is used to create communication framework between MARIE's components
- Navigation and localization are done with CARMEN
- Expressions control, behaviors control (avoid, wander, rest) and joystick control are done with RobotFlow/FlowDesigner
- Player device abstraction allows to use the same system configuration to run in simulation (with Stage,Gazebo) and on a real robot (Magellan Pro)
- Aria device abstraction allows to use the same system configuration to run in simulation (with SRIsim) and on a real robot (Pioneer)
- Application control, joystick control and manual expression control are done with wxWidgets.

2.5.4 OROCOS

OROCOS is an initiative to produce a functional basis for robotic software systems. Based on RTAI (Real-Time Application Interface is a hard real-time extension to the Linux kernel) it is able to offer an open source hard realtime control architecture for several machine types.
This is a very basic software intended for people writing realtime device drivers and using all the offered features such as advanced control methods with asynchronous data access. Cons:
small control problems do not require either real-time, or advanced control methods with asynchronous data access. For these cases, OROCOS can be overkill.

2.5.5 MAGE

MAGE[11] is an open source replacement similar to the RobotMgr program, but with certain limitations in the context of unsolicited/low latency data acquisition, it also lacks full DVI-30 ptz support. Mage is basically a very simple c interface to the rFlex controller on our robots. It was developed during the summer of 2000 for the authors entry into the AAAI mobile robots

competition, later he improved it somewhat for his thesis work.
Because of the limitations Ben Miners decided to write an own implementation, the RobotMgr software. MAGE also seems to be discontinued, as the latest release is from Juli 2004.

2.5.6 RobotMgr

RobotMgr is the replacement Ben Miners wrote for the closed source MOBILITY manager by iRobot. It was written in C and C++ to have a more flexible and better maintained system with full access to the source code. It compiles both on the original RedHat 6.2 of the robots with which they were delivered as also with the Fedora Core 3 of my test robot. Its requirements only include apart from the obvious GCC the GNU Common C++ library. GNU Common C++ is a C++ framework offering portable support for threading, sockets, file access, daemons, persistence, serial I/O, XML parsing, and system services. I compiled robotMgr successfully with the latest version of GNU Common C++, 1.3.1.

Another goal apart from the access to the source code was to enable direct access to the sensor data and other hardware devices of the robot. While sensor readings in MOBILITY are only updated if new values are available and in a more irregular pattern RobotMgr allows direct access to the sensors. This allows precise calibration of the robots sensors, which was done by Ben Miners for one of the Magellan pros.

MOBILITY's communication through Corba is replaced by sending direct messages with TCP or UDP and commands per telnet. The protocol used to send messages can be chosen between UDP or TCP by the code designer to allow both fast or secure message delivery. Messages can be send through the abstract concept of channels to which the robot subscribe to send messages to or listen to new messages. This way applications like a remote monitoring tools which displays all sensor values of the remote robot is possible and also implemented fully working.

Documentation is available in the form of a descriptive PDF file and automatic documentation generation from the source files by DOxygen.

Behaviour of robots can be added by simple C++ programs which use the robotmgr framework and are run directly on the robot. These behaviours are started by either using the telnet interface or calling them directly from a C++ program. They are scheduled as parallel threads

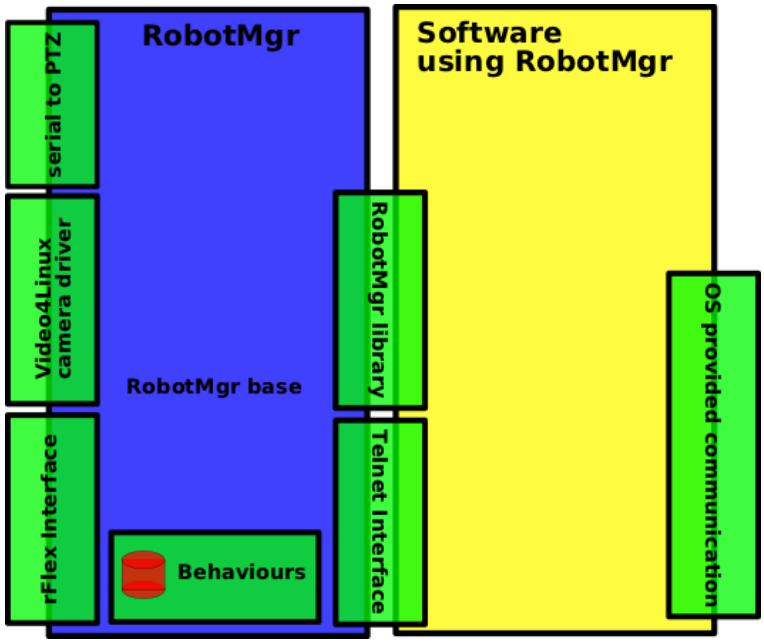

Figure 4: Schematic of the RobotMgr program

automatically as their class is an extension of the pThread class and have a function which is called upon the reception of new data from the rFlex device.

No naming service like in MOBILITY is provided, the older project all used hard coded robot names and addresses. The robots announce the startup of robotMgr on a public channel, but this information has not been used so far.

While testing out RobotMgr's features with small test programs I encountered several problems with the message handling and remote control of other robots. Ben Miners kindly helped me to solve some of those problem, but some remained. Due to these reasons I tried to find other software to control the robots and discovered PLAYER which is described in the following subsection.

2.5.7 PLAYER/STAGE/GAZEBO

These three programs are developed to be used together to write code to program both simulated robots and their real counterparts. PLAYER is the server part which either runs on the robot and gives access to its hardware and communication features or uses simulated sensors when run on a normal PC. To run the simulation either STAGE or GAZEBO is used, where STAGE focuses on simulating the environment just enough to let simulated robot use its sensors, this way enabling the fast simulation of bigger groups of robots in 2D GAZEBO is the high fidelity 3D counterpart, used to simulate only a few robots with much better environmental resolution.
PLAYER provides client libraries for several programming languages such as C, C++, Tcl, LISP, Java, and Python, allowing to combine different pieces of software to a whole project and utilize the relative strengths of the programming languages.

PLAYER also provides driver for a whole range of hardware devices such as the Magellan Pro's rFlex interface and Sony PTZ cam.
PLAYER drivers interesting in the scope of this project are the following:

- camerav4l:
  The basic, uncompressed raw camera images as they are provided by the LINUX v4l2 driver of the capturing card.

- cameracompress

This a driver for compressed camera pictures from a source using the camera interface. This driver can be used to display camera images remotely without stressing the network too much, although the compressing action does stress the robot quite a lot.

- cmvision
  A driver which takes the camera stream from camerav4l for example and uses the CMVision algorithm to detect blobs on the picture.

- rFlex:
  The most important driver enables interfaces to all devices in the robot controlled by the on-board rFlex controllers.

PLAYER runs on Linux and uses the communication features offered by it.
Because of the abstraction level of player and it being open source software programs developed by other groups can be used easily to enhance own projects. Integrated into PLAYER are algorithms such as a goal-seeking obstacle avoidance algorithm (VFH) or an Adaptive Monte Carlo Localization (AMCL).
At any time a player server is running a client called playerv can connect and display the actual robot's view of the world via his array of sensors. It is also possible to give simple move commands with this interface.

2.5.8 JADE

The Java Agent DEvelopment platform[7] provides a FIPA[3] specifications compliant framework for complex agent structures. Messaging is handled using FIPA ACL as the language to represent the messages. Each agent is represented by a globally unique adress called AID which contains the name of the agent and the host and port name on which the agent is listening. Agents can announce services at a global yellow page service running in the root container.
The agents use threaded tasks called behaviours in JADE, which are scheduled by a simple round robin algorithm and are non-preemptive, so one blocking behaviour can stop the whole robot until it is unblocked. There are different kinds of behaviour templates to choose from, the differences are for example the way they terminate (never, after one execution, user-defined).

[3]The FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies.

JADE is usually used with purely software agents as it can be seen in their example files given on the webpage. Those simple examples include the simulation of spreading of information through a group at a gathering or the dynamics of a market with groups of sellers and buyers for goods. In those cases quite a large number of agents do rather simple tasks, unlike the way this project will use JADE.

3 Goals of this research work

During this research and project work a framework for future use with the robots will be written in an iterative process using a rapid prototyping style software engineering approach.
The goal is to introduce a complete set of software to simulate, develop and test cooperative behaviours with reusable components. This set of software should allow easy modification and debugging while providing a high level communication interface to others robots in the group. The software should allow behaviour of the robots with emphasis on cooperation between them and the robots acting autonomously without the aid of a central server. Every robot will be treated equally by design and no predefined leader dictates the group's actions.
This architecture will then be implemented and a complex group behaviour written for it to demonstrate its capabilities.

3.1 Framework for recognition and behaviour

A multi layered framework as described in section 2.1 will be planned and implemented. The architecture consists of two layers of software which are the low level robot control and the higher level execution code.

3.2 Application of cooperative group behaviour

Initially this was one of the first goals of this project because it was assumed that I could use the work of previous groups and combine it to a new complex behaviour rather easily. It was supposed to combine the artificial landmark detection with the group formation command which were both described in section 1.3, but since I switched the used software platforms this application is a completely new design. The new goal is thus:
The group of robots will try to detect the landmark, the robots with the least distance to it is then selected as the group leader. The other robots will build a certain formation around the leader while it is following the landmark as it is moved by a user. Possible extensions are the recognition of a pattern added to the landmark which will then define the kind of formation build by the group or a certain behaviour when the landmark is moved in between the group or very close to another group member. Another extension of this idea is to change the leadership again should the landmark be moved so that its distance to the leader is greater than to another robot, this robot will become the new leader.

3.3 Subgoals

In order to implement the comlex behaviour described above small subsets were implemented and extended step-by-step following a rapid prototyping software engineering approach.
To start those smaller behaviours special messages have to be sent to the running agent, containing a keyword each and variables in some cases which are separated by a colon.
The subgoals are described following:

3.3.1 Goto behaviour

A behaviour which will accept coordinates as the input and will then compute the fastest way to go to this position and drive it with the robot. This will not include any collision detection or path planning.

3.3.2 Follow behaviour

A leader robot will be set arbitrarily and a robot follows this leader in a certain distance wherever it goes. To move around a similar algorithm to the Goto behaviour will be used, again without collision avoidance or path planning.

3.3.3 FindBlob behaviour

The robot tries to detect a landmark of a certain color or pattern in the environment and focuses on this landmark. The robot will also follow the landmark if it is moving and will try to keep a certain minimum and maximum distance.

3.3.4 FindClosest behaviour

This subbehaviour will communicate number (e.g, the distance to a landmark) between robots, then perform a computation on these numbers and get an AID (global unique agent name) as a result.

3.3.5 Lead & Follow behaviour

These subbehaviours will enable one robot to lead all other robots in the group and the others to listen to his movement commands. All the leader is going to do is propagate his current position

to the following robots, they will then compute their target position on their own and issue the corresponding commands to get to that position.

4 Design

The original design was my design for the first third of this project, when I assumed I would use the combination of RobotMgr and self written C++ communication like the previous projects did. These would be used to build the complex behaviours while reusing the already written subbehaviours by previous groups. As explained in section 5.1 I decided to switch to the more promising combination of PLAYER and JADE. Therefore the original design had to be revised and is given in section 4.2. To show the original ideas on which the revised design was based I will explain the first design here, although it was not implemented in a fully working way.

4.1 Original design for C++ and robotmgr

To avoid "reinventing the wheel" during the research a modular framework for recognition and behaviour is planned. It will provide a way to communicate for the different running tasks and robots in the group. A central server program on each robot representing the intelligent agent will coordinate the actions and decide which movement to make. Smaller Subprograms, running in different threads will be able to access the sensor data and propose possible movements to the server, weighted with a value corresponding to the importance of this action. The server will then select the most important action and execute it. This will enable the brake module to stop the robot when an object was hit, for example. The framework will do threading via its own wrapper around linux pthreads, or robotmgr's taskBase pthread wrapper.
Due to the modularity components will be easily to replace by a different implementation. These are the most important components I had in mind for the framework:

- Basic Behaviour:
  Behaviour to enable the robot to navigate without destroying anything such as stopping when bumping into anything or collision avoidance via sonar.
- component for object recognition:
  Modularity will allow the recognition of objects like an orange ball, any color or landmarks like the artificial p-similar ones. This module will set the robot in a state where he scans his environment once and align to the object so that it is in the middle of his point of view. From the angle of the camera, the angle of the robot, the known position of the robot and

the size of the object the position can be estimated. If multiple robots located the object then the localisation can be even improved by combining the gathered data.

- component for object tracking:
  Probably not more than one module is needed here, this part of the program makes sure that the robot follows the object with the camera. In the simple version this happens by scanning every image and trying to center the landmark. A more complex version will estimate the movement of the landmark and try to take up less computing power while still tracking it. This module could also be seen as en extended version of the object recognition component.

- component for task assignment: Decides which robot in the group is to do which task, tasks are defined in the behaviour modules. To decide this cost functions well be implemented to compute the optimal decision in regard to either speed, energy consumption or other priorities.

- component for behaviour:
  Different models of behaviour like "follow the object", make a certain formation, shoot a soccer ball, give out information about the object etc.
  Some of this behaviour is already implemented somewhere but "hard-coded" into small programs, the goal is to be able to reuse these components for the next object recogniser developed or to choose from different behaviours easily.

As you can see in figure 5, a central core of the software framework will be the server part. The functions of each component is going to be explained in the next subsections.

4.1.1 The server object

The server object is instantiated by the program written by the user or can be instantiated on its own. It will take care of setting up the robots rFlex controller as well as starting a listening thread to receive messages of this framework. For this task an own communication pthread is started which receives the messages and dispatches them by calling functions of the server object depending on the contents of the message.
The server object will keep records of the tasks started contained the pointers to the tasks, so messages can be forwarded to them. This is symbolized in the schematic in figure 5 by Thread

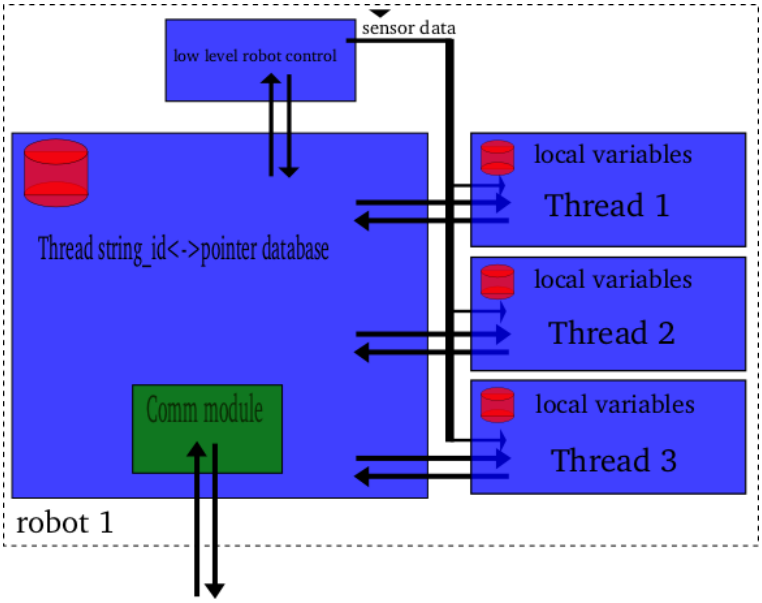

Figure 5: Schematic of the original designed architecture

string id connected to a pointer.
The server object will have at least the following public functions:

- int Server::startTask(char *name)
  This can be used to start a Task directly, i.e. without having to send a message to the server object.
- int Server::sendMessage(char *destination, Message *msg)
  This can be uses to send a precompiled message object to other robots.
- RobotMgr* Server::AccRobotMgr()
  If commands should be sent directly to the underlying robotmgr object, the pointer to this object can be read here.

4.1.2 The tasks

Each task will use its own pthread and run parallel to the others. The tasks will be able to access sensor data directly from the robotmgr. Each Task can communicate to the others through

sending a message which is then forwarded to the target thread by the server object. Thus the task will provide a function such as

- int Server::getMessage(Message *msg)
  This enables the task to receive messages objects.

4.1.3 The message class

The Message object consists of 5 char arrays which allow the flexible use of this message. The message definition is going to be as following:

```
    class Message {
public:
char senderHost[5]; e.g. mag1
char targetHost[5];
char targetThread[10];
char targetVariable[10];
char dataString[10]; flexible data payload of msg
Point dataPoint; Point data payload of msg
```

4.2 Revised design for Java, JADE and PLAYER

The code for this task will only based on the work done by the previous groups in a very small degree as they mostly used a different programming language and most important a different robot management platform, robotMgr instead of PLAYER. The reason for my switch to the PLAYER platform will also be explained in section 5.1.3.

The programming language which will be used to realize the framework is Java as this is supported by both the main software components used, PLAYER and JADE. JADE will be used as the platform and framework for inter-robot communication , it also provides all the necessary means to start and coordinate behaviours and much more. The Java code should be compileable on any POSIX machine which provides the necessary libraries. The emphasis was on object oriented code which allows to modularize smaller parts and change them without having to change any other code.

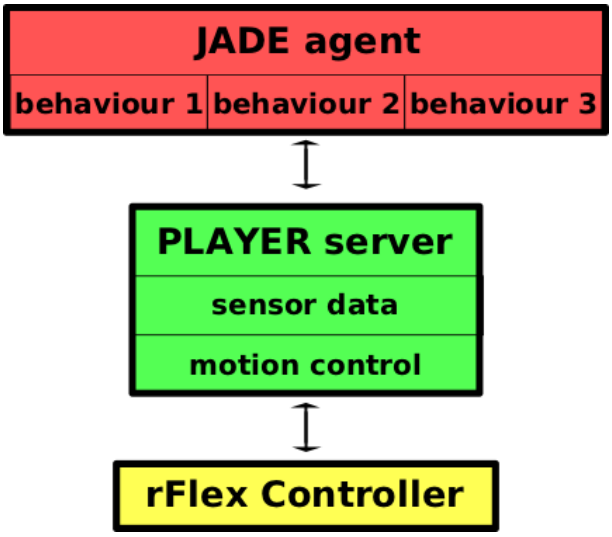

Figure 6: Schematic of the revised two-tier architecture design

The architecture will be a two-tier architecture as described in the section 2.1 and similar to the one discussed in [19, p.300]. It is displayed in figure 6.

The JADE part is displayed in detail in figure 7. The PLAYER part takes movement commands for the motor and other devices such as the PTZ unit and provides regular updates of sensor data to the PlayerClient proxy in the PlayerAgent Java object. A central data storage, a Java hashmap, in the PlayerAgent object allows the shared storage of inter-behavioural data and a list with AIDs of known robots is kept in the playerAgents array. Every behaviour, displayed here on the right with some of the more important ones such as the behaviour updating the list of known agents and the one receiving position updates and other data from those agents, storing them in the hashtable. The number of running or possible behaviours is not limited by the design but by the available hardware resources.

4.2.1 PLAYER part

Effectively the PLAYER software is replacing the part of the robotMgr in the original design due to reasons explained in section 5.1.3. The PLAYER program itself is running out-of-the-box and needs only little customisation to the robot on which it is used, the drivers which should be used have to be declared and initialized with configuration values in some cases. I made the following setup for the PLAYER server:

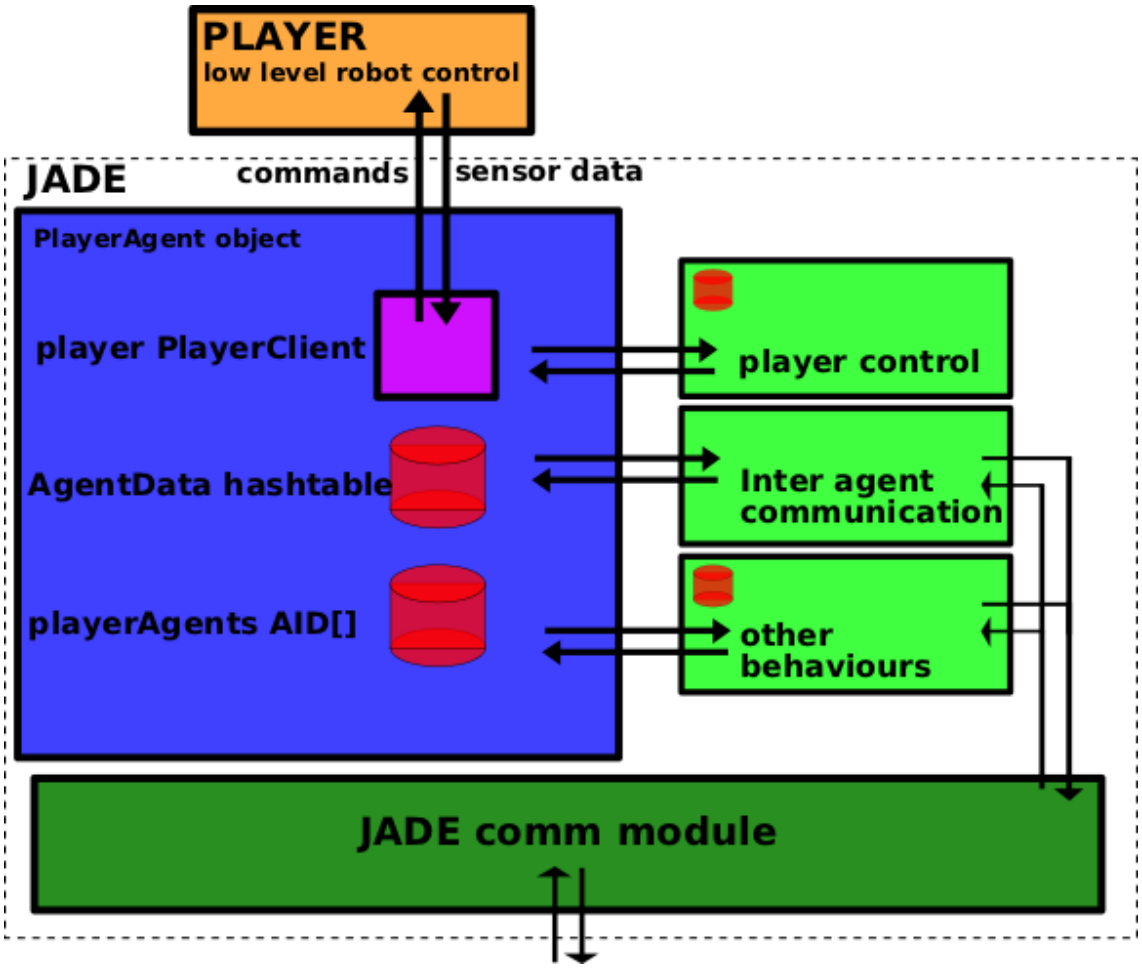

Figure 7: Schematic of the JADE part in the revised design of the software architecture

The PLAYER server will be running on every robot and listen to the port 6665, which is allowed by the firewall of the robot. The configuration file of the robot will define support for the different sensors the robot provides and also declare the position of those sensors, so that the visualisation in other programs such as playerv will display the correct sensor reading. The interfaces provided by PLAYER on a Magellan pro robot are the following:

- position
  This most basic interface allows the reading of odometry data from the robot as well the setting of the speed and direction of the motors.

- camera:0
  The basic, uncompressed raw camera images as they are provided by the LINUX v4l2 driver of the capturing card.

- camera:1
  This is an interface to compressed camera pictures from the same source as camera:0. I used this together with playerviewer to see the live camera stream on a remote copmuter.

- blobfinder
  This again uses the camera stream from camera:0 to detect colored blob in the pictures using the CMVision algorithm.

- sonar
  The array of 16 sonar sensors on the Magellan pro robot. Although they are managed by two separate embedded boards on the robot, they are accessible through one interface in player

- IR
  The array of 16 IR sensors on the Magellan pro robot. Although they are managed by two separate embedded boards on the robot, they are accessible through one interface in player

- Bumper
  The array of 16 bumper sensors on the Magellan pro robot. Although they are managed by two separate embedded boards on the robot, they are accessible through one interface in player

4.2.2 JADE part

The framework for the different behaviours is based on JADE and the Java-Client for the Player software also described in section 2.5. JADE allows a very easy way to receive and send messages, organise and manage multiple threads (called Behaviours here) and testing using the dummy agent to simulate messages sent in the group of agents. Behaviours are scheduled by a simple round robin algorithm and are non-preemptive, so one blocking behaviour can stop the whole robot until it is unblocked. There are different kinds of behaviour templates to choose from, the differences are for example the way they terminate (never, after one execution, user-defined).
On every robot one JADE agent will be running which hosts a set of the following behaviours. The robot hardware will be controlled by the basic behaviour which issues the final movement commands to it.

- Basic Behaviour:
  Behaviour to enable the robot to navigate without destroying anything such as stopping when bumping into anything or collision avoidance via sonar.

- component for object recognition:
  Player offers different drivers to provide the so called blobfinder interface, which can be considered as an object recognition interface. I used the CMVision driver which uses the CMVision algorithm to extract blobs of colours with a certain given range of YUV values. To use the artificial landmark detection algorithm the driver part in PLAYER would have to be modified, thus returning a blob value with a certain color when an artificial landmark has been detected in the camera stream.

- component for object tracking:
  Together with the blobfinder, this behaviour uses the position proxy to get the best view of the object. The PTZ unit is not used so far because this would require addition effort for determining the movements of the robot.

- component for task assignment:
  In the case of the group following the blob one of these components determines the leader, another one determines the optimal position of the following robots in the formation.

- component for behaviour:
  This component is implemented as behaviours in JADE like most of the other components

as well. The modular architecture of JADE allows behaviours to control other behaviours, this is what these components do. They coordinate running behaviours, start new ones and also provide the basic message receiving capabilities of the agents.

The first complex behaviour I implemented was a behaviour which starts by every robot looking for a landmark, then communicating if they found it, selecting a leader and following the landmark in a formation.

4.2.3 Java-client for PLAYER

To be able to access the PLAYER server which is written in C++ the third party Java-client[10] for PLAYER is used. Basically the Java-client is just a very simple Java interface exchanging messages with the PLAYER server. A proxy for every PLAYER device has to be instantiated at the beginning of the program, later simple function calls to those proxies allow communication with the PLAYER server. No special configuration is necessary and the use is absolutely transparent.

5 Implementation

5.1 First Implementation with C++ and RobotMgr

5.1.1 First steps with RobotMgr

First steps with the software used in the previous projects were done by modifying existing code to enhance the features. After getting familiar with existing examples for the robotMgr software an own behaviour was added which would use the previously unused bumper sensors on the robot. The new behaviour would run in parallel with other running behaviours and listen on bumper sensor events. In the case of a bumper touching a nearby surface a variable would be set which then stops the robot,causes him to turn around, go backwards or other reactions defined in the main program.
Next programs which were written for the robotMgr were test programs to remotely control other robots or exchanging messages between them. Most of those smaller test programs remained unfinished as the underlying robotMgr framework was not working completely.

5.1.2 First steps with C++ networking

On the higher level of group control I started with reading and modifying the existing programs from Slim and Raydan. It quickly showed that those programs were written to fit exactly their respective purposes and extending them to include other behaviours such as the landmark discovery proved to be much more complex then I expected. The existing way those programs worked only allowed a very limited way of reacting to events, so I decided to only keep small parts of the code and re implement most of the other parts to allow a greater flexibility for future behaviours. Special attention was given to have a flexible message format to use with different behaviours to communicate between the robots, a sort of automatic behaviour to synchronize sensor data between the robots and threaded handling of simultaneous behaviours. The original design included a descriptive outline to those features.

5.1.3 Reasons for the switch of the underlying software

Both the message handling and the threading quickly grew more complex and required most of the developing effort in contrast to the original goal of this research work, an autonomous group

behaviour architecture. To avoid this problem an existing framework with easy to use communication means between the robots is needed. A research on existing multi-robot communication frameworks in C++ brought no new discoveries and so I decided to reconsider the use of a different Programming language than C++. At that point JADE was suggested by my colleagues after a presentation of me about PLAYER.
PLAYER was chosen as a replacement for robotmgr for several reasons.

1. As Ben Miners is just finishing his master's thesis he was very busy during all the time of my project and will be probably to provide even less support for his program after he has left the university. There is no existing programmer's guide to the robotMgr and even with some mail support from Ben it took me quite some time to write basic programs to test the software's capabilities and understand its possibilities. There is an automatically generated DOxygen documentation available, but this won't help to much with basic understanding of the software.
   PLAYER on the other hand has several documents such as papers and programmers guides to help the understanding of the basic structure. An active mailing list is trying to help users with problems and advices. The mailing list archive is also a valuable knowledge base. The software is under constant development and this way improved with every new release.

2. PLAYER features 2D and 3D simulation using the STAGE and GAZEBO software. This feature is an invaluable help in developing complex behaviours such as group behaviour without risking the expensive hardware in possible collisions or accidents. Large populations can be simulated without the necessity to buy those robots, software can also be tested on different hardware than the originally intended one by using the interface abstraction for devices. Additional hardware such as SICK laser scanner can be tested for their efficiency in improving the robots capabilities and, if they are bought later, used right away with the already written software. The software could even be used with classes to allow a larger number of students to learn about this topic and have their own solutions to given problems. Also different environments such as close quarters in a hospital or an open field can be conveniently simulated by just drawing the map with a graphic or CAD program.

3. As mentioned above the PLAYER software already supports a vast collection of hardware, so future purchases of a laser range finder for example will work right away in the robots - without having to modify the robotMgr, just by changing some lines in the PLAYER configuration files.

4. This concept also allow the exchange of an complete implementation of some project with other researchers or people interesting in it - all they have to provide is a working installation of PLAYER, then they could just execute the Java binaries of my project for example.

5. PLAYER's included algorithms for tasks such as a goal-seeking obstacle avoidance algorithm (VFH, Vector Field Histogram) or an Adaptive Monte Carlo Localization (AMCL) add a whole new dimension of possibilities for work with the robots in the PAMI lab. This functionality will robotMgr probably never be able to offer - as it is just a small project compared to PLAYER.

5.2 Preparation of the robots for PLAYER

5.2.1 Requirements of PLAYER

The requirements for PLAYER include "a recent version of GNU GCC" which was not available on the about 3 year old RedHat installed on the robots since they were delivered. I also found no way to update those packets without probably messing up the whole installation as the GCC is a fundamental part of every distribution. I tried an upgrade of RedHat to the newer version which is called Fedora (Core 3) now, but I found no reports op people doing this successfully on the internet and my own attempts were unsuccessfully too. After discussing the matter with the other members of the research group on one of the weekly meetings we decided I would install Fedora Core 3 from scratch on the robots.

5.2.2 Installation of Fedora Core 3

For the installation I connected a local monitor and a local keyboard to the on-board computers, but there was no way to connect a CD-ROM drive or similar to install the distribution from. After researching a bit on the internet I found a solution for this problem: It is possible to connect a floppy drive to the robots, but an install from floppy was only supported by Fedora Core 1. So I started installing Fedora Core 1 on the robots after making both the bootable first floppy and the second floppy which includes drivers for the on-board network interface. The files were downloaded automatically from a HTTP server I found in the internet and the whole installation process took about 4 hours. After the initial Core 1 installation was configured and tested I continued the installation process with upgrading to Core 2 and then Core 3, which took about 3 hours each again. To upgrade the distribution some tricks had to be used which I found on the

internet and which are not in the scope of this research work. The special software installed for this project includes the Java JRE 5.0, PLAYER, JADE, JavaClient for PLAYER. JavaClient also requires ANT to compile on the robots.

5.2.3 Configuration of PLAYER and STAGE

Both PLAYER and STAGE need configuration files to be customised to the type of robot used or simulated. The configuration files for PLAYER include the physical location of the sensors on the actual robot together with their angles. These values were calculated by my by taking rough measures on the ATRV mini itself, they could most certainly be improved with better measurements, but I assumed that most work would be done on the Magellan pro's anyways, where those measurements are much easier due to the round and symmetric shape of those robots.
Every driver used by PLAYER has to be instantianted by lines such as:

```
    driver
(
name "cmvision"
provides ["blobfinder:0"]
requires ["camera:0"]
colorfile "color.cfg"
)
```

Where this is the section for the CMVision blobfinder algorithm. This driver requires the name of a camera interface ["camera:0"] and will then provide a blobfinder interface ["blobfinder:0"]. The only configuration option is the name of a colorfile which specifies the range of color detected by the blobfinder and the names and colors associated with those regions.
To get those values I used the videoplayer program from the third party software section of the PLAYER website. It allows to remotely display the picture the camera is providing at that moment. I then made a screenshot of this picture using GIMP and changed it to YUV colors using a plug-in for GIMP. The colours would now seem to be distorted because only the values are changed to YUV values of the original colour, but be displayed as RGB values. The yellow landmark which I chose as a target for the blobfinder was identified easily even in the distorted colour space and I selected colour values from all over its surface to find a range for

Color	Value range for blob
Y	170-240
U	60-90
V	130-145

Table 1: Colors and corresponding values in the YUV color space used to configure the CMVision algorithm of the blobfinder

the configuration file of the CMVision algorithm. The values I finally decided to take are ones in table 1

Other values that had to be set in the configuration file were constants to translate the abstract distance values reported by the rFlex interface to real distances in mm. Those constants were computed by driving the robot along a predefined distance and comparing the reported rFlex values with the actual distance. Those three values are connected in this relation: $real\_distance\_mm = \frac{rFlex\_distance}{odo\_distance\_conversion}$. These values are said to be different from robot model to robot model, so I would have to compute these values again for the ATRV-Mini's if they need real distance values. The complete configuration file for the Magellan pro is attached in the appendix C

5.3 Second Implementation in Java with JADE and PLAYER

5.3.1 First steps with PLAYER/STAGE - functionality test

This project used PLAYER in the version 1.6.4 and STAGE in the version 1.6.2.
After the installation and configuration of PLAYER on my laptop I started with running the example world file given by stage. In this worldfile 3 PLAYER instances are started, one for every moveable robot in the demo. I tested the playerv tool which allows the display of sensor and other data of the simulated robots and moved them around by using the "playerjoy" keyboard control program for PLAYER. I also tried connecting multiple instances of the playerv program to the PLAYER server and connecting multiple playerv instances to the different robots to view sensor data of all of them. All these tests were successfull and I found the programs intuitive and easy to use. Two small videos of this are on the CD provided with this report and also on my website [21].

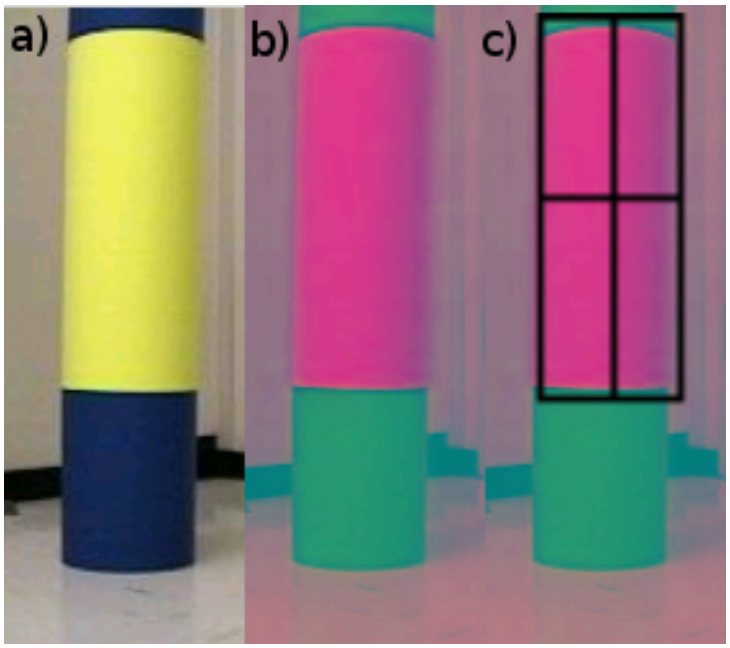

Figure 8: Example for the blobfinder:
(a) Original camera picture, (b) transferred to YUV, (c) with the blob found by the blobfinder

5.3.2 First steps with Java and JADE

I used JADE version 3.3 and the Java RE 5.0.
Because my experience in programming in Java was limited I started with looking at the given code and trying to modify a working example. My biggest problem at that point was the setting of the CLASSPATH variables which determine where the Java classes are to be found on the computer -a concept which is quite different to C++ programming. Luckily all necessary tools were already installed on my laptop so I didn't have problems to compile Java programs - unlike the fedora distribution on the robots which caused more problems when setting up the working environment.
First steps with JADE were made by testing the PingAgent given in the example code in the distribution of JADE, this simple agent reacts to a specific kind of message by sending back a reply upon receipt. This message is sent by using a so called dummy agent which can be started in the JADE GUI. It allows to select the type, target and content of a message and to send it afterwards.

5.3.3 First steps with the PLAYER Java-client

The Java client for PLAYER[10] is developed by two persons at the time and not too much documentation is available, the one on the website being a bit outdated. The developers answered to my requests on the mailing lists and kindly pointed me towards the right use of the client library. After those initial problems everything was working without further issues. Basically the Java-client is just a very simple interface exchanging messages with the PLAYER server and not much can go wrong there. I wrote short test programs in Java to test the position and blobfinder proxy and they worked.

5.3.4 Simple behaviours: Goto, FollowTheLeader

After implementing very small behaviours like Stop which simply stops any ongoing movement at that time I started with implementing the Goto behaviour known from the robotmgr. The Goto behaviour itself is a tickerBehaviour which is called in a certain frequency as defined in the program. When created this behaviour saves the target point and with every call of the onTick() function it will compute the angle and distance to the target point. If the angle is below a certain threshold the robot will move towards the target point, otherwise turn to lower the angle to the target. When the distance to the target is also below a certain threshold the robot stops.

The next behaviour to be implemented was called FollowTheLeader. This behaviour is started by sending an agent a message with the content field containing the following:
followtheleader:*leader_agent_name*
an example would be: followtheleader:*player1*
The receiving agent sends a message to the leader subscribing to its position updates which are then sent by the leader to every subscribed agent. Upon reception of these position updates the robot executes one step of the goto behaviour to move to a certain new position relative to the position of the leader.

5.3.5 Simple behaviours: FindBlob

The FindBlob behaviour was implemented in an iterative process because of problems with the underlying STAGE simulation software. The STAGE software would always report a blob even if none in sight of the virtual robot, together with garbage values for the blob. After I localised this problem I wrote a small workaround ignoring those fake blobs. This was possible because the fake blob would have a wrong value set in the windowsize, which provides the size of the camera picture. On my system the reported false value is always $22648x5$, which allowed sorting out those blobs. Another problem with STAGE is that every robot in sight is reported as a blob whereas in the real application only blobs of a certain color are detected. This made some additional code necessary to use the same code on both the real robot and in STAGE, but in total those were just 4 lines.
On the robot the blobfinder uses the CMVision algorithm[5] which is available as a driver in PLAYER. I configured it as described in section 5.2.3.
I used a large landmark so the distance finding algorithm in the behaviour would resturn similar results on both the real and simulated robot. This distance estimate was then used to tell the robot to come closer if he was to far away or back of if he was to close.
If no blob was found in immediate view of the camera the robot would start to turn slowly to the left until a certain time passes, in which he should turn around 360 degrees in total.
If a blob was found the robot turns in order to center the blob in the middle of his view. This will sometimes lead to a longer process of moving until the robot stops in real life because the blobs reported from the CMVision algorithm sometimes change their centre of gravitation due to light change or something else.

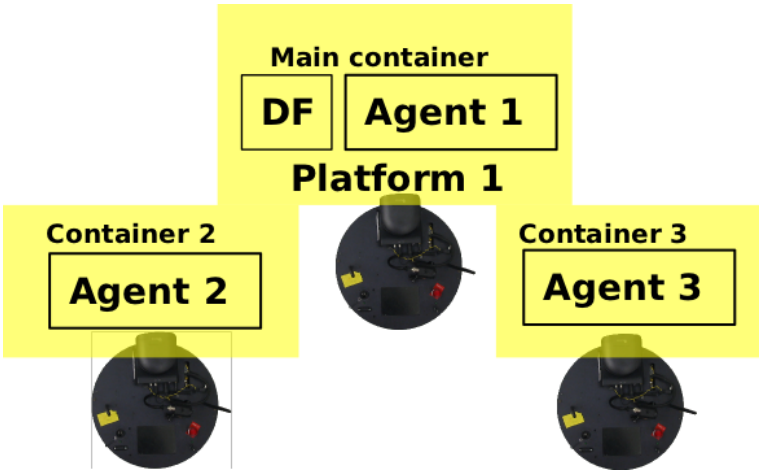

Figure 9: One JADE Platform hosting 3 containers

5.3.6 Coordinating the agents

To coordinate the actions of the group's robots the approach of domain-independent conventions as described in section 2.2 was used. Every robot executes the same algorithm with the same input values and thus every robot comes to the same plan of actions. This requires either some common shared Blackboard on a central server or messages which are sent to every robot.
To keep the autonomous character of the robots I chose to have the least centralised services as possible, thus the solution with sending messages to the robots directly was selected.
This still leaves the problem of how to get the other robot's AIDs in the beginning to send messages to them. Past project solved this by using a fixed list - which forced users to use the exact combination of robots every time. To improve this a very low level server is still needed because JADE supports no simple broadcast option to send messages to everyone. The JADE solution to this problem is to connect all the running JADE instances to one platform as described in [8] and displayed in figure 9. One running JADE instance (in my setup my laptop) hold the main container, all the other JADE instances connect to this platform and use its DF (Directory Facilitator). Every agent announces its presence to the DF on startup. Then periodically every agent queries the DF for the list of known agents, whose AIDs are saved in a data structure.
When a message has to be sent to every robot the small SendToAll behaviour I wrote can be used which will cycle through the known agents and add their AIDs are added as targets to the message before it is sent.

In my example behaviour those messages are sent after looking for the blob. The content of the message is a String with colon separators containing both the position of the robot, the size of the found blob and if a blob was found at all.
To store the data which is sent from other robots a Java hashtable is used. Every time a message with certain characteristics is received its data content is simply stored in the hashtable with the sender's AID as key value. This data can then be parsed by the behaviour in any way it's needed. The listening behaviour will quit in my case when a message from every robot known has been received.

5.3.7 Group formation

One of the goals of the framework was to have as little constraints on the group of robots as possible, for example on the number of robots in the group. A behaviour to form a group formation like in the previous projects by Slim and Raydan (described in section 1.3) always assumed a fixed number of robots, 3 in their case. On the other side their algorithm was already a quite computationally intensive brute force approach to find the best position for each robot in the group. The algorithm computed the cost of every possible combination for positions of every robot in the group in the formation (which is $O(n!)$), doing this once for every degree out of $360°$ in the orientation of the formation is almost negligible there.
To have a smooth movement of the group following the landmark this algorithm should be executed at least every second. So this computational cost would rise factorial with the number of robots, leading to an infeasible algorithm for even small numbers of robots.
For these reasons I decided to find a smarter solution for a group formation algorithm. The goals were to find an algorithm which would allow robots to follow the leader in a certain formation with the minimum amount of messages and minimal computational effort necessary. The final algorithm chosen is $O(n)$, thus allowing even a large group of robots to follow the leader in a formation, and was developed on my own. Because so far no collision avoidance is implemented the initial building of the formation may cause some collisions between robot while they are trying to reach their target point, but this can easily solved in the future by implementing a simple avoidance algorithm using the sonars for example.
After the initial attempt of every robot to detect the blob in their environment they will send out a message containing the size of the found blob. These values are used subsequently to compute the leader - and the position reported in the same message as input for the formation of the robots following the leader.

Based on those values which are assumed to be all different (in a real application a valid assumption) an order of the robots is established by every robot computing its own and for every other robot the respective distance to the leader, an operation which is $O(n)$. It then saves the AID of the sender of the message with the next bigger distance to the leader than its own - this robot is selected as the "child", the selecting robot will be called parent from now on. Due to the assumption of mutual exclusive distances to the leader this establishes a common order on every robot with exactly one "child" entry for every robot but the leader which is not the child of any other robot. The leader is the only robot which is free to move wherever it wants to, but is a parent itself and has a child. The robot furthest away from the leader is a child of another robot, but no parent itself. Once this order is established each robot starts to send out a Follow message only to his child, telling it to move to a position in a certain distance behind the position where the parent robot is supposed to be. The important part is the word "supposed", this means that a position change of the leader will result in subsequent fast updates of target positions for every robot in the group, making it move to the new position much fasten than following only the parent would be.

5.3.8 The final complex behaviour

The complex behaviour chosen to demonstrate the proposed architecture is a group following a moving landmark in a certain formation. Its structure is displayed in 10.
Several subbehaviours mentioned in the previous subsections are combined together in a finite state machine behaviour, one of them being a JADE ParallelBehaviour which executes two other behaviours in parallel. This will let the robot scan the environment for a landmark and additionally listen for new messages from the other robots.
Depending on the outcome of the computations in FindClosest either the Lead or Follow subbehaviour are started. The FSMBehaviour class allows easy changes in the order or additions of new subbehaviours. The state machine worked immediately after putting the behaviours together, although some testing with simpler dummy state machine were done before to understand the way they are built in JADE.

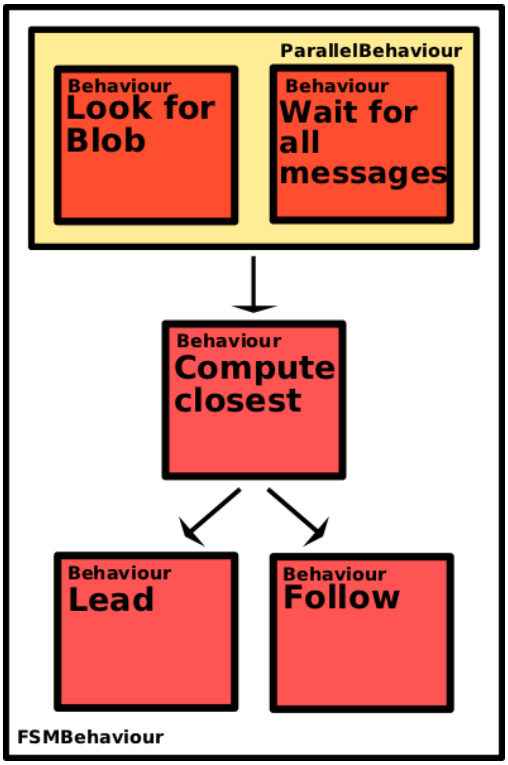

Figure 10: Schematic of the finite state machine used in the complex behaviour

6 Testing of the implementation

Testing of the implementation of the architecture was made in both the STAGE simulator and real world robots, the ATRV minis and Magellan pros of the lab.

6.1 Simulation in STAGE

Parallel to the development of the framework new features were constantly tested in the STAGE simulator using the JADE dummy agents to send and receive arbitrary messages to the single agents. The problem of giving all robots the same orientation and coordinate system was solved by starting the robots on the same position and displacing them with the mouse afterwars, just as it was done with the real robots in other projects where the robots were switched on, one after the others, on the same tile on the floor, facing the same direction. This way all actions and goto-behaviours will result into same actions for every robot which is important for behaviours such as the Follow behaviour.
To have an environment with enough space for larger groups of robots to drive in I modified the "hospital" map provided by STAGE by deleting most of the inner walls. The "ghost" object from the STAGE test file was used as blue landmark while the robots have a red color. To test the final group behaviour the robots were spread randomly over the map and the blue landmark was set somewhere in their middle. After the behaviour was started the robots tried to detect the landmark in their environment, communicated their results to the others and formed the formation afterwards just like expected. Due to the lacking collision avoidance it is sometimes necessary to displace the robots with the mouse a bit to keep them from being stuck together.
An example run of the simulation is shown in figure 11 and figure 12.
The single stages shown on the pictures are described here:

a The random setup of the group and the landmark. The robots were moved and turned by hand, so its not really random.

b After sending the corresponding message to every robot they start to turn and look for the blob, stopping when it is centered in their field of vision. After that the messages are exchanged and the robots begin to move into the formation behind the leader.

c The robots reach their goal position behind the leader and stop.

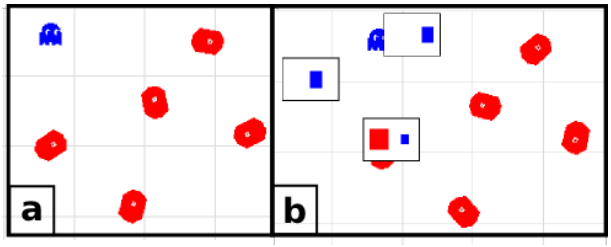

Figure 11: Simulation of the complex behaviour: a) random setup of group, b) robots are looking for the blob

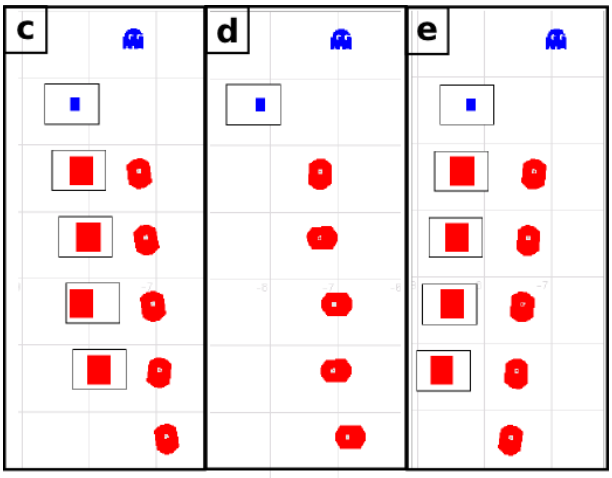

Figure 12: Simulation of the complex behaviour:c) initial formation, d) landmark is moved, robots move to new position, d) robots stay in new position

d The landmark is moved and the robots start to move into their positions in the formation behind the now turned leader.

e The robots reach their position in the formation again.

The boxes which can be sometimes seen next to the robots are the visualisation of the blobfinder data in stage.

6.2 Real life testing

The first test of the Goto behaviour on a real robot - The Magellan pro I used for testing - resulted in it turning around for a while when it was only supposed to turn 20 degrees. I realized that the

configuration file, especially the *odo_distance_conversion* values had to be set first, additionally I implemented a security check in the PlayerAgent to prevent the robot from ever moving faster than a predefined limit.
Other problems in real world testing were due to the limited space in the lab, extra care had to be taken with combined simulation/real world testing using STAGE in which the virtual robot would lead the real robot.
The FollowTheLeader behaviour was working just like in the simulator, no further changes had to be done. A problem with a mixed set of robots was again the difference in the *odo_distance_conversion* values which resulted in growing discrepancies between the two world views of of the leader and the follower in a two robot case with min1 and mag3 for example. Even worse are the differences in measuring the angles of the robots. Because the ATRV minis are using a four wheel powered approach for turning the odometry data is quite inaccurate. Tests where I let the min1 robot turn for 90 degrees and then turn 90 degrees in the other direction would result into an offset of approximately 10 degrees between the old world frame of the robot - thus giving a huge error after some movements which include turns. The Magellan Pro robots don't have these problems in this scale as their two wheeled method of turning is far less susceptible to inaccuracy.
This issue is discussed extensively in literature, and there are error models to describe it, for example in [19, pp.186]. Basically some advanced localization attempts have to be made if the odometry data is of only poor quality. PLAYER includes some virtual devices for this task such as the Advanced Monte-Carlo-Localization, but these often require sensors like a SICK laser scanner. This problem, described as "one of the most pervasive perception problems in robotics" [16, p.908], will not be considered further in this report and is a good recommendation for future work as stated in section 8.
The next behaviour tested was the FindBlob behaviour. As I knew from monitoring the results from the blobfinder interface of the real robot the real blobfinder detects a lot more than just one blob even if centered directly on the landmark. Most of the unwanted blobs are in the area of the wanted one but incur due to the used CMVision algorithm. These values were ignored in the FindBlob behaviour by sorting the blobs reported by the interface and only looking at the biggest ones. Also small blobs below a certain size can be safely ignored as they can result from a noisy camera picture. After some filtering has been done on the reported blob the coordinates of the largest blob's center of gravity are then used to give the robot commands to center the camera on this blob. This was tested initially by just giving out the commands the behaviour would give on the commandline, after those were confirmed to be right the move commands were

Distance	Size of largest blob
$\leq$10 cm	50,000
100cm	5000
$\geq$200cm	1000

Table 2: Listing of distances of the landmark and the corresponding amount of pixels reported by the blobfinder interface

given to the robot. Because the size of the real camera picture is bigger than the simulated one I also had to change the FindBlob algorithm a bit to give a blobsize relative to the camera window size.

While testing the blobfinder trained to recognise the two-colored large soccer field poles in the lab I found the values displayed in table 2 for the Area of the main blob.
Other problems in real world testing were due to the limited space in the lab, extra care had to be taken with combined simulation/real world testing using STAGE in which the virtual robot would lead the real robot.

7 Timeline

7.1 Comment on the original schedule

The original schedule was based on the assumption that the software setup in the lab would be sufficient to use it in the project, and no major switches to other software should be necessary. During the initial research and test phase of the project this assumption turned out to be invalid, as the Software basis used by the formed smaller projects, C++ and robotmgr didn't provide any framework like means to simplify the handling of messages between the robots. As communication between the robots via messages is a basic requirement for an autonomous mobile robot system this made the development of the C++ siftware unnecessarily complicated and required a lot of work for non research topic area programming of message handling and multi-threading for example. The weak documentation and the fact that the only programmer of the robotmgr software was about to finish his master's thesis didn't help with those programs either, as I couldn't get important message handling features in robotMgr to work in my test programs. After extending my research on multi-agent frameworks for other programming languages than C++ and talking to people using the robots for bigger projects such as Bram, I decided to try JADE as the framework for inter-robot communication. My discovery of PLAYER and the STAGE simulator, also including a JAVA client software to control the PLAYER server forced me to rethink the original timeline in the third week. I started switching completely to PLAYER, which also required a newer LINUX version on the robots. The test robot was ready for use in week 6 which allowed my to start on the actual work on the framework. At this point the original schedule was already assuming the work on the second objective, the cooperative box pushing, which obviously couldn't be done at that point due to the above reasons.

7.2 Revised timeline

The weekly reports I wrote are available in the Appendix A. Here follows a very short summary for them:

- First week (May 9.-14.):
  Introduction to the lab, the people working there and the software. Organising keys and accounts for the computer network. Searching papers and books which cover the projects topics.
- Second week(May 17.-20.):

Planning of the research topics. Modification of existing software to let a group of robots choose the one nearest to a landmark, this one now follows the landmark if it is moved. Extensive review of the old code.

- Third and Fourth week (May 23.-June 3.):
  Using the experience gained from reading and modifying the old code, the work the framework was started after evaluating different possible solutions the best method known to that time was selected. The framework was tested with some basic behaviours like collision avoidance and collision detection.
  After the discovery of PLAYER the switch to this platform was considered and PLAYER/STAGE was tested locally on my computer.

- Fifth and Sixth week (June 6.-June 17.):
  Unsuccessful attempts to extend the preexisting communication network in C++ in the Fifth week, the Sixth week was spent on updating the OS on the ATRV mini.

- Seventh and Eight week (June 20.-July 1.):
  In the Seventh week I explored the possibilities of PLAYER on the real robot and prepared a talk on PLAYER which I held for the focus group on Friday. The Eight week was spent with looking into JADE which I had discovered on the previous weekend.

- Ninth and Tenth week (July 4.-July 15.):
  After having finally decided on JADE and PLAYER as the basis for the architecture I started implementing it and some basic behaviours during those two weeks. I would test new behaviours in the STAGE simulator first and try them on the real robots later.

- Eleventh, Twelfth and Thirteenth week (July 18.-August 10.):
  The complex FSM behaviour was finished in the Eleventh week, the two following weeks were mostly spent with compiling the report and writing additional documentation for users of my framework.

8 Recommendations and future work

8.1 JADE

Even though I haven't used Java for more than small assignments in undergrad classes I had no problems with using it together with JADE. There is excellent documentation on JADE and I found the concepts very easy to understand and use. I would advise anyone who wants to start future work on group behaviour to use JADE instead of writing own implementations of message handling routines. Using Jade I was able to program basic behaviours which took other groups weeks in days only - of course being able to simulate the robots in STAGE helped a lot, too.

8.2 JESS

Because Jade lacks own functions for machine intelligence it would benefit from combining it with a machine intelligence software like JESS. This has be done at other places such as in [9]

8.3 Further use of PLAYER drivers

Now that Ben Miners is finished with his master's thesis future programmers using robotMgr as the base for their programs will most probably have to debug possible bugs in robotMgr by themselves, also Ben won't be able to introduce them to the basics of robotmgr. Although robotMgr is indeed a good program and successfully used in many applications in the PAMI lab so far in my opinion PLAYER is the better choice for future work with the robots. The Java client for PLAYER worked flawlessly for me, the whole project is developed and successfully used on more than 30 universities all around the world which shows its popularity. In my project I haven't even used the more advanced interfaces provided by PLAYER such as self-localization, map building, way planning and so forth. Each one of those features make a good project to explore alone, and show the superiority of a global developers community. The fact that PLAYER also supports different platforms than just rFlex alone is also important now that iRobot stopped producing civil robots other than pink autonomous brooms. Programs written for PLAYER can be used with only little extra effort on robots based on a different hardware platform but similar sensors. The simulation engines STAGE and GAZEBO offer also a big advantage compared to robotmgr, as they allow quick debugging and fast prototyping without having to fear to damage the real robots due to programming errors when they suddenly run wild.

8.4 Enhancements for PLAYER

The modular concept allows easy extension with own drivers, which could be the artificial landmark detection from the old project for example, using the blobfinder interface and basing on the CMVision driver which does indeed a quite similar task. So far such a landmark recognition driver is not available for player and this would be a suitable contribution from the PAMI lab to the PLAYER community. Instead of returning colored squares for detected colors from the camera this algorithm could return those square if an artificial landmark was detected.

9 Conclusions

Starting with analysing and extending previous projects I decided that a more general framework for robot behaviour and communication would be necessary to avoid spending most effort on the basic tasks every time a new project begins.
Research done on open source software used in this area brought some examples which were compared and two of them were selected to be combined to use in the future framework.
The robot's 3 year old operating system had to be upgraded to be able to support recent versions of the selected software and to ease future updates of the the OS a free distribution called Fedora was chosen.
The new software, PLAYER and JADE was installed on the robots, tested and configured. After it was considered suitable it was also installed on the other robots.
To demonstrate the effectiveness of the selected software combination a cooperative group behaviour was written. It is based on ideas of projects done by previous group but required a complete rewrite from scratch due to the different software used. The group behaviour is composed out of multiple smaller subbehaviours connected by more complex JADE behaviours to a finite state machine running autonomously on an own JADE instance on every robot in the group. Using the advanced interfaces of PLAYER these behaviours themselves are simple and easy to exchange. JADE provides flexible ways to connect those subbehaviours and provides simple means to communicate with other robots in the group, exchange data such as positions with them or make remote behaviour calls.
To develop these subbehaviours and the final complex behaviour the STAGE simulator was used to test the algorithms. When those algorithms would work reliably they could be tested without modifications on the robots which helped to speed up the development.
The final behaviour was then successfully tested in a mixed simulated/real world environment where both simulated and real robots follow an existing landmark moved by a user.

10 References

References

[1] AgeS: Filling a Gap between Single and Multiple Agent Systems, James Kramer and Matthias Scheutz, Dep. of Comp.Sci. and Engineering, University of Notre Dame, IN.

[2] L. E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. IEEE Transactions on Robotics and Automation, 14(2):220–240, 1998.

[3] ARIA is a Object Oriented interface to ActivMedia mobile robots.
http://www.activrobots.com/SOFTWARE/aria.html

[4] Reliable Mobile Robot Navigation From Unreliable Visual Cues
In Fourth International Workshop on Algorithmic Foundations of Robotics (WAFR 2000), Hanover, NH

[5] CMVision algorithm
http://www-2.cs.cmu.edu/ jbruce/cmvision/ ,2005

[6] Festival software
http://www.cstr.ed.ac.uk/projects/festival/download.html ,2005

[7] Java Agent DEvelopment framework http://jade.tilab.com/

[8] JADE Tutorial, JADE programming for beginners, Giovanni Caire, http://jade.cselt.it/doc/programmersguide.pdf

[9] Advanced uses of JADE http://jade.tilab.com/community-3rdpartysw.htm

[10] Java client for PLAYER http://java-player.sourceforge.net/

[11] The MAGE robot control software
http://robotics.swarthmore.edu/downloads.shtml

[12] Ct, C., Ltourneau, D., Michaud, F., Valin, J.-M., Brosseau, Y., Raevsky, C., Lemay, M., Tran, V., "Code Reusability Tools for Programming Mobile Robots", accepted for presentation to IROS2004. http://marie.sourceforge.net/

[13] Mobility Robot Integration Software Manual

[14] The Pattern Analysis and Machine Intelligence (PAMI) research group at the university of Waterloo
pami.uwaterloo.ca

[15] The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. Proceedings of the International Conference on Advanced Robotics (ICAR 2003),pages 317-323

[16] Stuart Russel,Peter Norvig. Artificial Intelligence - A Modern Approach.Pearson Education inc,Upper Saddle River,New Jersey,2003

[17] Andrea Schaerf, Yoav Shoham, and Moshe Tennenholtz. Adaptive load balancing: A study in multi-agent learning. Journal of Artificial Intelligence Research, 2:475-500, 1995.

[18] SFERES: Artificial evolution framework together with a multi-agent framework. http://sferes.lip6.fr/external/about.php ,2005

[19] Roland Siegwart, Illah R. Nourbakhsh, Introduction to Autonomous Mobile Robots, MIT Press, 2004

[20] Eiji Uchibe, Minoru Asada, and Koh Hosoda. Behavior coordination for a mobile robot using modular reinforcement learning. In Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS '96), pages 1329-1336, 1996.

[21] Homepage with material of this project http://www.eng.uwaterloo.ca/∼notippen/ ,2005

11 Appendix

All material will also be available on the CD which you should have received together with this copy of my report. If you need another copy of the CD please write an email to noleti@gmx.de

A Weekly summary

A.1 First week

The first week started with Mohammed showing me the now working implementation of the formation building by mazen and mackram on Monday. I also tried to get the necessary keys at the offices in EIT. The next days were spent in reading basic books on the topic and reading basic papers which were provided by professor Kamel. The books included the following:

- Mobile Robotics. A practical Introduction. Ulrich Nehmzow. Springer,Berlin, 2003
- Introduction to autonomous mobile robots. Roland Siegwart and Illah R. Nourbakhsh.
- Minimalist mobile robotics : a colony-style architecture for an artificial creature. Jonathan H. Connell.

I also collected the source code for the old projects by Mazen and Mackram and the other projects which were done before that. Until the end of the week I had the required keys to access the internal room with the robots. After I found out whom to ask about an account for the computers I requested an account.

A.2 Second week

In the beginning of the week I continued with reading the background informations from the books, in parallel I started to modify the existing code. One example is a small task to stop the robot on bumper contact which I wrote. I got the Windows network account to access the PCs in the lab.
Due to the specialised programming of the legacy programs I has difficulties in extending them to new tasks with reprogramming a considerable part. Those older programs were also mostly programmed for an older version of robotmgr so I had to do some bug fixing to even compile them to start with. Unfortunately my laptops's harddisk died the previous Friday, so I also had to spend considerable time on reinstalling the laptop's operating system, old backups and setting up a productive environment. Some written work done in the previous week was lost because I didn't make a backup of it.

A.3 Third week

During the week I set up printing with my Linux laptop (had trouble with printing from the Windows PCs).
Tue-Thu: Started to work on the construction of the framework for flexible behaviours with different triggers, including efforts to understand the old code by Mazen and mackram
Fri-Sun: Mostly research on existing open-source software suitable to use with our robots. This should considerably lower the effort to get a stable base for future developments in the PAMI lab. I also got CVS access to the robotmgr tree in this week.

Plans for the Behaviour framework so far:

- All robots run a server app, at the moment thats vidcap.cpp. This server starts basic stuff like message listening etc. This server also handles incoming messages and searches them for commands etc. The server will start behaviours modules depending on the received commands, and enables the behaviour modules to communicate to the other robots.
  The server will also allow other robots to access all data that this robot has gathered, such as position, angles of both robot and his PTZ unit and data on the landmark etc.

- The command.cpp will allow the user to enter commands at one robot. The command will then be spread to all robots by the command.cpp. Apart from one robot sunning command.cpp and spreading the messages, all robots are behaving totally equally.

- The behaviour modules implement different behaviours for the robots. These include the ported versions of the behaviours Mazen and mackram wrote, this time more cooperative. In the original version the code by Mazen and mackram just lets the two robots which don't run the command.cpp report their position, with which the "master robot" then computes their position in the formation and sends it to them.
  Some modules that are planned are:

  - Line - The "line behaviour" from Mazen and Mackram ported to fit the framework, more cooperative
  - triangle - The "triangle behaviour" from Mazen and Mackram ported to fit the framework
  - vidcap - The "vidcap behaviour" from Ben ported to fit the framework. The robot won't follow the landmark unless it is the selected "leader" in the group (local variable of the robot object). But it will try to find and follow it with the PTZ cam unit. Once it either found the landmark or finished the 360 degree scan it will send a message to the other robots to report its success or failure in finding the pattern. It will then wait for the other messages to arrive and then determine the leader, every robot does this

on its own. The robot selected as leader will then send a message to the others telling them the formation they should take (could be based on a barcode in the landmark).
Optional, might be implemented later:
The other robots continue to look for the landmark. They continue to send messages if they see it, if they are now closer than the leader the leadership changes to the closest robot.

- line_follow - The behaviour to follow the leader in a line formation. The two robots communicate their positions to each other and then look for the global best solution for their position, to which they move then.
- triangle_follow - The behaviour to follow the leader in a triangle formation

A.4 Fourth week

After discovering PLAYER/STAGE last week I compiled it, which took some time because of some smaller compilation issues. I looked for example applications of PLAYER/STAGE and found one at the project MARIE. First tries with PLAYER/STAGE on my laptop were working fine and looked promising. The player project still lacks a control and communication structure for group behaviour, so I continued to look for other software to take over those parts.
On my search for open source software to use on the robots I found other software as well, so I made a short summary of those programs in a separate paper, although I didn't find a software to take over the group communication part.
I tried to compile the PLAYER server locally in my home directory on one of the magellans (mag2) but the software (e.g. GCC) on the robots is far too old. I discussed with Ben to install a newer version of Linux on the robots and spend some time looking for a decent possibility to get the robot to boot from an installation medium, but found none so far. As the robots don't have a floppy, CD-ROM or other flexible installation drive we might need to use the /home partition (which has 22GB) as the partition for the installation. Another possibility would be to exchange the whole harddisk against another one/connect the harddisk in the robot to a normal PC to build the new linux on this one. The install procedure is described in the mobility handbook in the appendix A-2. The information needed to connect a serial console to the robot's PC is given in the robot's handbook (9600 8N1).

A.5 Fifth week

This week began with working on the reimplementation of the framework. The basis was again the second project from Mazen Slim and Mackram Raydan, which was cleared of any code not fitting to the concept of the framework. At the end the basic message sending and receiving and the robot class were the only code fragment kept for the new project. Then the first version of the framework was written, it had some simplifications at some points to allow a faster testing. The message passing between the robots from the old code turned out to be a bit more complicated

than expected and took the whole Tuesday without leading to a working product. The problem was the blocking nature of the sockets used for message passing even between threads or programs on the same robot. While listening to those sockets with the server no other second program would be able to use these.
I then reconsidered to use the message passing algorithms implemented in robotmgr on Wednesday, again without luck due to a non-existant documentation or working examples. In theory robots can join a message channel where commands and other messages are broadcasted to all participating robots, it is also possible to read sensor data from and give commands to remote robots running the robotmgr.
Most of the Thursday was spent documenting progress so far and finishing the material written so far. Some attempts were made to trace back the segfaults caused in the robotmgr by my simple testprogram, but none were successful.
On Friday I continued with writing documentation, I also collected more informations about the player/stage project. On the weekly meeting professor Alaa Khamis agreed to let me install a newer operating system on one of the robots, min1 (a Mini ATRV) was chosen because no-one uses it at the moment. I had problems with getting the on-board PC to start or getting a serial connection at the beginning, also all the on-board batteries seem to be dead by now.

A.6 Sixth week

I started with installing Fedora Core 1 on the min1 ATRV mini because as it seems this is the last distribution of Fedora which supports booting from a floppy disk. After some trouble with the network configuration and a failed attempt to simply upgrade the existing RedHat 6.2 version I finally managed to get the installation process starting. I decided to install only a minimal system without too many programs because most of them would be replaced with upgrades during the following upgrade from Core 1 to Core 3. The upgrade process itself was relatively painless, but took quite a lot of time, resulting in a fully operational Fedora Core 3 installation on Tuesday. While I tried to get robot specific configuration to work (shutdown via the rFlex interface for example) I also finalized the documentation written so far to hand them over to professor Kamel on Thursday after I met him on Wednesday. On Friday I started to compile both player and robotmgr on the robot which worked out OK. During the meeting with the research group I showed another small demonstration of stage on my laptop. Most of the weekend was spent reading documentation on Player/Stage/Gazebo.

A.7 Seventh week

After reading the documentation on Player/Stage/Gazebo on the weekend I started with getting the basic remote control of the robots via player to work on Monday. For this some configuration files had to written which had to include the exact position and orientation of the sonar sensors of the ATRV mini for example. As I had no exact ruler at hand at that time I approximated these

values for a start, the angles should be correct though. The whole process took some time because I found no documentation about how to write those configuration files apart from working examples given on the webpages. I also modified the stage configuration files to use a model of the ATRV mini in the simulation. I tried one of the example control codes called sonarobstactleavoid on the real robot in a court set up in the lab and it worked.
Tuesday I spent quite some time to get the camera to work, some issues with the video4linux2 driver and most importantly a probably defective black and white camera delayed this process to take a whole day.
Wednesday I took one of the Sony PTZ cameras from mag3 to try the PTZ control in Player. The control works now, but there seems to be a small problem with the serial connection from the camera to ttyS2 - sometimes it is necessary to replug the connection because it won't work otherwise.
Most of the Thursday was spent writing a small introduction to Player/Stage/Gazebo and capturing short movies to use during the presentation. The work on the presentation continued until Friday, when the presentation was held.
As final element of the robot which needed testing I tried the speech synthesys with Festival, supported by Player. This would test both the physical audio output hardware which was, to my knowledge, never used before, as also the ALSA Linux sound driver and the Festival program, of course. After finding and connecting a set of speakers to the right port in the robot, speech output worked right away, so I could assume the Linux install on min1 to be fully successfull.
The only remaining problem is more a minor annoyance: When shutting down the robot via the rFlex interface the rFlex controller waits for the words:"The system is halted" on the console and then kills the power to the system. It seems that with the software installed on min1 these words changed to "System halted", which prevents the rFLex device from shutting down power from itself.

A.8 Eighth week

After a presentation of Bram about his project with the Magellans and a following discussion I decided to have another look at using CORBA for Inter robot communication. I read before that one of the earlier groups with Fan,Wong and Yuen planned to use CORBA but then dropped this plan because it was too complicated for them - even though they were a group of 3 and they had 6 months of time during their project. This was the reason I ignored Corba at the beginning. Bram used the JADE Java Agent DEvelopment Framework which indeed looks quite powerfull, for example it already provides almost everything I mentioned as design ideas for the framework I proposed in my initial project proposal. The downside is that it uses Java in which I have mostly only theoretical knowledge. So I spent the weekend and Monday on reading on Jade and Corba in general, also looking for a possible C++ equivalent.
On Tuesday I started looking at the code Bram wrote for his project. He used robotmgr, of course - so one possibility for work of me could be porting the existing code to use Player as software to

control the robots, thus enabling the use of robot with different hardware than the rFlex interface which robotmgr uses to perform in Bram's experiment. Unfortunately I still couldn't find any version of Bram's soon-to-be=released paper - so the whole project was a bit hard to understand by just looking at the sourcecode.

A.9 Ninth week

After the basic setup of Jade was done in the previous week I started with extending the PingAgent example behaviour because it already included two features I wanted to have in my planned PlayerAgent, the registration at the DF and the handling of messages. I had to extend the handling in the course of the week heavily to support features such as selective handling of messages which fit to a template pattern, blocking of this Behaviour if no message is received and of course the implementation of new keywords to be detected in the messages which the Agent receives. Most of the time i used the DummyAgent in Jade to send messages to the Agent and test the handling. I also started using the JavaClient for Player which was mostly straight forward. Unlike the c++ player client the Java client seems to have to update the sensor data manually by calling readAll() on the player proxy object, which I implemented in a TickerBehaviour in Jade which is called every 100ms. This ticker behaviour also checks if a move command is currently active and issues move orders to the robot if this is the case. I then started to implement simple behaviours like move or stop which resemble the behaviours in robotMgr. After these were working to my satisfaction I implemented a goto behaviour which allows to specify a point in the coordinate system of the robot to which it will then try to move. The algorithm used is a quite straight forward approach I thought up: The robot calculates the angle to the optimal line connecting both points, tries to turn to that angle and then moves toward the target point until its angle to the target point get too big again, in which case it will rotate a bit and go again. This works pretty good in the simulation. On Thursday we held a presentation for grad students who had a tour in the PAMI lab.
Further tests were made with the blobfinder proxy of the javaclient. Basically the blobfinder-works which means that the robots can detect certain user defined ranges of colors, but there seem to be bugs in the implementation of either the javaclient or the player server, because the player.readAll() blocks and blocks the whole program if no blobs are insight of the robot. This way the robot effectively stops until a blob moves into sight by itself, at least in the simulator. I filed some requests at both the player/stage and the javaplayer mailinglist but got no reply so far.

A.10 Tenth week

After having finished the goto behaviour last week this week I started with reading about more advanced features of Jade such as the FSM behaviours for example which allows to arrange other behaviours to a complex finite state machine quite easily. I then designed a FSM to enable the

robots to try to detect a blob, send messages to all other active agents about it and collect their replies. To keep track of other agents in the network I used a central name service to which every client connects upon it's start. The agents register to that service upon startup and query it every 10 seconds or so to update their internal database with this data. When the agent wants to send a message to every other agent he calls the SendToAll behaviour which uses the database of the known clients and sends the message to everyone. Another behaviour called WaitForAllMsg waits for a certain message to come from every known robot. Once both behaviours have finished successfully the FSM will activate the next behaviour, in which the instances in the received messages are compared and the closest robot is selected. This is done on every robot individually, but since every robot has the same data this will lead to the same results for everyone. The closest robot is made the leader, every other robot will start the Follow behaviour while the leader starts the LEAD behaviour. These behaviours are again started by the FSM according to the return value of the CopmuteClosest behaviour, which will return '0' in case the robot will follow or '1' if its the leader. To manipulate the return value of behaviours the onEnd() method has to be overridden, the return value of this function is the value used by the FSM.

A.11 Eleventh week

The eleventh week started with continued work on the FSM behaviour. The problem with the blobfinder giving false values in the simulator was finally not resolved but I found a workaround - because falsely reported blob have always the same size they can be sorted out easily. Another problem is that the blobfinder in stage will report any colored blobs, so I have to sort out the unwanted red blobs from the robots as well.

A.12 Twelfth week

With the FSM behaviour working I started to finish the report so not too much exiting can be said about this week. I met Prof. Kamel and we agreed on me having a presentation on Friday the 9th - so I will have to prepare that one too. I plan to give a live presentation of the behaviour with both classes of robots and also the simulated ones - that's going to be interesting! I fixed the design of this report using templates provided on the waterloo website. It uses font size 11, line spacing of 1.2 and custom borders which all looks perfect to me. Additionally I decided to use the fancy style to get the page headers which I find quite helpful. The graphics for the report are almost done, I'm still undecided if I should prefer black and white or color ones... will have to decide on that soon.

A.13 Thirteenth week

With only 3 remaining days this week was quite short... I finished the report by typing this and are going to proofread it later.

B User guide to PLAYER/JADE robotic architecture

I wrote a user's guide to my project with practical hints on running and operating the project's demonstration. It is handed in separate print for greater convenience while using it. It is also available on the CD as a PDF file called Userguide.pdf.

C PLAYER/Stage configuration files

This is the configuration file of the ATRV mini called "rflex.atrv". The configuration file for the Magellan pro ("rflex.magellan") has additional sections for the IR devices. Both configuration files are also on the CD.

```
driver
(
  name "camerav4l"
  provides ["camera:0"]
  size [320 240]
  source 1
  save 0 #this would create screendumps - and generate big amounts of data fast!
  norm "ntsc"
)
driver
(
  name "cameracompress"
  provides ["camera:1"]
  requires ["camera:0"]  # Compress data from device camera:0
  save 0 #this would create screendumps - and generate big amounts of datra fast!
)
driver
(
  name "cmvision"
  provides ["blobfinder:0"]
  requires ["camera:0"]
  colorfile "color.cfg"
)
# This stuff doesn't really work so far but is interesting
#driver
#(
#  name "festival"
#  provides ["speech:0"]
#)

#driver
#(
#  name "sphinx2"
#  provides ["speech_recognition:0"]
```

```
#)

driver
(
  name "sonyevid30"
  provides ["ptz:0"]
  port "/dev/ttyS2"
  fov [3 30]
)

driver
(
  name "rflex"
  provides ["position:0"  "sonar::sonar:0" "sonar2::sonar:1" "power:0" ]

  rflex_serial_port  "/dev/ttyS1"
  mm_length  500.0
  mm_width  300.0
  odo_distance_conversion  190
  odo_angle_conversion  60000
  default_trans_acceleration  50.0
  default_rot_acceleration  1.0
  rflex_joystick 1
  rflex_joy_pos_ratio 6
  rflex_joy_ang_ratio -0.01
  range_distance_conversion  1.476
  sonar_age  1
  sonar_echo_delay 30000
  sonar_ping_delay 0
  sonar_set_delay 0
  max_num_sonars  224
  num_sonars  16
  num_sonar_banks  2
  num_sonars_possible_per_bank 16
  num_sonars_in_bank  [8 8]
  # theta (rads), x, y (mm) in robot coordinates (x is forward)
  mmrad_sonar_poses  [
0.00000  200.00000    100.00000
0.31416  180.00000    110.00000
0.62832  160.00000   130.00000
```

```
0.94248  140.00000   140.00000
1.25664   120.00000   150.00000
1.57079  100.00000   150.00000
1.57079   -200.00000   150.00000
3.14159   -200.00000   150.00000
0.00000  200.00000    -100.00000
-0.31416  180.00000    -110.00000
-0.62832  160.00000   -130.00000
-0.94248  140.00000   -140.00000
-1.25664   120.00000   -150.00000
-1.57079  100.00000   -150.00000
-1.57079   -200.00000   -150.00000
-3.14159   -200.00000   -150.00000]
  sonar_2nd_bank_start 8

  rflex_power_offset 12
)
```

D Jade PlayerAgent Code

The main source code of this project is in the PlayerAgent.java file on the CD in the source code directory. In total it is about 1000 lines long - 25 pages, that's why I didn't include it in print here.