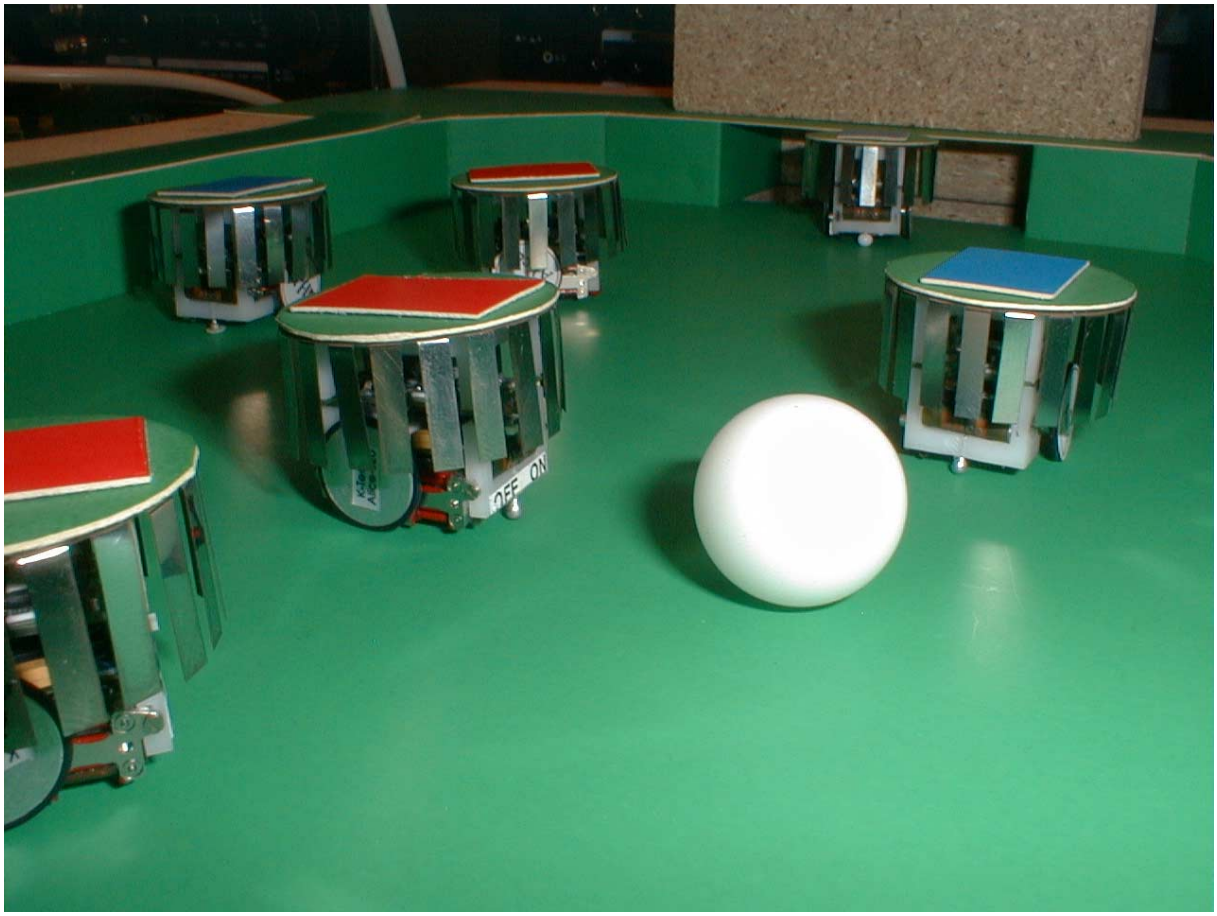
MICROENGINEERING DEPARTMENT
AUTONOMOUS SYSTEMS LAB (ASL)
ASS. PATRICK BALMER • ASS. GILLES CAPRARI
PROF. ROLAND SIEGWART

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

PATRICK RAMER • JEAN-CHRISTOPHE ZUFFEREY

7TH SEMESTER, MICROENGINEERING

SOCCER ALICE



LAUSANNE, FEBRUARY 2000

CONTENTS

1 INTRODUCTION 3

1.1 PLAYING SOCCER WITH ALICE 3
1.2 CONTROLLING THE ROBOTS 3
1.3 PROGRAMMING ENVIRONMENT 4
1.4 HOW TO READ THIS DOCUMENT 4

2 THE COMPONENTS 5

2.1 ALICE: THE ROBOT 5
2.2 THE BALL 5
2.3 FIELD & SETUP 6
2.4 IMAGE PROCESSING SYSTEM AND CAMERA 7
2.5 COMPUTER AND OPERATING SYSTEM 9

3 THE ROBOTS 11

3.1 MECHANICS 11
3.1.1 Construction 11
3.1.2 Speeds 11
3.1.3 Metal sheets 12
3.2 CONTROL OF THE ROBOTS 12
3.2.1 Global structure of the control system 12
3.2.2 IR Communication 13
3.2.3 Communication protocol 14
3.2.4 Motion control of the robots 15

4 THE COMPUTER RESOURCES 20

4.1 ORGANIZATION OF COMPUTER RESOURCES 20
4.2 COLOR OBJECT POSITION SERVER (COPS) 21
4.2.1 What is COPS? 21
4.2.2 TCP/IP connection with COPS 22
4.2.3 Integration of USB video camera 24
4.2.4 Camera settings and color recognition 24
4.2.5 Object tracking 26
4.2.6 Initial positions 27
4.3 THE SERVER PROGRAM: ALICE SOCCER MANAGER 28
4.3.1 General considerations 28
4.3.2 Alice thread 28
4.3.3 COPS thread 29
4.3.4 User threads 29
4.4 USER PROGRAMS 32
4.4.1 Connection with ASM 32
4.4.2 An example of soccer strategy 33
4.5 OVERVIEW: PROGRAMS AND CLASSES 34

5 PERFORMANCES 35

5.1 TIME BALANCE 35
5.2 UNSOLVED PROBLEMS 36

6 CONCLUSION 38
6.1 ALICE IS READY TO PLAY SOCCER 38
6.2 FUTURE DEVELOPMENTS 38
6.3 WHAT WE HAVE LEARNT 39
6.4 THANKS 39

7 BIBLIOGRAPHY 40

8 APPENDIX 41
8.1 GLOSSARY & ABBREVIATIONS 41
8.2 RELATED WEB SITES 41
8.3 ALICE 42
8.3.1 Speeds of Alice 42
8.3.2 Data sheet 43
8.4 DIAGRAM OF ORGANIZATION OF COMPUTER RESOURCES 44
8.5 ASSEMBLER CODE OF ADAPTED COMMUNICATION PROTOCOL 45
8.5.1 Alice 45
8.5.2 Remote control 56
8.6 DELPHI CODE OF ADAPTATION OF COPS 63
8.6.1 GUSB.pas (Class for USB video camera grabber) 63
8.6.2 Tracking module 71
8.7 C++ CODE OF ASM 75
8.7.1 AliceCom file 75
8.7.2 Alice thread 87
8.7.3 COPS thread 89
8.7.4 User thread 94
8.7.5 Main 99
8.8 C++ CODE OF USER PROGRAM 106
8.9 USB CAMERA COMPARISON 110
8.10 DISK CONTAINING ALL PROGRAMS 112

1 INTRODUCTION

1.1 Playing soccer with Alice

Soccer is certainly one of the widest spread sports in the world. But it is not just a sport, it is a game as well. And it is very simple. Soccer can be played nearly everywhere. A goal is easily made with two objects of any kind, and an old beverage tin can serve as ball. If a Soccer World Cup takes place every four years, this is an event throughout the world, and teams from any country you could imagine, take part.

In the last decade, soccer has been taken to another dimension. It is now being played by robots, and there are even World Cups organized for robot soccer. But it is still the people who are passionate to make the robots move and score. These robots have dimensions of about 10 to 20 cm. Technology has been put together with a very simple game. The same rules now serve to play the game on a different level. **Programming has become the tool**, instead of physical effort. But as in the initial game, **it will be the clever player, who scores the goal**.

We wanted to go a step further and miniaturize the game once more, with even smaller robots. First of all, this is a technological challenge. But even more important, this will allow to **combine soccer with another important domain: education**. With this new system it will be possible to **play soccer on a field, the size of an A4 sheet**. The lightness of the robots and the whole system is a big advantage. Thus, a **handy robot soccer** kit can be offered to schools to teach the pupils in programming.

The used robot is called **Alice** and has been developed at the EPFL. To fit the size of the field, there will be three robots per team, that is totally 6 robots on a A4 sheet.

The combination of a simple game, technology and education is the main achievement of this project. Therefore it is very suitable to be presented at exhibitions. It is planned to suggest this robot soccer to expo.02, the Swiss National Exhibition taking place in 2002. A robot soccer championship will be organized, where different school classes from the whole country will be playing against each other with their self programmed robots.

1.2 Controlling the robots

The aim of this project is to offer to pupils and students a system which allows them to control the robots and to make them play soccer. Hence, the question, how to control the robots. It is intended to offer **three different levels of difficulty** to the user.

1. **Speed control mode**: in this mode, the user has direct control of the speeds of the two wheels of the robots. This is the highest level, and it demands a certain knowledge about the motion control of a wheeled robot.
2. **Position control mode**: the user gives a target position, which the robot has to achieve. The motion control is done by the provided program.
3. **Behavior mode**: a certain number of implemented behaviors are accessible in libraries.

The first two modes have been implemented. The behavior mode is not implemented, but the program structure allows to add this mode at some time later.

The idea is, that the user can program a strategy with the tools above. **When the game is running, the user program, or Team Manager as we call it, will be completely autonomous.** There is absolutely no real time intervention with joysticks intended. It is the user program, which intervenes in real time. The stress is on the strategy programming before the game.

It is possible for the user to switch between the different modes described above at any time. This allows an even more flexible programming of the soccer strategy.

1.3 Programming environment

To implement this system, we basically have to work on two levels. First on the robot Alice to establish the communication with the PC, and second on the PC to manage the different tasks which have to be executed.

Alice is locally controlled by a PIC microcontroller, which is directly programmed in assembler programming language. The manufacturer of the PIC, Microchip Technology Inc., also provides an assembler programming environment called MPLAB, which we use.

As you will see later, there are three different programs running on a PC. All of these programs are implemented in an object oriented programming language; one of them in Delphi, the other two in C++. We use 'Borland Delphi 3' and 'Borland C++ Builder 4' for the development. We need a Client/Server version of both environments in order to establish the communication between the different programs.

1.4 How to read this document

In the first place, this document is the report of a semester project at the Autonomous Systems Lab (ASL), EPFL. However we were engaged by the ASL last summer, where we could do some important preliminary work for this project. We will therefore inevitably describe things which have been done last summer and are not part of the semester project actually. In the document we will not make a difference between semester project and summer work. However we shortly mention here which paragraphs partly contain information of the preliminary work. Completely general considerations made in the text are not mentioned here.

2 The components

3.1.3 Metal sheets

3.2.4 Motion control of the robots

A first implementation of motion control was done last summer. However most considerations in that paragraph have been made during the semester project.

4.2.2 TCP/IP connection with COPS

4.2.3 Integration of USB video camera

2 THE COMPONENTS

2.1 Alice: the robot

The micro robot Alice is a very small autonomous robot (21 mm x 20 mm x 19 mm). It is controlled by a PIC16F84 microcontroller, whose Flash EEPROM can be reprogrammed. Watch motors (Lavet) drive its wheels. Three watch batteries serve as energy source. Thus, the robot keeps its autonomy for about 10 hours.

The most recent version of Alice has got bigger wheels which allows **stable velocities up to 51 mm/s**.

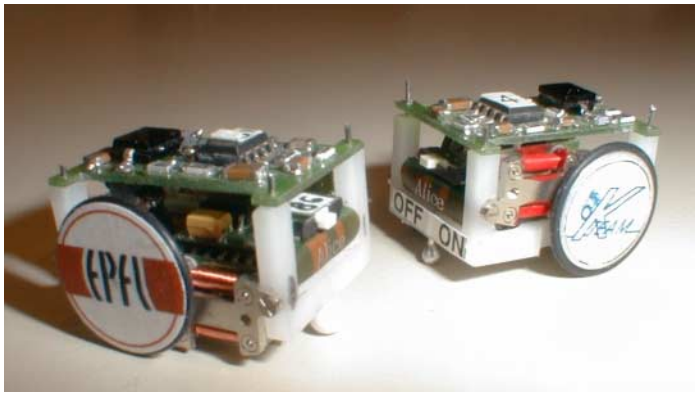


Fig 2.1 Mini robot Alice

To communicate with the robot, there are two possibilities: infrared (IR) communication (unidirectional) and radio transmission (bi-directional). For reasons which will be explained in § 3.2 we don't need to get any information from the robot itself, and we can therefore use the **IR communication**. This communication is much cheaper, simpler to handle and less power consuming.

Since the idea is to play soccer on an A4 sheet, there will be **6 Alices, 3 per team**. Each robot has a unique identification number.

Actually, the robot Alice is one of the central elements of the system and determines many of the system's specifications. This is why we dedicate an entire chapter to Alice. You can find all further details about the robot in chapter 3.

2.2 The ball

Several considerations have to be made to choose an appropriate ball. The color, the weight and the form all influence this choice. Especially the form and weight become important because of the miniaturization of the soccer game.

The surface has to be **as smooth and as round as possible** to assure a straight movement when only pushed with little force. Since the robot Alice is very light and not so fast in movement, this is very important. The weight should not be too big to allow Alice to push the ball. On the other hand it should not be to light to avoid a chaotic behavior due to surface irregularities.

The color must be entirely determined according to the demands of the vision system (camera and COPS, see § 2.4, 4.2.4). We can mention here that it should be rather dull to avoid light reflections.

The material does not have much importance as long as above criterions are respected.

For the size, a **diameter of about 20 mm** seems to match the system's proportionality.

Our choice so far, is a ball made of Polypropylene (PP) which is normally used in valves:

Fig 2.2 The ball with a player

This ball is provided by Maagtechnic AG, Dübendorf, Switzerland.

2.3 Field & setup

At the beginning, we deliberately decided to make Alice play soccer on an A4 sheet. On one hand, this is a good way to make advertisement, for example for expo.02. On the other hand, this is large enough for 6 Alice robots and small enough for a portable system. Actually, since the final purpose of the project is to **put the whole system in a briefcase**, we had better choose an adapted size of the field.

In fact, the current size of the field is 300x210mm with broken corners. The goals are simple holes on both sides:

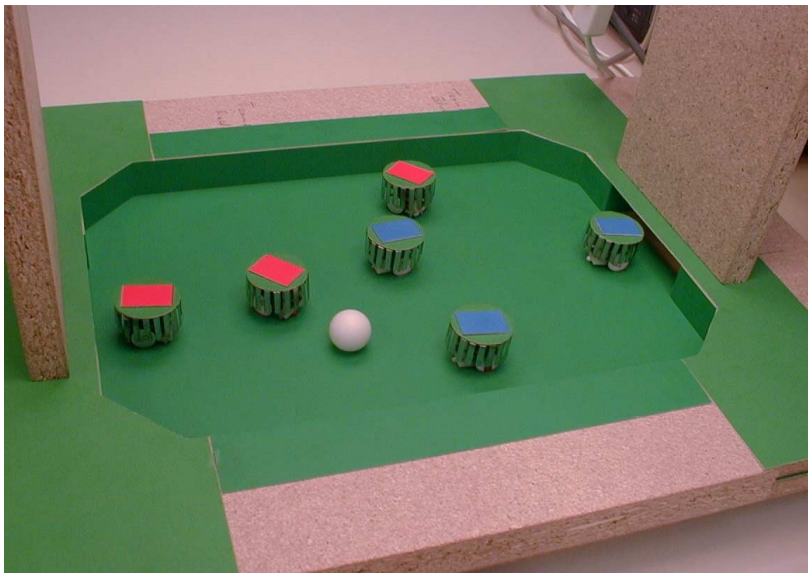


Fig 2.3 The current trial field

This construction is mounted on a board that can be adjusted with 4 screws in order to obtain a **perfectly flat surface**. Otherwise, the ball will always be attracted to a certain side of the field.

For the floor coverings, a **green cardboard** has been chosen. First, this color is the closest to the real soccer fields. Second, we have to pay attention to the image processing system. As we will see later, there are several difficulties to make an image processing system running well. For example, we can not choose a reflective surface, otherwise the camera will be dazzled. This way, the cardboard is a good choice. The **importance of the field's smoothness** made us give up other solutions like painting or colored paper.

A support for the vision system is placed over of the field: the video camera and the lighting (see § 2.4). Our temporary solution is made of planks with several steps like a shelf in order to have the possibility to change the height of the camera and the lighting (currently, the camera is on the fourth step from the bottom):

Fig 2.4 The current trial setup

2.4 Image processing system and camera

Getting the precise positions of robots on a plane is a quite difficult task. Since we had the possibility to use an existing system, we caught the opportunity. This image processing system **based on color recognition** is called COPS for Color Object Position Server. For more detailed information about COPS, please refer to § 4.2.

The basic principle of this image processing program is to put together pixels of about the same color and by doing that, extract blobs or objects of each set. Then, there is a filter that throws away all blobs that are out of a defined size. Finally, **the program calculates the inertia axis of the objects**. This is why we use color rectangles on the robots. So COPS can give us the **positions** and the **directions** of the players and the ball:

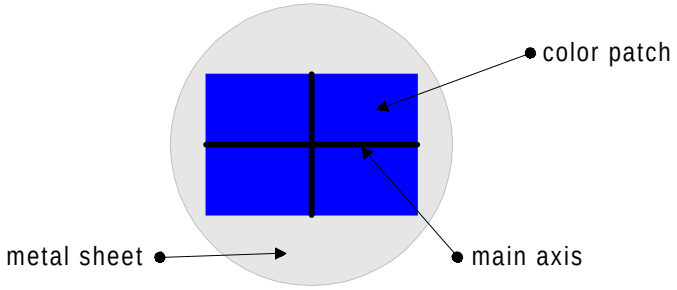


Fig 2.5 Color patch for recognition

For the ball, the given direction will be devoid of meaning, this is why we will not use it.

Since we wanted to build a cheap product, we chose to adapt COPS for a **USB video camera** (see § 4.2.3). After several trials, we chose the QuickCam Home by Logitech (see [i] in appendix 8.2):

Fig 2.6 QuickCam Home

With this camera, which costs about 100$, we can get a pretty good image of 320x240 pixels at 15FPS. Moreover, it has very **few lens distortion** in comparison with other QuickCams. Refer to appendix 8.9 for a USB video camera comparison.

The choice of the colors (field, robots and ball) is based on the behavior of this camera. And we see that the following colors are pretty good for both camera settings and COPS settings:

- Field: green
- Ball: white
- Robots of team 1: blue

- Robots of team 2: red

The **lighting** is a very critical point in a vision system. We have to provide a **homogeneous and intense light** on the field in order to avoid shade zones and bad color rendering. Eventually, a solution with an **indirect lighting** provided by four 60-Watt bulbs was chosen (see Fig 2.4).

2.5 Computer and operating system

First of all, we need MS Windows 98 for our system, because COPS has been written on Windows and the USB camera (QuickCam Home) runs only on Win98. Other operating systems like Windows NT or Linux are not possible because they do not support USB at the moment.

Thus our main server, called ASM for Alice Soccer Manager, has to run on a Windows 98 platform. But **we do not want to impose this choice to all users**. Moreover, our target public use, most of the time, Macintosh because a large part of Swiss schools are equipped with that kind of computers. This is why we opted for a **TCP/IP communication** between ASM server and the users. Please refer to § 4.1 for further information about the organization of computer resources in this project.

The possibilities for the final user interface are quite unlimited. The students will be able to implement their team manager program in whatever existing environment such as MS Visual Basic (even Visual Basic Application), MS Visual C++, Borland Delphi, Borland C++ Builder, and so on, and other developing environments on other operating systems. The only thing needed is access to a socket, in order to implement the client side of a TCP/IP communication. To do that, we have, for example, components that implement a TCP/IP client. For further precisions on the subject, see § 4.4.1.

Finally, we will have a **great flexibility** in the global organization of computer resources. This means we can run a soccer game in different configurations. Of course, **the computer on which COPS and ASM are running will be provided in the set with the briefcase**, in order to be sure to have a good computer with at least Windows 98, an Ethernet card (for TCP/IP communication with users), a serial port (for the remote control for the robots) and a USB port (for the video camera).

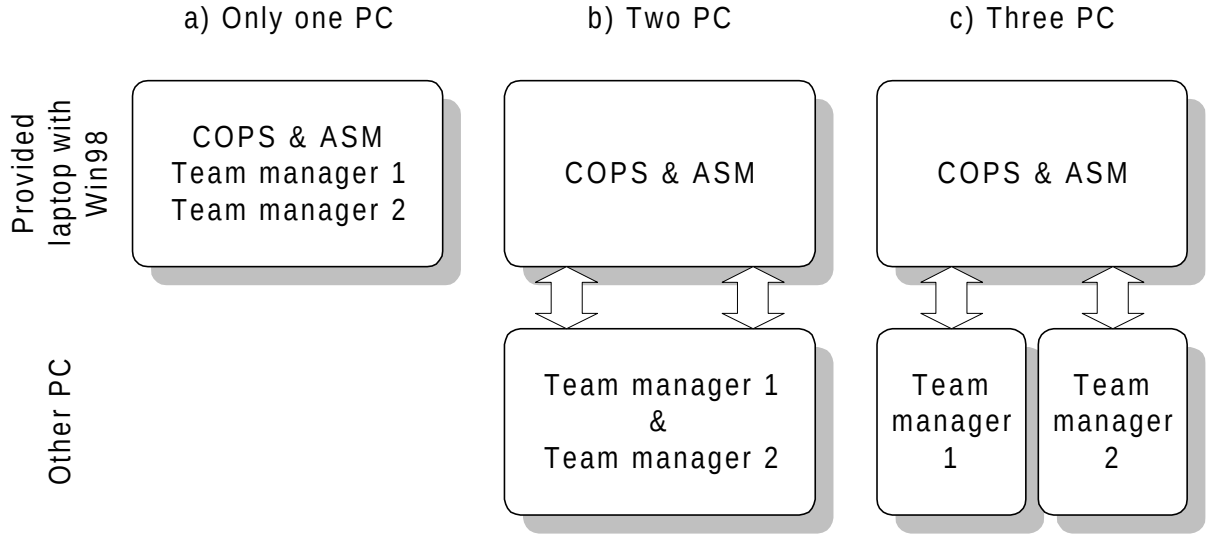


Fig 2.7

a) **Only one PC**: all programs (COPS & ASM, Team manager 1, Team manager 2) run on the same PC (TCP/IP communication on a local host is possible). A drawback of this configuration is the necessity for the team managers to be written on Windows 98.

b) **Two PCs**: COPS & ASM will run on the provided laptop. Both team managers will be installed on a second PC, for example a Macintosh. Note that we always have two TCP/IP channels between users and ASM, one for each team.

c) **Tree PCs**: COPS & ASM will still run on the provided laptop. Then, each team has its own PC on which runs the corresponding team manager.

Note that a simple interface for younger students, where the TCP/IP stuff would be hidden, could easily be implemented on top of the system.

3 THE ROBOTS

3.1 Mechanics

3.1.1 Construction

Alice is a differential drive robot with two wheels.

For this project we have obtained the most recent version, which has bigger wheels that are now placed on the same axis. Further, the resistor of the power supply has been diminished to allow a higher supply voltage for the motors. Thus, a maximal velocity of 51 mm/s can be reached.

The robot props itself on the ground with its two wheels and a third point which is fixed at the front. **A fourth point to prop** had to be fixed at the back of the robot to avoid that the robot leaves the ground at the front when pushing the ball or heading into another robot. For the same reason, some **additional weight** is fixed at the front.

3.1.2 Speeds

Two watch step motors drive the wheels of Alice. There is one default speed, which is set to 51 mm/s. But in order to make possible a stable and smooth motion control, several speeds must be available on Alice. This is possible with an upper and lower limit, of course. Likewise, there is **no continuous velocity available**, since we work with a discrete microcontroller system with very little resources.

Velocity can be varied by playing on two parameters: the phase period of the motors (M_PhPeriod) and the pause between two entire motor cycles (M_Pause). One cycle has 6 phases and the pause in-between is called the passive phase 0. The phase period and the pause are both multiples of the interrupt cycle (143 μs). M_Period and M_Pause indicate the number of multiples of 143 μs. M_PhPeriod can take reasonable values from 3 (429 μs) to about 10 (1430 μs). The higher it is, the slower the motor turns. Below a value of 5 the motors are not stable, above a value of 10 the energy consumption becomes too high compared to the velocity. M_Pause takes reasonable values from 0 to about 15. Higher values make the motors stop.

The turning sense of the motors (M_Dir) can also be defined to allow forward and backward movements.

To keep the speed information as short as possible, **we must choose a strict minimum of speeds to use**. The speeds currently used on Alice are the following:

Speed	PhPeriod	Pause	Dir	Velocity [mm/s]
3	5	0	1	51
2	7	0	1	37
1	7	11	1	29
0	-	-	-	0
-1	7	11	0	-29
-2	7	0	0	-37
-3	5	0	0	-51

These speeds can be coded in 3 bits per wheel.

See appendix 8.2 for a complete table of possible speeds.

As there is no continuous velocity available to control the robots, but only a little number of discrete speeds, this has consequences on the implementation of the motion control. Please refer to § 3.2.4 for further details.

3.1.3 Metal sheets

Since Alice is supposed to play soccer, it should be possible to shoot the ball. But Alice itself is too light and too slow to transmit enough impulse to the ball. Hence a system had to be found to shoot the ball.

To achieve this characteristic, very thin steel sheet (50 μm) are mounted on Alice all around the circumference. **These sheets create a spring effect and let the ball bounce back**. They also protect the sensitive parts of Alice (wheel axis, gearing) from shocks with the ball or other Alices.

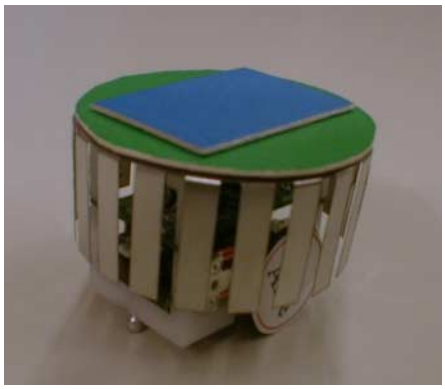


Fig 3.1 Metal sheet on Alice

On the other hand these sheets make the robots get stuck much more easily with other robots. This is a problem which should either be solved with a more appropriate construction of these sheets or with conditions and procedures in the software to take the robots apart when they get stuck.

3.2 Control of the robots

3.2.1 Global structure of the control system

The robots are entirely controlled by the PC. The connection between the PC and the robots is established by the remote control which exists along the robot Alice. This remote control is connected to the serial port of the PC, and the PC communicates via the standard RS232 protocol with the remote control. Hence, only entire bytes can be sent to the remote control. A PIC processor on the remote control manages the communication with the PC and then transmits the bits via an infra-red transmission to the robots according to a well-defined protocol.

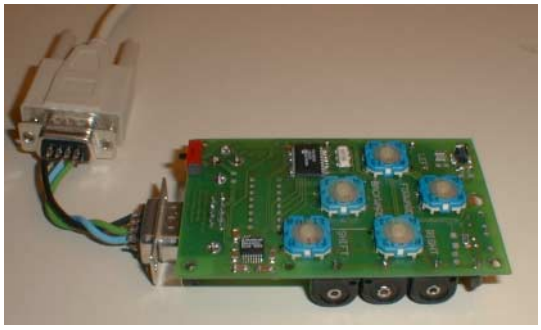


Fig 3.2 The remote control

The most direct feedback loop can be described as this: a camera takes an image of the situation of the robots; image processing is done on the PC and the information to be sent to Alice is calculated; this information is sent to the remote control, which transmits it further to the robots.

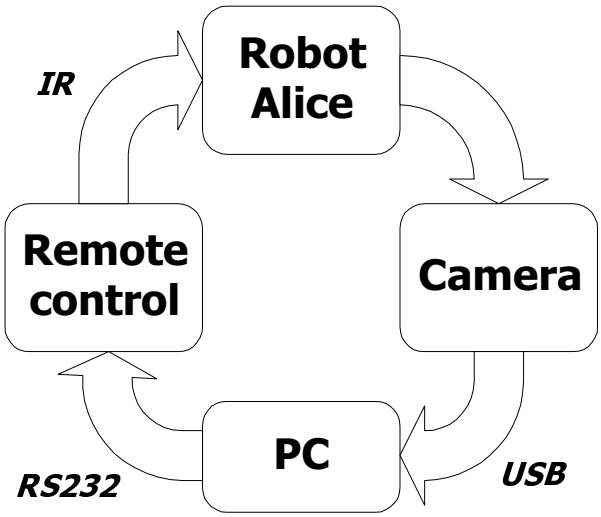


Fig 3.3 First feedback loop

There are no proximity sensors on the robots. Therefore communication between robots is not possible. **They are not autonomous and no local intelligence is implemented.** But since they are controlled by the PC, there is no need for communication between robots or local intelligence. This choice was made, because the IR transmission is too slow (2ms/bit). As only information, the robots receive the speeds for the two wheels. Any other calculations and decisions are made on the PC.

3.2.2 IR Communication

To communicate between the PC and the robots, an infra-red transmission is used. This transmission is **unidirectional**. **The speed is 2ms per bit, which is rather slow.** This is why we would like to keep the information to transmit to the robots as short as possible.

If one would like to have more intelligence on the robots, much more information would have to be sent to every robot. They would have to know the positions of the other robots and of the ball. The time to send all this information is much too long.

The infra-red transmission uses only a single frequency. **Thus, there are not several channels which can be assigned to the different robots; addressing is necessary.** But since the address should be included in the information bits transmitted, this will blow up the number of bits again. Hence, a new concept had to be found. The newly implemented protocol is described in the next paragraph.

IR transmission is disturbed if there is direct light incidence from bulbs or neon lights where transmission takes place. The steel sheets placed upon Alice contribute to avoid this sort of disturbance. To keep working the vision system, direct light incidence has to be avoided anyway. The field is lit up only by reflected diffuse light rays.

Likewise, the remote control should not be placed too close to the robots. This would prevent any proper transmission. As for the upper limit of the distance between remote control and robot, there is no problem for our soccer application. As long as a direct connection without obstacles is assured, everything will work fine.

For a detailed description of the IR transmission see [2].

3.2.3 Communication protocol

The IR communication protocol already existing for Alice is very general. It accepts only commands such as 'turn right' or 'forward'. Also to change speed, it is necessary to send the command 'parameter period', which means, that the next byte sent will be the parameter which defines the period duration of one of the motors. Since we are only interested in speeds, this is not what we needed at all.

All this led us to the decision of implementing a completely new communication protocol for Alice. The idea for the protocol is the following: **the speeds for all 6 robots are sent one after the other without any pause in-between. Then every robot counts the bits from the beginning and knows at which moment the speeds destined to it are sent.**

This way, addressing is omitted and the number of bits to send is kept as small as ever possible. Since no addresses are used, this can cause communication errors due to lost synchronization between robots and remote control. This happens typically when communication from the PC is started before switching on a robot. Any other disturbance like interrupting transmission with the hand would cause problems in the vision system anyway, and should therefore be avoided. If synchronization is lost for any other reason, the system is able to resynchronize in less than 100ms. Hence there is no disadvantage in omitting the addresses.

To keep the protocol as small as possible, we reserved 3 bits per wheel. As every robot has two wheels and there are 6 robots, the whole protocol consists of 36 bits plus 2 start bits. The 3 speed bits include 1 sign bit. This gives us 3 forward speeds, 3 backward speeds and zero speed. The bits for one robot are coded as follows:

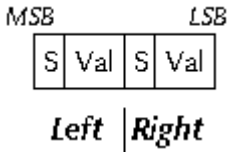


Fig 3.4 Bit coding for one robot

S stands for the sign bit, and Val is the absolute value of the speed.

The communication protocol is defined as follows:

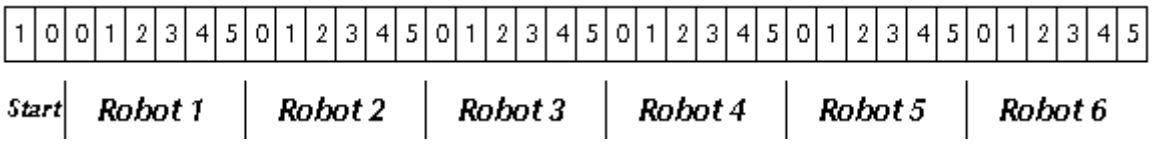


Fig 3.5 Communication protocol

The bits are shifted left for transmission. Of course, this new protocol had to be implemented on both the remote control and Alice.

The IR transmission of the speeds to all robots now only takes 76 ms. With the existing general protocol, this same operation takes up to 1.5 s !

The diminution of the duration of the IR transmission is a very important feature to make the whole system work. In § 5.1 we present a timing of the entire system.

3.2.4 Motion control of the robots

Up to now we have explained what to send to Alice, the speeds, and how to send it, in a defined protocol via infra-red transmission. What we want to explain now, is when to send which speed.

First of all we would like to present what is intended to be accessible to the user, who will implement a strategy. There are 3 possible modes:

mode 0 : direct speed control

mode 1 : position control

mode 2 : behavior (not yet implemented)

In this paragraph we shall explain the second possibility, that is the position control. This control is implemented in the AliceCtrl class. You can find the C++ code in appendix 8.7.1. The task of this class is to calculate the speeds to be sent to Alice, knowing its current position and the requested target position.

All the necessary parameters are stored in one record. This record is the same for all 6 robots, of course. The meaning of the different parameters will be explained in the following.

Robot

- int Mode : control mode
- bool BackwardFlag : arrival direction (mode 1 only)
- int x, y : target position (mode 1 only)
- double theta : target angle (mode 1 only)
- int x0, y0 : current position
- double theta0 : current angle
- int SpeedR, SpeedL : speeds of wheels (mode 0 only)

Control law

The control law is a well-known linear equation:

$$v = k_\rho \cdot \rho$$
$$\omega = k_\alpha \cdot \alpha + k_\Phi \cdot \Phi$$

where

v : velocity

ω : angular velocity

ρ : distance from target position

α : angle between current θ and direction from current to target position

Φ : difference between current and target theta

k_ρ, k_α, k_Φ : motion control constants

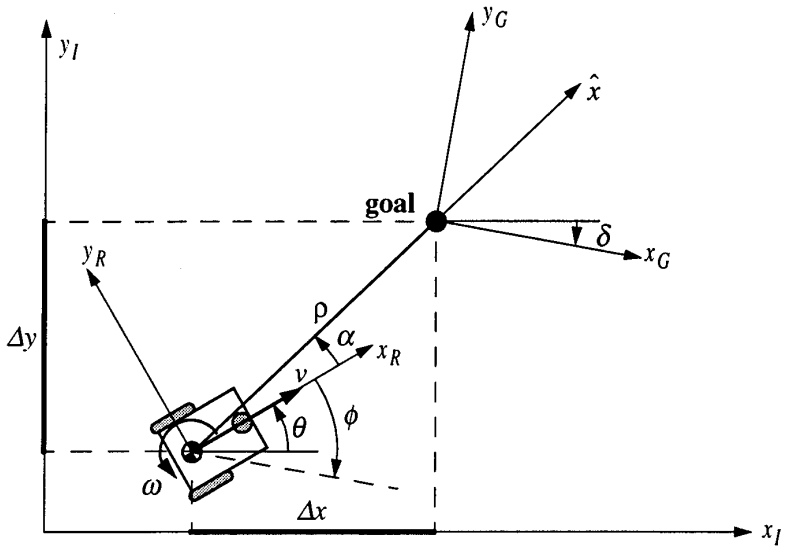


Fig 3.6 Robot kinematics (taken from [1])

All used angles are defined between $-\pi$ and $+\pi$. We did not make a difference between two intervals for α, as explained in [1]. Thus, the robot will always arrive in forward direction at the target position. This seems to be the natural behavior to shoot the ball. Nevertheless, the user has the possibility to make the robot arrive in backward direction. This can be done by setting the Backward Flag to 'true'. See § 4.3.4 for a detailed description of the user's possibilities.

In the position control mode it is not possible to stop a robot. However the user can switch to the speed control mode at any time and set the speed to zero.

The linear control law described above, has an **instability at the target point**. Therefore two basic measures have to be taken:

1. Instead of defining a precise target point and angle, **a target circle and a threshold for the angle has to be defined**. In this target circle, the distance ρ and the angle α can have no more influence to find the right angle theta and are therefore set to zero.

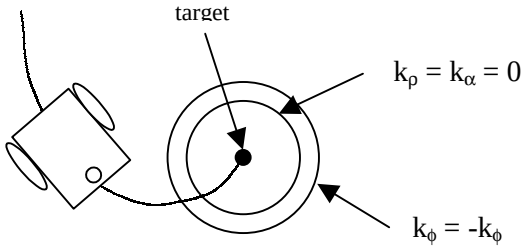


Fig 3.7

2. The k_Φ constant makes the robot move in a smooth manner to arrive with the requested target angle. But once arrived in the target circle, this constant will do exactly the opposite and move the robot further away from its target angle. Hence, in the target circle the sign of k_Φ has to be inverted. Additionally its absolute value is increased. But if the robot stops exactly on the circle and begins to turn to find its final angle, it will sometimes move out of the circle, which will invert the sign of k_Φ again. This is a new instability on the target circle. To avoid this, **the sign of k_Φ has to be changed sooner, on a second, larger circle**. Thus, the robot will still move forward when the sign of k_Φ changes and only stop when passing the inner circle.

A consequence of this control law, which maybe is not desirable, is, that the robot will always arrive with the slowest speed at the target position. If the robot has to shoot the ball, one would like to avoid this. To do so, a target further away can be requested.

Here are the values of the motion control constants described above:

k_ρ	1 s^{-1}
k_α	6 s^{-1}
k_Φ (inside target circle)	-0.5 s^{-1}
k_Φ (outside target circle)	1 s^{-1}
ρ threshold	10 mm
Φ threshold	$\pm$0.4 rad

Quantization of velocities

As already explained in § 3.1.2, Alice can not provide continuous velocity. First, the use of a microcontroller implies a discrete representation of velocity, second the slowness of infra-red transmission does not allow a great number of velocities. Hence, the continuous velocity provided by the control law has to be quantized.

Since there are 3 speeds plus zero speed available, three thresholds need to be defined. This leads to the following quantization:

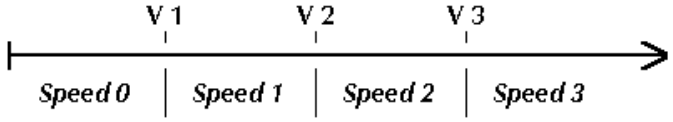


Fig 3.8 Quantization of speed

The values of the three thresholds are as follows:

V1	3 mm/s
V2	29 mm/s
V3	51 mm/s

This quantization is the same for both forward and backward direction. Likewise, right and left velocities are subject to the same quantization. This can lead to an undesirable behavior of the robot. For instance, when it is supposed to move straightforward, it would certainly do so with a continuous velocity, because the difference between left and right velocity would be very little. With quantized velocities, this slight difference can result in different speed values for right and left wheels, when the velocity is just around one of the thresholds. It is therefore **not possible to obtain a smooth trajectory**.

This phenomenon is particularly annoying around the target position. But the problem is rather hard to handle in motion control. To avoid further difficulties, one had better defining reasonable target positions, which are easy to achieve from the current position. For instance, to move along trajectories such as narrow circles, the user is advised to define intermediary positions.

Of course, it would have been possible to add one bit for the speed representation in the communication protocol. This would have doubled the number of available

speeds, and therefore a smoother motion control would have been possible. But on the other hand, this would have increased the length of the protocol by 12 bits, which is 33% (~ 24 ms). A choice had to be made, and we have opted for the 3 bits and the shorter protocol, without ever having tested the 4 bit variant.

Dependency between thresholds

As already mentioned, the behavior at the target position is particularly important, but unstable under certain circumstances. In this context, it exists an important dependency between three of the constants described above: the lowest velocity threshold V1, the Φ threshold and k_Φ inside the target circle.

The calculated continuous velocities for both wheels depend on k_Φ. If k_Φ is too high, oscillations will occur in the target circle. This upper limit is between 1 and 1.5 s^{-1}. This limit is fix and is given by the system's dynamics. The calculated velocity is set to zero if it is below the V1 threshold. V1 can not be set to low, because then oscillations will occur as well. Thus, the robot will not move anymore below V1. And now comes the important bit: velocity can be set to zero even if the angle Φ is still outside the margin defined by the Φ threshold. Hence, the velocity threshold V1 has to be designed in order to respect the Φ threshold at any time.

A dependency of the same kind exists outside the target circle between k_ρ and the other velocity thresholds V2 and V3.

Difficulties

The linear motion control law described above is not that simple to implement in the case of Alice. First of all this control law is not able to manage any possible situation correctly. In other terms, there are several special situations of current and target positions, where the control variables behave exactly the opposite way of what they should. But in a standard situation the same definition of these variables will work correctly.

If one tries to improve the motion control constants for the normal situation, this will automatically make worse a special case. Thus, one has to either treat every special case, which can be quite complex. Or the constants have to be adjusted in order to make the special case behave not to bad, but the normal case not so perfectly well. This is what we have done.

There are several reason, why in the case of Alice, a special situation is more likely to occur:

- The system's frequency, which is subject to variations and which can be rather low
- The time delay between real world and feedback to the robot
- Angle imprecisions
- Variation of the duration of all implied processes
- Quantization of velocity

This makes the adjustment of all parameters not an easy task at all.

Acceleration ramps

The motors of Alice tend to block when speed passes directly from backward to forward and inversely. Therefore a speed change, which changes the sign is not directly executed by the motion control. If this situation occurs, speed zero is first sent to Alice. Only the next time, the requested speed will be sent to Alice.

4 THE COMPUTER RESOURCES

4.1 Organization of computer resources

One of the great challenges of our project is to put together a lot of different computer resources and programs.

First of all, we have **to capture an image** of the field in order **to catch the current position of the players and the ball.** For this task, we chose an existing program written by Remy Blank, COPS, which is capable of processing a digital image and filtering out color blobs. This "Color Object Position Server" can distribute its data via a TCP/IP connection, which represents a great opportunity for us to easily obtain the position of the players in another self-written program. On the other hand and unfortunately, COPS has been written for high quality frame grabbers, which are too expensive for this project. Hence we took the decision to adapt COPS to grab images from a standard USB video camera (see § 4.2.3).

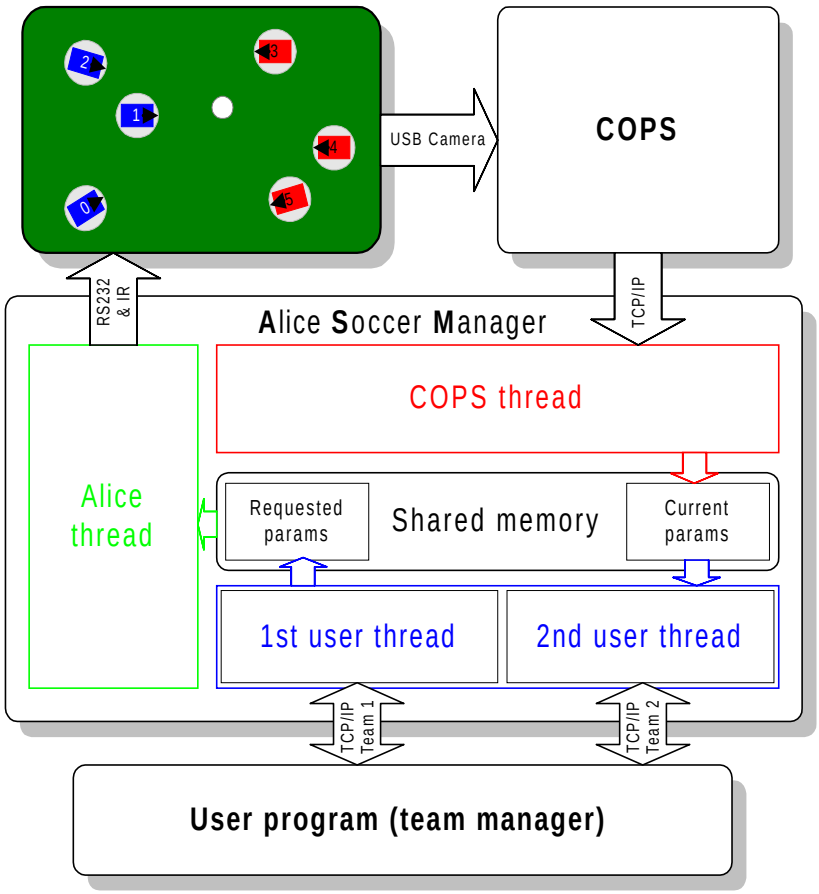


Fig 4.1 Organization of computer resources

After that, we have **the control of 6 robots.** This task requires a RS232 communication channel. Moreover, we have to implement this control in a permanent loop, which takes the parameters from COPS server and reacts in sending speed information for each wheel of each robot, and all that in the most limited period (see § 3.2).

And finally, **two clients or users** (one per team) should be able to interact with the robot control process. We want to maintain a full interoperability and a great flexibility in the possibilities for the users to implement their own programs for the team management. Therefore, we also opted for a TCP/IP communication between users and the main program.

To sum up, we have COPS, which takes images of the field situation, processes them and communicates via TCP/IP with the manager, "Alice Soccer Manager" or ASM. This program contains **several tasks or processes running in parallel**. In fact, we have 4 threads in running conditions. First, we need a thread that maintains the

communication with COPS, and prepares the current positions for writing them in **shared memory**. Then we have a thread for the robot control, which reads the users' whishes (also stored in shared memory). Then, when a user connects to ASM, a new thread, which is dedicated to this user, is automatically launched. This thread takes care of informing its user about the current positions of all players and the ball. It is also responsible for writing the user's commands in shared memory in order to inform the control thread of the client whishes. Of course, there is one thread like this for each user that manages a team.

In brief and for simplifying, ASM contains the 4 following processes running together:

- Alice thread: TAlices
- COPS thread: TCOPS
- 2x User threads: TUser (one per team)

For a more complete overview of this complex organization, you should take a look at appendix 8.4.

Then, for the user program side, we do not want to go more in depth for the moment. Our TCP/IP server, ASM, provides a very high flexibility for the development of a team manager program, and we have made an example in § 4.4.2.

4.2 Color object position server (COPS)

4.2.1 What is COPS?

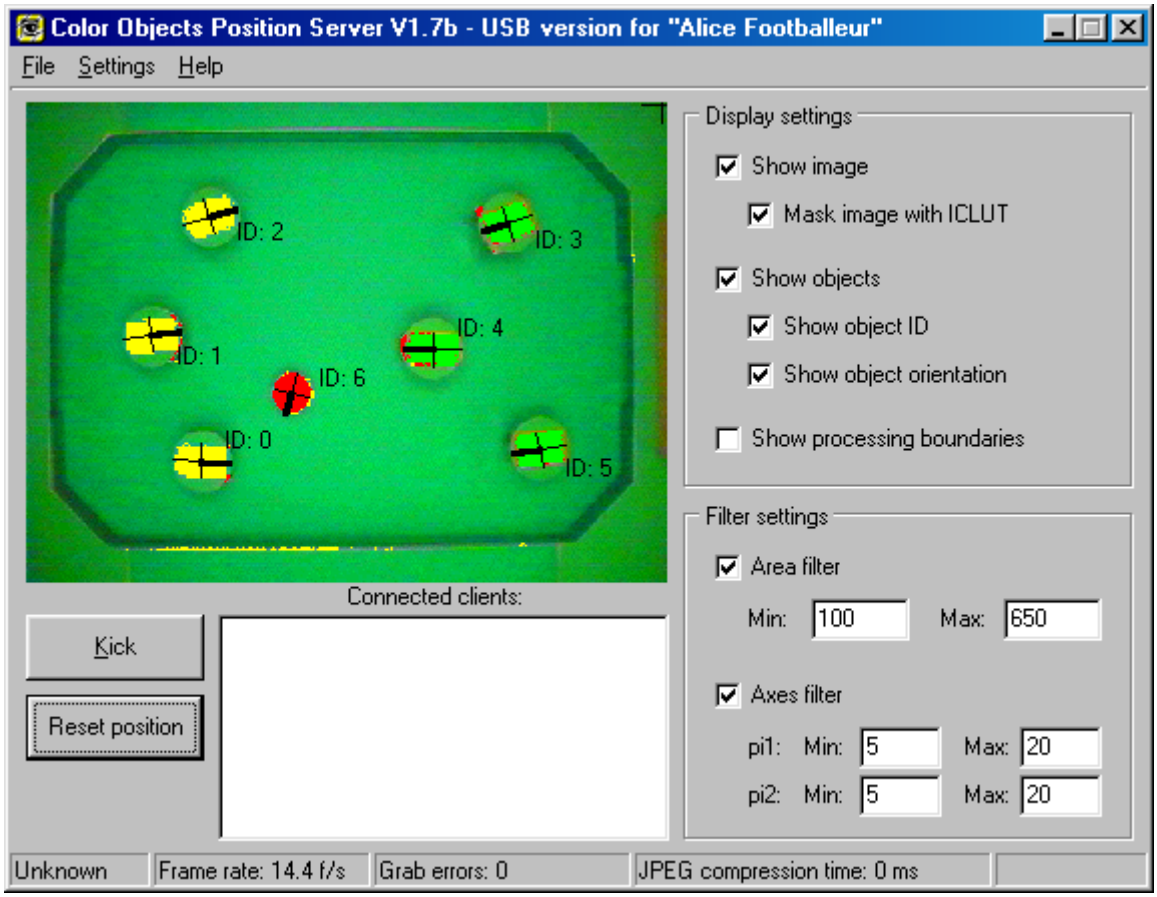


Fig 4.2 COPS user interface

COPS is an acronym for Color Object Position Server. It has been written in Delphi by Remy Blank for another project, "RobOnWeb". As said above, this program is capable of grabbing and processing digital images, **finding out color blobs**, processing their positions and **sharing those parameters via TCP/IP**.

Unfortunately, COPS has been written for high quality frame grabbers – as Matrox – which are too expensive for this project. Hence we took the decision to adapt COPS for grabbing images from a standard USB video camera.

Note that it is not our aim to make an in depth coverage of COPS or even a "user manual", because we are not the authors of this useful program. Since we have made some changes like USB video camera or tracking, we will only put these parts in the present report. Moreover we use the TCP/IP connection, so we will describe its protocol in the next paragraph.

4.2.2 TCP/IP connection with COPS

COPS has an integrated TCP/IP server, which allows us to obtain the necessary data. This server listens on port 1024. It waits for a request and sends back zero, one or more replies.

Requests:

Up to now, we have 8 types of requests:

- idBlobData = 0;
- idImage = 1;
- idRobImage = 2;
- idTimeStamp = 3;
- idGetProperty = 4;
- idSetProperty = 5;
- idListProperties = 6;
- idResetPosition = 7;

but only 3 are useful for our application: the coordinates request (ID=0), the time stamp request (ID=3) and the reset position request (ID=7).

Each request begins with an "int32" (or a "long") which specifies the size of the request without the size of the "Size" variable. After that, we have an "int32" containing the request's identifier, and finally, the parameters of the request.

Here you have **the coordinates request** (ID=0):

```
struct TMBlobData
{
  int32 MsgSize;   // = sizeof(TMBlobData)-sizeof(MsgSize)
  int32 ID;        // = 0
  int32 Period;
  int32 nFrames;
  uint16 BlobData; // = 11
  int32 Format;
};
```

"Period" defines the period at which the coordinates are sent (a value of 2 => 1 image every two captures).

"nFrames" represents the number of images that will be provided as a reply. In ASM, "Period=1 and nFrames=1" is used in order to control each request of coordinates and to know when we will receive a reply.

"BlobData" defines which data will be sent back for each object. It's a bit field where every bit represents a flag that determines if a particular data is sent or not. The interesting values for us are:

- Bit 0: X coordinate in pixel
- Bit 1: Y coordinate in pixel
- Bit 3: Angle in radian

"Format" defines the format of the returned values. The only used value is 0 and the coordinates are sent in float (32 bits) variables.

For the **time stamp request**, we have:

```
struct TMTimeStamp
{
  int32 MsgSize;  // = sizeof(TMTimeStamp)-sizeof(MsgSize)
  int32 ID;       // = 3
  int32 OnOff;
};
```

This request produces no reply. It just adds the time stamp information to the normal coordinates reply.

The third useful request is the **reset position request**, which was implemented specifically for this project. Its simple action is to make a reset of the ID and the direction for each robot, at the beginning of the game (see § 4.2.6):

```
struct TMResetPosition
{
  int32 MsgSize; // = sizeof(TMResetPosition)-sizeof(MsgSize)
  int32 ID;      // = 7
};
```

It either provides no reply.

Replies:

The replies also begin with an "int32" which give the size of the reply without the size of "Size". Then follows the identifier of the corresponding request and the data of the reply. In fact, the four bytes of the identifier will contain a request ID and a reply sub-ID but we are not interested in this detail. For the request number 0 (the coordinates request), we will obtain:

```
struct TABlobData
{
  int32 MsgSize;    // =sizeof(TABlobData)-sizeof(MsgSize)
  int32 ID;         // = (0 << 16) + Format
  int32 TimeStamp;  // if required with TMResetPosition
  int32 Count;
  uint16 RecSize;
  uint16 BlobData;
  struct TObjData
```

```
   {
     float X, Y, Theta;
   } Blobs[];
 };
```

"TimeStamp" gives the time, in milliseconds, when the image was grabbed.

"Count" gives the number of objects contained in reply. For our application, ASM, we always have seven objects.

"RecSize" gives the size of one object: "sizeof(TObjData)".

"BlobData" is the same bit field as in the request.

"Blobs[]" is an array containing a structure for each object. The above specified corresponds to "BlobData"=11, as seen before.

Note:

You will find no appendix of the code of this server because we have not modified it during our project. We just use it, like the users will use ASM. We just have to know the global architecture and protocol.

4.2.3 Integration of USB video camera

Since it was a job made during our summer work experience, we don't want to give a detailed account of this subject. However, it is good to mention some important points.

We spent a lot of time during the summer in trying to understand how to intervene in the frames stream between USB video camera and the screen. Eventually, the "Video for Windows" (VfW) technology was chosen. Then we derived a class from the existing "Grabber" class from COPS, integrating VfW in Delphi code. You can find this class in appendix 8.6.1.

With the USB video camera, we obtain an **average frame frequency of about 15 FPS** against 25 FPS with a good frame grabber and a analog video camera.

4.2.4 Camera settings and color recognition

There are mainly two levels of settings in the image processing system. The first one resides in the **dialog box "Video source" of the camera**. The second one is the **"scanner settings" box of COPS**.

We first have to confess that our solution, here, isn't so good, because it is a temporary solution. First of all, if such a complex system was delivered to students, it would certainly not be a good idea to let them have access to these settings. It would be better to implement, directly in COPS, a module, which tries to vary the different parameters, until it finds a match between a color rectangle on the playground and a reference. However, this was not the aim of this project, so we found a temporary solution in order to manually adjust the image processing parameters.

Video source

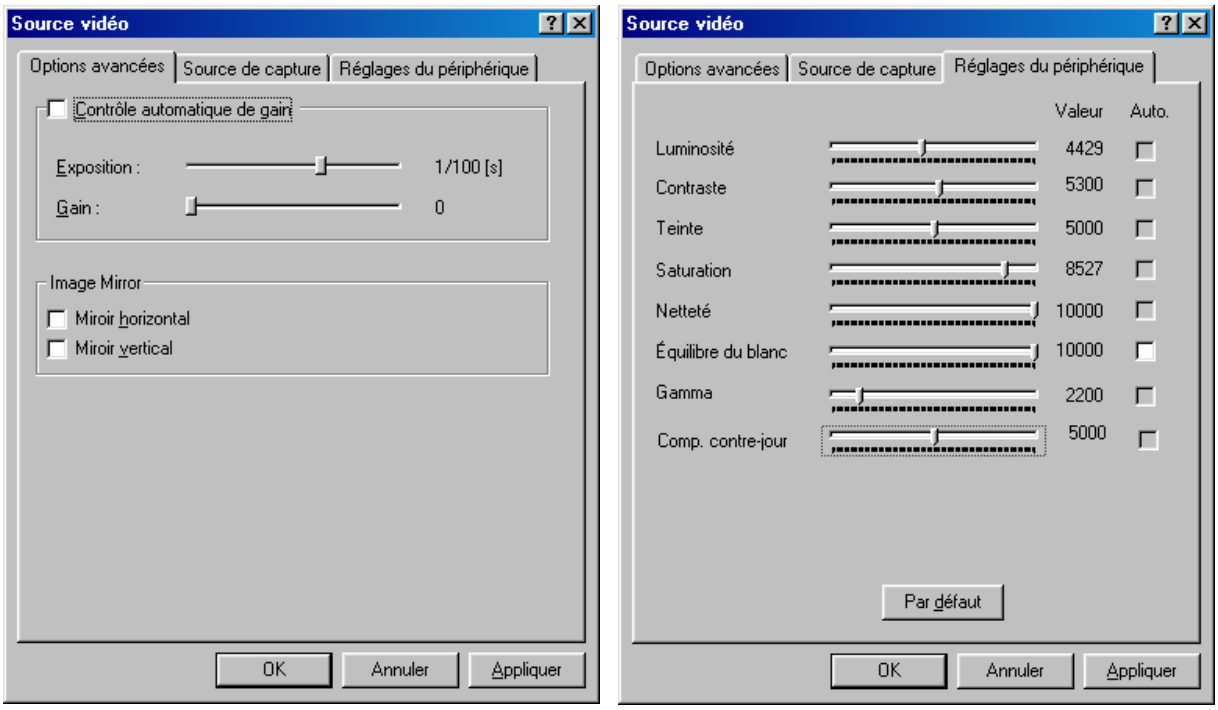


Fig 4.3 Video source dialog box

This dialog box, which comes from VfW or directly from the driver, is automatically opened on the launch of COPS. One can adjust here the video camera settings. In fact, it is preferable to always keep the same values, which were finely chosen at the beginning, according to the color of the field and of the players. Currently, exactly the settings that are on the above screen shot are used.

Scanner settings

In COPS, you can find this scanner settings in the "Settings" menu.

This dialog box allows to **select the range of pixel colors that will be interpreted as one type of object** (players or ball). These settings are relatively difficult to set and very **dependent on the ambient light and shade zones**.

This tool also determines the "wrong colors" attributed to each object. That way, we can easily recognize which pixels have been interpreted as being part of the same range of color.

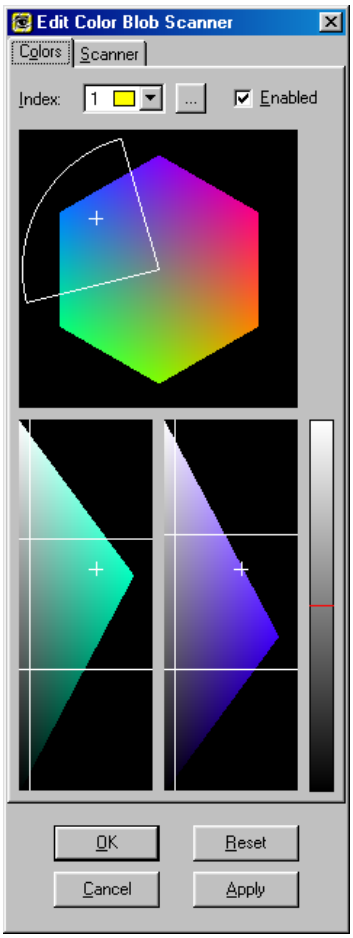


Fig 4.4 Color blob scanner settings :

Here below, you can see the normal image (on the left) and the same image (on the right) with the masks of wrong colors.

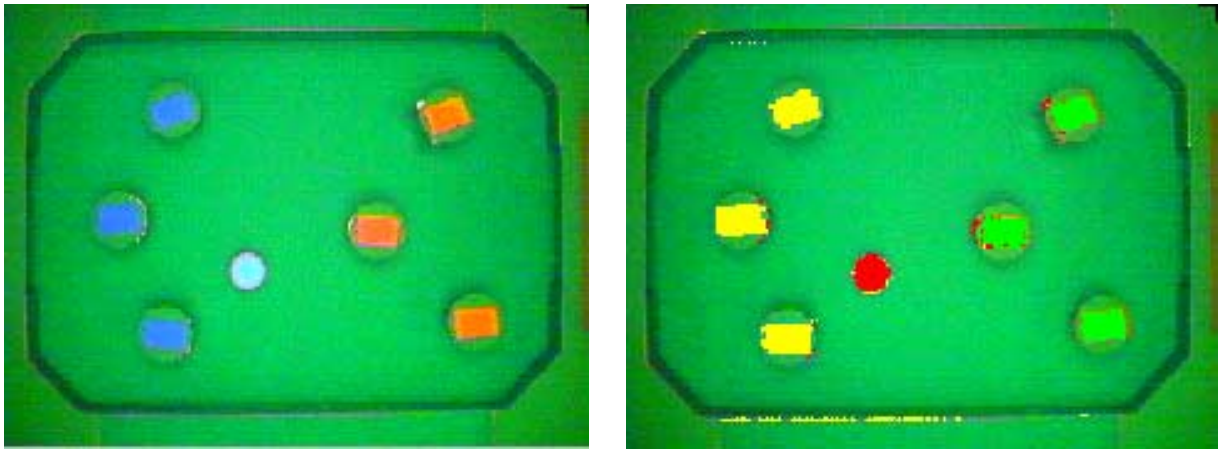


Fig 4.5 COPS image (left: without wrong color mask, right: with wrong color mask)

The choice of the color is very important. They have to clearly stand out in order to be well interpreted by COPS.

4.2.5 Object tracking

In its original version, COPS was not able to track objects because it was not necessary: each object had a different color. Now, we have 2 teams, each consisting of 3 robots of the same color. Thus we have 7 objects (with the ball) and only 3 colors. Therefore we implemented a **tracking module directly in COPS** (see appendix 8.6.2). Moreover we had to determine an initial position for each robot in order to give them the correct identifier, at the beginning (see § 4.2.6).

Note that COPS already contained an **angle tracking**. Of course, the form of the color patches globally determines the angles. But COPS has to track the direction because the rectangle doesn't give it. Indeed, if we take the orientation of a rectangle main axis, we have two possibilities to attribute an angle to it (±180°).

4.2.6 Initial positions

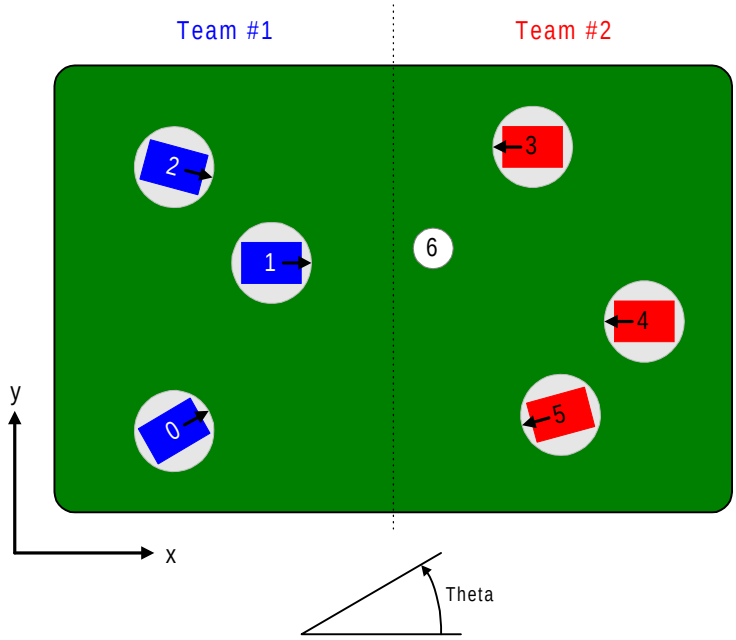


Fig 4.6 Initial positions on the field

At the beginning of a game, the identifiers of the robots and their directions have to be determined. Therefore an initial position for each player is defined. Each team must be on its side of the field and each player is in direction of the opposite team. Thus the ID can be attributed: ID 0 (for y min), 1 and 2 (for y max) for the blue team (#1) on the left side and 3, 4, 5 for the red team (number 2) on the right side.

The ball always has the ID 6.

Thus, in our module, on the reset event, objects are first ordered by color (blue->red->white) and then according to Y coordinates.

Here, you can see an image taken by COPS, just after a reset:

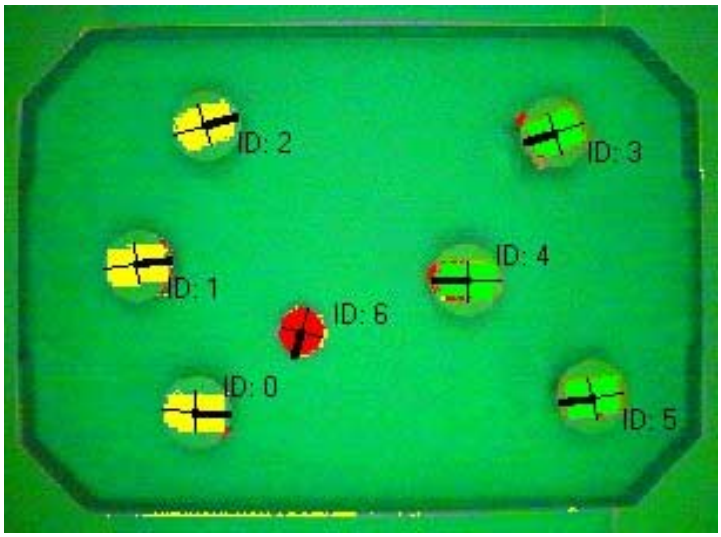


Fig 4.7 Initial position

4.3 The server program: Alice Soccer Manager

4.3.1 General considerations

This program has to simultaneously handle a lot of tasks: controlling the robots, communicating with COPS, and taking care of the users. Since the complexity and interconnectivity of these tasks are important, we deliberately opted for an **asynchronous architecture** with a **shared memory** in its center. There is a thread for each task, which runs separately and takes the required values from shared memory. This memory is updated by different threads as well.

Since ASM manages TCP/IP connections, it is important to know that the data will transit through these channels in form of successive bytes. ASM has been written on a Windows platform. Therefore we use an Intel format which means: the most significant byte is first.

In the following, we paid attention to the compatibility between the memory format and the TPC/IP format in order to have the possibility of simply putting a pointer to a structure into TCP/IP functions. For example, we only use types which take an even number of bytes in order to avoid unused space in memory:

- Long (or int32): integer on 4 bytes
- float: real on 4 bytes

This allows the use of the well-known "sizeof()" C++ function to calculate the size of packages that will be sent through TCP/IP.

Note that in the followings the name of "request" and "reply" are related to client and server notions. The client sends "requests" to the server. The server sends "replies" back to the client.

4.3.2 Alice thread

The task of the Alice thread is to **manage the communication with the robots** on the field. It updates the current (provided by COPS thread) and target (provided by User thread) positions in the class, which manages the motion control (AliceCtrl). The values are read in shared memory. Then it executes the motion control and the transmission of the speeds to Alice. Since the speeds are sent to all robots at once, 5 bytes (= 40 bits), containing all speeds, are sent to the remote control. Only 36 of the 40 bits are used. This is unavoidable since the RS232 protocol only allows entire bytes to be sent. The loss of time is negligible anyway. Please refer to § 3.2 for a detailed description of the infra-red transmission and the motion control of the robots.

Subsequently, this thread sleeps during the infra-red (IR) transmission of the speeds to all robots (managed by the remote control). The time taken for the IR transmission is very well known (currently this is 76 ms). But the 'Sleep'-command in C++, although being the most precise command, is not that precise due to the multitask character of the operating system. The sleep time never descends below the value stated in the command, but it can increase up to 50% of the value stated. Nevertheless the time between two transmissions is quite stable around 84 to 92 ms.

The sleep time should be kept as small as possible (~80 ms) to assure the most direct feedback loop for motion control. Likewise, it should be kept as precise as possible in order to apply stable constants in motion control. Hence, the observed variations of up to 50% should be diminished as much as possible. Increasing the priority of the Alice thread does not help. But there are other possibilities to achieve this:

- use a more precise timer
- sending a reply from the remote control to the PC (via RS232) when transmission is terminated

4.3.3 COPS thread

This thread implements **the client side of a TCP/IP connection in order to get the current value of coordinates, angles and time stamp from COPS**. Its architecture is quite simple. At the start up of ASM, this thread is automatically launched. At the beginning, it sends two requests (see § 4.2.2), one for the time stamp (ID=3) and another to obtain the first values from COPS (ID=0). Then it enters a permanent loop that terminates only when ASM is closed. This main loop receives the data from COPS, processes them, puts them in shared memory and sends the next request to COPS.

By "processing the value from COPS", we mean:

- Scaling X and Y coordinates (in pixels) in order to convert them into millimeters. This operation depends on the height of the camera.
- Adding an offset to this coordinates in order to match the lower left corner of the field with the origin of the Cartesian coordinate system.
- Making an inversion of coordinates for one of the teams (see in § 4.3.4 "Equality of both teams").
- Correcting the angle values (this problem comes from COPS).

You can find the code of this thread in appendix 8.7.3.

The main loop is automatically synchronized with COPS frequency, because it always has to wait for a COPS reply. COPS does not reply as long as there are no new values to send. Since COPS presents an average frequency of 15 FPS (see § 4.2.3), we obtain about the same value for this thread. **So we can say that the current parameters of the objects on the field are updated in shared memory about every 70ms, on average.**

Of course, the same structures as in COPS (see § 4.2.2) have to be defined in ASM in order to send the correct set of bytes. At the reception, the data is put in the corresponding structure, and because there is no missing and no inversion of bits between COPS and ASM, all bytes will reach their correct place in the structure.

4.3.4 User threads

The Alice Soccer Manager implements a **TCP/IP server**, which listens on port 1025. The protocol is a standard TPC/IP protocol with request and reply. The request and reply packages have a unique and well-determined structure.

ASM doesn't reply immediately when a client has sent a request. The maximum average frequency is given by image processing. This means, when a request from a user is received by ASM, it waits for an event from COPS which indicates that new values are available in shared memory, before sending back a reply.

Main articulation of the user thread

When a user connects to ASM, a new user thread is launched which processes only the communication with this user or team. **The first user that connects** to ASM **will receive the first team on the field**, the blue ones, and the second, the red ones. This also corresponds to the TeamID (TeamID 1 = blue, TeamID 2 = red), as you will see below.

Each team has its own process running in ASM. That means when everything is ready to begin a game, two user threads will be running on the server, regardless of the number of programs that are effectively connected. We can imagine a unique program that establishes two connections with ASM and calculates the new targets for both teams. However, this architecture has been created in order to allow two isolated computers to represent each a team.

Below, you can find the main articulation of the user thread:

```
on client connection : wait for the first request
while thread is not terminated
      wait for COPS event
      send data (current position of objects,…)
      wait for request (target of robots,… )
end while
```

For more details, you can find the C++ code in appendix 8.7.4.

Structures

The following structures are the same on both side of the TCP/IP connection. That means we will find the same structures in client (team manager) and server (ASM) programs.

- RobTarget (20 bytes):
  - long Mode : control mode
  - long Flag : depending on the mode
  - float SX : speed of left wheel or x coordinate
  - float SY : speed of right wheel or y coordinate
  - float Theta : arrival angle (only mode 1)
- Object (robot or ball) (16 bytes):
  - float X
  - float Y
  - float Theta
  - float Speed
- DataRequest (3x4 + 3x20 = 72 bytes):
  - long Size : size of request without size = 48
  - long ID : request identifier

  - long Period : period of replies
  - RobTarget Robot[3] : target of each robot

- DatatReply (5x5 + 2x3x16 + 1x16 = 137 bytes):
  - long Size : size of request without size = 133
  - long ID : reply identifier
  - long TeamID : team identifier
  - long Mode : control mode
  - long Error : number of an error
  - long TimeStamp : time stamp of image capture
  - Object MyRobot[3] : current parameters of robots of this team
  - Object YourRobot[3] : current parameters of robots of other team
  - Object Ball : current parameters of the ball

Definitions

- **Mode**: sets the current control mode (see § 1.2):
  - 0 : speed (this robot will receive a speed for each wheel)
  - 1 : position (this robot will receive a target position)
  - *2 : behavior (this robot will receive a number of behavior, not yet implemented)*
- **Flag**: depending on the current mode
  - Mode = 0: not used
  - Mode = 1: direction of the arrival (forward or backward) on the target
  - *Mode = 2: choice of behavior (not yet implemented)*
- **SX** and **SY**: depending on the current mode
  - Mode = 0: SX is the speed of the left wheel and SY of right wheel
  - Mode = 1: SX is the x coordinate and SY the y coordinate of target
  - *Mode = 2: parameters of behavior (not yet implemented)*
- **X** and **Y**: position in mm of an object in Cartesian coordinates.
- **Theta**: angle in radian of object vector. The vector points in forward direction of robot. In the case of the ball, this angle has no significance.
- **Speed**: *(not yet implemented) speed of each object.*
- **Size**: size of the TCP/IP package without the size of first variable (for example, SizeOf(DataRequest) – 4).
- **ID**: identifier of request or reply. Currently, we only have ID = 0 for both, request and reply. This byte allows future extensions of the protocol.
- **TeamID**: identifier of team. Same as team number. The first user connected receives number 1.
- **Period**: lets the user change the period of the replies. *At the moment, only n=1 has been implemented.*
- **Error**: *(not yet implemented)*
  - 0 : OK
  - 1 : one or more X or Y coordinates is/are out of range
  - 2 : one or more angle Theta is/are out of range
  - 3 : unknown request ID

    - 4 : unknown mode
- **TimeStamp**: time stamp of image capture. This number is given by COPS and represents the time (in ms) of the image capture. The t=0 is set at launch of COPS. Note: the uncertainty of this time indication is relatively important, about 10ms.

Equality of both teams

Since we want to have two different teams playing against each other, and not one team always being played by the computer, the problem of equality has to be solved. In other terms: because the coordinate system's origin is fix at a certain point, **it has to be decided which team plays on which side**. Strategy would naturally depend on the coordinate values chosen according to the side on which a team has to play. Hence, the strategy would have to be implemented in order to allow a team to play on whatever side it will be asked for by the user thread.

But it is not reasonable to ask of a user strategy to have this flexibility. It is much easier to manage this problem directly in the user thread. **Thus, every strategy could be implemented to play always on the same side**, and actually every user should then choose the same side. Naturally, we shall chose the side on which x values are lowest to be the side of the team's goal.

To achieve this, we proceed as follows:

In reality, there will be one team (in occurrence the 2nd team), which won't be playing on the side, its strategy has been implemented for. Hence, the user thread has to pretend to this team, that it is playing on the side its strategy is implemented for. This can be done by **a simple reflection on the center point of the field**. Before sending the current parameters of all robots and the ball to the user, they are reflected on the center point. The target positions then received from this user, are reflected to fit the reality again, before making them accessible to the Alice thread (motion control).

This way, **all users can implement their strategy exactly the same way**, and the system is completely free to have any team to play against any other team.

Obviously, the reflection of the user request only has to made in mode 1 (position control). In speed control mode, the same variables will contain speed values that must not be reflected. Refer to the preceding paragraph for the functionality of the different modes.

4.4 User programs

4.4.1 Connection with ASM

The users – or team managers – have to know a little about the user thread behavior and working. Therefore we invite the reader to take a look at § 4.3.4.

The user programs have to implement a TCP/IP client connection to ASM. In order to do this, we can use the Microsoft Winsocket component, for example in Visual Basic or in other Microsoft developer environment, or TClientSocket in Borland environment such as C++ Builder or Delphi. **The TCP/IP is a very easy to use protocol that allows interoperability**. That means you can develop a program on MacOS, in which you will implement a connection with ASM, which runs on Microsoft

Windows 98, as seen in § 2.5. This is why this kind of communication between users and ASM has been chosen.

As said in § 4.3.4, the first user connected to ASM receives the first team identifier. The coordinate system is **side independent** (see "Equality of both teams"). And the maximum sampling rate of request-reply is given by COPS and is about 15Hz.

4.4.2 An example of soccer strategy

During this project we have implemented a very simple strategy to test the system's performance, and to demonstrate the possibility to play soccer.

There is one goalkeeper who simply moves back and forth in front of the goal. Then one player in the front part of the field tries to shoot the ball towards the opposite goal, the second player does the same in the back part. When one player goes towards the ball at the left side of the field, for instance, the other one is always trying to block the fields for the opposite team at the right side.

It was not the goal of this project to implement an intelligent strategy. In fact, it is very hard to make a strategy, where the robot does not head into other robots. Therefore the robots often get stuck. It is up to the user to make a strategy in order to avoid this problem.

The user can simply use the structure of this C++ program to develop his own strategy. Every time the user program receives a reply from ASM, the 'GetNextRequest' function in the 'Strategy' class is executed. For each robot a 'GetReqRob' function is already implemented. The functions for all three robots are subsequently executed. The user can specify the control mode, the flag (the meaning depends on the mode), the x/y/θ coordinates in position control mode and the speeds for the two wheels in speed control mode. Please refer to the description of the user thread in ASM (§ 4.3.4) for a detailed description of the meaning of the different variables.

The user strategy should finish execution in less than 70 ms. This is the period of the image processing done by COPS. If the strategy time exceeds this limit, only every second image can be sent to the user. This will considerably increase the time delay between image acquisition and command execution on the robots, and should therefore be avoided.

4.5 Overview: programs and classes

To have a complete and direct overview of all files and programs we have implemented or modified, see the following table.

ASM Alice Soccer Manager

File	Classes	Header file	Description
ASM.cpp			Runs the application
Main.cpp	TFormASM	Main.h	Main form definition and launch of threads
TCOPS.cpp	TCOPS	TCOPS.h	COPS thread
TUSER.cpp	TUser	TUSER.h	User thread
TAlices.cpp	TAlices	TAlices.h	Alice thread
AliceCom.cpp	TAliceCtrl TSendAlice	AliceCom.h	Motion control of the robots Management of serial port COM1
Constants.h			General constants used in several files

USER Team Manager

File	Classes	Header file	Description
User.cpp			Runs the application
Main.cpp	TMainForm	Main.h	Main form definition and connection with ASM
Strategy.cpp	TStrategy	Strategy.h	Implementation of soccer strategy

COPS Color Object Position Server

Only added or modified files or functions are mentioned

File	Classes	Functions	Description
Main.pas	TMainForm	TrackBlobID SetTag SortEvenColor SortEvenTag SortEventPos	
BlobSRV.pas	TBlobServer	TrackBlobAngle	
GUSB.pas	TGUSB		Inherited from TGrabber

5 PERFORMANCES

5.1 Time balance

One of the main difficulties of this project is the management of the timing. Therefore, a balance of the different time values and delays is presented here.

All measures taken with the PC have a certain imprecision due to the use of the 'GetTime' function. The time values are subject to variations because of the non-real-time platform. Hence, the following considerations have to be taken with caution.

The most important value is the time delay between the real world situation and the command execution on the robots, that is the period of the whole loop, including every single step.

In the following figure, you can see an approximate timing between the different tasks. The vertical arrows represent the occurring events. All events occur in ASM. Arrows pointing to the time axis mean reception, arrows pointing away from the time axis represent going out messages. Events on the same time axis are synchronized.

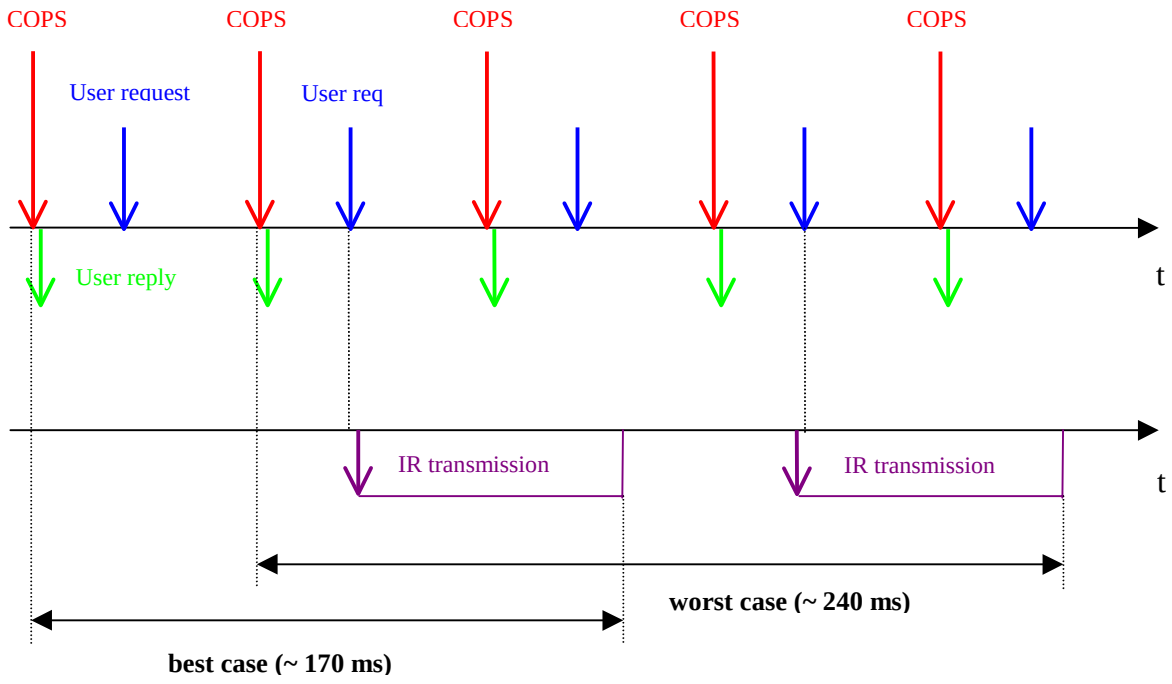


Fig 5.1 Timing diagram

The COPS event indicates that the acquisition of a new image is complete. At the same moment the next image grabbing is started. To make the sum of all necessary steps, one has to begin with a COPS event (beginning of image capture) and then continue with the next COPS event (end of image capture). Directly after the reception of a new image, a reply is sent to the user with the current positions of the robots. When the strategy calculations are finished in the user program, ASM receives a request from the user containing the new target positions (or speeds). The received values are stored in shared memory.

The Alice thread which manages the infra-red transmission works asynchronously to the rest of the system. IR transmission is done with a fix period independently of

image acquisition and user programs. For every IR transmission, the values in shared memory are taken, which are the ones recently updated by the user.

Obviously, the **best case** is, when IR transmission is executed right after the last update from the user. The **worst case**, on the other hand, is, when IR transmission is executed just before the reception of a new request. In this case one entire COPS cycle is added to the best case, because the reception of the user request has the same period as COPS for it is synchronous with COPS. We are speaking here of the total time delay between real world and command execution on the robots.

However, the period of the IR transmission is much shorter (~ 90 ms). Since the IR transmission takes longer than the image capture of COPS (~ 70 ms), we are sure to have always updated values to transmit. For the same reason, it is sometimes possible that a user request is not transmitted to Alice, as it is the case for the fourth user request in above figure.

It is interesting to know what distance Alice travels during one entire period of the time delay and during one period of the IR transmission. The values are the following.

Traveled distances				
during	at 29 mm/s		at 51 mm/s	
	min	max	min	max
total loop	4.9 mm	7.0 mm	8.7 mm	12.2 mm
IR transmission	2.4 mm	3.8 mm	4.3 mm	6.6 mm

It is important to mention, that all these values depend on the duration of the user strategy. In the above timing diagram it becomes obvious, that the user should take care to do strategy calculations in less time than COPS takes for image acquisition. Otherwise the user would only get every second image.

5.2 Unsolved problems

Alice getting stuck

As seen in § 3.1.3, the metal sheets around Alice can make the robots get stuck with other robots. This is a problem which should either be solved with a more appropriate construction of these sheets or with conditions and procedures in the software to take the robots apart when they get stuck. But in the speed and position control mode, we can certainly say that this is a problem to be solved by the users.

In another way, we encounter some problems with the borders of the field. Unfortunately they were built with the same height as the robots. So when an Alice wants to move into a wall, it tends to get stuck with its green cardboard cover on the field borders. This problem can easily be solved by increasing the height of the borders. Moreover, we could implement a special procedure in the control that avoids the wall. But this last solution is quite complex, above all in the case of position control, because a prediction of the trajectory would have to be made.

Alice achieving its target position

We have already explained that some problems persist in the motion control of the robots (§ 3.2.4). Sometimes Alice has some difficulties in achieving its target position. Particularly when Alice does not arrive in a favorable direction in the target circle, it can take longer to achieve the right angle.

However, since Alice is supposed to play soccer, it does not necessarily have to achieve every single target position, because target positions will change dynamically without Alice ever completely achieving them. On the other hand, the user can always define intermediary positions, which are favorable for the desired trajectory.

Timing accuracy

On a Windows platform and with a lot of processes running at the same time, it is quite difficult to make a good job in real time, as you can read in § 5.1. This is why we have chosen to build an asynchronous system.

But if the whole system could be implemented on a real-time platform and in a single program, a synchronous architecture could lead to a much better timing accuracy. For example, if we put COPS and ASM in a single, non-multithreaded program, we could define a good chronology of the events:

1. Take an image
2. Send the current parameters to users
3. Wait for user requests
4. Send the commands to robots

This would allow to have a better position accuracy with the control. However, in the current situation, with such timing fluctuation we had better keep our asynchronous system.

Automation of image processing settings

As said at the beginning of § 4.2, it would be necessary to implement a module that is able to automatically set the right parameters for the image processing system. We already have some theoretical solutions but it would represent another important project to implement them. However, such a module would be absolutely necessary in case of a large scale distribution of the system.

6 CONCLUSION

6.1 Alice is ready to play soccer

We are pleased to say that the system works fine and demonstrates the possibility of playing soccer with mini robots. We have certainly built the smallest robot soccer game in the world. Moreover, the small size of this setup opens up new horizons in the field of robotics soccer. Actually, if we can put a robot soccer game in a briefcase, we significantly increase the opportunity for everybody to manage and program such a fascinating game.

More important, this will allow to combine soccer with the education domain. Actually, a handy robot soccer kit can be proposed to schools to teach the students programming in a recreational way.

The small size of the used robots still offers other advantages. First, it is well-known that the lower the weight, the higher the robustness. Second, the power consumption also decreases significantly, which means that the robots have a longer running time. And Alice, with its ten hours of autonomy, is a good example for this last point.

These considerations made us declare that it is an excellent idea to propose this game with an improved setup for a robot soccer championship during "expo.02", the Swiss National Exhibition taking place in 2002. It would be great if many teams of students from the whole country could be playing against each other with their self-programmed robots.

Our setup is a very open system. It is a good base on which other developers can easily build some programs like a simpler user interface for younger students or advanced strategy for high level contests.

Finally, the developed system for the localization of mobile robots could be used in many other applications, particularly with mini robots in educational applications (see for example "Kit de robotique pédagogique", in [k], in appendix 8.2). An idea is to commercialize a handy kit for education, which allows to run several different applications and games.

We are looking forward to seeing Alice taking part at robot soccer contests and other exciting events we do not even imagine yet!

6.2 Future developments

First of all, the final handy setup has to be designed. It must be possible to put the whole system in a briefcase. This system consists of a field, a USB video camera, lighting and a laptop. The field must be adjustable in order to get a flat position. The camera and the lighting must be self positioned at the opening of the case. The laptop (at least 200MHz with 32MB RAM) must have an USB port, a serial port and an Ethernet card. An included hub would be a good thing in order to receive two user connections.

Other options could be integrated in the final setup such as:

- Automatic score counter
- Automatic return of the ball when a goal has been scored

- Automatic return to initial position of the robots

Of course, it would be interesting to take into account the remarks of § 5.2 Unsolved problems, if it is possible to improve those characteristics.

Finally, the integration of a simulator would be an excellent addition. Knowing that the system will not be that cheap, the use of one or two setups in a class of about twenty students could be made considerably easier, if students could test their strategy on a simulator, before taking them to the real setup.

Moreover, it does not seem to be a great challenge to implement this simulator. In this context, we have already contacted the creator of "Webots" (see [j] in appendix 8.2). Only a few changes must be done on the existing simulator of Alice.

6.3 What we have learnt

During this semester project, we covered a lot of fields of knowledge. For example, we have programmed in assembler (for the PIC on Alice and on the remote control), in Delphi (for the modification of COPS) and in C++. Particularly in this last language, we made important progresses. Beforehand, our only knowledge was C programming. So we learnt to manipulate the whole concept of object oriented programming. This implies the knowledge of terms such as class, object, inheritance, socket, multitasking, windows based messages, event handling and multimedia management. We were also lead to work with several types of communication like RS232, USB, IR and TCP/IP.

The fact that the project covers very different domains is typical of a micro engineering project. This is what made it very interesting for us. For example, we made a foray into mechanics (the physical setup, the metal sheets), informatics, robotics and image processing.

6.4 Thanks

We would especially like to thank Rémy Blank for his kind attention and his help during our hard moments in Delphi or C++ programming. We have learnt a lot with his precise answers and his explanations about COPS architecture.

A big thank you to our assistants, Patrick Balmer and Gilles Caprari, who were always available for help and interesting propositions and supported us at every moment.

We would also like to thank Michel Lauria for the first questions about robotics soccer, and Jean-Marc Koller for the first consideration about real time image processing.

Thanks to Antonio Lopez for the production of the metal sheets.

7 BIBLIOGRAPHY

[1] ROLAND SIEGWART, *Autonomous mobile robots*, Cours "Systèmes autonomes", EPFL – DMT – ISR – ASL, Lausanne, 1999

[2] GILLES CAPRARI, VLADIMIR VUSKOVIC, *Mobiler flexibler Miniroboter*, Diplomarbeit am Institut für Robotik IfR, ETHZ, Zürich, 1996

[3] KENT REISDORPH, *Borland C++ Builder 4*, Unleashed, Sams Publishing, June 1999

[4] JOE CASAD & BOB WILLSEY, *TCP/IP*, Le tout en poche, CampusPress, 1999

[5] MARCO CANTÙ , *Mastering Delphi 4*, SYBEX, 1998

8 APPENDIX

8.1 Glossary & abbreviations

- Thread: a separate task or process that runs parallel to other ones in a same program
- USB: Universal Serial Bus
- VfW: Video for Windows
- ASM: Alice Soccer Manager
- COPS: Color Object Position Server
- TCP/IP: Transport Control Protocol / Internet Protocol
- PC: Personal Computer
- IR: Infra Red

8.2 Related web sites

[a] http://dmtwww.epfl.ch/isr/asl/projects/alice_pj.html

[b] http://dmtwww.epfl.ch/~jzuffere/alice_soccer.htm

[c] http://www.k-team.com

[d] http://www.robotik.org

[e] http://www.robocup.org

[f] http://www.fira.net

[g] http://www.iroc.org/English/Robotolympiad/index_olympiad_eng.htm

[h] http://www.mein.nagoya-u.ac.jp/maze/index.html

[i] http://www.logitech.ch

[j] http://www.cyberbotics.com

[k] http://dmtwww.epfl.ch/~pramer/cps.html

8.3 Alice

8.3.1 Speeds of Alice

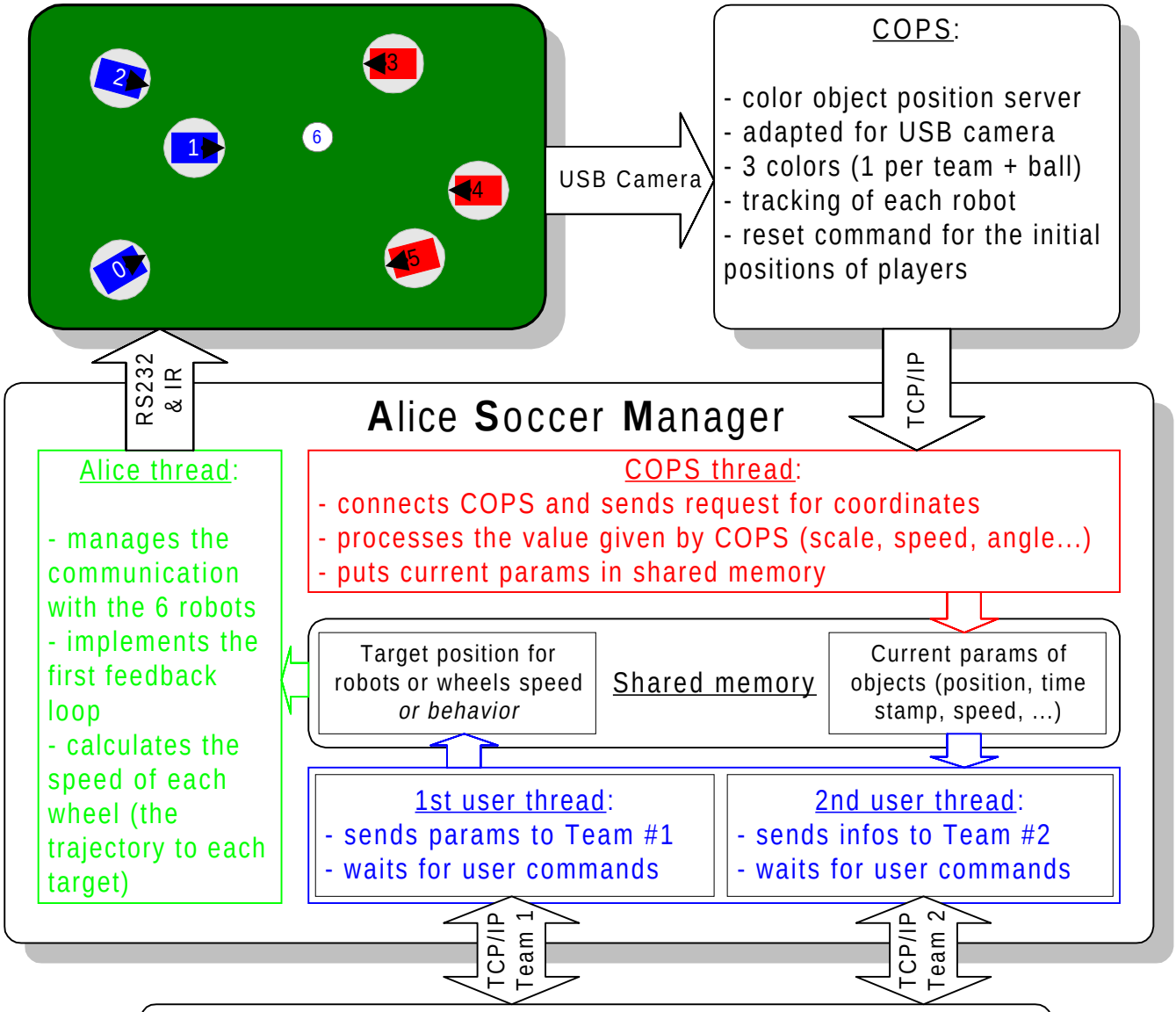
Speed	Phase Period [multiple of 143 us]	Pause between cycles	Calculated velocity [mm/s]		Velocity [mm/s]	Rotations [1/s]	Stability	Remarks
	20	0	12.8		12		**stable**	left faster
	15	0	16.0		16		**stable**	left faster
	11	0	23.3		23		**stable**	forward: right faster, backward: left faster
	10	0	25.6		25		**stable**	slight jump forward when moving backwards
	9	0	28.5		28		**stable**	
	8	0	32.0		32		**stable**	
0	7	11	29.0		29		**stable**	right wheel faster
1	7	8	30.8		30	0.44	**stable**	left wheel faster
				Loaded	30			
2	7	5	32.7		32	0.49	**stable**	
				Loaded	32			
3	7	2	34.9		34	0.58	**stable**	right wheel faster
				Loaded	34			
4	7	0	36.6		37	0.62	**stable**	
				Loaded	35			
5	6	0	42.7		43	0.75	**stable**	
				Loaded	42			
6	5	0	51.3		51	0.9	**stable**	
				Loaded	51			
7	4	0	64.1		62	1.1	not stable	
				Loaded	62			
8	3	0	85.4		86		not stable	
				Loaded	86			

Formula for velocity : v = pi*d/(143us *(6*PhPeriod + Pause)*180)

8.3.2 Data sheet

Alice specifications	
Dimensions	22mm x 20mm x 19mm
Power supply resistor	56 kΩ
Power supply voltage	3.3 V
Velocity	29 to 51 mm/s
Power consumption	< 10mW
Infrared communication	20 cm to 6 m
Power autonomy	up to 10 hours

8.4 Diagram of organization of computer resources



User program (team manager):

(of course each user receives the current position of all objects on the field)

Different levels of commanding the robots:
- speed of each wheel of each robot (vl, vr)
- target position (x, y, theta)
- *behavior (goalkeeper, forward, ...)*

Different ways of implementing a soccer strategy:
- self-made program (C++, Visual Basic, Delphi, Pascal) using TCP/IP protocol
- *using some dedicated functions for example in Matlab, Logo, etc.*
- *using a dedicated program (with a special language or a graphic interface)*

Different configurations of computer resources:
- all programs on the same machine as ASM and COPS
- another computer for user programs
- one computer per team

8.5 Assembler code of adapted communication protocol

8.5.1 Alice

```
;     ********************************************************
;     *     Software for Microrobot Alice                    *
;     *     Control of motors (1 KOhm)                       *
;     *     Command from IR module                           *
;     *                                          *
;     *     Authors:  G. Caprari                             *
;     *     Processor: MICROCHIP 16F84                       *
;     *     Date: 4.8.1998                                   *
;     *                                          *
;     *     Modification:                                    *
;     *     981022      no "incf GlobCounter"; use Dark values    *
;     *     981027      no TmProgr; motor speed ++           *
;     *     990914      just IR module, speed control        *
;     *     991117      new communication protocol (P. Ramer)     *
;     *     991125      no parity bit (P.Ramer)              *
;     *     991201      36 bit communication protocol (P. Ramer)*
;     *                                          *
;     ********************************************************


          LIST  P=PIC16F84
     __config    B'11111111110001'
          ; power-un enabled, watchdog disabled, XT osc

;============-============================================================
;     CONFIGURATION
;=====================================================================

;=====================================================================
;     Address of Minirobot used by remote control
;=====================================================================
NR_ROBOT    equ   .3    ; number assigned to this robot
                        ; this robot use channel 1

;=====================================================================
;     INLUDES
;=====================================================================

     include "p16f84.inc"    ; Standard header file
     include "p84_cont.inc"  ; Other usefull headers
                        ; missing in "p16f84.inc"

;=====================================================================
;     CONSTANTS
;=====================================================================

INTCON_INIT equ   b'00000000' ; intcon register init value
OPTION_INIT equ   b'11001000' ; option register init value
TRISA_INIT  equ   b'00000000' ; data direction port A init
PORTA_INIT  equ   b'00011000' ; port A init value
TRISB_INIT  equ   b'00000001' ; data direction port B init
PORTB_INIT  equ   b'00000000' ; port B init value

INT_PERIOD  equ   .143        ; 143µs at 4.00 MHz
;INT_PERIOD equ   .125        ; 125µs at 4.00 MHz
;INT_PERIOD equ   .100        ; 100µs at 4.00 MHz

GLOB_C_INIT equ   .7          ; Global_Counter init value (5)
```

```
;---------------------- Motors ---------------------------------------

#define           Mright1           PORTB, 5
#define           Mright23    PORTA, 0
#define           Mright4           PORTB, 4
#define           Mleft1            PORTB, 2
#define           Mleft23           PORTB, 3
#define           Mleft4            PORTB, 1

;     Bits of MotorFlags0
#define           MOTOR_OFF_ON      MotorFlags0, 0    ; Robot is on a point
#define           MOTOR_DIR   MotorFlags0, 1    ; Movement direction
#define           MOTOR_RIGHT MotorFlags0, 2    ; Right direction
#define           MOTOR_LEFT  MotorFlags0, 3    ; Left direction
#define           MR_OFF_ON   MotorFlags0, 4    ; 1 if motor is turned on
#define           ML_OFF_ON   MotorFlags0, 5    ;
#define           MR_DIR            MotorFlags0, 6    ; motor direction
#define           ML_DIR            MotorFlags0, 7    ; 1 forward
;     Bits of MotorFlags1
#define           MR_STEPS    MotorFlags1, 0    ; motorR runs a nr of steps
#define           ML_STEPS    MotorFlags1, 1    ; motorL runs a nr of steps

MOTOR_INIT  equ   b'11110011' ; Motor ON


;---------------------- Side Sensors ---------------------------------
;#define          LED_R PORTB, 6    ; power for right led
;#define          LED_L PORTB, 7    ; power for left led


;---------------------- Communication --------------------------------
NR_PERIODS  equ   .14   ; period of a normal bit (10)
NR_BITS           equ   .6    ; nr of bit to receive for one robot
NR_ROBOTS   equ   .6    ; total number of robots

;     bits of CommFlags0
#define           RECEIVING   CommFlags0, 0     ;
#define           RECEIVED    CommFlags0, 1     ;
#define           ST_BIT1           CommFlags0, 2     ; receiving start bit
#define           STOP        CommFlags0, 3     ; robot is in stop situation
#define           PARITY            CommFlags0, 4     ; parity bit, not used
#define           PARAMETER   CommFlags0, 5     ; next command should be a
parameter
#define           PERIOD_R    CommFlags0, 6     ; last command was PERIOD_R
#define           PERIOD_L    CommFlags0, 7     ; last command was PERIOD_L
;     bits of CommFlags1
#define           STEPS_R           CommFlags1, 0     ; last command was
STEPS_R
#define           STEPS_L           CommFlags1, 1     ; last command was
STEPS_L
#define           PAUSE_R           CommFlags1, 2     ; last command was
PAUSE_R
#define           PAUSE_L           CommFlags1, 3     ; last command was
PAUSE_L


;=====================================================================
;     VARIABLES
;=====================================================================

      CBLOCK      VARS
```

```
      GlobCounter ; used for the mixer
      StatusStack ; stack for status reg. in Timer Int.
      WStack            ; stack for w reg. in Timer Int.
      Temp0       ; temporary variable

;---------------------- Motors ----------------------------------------
      MR_Duty
      MR_Phase
      MR_PhPeriod
      MR_Rotations
      MR_Pause
      ML_Duty
      ML_Phase
      ML_PhPeriod
      ML_Rotations
      ML_Pause
      MotorFlags0
      MotorFlags1

;---------------------- Communication ---------------------------------
      PeriodCounter           ; start impulse lenght or Bit
                        ; lenght (used for two jobs)
      BitCounter
      RobCounter        ; counts the robots when receiving speed
      Word
      CommFlags0
      CommFlags1
      ENDC

      if (CommFlags1>0x4F)
            error "RAM oweflow"
      endif


;#########################################################################
;                  PROGRAM
;#########################################################################

            org   RESET

            goto  Init              ; in case of reset goto
                                    ; initialisation

;=========================================================================
;                  INTERRUPT SERVICE
;=========================================================================

            org   INT

Interrupt   movwf WStack            ; save registers
            movf  STATUS, W
            movwf StatusStack

            btfss INTCON, INTE      ; INT enable bit
            goto  TimerInt
            btfsc INTCON, INTF      ; INT interrupt occurred
            goto  RB0Int

;============== Timer-Interrupt ======================================

TimerInt    bcf   INTCON, RTIF
            movlw (.258 - INT_PERIOD)     ; set pulse period
            addwf RTCC, F
```

```
            btfsc RECEIVING
            goto  Communication
            btfss ST_BIT1
            goto  TimerInt1
            incf  PeriodCounter, F  ; receiving start bit
            movlw (NR_PERIODS*2)
            subwf PeriodCounter,W        ; PeriodCounter-(NR_PERIOD*2)
            btfss STATUS, C
            goto  TimerInt1
      ; start error
            bcf   ST_BIT1
            bsf   STATUS, RP0
            bsf   OPTION_REG, INTEDG
            bcf   STATUS, RP0
            bcf   INTCON, INTF
            bsf   INTCON, INTE
            ;goto TimerInt1

TimerInt1   goto  MotorR

TmIntRest   decfsz      GlobCounter, F
            goto  Scheduler
            movlw GLOB_C_INIT
            movwf GlobCounter

Scheduler   movf  GlobCounter, W    ; here is decided which procedure
            addwf PCL, F            ; will be executed during this
                              ; cycle (just one pro cycle)
            nop
            goto  TmEnd       ;     7'
            goto  TmEnd       ;     6'
            goto  TmEnd       ;     5'
            goto  TmEnd       ;     4'
            goto  TmEnd       ;     3'
            goto  TmEnd       ;     2'
            goto  TmEnd       ;     1'


TmEnd       movf  StatusStack, W
            movwf STATUS
            movf  WStack, W
            retfie                  ; back from timer interrupt


;============== TABLES ======================================
; tables have to be in the first 256 Word of the program memory
;============================================================

; --- table for the received commands from the remote control
; --- completely modified for new communication protocol
SpeedRTable addwf PCL, F            ; jump to Speed
            goto  SpeedR0           ;
            goto  SpeedR1           ;
            goto  SpeedR2           ;
            goto  SpeedR3           ;

SpeedLTable addwf PCL, F            ; jump to Speed
            goto  SpeedL0           ;
            goto  SpeedL1           ;
            goto  SpeedL2           ;
            goto  SpeedL3           ;
```

```
;-------------------------------------------------------TABLES--


;============== Right Motor Control ====================================
; right motor
MotorR            decfsz      MR_Duty, F
            goto  NextMotor
            movf  MR_PhPeriod, W
            movwf MR_Duty

            btfss MR_OFF_ON
            goto  MR_stop
            btfss MR_DIR
            decf  MR_Phase, F
            btfsc MR_DIR
            incf  MR_Phase, F
            movf  MR_Phase, W
            andlw b'00000111' ; mask the phase bits
            addwf PCL, F            ; jump to phase
            goto  MR_P0
            goto  MR_P1
            goto  MR_P2
            goto  MR_P3
            goto  MR_P4
            goto  MR_P5
            goto  MR_P6
            goto  MR_P7
;------------------------------------------------------------ MotorR --

;============== Left Motor Control =====================================
NextMotor
; left motor
MotorL            decfsz      ML_Duty, F
            goto  TmIntRest
            movf  ML_PhPeriod, W
            movwf ML_Duty

            btfss ML_OFF_ON
            goto  ML_stop
            btfss ML_DIR
            decf  ML_Phase, F
            btfsc ML_DIR
            incf  ML_Phase, F
            movf  ML_Phase, W
            andlw b'00000111' ; mask the phase bits
            addwf PCL, F            ; jump to phase
            goto  ML_P0
            goto  ML_P1
            goto  ML_P2
            goto  ML_P3
            goto  ML_P4
            goto  ML_P5
            goto  ML_P6
            goto  ML_P7
;------------------------------------------------------------ MotorL --

;------------------
LastTable
      if (LastTable>0xFF)
            error "PCL oveflow in Tables"
      endif
;-------------------
```

```
            nop
            nop
            nop
            nop

;=================================================================
; right motor continuation

MR_stop           bcf   Mright1
            bcf   Mright4
            bcf   Mright23
            movlw 1
            movwf MR_Phase
            goto  NextMotor

MR_P0       movf  MR_Pause, W ; I'm turning backwards
            btfsc STATUS, Z
            goto  MR_P6
            movwf MR_Duty           ; MR_Duty:=MR_Pause
            movlw 7
            movwf MR_Phase    ; next P6
            bcf   Mright1
            bcf   Mright4
            bcf   Mright23
            goto  NextMotor

MR_P7       movf  MR_Pause, W ; I'm turning forwards
            btfsc STATUS, Z
            goto  MR_P1
            movwf MR_Duty           ; MR_Duty:=MR_Pause
            movlw 0
            movwf MR_Phase    ; next P1
            bcf   Mright1
            bcf   Mright4
            bcf   Mright23
            goto  NextMotor

MR_P1       bsf   Mright1
            bcf   Mright4
            bcf   Mright23
            movlw 1
            movwf MR_Phase
            btfss MR_STEPS    ; position controlled?
            goto  NextMotor
            decfsz      MR_Rotations, F
            goto  NextMotor
            bcf   MR_OFF_ON   ; 0 steps -> stop
            bcf   MR_STEPS
            goto  NextMotor

MR_P2       bsf   Mright1
            bcf   Mright4
            bsf   Mright23
            goto  NextMotor
MR_P3       bcf   Mright1
            bcf   Mright4
            bsf   Mright23
            goto  NextMotor
MR_P4       bcf   Mright1
            bsf   Mright4
            bsf   Mright23
            goto  NextMotor
```

```
MR_P5       bcf   Mright1
            bsf   Mright4
            bcf   Mright23
            goto  NextMotor
MR_P6       bsf   Mright1
            bsf   Mright4
            bcf   Mright23
            movlw 6
            movwf MR_Phase
            goto  NextMotor
;------------------------------------------------------------ MotorR --

;=================================================================
; left motor continuation

ML_stop           bcf   Mleft4
            bcf   Mleft1
            bcf   Mleft23
            movlw 1
            movwf ML_Phase
            goto  TmIntRest

ML_P0       movf  ML_Pause, W ; I'm turning backwards
            btfsc STATUS, Z
            goto  ML_P6
            movwf ML_Duty           ; ML_Duty:=ML_Pause
            movlw 7
            movwf ML_Phase    ; next P6
            bcf   Mleft4
            bcf   Mleft1
            bcf   Mleft23
            goto  TmIntRest

ML_P7       movf  ML_Pause, W ; I'm turning forwards
            btfsc STATUS, Z
            goto  ML_P1
            movwf ML_Duty           ; ML_Duty:=ML_Pause
            movlw 0
            movwf ML_Phase    ; next P1
            bcf   Mleft4
            bcf   Mleft1
            bcf   Mleft23
            goto  TmIntRest

ML_P1       bsf   Mleft4
            bcf   Mleft1
            bcf   Mleft23
            movlw 1
            movwf ML_Phase
            btfss ML_STEPS    ; position cotrolled?
            goto  TmIntRest
            decfsz      ML_Rotations, F
            goto  TmIntRest
            bcf   ML_OFF_ON   ; 0 steps -> stop
            bcf   ML_STEPS
            goto  TmIntRest

ML_P2       bsf   Mleft4
            bcf   Mleft1
            bsf   Mleft23
            goto  TmIntRest
ML_P3       bcf   Mleft4
            bcf   Mleft1
```

```
            bsf   Mleft23
            goto  TmIntRest
ML_P4       bcf   Mleft4
            bsf   Mleft1
            bsf   Mleft23
            goto  TmIntRest
ML_P5       bcf   Mleft4
            bsf   Mleft1
            bcf   Mleft23
            goto  TmIntRest
ML_P6       bsf   Mleft4
            bsf   Mleft1
            bcf   Mleft23
            movlw 6
            movwf ML_Phase
            goto  TmIntRest
;------------------------------------------------------------ MotorL --


;============== RB0-Interrupt =======================================

RB0Int            bcf   INTCON, INTF

            btfsc ST_BIT1
            goto  StartBit0

StartBit1   bsf   ST_BIT1                ; start bit just arrived
            bsf   STATUS, RP0       ; prepare to catch end of start bit
            bcf   OPTION_REG, INTEDG     ; interrupt on falling edge of RB0
            bcf   STATUS, RP0
            clrf  PeriodCounter
            goto  SetRestTr

StartBit0   bcf   ST_BIT1
            bsf   STATUS, RP0
            bsf   OPTION_REG, INTEDG     ; interrupt on rising edge of RB0
            bcf   STATUS, RP0
            movf  PeriodCounter, W
            sublw .2
            btfsc STATUS, C         ; test if is a false start bit
            goto  RetFromRB0I       ; 2-W pos (W <= 2)
            bsf   RECEIVING         ; 2-W neg (W > 2)
            bcf   INTCON, INTE           ; disable RB0 interrupt

            bcf   STATUS, C
            rrf   PeriodCounter, W  ; W=PeriodCounter/2
            sublw NR_PERIODS        ; W=NR_P - W
            addlw NR_PERIODS
            movwf PeriodCounter          ; Wait to center of next bit
            movlw NR_BITS
            movwf BitCounter        ; initialize BitCounter
            movlw .1
            movwf RobCounter        ; initialize Robot Counter
            ;goto SetRestTr

SetRestTr
RetFromRB0I movf  StatusStack, W
            movwf STATUS
            movf  WStack, W
            retfie
;------------------------------------------------------------ RB0-Interrupt
--
```

```
;============== Communication ======================================

Communication
            decfsz      PeriodCounter, F
            goto  TimerInt1

            movlw NR_ROBOT          ; number of this robot
            subwf RobCounter, W           ; = NR_ROBOT ?
            btfss STATUS, Z
            goto  NextBit

BitRcvd           bcf   STATUS, C
            btfsc PORTB, 0          ; read input and store in carry
            bsf   STATUS, C
            rrf   Word, F                 ; store received bit

NextBit           movlw NR_PERIODS
            movwf PeriodCounter
            decfsz      BitCounter, F
            goto  TimerInt1

NextRob           incf  RobCounter, F
            movlw NR_BITS
            movwf BitCounter        ; reinitialize BitCounter
            movlw .7                ; last rob
            subwf RobCounter, W           ; = 7 ?
            btfss STATUS, Z
            goto  TimerInt1

WordRcvd    bcf   RECEIVING         ; end of 36 bit transmission
            bsf   RECEIVED
            bcf   INTCON, INTF
            bsf   INTCON, INTE
            goto  TimerInt1

;-------------------------------------------------------Communication--


;=====================================================================
;                 INITIALISATION
;=====================================================================

Init        bsf   STATUS, RP0
            movlw TRISA_INIT
            movwf TRISA
            movlw TRISB_INIT
            movwf TRISB
            movlw OPTION_INIT
            movwf OPTION_REG
            bcf   STATUS, RP0

            movlw PORTB_INIT
            movwf PORTB
            movlw PORTA_INIT
            movwf PORTA

            movlw INTCON_INIT
            movwf INTCON


            clrf  CommFlags0
            clrf  CommFlags1
```

```
            movlw MOTOR_INIT
            movwf MotorFlags0       ; set default motor values
            clrf  MotorFlags1
            clrf  MR_Phase
            clrf  ML_Phase
            clrf  MR_Pause
            clrf  ML_Pause
            movlw .6
            movwf MR_Duty
            movwf ML_Duty

            movlw .6          ;£äü!£!£ää!££ä!£!ä£!££!£!ä
            movwf MR_PhPeriod
            movwf ML_PhPeriod

            movlw (GLOB_C_INIT+1)
            movwf GlobCounter
            clrf  RTCC              ; reset timer counter
            bsf   INTCON, RTIE            ; and enabled timer interrupt
            bsf   INTCON, INTE            ; and enabled hardware interrupt
            bsf   INTCON, GIE       ; enabled general interrupt


;=====================================================================
;           MAIN PROGRAM
;=====================================================================


Main
MainLoop    btfss RECEIVED
            goto  MainLoop1   ; Nothing new

            bcf   RECEIVED    ; command received
            bcf   STOP        ; new command received, dont sleep

;---------------right speed-----------------------------
RightSpeed  rrf   Word, F
            rrf   Word, F           ; right speed in bits 0-2, left in bits
3-5
            movf  Word, W           ; check speed parameter
            andlw b'00000011' ; mask right speed parameter bits
            goto  SpeedRTable

SpeedR0           bcf   MR_OFF_ON   ; motor off
            goto  LeftSpeed

SpeedR1           movlw .7
            movwf MR_PhPeriod
            movlw .11
            movwf MR_Pause
            goto  SpeedRDir

SpeedR2           movlw .7
            movwf MR_PhPeriod
            goto  SpeedRPause0

SpeedR3           movlw .5
            movwf MR_PhPeriod
            goto  SpeedRPause0

SpeedRPause0      movlw .0          ; Pause = 0
```

```
            movwf MR_Pause

SpeedRDir   bcf   MR_DIR
            btfss Word, 2           ; parameter 0 pos or 1 neg
            bsf   MR_DIR
            bsf   MR_OFF_ON   ; motor on


;---------------left speed-----------------------------
LeftSpeed   rrf   Word, F
            rrf   Word, F
            rrf   Word, F           ; left speed in bits 0-2
            movf  Word, W           ; check speed parameter
            andlw b'00000011' ; mask left speed parameter bits
            goto  SpeedLTable

SpeedL0           bcf   ML_OFF_ON   ; motor off
            goto  MainLoop1

SpeedL1           movlw .7
            movwf ML_PhPeriod
            movlw .11
            movwf ML_Pause
            goto  SpeedLDir

SpeedL2           movlw .7
            movwf ML_PhPeriod
            goto  SpeedLPause0

SpeedL3           movlw .5
            movwf ML_PhPeriod
            goto  SpeedLPause0

SpeedLPause0      movlw .0          ; Pause = 0
            movwf ML_Pause

SpeedLDir   bcf   ML_DIR
            btfss Word, 2           ; parameter 0 pos or 1 neg
            bsf   ML_DIR
            bsf   ML_OFF_ON   ; motor on

MainLoop1   goto  MainLoop

;*****************************************************************
            END
```

8.5.2 Remote control

```
;     ********************************************************
;     *     Software for Transmitter                 *
;     *     (Control of buttons and word             *
;     *      to transmitt )                          *
;     *                                        *
;     *     Authors:  G. Caprari and  V. Vuskovic          *
;     *     Processor: MICROCHIP 16F84               *
;     *     Date: 24.7.1996                          *
;     *     modification                             *
;     *     18.12.1996                          *
;     *     9.12.1997   RS232 comm                   *
;     *     20.04.1998   new comm 4+4+1 bit               *
;     *     G. Caprari 10.06.1998 faster comm        *
;     *     980806      osc 4 or 4.19 MHz            *
;     *     980806      new commands; no RS interruptor          *
;     *     981102      NR_RUN with SHIFT and FORWARD       *
;     *     991125      No parity bit (P. Ramer)          *
;     *     991201      36 bit communication protocol (P. Ramer)*
;     ********************************************************


          LIST  P=PIC16F84
      __config    B'11111111110001'
          ; power-un enabled, watchdog disabled, XT osc

;=====================================================================
;     CONFIGURATION
;=====================================================================
PIC_419           equ   0
PIC_400           equ   1

;     Address of Minirobot used by remote control pushing the buttons

NR_ROBOT    equ   .1    ; Address of robot in bits 6,5,4
                        ; of communication word
                        ; this remote controller use channel 1

;=====================================================================
=
;     INLUDES
;=====================================================================
=

      include "p16f84.inc"    ; Standard header file,
      include "p84_cont.inc"  ; Other usefull headers

;=====================================================================
=
;     CONSTANTS
;=====================================================================
=

INTCON_INIT equ   b'00100000' ; intcon register init value
OPTION_INIT equ   b'00001000' ; option register init value
TRISA_INIT  equ   b'00001000' ; data direction port A init value
TRISB_INIT  equ   b'11111111' ; data direction port B init value
PORTA_INIT  equ   b'00000100' ; port A init value
PORTB_INIT  equ   b'00000000' ; port B init value

NR_PERIODS  equ   .20   ;.40
```

```
NR_PULSES   equ   .14   ;.26
NR_BITS_START     equ   .10   ; number of bits of first word including start
sequence (10)
NR_BITS           equ   .8    ; number of bits per word
NR_BITS_1   equ   .9    ; NR_BITS + 1
NR_WORDS    equ   .5    ; speeds for all robs are packed into 5 words

;     bits of Flags
SENDING           equ   0
SENT        equ   1
;FROM_RS    equ   2
SHIFT       equ   3

      if PIC_419 == 1
PULSE_PERIOD      equ   .163  ; ca. 100µs at 4.19 MHz (164)
WAIT_ONE_BIT      equ   .32   ; 104 us for 9600 Baud at 4.19 MHz
WAIT_FIRST_BIT    equ   .15   ;  52 us for 9600 Baud at 4.19 MHz
      else
PULSE_PERIOD      equ   .168  ; ca. 100µs at 4.00 MHz
WAIT_ONE_BIT      equ   .31   ; 104 us for 9600 Baud at 4 MHz
WAIT_FIRST_BIT    equ   .14   ;  52 us for 9600 Baud at 4 MHz
      endif

;=============================================================================
=
;     IN/OUT
;=============================================================================
=
#define           RS_IN       PORTA, 3    ; old 1
#define           RS_OUT            PORTA, 2
#define           IR_LED            PORTA, 1    ; old 2 (in SendBit1!!)

;=============================================================================
=
;     VARIABLES
;=============================================================================
=
      CBLOCK      VARS
            PortB_Flags
            PeriodCounter           ; period of a bit
            BitCounter
            WordCounter       ; words needed for whole transmission
            Word
            Flags
            BitToSend         ; used bit 7

            Word1             ; Word number 1 to send
            Word2             ; Word number 2 to send
            Word3             ; Word number 3 to send
            Word4             ; Word number 4 to send
            Word5             ; Word number 5 to send, only bits 0-3 used

            W_STACK
            STATUS_STACK
            ImpulseCounter          ; number of impulse of 1 bit
            WaitCounter       ; used for waiting
            WaitCounter1

            RS_BitCount       ; RS232 bit counter
            InByte                  ; input byte from RS232

      ENDC
```

```
;*******************************************************************************
#
;     PROGRAM
;*******************************************************************************
#

            org   RESET

            goto  Init        ; in case of reset goto initialisation
            goto  Init

;*******************************************************************************
=
;     INTERRUPT SERVICE
;*******************************************************************************
=

            org   INT

Interrupt   btfss INTCON, RTIF            ; timer interrupt ?
            retfie
            btfss INTCON, RTIE
            retfie


;============== Timer-Interrupt
==========================================

Timer       movwf W_STACK           ; save registers
            movf  STATUS, W
            movwf STATUS_STACK

            movlw PULSE_PERIOD      ; set pulse period
            movwf RTCC

            btfss Flags, SENDING
            goto  RetFromTI

SendImp           btfss BitToSend, 7
            goto  SendBit0

SendBit1    movlw b'00000010'       ; change output port A1=IR_LED
            xorwf PORTA, F
            decfsz      ImpulseCounter, F
            goto  EndTransm
            movlw NR_PULSES
            movwf ImpulseCounter
            bcf   BitToSend, 7
            goto  EndTransm

SendBit0    bcf   IR_LED

EndTransm   decfsz      PeriodCounter, F
            goto  RetFromTI

            movlw NR_PERIODS
            movwf PeriodCounter

            movlw NR_BITS_START
            subwf BitCounter, W     ; = 10 ?
            btfsc STATUS, Z
            goto  SendStart
            movlw .1
```

```
            subwf BitCounter, W     ; = 1 ?
            btfss STATUS, Z
            goto  NextBit

WordEnd           movlw .0
            subwf WordCounter, W    ; = 0 ?
            btfsc STATUS, Z
            goto  SendEnd           ; end of 36 bit transmission
            decf  WordCounter, F
            movlw NR_BITS_1   ; number of bits of words 2-5, +1 to match
with dec
            movwf BitCounter

NextBit           movf  WordCounter, W    ; if a word is transmitted the
program
            goto  SendWordTable     ; continuous here to prepare next word

SendWord1   rrf   Word1, F    ; move bit to send in a variable
            goto  SendWord
SendWord2   rrf   Word2, F    ; move bit to send in a variable
            goto  SendWord
SendWord3   rrf   Word3, F    ; move bit to send in a variable
            goto  SendWord
SendWord4   rrf   Word4, F    ; move bit to send in a variable
            goto  SendWord
SendWord5   movlw .5          ; only first 4 bits are used (8-5)
            subwf BitCounter, W     ; = 5 ?
            btfsc STATUS, Z
            goto  SendEnd           ; end of 36 bit transmission
            rrf   Word5, F    ; move bit to send in a variable
            ;goto SendWord

SendWord    rrf   BitToSend, F
            decf  BitCounter, F
            goto  RetFromTI

SendStart   ; sending starting sequence (10)
            bcf   BitToSend, 7
            decf  BitCounter, F
            goto  RetFromTI


SendEnd           bcf   Flags, SENDING    ; end of sending
            movlw NR_WORDS
            movwf WordCounter ; reinitialization for receiving

RetFromTI   movf  STATUS_STACK, W
            movwf STATUS
            movf  W_STACK, W
            bcf   INTCON, RTIF
            retfie                  ; back from timer interrupt

;============== TABLES ======================================
; tables have to be in the first 256 Word of the program memory
;============================================================

; --- table for the received words from rs
RecWordTable      addwf PCL, F            ; jump to Word to record
            goto  RecWord5    ;
            goto  RecWord4    ;
            goto  RecWord3    ;
            goto  RecWord2    ;
            goto  RecWord1    ;
```

```
; --- table for words to send
SendWordTable     addwf PCL, F            ; jump to Word to send
            goto  SendWord5   ;
            goto  SendWord4   ;
            goto  SendWord3   ;
            goto  SendWord2   ;
            goto  SendWord1   ;

;-----------------------------------------------------TABLES--


;=====================================================================
=
;     INITIALISATION
;=====================================================================
=

Init        clrf  PORTA
            bsf   STATUS, RP0
            movlw TRISA_INIT
            movwf TRISA
            movlw TRISB_INIT
            movwf TRISB
            movlw OPTION_INIT
            movwf OPTION_REG
            bcf   STATUS, RP0

            movlw PORTB_INIT
            movwf PORTB
            movlw PORTA_INIT
            movwf PORTA

            movlw INTCON_INIT
            movwf INTCON

            bcf   IR_LED
            clrf  Flags
            clrf  PortB_Flags
            movlw PULSE_PERIOD            ; set pulse period
            movwf RTCC
            bsf   INTCON, RTIE
            bsf   INTCON, GIE

            movlw NR_WORDS
            movwf WordCounter


;=====================================================================
=
;     MAIN PROGRAM
;=====================================================================
=

Main        btfsc Flags, SENDING
            goto  Main

            bcf   INTCON, GIE ; no interrupt when reading buttons or RS

            btfsc RS_IN
            goto  Main

SerialRec   btfsc RS_IN ; check start bit from input pin
```

```
            goto  Main

            movlw WAIT_FIRST_BIT
            call    Wait
            movlw   .8
            movwf   RS_BitCount

ReadRx            movlw WAIT_ONE_BIT
            call    Wait
            bcf   STATUS, C   ; copy RS_IN in Carry bit
            btfsc RS_IN
            bsf   STATUS, C
            rrf   InByte, F
            decfsz      RS_BitCount, F
            goto  ReadRx

            movlw WAIT_ONE_BIT      ; wait for end of last bit
            call    Wait

            decf  WordCounter, F    ; count the received words
            movf  WordCounter, W
            goto  RecWordTable      ;

RecWord1    movf  InByte, W   ; moves the received byte to the word variable
            movwf Word1
            goto  Main
RecWord2    movf  InByte, W   ; moves the received byte to the word variable
            movwf Word2
            goto  Main
RecWord3    movf  InByte, W   ; moves the received byte to the word variable
            movwf Word3
            goto  Main
RecWord4    movf  InByte, W   ; moves the received byte to the word variable
            movwf Word4
            goto  Main
RecWord5    movf  InByte, W   ; moves the received byte to the word variable
            movwf Word5

SendIt            bcf   Flags, SHIFT
            movlw NR_WORDS
            movwf WordCounter ; reinitalize for sending
            decf  WordCounter, F    ; WordCounter=4 to match with
SendWordTable
            movlw NR_BITS_START     ; first word + start sequence
            movwf BitCounter
            movlw NR_PERIODS
            movwf PeriodCounter
            movlw NR_PULSES
            movwf ImpulseCounter
            bsf   BitToSend, 7
            bsf   Flags, SENDING
            bsf   INTCON, GIE
            goto  Main


;-------------------------------------------------------------------
; Wait: Wait for specified time loaded into W register.
;     Every loop takes 2.86 us @ 4.19 MHz.
;     Every loop takes 3 us @ 4 MHz.
;-------------------------------------------------------------------
Wait        movwf WaitCounter
Wait1       decfsz      WaitCounter, F    ; Wait for specified Time
            goto  Wait1
```

```
            return

;-----------------------------------------------------------------
; WaitLong : Wait for specified time loaded into W register.
;     Every loop takes ... us @ 4.19MHz.
;-----------------------------------------------------------------
WaitLong    movwf WaitCounter
            clrf  WaitCounter1
WaitLong1   nop
            decfsz      WaitCounter1, F
            goto  WaitLong1
            decfsz      WaitCounter, F
            goto  WaitLong1
            return

;
*********************************************************************************
            END
```

8.6 Delphi code of adaptation of COPS

8.6.1 GUSB.pas (Class for USB video camera grabber)

Note that the code below doesn't contain the TGUSBForm class.

```
{*************************************************************************

  GUSB.PAS   Grabber control for USB video camera.

  Author:   Rémy Blank / Jean-Christophe Zufferey
  Date:     21.09.99
  Purpose:  Provides a Video for Windows control class for an USB video camera.
  Program:  Delphi V3.02

  Classes:  TGUSB      Control class for USB frame grabber
            TGUSBForm  Associated settings form

  Remarks:  Based on Video for Windows
            Implemented for "Alice Footballeur"
            USES VfW.pas

  History:  adapted from TGPCX200.PAS

*************************************************************************}

unit GUSB;

interface

uses
       Messages, SysUtils, Forms,

  Windows, ExtCtrls, Graphics, Classes, Dialogs, Controls, StdCtrls,
  FObject, Grabber, ComCtrls, VfW;


{*****   TGUSB class   ************************************************}
type
  TGUSB=class(TGrabber)
  private
    FhWndCap : HWND;                    // Handle of capture window

    FiDrvIndex : integer;               // Index of connected driver

                                        // (-1 = no driver connected)
    procedure GetCaptureFormat;
    procedure SetCaptureFormat;

  protected
                                        // Capture window allocation
    procedure AllocCapWnd
    procedure FreeCapWnd;

                                        // Capture driver connection
    procedure ConnectDrv;
    procedure DisconnectDrv;

                                        // Setting video callback function
    procedure SetVideoCallback;
    procedure FreeVideoCallback;
                                        // Open the dialog box "Video Source"
    procedure DlgVideoSource;
                                        // Property methods
    function GetCurVideoStd:TVideoStd; override;
    procedure SetImageFormat(const Val:TImageFormat); override;
    procedure SetXResolution(const Val:integer); override;
    procedure SetYResolution(const Val:integer); override;
```

```
    procedure SetLeft(const Val:integer); override;
    procedure SetTop(const Val:integer); override;
    procedure SetWidth(const Val:integer); override;
    procedure SetHeight(const Val:integer); override;
    function GetInputNum:integer; override;
    procedure SetInputNum(const Val:integer); override;
    function GetInputType:TInputType; override;
    procedure SetInputType(const Val:TInputType); override;
    procedure SetVideoStd(const Val:TVideoStd); override;
    function GetBrightness:integer; override;
    procedure SetBrightness(const Val:integer); override;
    function GetContrast:integer; override;
    procedure SetContrast(const Val:integer); override;
    function GetSaturation:integer; override;
    procedure SetSaturation(const Val:integer); override;
    function GetHue:integer; override;
    procedure SetHue(const Val:integer); override;


  public
    property hWndCap : HWND read FhWndCap;

                                        // Maintenance
    constructor Create; override;
    destructor Destroy; override;
                                        // Get description of class
    class function ClassDescription:string; override;
                                        // Start continuous grab
    procedure Start; override;
                                        // Stop continuous grab
    procedure Stop; override;
                                        // Perform single grab
    procedure Single; override;


  published

  end;

                                          // VfW callback
function VideoCallback(hWnd: HWND; lpVHdr: LPVIDEOHDR): LRESULT; stdcall;


{*****   TGUSBForm class   ******************************************}
type
  TGUSBForm=class(TGrabberForm)


  public
                                        // Copy properties to/from form
    procedure FObjectToForm; override;
  end;

implementation

{$R *.DFM}

{*****************************************************************************

       Class:    TGUSB
       Ancestor:    TGrabber

       Purpose:     Control class for USB video camera.

*****************************************************************************}

{*****   Maintenance   *****}

constructor TGUSB.Create;
begin
```

```
  Inherited Create;

  FormClass:=TGUSBForm;                 // Set associated form class

  FGrabMode:=grmFrames;
  FVideoStd:=vsAutoDetect;
  FImageFormat:=ifRGB15;
  FiDrvIndex := -1;
       // No driver connected
  AllocCapWnd;                          // Create capture window
  ConnectDrv;

  // Store a pointer on TGUSB class in order to retrieve
  // private property in callback function
  SendMessage(FhWndCap, WM_CAP_SET_USER_DATA, 0, integer(Self));

  // Retrieve current values of image width and height
  GetCaptureFormat;

  FXResolution := FWidth;              // No difference between X-YResoulution
  FYResolution := FHeight;             // and image width and height

  FLeft := 0;
  FTop := 0;

  FInputNum := 0;

  SetVideoCallback;
       // Set video callback function
end;

destructor TGUSB.Destroy;
begin
  Stop;
  FreeVideoCallback;
       DisconnectDrv;
  FreeCapWnd;                           // Free capture window's handle
  Inherited Destroy;
end;


{*****   Get description of class   *****}

class function TGUSB.ClassDescription:string;
begin
  Result:='USB frame grabber';
end;


{*****   Capture window allocation   *****}

procedure TGUSB.AllocCapWnd;
begin
       // Create an invisible capture window
       FhWndCap := capCreateCaptureWindowA(NIL,  // No name
                                           0,    // No style (default to invisible)
                                           0, 0, 150, 150, // An arbitrary size
                                           0,    // No parent
                                           0);   // Don't care about the ID
end;

procedure TGUSB.FreeCapWnd;
begin
       // Release the capture window's handle
       DestroyWindow(FhWndCap);
end;

{*****   Capture driver connection   *****}

procedure TGUSB.ConnectDrv;
```

```
var
  i : integer;
  CapParms : CAPTUREPARMS;
begin
      i := 0;
      while (SendMessage(FhWndCap, WM_CAP_DRIVER_CONNECT, i, 0) = 0)
      and (i<=MAXVIDDRIVERS) do
  begin
      i := i+1;
  end;
      FiDrvIndex := i;

      // Settings of video capture sequence :
      SendMessage(FhWndCap, WM_CAP_GET_SEQUENCE_SETUP,
                                        sizeof(CAPTUREPARMS), integer(@CapParms));
  CapParms.fMakeUserHitOKToCapture := false;
  CapParms.fYield := true;
  CapParms.wNumVideoRequested := 1;
  CapParms.fCaptureAudio := false;
  CapParms.vKeyAbort := 0;
  CapParms.fAbortLeftMouse := false;
  CapParms.fAbortRightMouse := false;
  CapParms.fLimitEnabled := false;
  CapParms.fMCIControl := false;
  SendMessage(FhWndCap, WM_CAP_SET_SEQUENCE_SETUP,
              sizeof(CAPTUREPARMS), integer(@CapParms));
end;

procedure TGUSB.DisconnectDrv;
begin
      if FiDrvIndex <> -1 then  // Only if a driver is already connected
  begin
      SendMessage(FhWndCap, WM_CAP_STOP, 0, 0);
      Sleep(500);
      // WM_CAP_STOP takes a few millisecs...
      if SendMessage(FhWndCap, WM_CAP_DRIVER_DISCONNECT, FiDrvIndex, 0) <> 0 then
      FiDrvIndex := -1; // No driver connected
  end;
end;


{*****   Setting video callback function   *****}

procedure TGUSB.SetVideoCallback;
begin
  SendMessage(FhWndCap, WM_CAP_SET_CALLBACK_VIDEOSTREAM, 0,
                                        integer(@VideoCallback));
end;

procedure TGUSB.FreeVideoCallback;
begin
  SendMessage(FhWndCap, WM_CAP_SET_CALLBACK_VIDEOSTREAM, 0, 0);
//  FreeProcInstance(@TGUSB.VideoCallback);
end;


{*****   Video callback function   *****}

function VideoCallback(hWnd: HWND; lpVHdr: LPVIDEOHDR): LRESULT; stdcall;
var
      Image : TImageInfo;
  Me : TGUSB;
begin
  // Retrieve the pointer on TGUSB class stored in USER_DATA of capture window
  Me:=TGUSB(SendMessage(hWnd, WM_CAP_GET_USER_DATA, 0, 0));
  if not Me.IsRunning then
  begin
    result := integer(true);
    Exit;
  end;
```

```
  // Set image settings
  Image.Width := Me.FWidth;
  Image.Height := Me.FHeight;
  Image.BPP := 2;                       // Byte per pixel
  Image.ScanLineLen := ((Me.FWidth*Image.BPP)+3) and not 3;
  Image.Size := Image.ScanLineLen*Image.Height;
  Image.Format := Me.FImageFormat;
  Image.FieldType := ftFrame;
  Image.PalettePtr := nil;
  Image.Buffer := lpVHdr^.lpData;
  Image.hBMP := 0;                     // Not available with VfW
  Image.pBI := nil;                    // Not available with VfW

  Me.DoNewImage(Image);

  result := integer(true);
end;

{*****   Capture format   *****}

procedure TGUSB.GetCaptureFormat;
var
       biInfo : TBITMAPINFO;
begin
       SendMessage(FhWndCap, WM_CAP_GET_VIDEOFORMAT, sizeof(TBITMAPINFO),
integer(@biInfo));

  FWidth := biInfo.bmiHeader.biWidth;
  FHeight := biInfo.bmiHeader.biHeight;
end;

procedure TGUSB.SetCaptureFormat;
var
       biInfo : TBITMAPINFO;
begin
       SendMessage(FhWndCap, WM_CAP_GET_VIDEOFORMAT, sizeof(TBITMAPINFO),
integer(@biInfo));
  biInfo.bmiHeader.biSize := 40;
  biInfo.bmiHeader.biWidth := FWidth;
  biInfo.bmiHeader.biHeight := FHeight;
  biInfo.bmiHeader.biPlanes := 1;
  case FImageFormat of
    ifY8:
      biInfo.bmiHeader.biBitCount := 8;

    ifRGB15:
      biInfo.bmiHeader.biBitCount := 16;

    ifRGB16:
      biInfo.bmiHeader.biBitCount := 16;

    ifRGB24:
      biInfo.bmiHeader.biBitCount := 24;

    ifRGB32:
      biInfo.bmiHeader.biBitCount := 32;

    else
      biInfo.bmiHeader.biBitCount := 0;
  end;

  biInfo.bmiHeader.biCompression := 0;
  biInfo.bmiHeader.biSizeImage := FWidth*FHeight*
                                  (biInfo.bmiHeader.biBitCount DIV 8);
  biInfo.bmiHeader.biXPelsPerMeter := 0;
  biInfo.bmiHeader.biYPelsPerMeter := 0;
  biInfo.bmiHeader.biClrUsed := 0;
  biInfo.bmiHeader.biClrImportant := 0;
  SendMessage(FhWndCap, WM_CAP_SET_VIDEOFORMAT, sizeof(TBITMAPINFO),
```

```
              integer(@biInfo));

end;


{*****   Open the dialog box "Video Source", if it exists    *****}

procedure TGUSB.DlgVideoSource;
begin
  PostMessage(FhWndCap, WM_CAP_DLG_VIDEOSOURCE, 0, 0);
end;

{*****   Start continuous grab   *****}

procedure TGUSB.Start;
begin
  if not FIsRunning then
  begin
    Inherited Start;
    // Start capture
    SendMessage(FhWndCap, WM_CAP_SEQUENCE_NOFILE, 0, 0);
    DlgVideoSource;
  end;
end;

{*****   Stop continuous grab   *****}

procedure TGUSB.Stop;
begin
  if FIsRunning then
  begin
    Inherited Stop;
    SendMessage(FhWndCap, WM_CAP_STOP, 0, 0);
  end;
end;

{*****   Perform single grab   *****}

procedure TGUSB.Single;

begin
       // Not necessary at the moment
end;

{*****   Property methods   *****}

function TGUSB.GetCurVideoStd:TVideoStd;
begin
       Result:=vsAutoDetect;
end;


procedure TGUSB.SetImageFormat(const Val:TImageFormat);
begin
  if Val<>FImageFormat then
  begin
    Stop;
    Inherited SetImageFormat(Val);      // Set stored value
    SetCaptureFormat;
  end;
end;


procedure TGUSB.SetXResolution(const Val:integer);
begin
  if Val<>FXResolution then
    begin
      Inherited SetXResolution(Val);
      SetWidth(Val);                    // XResolution = Width for USB cam
    end
```

```
end;


procedure TGUSB.SetYResolution(const Val:integer);
begin
  if Val<>FYResolution then
    begin
      Inherited SetYResolution(Val);
      SetHeight(Val);                    // YResolution = Height for USB cam
    end
end;

procedure TGUSB.SetLeft(const Val:integer);
begin
  if Val<>FLeft then
    Inherited SetLeft(0);
end;

procedure TGUSB.SetTop(const Val:integer);
begin
  if Val<>FTop then
    Inherited SetTop(0);
end;

procedure TGUSB.SetWidth(const Val:integer);
begin
  if Val<>FWidth then
    begin
      Stop;
      Inherited SetWidth(Val);
      SetCaptureFormat;
    end;
end;

procedure TGUSB.SetHeight(const Val:integer);
begin
  if Val<>FHeight then
    begin
      Stop;
      Inherited SetHeight(Val);
      SetCaptureFormat;
    end;
end;

function TGUSB.GetInputNum:integer;
begin
  Result:=0;
end;

procedure TGUSB.SetInputNum(const Val:integer);
begin
end;

function TGUSB.GetInputType:TInputType;
begin
       Result:=itComposite;
end;

procedure TGUSB.SetInputType(const Val:TInputType);
begin
end;

procedure TGUSB.SetVideoStd(const Val:TVideoStd);
begin
end;

function TGUSB.GetBrightness:integer;
begin
  Result:=0;
end;
```

```
procedure TGUSB.SetBrightness(const Val:integer);
begin
end;

function TGUSB.GetContrast:integer;
begin
  Result:=0;
end;

procedure TGUSB.SetContrast(const Val:integer);
begin
end;

function TGUSB.GetSaturation:integer;
begin
  Result:=0;
end;

procedure TGUSB.SetSaturation(const Val:integer);
begin
end;

function TGUSB.GetHue:integer;
begin
  Result:=0;
end;

procedure TGUSB.SetHue(const Val:integer);
begin
end;
```

8.6.2 Tracking module

The following functions comes from TmainForm class:

```
{*****   Event: Track blob ID   *****}
// This procedure as been adapted for "Alice Footballeur". (jcz/15.11.99)

procedure TMainForm.TrackBlobID(Sender:TBlobServer; var BlobList:TBlobList;
                                PrevBlobList:TBlobList; bReset:boolean);
const
  nRobPerTeam=3;

var
  i,IDBall:integer;
  Dummy:TBlob;
  nBlobs:integer;

begin
  FillChar(Dummy, SizeOf(Dummy), 0);    // Make dummy blob

  BlobList.SortEv(SortEventColor);      // Sort list by color

  for i:=0 to BlobList.Count-1 do       // Tag=-2 => object found
    BlobList[i].Tag:=-2;

// First team (color=1, IDs=0,1,2)
  nBlobs:=BlobList.ColUsage[1];
  if nBlobs<nRobPerTeam then            // Not enough robots for current team ?
  begin
    for i:=0 to (nRobPerTeam-nBlobs)-1 do
    begin
      Dummy.Color:=1;
      BlobList.Insert(i, Dummy);        // Insert dummy blob
      BlobList[i].Tag:=-1;              // Tag=-1 => dummy blob
    end;
  end
  else if nBlobs>nRobPerTeam then       // Too much robots for current team ?
  begin

  end;

// Second team (color=2, IDs=3,4,5)
  nBlobs:=BlobList.ColUsage[2];
  if nBlobs<nRobPerTeam then            // Not enough robots for current team ?
  begin
    for i:=nRobPerTeam to (2*nRobPerTeam-nblobs)-1 do
    begin
      Dummy.Color:=2;
      BlobList.Insert(i, Dummy);        // Insert dummy blob
      BlobList[i].Tag:=-1;              // Tag=-1 => dummy blob
    end;
  end
  else if nBlobs>nRobPerTeam then       // Too much robots for current team ?
  begin

  end;

// The ball (color=3, ID=6)
  IDBall:=2*nRobPerTeam;
  nBlobs:=BlobList.ColUsage[3];
  if nBlobs=0 then                      // No ball ?
  begin
    Dummy.Color:=3;                     // Set dummy blob color
    if Assigned(PrevBlobList) then
    begin
      if PrevBlobList.Tag=1 then        // Found in previous list
      begin
        BlobList.Insert(IDBall, PrevBlobList[IDBall]^);
```

```
        BlobList[i].Tag:=-1             // Tag=-1 => dummy blob
      end
      else
        BlobList.Insert(IDBall, Dummy); // Not found, insert dummy blob
        BlobList[i].Tag:=-1             // Tag=-1 => dummy blob
    end
    else
      BlobList.Insert(IDBall, Dummy);   // List not valid, insert dummy blob
      BlobList[i].Tag:=-1;              // Tag=-1 => dummy blob
  end
  else if nBlobs>1 then                 // More than one ball ?
  begin

  end;

  if Assigned(PrevBlobList) then
  begin
    if bReset then                      // Reset to initial position
    begin
      BlobList.SortEv(SortEventPos);    // Sort by Y
    end
    else
    begin
      SetTag(BlobList, PrevBlobList);   // Set tag = ID (jcz/17.11.99)
      BlobList.SortEv(SortEventTag);    // Sort list by tag
    end
  end;

  BlobList.Tag:=1;                      // Flag for "structured list"
end;

// This procedure as been added for "Alice Footballeur". (jcz/15.11.99)
procedure TmainForm.SetTag(var BlobList:TBlobList; PrevBlobList:TBlobList);

const
  nRobPerTeam=3;

var
  o,n:integer;
  dist,odist:double;

begin

  for n:=0 to nRobPerTeam-1 do          // First team (color=1, IDs=0,1,2)
  begin
    odist:=1000000;
    for o:=0 to nRobPerTeam-1 do
    begin
      dist:=Sqrt(Sqr(PrevBlobList[o].pX-BlobList[n].pX)+
            Sqr(PrevBlobList[o].pY-BlobList[n].pY));
      if dist<odist then
      begin
        BlobList[n].Tag:=o;
        odist:=dist;
      end;
    end;
  end;

  for n:=nRobPerTeam to 2*nRobPerTeam-1 do  // Second team (color=2, IDs=3,4,5)
  begin
    odist:=1000000;
    for o:=nRobPerTeam to 2*nRobPerTeam-1 do
    begin
      dist:=Sqrt(Sqr(PrevBlobList[o].pX-BlobList[n].pX)+
            Sqr(PrevBlobList[o].pY-BlobList[n].pY));
      if dist<odist then
      begin
        BlobList[n].Tag:=o;
        odist:=dist;
      end;
```

```
    end;
  end;

  BlobList.Items[2*nRobPerTeam].Tag:=2*nRobPerTeam; // Ball

end;

{*****   Event: Sort blob list   *****}
// This procedures have been adapted for "Alice Footballeur". (jcz/15.11.99)

function TMainForm.SortEventColor(const Blob1, Blob2:TBlob):integer;
begin
  if Blob1.Color<Blob2.Color then
    Result:=-1
  else if Blob1.Color>Blob2.Color then
    Result:=1
  else
    Result:=0;
end;

function TMainForm.SortEventTag(const Blob1, Blob2:TBlob):integer;
begin
  if Blob1.Tag<Blob2.Tag then
    Result:=-1
  else if Blob1.Tag>Blob2.Tag then
    Result:=1
  else
    Result:=0;
end;

function TMainForm.SortEventPos(const Blob1, Blob2:TBlob):integer;
begin
  if Blob1.Color<Blob2.Color then
    Result:=-1
  else if Blob1.Color>Blob2.Color then
    Result:=1
  else if Blob1.Y<Blob2.Y then
    if Blob1.Color<2 then Result:=-1
    else Result:=1
  else if Blob1.Y>Blob2.Y then
    if Blob1.Color<2 then Result:=1
    else Result:=-1
  else
    Result:=0;
end;
```

The following function comes from TblobServer class:

```
{*****   Track blob angle   *****}

procedure TBlobServer.TrackBlobAngle(bReset:boolean);
var
  i:integer;

begin
  if BlobList.Count=OldBlobList.Count then
  begin
    for i:=0 to BlobList.Count-1 do
      with BlobList[i]^ do
      begin
// ***************** Added by jcz for Alice Soccer
        if bReset then
        begin
         if (i<3) and (Abs(pTheta)>Pi/2) then
          begin
            Theta:=ModuloPi(Theta-Pi);
            pTheta:=ModuloPi(pTheta-Pi);
          end;
         if (i>=3) and (Abs(pTheta)<Pi/2) then
          begin
```

```
            Theta:=ModuloPi(Theta-Pi);
            pTheta:=ModuloPi(pTheta-Pi);
          end
        end
// *************** End
        else if Abs(ModuloPi(pTheta-OldBlobList[i]^.pTheta))>Pi/2 then
        begin
          Theta:=ModuloPi(Theta-Pi);
          pTheta:=ModuloPi(pTheta-Pi);
        end;

      end;
  end;
end;
```

8.7 C++ code of ASM

8.7.1 AliceCom file

AliceCom.h

```
//---------------------------------------------------------------------------
#ifndef AliceComH
#define AliceComH

#include <windows.h>    // includes basic windows functionality
#include <math.h>
#include <stdio.h>
#include <conio.h>

#include "Constants.h"

//---------------------------------------------------------------------------
// General constants
//---------------------------------------------------------------------------

// Speed definitions
#define SPEED0 0
#define SPEED1 1
#define SPEED2 2
#define SPEED3 3
#define SPEED_MIN SPEED1

// Motion control constants
#define K_RHO 1      // K_RHO > 0
#define K_ALPHA 6    // K_ALPHA > K_RHO - K_PHI
#define K_PHI -0.5     // K_PHI < 0
#define RHO_THRESHOLD 10 // distance in mm at which goal position is achieved
#define PHI_THRESHOLD 0.4 // angle in rad at which goal position is achieved
#define V_THRESHOLD 20     // velocity can't be higher than sum of V_THRESHOLD
                           // and VELOCITY_MAX

// Physical constants
#define WHEEL_AXLE 9 // half the length of the wheel axle of Alice (in mm)
#define VELOCITY1 3 // threshold below which velocity is zero
#define VELOCITY2 29
//#define VELOCITY2 37
#define VELOCITY3 51
#define VELOCITY_MAX VELOCITY3


//---------------------------------------------------------------------------
// TSendAlice class
//---------------------------------------------------------------------------

class TSendAlice : public TObject
{
public:
       __fastcall TSendAlice();
       virtual __fastcall ~TSendAlice();
       void __fastcall SendByte(BYTE byte_to_send);

private:
       HANDLE handle;
       HANDLE __fastcall OpenCOM1();
       void __fastcall CloseCOM1(HANDLE handle);
};


//---------------------------------------------------------------------------
// TAliceCtrl class
//---------------------------------------------------------------------------
```

```
class TAliceCtrl : public TObject
{
public:
   // Methods
       __fastcall TAliceCtrl();
       virtual __fastcall ~TAliceCtrl();
   void __fastcall SetCurrentPosition(int nrRob, int x, int y, double theta);
   void __fastcall SetTarget(int nrRob, int mode, int flag, int x, int y, double
theta);


   //void __fastcall ClearPositionFlag(int nrRob);
   //bool __fastcall GetPositionFlag(int nrRob);
   void __fastcall SendSpeed();


   // Debug methods
   void __fastcall GetDebugVar(int *dx, int *dy,
      double *vr, double *v, double *omega, double *rho, double *alpha, double
*phi, double *arctangent,
      double *VelocityRight, double *VelocityLeft,
      int *x0, int *y0, double *theta0, int *x, int *y, double *theta,
      int *SpeedRight, int *SpeedLeft,
      int *PreSpeedRight, int *PreSpeedLeft,
      int *TestPassing);

   // declared public only for debugging
   void __fastcall StopRob();  // to be used in MotionCtrlForm

private:
   // Variables
   struct RobotPositions
   {
      int Mode;
      int x, y, x0, y0;
      double theta, theta0;
      int SpeedR, SpeedL;
      bool BackwardFlag;    // defines if robot arrives backwards at target
position
   } Robot[NUMBER_ROBOTS];

   //bool PositionFlag[NUMBER_ROBOTS]; // = true if goal position achieved

   struct History
   {
      int PreviousSpeedR, PreviousSpeedL;
   } CommandHistory[NUMBER_ROBOTS];

   TSendAlice* SendAlice;
// add this again later
/*
   // Variable Constants
   double K_PHI;
   double K_ALPHA;
   double K_RHO;
   double RHO_THRESHOLD;
   double PHI_THRESHOLD;
*/
   //Debug variables
   struct DebugVar
   {
      int dx, dy;
      double vr, v, omega, rho, alpha, phi, arctangent;
      double VelocityRight, VelocityLeft;
      int x0, y0, x, y;
      double theta0, theta;
      int SpeedRight, SpeedLeft;
      int PreSpeedRight, PreSpeedLeft;
      int TestPassing;
   } DVar;
```

```
   //Methods
   void __fastcall InitHistory();
   void __fastcall SetGoalPosition(int nrRob, int flag, int x, int y, double
theta);
   void __fastcall SetRequestedSpeed(int nrRob, int SpeedR, int SpeedL);
   void __fastcall GetRequestedSpeed(int nrRob, int *SpeedR, int *SpeedL);
   void __fastcall SetPreviousSpeed(int nrRob, int SpeedR, int SpeedL);
   void __fastcall GetPreviousSpeed(int nrRob, int *SpeedR, int *SpeedL);
   //void __fastcall SetPositionFlag(int nrRob);
   void __fastcall ReadConstants();
   void __fastcall Velocity(RobotPositions *Rob, int *SpeedRight, int *SpeedLeft,
int nrRob);
   void __fastcall GetSpeed(double VelocityR, double VelocityL, double omega,
                            int *SpeedR, int *SpeedL);
   int __fastcall QuantizeVelocity(int velocity);
   int __fastcall Sign(int value);

   //Debug methods
   void __fastcall SetDebugVar(int dx, int dy,
      double vr, double v, double omega, double rho, double alpha, double phi,
double arctangent,
      double VelocityRight, double VelocityLeft,
      int x0, int y0, double theta0, int x, int y, double theta,
      int SpeedRight, int SpeedLeft,
      int TestPassing);
   void __fastcall SetMoreDebugVar(int PreSpeedRight, int PreSpeedLeft);
};

//---------------------------------------------------------------------------
#endif
```

AliceCom.cpp

```
/*
   AliceCom.cpp

       Author:        Patrick Ramer
       Date:          21.09.1999
       Abstract:    Manages Serial Port
            Provides the Speed and sends them to Alice

  Classes:  TSendAlice
            TAliceCtrl

       Software:    Borland C++ Builder 4
       Modifications:  991811 New Communication Protocol (foot1.asm)


       Conclusions:
       Errors:

       Version: 0.0
       Used files:
       Organization: ASL EPFL
*/


//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "AliceCom.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)

// Extern variables
extern TMultiReadExclusiveWriteSynchronizer* Sync;
```

```
// add this again later, activate ReadConstants in SendSpeed

/*
// Constants in Shared memory
extern double KPhiShared;
extern double KAlphaShared;
extern double KRhoShared;
extern double PhiThresholdShared;
extern double RhoThresholdShared;
*/

//---------------------------------------------------------------------------
// TSendAlice class
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Construction/Destruction

__fastcall TSendAlice::TSendAlice()
{
      handle = OpenCOM1();
}

__fastcall TSendAlice::~TSendAlice()
{
      CloseCOM1(handle);
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Open Serial Port
HANDLE __fastcall TSendAlice::OpenCOM1()
{
   HANDLE    handle;
   DCB       com1DCB;
   LPDCB     ptrDCB;
   COMMTIMEOUTS     CommTimeOuts;
   int       result=1;

   ptrDCB = (LPDCB) &com1DCB;

   if ((handle = CreateFile( "COM1", (GENERIC_WRITE | GENERIC_READ),
                    0, NULL, OPEN_EXISTING,
                   FILE_ATTRIBUTE_NORMAL, NULL))
       == (HANDLE) -1)
   {
      Application->MessageBox("Error opening COM1", "Error Serial Port", 0);
      return(0);
   }

   CommTimeOuts.ReadIntervalTimeout=MAXDWORD;
   CommTimeOuts.ReadTotalTimeoutMultiplier=0;
   CommTimeOuts.ReadTotalTimeoutConstant=500;        // 500 milliseconds
   CommTimeOuts.WriteTotalTimeoutMultiplier=0;
   CommTimeOuts.WriteTotalTimeoutConstant=1000;
   result *= SetCommTimeouts(handle, &CommTimeOuts);
   result *= SetupComm(handle,100,100);

   result *= GetCommState(handle, ptrDCB);
   com1DCB.BaudRate = CBR_9600;
   com1DCB.fDsrSensitivity = FALSE;
   com1DCB.ByteSize = 8;
   com1DCB.Parity = NOPARITY;
   com1DCB.StopBits = ONESTOPBIT;
   result *= SetCommState(handle, ptrDCB);

   if (!result)
```

```
      Application->MessageBox("Cannot open COM1", "Error Serial Port", 0);
   //else
      //Application->MessageBox("COM1 has been opened", "Serial Port COM1", 0);

   return(handle);
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Close Serial Port
void __fastcall TSendAlice::CloseCOM1(HANDLE handle)
{
   CloseHandle (handle);
   //Application->MessageBox("COM1 has been closed", "Serial Port COM1", 0);
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Send one byte to the serial port
void __fastcall TSendAlice::SendByte(BYTE byte_to_send)
{
   int       result=1;
   DWORD           Written;

   result *= WriteFile (handle, &byte_to_send, 1, &Written, NULL);

   if (Written !=1)
   {
      Application->MessageBox("Unable to write to COM1", "Error Serial Port", 0);
      return;
   }

   if (!result)
      Application->MessageBox("Byte not sent", "Error Serial Port", 0);
}
// End TSendAlice class
//---------------------------------------------------------------------------




//---------------------------------------------------------------------------
// TAliceCtrl class
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Construction/Destruction

__fastcall TAliceCtrl::TAliceCtrl()
{
  InitHistory();
  SendAlice = new TSendAlice;
}

__fastcall TAliceCtrl::~TAliceCtrl()
{
  SendAlice->Free();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Initialize Command History
void __fastcall TAliceCtrl::InitHistory()
{
   int i;
```

```
   for (i=0;i<NUMBER_ROBOTS;i++)
      SetPreviousSpeed(i,0,0);
}


//---------------------------------------------------------------------------
// Set actual position
void __fastcall TAliceCtrl::SetCurrentPosition(int nrRob, int x, int y, double
theta)
{
   Robot[nrRob].x0 = x;
   Robot[nrRob].y0 = y;
   Robot[nrRob].theta0 = theta;
}

//---------------------------------------------------------------------------
// Set goal position
void __fastcall TAliceCtrl::SetGoalPosition(int nrRob, int flag, int x, int y,
double theta)
{
  if (flag == 1)
    Robot[nrRob].BackwardFlag = true;
  else
    Robot[nrRob].BackwardFlag = false;
  Robot[nrRob].x = x;
  Robot[nrRob].y = y;
  Robot[nrRob].theta = theta;
}

//---------------------------------------------------------------------------
// Set Speed
void __fastcall TAliceCtrl::SetRequestedSpeed(int nrRob, int SpeedR, int SpeedL)
{
   Robot[nrRob].SpeedR = SpeedR;
   Robot[nrRob].SpeedL = SpeedL;
}

//---------------------------------------------------------------------------
// Get position
void __fastcall TAliceCtrl::GetRequestedSpeed(int nrRob, int *SpeedR, int *SpeedL)
{
   *SpeedR = Robot[nrRob].SpeedR;
   *SpeedL = Robot[nrRob].SpeedL;
}

//---------------------------------------------------------------------------
// Set previous Speed
void __fastcall TAliceCtrl::SetPreviousSpeed(int nrRob, int SpeedR, int SpeedL)
{
   CommandHistory[nrRob].PreviousSpeedR = SpeedR;
   CommandHistory[nrRob].PreviousSpeedL = SpeedL;
}

//---------------------------------------------------------------------------
// Get previous Speed
void __fastcall TAliceCtrl::GetPreviousSpeed(int nrRob, int *SpeedR, int *SpeedL)
{
   *SpeedR = CommandHistory[nrRob].PreviousSpeedR;
   *SpeedL = CommandHistory[nrRob].PreviousSpeedL;
}

//---------------------------------------------------------------------------
// Set requested target depending on mode
void __fastcall TAliceCtrl::SetTarget(int nrRob, int mode, int flag, int x, int y,
double theta)
{
  Robot[nrRob].Mode = mode;

  switch(mode)
  {
```

```
    case(0) : SetRequestedSpeed(nrRob,y,x); // y is right Speed, x is left Speed
              break;
    case(1) : SetGoalPosition(nrRob,flag,x,y,theta);
              break;
    case(2) : break; // not yet implemented
    default : SetGoalPosition(nrRob,flag,x,y,theta);
  }
}

//PositionFlag is now done in Strategy
/*
//---------------------------------------------------------------------------
// Set position flag
void __fastcall TAliceCtrl::SetPositionFlag(int nrRob)
{
   PositionFlag[nrRob] = true;
}

//---------------------------------------------------------------------------
// Clear position flag
void __fastcall TAliceCtrl::ClearPositionFlag(int nrRob)
{
   PositionFlag[nrRob] = false;
}

//---------------------------------------------------------------------------
// Get position flag
bool __fastcall TAliceCtrl::GetPositionFlag(int nrRob)
{
   return(PositionFlag[nrRob]);
}
*/


//---------------------------------------------------------------------------
// Set debug variables
void __fastcall TAliceCtrl::SetDebugVar(int dx, int dy,
   double vr, double v, double omega, double rho, double alpha, double phi, double
arctangent,
   double VelocityRight, double VelocityLeft,
   int x0, int y0, double theta0, int x, int y, double theta,
   int SpeedRight, int SpeedLeft,
   int TestPassing)
{
   DVar.dx = dx;
   DVar.dy = dy;
   DVar.vr = vr;
   DVar.v = v;
   DVar.omega = omega;
   DVar.rho = rho;
   DVar.alpha = alpha;
   DVar.phi = phi;
   DVar.arctangent = arctangent;
   DVar.VelocityRight = VelocityRight;
   DVar.VelocityLeft = VelocityLeft;
   DVar.x0 = x0;
   DVar.y0 = y0;
   DVar.theta0 = theta0;
   DVar.x = x;
   DVar.y = y;
   DVar.theta = theta;
   DVar.SpeedRight = SpeedRight;
   DVar.SpeedLeft = SpeedLeft;
   DVar.TestPassing = TestPassing;
}

//---------------------------------------------------------------------------
// Set more debug variables
void __fastcall TAliceCtrl::SetMoreDebugVar(int PreSpeedRight, int PreSpeedLeft)
{
```

```
   DVar.PreSpeedRight = PreSpeedRight;
   DVar.PreSpeedLeft = PreSpeedLeft;
}


//---------------------------------------------------------------------------
// Get debug variables
void __fastcall TAliceCtrl::GetDebugVar(int *dx, int *dy,
   double *vr, double *v, double *omega, double *rho, double *alpha, double *phi,
double *arctangent,
   double *VelocityRight, double *VelocityLeft,
   int *x0, int *y0, double *theta0, int *x, int *y, double *theta,
   int *SpeedRight, int *SpeedLeft,
   int *PreSpeedRight, int *PreSpeedLeft,
   int *TestPassing)
{
   *dx = DVar.dx;
   *dy = DVar.dy;
   *vr = DVar.vr;
   *v = DVar.v;
   *omega = DVar.omega;
   *rho = DVar.rho;
   *alpha = DVar.alpha;
   *phi = DVar.phi;
   *arctangent = DVar.arctangent;
   *VelocityRight = DVar.VelocityRight;
   *VelocityLeft = DVar.VelocityLeft;
   *x0 = DVar.x0;
   *y0 = DVar.y0;
   *theta0 = DVar.theta0;
   *x = DVar.x;
   *y = DVar.y;
   *theta = DVar.theta;
   *SpeedRight = DVar.SpeedRight;
   *SpeedLeft = DVar.SpeedLeft;
   *PreSpeedRight = DVar.PreSpeedRight;
   *PreSpeedLeft = DVar.PreSpeedLeft;
   *TestPassing = DVar.TestPassing;
}

//---------------------------------------------------------------------------
// Read constants in shared memory
void __fastcall TAliceCtrl::ReadConstants()
{
/*  Sync->BeginRead();       // synchronize read of shared memory
  K_PHI = KPhiShared;
  K_ALPHA = KAlphaShared;
  K_RHO = KRhoShared;
  RHO_THRESHOLD = RhoThresholdShared;
  PHI_THRESHOLD = PhiThresholdShared;
  Sync->EndRead();*/
}

//---------------------------------------------------------------------------
// Calculate the velocity of each wheel of Alice
void __fastcall TAliceCtrl::Velocity(RobotPositions *Rob,
                                     int *SpeedRight, int *SpeedLeft, int nrRob)
//nrRob for debugging
{
   // x, y, x0, y0 in mm
   // theta, theta0 in rad : (-pi,pi] !!!!
   int dx, dy;
   double vr, v, omega, rho, alpha, phi, arcsine;
   double VelocityRight, VelocityLeft;
   double k_phi;

   //Debugging
   int TestPassing = 0;

   dx = (*Rob).x - (*Rob).x0;
```

```
dy = (*Rob).y - (*Rob).y0;

rho = sqrt(dx*dx + dy*dy);
phi = -(*Rob).theta0 + (*Rob).theta;
if (phi > M_PI)
   phi =  -2*M_PI + phi;       // phi : (-pi,pi]
if (phi <= -M_PI)
   phi =  2*M_PI + phi;       // phi : (-pi,pi]

k_phi = K_PHI;
// necessary to avoid instabilities at goal position
if (rho < RHO_THRESHOLD+3)
   k_phi = -2 * k_phi;

// if goal is not achieved yet, then calculate velocity
if (fabs(phi) > PHI_THRESHOLD || rho > RHO_THRESHOLD)
{
   TestPassing = 1; // for debugging

   // calculate alpha only if robot not yet in goal circle
   if (rho > RHO_THRESHOLD)
   {
     // atan(dy/dx) didn't work
     if (rho != 0)
        arcsine = asin(dy/rho);
        else
           arcsine = 0; // if rho = 0, which can not be the case here
     if (dx < 0)
        arcsine = M_PI - arcsine;   // asin : [-pi/2, 3pi/2]
     if (arcsine > M_PI)
        arcsine -= 2*M_PI;  // asin : (-pi, pi]

     // alpha > 0 when turning left
     alpha = -(*Rob).theta0 + arcsine;

     // if backwards arrival is desired
     if ((*Rob).BackwardFlag == true)
       alpha = alpha - M_PI;

     // alpha : (-pi,+pi]
     if (alpha > M_PI)
       alpha = -2*M_PI + alpha;       // alpha : (-pi,+pi]
     if (alpha <= -M_PI)
       alpha = 2*M_PI + alpha;       // alpha : (-pi,+pi]
   }
   // if robot in goal circle
   else
   {
     rho = 0;
     alpha = 0;
   }

   // calculate central velocity of robot in mm/s
   v = K_RHO * rho;
   if (v > VELOCITY_MAX + V_THRESHOLD) // if velocity is higher than available
      v = VELOCITY_MAX;                // velocities of Alice
   if (v < -VELOCITY_MAX - V_THRESHOLD) // if velocity is higher than available
      v = -VELOCITY_MAX;                // velocities of Alice

   // if backwards arrival is desired
   if ((*Rob).BackwardFlag == true)
     v = -v;

   // calculate angular velocity of robot in rad/s
   omega = K_ALPHA * alpha + k_phi * phi; // omega > 0 for turning left

   vr = omega * WHEEL_AXLE;      // radial velocity

   // Attention +/- sens of omega depends on treatment done by COPS
   // At the moment COPS makes a reflection on the y axis
```

```
      // Without any reflection: right +, left -
      VelocityRight = v - vr;
      VelocityLeft = v + vr;

      GetSpeed(VelocityRight, VelocityLeft, omega, SpeedRight, SpeedLeft);
   }
   else
      *SpeedRight = *SpeedLeft = 0; // goal position achieved

   if (nrRob == 0) // Set debug variables for Rob 1
    SetDebugVar(dx, dy,
      vr, v, omega, rho, alpha, phi, arcsine,
      VelocityRight, VelocityLeft,
      (*Rob).x0, (*Rob).y0, (*Rob).theta0, (*Rob).x, (*Rob).y, (*Rob).theta,
      *SpeedRight, *SpeedLeft,
      TestPassing);
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Get Speed
// return values are -3, -2, -1, 0, 1, 2, 3
void __fastcall TAliceCtrl::GetSpeed(double VelocityR, double VelocityL, double
omega,
                                     int *SpeedR, int *SpeedL)
{
   *SpeedR = QuantizeVelocity(VelocityR);
   *SpeedL = QuantizeVelocity(VelocityL);

   // always turn at minimum speed (time delay)
   if (Sign(*SpeedR) == -Sign(*SpeedL) && *SpeedR != 0)
      if (*SpeedR < 0)
      {
         *SpeedR = -SPEED_MIN;
         *SpeedL = SPEED_MIN;
      }
      else
      {
         *SpeedR = SPEED_MIN;
         *SpeedL = -SPEED_MIN;
      }

   // always turn with one wheel only at minimum speed: was not useful
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Quantize Velocity
// return values are -3, -2, -1, 0, 1, 2, 3
int __fastcall TAliceCtrl::QuantizeVelocity(int velocity)
{
   int signum, absvelocity;

   signum = Sign(velocity);
   absvelocity = abs(velocity);

   if (absvelocity > VELOCITY3)
      return(signum * SPEED3);
   if (absvelocity < VELOCITY2)
      if (absvelocity > VELOCITY1)
         return(signum * SPEED1);
      else return(0);
   else return(signum * SPEED2);
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
```

```
// Send speed to Alice
// Evaluate speed of each robot and build 36 bit communication protocol
// Each robot has 6 bits (3 per wheel with 1 sign bit)
// LSB: right speed, MSB: left speed
void __fastcall TAliceCtrl::SendSpeed()
{
   int nrRob, SpeedR, SpeedL;
   int PreSpeedR, PreSpeedL;  // previous speed
   int Word1, Word2, Word3, Word4, Word5, negR, negL;

   //ReadConstants(); // Read constants in shared memory

   for(nrRob=0;nrRob<NUMBER_ROBOTS;nrRob++)
   {
      switch(Robot[nrRob].Mode)
      {
        case(0) : GetRequestedSpeed(nrRob, &SpeedR, &SpeedL);
                  break;
        case(1) : Velocity(&Robot[nrRob], &SpeedR, &SpeedL, nrRob); //nrRob only
for debugging
                  break;
        case(2) : break; // not yet implemented
        default : Velocity(&Robot[nrRob], &SpeedR, &SpeedL, nrRob);
      }

      // Set speed to 0 when passing from + to - or - to +
      GetPreviousSpeed(nrRob, &PreSpeedR, &PreSpeedL);
      if (nrRob==0)
        SetMoreDebugVar(PreSpeedR, PreSpeedL);

      // Process right speed
      if (Sign(SpeedR) == -Sign(PreSpeedR))
        SpeedR = SPEED0;    // acceleration ramp for motors

      // Process left speed
      if (Sign(SpeedL) == -Sign(PreSpeedL))
        SpeedL = SPEED0;    // acceleration ramp for motors

      SetPreviousSpeed(nrRob, SpeedR, SpeedL);
      // end acceleration/deceleration ramps

      // Process right speed
      negR=(SpeedR<0);                 // neg=1 if SpeedR<0
      if (SpeedR<0)
         SpeedR=-SpeedR;

      // Process left speed
      negL=(SpeedL<0);                 // neg=1 if SpeedL<0
      if (SpeedL<0)
         SpeedL=-SpeedL;

      switch(nrRob)
      {
         case(0): Word1 = 0x20*negL + 0x08*SpeedL + 0x04*negR + SpeedR;
                  break;
         case(1): Word1 += 0x40*SpeedR;
                  Word2 = 0x08*negL + 0x02*SpeedL + negR;
                  break;
         case(2): Word2 += 0x80*SpeedL + 0x40*negR + 0x10*SpeedR;
                  Word3 = 0x02*negL;
                  if (SpeedL>1)
                     Word3 += 1;
                  break;
         case(3): Word3 += 0x80*negL + 0x20*SpeedL + 0x10*negR + 0x04*SpeedR;
                  break;
         case(4): Word4 = 0x20*negL + 0x08*SpeedL + 0x04*negR + SpeedR;
                  break;
         case(5): Word4 += 0x40*SpeedR;
                  Word5 = 0x08*negL + 0x02*SpeedL + negR;
      }
```

```
/*      if (SpeedR == 0 && SpeedL == 0)  // if goal position achieved
        SetPositionFlag(nrRob); // Set Flag
      else
        ClearPositionFlag(nrRob);   // if goal position not achieved yet
*/
   }

   // Send all 5 bytes;
   SendAlice->SendByte(Word1);
   SendAlice->SendByte(Word2);
   SendAlice->SendByte(Word3);
   SendAlice->SendByte(Word4);
   SendAlice->SendByte(Word5);
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Stop all Alices
void __fastcall TAliceCtrl::StopRob()
{
   SendAlice->SendByte(0);
   SendAlice->SendByte(0);
   SendAlice->SendByte(0);
   SendAlice->SendByte(0);
   SendAlice->SendByte(0);
}

//---------------------------------------------------------------------------
// Sign: returns
int __fastcall TAliceCtrl::Sign(int value)
{
   if (value > 0)
      return(1);
   else if (value < 0)
      return(-1);
      else
         return(0);
}

// End TAliceCtrl class
//---------------------------------------------------------------------------
```

8.7.2 Alice thread

TAlices.h

```
//---------------------------------------------------------------------------
#ifndef TAlicesH
#define TAlicesH
//---------------------------------------------------------------------------
#include <Classes.hpp>
//#include "Constants.h"
#include "Main.h"
//---------------------------------------------------------------------------
class TAlices : public TThread
{
private:
  //variables
  bool InitFlag; // set for initialization, sends speed 0 to all robots

protected:
  void __fastcall Execute();
public:
  __fastcall TAlices(bool CreateSuspended);
  void __fastcall SetInitFlag();
  void __fastcall ClearInitFlag();
};
//---------------------------------------------------------------------------
#endif
```

TAlices.cpp

```
/*****************************************************************************

  TAlices.cpp  Implements the thread for Alices connection.

  Author:   Patrick Ramer
  Date:     28.01.2000
  Purpose:  Updates Positions in AliceCtrl class and then executes SendSpeed
command

  Program:  Borland 4.0

  Classes:  TAlices: Thread for Alices

  Remarks:

  History:

*****************************************************************************/
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "TAlices.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------
//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall TAlices::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------
```

```
// Extern variables
extern TMultiReadExclusiveWriteSynchronizer* Sync;
extern TAliceCtrl *AliceCtrl;

// Shared memory (TBlob structure is provided by "Main.h")
extern TBlob Rob[NUMBER_ROBOTS];
extern TBlob Ball;
extern TRobTarget RobTarget[NUMBER_ROBOTS];        // Target of each robot

//---------------------------------------------------------------------------
__fastcall TAlices::TAlices(bool CreateSuspended)
  : TThread(CreateSuspended)
{

}
//---------------------------------------------------------------------------
void __fastcall TAlices::Execute()
{
  int i;
// Make the thread object automatically destroyed when the thread terminates
  FreeOnTerminate = true;

  while (!Terminated)
  {

    // Update current positions
    Sync->BeginRead();
    for (i=0;i<NUMBER_ROBOTS;i++)
    {
      AliceCtrl->SetCurrentPosition(i,Rob[i].X,Rob[i].Y,Rob[i].Theta);
      AliceCtrl-
>SetTarget(i,RobTarget[i].Mode,RobTarget[i].Flag,RobTarget[i].SX,RobTarget[i].SY,Ro
bTarget[i].Theta);
    }
    Sync->EndRead();

    if (!InitFlag)
      AliceCtrl->SendSpeed();
    else
      AliceCtrl->StopRob(); // Send speed 0 when initializing

    Sleep(78);  // Wait for Speeds to be transmitted to all Robots
  }
}
//---------------------------------------------------------------------------

// Initialization Flag
void __fastcall TAlices::SetInitFlag()
{
  InitFlag = true;
}

void __fastcall TAlices::ClearInitFlag()
{
  InitFlag = false;
}
```

8.7.3 COPS thread

TCOPS.h

```
//---------------------------------------------------------------------------
#ifndef TCOPSH
#define TCOPSH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <ScktComp.hpp>         // For ClientSocket
#include <Math.h>
#include <stdlib.h>

#include "Constants.h"
// Constants
#define WAC 3        // Wheel Axis Correction in mm
#define XSCALE 0.992 // 0.973 when camera was closer
#define YSCALE 0.92  // 0.87 when camera was closer
#define XOFFSET 8    // Pixels
#define YOFFSET 8    // Pixels

#include "Main.h"
//---------------------------------------------------------------------------
class TCOPS : public TThread
{
private:

  // Structures required by COPS's connection (jcz, 27.09.99)
  struct TDataRequestReset {
    long Size;                      // = 4   -> Size of request, without "Size"
    long ID;                        // = 7    -> Request for Reset
  };

  struct TDataRequestTimeStamp {
    long Size;                      // = 8   -> Size of request, without "Size"
    long ID;                        // = 3    -> Request for TimeStamp
    long OnOff;                     // = 1    -> 1 = ON
  };

  struct TDataRequest {
    long Size;                      // = 14   -> Size of request, without "Size"
    long ID;                        // = 0    -> Request for coordinates
    long Period;                    // = 1    -> Period of reply
    long nFrames;                   // = 1    -> Number of image
    short BlobData;                 // = 11   -> Data to return: X, Y, Theta
  };

  struct TObj {                     // Objects are robots or ball
    float X;
    float Y;
    float Theta;
  };

  struct TDataReplyHeader {
    long Size;                      // Size of reply, without "Size"
    long ID;                        // = 0    -> Like request
  };

  struct TDataReply {
    long TimeStamp;                 //
    long Count;                     // = 7    -> Number of objects
    short RecSize;                  // = 12   -> 12 bytes per objects
    short BlobData;                 // = 11   -> Data type: X, Y, Theta
    TObj Objects[NUMBER_ROBOTS+1];  //        -> Array of objects
  };

  double __fastcall AngleCorrector(double Angle);
  void __fastcall UpdateSharedMemory(TDataReply * DataRep);
```

```
protected:
  void __fastcall Execute();
public:

  // Connection
  bool Connection;

  // TimeStamp, Period and Frequecy of COPS images
  long TS;
  long T;
  float f;

  bool  bReset;

  __fastcall TCOPS(bool CreateSuspended);

};
//---------------------------------------------------------------------------
#endif
```

TCOPS.cpp

```
/*****************************************************************************

  TCOPS.cpp  Implements the thread for COPS connection.

  Author:   Jean-Christophe Zufferey
  Date:     20.01.2000
  Purpose:  Provides current position of players (Alices robots) connecting
            the COPS (Color Object Position Server by Rémy Blank) program,
            using TCP/IP protocol

  Program:  Borland 4.0

  Classes:  TCOPS: Thread for COPS (Color Object Position Server) connection

  Modifications:
            08.02.2000/jcz  Addition of EventCOPS

  Remarks:

  History:

*****************************************************************************/
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "TCOPS.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------
//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall TCOPS::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------

TClientSocket* ClientSocket;

// Extern variables
extern TMultiReadExclusiveWriteSynchronizer* Sync;
extern TEvent* EventCOPS;
```

```
// Shared memory
extern TBlob Rob[NUMBER_ROBOTS];             // Current position of each robot
extern TBlob Ball;                           // Current position of the ball
extern long TimeStamp;                       // Time stamp of current image
extern short EventCounter;                   // For EventCOPS

//---------------------------------------------------------------------------
__fastcall TCOPS::TCOPS(bool CreateSuspended)
  : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
// ***  ***
//
void __fastcall TCOPS::Execute()
{
  TDataRequestTimeStamp DataReqTS;
  TDataRequest DataReq;
  TDataReplyHeader * DataRepHeader;
  TDataReply * DataRep;
  int nBytes;

  // Make the thread object automatically destroyed when the thread terminates
  FreeOnTerminate = true;

  // Initialisation of params
  Connection = false;
  bReset = false;

  // Create an instance of TClientSocket
  TClientSocket* ClientSocket = new TClientSocket(FormASM);

  // Allocate DataRep for the responses of COPS
  DataRepHeader = (TDataReplyHeader *) malloc(sizeof(TDataReplyHeader));
  DataRep = (TDataReply *) malloc(sizeof(TDataReply));

  // Activate the TCP/IP connection with COPS
  if (ClientSocket->Active != true)
  {
    ClientSocket->ClientType=ctBlocking;
    ClientSocket->Port = 1024;
    ClientSocket->Host ="localhost";
  }
  ClientSocket->Open();

  if (ClientSocket->Active == false)
    ShowMessage("Problem !!!");

  Connection = true;

  // Prepare requests to COPS
  DataReqTS.Size = 8;     // Size of request
  DataReqTS.ID = 3;       // Request for TimeStamp
  DataReqTS.OnOff = 1;    // ON

  DataReq.Size = 14;      // Size of request
  DataReq.ID = 0;         // Request for coordinates
  DataReq.Period = 1;     // Period of response
  DataReq.nFrames = 1;    // Number of images (-1=continuous)
  DataReq.BlobData = 11;  // Data to return (X,Y,Theta)

  // Write the TimeStamp request to the server (COPS)
  ClientSocket->Socket->SendBuf(&DataReqTS, 12);

// Main loop of the thread

  // Create a TWinSocketStream for reading (and writing)
  TWinSocketStream *pStream = new TWinSocketStream(ClientSocket->Socket, 60000);
  try
```

```
  {
    // fetch and process commands until the connection or thread is terminated
    while (!Terminated && ClientSocket->Active)
    {
      try
      {
        if (bReset)
        {
          TDataRequestReset DataReqReset;
          DataReqReset.ID = 7;
          DataReqReset.Size = 4;
          ClientSocket->Socket->SendBuf(&DataReqReset, sizeof(TDataRequestReset));
          bReset = false;
        }

        // Write the request to the server (COPS)
        ClientSocket->Socket->SendBuf(&DataReq, 18);

        // What sort of reply do we have ?
        nBytes=pStream->Read((void *) DataRepHeader, sizeof(TDataReplyHeader));

        // Read a response from the server)
        nBytes=pStream->Read((void *) DataRep, sizeof(TDataReply));

        if (nBytes==sizeof(TDataReply))
        {
          T = DataRep->TimeStamp-TS;
          f = 1/(T*0.001);
          TS = DataRep->TimeStamp;
          UpdateSharedMemory(DataRep);

          // Signal to user thread that shared memory has been updated
          Sync->BeginWrite();
            EventCounter=2;
          Sync->EndWrite();
          //EventCOPS->ResetEvent();
          EventCOPS->SetEvent();    // for synchronization with User Thread
        }

      }
      catch (Exception &E)
      {
//        if (!E.ClassNameIs("EAbort"))
//          Synchronize(HandleThreadException()); // you must write
HandleThreadException
      }
    }
  }
  __finally
  {
    delete pStream;
  }

  ClientSocket->Active = false;
  ClientSocket->Close();
  delete ClientSocket;

}
//---------------------------------------------------------------------------
//  Angle Corrector : fonction utilisee le temps que Remy corrige le bug
//  d'angle de COPS ...
double __fastcall TCOPS::AngleCorrector(double Angle)
{
  Angle += M_PI / 2;

  if ((Angle >= -M_PI/2) && (Angle < 0))
      Angle += M_PI;
  else if ((Angle > M_PI/2) && (Angle < M_PI))
      Angle += M_PI;
```

```
  // change interval
  if (Angle > M_PI)
    Angle -= 2*M_PI; // Angle : [-pi,pi]
  return(Angle);
}
//---------------------------------------------------------------------------

void _fastcall TCOPS::UpdateSharedMemory(TDataReply * DataRep)
{
  int n;
  float Theta;

  Sync->BeginWrite();
    TimeStamp = DataRep->TimeStamp;
    for (n=0;n<NUMBER_ROBOTS;n++)
    {
      Theta = AngleCorrector(DataRep->Objects[n].Theta);
      Rob[n].X = (DataRep->Objects[n].X-XOFFSET)*XSCALE-WAC*cos(Theta);
      Rob[n].Y = (DataRep->Objects[n].Y-YOFFSET)*YSCALE-WAC*sin(Theta);
      Rob[n].Theta = Theta;
      Rob[n].Speed = 0;
    }
    Ball.X = DataRep->Objects[n].X;
    Ball.Y = DataRep->Objects[n].Y;
    Ball.Theta = DataRep->Objects[n].Theta;
    Ball.Speed = 0;
  Sync->EndWrite();
}
//---------------------------------------------------------------------------
```

8.7.4 User thread

TUSER.h

```
//---------------------------------------------------------------------------
#ifndef TUSERH
#define TUSERH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <ScktComp.hpp>
#include <alloc.h>

#include "Main.h"
#include "Constants.h"
//---------------------------------------------------------------------------
class TUSER : public TServerClientThread
{
private:

  struct TRobTarget                    // (20 bytes)
  {
    long Mode;                         // mode of command (0,1,2)
    long Flag;                         // depending on mode
    float SX;                          // speed of left wheel or x coordinate
    float SY;                          // speed of right wheel or y coordinate
    float Theta;                       // arrival angle (mode 1 only)
  };

  struct TObject                       // (16 bytes)
  {
    float X;
    float Y;
    float Theta;
    float Speed;
  };

  struct TDataRequest                  // (3x4 + 3x20 = 72 bytes)
  {
    long Size;                         // size of request without size = 48
    long ID;                           // request identifier
    long Period;                       // period of replies
    TRobTarget Robot[NUMBER_ROBOTS/2]; // target of each robot
  };

  struct TDataReply                     // (5x5 + 2x3x16 + 1x16 = 137 bytes)
  {
    long Size;                          // size of request without size = 133
    long ID;                            // reply identifier
    long TeamID;                        // team identifier
    long Mode;                          // mode of command
    long Error;                         // number of an error
    long TimeStamp;                     // time stamp of image capture
    TObject MyRobot[NUMBER_ROBOTS/2];   // current data of robots of this team
    TObject YourRobot[NUMBER_ROBOTS/2]; // current data of robots of other team
    TObject Ball;                       // current data of the ball
  };

  int noSocket;

  // Reflection methods
  float __fastcall ReflectX(float X);
  float __fastcall ReflectY(float Y);
  float __fastcall ReflectTheta(float Theta);

protected:

  void __fastcall ClientExecute();
```

```
public:
  short TeamID;

  TUSER(bool CreateSuspended, TServerClientWinSocket* ASocket);

};
//---------------------------------------------------------------------------
#endif
```

TUSER.cpp

```
/*****************************************************************************

  Main.cpp  USER Thread

  Author:   Jean-Christophe Zufferey & Patrick Ramer
  Date:     20.01.2000
  Purpose:  Manages connection with USER
  Program:  Borland 4.0

  Classes:  TUSER     Thread for USER

  Modifications:
            31.01.2000/pr  Reflection on y axis
            03.02.2000/pr  Reflection on center point
            08.02.2000/jcz Addition of EventCOPS

  Remarks:

  History:

*****************************************************************************/
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "TUSER.h"
#pragma package(smart_init)

// Extern variables
extern TMultiReadExclusiveWriteSynchronizer* Sync;
extern TEvent* EventCOPS;

// Shared memory
extern TBlob Rob[NUMBER_ROBOTS];              // Current position of each robot
extern TBlob Ball;                            // Current position of the ball
extern long TimeStamp;                        // Time stamp of current image

extern TRobTarget RobTarget[NUMBER_ROBOTS];   // Target of each robot
extern short EventCounter;                    // For EventCOPS

//---------------------------------------------------------------------------
// *** This construcor is an heritage of TServerClientThread ***
//
TUSER::TUSER(bool CreateSuspended, TServerClientWinSocket* ASocket)
  :TServerClientThread(CreateSuspended, ASocket)
{
  noSocket=ASocket->SocketHandle;
}
//---------------------------------------------------------------------------
// *** ***
//
void __fastcall TUSER::ClientExecute()
{
  TWinSocketStream *pStream;
  TDataRequest * DataReq;
  TDataReply DataRep;
  int nBytes,i;

  // Make the thread object automatically destroyed when the thread terminates
```

```
  FreeOnTerminate = true;

  // Initialisation of params


  // Allocate DataReq for the request of client
  DataReq = (TDataRequest *) malloc(sizeof(TDataRequest));

// Main loop of the thread

  // Create a TWinSocketStream for reading and writing
  pStream = new TWinSocketStream(ClientSocket, 60000);
  try
  {
    // fetch and process commands until the connection or thread is terminated
    while (!Terminated && ClientSocket->Connected)
    {
      try
      {
        // Read a request from the client
        nBytes=pStream->Read((void *) DataReq, sizeof(TDataRequest));

        if (nBytes==sizeof(TDataRequest))
        {
          if (TeamID==1)
            for (i=0;i<NUMBER_ROBOTS/2;i++)
            {
              // Write the requested target in shared memory
              Sync->BeginWrite();
                RobTarget[i].Mode = DataReq->Robot[i].Mode;
                RobTarget[i].Flag = DataReq->Robot[i].Flag;
                RobTarget[i].SX = DataReq->Robot[i].SX;
                RobTarget[i].SY = DataReq->Robot[i].SY;
                RobTarget[i].Theta = DataReq->Robot[i].Theta;
              Sync->EndWrite();
            }
          else
            for (i=0;i<NUMBER_ROBOTS/2;i++)
            {
              // Write the requested target in shared memory
              Sync->BeginWrite();
                RobTarget[i+3].Mode = DataReq->Robot[i].Mode;
                RobTarget[i+3].Flag = DataReq->Robot[i].Flag;
                // in Position control mode (1) SX and SY contain positions
                // -> reflection
                if (DataReq->Robot[i].Mode == 1)
                {
                  RobTarget[i+3].SX = ReflectX(DataReq->Robot[i].SX);
                  RobTarget[i+3].SY = ReflectY(DataReq->Robot[i].SY);
                  RobTarget[i+3].Theta = ReflectTheta(DataReq->Robot[i].Theta);
                }
                else
                {
                  RobTarget[i+3].SX = DataReq->Robot[i].SX;
                  RobTarget[i+3].SY = DataReq->Robot[i].SY;
                  RobTarget[i+3].Theta = DataReq->Robot[i].Theta;
                }
              Sync->EndWrite();
            }

          EventCOPS->WaitFor(2000);  // Wait for COPS signal (new values)
          Sync->BeginRead();
            EventCounter = EventCounter-1;
            if (EventCounter==0)
              EventCOPS->ResetEvent();
          Sync->EndRead();

          // Prepare the reply
          DataRep.Size = sizeof(TDataReply)-4;      // Size of reply
          DataRep.ID = 0;
```

```
          DataRep.TeamID = TeamID;
          DataRep.Mode = 0;
          DataRep.Error = 0;
          Sync->BeginRead();
            DataRep.TimeStamp = TimeStamp;
            if (TeamID==1)
            {
              for (i=0;i<NUMBER_ROBOTS/2;i++)
              {
                DataRep.MyRobot[i].X = Rob[i].X;
                DataRep.MyRobot[i].Y = Rob[i].Y;
                DataRep.MyRobot[i].Theta = Rob[i].Theta;
                DataRep.MyRobot[i].Speed = Rob[i].Speed;
                DataRep.YourRobot[i].X = Rob[i+3].X;
                DataRep.YourRobot[i].Y = Rob[i+3].Y;
                DataRep.YourRobot[i].Theta = Rob[i+3].Theta;
                DataRep.YourRobot[i].Speed = Rob[i+3].Speed;
              }
              DataRep.Ball.X = Ball.X;
              DataRep.Ball.Y = Ball.Y;
            }
            else
            {
              for (i=0;i<NUMBER_ROBOTS/2;i++)
              {
                DataRep.MyRobot[i].X = ReflectX(Rob[i+3].X);
                DataRep.MyRobot[i].Y = ReflectY(Rob[i+3].Y);
                DataRep.MyRobot[i].Theta = ReflectTheta(Rob[i+3].Theta);
                DataRep.MyRobot[i].Speed = Rob[i+3].Speed;
                DataRep.YourRobot[i].X = ReflectX(Rob[i].X);
                DataRep.YourRobot[i].Y = ReflectY(Rob[i].Y);
                DataRep.YourRobot[i].Theta = ReflectTheta(Rob[i].Theta);
                DataRep.YourRobot[i].Speed = Rob[i].Speed;
              }
              DataRep.Ball.X = ReflectX(Ball.X);
              DataRep.Ball.Y = ReflectY(Ball.Y);
            }

            DataRep.Ball.Theta = Ball.Theta;
            DataRep.Ball.Speed = Ball.Speed;
          Sync->EndRead();

          // Write the reply to the client
          ClientSocket->SendBuf(&DataRep, sizeof(TDataReply));
        }
      }
      catch (Exception &E)
      {
        ClientSocket->Disconnect(noSocket);
        ClientSocket->Close();
//        if (!E.ClassNameIs("EAbort"))
//          Synchronize(HandleThreadException()); // you must write
HandleThreadException
      }
    }
  }
  __finally
  {
    delete pStream;
  }

//  ClientSocket->Disconnect();
//  ClientSocket->Close();
//  delete ClientSocket;

}
//---------------------------------------------------------------------------
// Reflection Methods
// Reflection has to be done for the second team before sending positions from
// COPS and after receiving target positions. This way, both teams can have
```

```
// a strategy as if they were on the same side, and it won't matter which team
// gets which Team ID.


//-------------------------------------------------------------------------
// Reflect X on y axis
float __fastcall TUSER::ReflectX(float X)
{
  return(FIELD_LENGTH - X);
}

//-------------------------------------------------------------------------
// Reflect Y on x axis
float __fastcall TUSER::ReflectY(float Y)
{
  return(FIELD_WIDTH - Y);
}

//-------------------------------------------------------------------------
// Reflect Theta on y axis
float __fastcall TUSER::ReflectTheta(float Theta)
{
  float t;

  t = Theta + M_PI;
  if (t > M_PI)
    t =  -2*M_PI + t;       // t : (-pi,pi]
  if (t <= -M_PI)
    t =  2*M_PI + t;        // t : (-pi,pi]

  return(t);
}
```

8.7.5 Main

Main.h

```
//---------------------------------------------------------------------------
#ifndef MainH
#define MainH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>

#include "Constants.h"

#include "TCOPS.h"
#include "TAlices.h"
#include "TUSER.h"

#include "AliceCom.h"

#include <ExtCtrls.hpp>
#include <ScktComp.hpp>
#include "ActiveCOPSProj_OCX.h"
#include <OleCtrls.hpp>

// *** Constants ***

#define COPS_PATH "D:/Programs/COPS/COPS.exe"

// *** For shared memory ***
struct TBlob
{
  float X;
  float Y;
  float Theta;
  float Speed;
};

struct TRobTarget
{
  long Mode;                          // mode of command (0,1,2)
  long Flag;                          // depending on mode
  float SX;                           // speed of left wheel or x coordinate
  float SY;                           // speed of right wheel or y coordinate
  float Theta;                        // arrival angle (mode 1 only)
};

struct TTeam
{
  short TeamID;         // 1 or 2
  String RemoteHost;
  String RemoteAddress;
  String RemotePort;
};

//---------------------------------------------------------------------------
class TFormASM : public TForm
{
__published: // IDE-managed Components
  TLabel *Label1;
  TLabel *Label2;
  TTimer *Timer1;
  TGroupBox *Robot1;
  TLabel *Label3;
  TLabel *Label4;
  TLabel *Label5;
  TEdit *x1;
```

```
  TEdit *y1;
  TEdit *theta1;
  TGroupBox *Robot2;
  TLabel *Label6;
  TLabel *Label7;
  TLabel *Label8;
  TEdit *x2;
  TEdit *y2;
  TEdit *theta2;
  TGroupBox *Robot3;
  TLabel *Label9;
  TLabel *Label10;
  TLabel *Label11;
  TEdit *x3;
  TEdit *y3;
  TEdit *theta3;
  TGroupBox *Robot4;
  TLabel *Label12;
  TLabel *Label13;
  TLabel *Label14;
  TEdit *x4;
  TEdit *y4;
  TEdit *theta4;
  TGroupBox *Robot5;
  TLabel *Label15;
  TLabel *Label16;
  TLabel *Label17;
  TEdit *x5;
  TEdit *y5;
  TEdit *theta5;
  TGroupBox *Robot6;
  TLabel *Label18;
  TLabel *Label19;
  TLabel *Label20;
  TEdit *x6;
  TEdit *y6;
  TEdit *theta6;
  TLabel *Label21;
  TLabel *Label22;
  TGroupBox *GroupBox1;
  TLabel *Label23;
  TLabel *Label24;
  TLabel *Label25;
  TEdit *xb;
  TEdit *yb;
  TEdit *thetab;
  TServerSocket *ServerSocket;
  TLabel *Label26;
  TLabel *Label27;
  TEdit *EditHost1;
  TEdit *EditHost2;
  TLabel *Label28;
  TLabel *Label29;
  TEdit *EditAddress1;
  TEdit *EditAddress2;
  TLabel *Label30;
  TLabel *Label31;
  TEdit *EditPort1;
  TEdit *EditPort2;
  TLabel *Label32;
  TLabel *Label33;
  TGroupBox *GroupBox2;
  TLabel *Label34;
  TLabel *Label35;
  TLabel *Label36;
  TEdit *tx1;
  TEdit *ty1;
  TEdit *ttheta1;
  TGroupBox *GroupBox3;
  TLabel *Label37;
```

```
  TLabel *Label38;
  TLabel *Label39;
  TGroupBox *GroupBox4;
  TLabel *Label40;
  TLabel *Label41;
  TLabel *Label42;
  TEdit *tx3;
  TEdit *ty3;
  TEdit *ttheta3;
  TGroupBox *GroupBox5;
  TLabel *Label43;
  TLabel *Label44;
  TLabel *Label45;
  TEdit *tx4;
  TEdit *ty4;
  TEdit *ttheta4;
  TGroupBox *GroupBox6;
  TLabel *Label46;
  TLabel *Label47;
  TLabel *Label48;
  TEdit *tx5;
  TEdit *ty5;
  TEdit *ttheta5;
  TGroupBox *GroupBox7;
  TLabel *Label49;
  TLabel *Label50;
  TLabel *Label51;
  TEdit *tx6;
  TEdit *ty6;
  TEdit *ttheta6;
  TLabel *Label52;
  TEdit *tx2;
  TEdit *ty2;
  TEdit *ttheta2;
  TButton *Button1;
  TButton *InitButton;
  TButton *StartButton;
  void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
  void __fastcall Timer1Timer(TObject *Sender);
  void __fastcall ServerSocketGetThread(TObject *Sender,
          TServerClientWinSocket *ClientSocket,
          TServerClientThread *&SocketThread);
  void __fastcall Button1Click(TObject *Sender);
  void __fastcall InitButtonClick(TObject *Sender);
  void __fastcall StartButtonClick(TObject *Sender);


private:     // User declarations

  void __fastcall LaunchCOPS();
  void __fastcall InitCOPSThread();
  void __fastcall InitAlicesThread();
  void __fastcall InitUSERThread();
  void __fastcall Initializing();
  void __fastcall Starting();

public:             // User declarations

  __fastcall TFormASM(TComponent* Owner);

};
//---------------------------------------------------------------------------
extern PACKAGE TFormASM *FormASM;
//---------------------------------------------------------------------------
#endif
```

Main.cpp

```
// *** Thread pointers ***
TCOPS *COPSThread;                        // Handle of COPS thread
```

```
TUSER *USERThread1, *USERThread2;          // Handle of USERs threads

TAlices *AlicesThread;                     // Handle of Alices thread

// *** Tools for thread management ***
TMultiReadExclusiveWriteSynchronizer *Sync; // Synchronizer for shared memory
access
TEvent *EventCOPS;                         // Event handler for COPS signal

// *** TAliceCtrl class ***
TAliceCtrl *AliceCtrl;

// *** Shared memory ***
TBlob Rob[NUMBER_ROBOTS];                  // Current position of each robot
TBlob Ball;                                // Current position of the ball
long TimeStamp;                            // Time stamp of current image
short EventCounter;                        // For EventCOPS

TRobTarget RobTarget[NUMBER_ROBOTS];       // Target of each robot

TTeam Team1, Team2;                        //

//---------------------------------------------------------------------------
// *** Initialisations ***
//
__fastcall TFormASM::TFormASM(TComponent* Owner)
  : TForm(Owner)
{
//  LaunchCOPS();       // Launch COPS program (image processing)

  Sync = new TMultiReadExclusiveWriteSynchronizer;  // for shared memory
  EventCOPS = new TEvent(NULL, true, false, NULL); // for COPS signal

  InitCOPSThread();
//  InitConstants();  // do this before InitAlicesThread
  InitAlicesThread();
  InitUSERThread();
  Timer1->Enabled = true; // start timer to display positions from COPS
}
//---------------------------------------------------------------------------
// *** Launch of COPS program ***
//
void __fastcall TFormASM::LaunchCOPS()
{
  LPPROCESS_INFORMATION lpProcessInformation;
  LPSTARTUPINFO  lpStartupInfo;

  lpStartupInfo = (LPSTARTUPINFO) malloc(sizeof(STARTUPINFO));

  lpStartupInfo->cb = sizeof(STARTUPINFO);
  lpStartupInfo->lpReserved = NULL;
  lpStartupInfo->lpTitle = NULL;
  lpStartupInfo->dwX = 0;
  lpStartupInfo->dwY = 0;
  lpStartupInfo->dwFlags = STARTF_USEPOSITION;
  lpStartupInfo->cbReserved2 = 0;
  lpStartupInfo->lpReserved2 = NULL;

  lpProcessInformation = (LPPROCESS_INFORMATION)
malloc(sizeof(PROCESS_INFORMATION));

  if (!CreateProcess(
        NULL,// pointer to name of executable module
        COPS_PATH,  // pointer to command line string
        NULL,// pointer to process security attributes
        NULL,// pointer to thread security attributes
        false,      // handle inheritance flag
        0,   // creation flags
        NULL,// pointer to new environment block
        NULL,// pointer to current directory name
```

```
        lpStartupInfo,     // pointer to STARTUPINFO
        lpProcessInformation     // pointer to PROCESS_INFORMATION
     ))
    ShowMessage("Error : COPS program was not launched");
  else
    hCOPS=lpProcessInformation->hProcess;

}

//---------------------------------------------------------------------------
// *** Initialisation of COPS Thread ***
//
void __fastcall TFormASM::InitCOPSThread()
{
  COPSThread = new TCOPS(false);    // launch the COPS Thread
}

//---------------------------------------------------------------------------
// *** Initialisation of ALICES Thread ***
//
void __fastcall TFormASM::InitAlicesThread()
{
  AliceCtrl = new TAliceCtrl;
  AlicesThread = new TAlices(false);
  SetThreadPriority(AlicesThread,THREAD_PRIORITY_TIME_CRITICAL);
  Initializing();
}

//---------------------------------------------------------------------------
// *** Initialisation of USER Thread ***
//
void __fastcall TFormASM::InitUSERThread()
{
  Team1.TeamID = 0;
  Team2.TeamID = 0;
}

void __fastcall TFormASM::Initializing()
{
  AlicesThread->SetInitFlag();
}
//---------------------------------------------------------------------------

void __fastcall TFormASM::Starting()
{
  AlicesThread->ClearInitFlag();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// *** Terminate Threads and free pointers
//
void __fastcall TFormASM::FormClose(TObject *Sender, TCloseAction &Action)
{

// stop timer to display positions from COPS
  Timer1->Enabled = false;

// Terminate Alices thread
  if (AlicesThread!=NULL)
  {
    AlicesThread->Terminate();
    AlicesThread->WaitFor();
  }

// Terminate USERs threads
  if (USERThread1!=NULL)
  {
    USERThread1->Terminate();
```

```
    USERThread1->WaitFor();
  }
  if (USERThread2!=NULL)
  {
    USERThread2->Terminate();
    USERThread2->WaitFor();
  }

// Terminate COPS thread
  if (COPSThread!=NULL)
  {
    COPSThread->Terminate();
    COPSThread->WaitFor();
  }

  Sync->Free();
  AliceCtrl->Free();

//  EnumWindows(NULL, NULL);
//  GetWindowText

/*  UINT uExitCode;
  LPDWORD  lpExitCode;

  if (hCOPS!=NULL)
  {
    GetExitCodeProcess(hCOPS,lpExitCode);
    ExitProcess((UINT) lpExitCode);
//    TerminateProcess(hCOPS,uExitCode);
  }
*/

}
//---------------------------------------------------------------------------
// *** Refresh of Robot position
//
void __fastcall TFormASM::Timer1Timer(TObject *Sender)
{
  Sync->BeginRead();
    x1->Text = Rob[0].X;
    y1->Text = Rob[0].Y;
    theta1->Text = Rob[0].Theta;
    x2->Text = Rob[1].X;
    y2->Text = Rob[1].Y;
    theta2->Text = Rob[1].Theta;
    x3->Text = Rob[2].X;
    y3->Text = Rob[2].Y;
    theta3->Text = Rob[2].Theta;
    x4->Text = Rob[3].X;
    y4->Text = Rob[3].Y;
    theta4->Text = Rob[3].Theta;
    x5->Text = Rob[4].X;
    y5->Text = Rob[4].Y;
    theta5->Text = Rob[4].Theta;
    x6->Text = Rob[5].X;
    y6->Text = Rob[5].Y;
    theta6->Text = Rob[5].Theta;
    xb->Text = Rob[6].X;
    yb->Text = Rob[6].Y;
    thetab->Text = Rob[6].Theta;

    // RobTarget
    tx1->Text = RobTarget[0].SX;
    ty1->Text = RobTarget[0].SY;
    ttheta1->Text = RobTarget[0].Theta;
    tx2->Text = RobTarget[1].SX;
    ty2->Text = RobTarget[1].SY;
    ttheta2->Text = RobTarget[1].Theta;
    tx3->Text = RobTarget[2].SX;
    ty3->Text = RobTarget[2].SY;
```

```
    ttheta3->Text = RobTarget[2].Theta;
    tx4->Text = RobTarget[3].SX;
    ty4->Text = RobTarget[3].SY;
    ttheta4->Text = RobTarget[3].Theta;
    tx5->Text = RobTarget[4].SX;
    ty5->Text = RobTarget[4].SY;
    ttheta5->Text = RobTarget[4].Theta;
    tx6->Text = RobTarget[5].SX;
    ty6->Text = RobTarget[5].SY;
    ttheta6->Text = RobTarget[5].Theta;

  Sync->EndRead();
}
//---------------------------------------------------------------------------
// *** Launch a new TUSER thread for each client ***
//
void __fastcall TFormASM::ServerSocketGetThread(TObject *Sender,
      TServerClientWinSocket *ClientSocket,
      TServerClientThread *&SocketThread)
{
  if (Team1.TeamID==0)
  {
    Team1.TeamID = 1;
    Team1.RemoteHost = ClientSocket->RemoteHost;
    Team1.RemoteAddress = ClientSocket->RemoteAddress;
    Team1.RemotePort = ClientSocket->RemotePort;
    EditHost1->Text = Team1.RemoteHost;
    EditAddress1->Text = Team1.RemoteAddress;
    EditPort1->Text = Team1.RemotePort;
    USERThread1 = new TUSER(false, ClientSocket);
    SocketThread = USERThread1;
    USERThread1->TeamID = 1;
  }
  else if (Team2.TeamID==0)
  {
    Team2.TeamID = 2;
    Team2.RemoteHost = ClientSocket->RemoteHost;
    Team2.RemoteAddress = ClientSocket->RemoteAddress;
    Team2.RemotePort = ClientSocket->RemotePort;
    EditHost2->Text = Team2.RemoteHost;
    EditAddress2->Text = Team2.RemoteAddress;
    EditPort2->Text = Team2.RemotePort;
    USERThread2 = new TUSER(false, ClientSocket);
    SocketThread = USERThread2;
    USERThread2->TeamID = 2;
  }


}
//---------------------------------------------------------------------------

void __fastcall TFormASM::Button1Click(TObject *Sender)
{
  COPSThread->bReset = true;
}
//---------------------------------------------------------------------------

void __fastcall TFormASM::InitButtonClick(TObject *Sender)
{
  Initializing();
}
//---------------------------------------------------------------------------

void __fastcall TFormASM::StartButtonClick(TObject *Sender)
{
  Starting();
}
//---------------------------------------------------------------------------
```

8.8 C++ code of user program

Main.h

```
//---------------------------------------------------------------------------
#ifndef MainH
#define MainH

#define NUMBER_ROBOTS 6 // Number of robots to control
#define NUMBER_ROBOTS_TEAM 3  // number of robots per team

//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ScktComp.hpp>
#include <alloc.h>

#include "Strategy.h"

  struct TRobTarget
  {
    long Mode;    //mode of command
    long Flag;    //depending on the mode
    float SX;     //speed of left wheel or x coordinate
    float SY;     //speed of right wheel or y coordinate
    float Theta;  //arrival angle (only mode 1)
  };

  struct TObj
  {
    float X;
    float Y;
    float Theta;
    float Speed;
  };

  struct TDataRequest
  {
    long Size;
    long ID;
    long Period;
    TRobTarget Robot[NUMBER_ROBOTS/2];
  };

  struct TDataReply
  {
    long Size;
    long ID;
    long TeamID;
    long Mode;
    long Error;
    long TimeStamp;
    TObj MyRobot[NUMBER_ROBOTS/2];
    TObj YourRobot[NUMBER_ROBOTS/2];
    TObj Ball;
  };

//---------------------------------------------------------------------------
class TMainForm : public TForm
{
__published: // IDE-managed Components
        TClientSocket *ClientSocket1;
        TEdit *IPAddress1;
        TButton *ConnectUser1;
  TEdit *EditX;
  TMemo *DisplayPosition;
```

```
  TEdit *Edit1;
        void __fastcall ConnectUser1Click(TObject *Sender);
  void __fastcall ClientSocket1Connect(TObject *Sender,
          TCustomWinSocket *Socket);
  void __fastcall ClientSocket1Read(TObject *Sender,
          TCustomWinSocket *Socket);
  void __fastcall FormClose(TObject *Sender, TCloseAction &Action);


private:     // User declarations
  void __fastcall InitStrategy();

public:             // User declarations
        __fastcall TMainForm(TComponent* Owner);

};
//---------------------------------------------------------------------------
extern PACKAGE TMainForm *MainForm;
//---------------------------------------------------------------------------
#endif
```

Main.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Main.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

TStrategy *Strategy;

TMainForm *MainForm;
long OldTimeStamp, moyenne=0;
//---------------------------------------------------------------------------
__fastcall TMainForm::TMainForm(TComponent* Owner)
        : TForm(Owner)
{
  InitStrategy();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void __fastcall TMainForm::InitStrategy()
{
  Strategy = new TStrategy;
}

void __fastcall TMainForm::ConnectUser1Click(TObject *Sender)
{
  ClientSocket1->Host = IPAddress1->Text;
  ClientSocket1->Active = true;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::ClientSocket1Connect(TObject *Sender,
      TCustomWinSocket *Socket)
{
  TDataRequest DataReq;
  int i,mode,flag,x,y,result;
  double theta;

  DataReq.Size=sizeof(TDataRequest)-4;
  DataReq.ID=0;
  DataReq.Period=1;

  for (i=0;i<NUMBER_ROBOTS_TEAM;i++)
  {
```

```
    result = Strategy->GetNextRequest(i,&mode,&flag,&x,&y,&theta);
    DataReq.Robot[i].Mode = mode;
    DataReq.Robot[i].Flag = flag;
    DataReq.Robot[i].SX = x;
    DataReq.Robot[i].SY = y;
    DataReq.Robot[i].Theta = theta;
  }

  ClientSocket1->Socket->SendBuf(&DataReq, sizeof(TDataRequest));
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::ClientSocket1Read(TObject *Sender,
      TCustomWinSocket *Socket)
{
  TDataReply * DataRep;
  TDataRequest DataReq;
  int i,mode,flag,x,y,result;
  double theta;


  // Receive reply
  DataRep = (TDataReply *) malloc(sizeof(TDataReply));

  ClientSocket1->Socket->ReceiveBuf((void *) DataRep, sizeof(TDataReply));

  // Update current positions
  for (i=0;i<NUMBER_ROBOTS_TEAM;i++)
  {
    Strategy->SetCurrentPosition(i,DataRep->MyRobot[i].X,DataRep-
>MyRobot[i].Y,DataRep->MyRobot[i].Theta);
    Strategy->SetCurrentPosition(i+NUMBER_ROBOTS_TEAM,DataRep-
>YourRobot[i].X,DataRep->YourRobot[i].Y,DataRep->YourRobot[i].Theta);
  }
  Strategy->SetBallPosition(DataRep->Ball.X,DataRep->Ball.Y);

  // Display received positions
  EditX->Text = DataRep->MyRobot[0].X;
  DisplayPosition->Text = "";
  DisplayPosition->Lines->Add("X0 = "+(String)DataRep->MyRobot[0].X);
  DisplayPosition->Lines->Add("Y0 = "+(String)DataRep->MyRobot[0].Y);
  DisplayPosition->Lines->Add("Theta0 = "+(String)DataRep->MyRobot[0].Theta);
  DisplayPosition->Lines->Add("Speed0 = "+(String)DataRep->MyRobot[0].Speed);
  DisplayPosition->Lines->Add("X1 = "+(String)DataRep->MyRobot[1].X);
  DisplayPosition->Lines->Add("Y1 = "+(String)DataRep->MyRobot[1].Y);
  DisplayPosition->Lines->Add("Theta1 = "+(String)DataRep->MyRobot[1].Theta);
  DisplayPosition->Lines->Add("Speed1 = "+(String)DataRep->MyRobot[1].Speed);
  DisplayPosition->Lines->Add("X2 = "+(String)DataRep->MyRobot[2].X);
  DisplayPosition->Lines->Add("Y2 = "+(String)DataRep->MyRobot[2].Y);
  DisplayPosition->Lines->Add("Theta2 = "+(String)DataRep->MyRobot[2].Theta);
  DisplayPosition->Lines->Add("Speed2 = "+(String)DataRep->MyRobot[2].Speed);

  if (DataRep->TimeStamp-OldTimeStamp!=0)
    moyenne = (moyenne + DataRep->TimeStamp-OldTimeStamp)/2;
  Edit1->Text = moyenne;
  OldTimeStamp = DataRep->TimeStamp;

  free(DataRep);

  // prepare next request
  DataReq.Size=sizeof(TDataRequest)-4;
  DataReq.ID=0;
  DataReq.Period=1;

  for (i=0;i<NUMBER_ROBOTS_TEAM;i++)
  {
    result = Strategy->GetNextRequest(i,&mode,&flag,&x,&y,&theta);
    DataReq.Robot[i].Mode = mode;
    DataReq.Robot[i].Flag = flag;
    DataReq.Robot[i].SX = x;
```

```
    DataReq.Robot[i].SY = y;
    DataReq.Robot[i].Theta = theta;
    DisplayPosition->Lines->Add("X = "+(String)x);
    DisplayPosition->Lines->Add("Y = "+(String)y);
    DisplayPosition->Lines->Add("Theta = "+(String)theta);
  }

  // send next request
  ClientSocket1->Socket->SendBuf(&DataReq, sizeof(TDataRequest));
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::FormClose(TObject *Sender, TCloseAction &Action)
{
  Strategy->Free();
}
//---------------------------------------------------------------------------
```

8.9 USB camera comparison

		Pro (ISR)	**Express (du LAMI)**	**Home (JCZ)**
	Puce	Connectix	Logitech (NEW)	Logitech (NEW)
	Prix	Fr. 250.00	Fr. 90.00	Fr. 150.00
Essais avec GraphEdit	*Légende*	*Images/sec max.*	*Images/sec max.*	*Images/sec max.*
A l'EPFL (PII 266MHz / 32 MB ?)	Capture Filter Logitech	8.65	6.33	
	Capture Filter SDK (VidCap.ax)	6.04	4.92	
A la maison (K6II 350MHz / 64MB)	Capture Filter Logitech	8.96	7.38	**14.43**
	Capture Filter SDK (VidCap.ax)	8.35	7.38	14.6 (7.58 preview)
Distorsion de l'image		Convexe	Concave	Légèrement convexe
Retard approximatif (expérimentation subjective)		~0.3 sec	~0.1 sec	~0.1 sec (voire presque mieux que Express…)
Qualité de l'image (1 = meilleure note)		1	3	2
Flou lors des mouvements rapide (1=bon, 3=très flou)		2	3	1

Quick Cam Pro :

Quick Cam Home :

Quick Cam Express :

8.10 Disk containing all programs