Military Academy Of Tunisia

Royal Military Academy Of Belgium

Target Tracking For Robot Collaboration

Elaborated By

Officer Student

Wahiba JOMAA

Graduation Project Report (March - June 2008) submitted in the fulfillment of the requirements for the degree of Engineer on Computer Sciences

Promoter: Professor Yvan **BAUDOIN**

Supervisor: Ir. Geert **DE CUBBER**

Abstract

WAHIBA JOMAA.”Target Tracking for Robot Collaboration” Research Project Dissertation under the direction of Professor BAUDOIN.

Nowadays robots have reached a level of sophistication such that it is now productive to include robot-assisted search teams in urban search and rescue scenarios.

The perceptual interface to the human operator is often a video display that relays images from a camera on the robot. Research into issues concerned with human-robot interaction has suggested that if more intelligence were placed on the robot, it could lessen the load on the human operator and improve performance in the search task.

Within this thesis we focus on the relative performance in a search task in which the robot is controlled by a remote human operator.

The goal of this thesis is to enable the mobile robot CASTOR to follow an a priori defined target. This target can be any colored object (in the thesis a red one chosen). In the case of this application, the target object will be another (master) robot, called ROBUDEM. By implementing this robot following behavior, further robot collaboration tasks can be achieved.

Résumé

Wahiba JOMAA. "Target Tracking for Robot Collaboration", thèse du projet de recherche sous la direction du professeur BAUDOIN.

De nos jours les robots ont atteint un bon niveau de sophistication qui est maintenant productif et capable d’inclure les groupes de recherche des robots assistés dans des zones urbaines et des scénarios de sauvetage.

L’interface de perception pour l’opérateur humain est souvent un affichage vidéo qui récupère les images d’une caméra montée sur le robot. Les recherches sur des questions entourant les interactions entre l'homme et le robot ont suggéré que plus d'intelligence est développée pour le robot plus que la charge sur l'opérateur humain est diminuée et les performances dans la tâche de recherche sont améliorés.

Dans cette thèse nous avons essayé d’améliorer la performance relative à une tâche de recherche où le robot est contrôlé à distance par un opérateur humain.

L'objectif de cette thèse est de permettre au robot mobile CASTOR de suivre, à priori, une cible bien défini. Cette cible peut être n'importe quel objet coloré (dans cette application la couleur rouge est choisit). Dans le cas de cette application, l'objet cible sera un autre (master) robot, appelé ROBUDEM. En mettant en œuvre ce comportement de suivi du robot, des tâches de collaboration peuvent être atteints.

To my parents for their patience and their permanent support,

To my brothers Hamza, Bilel, Mabrouk and my sister Samira for their warm heartedness and love,

To my aunt Monia and her husband Kouthair,

To all those who saved no efforts to assist me to go forward towards success

Acknowledgments

I am grateful to Professor BAUDOUIN, the supervisor of my internship for allowing me to work with him on this project.
Special thanks to my promoter Mr. DE CUBEER for all guidance through my internship. I am in dept to him for his open door policy as well as his competence and experience. His guidance, encouragement and availability equipped me with patience and eagerness to meet my needs.
I like to express thanks to all the people who have contributed for the achievement of this work and who gave me the opportunity to enjoy this instructive and enriching in Belgium.
I want also to thank Miss VERBIEST for her help guidance during the work and for my English Teacher Ms KHEDHER for her help on the redaction of this report.
Last but definitively not least, I would to thank especially my parents and my friends who have stood by me in need.

Table of Contents

List of Figures

List of Tables

Chapter 1: Teleoperation Overview

I. Introduction

The major goals of robotics is to realize multipurpose service robots that can solve several complex tasks while acting within changing environments such as home infrastructure or outdoors. However robots still have neither creativity nor the ability to think. Therefore, robots will necessarily need to be supervised or directly teleoperated at some point.

Teleoperation of robotic systems in hostile and unknown environment is of particular importance in that it aids or replaces the manipulator in handling difficult and possibly dangerous activities.

II. Definitions

The term teleoperation refers simply to the operation of a vehicle or system over a distance. Broadly, understanding all interaction with a mobile robot is part of this definition. Traditionally, teleoperation is divided into direct teleoperation and supervisory control.

In direct teleoperation, the operator closes all control loops himself, where as in supervisory control a remarkable amount of the control is exercised by the teleoperator, i.e., the teleoperator is a robot.

The term teleoperation refers to direct teleoperation, while supervisory control is handled under human - robot interaction. Moreover, the term traditional teleoperation refers to direct teleoperation over a distance without a line of sight, but telepresence equipment is noticeable.

In today’s digital world, one has to note that even in the case of the direct teleoperation there usually exist control loops in the teleoperator. Typically these loops control the position or the velocity of the “directly” controlled actuators.

III. Introducing Teleoperated Robots

Many dangerous tasks are more and more expensive and painful for humans. To get to humans demining for example, it is rather slow, tedious, dangerous and expensive. The detection is not always reliable. Big effort is made to remove a single mine.

Robots do not require such a complex infrastructure to perform this task. Indeed, a robot could be cheaper to achieve some messy tasks in more than one dangerous environment than a human being, in space a robot costs less than Human. Once a robotic system is successfully built and tested, it can be duplicated easily at reduced costs. Human, on the other hand, is priceless. Robots can be used to take over the most dangerous jobs.

“Time is money”. Therefore Humans need time to be trained to accomplish some tasks and get some rest after wards. Robots are never tired and can be deployed 24 hours a day, provided there is enough energy.

Teleoperation platform allows a user to execute or to control remote tasks, i.e. without being present simultaneously where the action takes place. Tele-operation tries to minimize risks during dangerous works: spatial exploration, toxic device to operate…etc. It could also allow scientists to go where a human cannot: volcanoes, smart caves, mined areas…etc.

To help the user to accomplish such tasks, other users or an autonomous or teleautonomous robot could be used. Assistance robots complete human faculties and allow the system to take advantage over computer capacities to realize repetitive tasks, physically hard work, and to use at its best the expert dexterity to look, fall and react at the right time.

IV. A literature overview of the first Teleoperated machine

When the tasks to be done demanded better skills from robots, teleoperation was born. This is a new technique which combines human abilities and machine capacities.

The first teleoperated systems were created after the Second World War, to manipulate radioactive substances in nuclear industry. They were composed of two symmetrical arms, one of them (slave) reproduced the master's movements; the other arm (master) was controlled by a human operator

In 1951, in France, Raymond Goertz designs the first teleoperated articulated arm E-1(see figure1), for the Atomic Energy Commission. The design is based entirely on mechanical coupling between the master and slave arms (using steel cables and pulleys). Derivatives of this design are still seen in places where handling of small nuclear samples is required. This is generally regarded as the major milestone in force feedback technology.

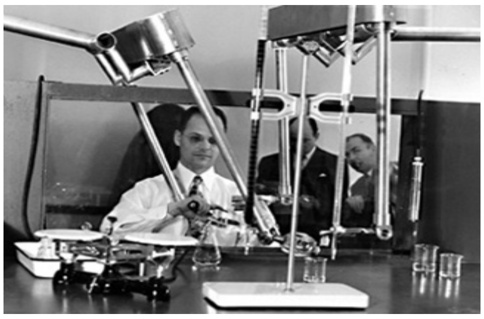


Figure 1 : E-1 by Goertz

V. Fields of Applications

Robots are designed for many purposes. In manufacturing, they are used for welding, riveting, scraping and painting. They are also deployed for demolition, fire and bomb fighting, nuclear site inspection, industrial cleaning, laboratory use, medical surgery, agriculture, forestry, office mail delivery as well as a myriad of other tasks. Increasingly, more artificial intelligence is being added. For example, some robots can identify objects in a pile, select the objects in the appropriate sequence and assemble them into a unit.

V.1. Submarines Field

Robots are mainly used for submarine mapping and exploration. VICTOR is an example of submarines; the vehicle can be reached 6000m depth in several times and used continuously during more than 70hours.

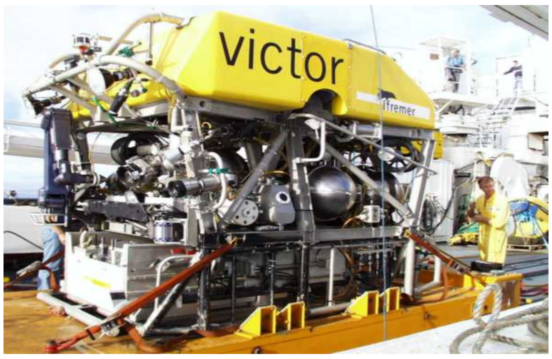


Figure 2 : Victor by GENAVIR

V.2. Space Field

Using robots in space exploration is perfect for teleoperation for more safety and the law costs. Many robots were designed for space discovery and since space is an unfamiliar environment the use of robots has been very useful and helpful.

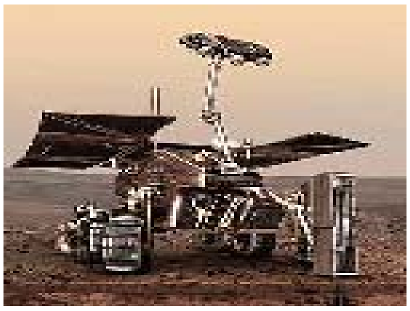


Figure 3 : Exomars by ESA

V.3. Military Field

Robots can be used underwater, on the air or on the ground. The most recurrent applications are: demining, mine detection, closed loop controlling, taking surveillance photographs, launching missiles at ground targets and many other tasks.

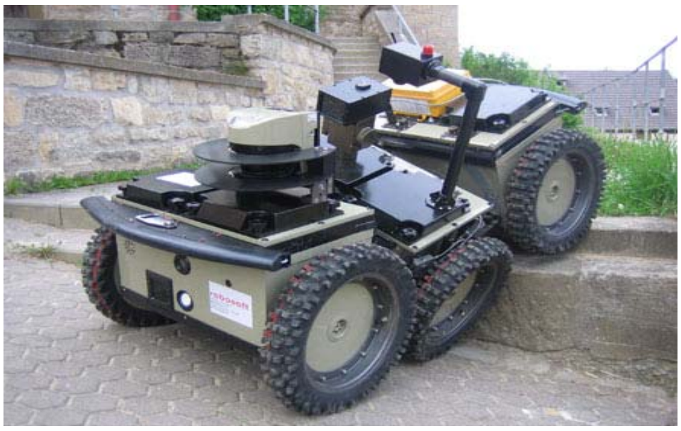


Figure 4 : Roburoc6 by Robosoft

V.4. Medical Field

Robots can be used in Endoscopic surgery, with micro mechanicals manipulations in which they minimize risks and damages, in Telesurgery, like that specialists can operate over distances (teleoperated robots).And they can also replace a member of the body.

This is an example of a prototype machine that sleeps and wakes up the patient alone, it has been invented by doctors at the hospital Foch in Suresnes.

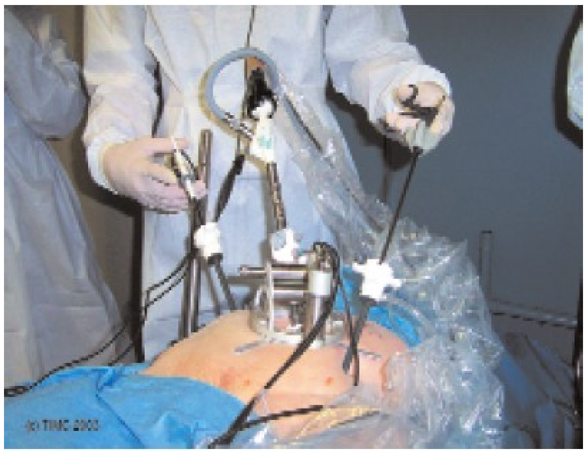


Figure 5 : The first robot anaesthetist

V.5. Industry Field

Robots are used in many sectors of industry, but the automotive industry is the leader in robot utilization and applications worldwide: parts assembling and production, cleaning, painting, welding of instruments….

Figure 6 : Example of robot application in the industry

VI. Thesis Goal

In this project, we present the implementation of the Hue Tracking model to the CASTOR platform based on teleoperation and using simple joystick interface as input device, and the experimental results using this model to teleoperate and control the CASTOR robot.

VII. Thesis Outline

This Chapter ketch the relevant background on teleoperation and robotics fields of application. Chapter 2 gives a general view of our system and some technical specifications of the robot. Chapter 3 discusses the target detection and representation issues, giving details of the color target detection, representation, noise suppression, target size evaluation and all related models of the tracking. Chapter 4 presents our approach to implement the tracking model on robot system. Appendices are added at the end of the dissertation for more details on some implementation aspects.

Chapter 2: System description

I. Introduction

This chapter is a description of the CASTOR robot, which is mainly a mobile intervention robot. It can be used in hostile environments, like for instance nuclear, chemical, bacteriological ... It can also be used in dangerous undertakings like demining, rendering explosives harmless...

This chapter describes the hardware and software aspects and specifications of the robot without going into too much detail.

II. Hardware

II.1 Parts

The most prominent parts are (see Figure7):

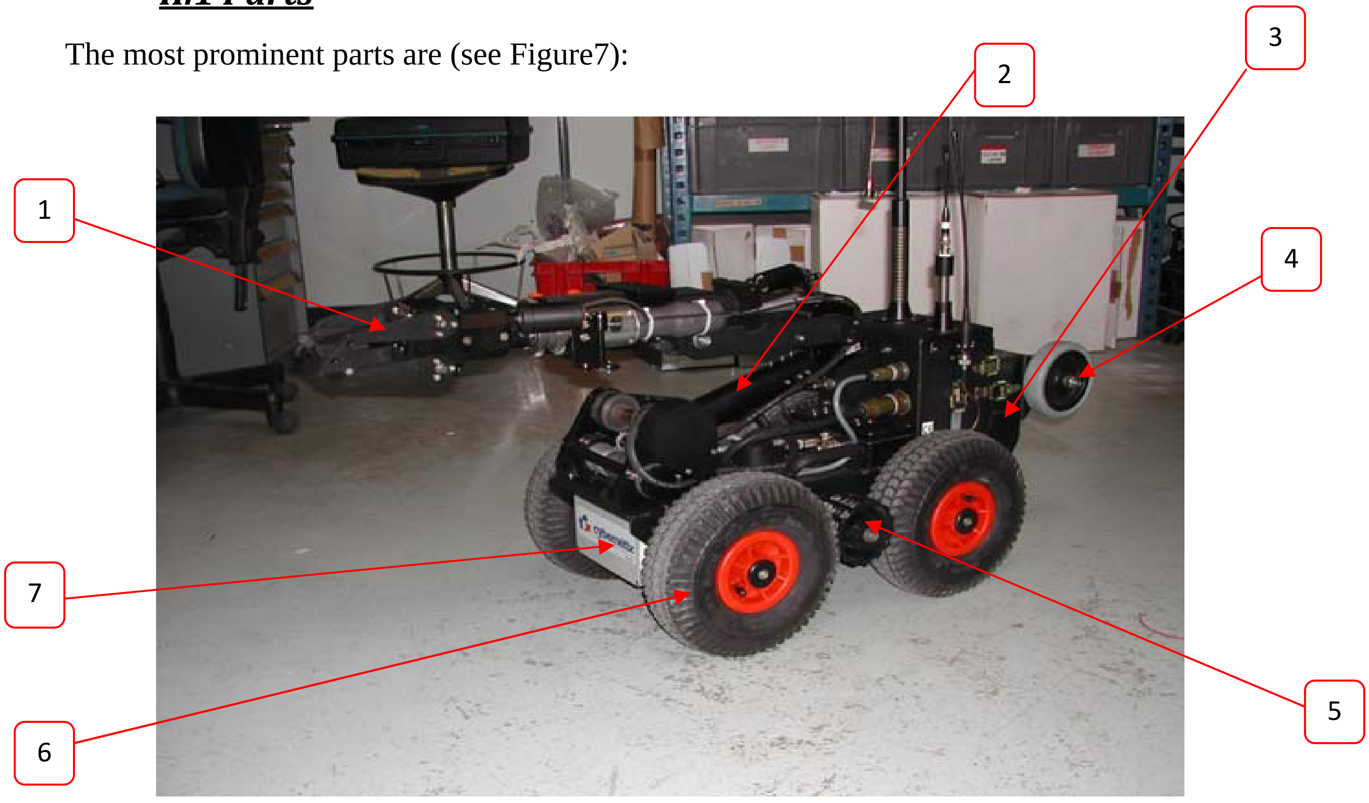


Figure 7: The CASTOR Robot

- (1) pair of pincers
- (2) arm
- (3) communication box: radio-communication + video-transmission (alternatively the rollable cable can be used for communication)
- (4) counterweight
- (5) motor
- (6) wheel (with caterpillar track)
- (7) box with electronics

II.2 Specifications

- Dimensions: length x width x height = 730 x 400 x 520 mm
- Weight:
  - Basic version: 47kg
  - Pair of pincers: 2,7 kg
  - Counterweight: 15 kg
- Propulsion: 4 wheels with diameter of 260 mm driven by electrical motors
- Energy: electrical energy provided by 2 rechargeable cadmium/nickel batteries, 24 volt consisting out of 12 elements of each 1.2 volt
- Communication:
  - range communication via radio: 300 m
  - length cable that is connected to the control panel and the robot: 125 m

  Radio Communication

  A half duplex digital connection with:
  - Power: 100m W
  - Symbols/sec: 4800 bauds
  - Frequency: 224 MHz

  This connection can be used in 10 channels around the 224 MHz frequency.

  Video Communication
  - High Frequency HF: 1.25 GHz
  - Power: 1 W

- Performance:
    - speed: from 0 to 50 m/min
    - maximum gradient: 25°
    - maximum attainable height: 860 mm
    - surmountable threshold: 120 mm
    - lifting capacity:
        - ✓ arm + pair of pincers half expanded: 10 kg
        - ✓ arm + pair of pincers completely expanded: 5 kg

II.3 Camera

There are several cameras (see Figures 8, 9):

- camera pair of pincers (1)
- rear side camera (2)
- steering camera (3)

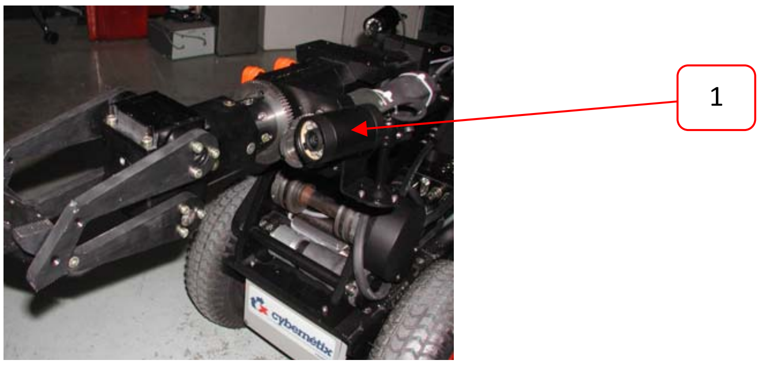


Figure 8: Camera pair of pincers

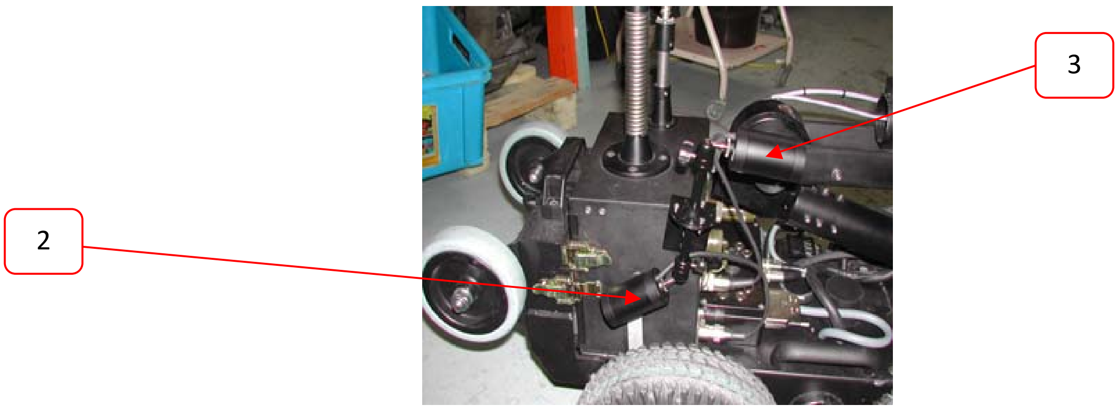


Figure 9: Steering camera and rear side camera

II.4 Arm

The arm is fixed on the platform for the accessories. The arm consists out of three prominent parts:

- The arm
- The forearm
- Equipped with the necessary to fix the pair of pincers. Optional is rotation of the pair of pincers. The pair of pincers can be manually attached in 5 different positions: -30°, 0°, 30°, 60°, and 90°.
  - ✓ Range arm: 0 to 210°
  - ✓ Range forearm: 0 to 230°

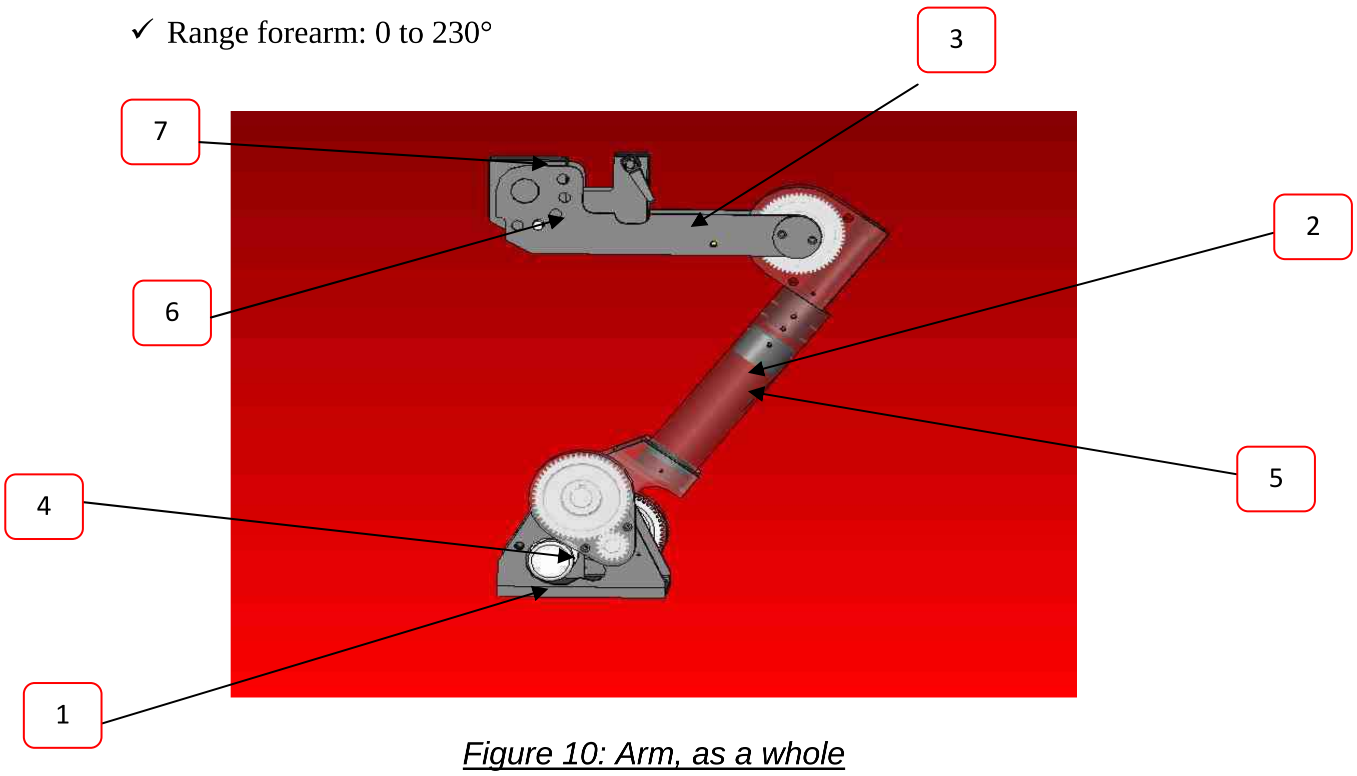


Figure 10: Arm, as a whole

- (1) Sub base fixing arm
- (2) Arm
- (3) Forearm
- (4) Moto-reducer arm
- (5) Moto-reducer of forearm (engine in the tube)
- (7) Pincer support

II.5 Communication Box

Parts (see figure 11):

- A rollable cable (rolls by certain command on the control panel)
- A transmitter for video-signals
- An antenna for radio-communication

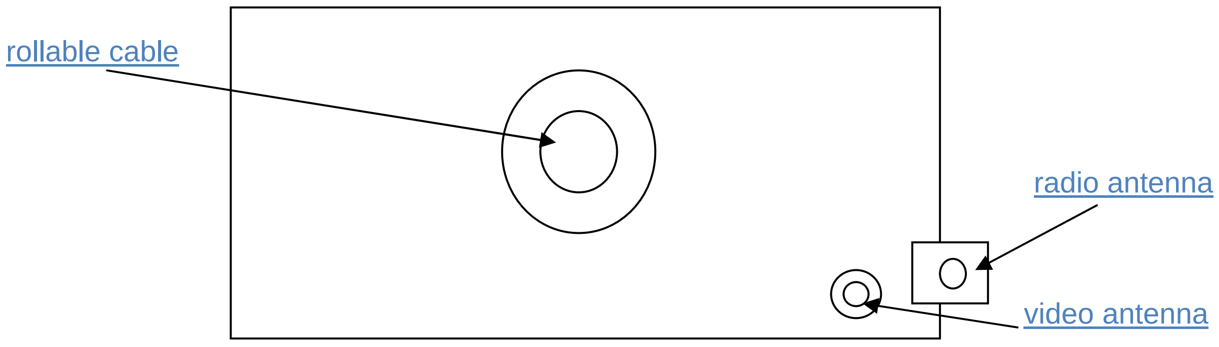


Figure 11: Communication box seen from the top

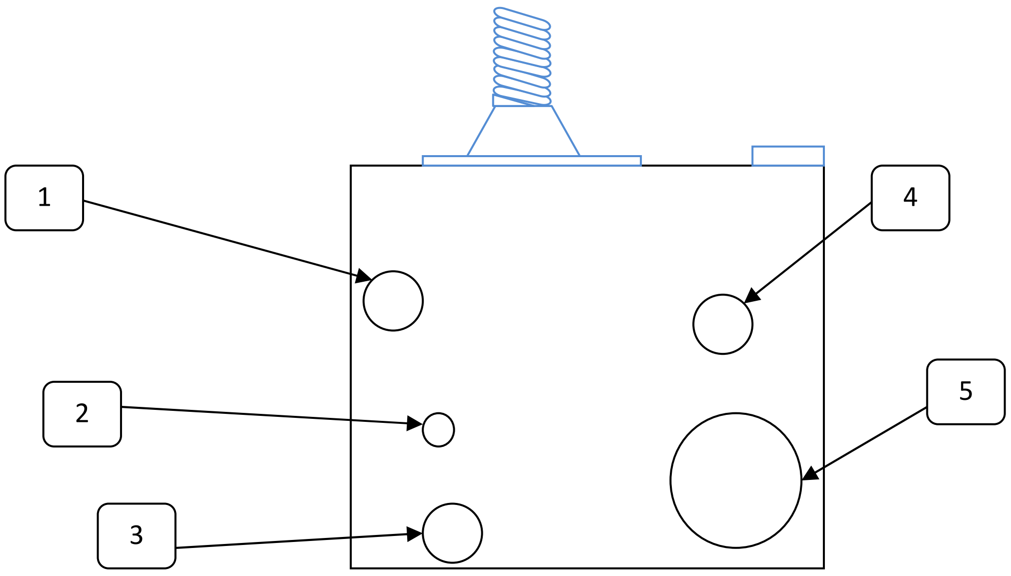


Figure 12: Communication Box Seen From the Front

The communication box is equipped with connections for the accessories: connection for the arm, the cameras, the electronics box (see Figure 12):

- (1) connection rear camera with zoom / pan / tilt (optional)
- (2) connection for rear camera
- (3) connection for extra rear camera
- (4) connection for the arm and (5) connection for the electronics box

II.6 Pair of Pincers

The pair of pincers is directly fixed on the extremity of the arm. The capacity of the opening is approximately 250 mm with a clamping force of 30 daN to 180mm opening. Parts (Figure 12):

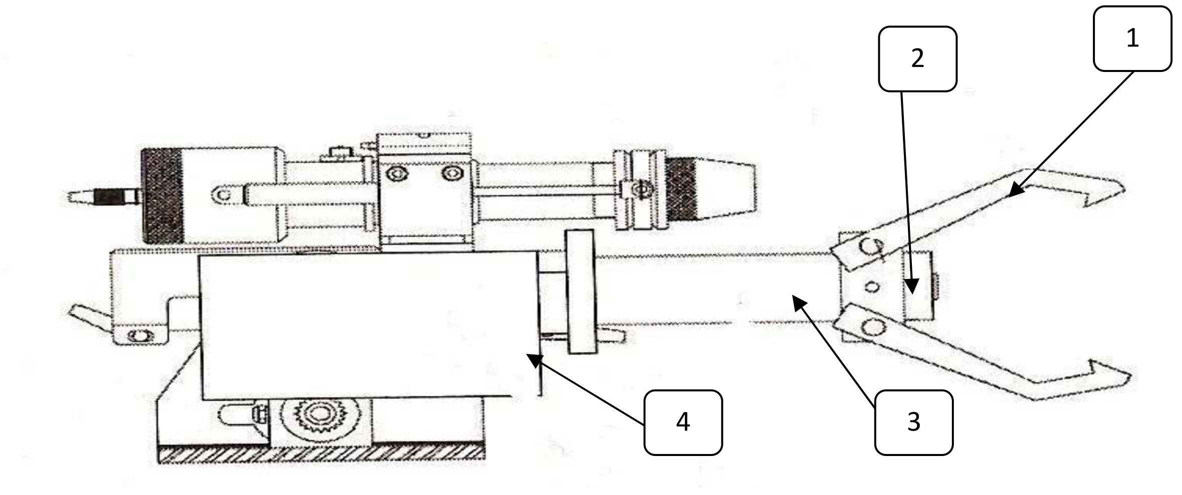


Figure 13: Pair of Pincers

- (1) the pair of pincers themselves
- (2) mechanism for open and closing
- (3) motor
- (4) motor for rotation

II.7 Control Panel Case

The case allows the control of the robot from a distance by means of a joystick (see Figure 14). The main parts of the case are made up out of a transmitter/receptor and a portable computer, with a 15 inch LCD TFT screen that visualizes the images sent from the robot by cable or HF (Radio).

As for the power supply there are two options: either the rechargeable battery (lead 12 V, 4.5 AH) incorporated in the case or the electrical mains of 220V.

- autonomy: 3 h
- weight: 13 kg

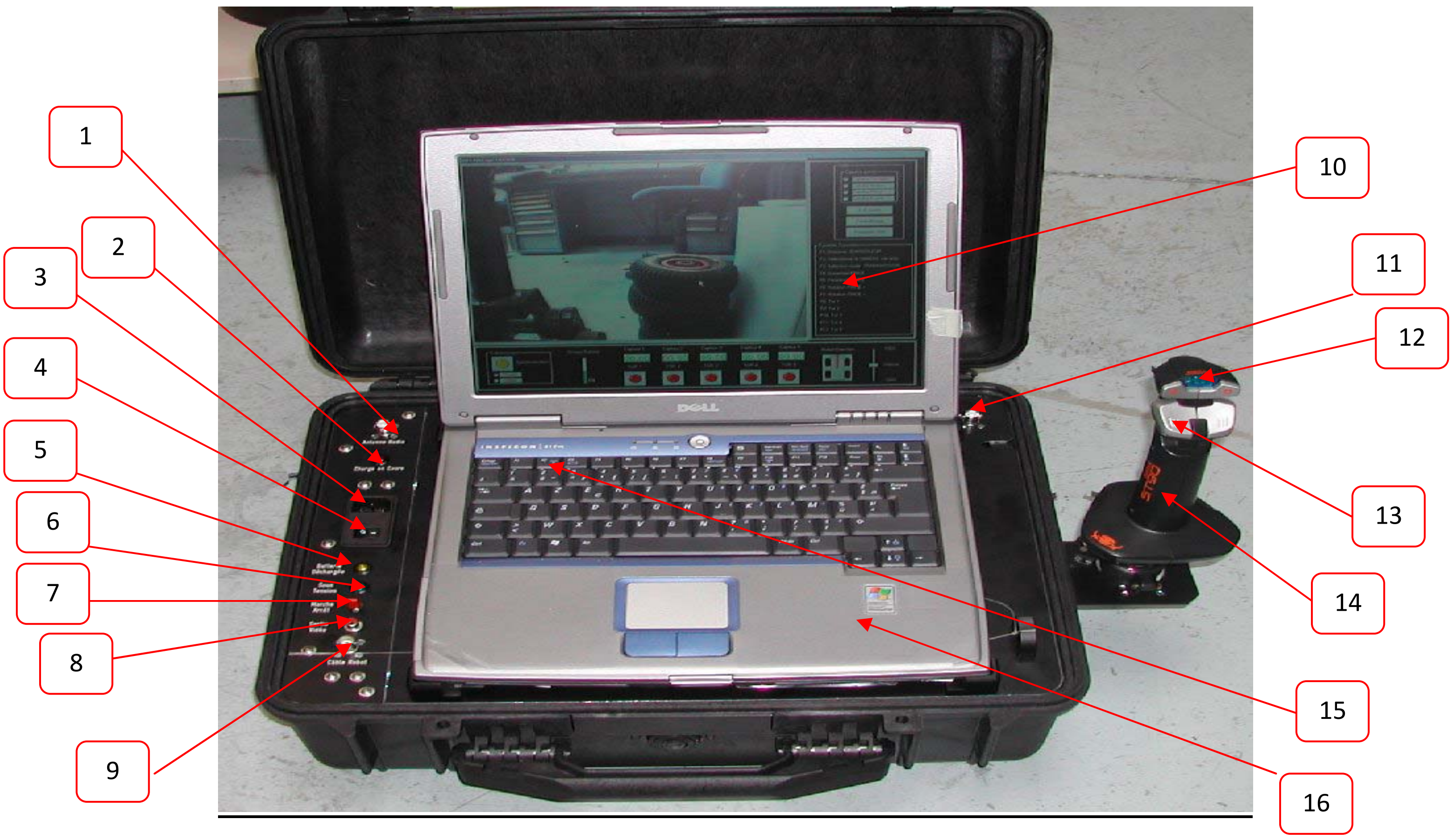


Figure 14: The Control Panel Case

- (1) Antenna radio
- (2) Led power under 220V and charge underway
- (3) Socket
- (4) Switch put in charge for the Control Panel Case under 220V
- (5) Led battery relieved (80%)
- (6) Powered Led
- (7) On/Off switch
- (8) Video release on RCA
- (9) umbilical cable connector
- (10) PC screen with HMI
- (11) Video antenna
- (12) Joystick
- (13) Joystick selection for locomotion
- (14) Joystick selection for the arm
- (15) Laptop and (16) key function to command the robot

Elements inside Control Panel Case:

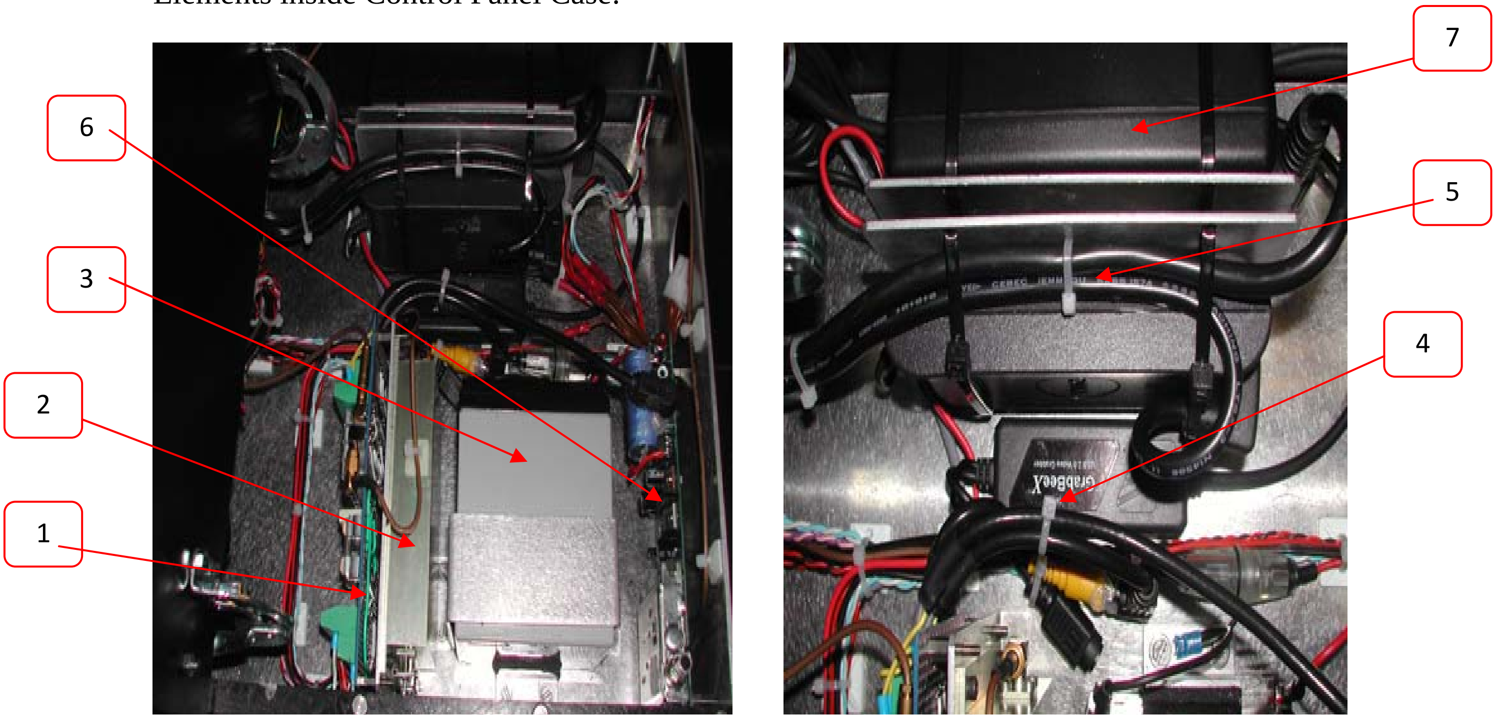


Figure 15: Inside Control Panel

- (1) interface card
- (2) modem radio
- (3) battery
- (4) USB grabber (permits to transform the video signal into a digital signal compatible with USB port)
- (5) PC charger
- (6) video receiver
- (7) Control Panel Case charger

II.8 Sensor Interface Box

The robot is equipped with a box where several sensors can be connected to, see Figure 16. These sensors are controlled from the control panel. The measurements made by these sensors are sent to the control panel and are processed there.

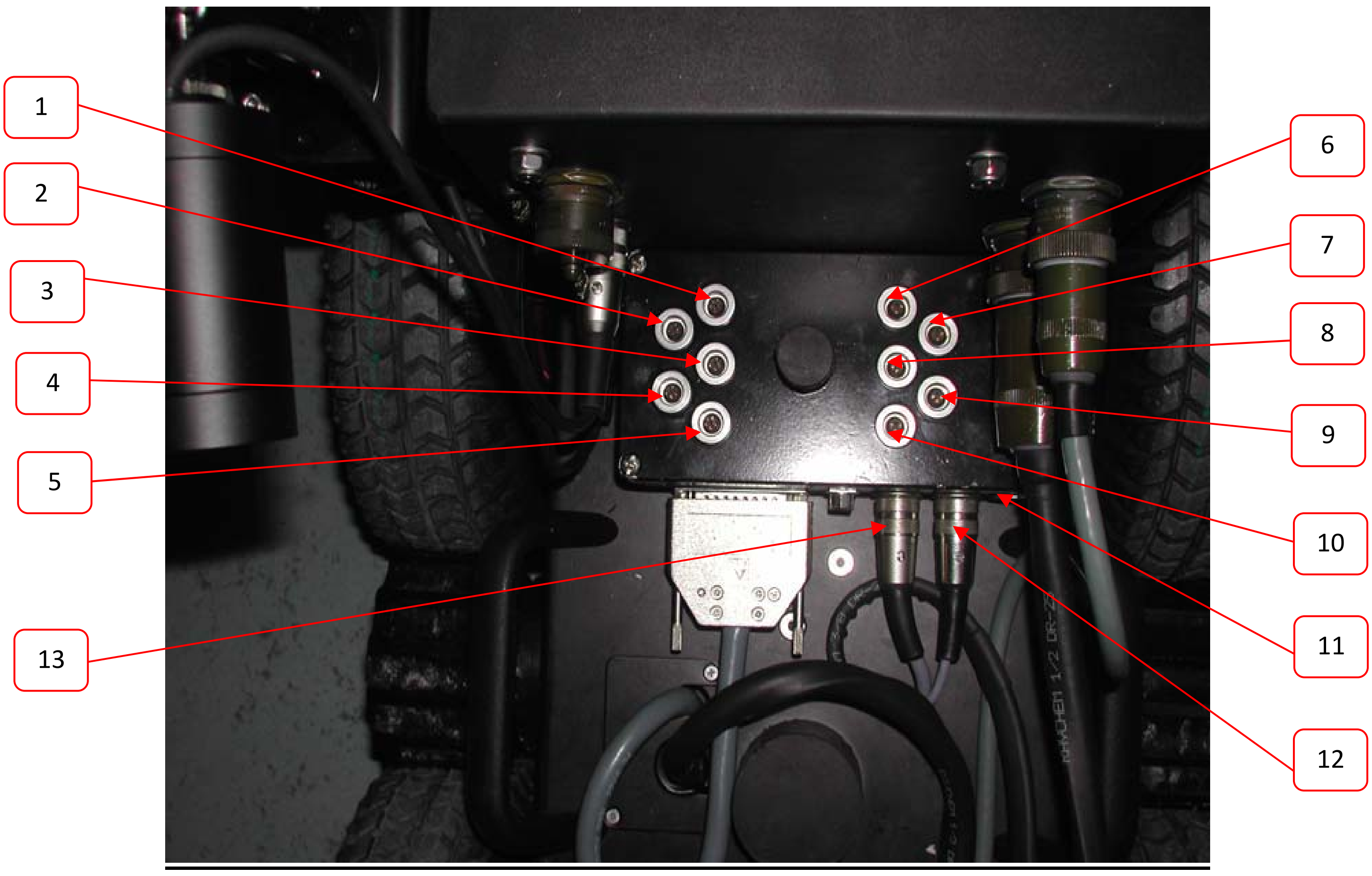


Figure 16: Sensor Interface Box

- (1) J22 connector 5 pts for activation of sensor N°1 (TOR1 on command post)
- (2) J23 connector 5 pts for activation of sensor N°2 (TOR2 on command post)
- (3) J24 connector 5 pts for activation of sensor N°3 (TOR3 on command post)
- (4) J25 connector 5 pts for activation of sensor N°4 (TOR4 on command post)
- (5) J21 connector 5 pts for activation of sensor N°5 (TOR5 on command post)
- (6) J29 connector 3 pts for analogue input sensor N°1 (Sensor 1 on command post)
- (7) J30 Connector 3 pts for analogue input sensor N°2 (Sensor 2 on command post)
- (8) J31 connector 3 pts for analogue input sensor N°3 (Sensor 3 on command post)
- (9) J32 connector 3 pts for analogue input sensor N°4 (sensor 4 on command post)
- (10) J33 Connector 3 pts for analogue input sensor N°5 (Sensor 5 on command post)
- (11) J28 free connector for 4th camera
- (12) J27 connector for camera pair of pincers
- (13) J26 connector for complete control of pincers

II.9 Box with electronics

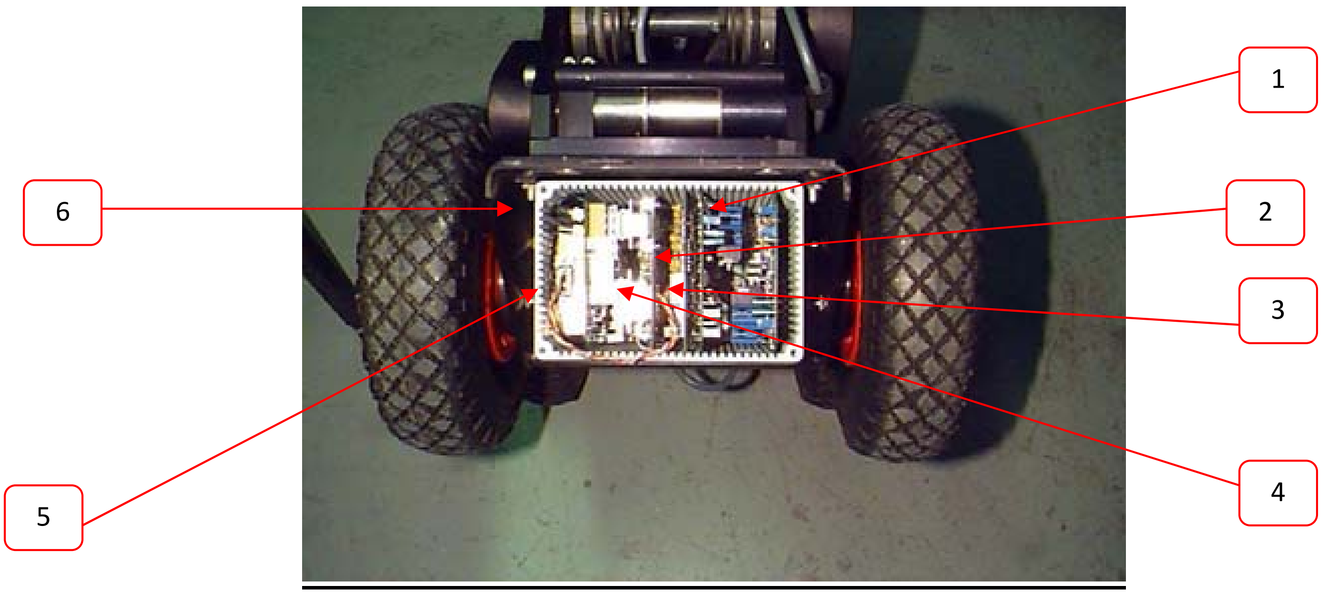


Figure 17:Elements inside electronics box

- (1) Robot movement card
- (2 Arm movement card
- (3) Microcontroller card
- (4) TOR Input/ output carda
- (5) Interface carda
- (6) Modem

II.10 Loader/Unloader

The loader / unloader allows recharging or unloading of two blocks batteries simultaneously. It works in charge with a common of 2.5A and cuts automatically by the end of load (still with small a common of maintenance).

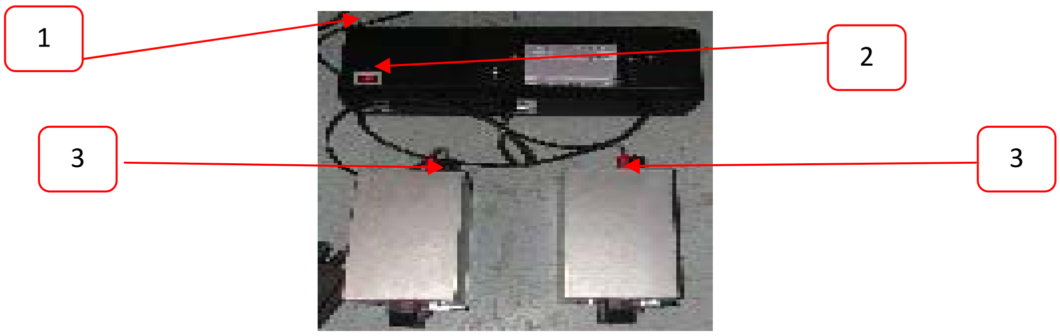
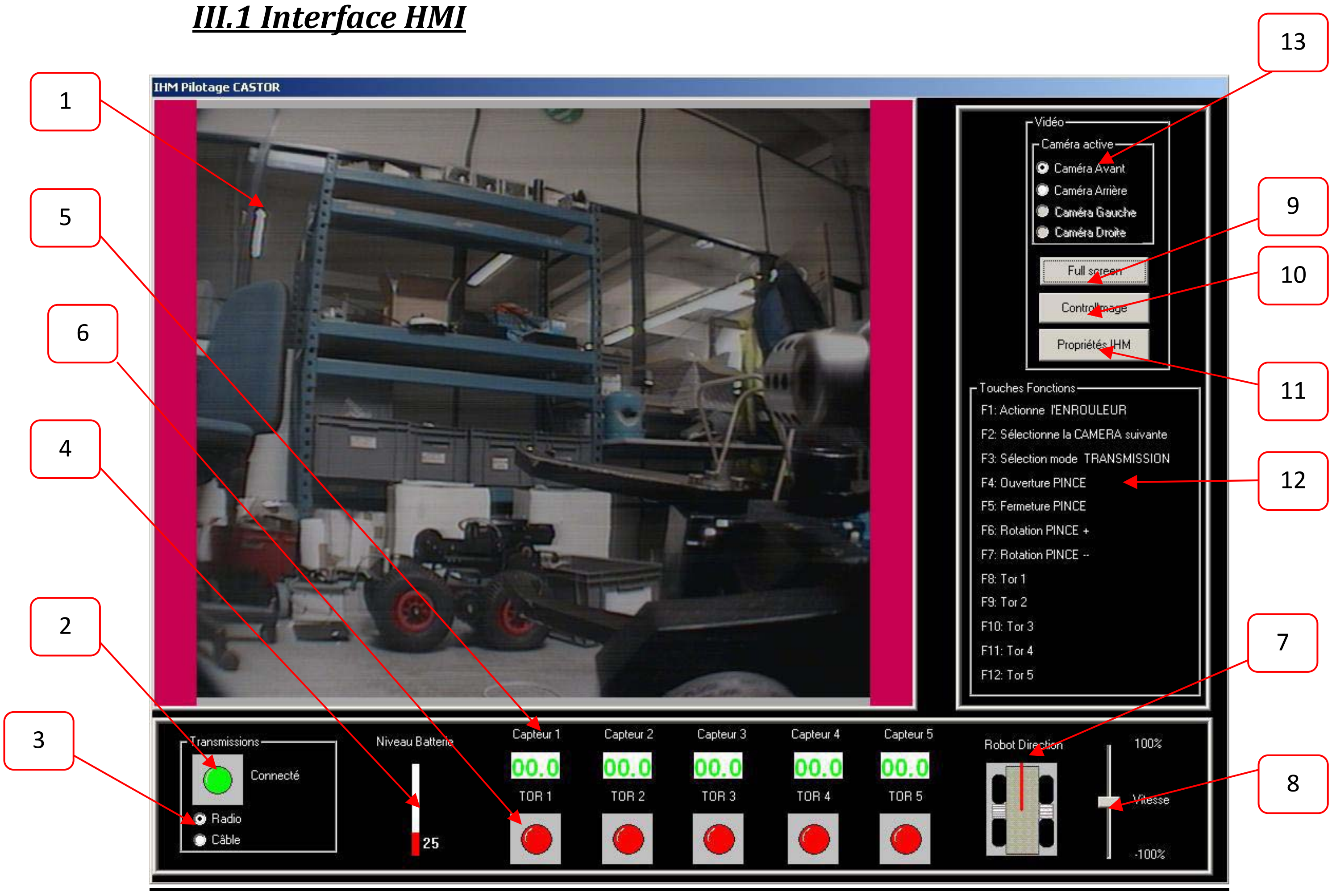


Figure 18: Loader/Unloader and two batteries

- (1) Sector connector
- (2) LED charging indicator
- (3) Load Connectors

III. Software

III.1 Interface HMI



Caméra active: [✓ Caméra Avant] Caméra Arrière Caméra Gauche Caméra Droite

Transmissions: [✓ Radio] Câble

Touches Fonctions

F1:	Actionne l'ENROULEUR
F2:	Sélectionne la CAMERA suivante
F3:	Sélection mode TRANSMISSION
F4:	Ouverture PINCE
F5:	Fermeture PINCE
F6:	Rotation PINCE +
F7:	Rotation PINCE --
F8:	Tor 1
F9:	Tor 2
F10:	Tor 3
F11:	Tor 4
F12:	Tor 5

Figure 19:The HMI Interface

- (1) video signal coming from the robot
- (2) indicator transmission state: synchronization / connection
- (3) indicator mode of transmission (radio or cable)
- (4) indicator battery level
- (5) 5 analogue values coming from the sensors on the robot
- (6) indicator of the state of (ON or OFF) the 5 devices
- (7) indicator angle of the displacement of the robot
- (8) indicator for the speed of the displacement of the robot (forward or backwards)

- (9) button to activate full screen
- (10) button to activate access to the settings of the video (hue, contrast, brightness, color)
- (11) button to activate access to the settings of the dimensions of the video signal and the serial port
- (12) function buttons action menu
- (13) indicator of the active camera on the robot

Function Buttons

- ✓ F1: Command to roll cable
- ✓ F2: Command to change camera viewpoint
- ✓ F3: Command for transmission mode: Cable or RF
- ✓ F4: Command to open pair of pincers
- ✓ F5: Command to close pair of pincers
- ✓ F6: Command to rotate pair of pincers in clockwise direction
- ✓ F7: Command to rotate pair of pincers in counter clockwise direction
- ✓ F8: Command to activate TOR 1
- ✓ F9: Command to activate TOR 2
- ✓ F10: Command to activate TOR 3
- ✓ F11: Command to activate TOR4
- ✓ F12: Command to activate TOR 5

III.2 Software Specifications

The software of the CASTOR robot consists of two distinct parts:

The software on the PC it allows sending information to control the robot. It also allows receiving and handling a bit stream coming from the robot which contains different measurements made by the sensors on the robot.

The embedded software it allows the control over the actions of the robot dependent on the information received from the control panel. It also allows sending a bit stream with the measurements from the sensors to the control panel.

III.2.1 Software on the PC

The software that governs the control over the robot is located on the PC (Windows XP) and consists out of an HMI written in C++, developed under Visual C++.

It visualizes the video in a window and represents the state of the robot by indicators or graph bars (level of the battery of the robot, quality of the radio-connection, values from the analogue sensors, etc…)

The software gets information from the command-buttons pushed on the keyboard and constructs the bit stream TC (Tele Command) that will be sent to the robot via the serial port. Similarly the PC will periodically receive the bit stream TM (Tele Measurement) coming from the robot, and display the information in this stream on the screen.

i. Different Classes

- *Class CCaptureVideo*: related to the video.
- *Class CControlImageDlg*: related to the settings for the video.
  - ✓ Hue
  - ✓ Saturation
  - ✓ Brightness
  - ✓ Contrast
- *Class CVisuRob*: required to display the angle of the robot orientation while moving.
- *Class CFullScreenDlg*: required to display the video in full screen.
- *Class CAfficheur*: required to display the analogue sensor values in the HMI interface.
- *Class CJauge*: required to display the bar graph for the battery level.
- *Class InterfComm*: related to:
  - ✓ Initializing communication
  - ✓ Building bit stream
  - ✓ Sending bit stream
  - ✓ Receiving bit stream, CRC
  - Function InitComm(): initializes communication
  - Function BuildTrame (BYTE * content): Builds the bit stream in the right format:
    - ✓ Heading
    - ✓ Data
    - ✓ CRC
  - Function SendTrame(): vehicles the data to the class PortSerie (function PortWrite(unsigned char* Byte, int nNumberOfBytesToWrite)) to handle the details of data sending.

- Function ReceiveTrame(): Gets a hold of the data coming from the robot through the class PortSerie (function PortRead(unsigned char* Data, int nb_read_need, int& nb_read)) and Checks the validity of the data through CRC.

- *Class PortSerie*: Deals with communication through the serial port:
    - ✓ initializing port (Portlnitialize(...))
    - ✓ Writing (PortWrite(...))
    - ✓ Reading (PortRead(...))
- *Class CIHM_PilotApp*: Defines and initializes the application.
- *Class ClHM_PilotDlg*:
    - Initializes the main dialog
    - Processes:
        - ✓ the joystick input
        - ✓ the function buttons
        - ✓ the values of the analogue sensor entries
        - ✓ other buttons pushed on the dialog (FullScreen, Properties Video, Settings Dimensions + Communication Port)
    - Function OnInitDialog():
        - ✓ Reads parameters out of configuration file: LitParametres()
        - ✓ Defines joystick
        - ✓ Initializes video
        - ✓ Initializes communication port
        - ✓ Initializes interface
        - ✓ Starts thread :InterfaceComThread = HANDLE_begin thread (startlnterfaceCom, ...)
    - Function OnTimer(UINT nlDEvent): tasks:
        - ✓ To refresh joystick
            - speed + direction determined
            - movement arm + forearm determined
            - this information is stored in the bit stream being sent to the robot
        - ✓ which function buttons are being pushed
            - every button is tested
            - accordingly the bit stream is adjusted

- ✓ The HMI interface reflects the above information by means of several indicators
- ✓ After the data are collected they are copied in the global variable Global_OutData; the thread startInterfaceCom (LPVOID lp) will send these data to the robot.
- ✓ The bit stream coming from the robot is received and will be handled here, the analogue values are extracted from the bit stream and displayed on the interface
- ✓ The function is called every 200 ms

- *Global function*: startInterfaceCom(LPVOID lp)
  - o Sending and receiving bit streams
    - ✓ For synchronization
    - ✓ For data exchange
  - o Thread: Constant loop of
    - ✓ Building bit stream with Global_OutData
    - ✓ Sending bit stream
    - ✓ Receiving bit stream

ii. Most Important Files

- *IHM_Pilot.h*: it is the main header file for the application. It includes other project specific headers and declares the **CIHM_PilotApp** application class.
- *IHMPilot.cpp*: it is the main application source file that contains the application class **CIHM_PilotApp**.
- *IHM_PilotDlg.h*: it contains the declaration of one's CIHM_PilotDlg class. This class defines the behavior of your application's main dialog.
- *IHM_PilotDlg.cpp*: This file contains the definition of one's **CIHM_PilotDlg** class. This class defines the behavior of one's application's main dialog. Its important functions are: **OnlnitDialog() OnTimer(UINT nIDEvent).**
- *CaptureVideoSimple.h*: contains declarations of: class **CCaptureVideo** class **CWndCaptureVideo.**
- *CaptureVideo.cpp*: contains definition of: class **CCaptureVideo**
- *InterfComm.h*: contains declaration of: class **CinterfComm**, global variable **InterfaceComThread**

- *InterfComm.cpp*: contains definition of: class **CinterfComm**
- *InterfaceCommunication.cpp*: contains definition of: global function (thread): **startlnterfaceCom(LPVOID lp)**
- *PortSerie.h*: contains declaration of: class **PortSerie**
- *PortSerie.cpp*: contains definition of: class **PortSerie**

III.2.2 Bit streams Exchanged between the Control Panel and the Robot

The control panel sends a bit stream to the robot every 200 ms. When the robot receives this bit stream, a response bit stream is sent from the robot to the control panel.

i. Bit stream sent from the control panel to the robot:

This bit stream consists out of 11 octets:

- 2 octets in the beginning of the stream
- 4 octets for analogue entries (on the side of the PC)
- 3 octets for ON / OFF entries (on the side of the PC)
- 2 octets for Cyclic Redundancy Check (CRC)

In more detail:

- Octet n°1: hexadecimal value: 0x12
- Octet n°2; hexadecimal value: 0x34
- Octet n°3: Analogue entry: instruction forearm
  Value coming from the joystick, X-axis, right button pushed. Represented by 8 bits.
- Octet 0°4: Analogue entry: instruction arm
  Value coming from the joystick, Y-axis, right button pushed. Represented by 8 bits.
- Octet n°5: Analogue entry: instruction forwards / backwards
  Value coming from the joystick, Y-axis, left button pushed. Represented by 8 bits.
- Octet n°6: Analogue entry: instruction left / right
  Value coming from the joystick, X-axis, left button pushed. Represented by 8 bits.

- Octet n°7: TOR 1: ON / OFF states

  - ✓ Bit 7: Brake (no brake=0, brake=1)
  - ✓ Bit 6 :Active
  - ✓ Bit 5 : Direction
  - ✓ Bit 4: Pan / Tilt / zoom 1 & 2/ Pincers / Rotation pincers control referenced motors
  - ✓ Bit 3: Pan / Tilt / zoom 1 & 2 / Pincers / Rotation pincers
  - ✓ Bit 2 : Pan / Tilt / zoom 1 & 2 / Pincers / Rotation pincers

  control referenced motors

    - 00xxx = motors pincers, rotation pincers, pan, tilt. zoom 1, zoom 2 **deactivated**
    - 10000 = **motor pincers** activated in **opening mode**
    - 11000 = **motor pincers** activated in **closing mode**
    - 10001 = **rotation motor** pincers activated in **clockwise mode**
    - 11001 = **rotation motor pincers** activated in **counter clockwise mode**
    - 10010 = motor **zoom 2** activated in **clockwise mode**
    - 11010 = motor **zoom 2** activated in **counter clockwise mode**
    - 10011 = motor **zoom 1** activated in **clockwise mode**
    - 11011 = motor **zoom 1** activated in **counter clockwise mode**
    - 10100 = motor **tilt** activated in **up mode**
    - 11100 = motor **tilt** activated in **down mode**
    - 10110 = motor **pan** activated in **clockwise mode**
    - 11110 = motor **pan** activated in **counter clockwise mode**

  - ✓ Bit 1: Selection camera
  - ✓ Bit 0: Selection camera

  Selection

  00 = camera n°1 pincers
  10 = camera n°2 extra
  01 = camera n°3 steering
  11 = camera n°4 rear

- Octet n°8: TOR 2: ON / OFF states

  - ✓ Bit 7: TOR 4 OFF=0 / ON=1
  - ✓ Bit 6: TOR 2 OFF=0 / ON=1
  - ✓ Bit 5: roll cable = 1 / Deactivate roll cable = 0
  - ✓ Bit 4: TOR 1 OFF=0 / ON=1
  - ✓ Bit3:TOR 3 OFF=0 / ON=1
  - ✓ Bit 2: validation tir, set to 1

- ✓ Bit 1: TOR 5 OFF=0 / ON=1
- ✓ Bit 0: RF mode= 0 / Cable mode = 1

- Octet n°9: TOR 3: ON / OFF states
- Octet n°10: CRC
- Octet n°11: CRC

The CRC octets are calculated as follows: CRC with 16 bits = Octet 1 + Octet 2+…+ Octet 9

Duration of the bit stream TC:

- ✓ 4800 baud/s
- ✓ 1 symbol = 1 bit
- ✓ 4800 bps
- ✓ 11 octets = 11 x 8= 88 bits
- ✓ Duration = 88/4800~18 ms

ii. Bit stream sent from the robot to the control panel

This bit stream consists out of 14 octets:

- 2 octets in the beginning of the stream
- 9 octets for the analogue entries (on the side of the robot)
- 1 octet for ON / OFF entries (on the side of the robot)
- 2 octets for Cyclic Redundancy Check (CRC)

In more detail:

- Octet n°1: hexadecimal value: 0x12
- Octet n°2: hexadecimal value: 0x34
- Octet n°3: **Value battery**

  12 bits are used to represent the value: the 8 MSB here and the 4 LSB in octet n°9
- Octet n°4: **Value analogue entry 1**

  12 bits are used to represent the value: the 8 MSB here and the 4 LSB in octet n°9
- Octet n°5: **Value analogue entry 2**

  12 bits are used to represent the value: the 8 MSB here and the 4 LSB in octet n°10
- Octet n°6: **Value analogue entry 3**

  12 bits are used to represent the value: the 8 MSB here and the 4 LSB in octet n°10
- Octet n°7: **Value analogue entry 4**

  12 bits are used to represent the value: the 8 MSB here and the 4 LSB in octet n°11

- Octet n°8: **Value analogue entry 5**

  12 bits are used to represent the value: the 8 MSB here and the4 LSB in octet n°11
- Octet n°9: **Value battery** (4 LSB)/**Value analogue entry 1**(4 LSB)
  - ✓ Bit 0 to 3 = 4 LSB **value battery**
  - ✓ Bit 4 to 7 = 4 LSB **value analogue entry 1**
- Octet n°10: **Value analogue entry 2** (4 LSB)/ **Value analogue entry 3** (4 LSB)
  - ✓ Bit 0 to 3 = 4 LSB **value analogue entry 2**
  - ✓ Bit 4 to 7 = 4 LSB **value analogue entry 3**
- Octet n°11: **Value analogue entry 4** (4 LSB) / **Value analogue entry 5** (4 LSB)
  - ✓ Bit 0 to 3 = 4 LSB **value analogue entry 4**
  - ✓ Bit 4 to 7 = 4 LSB **value analogue entry 5**
- Octet n°12: Not used, Bit 0 to 7 set to 0
- Octet n°13: CRC
- Octet n°14: CRC

The CRC octets are calculated in the following manner: CRC with 16 bits = Octet1+Octet2+ … + Octet 12

<u>*Duration of the bit stream TM:*</u>

- ✓ 4800 Baud/s
- ✓ 4800 symbol = 1 bit
- ✓ 4800 bps
- ✓ 14 octets = 14 x 8 = 112 bits

Duration = 112/4800~ 23 ms

Duration TC: 18 ms

Duration TM: 23 ms

Frequency thus is 1/ (23+18) ~24 Hz

iii. <u>*Calibration Analogue Values*</u>

For the calibration of the analogue entries the following procedure must be followed:

Calculate the gains A, B, C to obtain a measurement curve following the following equation:

$y=Ax^2+Bx+C$

With: x, the value of potential difference volt

y, the output in the desired measurement unit

The gains are then multiplied as follows:

$A=A*Coeff^2$

$B = B*Coeff$

$C=C$

The coefficient multiplication depends on the resolution of the analogue entries:

For example, when 12 bits are used:

Analogue		**Resolution = Coeff**
	0 to +5 V =>	$5 / 2^{12}$
	0 to +10 V =>	$10 / 2^{12}$
	-5 to +5V =>	$10 / 2^{12}$
	-10 to +10V =>	$20 / 2^{12}$

- *Config_IHM.txt*: file with parameters for the configuration of the analogue entries between 0 and +/- 10 V.
- *Config_IHM_-10_+10.txt*: file with parameters for the configuration of the analogue entries between -10 and +10 V.

III.2.3 Input/output Cards

i. Input/output Card Robot:

Configuration Card:

- ✓ Port A output
- ✓ Port B input
- ✓ Port C low when input, high when output

Configuration word: 0x81 = 1000 0001b

	Bit n°	
PORT A:	0	Bit 1 (MSB) selection camera
	1	Bit 0 (LSB) selection camera
	2	relay communication RF/Cable
	3	relay cut video
	4	free output
	5	free output
	6	Command roll back
	7	free output

	Bit n°	
PORT B:	0	Command break
	1	Bit 4 P/T/Z/P/O (pan/tilt/zoom/orientation)
	2	Bit 3 P/T/Z/P/O
	3	Bit 2 P/T/Z/P/O
	4	Bit 1 P/T/Z/P/O
	5	Bit 0 P/T/Z/P/O
	6	free output
	7	to be forced to 1

	Bit n°	
PORT C:	0	Not used
	1	Not used
	2	Not used
	3	Not used
	4	Break orientation pair of pincers
	5	to be forced to 0
	6	free output
	7	to be forced to 0

<u>Analogue entries</u>

Channel n° 0 Level battery robot (0/10V)

1 Entry sensor n°1 (0/10V)

2 Entry sensor 0°2 (0/10V)

3 Entry sensor n°3 (0/10V)

4 Entry sensor 0°4 (0/10V)

5 Entry sensor 0°5 (0/10V)

6 Not used

7 Not used

<u>Analogue outputs</u>

Channel n° 1 Instruction forward / backwards

2 Instruction left / right

3 Instruction arm

4 Instruction forearm

ii. Input/output Card PC

The **entries** that need to be managed on the PC side are:

- Joystick (port USB 1)
  - X-axis for the direction of the robot, or the control over forearm
  - Y-axis for moving the robot forward or backwards, or the control over the arm
- Left joystick button for the control over the movement of the robot
- Right joystick button for the control over the arms
- Video (port USB 2)
- Function buttons

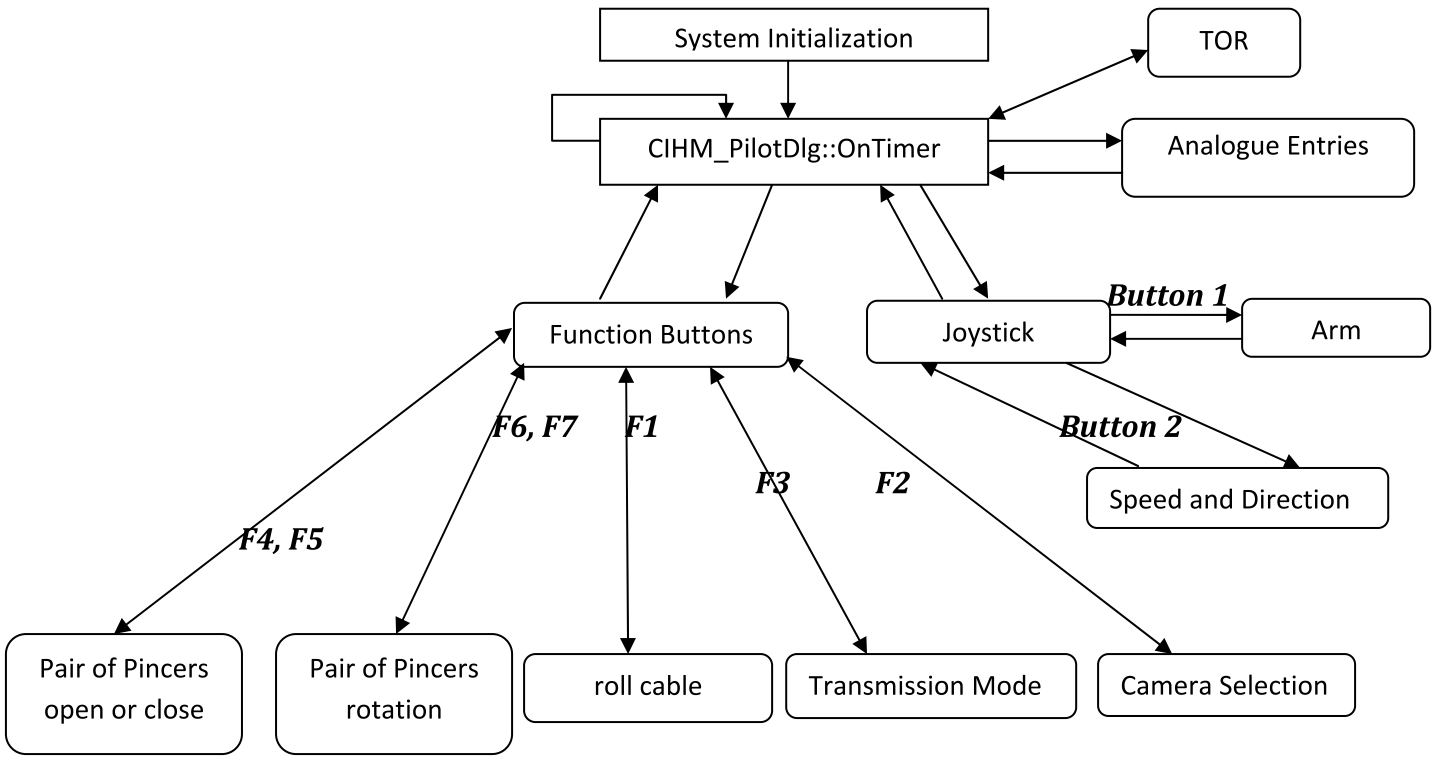
The **outputs** that need to be managed on the PC side are:

- Displaying the video in a window or in full screen
- Displaying the battery level of the robot on an analogical bar graph with 100%=27 V to 10%=24 V.
- Displaying the 5 analogue sensor values from the with 4 digits
- Displaying by means of indicators the ON/OFF states of the devices connected to the interface box (Red=OFF, Green=ON) TOR1... TOR5.

IV. Operation of the program

The next graph explains the program running, the system initialization (video, port, interface initializations…etc) are made first, than the program is running in a loopback, this is the manual mode (the control of the robot is made by the operator):
The operator:

- selects the camera
- selects the mode of transmission
- Controls the pair of pincers
- Controls the direction and the speed with the joystick



V. Conclusion

The CASTOR robot is a teleoprated platform, based on the continuous exchange of video frames with the Control Panel which makes the control of the robot more easier and allows the use of the robot in different tasks specially when the task take place where the operator shouldn’t or can’t go.

Chapter 3: Hue tracking overview

I. Introduction

For our application the color of the target is used as criterion of recognition, thus the HueTracker project is implemented to our program. This chapter is a description of the methodology adopted on this project. The approach of color detection, target parameters (center, size, position…) estimation and noise suppression will be explained.

II. Color Target Detection and Parameters Estimation

Many methods have been developed for target recognition and tracking purposes: the active contour method [1], the optical flow based method [2], the color based method [3] [4]. For the HueTracker project the advantage of color image is taken and color was used as a feature for recognizing the target. Color target detection and description could be done using either edge detection methods or region segmentation approaches. The digital color image pixel is considered as a vector, the digitalized color image as a vector field, and the edge detection has been made in a vector space. For this application, the HUE was used as chromatic identification criteria, and simplified the identification and the segmentation process for saving the computational cost, in order to meet the requirements of a real-time application. [5]

I.1 Color Models

To represent a colored image, several models are available, the three most widely used being: RGB1, YIQ2 and HIS3. [6][7]

I.1.1 RGB Color Model

The most frequently used color model is the RGB color model. In this model, each color image pixel signal is decomposed into three primary color components: Red, Green and Blue. Each primary color component includes both color and intensity information that are mixed together. One color is represented as a vector in the RGB space, as shown in Figure 20. This color model is often used in image display on a color monitor.

[1] Stands for Red, Green and Blue
[2] stands for Y-the luminance component, I-the in phase chromatic component and Q-the quadrature chromatic component.
[3] stands for Hue, Lightness and Saturation.

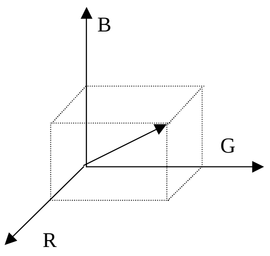


Figure 20 : RGB Color Model

I.1.2 YIQ Color Model

The second model is the YIQ color model, which is used in commercial color TV broadcasting. Basically, YIQ is a recoding of RGB for ensuring the transmission efficiency (because it transmits two chromatic signals, instead of three, it uses the same bandwidth of frequency) for maintaining the compatibility with monochrome TV standards and also the color information. The principal advantage of the YIQ model in image processing is that the luminance (Y) and the color information (I and Q) are decoupled. The relationship between the RGB model and the YIQ model is defined as follows:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (1)$$

I.1.3 HIS Color Model

In the HIS color model, the characteristics used to distinguish one color from another are brightness (I), hue (H), and saturation (S). Brightness embodies the chromatic notion of intensity. Hue is an attribute associated with the dominant wavelength in a mixture of light waves. Saturation refers to the relative purity of the amount of white light mixed with a hue. The model is shown in Figure 21:

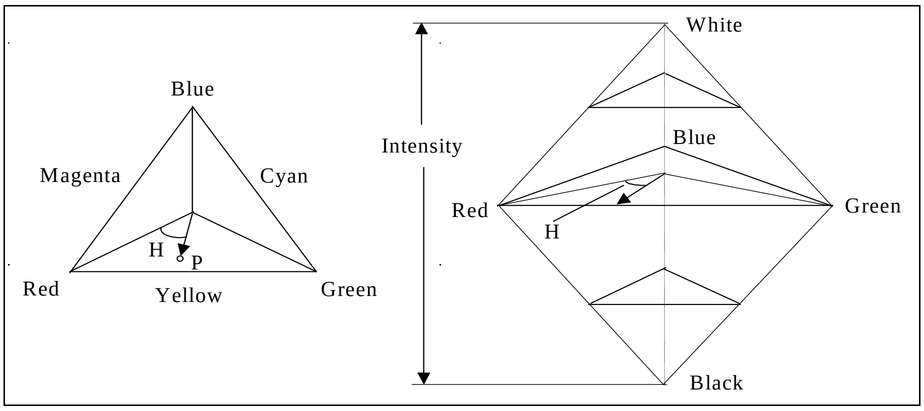


Figure 21: HIS Color Model

The hue value is represented as the angle between the pure red color and the color of interest. From Figure22 we can clearly see that a constant hue value (a constant angle value), corresponds in fact to a vertical plane in the model. This angle value is independent from the intensity or lightness (the height of the solid in Figure21 or the height of the horizontal plane in Figure 22) of the color of interest; it is also independent from the saturation (the length of the vector in Figure21 or the diameter of the cone in Figure 22) of the color of interest.
The relationships between the HIS model and the RGB model are defined as follows [6]:

$$I=\tfrac{1}{3}\left(R+G+B\right) \tag{2}$$

$$S=1-\frac{3}{(R+G+B)}\left[\min\left(R,G,B\right)\right] \tag{3}$$

$$H=\begin{cases}\cos^{-1}\left\{\dfrac{\tfrac{1}{2}\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{1/2}}\right\} & B\le G\\ 2\pi-\cos^{-1}\left\{\dfrac{\tfrac{1}{2}\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{1/2}}\right\} & B>G\end{cases} \tag{4}$$

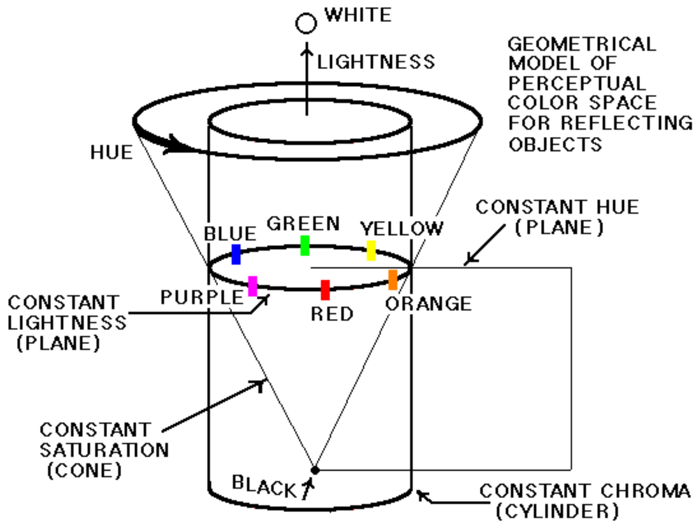

WHITE
LIGHTNESS
GEOMETRICAL MODEL OF PERCEPTUAL COLOR SPACE FOR REFLECTING OBJECTS
HUE
CONSTANT HUE (PLANE)
BLUE
GREEN
YELLOW
PURPLE
RED
ORANGE
CONSTANT LIGHTNESS (PLANE)
CONSTANT SATURATION (CONE)
BLACK
CONSTANT CHROMA (CYLINDER)

Figure 22 : Geometrical Model of Perceptual Color Space

I.2 Color Target Detection

The hue describes the color information. Using it for target detection the influence of intensity and saturation can be reduced. The major problem in using the HIS color space for target description and detection is the computational time needed to transform the original (acquired) image from the RGB space to the SHI space. In the Hue Tracking project a method using the hue information for target detection is proposed without any color space transformation.

I.2.2 Object's Hue

As it can be seen from Figures 21 and 22, a pure color can be clearly defined by a single hue value. However, due to the non-uniformity of the reflection and absorption properties of light falling colored surface, a range of hue values representing an interval is needed to represent a given color. The hue interval for a given color could be represented by the minimum hue value $(H_{\min})$ and the maximum hue value $(H_{\max})$. Each color may be parameterized by the two planar surfaces, in the HIS space, defined by $(H_{\min})$ and $(H_{\max})$. These mentioned planar surfaces, Figure 23, could be described directly in the RGB space.

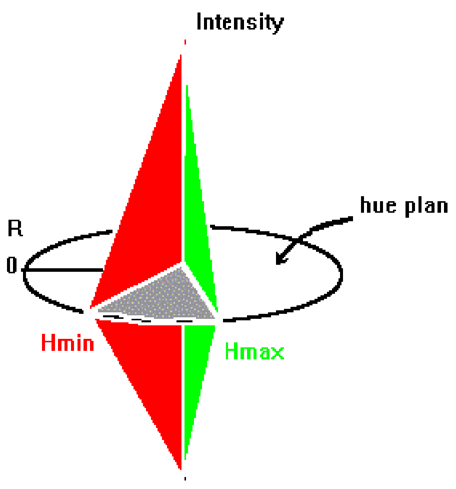


Figure 23 : Hue Section

Figure 23 shows the RGB histogram of a red color, as it can be seen, the distribution is elongated in the R direction.

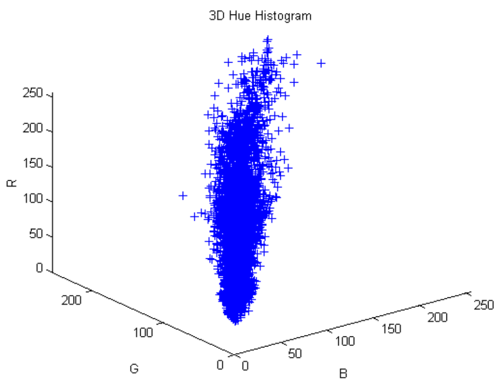


Figure 24 : RGB Histogram of Red Sphere

Figure 24 shows the RGB values of a red sphere. Notice that, the Green and Blue values are nearly equal.

I.2.3 Pixel's Classification

Let $\vec{p}_{\min}$ and $\vec{p}_{\max}$ the two points of the RGB space, corresponding to $H_{\min}$ and $H_{\max}$ respectively:

$$\vec{p}_{\min} = (R_{\min}, G_{\min}, B_{\min}) = R_{\min} \cdot \vec{i} + G_{\min} \cdot \vec{j} + B_{\min} \cdot \vec{k} \qquad (5)$$

$$\vec{p}_{\max} = (R_{\max}, G_{\max}, B_{\max}) = R_{\max} \cdot \vec{i} + G_{\max} \cdot \vec{j} + B_{\max} \cdot \vec{k} \qquad (6)$$

Where $\vec{i}$, $\vec{j}$ and $\vec{k}$ the unit direction vectors of the Red, Green and Blue axis; respectively. The equations of the two vertical planes can be set in the RGB space. One passes through the $(R_{\min}, G_{\min}, B_{\min})$ point, the black point and the white point, and the other passes through the $(R_{\max}, G_{\max}, B_{\max})$ point, the black point and the white point. Both are orthogonal to the hue plane, as shown in Figure 23.

The normal to the horizontal hue plane is (1, 1, 1), which is denote as $\vec{1} = (1,1,1) = \vec{i} + \vec{j} + \vec{k}$. Therefore, the normal to these two vertical planes can be found by using the following two vector cross product functions:

$$\vec{n}_{\min} = \vec{1} \times \vec{p}_{\min} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 1 & 1 & 1 \\ R_{\min} & G_{\min} & B_{\min} \end{vmatrix} = (B_{\min} - G_{\min})\vec{i} + (R_{\min} - B_{\min})\vec{j} + (G_{\min} - R_{\min})\vec{k} \qquad (7)$$

Where $\vec{n}_{\min}$ is the normal to the vertical plane that passes through $H_{\min}$, and

$$\vec{n}_{\max} = \vec{1} \times \vec{p}_{\max} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 1 & 1 & 1 \\ R_{\max} & G_{\max} & B_{\max} \end{vmatrix} = (B_{\max} - G_{\max})\vec{i} + (R_{\max} - B_{\max})\vec{j} + (G_{\max} - R_{\max})\vec{k} \qquad (8)$$

Where $\vec{n}_{\max}$ is the normal to the vertical plane that passes through $H_{\max}$. These two vertical planes separate the hue plane into two sections, as shown in Figure23.

To classify a given (R, G, B) pixel in a section of a hue region, which side of the vertical planes the pixel belongs to, should be checked. This can be done mathematically using the dot product of the pixel vector and the normal to the vertical planes. Suppose the pixel is expressed as

$$\vec{p} = R \cdot \vec{i} + G \cdot \vec{j} + B \cdot \vec{k} = (R, G, B) \tag{9}$$

Then, for a pixel that belongs to the vertical plane defined by $H_{\min}$, we get

$$\vec{p} \bullet \vec{n}_{\min} = (B_{\min} - G_{\min})(R - B) + (R_{\min} - B_{\min})(G - B) = 0 \tag{10}$$

A pixel that belongs to the right side of the vertical plane defined by $H_{\min}$, verifies the following equation:

$$\vec{p} \bullet \vec{n}_{\min} = (B_{\min} - G_{\min})(R - B) + (R_{\min} - B_{\min})(G - B) > 0 \tag{11}$$

The same holds to vertical plane defined by $H_{\max}$ for a pixel that is on the left side of the plane or on the plane, the following function is true:

$$\vec{p} \bullet \vec{n}_{\max} = (B_{\max} - G_{\max})(R - B) + (R_{\max} - B_{\max})(G - B) \leq 0 \tag{12}$$

Therefore, in the RGB space, we use Equations (10),(11) and (12) to classify the image pixels according to their hue information, without any transformation from the RGB space to the HIS space. For a real time application this saves a lot of computation time. In summary, the classification scheme is given by:

$$F(R, G, B, H_{\min}, H_{\max}) = q = \begin{cases} 1 & (\vec{p} \bullet \vec{n}_{\min} \geq 0) \ \ and (\vec{p} \bullet \vec{n}_{\max} \leq 0) \\ & 0 \ \ otherwise \end{cases} \tag{13}$$

Where ***q*** is the value of the pixel in the obtained binary image representing the classification results. This binary image is used for the next image processing steps.

I.3 Noise Suppression and Image Segmentation

The misclassified pixels in the Target detection are considered as noise so they should be filtered. Two filtering methods are introduced, namely a threshold based filtering and a mathematical morphology approaches.

I.3.1 Threshold Filtering

In general, the worst case of pixel classification occurs when the pixel is too dark (the amplitude of the RGB value is very small) or too close to white (the relative differences among amplitudes of the R, G and B values are very small). These pixels have limited color information and their hue values are very sensitive to noise. Two approaches are used to suppress these two kinds of noise.

The detection of the very dark pixels, having an intensity value less than an estimated low threshold at first. The threshold is estimated using Equation (2).

Then the detection of the bright pixels, i.e. pixels with a very low saturation value. A threshold for the saturation value of a pixel is estimated using Equation (3).

The thresholding applied during the pixel classification process, does not affect the speed of the method. From equations (2) and (3) only a sum is used to estimate the intensity and saturation parameters the thresholds values are determined experimentally, by selecting very dark and very bright areas, respectively. The results of this approach are illustrated in the following figures, for the detection of a red sphere.

Figure 25 shows the detection result without thresholding.

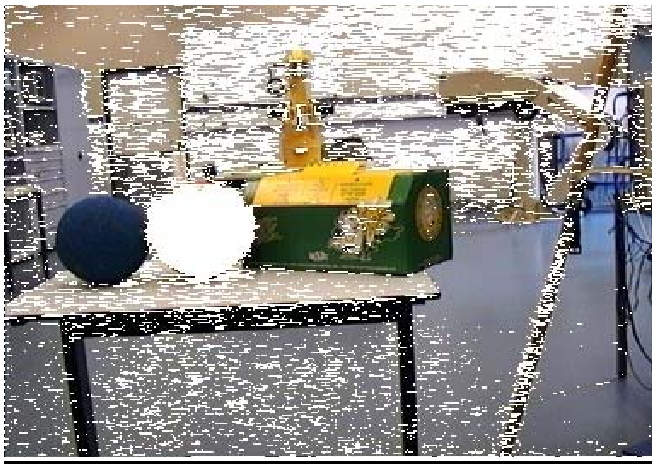


Figure 25: Detection Results without Thresholding

Figure 26 shows the detection results, after application of the intensity threshold only.

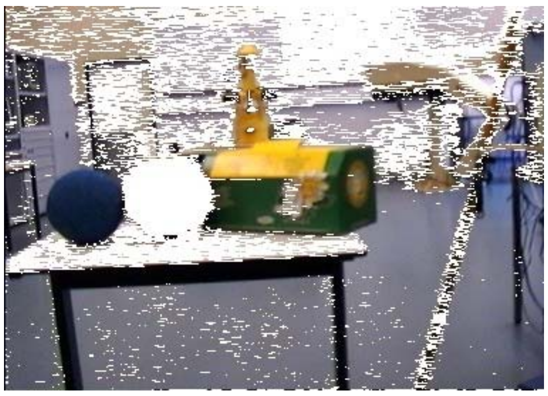


Figure 26 : Application of the Intensity Threshold

Figure 27 shows the detection results after application of the saturation threshold only.

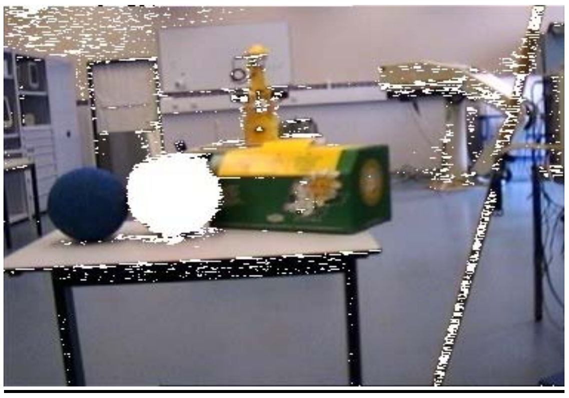


Figure 27: Application of the Saturation Threshold

Finally, Figure 28 shows the detection results after the application of both thresholds

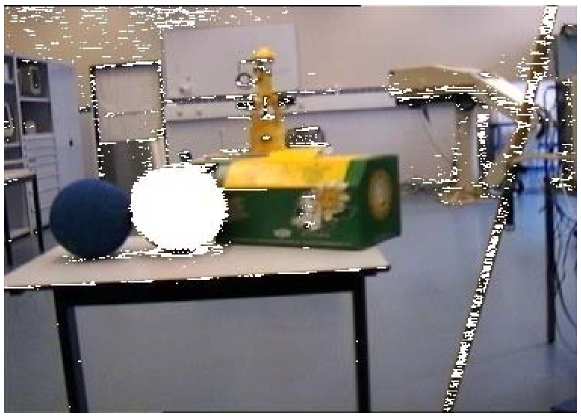


Figure 28: Application of both Intensity and Saturation Thresholds

I.3.2 Morphological Filtering

Morphological Filtering is applied for noise removal, region connection and region segmentation
Two basic operations, namely, Erosion and Dilation, are used with two other operations Opening and Closing.

In practice, two image buffers are used. One contains the original image, the second contains the structuring element as mask.
The applications of these operations will give the following results:

- Wipeout all the isolated dots whose size is smaller than the size of the mask and thin lines whose width is less than the width or height of the mask
- Fill out holes in the detected objects, and produce connected regions.
- Clear up all the boundaries of the objects.

In this example, a circular structuring element has been used for both the erosion and dilation operations. It shows first the original image (Figure29), represents the classification results (Figure 30), and shows the segmentation results (Figure 31).

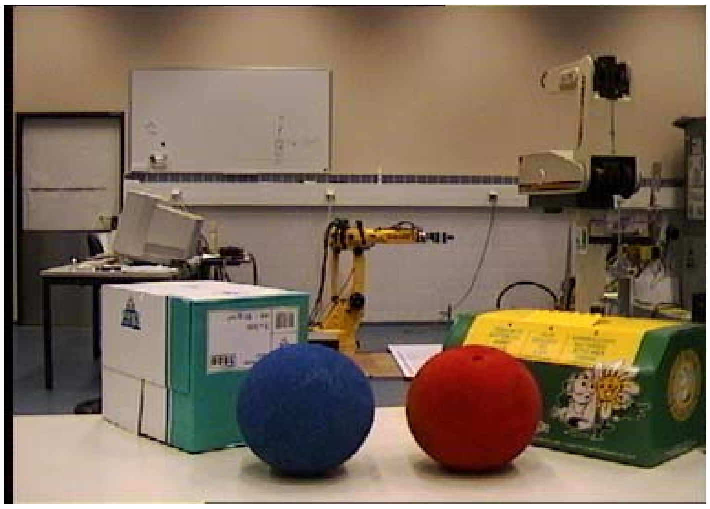


Figure 29:Original Target Image

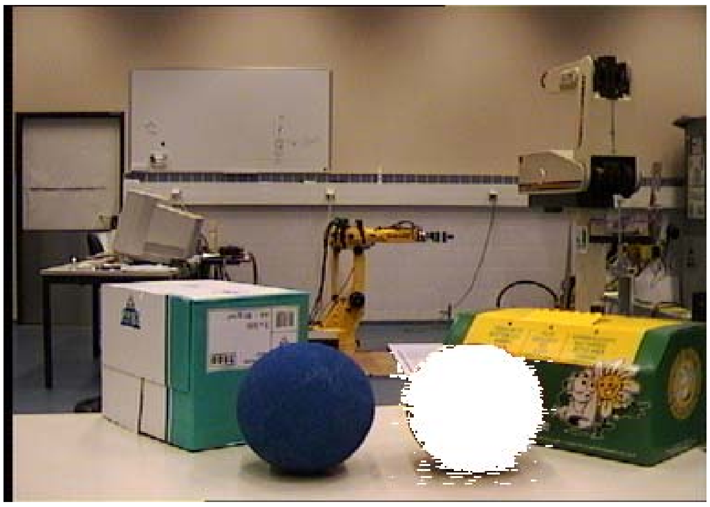


Figure 30: Classification Results

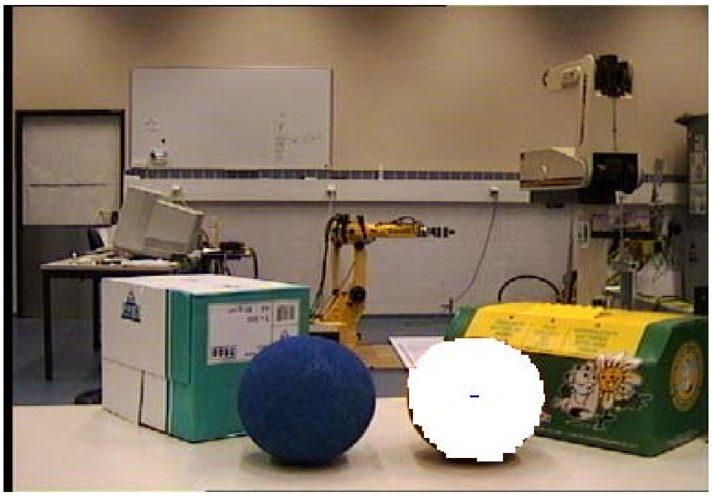


Figure 31: Detected Target after Morphological Filtering

I.4 Target Image Position and Size Estimation

After noise suppression, a connected and well-segmented target image is extracted, as shown in Figure31. For the purpose of target tracking and three-dimensional position estimation, an estimation of the position and size of the target image is needed. For the convenience of estimation, as a target, a **colored sphere** is used, so the target shape, as viewed by the camera, will always appear the same: **a circle**.

I.4.1 Region of Interest

Two methods are used for the target search during the target detection and tracking step. A full search is made at the initialization of the system. During this, the full image is used for color pixel classification and target detection. Having detected the target, adaptive filters are used to predict the position and size of the tracked target for the next coming images. These parameters (position and size) are used to define a search window (or region of interest), it thus the reduction of the computational time and provides a higher signal to noise ratio.

I.4.2 Target Image Diameter Estimation

A colored sphere is used as the target so as it appears (circular shape). It will not change when the camera sees it from different points of views.

I.4.3 Target Image Size and Position Estimation

The estimation of the target image size and its position is in fact an image description problem. The gravity center and the inertial moments are used as descriptors. The mean values of the coordinates of the detected target image pixels are defined as the target gravity center. The second order moments of the detected target boundaries are used as target size descriptors.

The target image gravity center is shown as the point $(\bar{x}, \bar{y})$ in figure32.

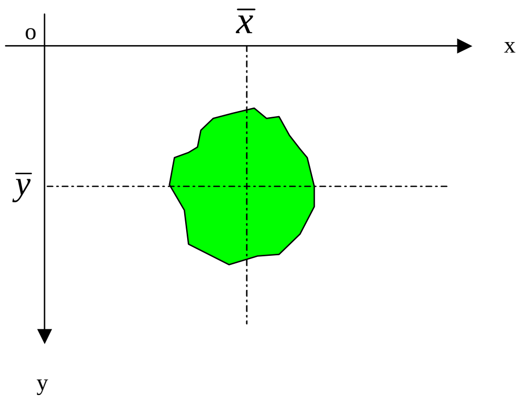


Figure 32 : Gravity Center and Moments of the Detected Target Image

III. Camera Model

The camera used in the HueTracker project was a controllable camera, but in our project cameras mounted in the robot aren't controllable. We will be interested just in only one camera.

The camera used should be mounted parallel to the robot axe; this way the orientation of the robot will be easier. So we have chosen to use the camera pair of pincers in which the position has been changed.

IV. Most Important files and classes

Most Important Files

- *HueTracker.h:* the main header file for the HueTracker application. It declares the CHueTracker application class.
- *HueTracker.cpp:* This is the main source file that contains the definition of: class CHueTracker.

Different Classes and procedures

- *Class CHueTracker*: required to get all the target data.
- *OnHueClassification():*Classifies pixels as belonging to the target object or not.
- *OnMorphologyFiltering():* Main function for morphology filter.
- *OnErode():* Erodes image for morphology filter.
- *OnDilate():* Dilates image for morphology filter.
- *OnOpen() and OnClose():* Sub functions for morphology filter.
- *OnShowResult():* Outputs the morphology filtered image buffer.
- *OnTargetCenter():* Calculates the center of the recognized target object in the image plane.
- *OnTargetSize():*Calculates the size of the recognized target object in the image plane.
- *OnWindowSizeAndPosition():*Calculates the new size of the window surrounding the target object, which will be processed in the next loop.
- *SearchTargetInit():* Initialization of the target tracking procedure , put the camera in home position
- *OnImageConfigure():* Tests if camera reacts
- *OnCalculateDistance():* Calculates the distance to the target object by scaling its size in the image plane
- *OnCameraOrientationControl():* Controls the camera, thus the target object stays centered in the image plane
- *ZoomControl()*: Controls the zoom of the camera
- *OnCircleEstimation*: Controls if the target is a circle
- *SearchTarget(BYTE* image):* This function is given one grabbed frame (image) and it returns 0 if target is not found, 1 if the target is found.

V. Conclusion

In this Chapter the approach used for the target detection and parameters estimation was described. As mentioned, the target used is a colored sphere. A color based classification scheme was proposed for the discrimination between the target and background pixels, and mathematical morphology operators then applied for region segmentation and noise reduction. Finally, the target is parameterized by its center position and size. In the next chapter we will explain our approach to implement the Hue Tracking model to the CASTOR robot platform in order to track the moving color target.

Chapter 4: Hue Tracking Model for CASTOR robot platform

I. Introduction

This chapter is a description of the work carried out and the level of advancement.
The first step was to retrieve the image from the camera, since HueTracker project is based on image treatment.
The second step was to treat the image with the Hue Tracking model to determine the target size, position, center…and finally to enable the robot to follow the target with specified descriptions.
In the next parts we will explain the methodology adopted and the level of the work achieved.

II. Image Grabbing

The MSDN library gives many methods to retrieve an image from a camera.
The first function tried was the GetCurrentImage method that retrieves a copy of the current image being displayed by the VMR.
This method can be called at any time, no matter what state the filter is in, whether running stopped or paused. However, frequent calls to this method will degrade video playback performance.
The second method is the GetCurrentBuffer, the most used with DirectX and it's the chosen one to be used in our application.
The GetCurrentBuffer method retrieves a copy of the buffer associated with the most recent sample. The buffer is directly used with the Tracking module.
The function GrabData( ) is the function responsible for the call of the GetCurrentBuffer and for retrieving the buffer.

The next image is an image captured from the robot camera; the current running image is saved in the folder “c:\DevVisual\Debug\img1.bmp”.

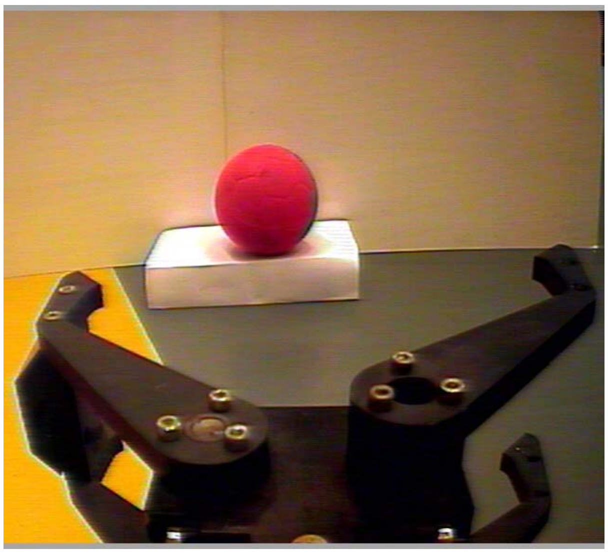


Figure 33 : Image captured from the robot camera

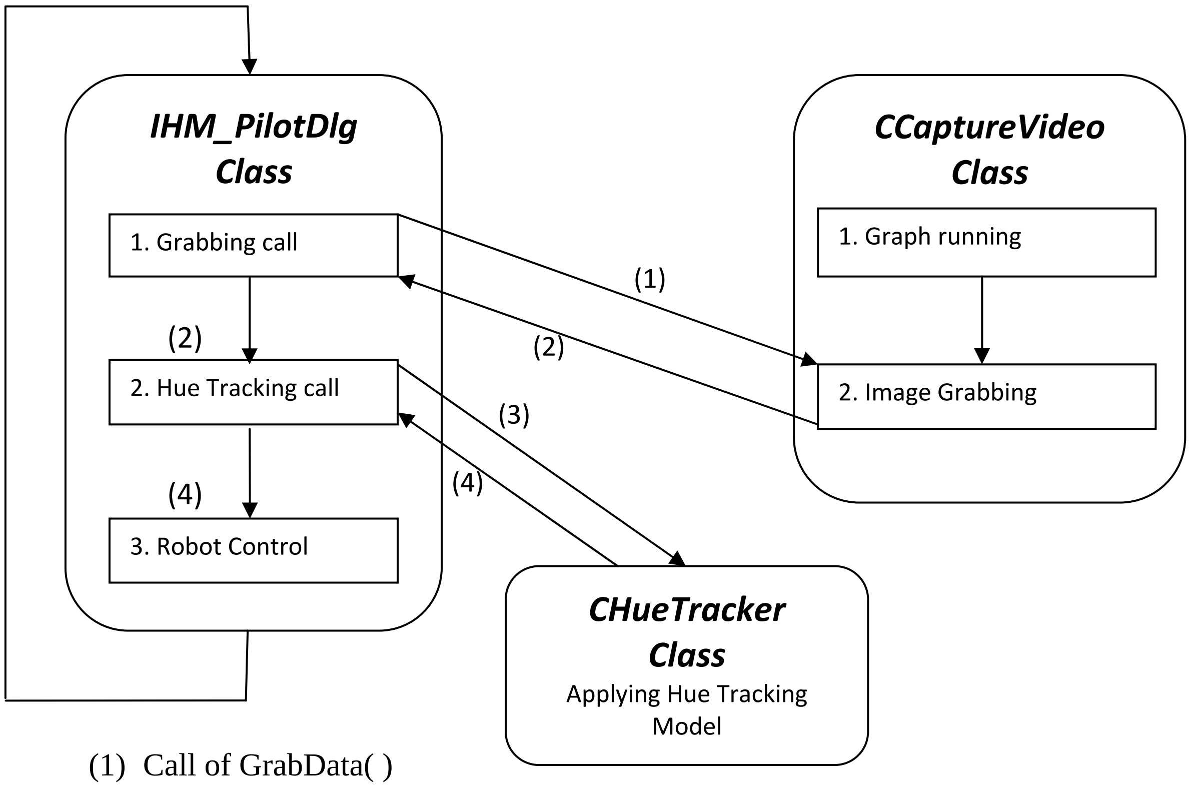
III. Hue Tracking

The buffer retrieved from the GrabData( ) is used with the function SearchTarget(BYTE *buffer). The SearchTarget applies a succession of functions to the buffer to get the target size, position and center. The target data will be used to control the robot.

The most important functions used to the target detection in the HueTracker model are:

- ✓ *OnHueClassification()*
- ✓ *OnMorphologyFiltering()*
- ✓ *OnErode()*
- ✓ *OnDilate()*
- ✓ *OnOpen() and OnClose()*
- ✓ *OnShowResult()*
- ✓ *OnTargetCenter()*
- ✓ *OnTargetSize()*
- ✓ *SearchTarget(BYTE* image)*

The call of the functions and their use between classes is illustrated as follows:



(1) Call of GrabData( )

(2) Buffer

(3) Call of SearchTarget(BYTE* Buffer)

(4) Target Data

III.1. H_{min} and H_{max} Values

The different conditions of lightness, the variation of the environment and the background have an influence on the detected zone.

We have used the Adobe Photoshop, to get in different distances and conditions the (R, G, B) values of the center, thus we can determine the $H_{\min}$ and $H_{\max}$.

The different hue values are fixed as follows:

- ✓ Rmax =200
- ✓ Rmin =150
- ✓ Gmax =50
- ✓ Gmin =15
- ✓ Bmax =70
- ✓ Bmin =20

The findings results are illustrated in the following table:

	Distance / ***(R,G,B)***	***1m***	***2m***	***3m***	***More than 4m***
White Background	***R***	205	206	193	189
	G	12	44	34	50
	B	60	67	38	72
The soil as a background	***R***	178	170	157	165
	G	49	35	30	53
	B	57	59	69	59

Table 1 : The different resulting (R, G, B) values

III.2 Center and Size detection

We have taken several catches to check the modified parameters; the following table illustrates the detected target center and size from different distances:

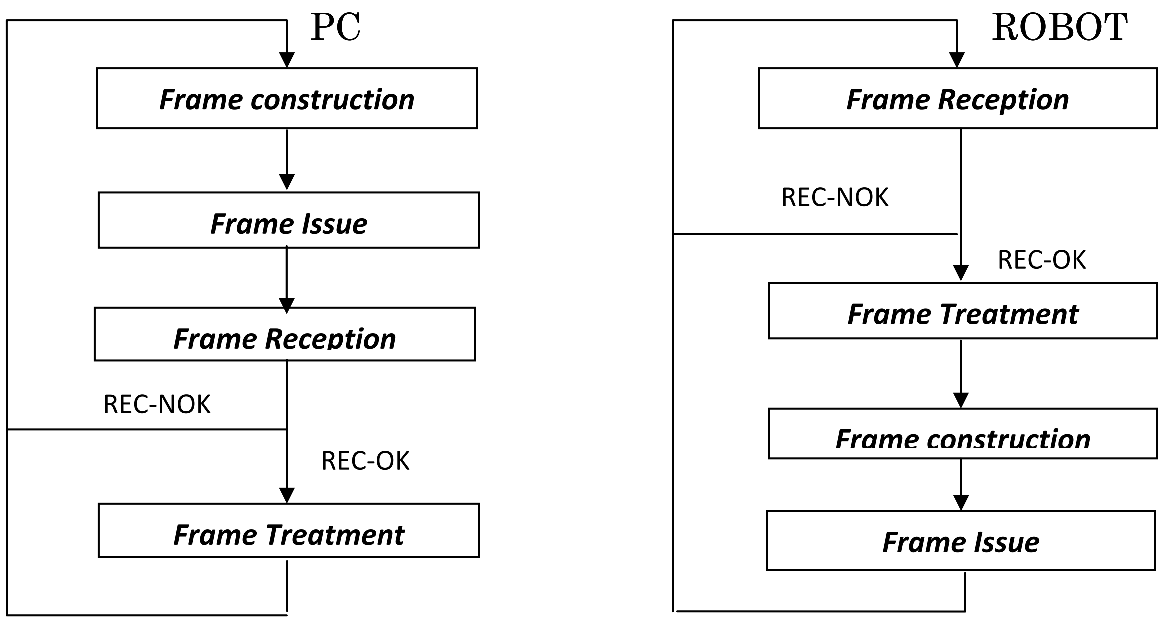
Distance / ***Background***		***1m***	***2m***	***3m***	***4m***
White Background	***m_CenterX***	370	362	354	342
	m_CenterY	381	483	531	499
	m_Total	8446	2914	1248	640
The soil as a background	***m_CenterX***	274	304	322	340
	m_CenterY	375	467	500	515
	m_Total	9986	2608	1562	754

Table 2 : Target size and center detection

- ✓ (m_CenterX, m_CenterY): Target center coordinates
- ✓ m_Total: Target size (with number of pixels)

IV. Robot Control

The robot control is based on an exchange of bit streams between the robot and the control panel case, the next graph shows the frames construction, treatment and circulation on both the PC and the robot.



To control the robot the speed and direction are required. The values of the speed and the direction have been defined as follows

- ✓ For left turning : Direction = 0.3
  Speed = 0.5
- ✓ For right turning : Direction = -0.3
  Speed = 0.5
- ✓ For straight way : Direction = 0.5
  Speed = 1

Where -1<Speed<1 and -1<Direction<1

- Speed = 0.5 ⟹ Speed = Null
- Speed = 1 ⟹ Speed = Max
- Direction = 0.5 ⟹ Direction = Straight

For our application, the control of the robot arm is not taken into consideration it will be done manually with the Joystick.

Three modes for the robot have been defined with these specified parameters: Left, Right and Straight. These defined modes will not be changed during the program running but the number of repetition of those modes is variable.

The two most important data for our application, returned by the Hue Tracking functions, are the target size and the target center coordinates. These data are required to define time duration of the defined modes to be executed. Therefore two counters are used for that: Counter 1 and Counter 2.
If the target center is in the right part of the image (m_CenterX>320), similarly if it is on the left of the image (m_CenterX<320), the robot should turn to the right (or to the left). As a consequence, the running time of the robot has been estimated under the control of Counter 1.
Straight ahead the distance has to be estimated under the control of Counter 2.
The different counters values depend on the m_CenterX and m_Total.
The next two relations define the manner in which the counters are calculated:

- ✓ Counter 1 = |m_CenterX - 320|/ C1 [a]
- ✓ Counter 2 = C2 / m_Total [b]

Where C1 and C2 are two constants experimentally determined.
The following table illustrates the number of counters made on the left mode or the right one to have the target in the middle of the image:

m_CenterX	23	299	190	633	64
Counter 1	3	0	1	3	2

Table 3: Counter 1 variation with m_CenterX

We have estimate C1=110, relation [a] is also:

Counter 1 = |m_CenterX - 320|/ 110

The following table illustrates for different values of Counter 2, the distance made by the robot and the correspondent m_Total:

Counter 2	1	2	3	4	5
Distance (m)	1.3	2.5	3.7	4.5	5.6
m_Total	7022	2599	1799	712	153

Table 4 : Distance and m_Total variation with the Counter 2

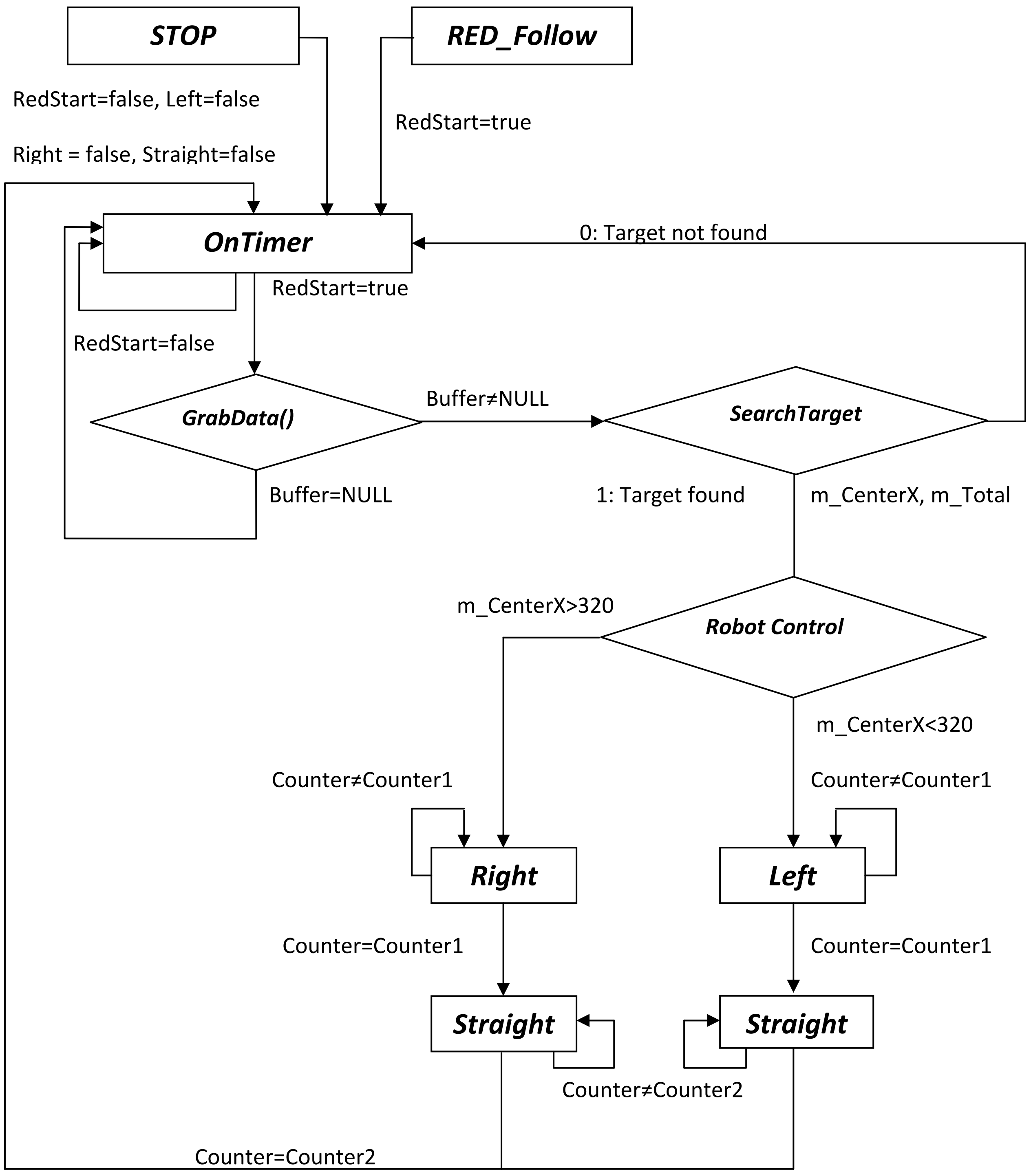
We can conclude from the table that m_Total made an important variation with the distance.
We have estimate C2=8000, relation [b] is also:

Counter 2 = 8000 / m_Total

The next graph describes the algorithm used in the program:

- ✓ RED_Follow: when the button “RED FOLLOW” is pushed, it returns RedStart = true and the program of grabbing and tracking starts.
- ✓ STOP: when the button “STOP“ is pushed all modes are putted to false and the application is stopped.

The program is executed in loopback: the function OnTimer. The next graph describes the new loop of control added to the original program.
The control of the arm, pair of pincers, selected camera and roll of the cable is still under the operator direction.



If RedStart = false, the manual control of the speed and the direction is activated.

V. System Modifications

V.1 Hardware Modification

The pair of pincers camera position has been changed parallel to the robot axis, like that the target will be easy localized and we can keep it in the center of the image by orientating the robot. Figure 34 shows the new position of the camera:

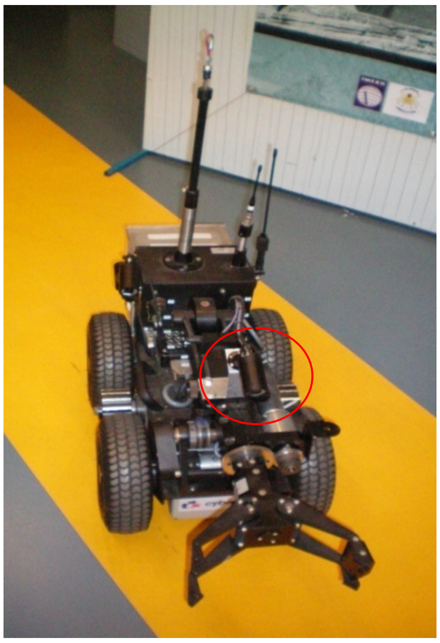


Figure 34 : CASTOR modification

V.2 Software Modification

The next image shows the new HMI interface, two buttons have been added

- RED FOLLOW : to start the grabbing and the tracking aplication
- STOP : to stop the application and to return manual mode

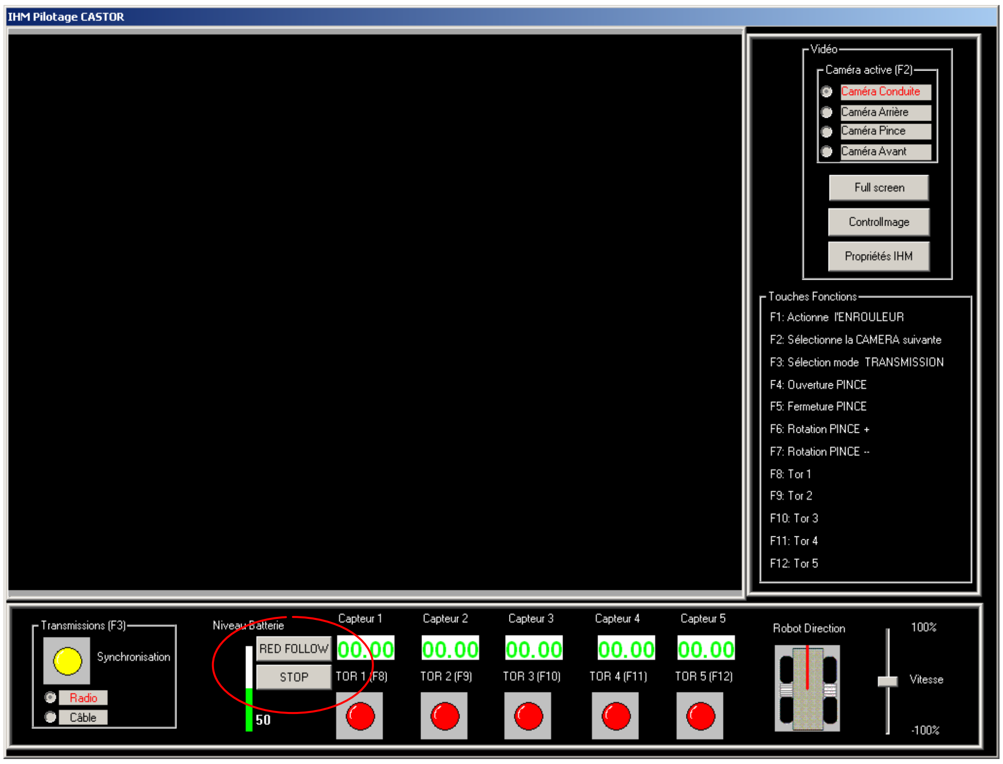


Figure 35 : New HMI Interface

VI. Technical Problems

During our work, we have faced some technical problems.

- ✓ Batteries problems: the batteries of the robot were very old and didn't maintain the load. At the last part of the project, which was the test of the robot control, we were obliged to do the work with a single battery because the second has become unusable. This generated a loss of time.

- ✓ Interference problems: even if the robot is used in the laboratory (indoor), sometimes the signal is disturbed, which makes the control of the robot very difficult.

Conclusion

This paper described the CASTOR robot platform and the implementation of the Hue Tracking model in order to recognize the target and to allow for the robot to follow it.

It can be pointed out that the objectives of this project have been successfully achieved, even if we had some trouble to resolve some problems with the robot control. The Hue Tracking model has been functional and robot is able to detect the target and to localize it.
As the proposed interface does not provide force feedbacks, the operator detects physical contacts and collisions between the robot and the environment visually from the robot's camera images, which is rather difficult especially when using only one camera as it is in our application.
There are also difficulties in monitoring collisions between the environment and the robot which are out of the camera image range. We hope to cope with these difficulties by enhancing the autonomy of the robot in the future practice.

Adding some sensors to objects avoidance and using the work previously performed, may lead to greater efficiency in the control of robot.

Concerning the personal experience, this work helped me to improve and to expand my knowledge with the C + + and DirectX and to learn more about an important domain as robotic.

It helps me to have a close view on robots platform, robots control and the development of robots intelligence and autonomy.

Bibliography

MUT008-User's Manual castor RMA_vf.

MUT0012-User's Manual Logiciel_vf.

Ping Hong, "VISUAL SERVOING FOR ROBOT NAVIGATION: Application in Humanitarian Demining", VUB-RMA 2001.

Kristel Verbiest, " Report CASTOR", 2006.

[1] Nicholas Hollinghurst and Roberto Cipolla, "Uncalibrated Stereo Hand-Eye Coordination", Image and Vision Computing, Volume 12, Number 3, pp. 187-192, April 1994.

[2] Ryuzo Okada et al., "Object Tracking based on Optical Flow and Depth", Proceedings of IEEE/SICE/RSJ International Conference on MFI, pp. 565-571, 1996.

[3] Stephen J. McKenna, Yogesh Raja and Shaogang Gong, "Tracking Color Objects Using adaptive mixture Models", Image and Vision Computing, Vol. 17, pp. 225-231, 1999.

[4] Rafael Garcia, Joan Batlle and Marc Carreras, "Real-Time Tracking of Mobile Robots in Structured Environments", Proceedings of Automation '99, pp. 207-212.

[6] Rafael G.Gonzalez and Richard E.Woods, "Digital Image Processing", Addison-Wesley Publishing Company, Inc., 1993.

[7] Charles Poynton, "A Guided Tour of Color Space", Proceedings of the SMPTE Advanced Television and Electronic Imaging Conference, San Francisco, pp. 167-180, Feb. 1995.

[8] Hannes Filippi. "Wireless Teleoperation of Robotic Manipulators", Espoo Finland, August 2007.

[9] Casper, J., and Murphy, R. R. "Human-Robot Interactions During the Robot Assisted Urban Search and Rescue Effort at the World Trade Center," IEEE Tansactions on Systems Man, and Cybernetics, Part B, vol. 33, no. 3,pp. 367-385, Jun. 2003.

Webography

http://www.used-robots.com/robot-education.php?page=robot+timeline

http://www.answers.com/topic/robot?cat=technology

http://hapticshistory.chc61.uci.cu/haptic/site/pages/Machine-Haptics-BackGround.php

http://www.geckosystems.com/industries/robot_history.php

blog.wired.com/cars/2007/11/motown-blues--a.html.

Appendices

ABBREVIATIONS

- **HMI** = Human Machine Interface (Interface Homme Machine)
- **TOR** = Tout Ou Rien (All or Nothing)
- **TC** = Telecommande (the data which make up the commands for the different elements of the robot, it is sent from the control panel to the robot)
- **TM** = Telemesure (measurement of the data of the environment that is sent to the control panel where it will be displayed)
- **CRC** = Cyclic Redundancy Check
- **LSB** = Least Significant Bit
- **MSB =** Most Significant Bit
- **VMR =** Video Mixing Renderer

C++ CODE FOR IMAGE GRABBING

CaptureVideoSimple.h

```
#if
!defined(AFX_CAPTUREVIDEO_H__057736B5_B61B_4850_8D82_E181E0B25B61__IN
CLUDED_)
#define
AFX_CAPTUREVIDEO_H__057736B5_B61B_4850_8D82_E181E0B25B61__INCLUDED
//---------------------------------------------------------------------//
// Copyright DILLIP KUMAR KARA 2004
// You may do whatever you want with this code, as long as you include this
// copyright notice in your implementation files.
// Comments and bug Reports: codeguru_bank@yahoo.com
//---------------------------------------------------------------------//
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
//CaptureVideo.h : header file
#include <atlbase.h>
#include <windows.h>
#include <dshow.h>
#include "D3d9.h"
#include "vmr9.h"
#include "header.h"
#include <time.h>
enum PLAYSTATE {STOPPED, PAUSED, RUNNING, INIT};
#define WM_GRAPHNOTIFY  WM_USER+1
#ifndef SAFE_RELEASE

#define SAFE_RELEASE( x )
   if ( NULL != x )
   {
      x->Release( );
      x = NULL;
   }
#endif
typedef struct strImageParams{
         long minBright;
         long maxBright;
         long valBright;

         long minContrast;
         long maxContrast;
         long valContrast;

         long minSaturation;
         long maxSaturation;
         long valSaturation;
```

```
        long minHue;
        long maxHue;
        long valHue;

        long minAlpha;
        long maxAlpha;
        long valAlpha;
}strImageParams;

// CCaptureVideo window
class CCaptureVideo
{
// Construction
public:
        CCaptureVideo();
// Attributes
public:
        void StopCapture();
        bool StartCompositeVideo();
        void StartSVideo();
        void RemoveGraphFromRot(DWORD pdwRegister);
        void UnIntializeVideo();
        void zoomRectangle(CPoint p1, CPoint p2, int tailleX, int tailleY);
        void DrawRect();
        void FullImage(int sizex, int sizey);
        void SethWnd(HWND hWnd);
        void getTailleImage(int *x, int *y);
        void RespectRatio(BOOL ratio);
        strImageParams GetImageParams(void);
        HRESULT GetCurrentValue(void);
        HRESULT InitializeVideo(HWND hWnd);
        void repaint(HWND hWnd, HDC hdc);
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CCaptureVideo)
        //}}AFX_VIRTUAL
// Implementation
public:
        virtual ~CCaptureVideo();
        // Generated message map functions
protected:
        //{{AFX_MSG(CCaptureVideo)
                // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        afx_msg HRESULT OnGraphNotify(WPARAM wp , LPARAM lp);
protected:
        //------------------Video--------------------//
        void DisplayMesg(TCHAR* szFormat, ...);
        LRESULT ClearInterfaces(WPARAM wp, LPARAM lp);
        void CloseInterfaces();
        HRESULT AddGraphToRot(IUnknown* pUnkGraph, DWORD* pdwRegister);
        HRESULT CaptureVideo();
        HRESULT HandleGraphEvent();
        HRESULT ChangePreviewState(int nShow);
        HRESULT FindCaptureDevice(IBaseFilter** ppSrcFilter);
        HRESULT GetInterfaces();
        HRESULT SetupVideoWindow();
        //------------------Renderer-----------------//
        HRESULT ConfigRenderer();
private:
        UINT chSVideo, chCompVideo , chWebCam , chFullScreen , chAlwaysOnTop ;
        int nVSourceCount;
        int nAnalogCount;
        int sizeX;// taille de l'image acquise
        int sizeY;
        strImageParams imageParams;
        CBrush m_emptyBrush;
        DWORD m_dwGraphRegister;
        BOOL bDevCheck;
        HWND m_hApp;
        IBaseFilter *pVmr;
        //-----------------Renderer----------------//
        IVMRWindowlessControl9* pWc;
        //-----------------Video-------------------//
        IMediaControl* m_pMC;
        IMediaEventEx* m_pME;
        IGraphBuilder* m_pGraph;
        ICaptureGraphBuilder2* m_pCapture ;
        IBaseFilter* pSrcF;
        PLAYSTATE m_psCurrent;
        BOOL bVideo;
        int vType;
        ISampleGrabber *pGrabber;
        IBaseFilter *pGrabberF;
public:
        IAMVideoProcAmp *pProcAmp;
        inline BOOL getIsVideo(){return bVideo;};
        unsigned char* GrabData();          //call grab data first
};
/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous
line.
#endif //
!defined(AFX_CAPTUREVIDEO_H__057736B5_B61B_4850_8D82_E181E0B25B61__IN
CLUDED_)
```

CaptureVideoSimple.cpp

```
// CaptureVideo.cpp : implementation file
///-------------------------------------------------------------------------//
// Copyright DILLIP KUMAR KARA 2004
// You may do whatever you want with this code, as long as you include this
// copyright notice in your implementation files.
// Comments and bug Reports: codeguru_bank@yahoo.com
//-------------------------------------------------------------------------//
#include "stdafx.h"
#include "CaptureVideoSimple.h"
#include "header.h"
#include <math.h>
#include <SYS/timeb.h>
#include <wingdi.h>
#include <windows.h>
#include <winuser.h>
#include <time.h>
#include <AFXWIN.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#define REGISTER_FILTERGRAPH
// define the ratio of focal length over pixel size
#define k3  (0.0521488*768/(32.0*1.e-3))
#define k4  (0.0521488*576/(26.6*1.e-3))
#define  OutOfBorders(x,y,width1,width2,height1,height2)  ((x)<(width1)  ||  (y)<(height1)  ||
(x)>=(width2) || (y)>=(height2))

long pBufferSize;
unsigned char* pBuffer;
unsigned int gWidth=0;
unsigned int gHeight=0;
unsigned int gChannels=0;
VIDEOINFOHEADER *pVih;
/////////////////////////////////////////////////////////////////////////////
// CCaptureVideo
CCaptureVideo::CCaptureVideo()
{
        // Initialization
        m_hApp=NULL;
        m_dwGraphRegister=0;
        nAnalogCount =0; // Counting Analog devices
        m_pMC = NULL;
        m_pME = NULL;
        m_pGraph = NULL;  // IFilterGraph2 provides AddSourceFileForMoniker()
        m_pCapture = NULL;
        pSrcF = NULL ;
```

```
          pProcAmp = NULL;
          pVmr = NULL;
          pWc= NULL;
}
CCaptureVideo::~CCaptureVideo()
{

}
HRESULT CCaptureVideo::CaptureVideo()
{
   HRESULT hr;
   IBaseFilter *pSrcFilter=NULL;
   // Get DirectShow interfaces
hr = GetInterfaces();
if (FAILED(hr))
   {
      DisplayMesg(TEXT("Failed to get video interfaces!  hr=0x%x"), hr);
      return hr;
   }
   // Attach the filter graph to the capture graph
hr = m_pCapture->SetFiltergraph(m_pGraph);
if (FAILED(hr))
   {
      DisplayMesg(TEXT("Failed to set capture filter graph!  hr=0x%x"), hr);
      return hr;
   }
   // Use the system device enumerator and class enumerator to find
   // a video capture/preview device, such as a desktop USB video camera.
hr = FindCaptureDevice(&pSrcFilter);
if (FAILED(hr))
   {
      // Don't display a message because FindCaptureDevice will handle it
      return hr;
   }
          if( bDevCheck == FALSE)
          {                  return E_FAIL;          }

  // Add Capture filter to our graph.
   hr = m_pGraph->AddFilter(pSrcFilter, L"Video Capture");
if (FAILED(hr))
   {
      DisplayMesg(TEXT("Couldn't add the capture filter to the graph!  hr=0x%x\r\n\r\n")
      TEXT("If you have a working video capture device, please make sure\r\n")
      TEXT("that it is connected and is not being used by another application.\r\n\r\n"), hr);
      pSrcFilter->Release();
      return hr;
   }
 // Create the Sample Grabber.
hr = CoCreateInstance(CLSID_SampleGrabber, NULL, CLSCTX_INPROC_SERVER,
                              IID_IBaseFilter, (void**) & pGrabberF);
```

```
        if (FAILED(hr)) {                return hr;                       }
hr = m_pGraph->AddFilter(pGrabberF, L"Sample Grabber");
        if (FAILED(hr)) {                return hr;                      }
 pGrabberF->QueryInterface(IID_ISampleGrabber, (void**)&pGrabber);
 AM_MEDIA_TYPE mt;
 ZeroMemory(&mt, sizeof(AM_MEDIA_TYPE));
 mt.majortype = MEDIATYPE_Video;
 mt.subtype = MEDIASUBTYPE_RGB24;
hr = pGrabber->SetMediaType(&mt);
if (FAILED(hr)) {                return hr;                       }
hr = pGrabber->SetOneShot(FALSE);
hr = pGrabber->SetBufferSamples(TRUE);
hr = pSrcFilter->QueryInterface(IID_IAMVideoProcAmp, (void**)&pProcAmp);
ConfigRenderer();
// Add Renderer filter to our graph.
    hr = m_pGraph->AddFilter(pVmr, L"VMR9");
 // Render the preview pin on the video capture filter
 // Use this instead of m_pGraph->RenderFile
hr = m_pCapture->RenderStream (&PIN_CATEGORY_PREVIEW, MEDIATYPE_Video,
                             pSrcFilter,pGrabberF, pVmr);
if (FAILED(hr))
    {
        DisplayMesg(TEXT("Couldn't render the video capture stream.  hr=0x%x\r\n")
        TEXT("The capture device may already be in use by another application.\r\n\r\n")
        TEXT("The sample will now close."), hr);
        pSrcFilter->Release();
        return hr;
    }
 hr = pGrabber->GetConnectedMediaType(&mt);
 if (FAILED(hr)) {                return hr;                       }
        pVih = (VIDEOINFOHEADER *)mt.pbFormat;
        gChannels = pVih->bmiHeader.biBitCount / 8;
        gWidth = pVih->bmiHeader.biWidth;
        gHeight = pVih->bmiHeader.biHeight;
    // Now that the filter has been added to the graph and we have
    // rendered its stream, we can release this reference to the filter.
    pSrcFilter->Release();
#ifdef REGISTER_FILTERGRAPH
    // Add our graph to the running object table, which will allow
    // the GraphEdit application to "spy" on our graph
hr = AddGraphToRot(m_pGraph, &m_dwGraphRegister);

if (FAILED(hr))
    {
        DisplayMesg(TEXT("Failed to register filter graph with ROT!  hr=0x%x"), hr);
        m_dwGraphRegister = 0;
    }
#endif
// Start previewing video data
hr = m_pMC->Run();
```

```
 if (FAILED(hr))
    {
       DisplayMesg(TEXT("Couldn't run the graph!  hr=0x%x"), hr);
       return hr;
    }
 // Remember current state
 m_psCurrent = RUNNING;
Sleep(300); // wait until the graph running
return S_OK;
}
unsigned char* CCaptureVideo::GrabData()
{
HRESULT hr;
HANDLE fh;
BITMAPFILEHEADER bmphdr;
DWORD nWritten;
   if (pGrabber == 0)
             return 0;
 long Size = 0;
hr = pGrabber->GetCurrentBuffer(&Size, NULL);
       if (FAILED(hr)) {             return 0;            }
       else if (Size != pBufferSize) {
             pBufferSize = Size;
             if (pBuffer != 0)
                   delete[] pBuffer;
             pBuffer = new unsigned char[pBufferSize];
       }
 hr = pGrabber->GetCurrentBuffer(&pBufferSize, (long*)pBuffer);
 if (FAILED(hr))
                   {              return 0;              }
 else {
memset(&bmphdr, 0, sizeof(bmphdr));//allocation de la mémoire
bmphdr.bfType = ('M' << 8) | 'B';// spécifier le type du fichier:bmp
// spécifier la taille du fichier
bmphdr.bfSize = sizeof(bmphdr) + sizeof(BITMAPINFOHEADER) + pBufferSize;
//spécifier le offset, en bit, dés le début du fichier jusqu'au les bits du la bitmap
bmphdr.bfOffBits = sizeof(bmphdr) + sizeof(BITMAPINFOHEADER);
fh = CreateFile("img1.bmp", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);
WriteFile(fh, &bmphdr, sizeof(bmphdr), &nWritten, NULL);
WriteFile(fh, &pVih->bmiHeader, sizeof(BITMAPINFOHEADER), &nWritten, NULL);
WriteFile(fh, pBuffer, pBufferSize, &nWritten, NULL);
CloseHandle(fh);
return pBuffer;
          }
}
```

C++ CODE FOR HUE TRACKING

HueTracker.h

```
////////////////////////////////////////////////////////////////////////////////
//// FILE: HueTracker.h
//// PROJECT: HueTracker
//// TASK: Implementation of the CHueTracker class
//// ORIGINALLY WRITTEN BY: Hong Ping
//// EDITED BY: Geert De Cubber and Wahiba Jomaa
//// REQUIRES: -
////////////////////////////////////////////////////////////////////////////////
//moved from FirstEasyC24.cpp
#include ".\cppsource\PIDController.h"
#include ".\cppsource\SystemIdentification.h"
#include ".\cppsource\KalmanFilter.h"
#include ".\cppsource\FullStateFeedbackController.h"
#include ".\cppsource\AdaptiveFilter.h"
#include ".\cppsource\AdaptiveFilter1.h"
#include ".\cppsource\LowPassFilter.h"
class CHueTracker
{
public:
	CHueTracker(int width, int height);
	~CHueTracker();
	BYTE	*m_pun8ImageBuffer, *m_pun8AlignedBuffer;
	UINT32	m_un32ImageHeight,m_un32ImageWidth, m_un32ImageSize;
protected:
	//removed static keyword before these functions
	void OnShowResult();
	void OnClose();
	void OnOpen();
	void OnDilate();
	void OnErode();
	void OnMorphologyFiltering();
	void OnHueClassification();
// Implementation
//move members to protected or private as much as possible
public:
	float	m_AngleX, m_AngleY;
	int		m_g1, m_g2;
	int		m_r1, m_r2;
	int		m_gL1, m_gL2;
	int		m_rL1, m_rL2;
//pointers to objects, instances created in constructor
CPIDController				*pPIDx, *pPIDy;
CSystemIdentification			*pSysIdentX, *pSysIdentY;
CKalmanFilter				*pKalmX, *pKalmY;
CFullStateFeedbackController            *pFeBaCtrX,*pFeBaCtrY;
CAdaptiveFilter                         *pAdapX, *pAdapY, *pAdapH, *pAdapW;
CAdaptiveFilterD                        *pAdapIdenX, *pAdapIdenY;
CLowpassFilter2                         *pLoPass1, *pLoPass2, *pLoPass3;
CLowpassFilter2* pLoPass31;
CLowpassFilter2* pLoPass21;
CLowpassFilter2* pLoPass11
long int m_EndTime;
long int m_BeginTime;
//removed static keyword before these functions
void OnTargetSize();
void OnTargetCenter();
void OnWindowSizeAndPosition();
BOOL SearchTarget(BYTE * image);
long int m_Total;
unsigned char *m_pBinaryImageBuffer1;
unsigned char *m_pBinaryImageBuffer;
//removed from globally defined in HueTracker.cpp
//removed static keyword
int m_WindowPositionY1;
int m_WindowPositionX1;
int m_WindowPositionY2;
int m_WindowPositionX2;
int m_WindowHeight;
int m_WindowLength;
int m_MomentY;
int m_MomentX;
int m_CenterY;
int m_CenterX;
int m_r11;      // (Bmin-Gmin)
int m_g11;      // (Bmax-Gmax)
int m_r22;      // (Rmin-Bmin)
int m_g22;      // (Rmax-Bmax)
int m_Threshold3;
int m_Threshold2;
// moved from SearchTarget
BOOL m_bTargetFound;
private:
        float   *m_RequiredSysParametersX, *m_RequiredSysParametersY;
        int             m_MaskSize;
        int             m_Tolerance;
};
```

HueTracker.cpp

```
////////////////////////////////////////////////////////////////////////////////
//// FILE: HueTracker.cpp
//// PROJECT: HueTracker
//// TASK: Implementation of the target tracking algorithm
//// ORIGINALLY WRITTEN BY: Hong Ping
```

```
//// EDITED BY: Geert De Cubber and Wahiba Jomaa
//// REQUIRES: Camera library for the control of the camera
///////////////////////////////////////////////////////////////////////////////////
#include "StdAfx.h"
#include <SYS/timeb.h>
#include "HueTracker.h"
#include <stdio.h>
// define the ratio of focal length over pixel size
#define k3  (0.0521488*768/(32.0*1.e-3))
#define k4  (0.0521488*576/(26.6*1.e-3))
#define OutOfBorders(x,y,width1,width2,height1,height2) ((x)<(width1) || (y)<(height1) ||
(x)>=(width2) || (y)>=(height2))
extern  float *myvector(long nl, long nh);
extern  void free_myvector(float *v, long nl, long nh);
// CHueTracker construction/destruction
CHueTracker::CHueTracker(int width, int height)
{
//moved from global static defined above
//initialize member variables
        m_CenterX = 0;
        m_CenterY = 0;
        m_MomentX = 0;
        m_MomentY = 0;
        m_WindowHeight = 0;
        m_WindowLength = 0;
        m_WindowPositionY1 = 0;
        m_WindowPositionX1 = 0;
        m_WindowPositionY2 = 0;
        m_WindowPositionX2 = 0;
        m_Total = 0;
// create PIDController objects
pPIDx = new CPIDController((float)0.9,(float)0.0,(float)0.0,(float)1.570796326795 ,(float)-
1.570796326795);
pPIDy = new CPIDController((float)0.9,(float)0.0,(float)0.0,(float)0.7853981633974,(float)-
0.7853981633974);
// create System Identification objects
pAdapIdenX = new CAdaptiveFilterD(2,(float)0.71,(float)0.3,(float)0.001,(float)0.00001,0);
pAdapIdenY = new CAdaptiveFilterD(2,(float)0.81,(float)0.5,(float)0.001,(float)0.00001,0);
// create Kalman filter objects
        pKalmX = new CKalmanFilter(4,2,(float) 8.81,(float) 0.51);
        pKalmY = new CKalmanFilter(4,2,(float) 6.81,(float) 0.51);
// create Full State Feedback Controller objects
        pFeBaCtrX = new CFullStateFeedbackController(2);
        pFeBaCtrY = new CFullStateFeedbackController(2);
// initialize the required system parameters
        m_RequiredSysParametersX = myvector(1,2);
        m_RequiredSysParametersY = myvector(1,2);
        m_RequiredSysParametersX[1] = (float) (0.336*2);
        m_RequiredSysParametersX[2] = (float) (-0.288*2);
        m_RequiredSysParametersY[1] = (float) (0.336*0.5);//was 2 instead of 0.2
        m_RequiredSysParametersY[2] = (float) (-0.288*0.5);//was 2 instead of 0.2
// create Adaptive Filter objects
pAdapX = new CAdaptiveFilter(2,(float)0.000021,(float)0.000001,(float)1.0e-
11,(float)0.00001);
pAdapY = new CAdaptiveFilter(2,(float)0.000021,(float)0.000001,(float)1.0e-
11,(float)0.00001);
pAdapH = new CAdaptiveFilter(2,(float)0.0000001,(float)0.000000001,(float)5.0e-
18,(float)0.00001);
pAdapW = new CAdaptiveFilter(2,(float)0.0000001,(float)0.000000001,(float)5.0e-
18,(float)0.00001);
// create Lowpass Filters for distance measurement
pLoPass1 = new CLowpassFilter2((float)-
0.532697,(float)0.607466,(float)0.2112234,(float)0.4224468,(float)0.2112234);
pLoPass2 = new CLowpassFilter2((float)-
0.397552,(float)0.199651,(float)0.2112234,(float)0.4224468,(float)0.2112234);
pLoPass3 = new CLowpassFilter2((float)-
0.3467604,(float)0.046383,(float)0.2112234,(float)0.4224468,(float)0.2112234);
// create Lowpass Filters for diameter measurement
// the definition of parameters can be seen from the class definition
pLoPass11 = new CLowpassFilter2((float)-0.532697,(float)0.607466,
(float)0.2112234,(float)0.4224468,(float)0.2112234);
pLoPass21 = new CLowpassFilter2((float)-0.397552,(float)0.199651,
(float)0.2112234,(float)0.4224468,(float)0.2112234);
pLoPass31 = new CLowpassFilter2((float)-0.3467604,(float)0.046383,
(float)0.2112234,(float)0.4224468,(float)0.2112234);
//initialize image size
        m_un32ImageSize=width*height*3;
        m_un32ImageWidth=width;
        m_un32ImageHeight=height;
// -----------------------------------------------------------------------
// Allocate image buffer
// -----------------------------------------------------------------------
m_pBinaryImageBuffer = (PUINT8)VirtualAlloc(NULL,
m_un32ImageHeight*m_un32ImageWidth, MEM_COMMIT, PAGE_READWRITE);
if(m_pBinaryImageBuffer == NULL)
        {
AfxMessageBox("Cannot allocate a new binary buffer", MB_OK | MB_APPLMODAL |
MB_ICONSTOP );
        return;
        }
// -----------------------------------------------------------------------
// Allocate image buffer
// -----------------------------------------------------------------------
m_pBinaryImageBuffer1 =  (PUINT8)VirtualAlloc(NULL,
m_un32ImageHeight*m_un32ImageWidth, MEM_COMMIT, PAGE_READWRITE);
if(m_pBinaryImageBuffer1 == NULL)
        {
AfxMessageBox("Cannot allocate a new binary buffer1", MB_OK | MB_APPLMODAL |
MB_ICONSTOP );
        return;
        }
m_Tolerance = 30;
m_WindowPositionX1 = 0;
m_WindowPositionX2 = m_un32ImageWidth;
m_WindowPositionY1 = 0;
m_WindowPositionY2 = m_un32ImageHeight;

// for hue classification
m_r1 = m_r11 = 100-15;     // (Bmin-Gmin)
m_g1 = m_g11 = 20-50;     // (Bmax-Gmax)
m_r2 = m_r22 = 200-100;    // (Rmin-Bmin)
m_g2 = m_g22 = 120-20;    // (Rmax-Bmax)

m_rL1 = m_r1;// 100-15; // (Bmin-Gmin)
m_gL1 = m_g1;// 20-50; // (Bmax-Gmax)
m_rL2 = m_r2;//200-100; // (Rmin-Bmin)
m_gL2 = m_g2;//120-20; // (Rmax-Bmax)

m_Threshold3=20;
m_Threshold2=40;

// morphology filter
        m_MaskSize = 5;
        m_AngleX = 0.0;
        m_AngleY = 0.0;
// The target size
        float    m_RealDiameter = (float) 0.17;
//load stored parameters
FILE *filep;
int i;
filep = fopen("Para.txt","r");
if (filep==NULL)
        {
                AfxMessageBox("file error");
        }
else
        {
                for (i=1;i<=4;i++)
                {
                        fscanf(filep,"%f",&pAdapIdenX->m_TapWeightVector[i]);
                }

                for (i=1;i<=4;i++)
                {
                        fscanf(filep,"%f",&pAdapIdenY->m_TapWeightVector[i]);
                }
        }
        fclose(filep);
}
CHueTracker::~CHueTracker()
```

```
{
          if ( m_pun8ImageBuffer )
                    delete [] m_pun8ImageBuffer;
          if(m_pBinaryImageBuffer)
                    delete [] m_pBinaryImageBuffer;
          if(m_pBinaryImageBuffer1)
                    delete [] m_pBinaryImageBuffer1;
 free_myvector(m_RequiredSysParametersX,1,2);
 free_myvector(m_RequiredSysParametersY,1,2);
}

// PROCEDURE: OnHueClassification
// TASK: Classify pixels as belonging to the target object or not
void CHueTracker::OnHueClassification()
{
 int i, j, k, w, bp=3,w1;
 int i1, i2, j1, j2;
 int R, G, B, r, m, threshold2,
 threshold3,r11, r22, g11, g22;
unsigned char *buf, *buf1;
  r11 = m_r11;          // (Bmin-Gmin)
  g11 = m_g11;          // (Bmax-Gmax)
  r22 = m_r22;          // (Rmin-Bmin)
  g22 = m_g22;          // (Rmax-Bmax)
    threshold3 = m_Threshold3;
  threshold2 = m_Threshold2;
  w  = m_un32ImageWidth*bp;
  w1 = m_un32ImageWidth;
  /* the buffer is used to store the original image */
  buf = m_pun8ImageBuffer;
  /* the buffer is used to store the created binary image */
  buf1 = m_pBinaryImageBuffer;
  i1 = m_WindowPositionY1;
  i2 = m_WindowPositionY2;
  j1 = m_WindowPositionX1;
  j2 = m_WindowPositionX2;
          for (i=i1; i<i2; i++)
          {
                    for(j=j1; j<j2; j++)
                    {
                              buf1[i*w1 + j]=0;  /* clear the buffer */
                              k = i*w + j*bp;
                              B = (int) buf[0+k];
                              G = (int) buf[1+k];
                              R = (int) buf[2+k];
                              m = R+G+B;

                              /* if the pixel's intensity is too small */
                              if(m<threshold2) goto label;
                              r=R;
```

```
                        if(G<r) r=G;
                        if(B<r) r=B;
                        /* if the saturation is too small */
                        if(m*(100-threshold3)<300*r) goto label;
                        /* if the pixel is belong to the specified hue region */
                        if((r11*(R-B)+r22*(G-B))<0) goto label;
                        if((g11*(R-B)+g22*(G-B))>0) goto label;
                        /* if true then set a flag to create the binary image */
                        buf1[i*w1+j]=1;
                        label:          ;
                }
        }
}
// PROCEDURE: OnMorphologyFiltering
// TASK: Main function for morphology filter
void CHueTracker::OnMorphologyFiltering()
{
        OnOpen();
        OnClose();
}
// PROCEDURE: OnErode
// TASK: Erode image for morphology filter
void CHueTracker::OnErode()
{
  int         i,j;
  int         height1,height2,width1,width2,w;
  int         l1, l2;
  unsigned char         *eroded;
  unsigned char         *image;
  unsigned char         *buffer;
  int         mask_size2 = m_MaskSize>>1;
  image          =       m_pBinaryImageBuffer;
  eroded         =       m_pBinaryImageBuffer1;
  w                      =       m_un32ImageWidth;
  height1        =       m_WindowPositionY1;
  height2        =       m_WindowPositionY2;
  width1         =       m_WindowPositionX1;
  width2         =       m_WindowPositionX2;

  for(i=height1; i<height2; i++) {
    for(j=width1; j<width2; j++) {
      eroded[i*w+j] = image[i*w+j];
      for(l1=-mask_size2; l1<=mask_size2; l1++) {
                for(l2=-mask_size2; l2<=mask_size2; l2++) {
                        if(!OutOfBorders(j+l2,i+l1,width1,width2,height1,height2) &&
                                (image[(i+l1)*w+j+l2] < eroded[i*w+j]) )
                                        eroded[i*w+j] = image[(i+l1)*w+j+l2];
                }
      }
    }
  }
  buffer=        m_pBinaryImageBuffer;
  m_pBinaryImageBuffer      =        m_pBinaryImageBuffer1;
  m_pBinaryImageBuffer1     =        buffer;
  return;
}
// PROCEDURE: OnDilate
// TASK: Dilate image for morphology filter
void CHueTracker::OnDilate()
{
 int          i,j;
 int          l1, l2;
 int                          height1,height2,width1,width2,w;
 unsigned char*dilated;
 unsigned char*image;
 unsigned char *buffer;
  int          mask_size2 = m_MaskSize>>1;
  image          =        m_pBinaryImageBuffer;
  dilated        =        m_pBinaryImageBuffer1;
  w                       =        m_un32ImageWidth;
  height1        =        m_WindowPositionY1;
  height2        =        m_WindowPositionY2;
  width1         =        m_WindowPositionX1;
  width2         =        m_WindowPositionX2;
  for(i=height1; i<height2; i++) {
    for(j=width1; j<width2; j++) {
      dilated[i*w+j] = image[i*w+j];
      for(l1=-mask_size2; l1<=mask_size2; l1++) {
                     for(l2=-mask_size2; l2<=mask_size2; l2++) {
                              if(!OutOfBorders(j+l2,i+l1,width1,width2,height1,height2) &&
                                       (image[(i+l1)*w+j+l2] > dilated[i*w+j]) )
                                       dilated[i*w+j] = image[(i+l1)*w+j+l2];
                     }
      }
    }
  }
  buffer=        m_pBinaryImageBuffer;
  m_pBinaryImageBuffer      =        m_pBinaryImageBuffer1;
  m_pBinaryImageBuffer1     =        buffer;
  return;
}


// PROCEDURE: OnOpen
// TASK: Sub function for morphology filter
void CHueTracker::OnOpen()
{
  OnErode();
  OnDilate();
  return;}
```

```
// PROCEDURE: OnClose
// TASK: Sub function for morphology filter
void CHueTracker::OnClose()
{
  OnDilate();
  OnErode();
  return;
}
// PROCEDURE: OnShowResult
// TASK: Output the morphology filtered image buffer
void CHueTracker::OnShowResult()
{
        int j, k, w, bp=3,w1,k1,k2;
        unsigned char *buf, *buf1;
        w  = m_un32ImageHeight*m_un32ImageWidth*bp;
        w1 = m_un32ImageHeight*m_un32ImageWidth;
        k1 = m_CenterX + m_CenterY*m_un32ImageWidth - 3;
        k2 = m_CenterX + m_CenterY*m_un32ImageWidth + 3;
        /* the buffer is used to store the original image */
        buf = m_pun8ImageBuffer;
        /* the buffer is used to store the created binary image */
        buf1 = m_pBinaryImageBuffer;
        for (j=0, k=0 ; j<w ; j+=bp, k++)
        {
                if(buf1[k]==1)
                {
                /* to visualize it */
                        /*
                buf[0+j]=255;
                buf[1+j]=255;
                buf[2+j]=255;
                        */
                }
                if(buf1[k]==2)
                {
                /* to visualize it */
                buf[0+j]=0;
                buf[1+j]=255;
                buf[2+j]=0;
                }
                if(k>k1 && k<k2)
                {
                        buf[1+j] = 0;
                        buf[2+j] = 0;
                }
        }
        return;
}
// PROCEDURE: OnTargetCenter
// TASK: Calculate the center of the recognized target object in the image plane
```

```
void CHueTracker::OnTargetCenter()
{
        int height1, height2, width1, width2, width;
        int i,j;
        int r11,r22,g11,g22,m,threshold2,threshold3;
        int rL1,rL2,gL1,gL2;
        unsigned char *buffer, *buffer1, R, G, B, r;
        long int X,Y;
        static int counter = 0;
        char bufferx[20];
        height1 = m_WindowPositionY1;
        height2 = m_WindowPositionY2;
        width1  = m_WindowPositionX1;
        width2  = m_WindowPositionX2;
        width   = m_un32ImageWidth;
        buffer  = m_pBinaryImageBuffer;
        buffer1 = m_pun8ImageBuffer;
        m_CenterX = m_CenterY = X = Y = m_Total = 0;
        counter++;
        if(counter == 7)
        {
                r11 = m_r1;
                r22 = m_r2;
                g11 = m_g1;
                g22 = m_g2;
                rL1 = m_rL1;
                rL2 = m_rL2;
                gL1 = m_gL1;
                gL2 = m_gL2;
                threshold2 = m_Threshold2;
                threshold3 = m_Threshold3;
        }
        for(i=height1;i<height2;i++)
        {
                for(j=width1;j<width2;j++)
                {
                        if(buffer[i*width+j] == 1)
                        {
                                Y += i;
                                X += j;
                                m_Total++;
                                if(counter != 7) goto label;
                                B = buffer1[(i*width+j)*3];
                                G = buffer1[(i*width+j)*3+1];
                                R = buffer1[(i*width+j)*3+2];
                                m = R+G+B;
                                // if the intensity is too small
                                if(m<threshold2) goto label;
                                // select the smallest one
                                r = R;
```

```
                                        if(G<r) r = G;
                                        if(B<r) r = B;
                                        // if the saturation is too small
                                        if(m*(100-threshold3)<300*r) goto label;
                                        // to find the new target hue region
                                        if((r11*(R-B)+r22*(G-B))<0 && (rL1*(R-B)+rL2*(G-B))>0)
                                        {
                                                r11 = B - G;
                                                r22 = R - B;
                                                goto label;
                                        }
                                        if((g11*(R-B)+g22*(G-B))>0 && (gL1*(R-B)+gL2*(G-B))<0)
                                        {
                                                g11 = B - G;
                                                g22 = R - B;
                                        }
label:      ;
                                }
                        }
        }



        if(counter == 7)
        {
                // update the hue region
                m_r11 = r11;
                m_r22 = r22;
                m_g11 = g11;
                m_g22 = g22;
                // reset the counter
                counter = 0;
        }
        if(m_Total != 0)
        {
        m_CenterX = X/m_Total;
        m_CenterY = Y/m_Total;
        }
        // if the target is not found
        if(m_Total == 0)
        {
                m_CenterX = 0;
                m_CenterY = 0;
                m_WindowPositionY1 = 0;
                m_WindowPositionY2 = m_un32ImageHeight;
                m_WindowPositionX1 = 0;
                m_WindowPositionX2 = m_un32ImageWidth;
        }
}
```

```
// PROCEDURE: OnTargetSize
// TASK: Calculate the size of the recognized target object in the image plane
void CHueTracker::OnTargetSize()
{
        int height1, height2, width1, width2, width, height;
        int i,j;
        unsigned char *buffer;
        long int X, Y, Total;
        height1 = m_WindowPositionY1;
        height2 = m_WindowPositionY2;
        width1  = m_WindowPositionX1;
        width2  = m_WindowPositionX2;
        width   = m_un32ImageWidth;
        height  = m_un32ImageHeight;
        buffer  = m_pBinaryImageBuffer;
        m_MomentX = m_MomentY = X = Y = Total = 0;
        for(i=height1;i<height2-1;i++)
        {
                for(j=width1;j<width2;j++)
                {
                        if(buffer[i*width+j] == 1)
                        {
                                X += (m_CenterX - j)*(m_CenterX - j);
                                Total++;
                                goto label;
                        }
                }
label:  for(j=width1;j<width2;j++)
                {
                        if(buffer[(i+1)*width - j] == 1)
                        {
                                X += (m_CenterX - (width-j))*
                                                 (m_CenterX - (width-j));
                                Total++;
                                goto label1;
                        }
                }
label1:   ;
        }
        if(Total != 0)
        {
                m_MomentX = X/Total;
        }
        Total = 0;
        for(j=width1;j<width2;j++)
        {
                for(i=height1;i<height2-1;i++)
                {       if(buffer[i*width+j] == 1)
                        {
                                Y += (m_CenterY - i)*(m_CenterY - i);
                                Total++;
                                goto label2;
                        }
                }
label2: for(i=height1;i<height2-1;i++)
                {
                        if(buffer[(height-1-i)*width + j] == 1)
                        {
                                Y += (m_CenterY - (height-i))*(m_CenterY - (height-i));
                                Total++;
                                goto label3;
                        }
                }
label3:    ;
        }

        if(Total != 0)
        {
                m_MomentY = Y/Total;
        }
        return;
}
// PROCEDURE: OnWindowSizeAndPosition
// TASK: Calculate the new size of the window surrounding the target object, which will be
//                 processed in the next loop
void CHueTracker::OnWindowSizeAndPosition()
{
        int W = m_un32ImageWidth;
        int H = m_un32ImageHeight;
        // The window's half size
        m_WindowHeight = (int)(0.042*m_MomentY + m_Tolerance);
        m_WindowLength = (int)(0.042*m_MomentX + m_Tolerance);
        // The upper-left corner position of the window
        m_WindowPositionX1 = m_CenterX - m_WindowLength;
        if(m_WindowPositionX1 < 0 ) m_WindowPositionX1 = 0;
        m_WindowPositionX2 = m_CenterX + m_WindowLength;
        if(m_WindowPositionX2 > W) m_WindowPositionX2 = W;
        m_WindowPositionY1 = m_CenterY - m_WindowHeight;
        if(m_WindowPositionY1 < 0) m_WindowPositionY1 = 0;
        m_WindowPositionY2 = m_CenterY + m_WindowHeight;
        if(m_WindowPositionY2 > H) m_WindowPositionY2 = H;
        // if the target is not found reset the window's size
        if( m_CenterX == 0 && m_CenterY == 0)
        {
                m_WindowPositionX1 = 0;
                m_WindowPositionX2 = W;
                m_WindowPositionY1 = 0;
                m_WindowPositionY2 = H;
        }
        return;
```

```
}

// PROCEDURE: SearchTarget
// TASK: Target tracking procedure: the function is given one grabbed frame (image)
//      and must position the camera so the detected target object is centered in the
//      image plane and the distance to this target object is calculated.
//      Repeat this function for continuous target tracking
//      Returns 0 if target is not found (means target is not positioned in the middle)

BOOL CHueTracker::SearchTarget(BYTE* image)
{
        unsigned char VV = 0x18;
        unsigned char WW = 0x14;
        unsigned char ch1,ch2,ch3,ch4,ch5,ch6,ch7,ch8;
        ch1 = 0xF;
        ch2 = 0xC;
        ch3 = 0x9;
        ch4 = 0x0;
        ch5 = 0xF;
        ch6 = 0xE;
        ch7 = 0xD,
        ch8 = 0x4;
        struct timeb t;
        // record the beginning time
        ftime(&t);
        m_BeginTime = t.time*1000+t.millitm;
        m_pun8ImageBuffer = image;
        //copy frame into buffer
        if (m_pun8ImageBuffer==NULL)
                AfxMessageBox("buffer unitialized"); // No image buffer acquired = problem
        static RECT RectImageSize = { 0, 0, 640, 576 };
        //-------------------------------------------------------------------
        // Do the pixel hue classification
        //-------------------------------------------------------------------
        OnHueClassification();
        //-------------------------------------------------------------------
        // Do morphology filtering
        //-------------------------------------------------------------------
        OnMorphologyFiltering();
        //-------------------------------------------------------------------
        // Calculate the target image geometry center
        //-------------------------------------------------------------------
        OnTargetCenter();
        //-------------------------------------------------------------------
        // Calculate the target image size
        //-------------------------------------------------------------------
        OnTargetSize();
        //-------------------------------------------------------------------
        // Show the result
        //-------------------------------------------------------------------
        OnShowResult();
        //-------------------------------------------------------------------
        // Calculate the window's size and position
        //-------------------------------------------------------------------
        OnWindowSizeAndPosition();
        // record the ending time
        ftime(&t);
        m_EndTime = t.time*1000+t.millitm;
        //check condition to be here
        m_bTargetFound = 0;
                if(m_Total != 0)
        {
                m_bTargetFound = 1;
        }
        return m_bTargetFound;
}
```

C++ CODE FOR ROBOT CONTROL

IHM_PilotDlg.h

```
// IHM_PilotDlg.h : header file
//{{AFX_INCLUDES()
#include "jauge12.h"
#include "afficheur.h"
#include "visurob4.h"
//}}AFX_INCLUDES
#if
!defined(AFX_IHM_PILOTDLG_H__3D5EC0E6_1BD8_4AF9_9619_6B4E3343F241__IN
CLUDED_)
#define
AFX_IHM_PILOTDLG_H__3D5EC0E6_1BD8_4AF9_9619_6B4E3343F241__INCLUDED
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "header.h"
#include "CaptureVideoSimple.h"
#include "cal_joystick.h"
#include "cyxfiltre.h"
#include "InterfComm.h"
#include "DefTrames.h"
#include "StaticBitmap.h"
#include "MyPropSheet.h"
#include <vector>
#include "CaptureVideoSimple.h"
#include <wingdi.h>
#include ".\huetracker\huetracker.h"
#define BITMAP_ID 0x4D42 // ID bitmap universel
using namespace std;
/////////////////////////////////////////////////////////////////////////////
// CIHM_PilotDlg dialog
#define FILE_CONFIG "Config_IHM.txt"
#define NAME_ENTREES_ANA( canal ) "CALIB_ENTREES_ANA_"#canal
class stCalib
{
/// Variables
public:
	double m_a;
	double m_b;
	double m_c;
	double m_plage_morte;
	double m_valmin;
	double m_valmax;
	CFiltrePasseBas		m_FiltrePasseBas;
private:
        char    m_Canal[MAX_PATH];
/// Constructeur
public:
        stCalib(char* Canal,double Freq);
        ~stCalib();
};
class CIHM_PilotDlg : public CDialog
{
// Construction
public:
        CIHM_PilotDlg(CWnd* pParent = NULL);// standard constructor
        CCaptureVideo capVideo;
        Joystick *Joy;
        stCalib m_CalibJoyX;
        stCalib m_CalibJoyY;
        vector<stCalib> m_ListCalibInputAna;
        CPoint CadreZoom1;
        CPoint CadreZoom2;
        bool bGettingZoom;
        bool bZoomed;
        UINT TimerID;
/// Liaison RS232
        t_Port          m_ComPort;
        t_BaudRate      m_BdRate;
/// Controle de la Video
        int             m_tailleVideoX;
        int             m_tailleVideoY;
        bool    m_ratio;
        bool    m_ParaAnaIsBipolaire;
//////////////////////////////
CHueTracker *pHue;
//////////////////////////////
// Dialog Data
        //{{AFX_DATA(CIHM_PilotDlg)
        enum { IDD = IDD_IHM_PILOT_DIALOG };
        CStaticm_StaticCamConduiteCtrl;
        CStaticm_StaticCamArCtrl;
        CStaticm_StaticCamPinceCtrl;
        CStaticm_StaticCamAvantCtrl;
        CStaticm_StaticRadio;
        CStaticm_StaticCable;
        CStaticBitmapm_Tor5Ctrl;
        CStaticBitmapm_Tor4Ctrl;
        CStaticBitmapm_Tor3Ctrl;
        CStaticBitmapm_Tor2Ctrl;
        CStaticBitmapm_Tor1Ctrl;
        CStaticm_textTrans;
        CStaticm_robot;
        CStaticm_Aiguille;
        CSliderCtrl     m_Vitesse;
```

```
	CStaticBitmapm_defautTrans;
	CJaugem_JaugeBatterie;
	CVisuRob	m_VitRob;
	CAfficheur	m_AffCapt1Ctrl;
	CAfficheur	m_AffCapt2Ctrl;
	CAfficheur	m_AffCapt3Ctrl;
	CAfficheur	m_AffCapt4Ctrl;
	CAfficheur	m_AffCapt5Ctrl;
	//}}AFX_DATA
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CIHM_PilotDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL
// Implementation
protected:
	HICON		m_hIcon;
	CBrush		m_StaticBrush;
	bool bControlImage;
	tDefTC		defTrame[NB_TC];
	int nbCam;
	int nbComm;
	int cur_sel_cam;
	int cur_sel_comm;
	bool test_touche(int touche);
	int  test_all_touche(bool& IsAlreadyPress);
	void LitParametres();
	void InitailiseTrame();
	bool straight;
	bool left;
	bool right;
	int counter;
	int m_weigth;
	int m_height;
	int counter1;
	int counter2;
	bool RedStart;
	// Generated message map functions
	//{{AFX_MSG(CIHM_PilotDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnControlimage();
	afx_msg void OnTimer(UINT nIDEvent);
	afx_msg void OnFullscreen();
	afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
	afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
	afx_msg void OnProprietes();
	afx_msg void OnDestroy();
```

```
        afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor);
        afx_msg void OnRED_FOLLOW();
        afx_msg void OnSTOP();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous
line.
#endif //
!defined(AFX_IHM_PILOTDLG_H__3D5EC0E6_1BD8_4AF9_9619_6B4E3343F241__IN
CLUDED_)
```

IHM_PilotDlg.cpp

```
// IHM_PilotDlg.cpp : implementation file
#include "stdafx.h"
#include <stdlib.h>
#include <afxwin.h>
#include <process.h>
#include <math.h>
#include <direct.h>
#include <dshow.h>
#include <SYS/timeb.h>
#include <stdio.h>
#include <time.h>
#include "IHM_Pilot.h"
#include "IHM_PilotDlg.h"
#include "ControlImageDlg.h"
#include "PropPage1.h"
#include "PropPage2.h"
#include "MyPropSheet.h"
#include "FullScreenDlg.h"
#include "CyxConstant.h"
#define SAFE_RELEASE(x) { if (x) x->Release(); x = NULL; }
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
extern CIHM_PilotApp theApp;
#define k3  (0.0521488*768/(32.0*1.e-3))
#define k4  (0.0521488*576/(26.6*1.e-3))
#define OutOfBorders(x,y,width1,width2,height1,height2) ((x)<(width1) || (y)<(height1) ||
(x)>=(width2) || (y)>=(height2))
#define NUM_FRAMES_TO_GRAB 360
int X,Total;
/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
```

```
{
public:
	CAboutDlg();
// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL
// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////////
// CIHM_PilotDlg dialog
CIHM_PilotDlg::CIHM_PilotDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CIHM_PilotDlg::IDD, 
pParent),m_CalibJoyX("CALIB_JOY_AXE_X",5),m_CalibJoyY("CALIB_JOY_AXE_Y",5)
{
	//{{AFX_DATA_INIT(CIHM_PilotDlg)
	//}}AFX_DATA_INIT
// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
	bControlImage = false;
	bGettingZoom = false;
	bZoomed = false;
	nbCam = 4;
	nbComm = 2;
/// controle de la video
```

```
        m_tailleVideoX = 720;
        m_tailleVideoY = 576;
        m_ratio = false;
        m_ParaAnaIsBipolaire = false;
        cur_sel_cam = 0;
        cur_sel_comm = 0;
///red_follow declarations
        straight = false;
        left = false;
        right = false;
        counter = 0;
        counter1 = 0;
        counter2= 0;
        RedStart= false;
BEGIN_MESSAGE_MAP(CIHM_PilotDlg, CDialog)
        //{{AFX_MSG_MAP(CIHM_PilotDlg)
        ON_WM_SYSCOMMAND()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
        ON_BN_CLICKED(IDC_CONTROLIMAGE, OnControlimage)
        ON_WM_TIMER()
        ON_BN_CLICKED(IDC_FULLSCREEN, OnFullscreen)
        ON_WM_LBUTTONDOWN()
        ON_WM_LBUTTONUP()
        ON_BN_CLICKED(IDC_PROPRIETES, OnProprietes)
        ON_WM_DESTROY()
        ON_WM_CTLCOLOR()
        ON_BN_CLICKED(IDC_RED_FOLLOW, OnRED_FOLLOW)
        ON_BN_CLICKED(IDC_STOP, OnSTOP)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP();
}strPosOCX;
/////////////////////////////////////////////////////////////////////////////
// CIHM_PilotDlg message handlers
BOOL CIHM_PilotDlg::OnInitDialog()
{
        CDialog::OnInitDialog();
        DWORD id_int;
        // Add "About..." menu item to system menu.
                m_weigth=640;
                m_height=576;
                pHue = (CHueTracker*)new CHueTracker(m_weigth,m_height);
        // IDM_ABOUTBOX must be in the system command range.
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);
        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
                CString strAboutMenu;
                strAboutMenu.LoadString(IDS_ABOUTBOX);
                if (!strAboutMenu.IsEmpty())
                {
        pSysMenu->AppendMenu(MF_SEPARATOR);
        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
                }
        }
        // Set the icon for this dialog.  The framework does this automatically
        //  when the application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);                     // Set big icon
        SetIcon(m_hIcon, FALSE);            // Set small icon
        /// Lecture du fichier de paramètres
        LitParametres();
        /// Definir un Joystick
        Joy = GetFirstJoystick((int)theApp.m_hInstance, (int)m_hWnd, false);
        if(!Joy->good())
                AfxMessageBox("Le joystick n'est pas branché !") ;
        /// Video Intialization
        HRESULT hr = capVideo.InitializeVideo(m_hWnd) ;
        if(FAILED(hr) || !capVideo.StartCompositeVideo())
                AfxMessageBox("Probleme de capture image Video !") ;
        else
                capVideo.FullImage(m_tailleVideoX, m_tailleVideoY);
CenterWindow(CWnd::GetDesktopWindow() ) ;
CheckRadioButton(IDC_TRANS_CABLE, IDC_TRANS_RF, IDC_TRANS_CABLE);
CheckRadioButton(IDC_CAM_CONDUITE, IDC_CAM_AVANT, IDC_CAM_AV);
        Interf.SetComPort(m_ComPort);
        Interf.SetBaudRate(T_4800);
        Interf.SetDataLength(TAILLE_TRAME_OUT);
        if(Interf.InitComm() == NOK)
AfxMessageBox("Impossible d'ouvrir le port comm spcécifié, spécifier un nouveau dans
Propriétés IHM");
        SetWindowPos(NULL, 0,0,1024, 768, SWP_SHOWWINDOW);
        /// Positionement de l'IHM
m_JaugeBatterie.SetWindowPos(&wndTop ,245, 630, 0, 0, SWP_NOSIZE  |
SWP_SHOWWINDOW);
m_VitRob.SetWindowPos(&wndTop,790, 630, 0, 0, SWP_NOSIZE  |
SWP_SHOWWINDOW);
m_VitRob.SetAngle(0.0);
m_AffCapt1Ctrl.SetWindowPos(&wndTop   , 340, 620, 0, 0, SWP_NOSIZE  |
SWP_SHOWWINDOW);
m_AffCapt2Ctrl.SetWindowPos(&wndTop   , 427, 620, 0, 0, SWP_NOSIZE  |
SWP_SHOWWINDOW);
m_AffCapt3Ctrl.SetWindowPos(&wndTop   , 513, 620, 0, 0, SWP_NOSIZE  |
SWP_SHOWWINDOW);;
m_AffCapt4Ctrl.SetWindowPos(&wndTop   , 608, 620, 0, 0, SWP_NOSIZE  |
SWP_SHOWWINDOW);
m_AffCapt5Ctrl.SetWindowPos(&wndTop   , 690, 620, 0, 0, SWP_NOSIZE  |
SWP_SHOWWINDOW);
/// Callage de la vitesse
        m_Vitesse.SetRange(-100, 100, TRUE);
        m_Vitesse.SetPos(0);
        m_Vitesse.ShowWindow(SW_SHOW);
GetDlgItem(IDC_CADRE_VIDEO)->SetWindowPos(NULL, 0, 0, m_tailleVideoX+5,
m_tailleVideoY+5, SWP_SHOWWINDOW);
if(capVideo.getIsVideo())
                capVideo.RespectRatio(m_ratio);
/// On initilialise la trame a 0
        InitailiseTrame();
        m_StaticBrush.CreateSolidBrush(::GetSysColor(COLOR_BTNFACE));
/// creation de la tache de visualisation
        TimerID=SetTimer(1, 200, NULL);
/// La tache d'affichage est valide
        IsIHMOk=true;
/// creation de la tache d'acquisition
InterfaceComhThread =(HANDLE)_beginthread(startInterfaceCom, 0xF4240, &id_int);
return TRUE;  // return TRUE  unless you set the focus to a control
//////////tracking
}
void CIHM_PilotDlg::InitailiseTrame()
{
        /// On initilialise la trame a 0
        memset(Global_OutData,0,TAILLE_TRAME_OUT);
        Global_OutData[defTrame[CMD_DIRECTION].octet-1]          = 0x80;
        Global_OutData[defTrame[CMD_AVANCE].octet-1]             = 0x80;
        Global_OutData[defTrame[CMD_AVANT_BRAS].octet-1]         = 0x80;
        Global_OutData[defTrame[CMD_BRAS].octet-1]               = 0x80;
}
void CIHM_PilotDlg::OnTimer(UINT nIDEvent)
{
        static bool attente_reponse = false;
        static bool IsAlreadyPush=false;
        BYTE OutData[TAILLE_TRAME_MAX],vitesse=0,direction=0;
        double vit=0, dir=0;
        static int tentative = 0;
        char str[50];
        int res,touche;
        strTrame theTrameTM;
        bool IsAlreadyPress=false;
        unsigned char * the_image;
        /// Verrue spécialement pour rafraichir la jauge
        /// seulement quand on ne tire pas sur la batterie (moteur ,..)
        bool IsRefresh=true;
        /// Rafraichissement Joystick
        if(Joy->good())
        {
                /// Lecture du joystick
                Joy->UpdateState();
                /// Calibration de la mesure du joystick
                vit = m_CalibJoyX.m_a*Joy->GetAxisPos(1)*Joy->GetAxisPos(1)
                        + m_CalibJoyX.m_b*Joy->GetAxisPos(1)
                + m_CalibJoyX.m_c;
            dir = m_CalibJoyY.m_a*Joy->GetAxisPos(0)*Joy->GetAxisPos(0)
                + m_CalibJoyY.m_b*Joy->GetAxisPos(0)
                + m_CalibJoyY.m_c;
            if(Joy->GetIsForceFeedback())
                Joy->SetForceFeedback(100,0);
        }
        /// Gestion plage morte
vit = (vit>-m_CalibJoyX.m_plage_morte && vit<m_CalibJoyX.m_plage_morte) ? 0.0:vit;
dir = (dir>-m_CalibJoyY.m_plage_morte && dir<m_CalibJoyY.m_plage_morte) ? 0.0:dir;
        //vit = m_CalibJoyX.m_FiltrePasseBas.Getvalue(vit);
        //dir = m_CalibJoyY.m_FiltrePasseBas.Getvalue(dir);
        /// Gestion des saturations
        vit = (vit>1.0) ? 1.0:vit;
        vit = (vit<-1.0) ? -1.0:vit;
        dir = (dir>1.0) ? 1.0:dir;
        dir = (dir<-1.0) ? -1.0:dir;
        /// Transformation en données Castor
        vit = 0.5*(-1*vit + 1);
        dir = 0.5*(-1*dir + 1);
        /// Transformation ens octets
        vitesse= (BYTE)(vit*255.0);
        //if(vitesse < 0x90 && vitesse > 0x70) vitesse = 0x80;
        direction = (BYTE)(dir*255.0);
        //if(direction < 0x87 && direction > 0x7A) direction = 0x80;
        /// Initialisation du contenu de la trame
        memset(OutData, 0, TAILLE_TRAME_OUT);
        /// Gestion des touches de fonction pour envoi ordres dans trame TC
        /// le bouton 1 commande la LOCOMOTION
        if(Joy->good() && Joy->GetButtonPos(1))
        {
            OutData[defTrame[CMD_DIRECTION].octet-1]    = direction ;
            OutData[defTrame[CMD_AVANCE].octet-1]               = vitesse;
            OutData[defTrame[CMD_AVANT_BRAS].octet-1]           = 0x80;
            OutData[defTrame[CMD_BRAS].octet-1]                 = 0x80;
            double dir_radian = (dir*PI);
            m_Vitesse.SetPos((int)(100 - 200*vit));
            if(vit<0.5)
                m_VitRob.SetAngle(dir_radian);
            else
                m_VitRob.SetAngle(-dir_radian);
            IsRefresh=false;
        }
        /// le bouton 2 commande le BRAS
        else if(Joy->good() && Joy->GetButtonPos(2))
        {
            OutData[defTrame[CMD_DIRECTION].octet-1] = 0x80;
            OutData[defTrame[CMD_AVANCE].octet-1] = 0x80;
            OutData[defTrame[CMD_AVANT_BRAS].octet-1] = vitesse;
            OutData[defTrame[CMD_BRAS].octet-1] = direction;
```

```
          m_Vitesse.SetPos(0);
          m_VitRob.SetAngle(-PI/2.0);
          IsRefresh=false;
}
else
{
          OutData[0] = 0x80;
          OutData[1] = 0x80;
          OutData[2] = 0x80;
          OutData[3] = 0x80;
          m_Vitesse.SetPos(0);
          m_VitRob.SetAngle(-PI/2.0);
}
if(RedStart)
{
          the_image = capVideo.GrabData();
          bool b1 = pHue->SearchTarget((BYTE*) the_image);
          if(b1 = 1)
                    {
                              X = pHue->m_CenterX;
                              Total = pHue->m_Total;
                              counter1 = abs(X-320)/150;
                                        if(Total!=0)
                                                  {         if(Total<15000 && Total>500)
                                                                      counter2 = 8000/Total;
                                                            else if(Total>15000)
                                                                      counter2=1;
                                                  }
                                        else
                                                            counter2=0;
                    }
          if(X!=0)
                    {
                              if(X<320)
                                        {
                                                  right=true;
                                                  straight=true;
                                        }
                              else
                                        {
                                                  left=true;
                                                  straight=true;
                                        }
                    }
}
if (straight)
                    {
                              for(int i=0;i<counter2;i++)
                              {
                              dir=0.5;
```

```
                                vit=1;
                                direction = (BYTE)(dir*255.0);
                                vitesse= (BYTE)(vit*255.0);
        OutData[defTrame[CMD_DIRECTION].octet-1]   = direction ;
        OutData[defTrame[CMD_AVANCE].octet-1]       = vitesse;
        OutData[defTrame[CMD_AVANT_BRAS].octet-1]          = 0x80;
        OutData[defTrame[CMD_BRAS].octet-1]                = 0x80;
                                }
                                straight=false;
        if (left)
                {
                        for(int i=0;i<counter1;i++)
                                {
                                dir=0.3;
                                vit=0.6;
                                direction = (BYTE)(dir*255.0);
                                vitesse= (BYTE)(vit*255.0);
OutData[defTrame[CMD_DIRECTION].octet-1]   = direction ;
OutData[defTrame[CMD_AVANCE].octet-1]       = vitesse;
OutData[defTrame[CMD_AVANT_BRAS].octet-1]          = 0x80;
OutData[defTrame[CMD_BRAS].octet-1]                = 0x80;
                                }
        left=false;
        }
        if (right)
        {
                for(int i=0;i<counter1;i++)
                {
                                dir=-0.3;
                                vit=0.6;
                                direction = (BYTE)(dir*255.0);
                                vitesse= (BYTE)(vit*255.0);
OutData[defTrame[CMD_DIRECTION].octet-1]   = direction ;
OutData[defTrame[CMD_AVANCE].octet-1]              = vitesse;
OutData[defTrame[CMD_AVANT_BRAS].octet-1]          = 0x80;
OutData[defTrame[CMD_BRAS].octet-1]                = 0x80;
                }
right=false;
    }
}
CDialog::OnTimer(nIDEvent);
}
void CIHM_PilotDlg::OnRED_FOLLOW()
{
        // TODO: Add your control notification handler code here
        RedStart = true;
}
void CIHM_PilotDlg::OnSTOP()
{
        // TODO: Add your control notification handler code here
```

```
             counter=0;
            RedStart=false;
            right=false;
            left=false;
            straight=false;
}
```