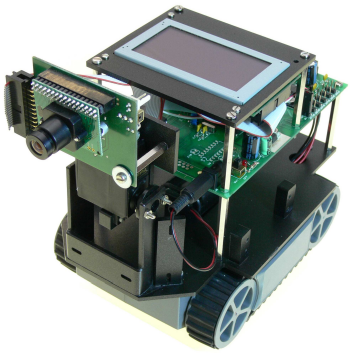
POB TECHNOLOGY

Robot Maker

For Innovative Leisure and For Education



POB-Bot User's Manual

POB EYE

POB POB-PROTO

POB LCD-128

Table of contents

1 INTRODUCTION 5

2 POBEYE MODULE 7

2.1 Hardware description of the POB-Eye 8

2.2 Power supply 9

2.3 Programming and execution 10

2.4 Serial port (UART) 11

2.5 HE10 Connector 12

2.6 POB-EYE bus 14

2.7 Using POB-Eye's bus from an extension board 16

2.8 Sensor CMOS connector 22

2.9 Dimensions of the module 22

2.10 Connecting POBEYE to a computer 23

3 POB-PROTO 25

3.1 Board elements description 26

3.2 PORTA, PORTC and PORTD configuration: 36

3.3 Assembly examples with POB-PROTO: 37

3.4 Mechanical information: 40

4 POB-LCD 42

4.1 Hardware description 42

4.2 Connecting POB-LCD128 with POB-EYE 43

4.3 Drawing with POB-LCD128 44

5 POB-TOOLS 47

5.1 POB-TOOLS installation 48

5.2 POB-TOOLS configuration 51

5.3 Manage a project application 52

5.4 POB-Compiler 53

5.5 POB-Loader 56

5.6 POB-Bitmap 58

5.7 POB-Pattern 64

5.8 POB-Terminal 69

6 LIBRARY HELP FILES 71

7 SAMPLE APPLICATION 73

Document management

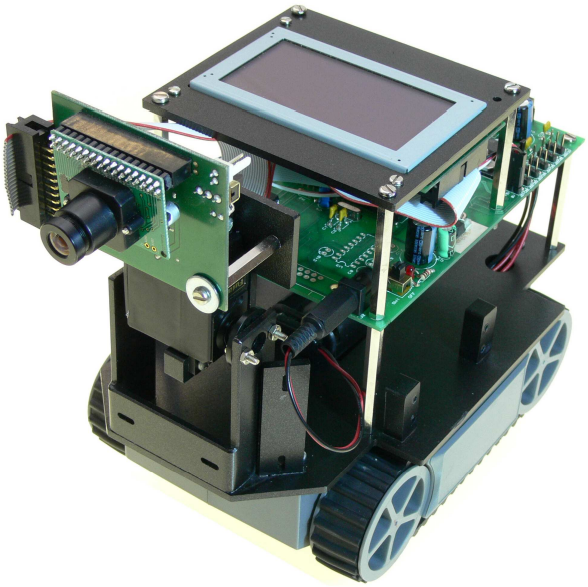
Filename	POB-Technology US.doc
Creation date	14/10/2005
Author	Pierre Séguin

POB-Technology contacts

Address	POB-TECHNOLOGY 4, rue nicéphore niépce 69 680 CHASSIEU, FRANCE
Mail address	contact@pob-technology.com
Phone	+33 (0)4 72 43 02 36
Fax	+33 (0)4 78 58 04 92

1 Introduction

The **POB-Bot** is a mobile programmable robot. It has been designed to allow you to build your own robot.



The **POB-Bot** includes a **POB-Eye**, the intelligent color camera; a **POB-Proto**, the Input/Output management board; and a **POB-LCD128**, the graphical screen.

The **POB-Eye** is the eyes, the heart and the head of your robot. Develop with C, Java or Basic, using the appropriate **POB-Tools**; you can also develop with the **POB-Builder**, the graphical development tools from **POB**.
Libraries are available to easily program native form recognition, moves, and servomotors. Other boards are connected to the **POB-Eye** via the POB Bus. But an I2C Bus is available to connect other robot parts.

The **POB-Prot** is managed by the **POB-Eye**. With a few code you act on the DC motors, servomotors. You check incoming values from sensors and you manage the joystick for a graphical user interface.

The **POB-LCD128** is the screen. You can watch in real time which is coming from the camera or use it for a graphical user interface.

POB EYE

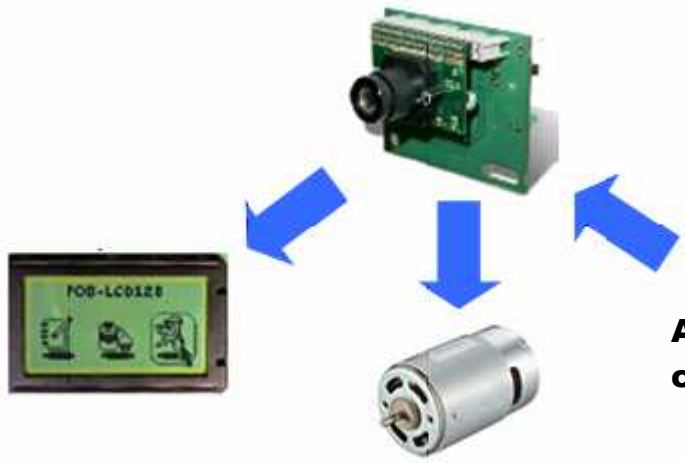
Color Camera Module

Programmable with the Pob-Tools
(en C, Java or Basic) or the Pob-Builder

Pob-Technology Document – All Right Reserved

2 POBEYE Module

The POB-EYE is a camera device used for real time pattern recognition. Due to its 15 I/O, its serial port and its I2C bus, POB-EYE was designed to be the center application for your project. The POB-EYE is delivered with the necessary tools for a fast and simple development.



Analog to digital converter

Flash	128Koctets
Ram	64Koctets
Bus I2C	Yes
UART	Yes
Input/Output	15
Dimensions	64.28 * 49.50 mm
Size of the captured image	88 * 120 pixels
Color image	Yes (1 byte per component)
Memory size of an image	32Koctets
Input voltage	6V to 9V
Input courant	2A maximum
Digital signals voltage	3.3V (5 Volt tolerant)
Digital signals courant	10mA maximum

2.1 Hardware description of the POB-Eye

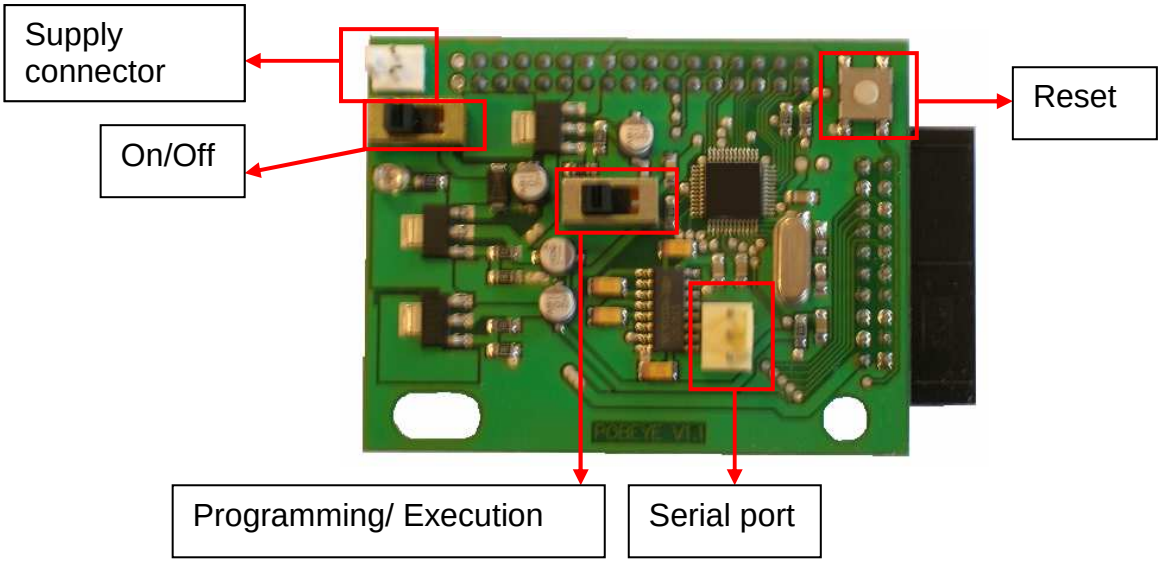


Figure 1 POB-EYE module (supply face)

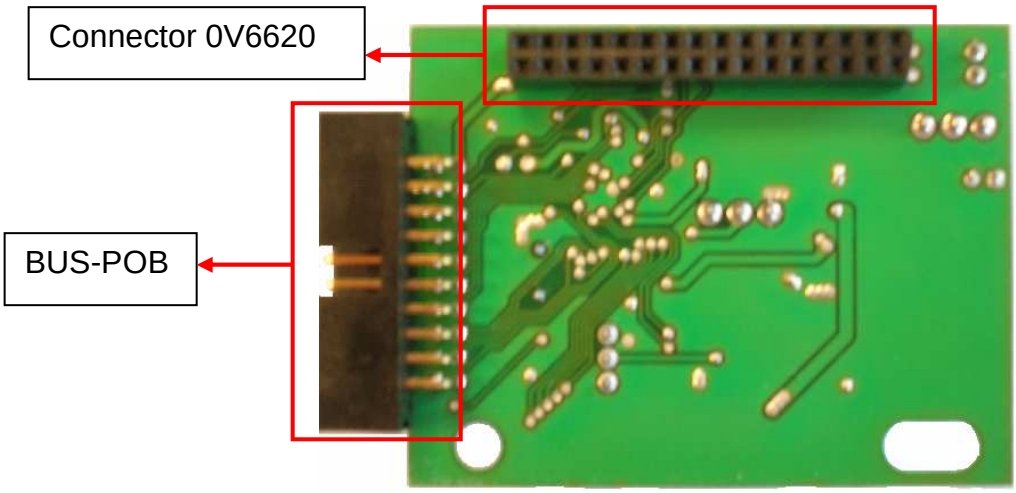


Figure 2 POB-EYE module (CMOS connector face)

2.2 Power supply

The power supply must be between **6V and 12V**. There are two ways of supplying the module:

Dedicated connector

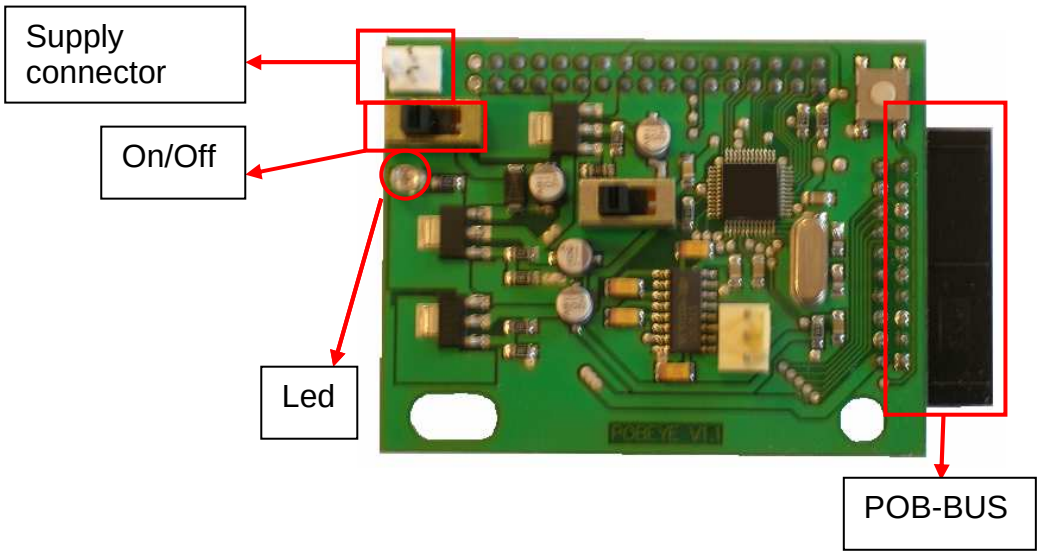


Figure 3 POBEYE power supply

To switch on the module, the power cable must be connected and the "on/off" switch and set to "on" position.

The power cable is supplied with the POBEYE module.

HE10 connector

By using pin 2 (+Power) and 20 (GND) of the input/output bus, the user can supply POBEYE module with a daughterboard through HE10 connector. The on/off switch from the POB-EYE board is no longer used in this configuration, the user can therefore use the supply switch form his own board.

Remark:

Once the POBEYE module is switched on, the green led comes on.

Warning: If you choose to supply your POB-EYE through the POB-BUS make sure not to use the supply connector of the POB-EYE otherwise you might damage the board.

2.3 Programming and execution

Module programming

To load a new program in flash memory, you need to set the « programming/execution switch » to the programming position and switch on the module.

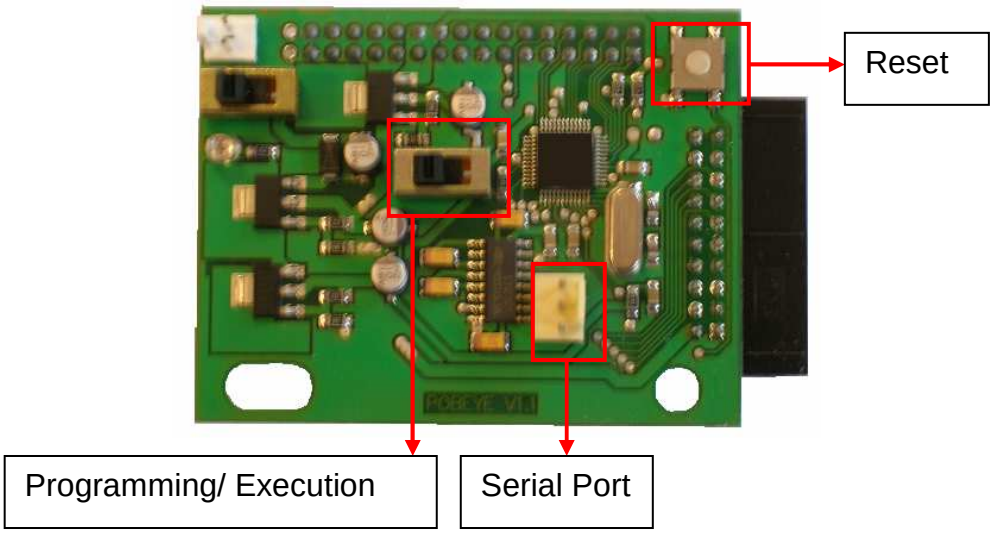


Figure 4 POBEYE programming

Remark: If the module is already on, simply switch on the programming position and press the reset button.

Program execution

To run a program, set the "Programming/execution" switch to the execution position, and then switch on the module.

Remark: If the module is already on, simply switch on the execution position, then press the reset button. Nevertheless, if you are using other daughter boards than the POB-LCD128 you might have to switch off the whole system and then switch it back on.

2.4 Serial port (UART)

The serial port is used for module programming and external communication while the program is running.

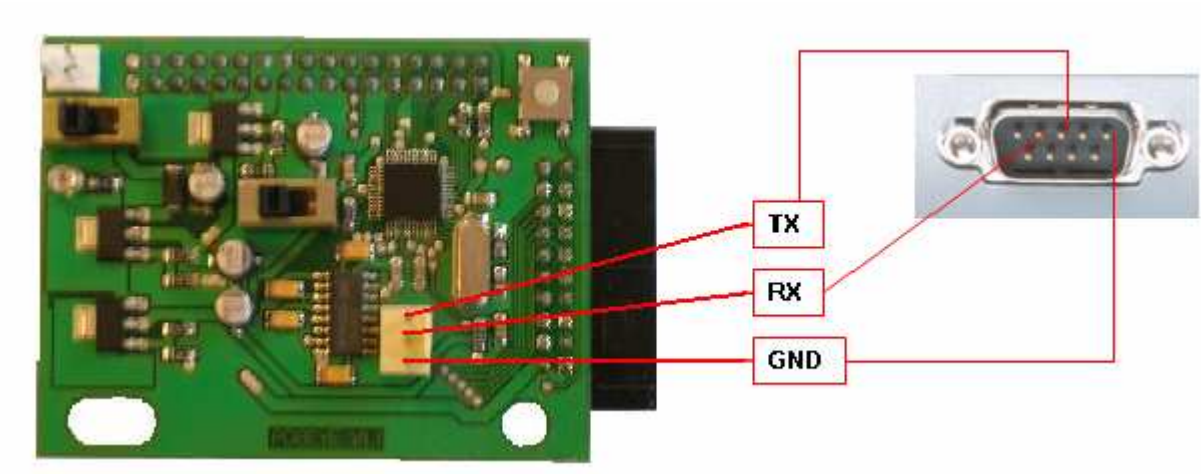
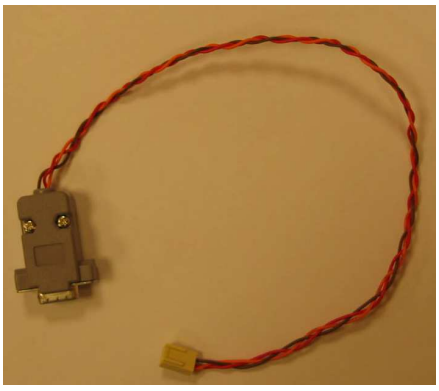


Figure 5 Serial port

Remark: To connect POBEYE to a computer, you need a standard cable. This cable is supplied with the POBEYE module.



Caution: To convert the various voltage levels, a MAX232 is already set up in the POB-EYE module. The values Tx, Rx and GND are showed from a computer point of view.

2.5 HE10 Connector

In addition to the 0V and supply voltage, the I2C bus and the 15 input/output are on the HE10 connector.

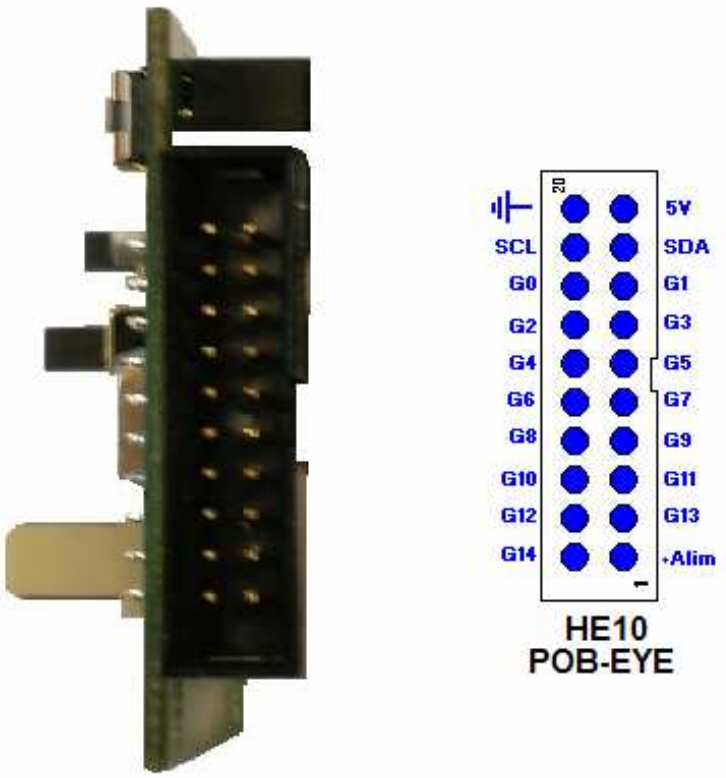


Figure 6 I2C bus and input/output

The HE10 connector allows the user to connect an extension board to the POBEYE module.

This connector gives access to the I2C bus (which has already its pull up resistance). In addition to the I/O, there is the +5V, the Ground and the POBEYE power supply.

Warning: The POBEYE module can directly be supplied by an extension board that provides the +Power.

Otherwise an extension board can use the +5V from the POB-EYE for its own power supply.

Warning: The output voltage of the input/output is of 3.3V. Despite a 5 volt tolerance, a current of more than 10 mA is not possible.

Information concerning the HE10 connector is defined as following:

Pin	Type	Number on the HE10 connector
G0	Input/Output	16
G1	Input/Output	15
G2	Input/Output	14
G3	Input/Output	13
G4	Input/Output	12
G5	Input/Output	11
G6	Input/Output	10
G7	Input/Output	9
G8	Input/Output	8
G9	Input/Output	7
G10	Input/Output	6
G11	Input/Output	5
G12	Input/Output	4
G13	Input/Output	3
G14	Input/Output	2
SCL	Output	18
SDA	Input/Output	17
+5V	Output	19
+Power	Input/Output	1
GND		20

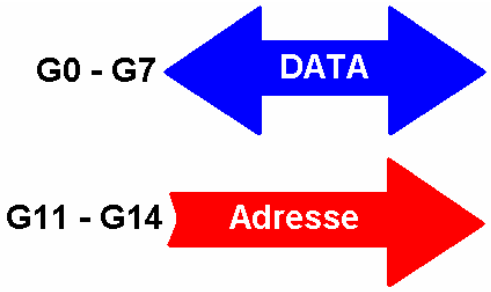
Figure 6 HE10 connector

2.6 POB-EYE bus

POB-Technology uses the input/output on the HE10 connector as an address and data bus.

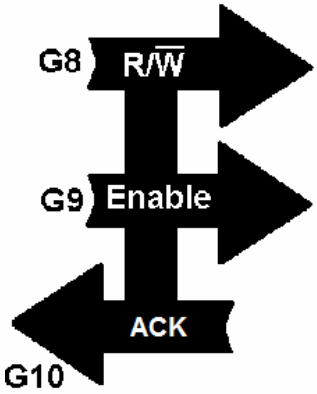
The pins G0 to G7 are the data bus. The pins G11 to G14 are the address bus. This system makes it possible to have 16 addressable cards on the POBEYE bus.

This means it is possible to have on the same bus: cards for direct current motors, card for servomotors management or others such as distance measurements sensors.



To complete the bus, there are 3 other signals:

- G8 pin: « R/W » signal indicates a read/write to the peripheral. If G8 is in low state, POBEYE writes data to the extension board. If G8 is in up state, POBEYE reads data from the peripheral.
- G9 pin: « ENABLE » signal, allows signals validation on the bus.
- G10 pin: « ACK » signal, allows to know peripheral states (the peripheral is ready to communicate)



Next table outlines the role of each pin:

Pin	Name on the POBEYE bus	Type	Number on the HE10 connector
G0	DATA 0	Input/Output	16
G1	DATA 1	Input/Output	15
G2	DATA 2	Input/Output	14
G3	DATA 3	Input/Output	13
G4	DATA 4	Input/Output	12
G5	DATA 5	Input/Output	11
G6	DATA 6	Input/Output	10
G7	DATA 7	Input/Output	9
G8	R/W	Output	8
G9	Enable	Output	7
G10	ACK	Input	6
G11	Address 0	Output	5
G12	Address 1	Output	4
G13	Address 2	Output	3
G14	Address 3	Output	2

Figure 7 Bus POBEYE

The peripheral on the POBEYE bus have this address:

Peripheral Address	Peripheral name
0	POBLCD128
1	POBLCD128
2	*POB-PROTO*
3	*reserved*
4	*reserved*
5	*reserved*
6	*reserved*
7	*reserved*
8	*reserved*
9	*reserved*
10	*reserved*
11	*reserved*
12	For the user
13	For the user
14	For the user
15	For the user

The reserved addresses are the ones POB-Technology is using for its peripheral. Nevertheless, they can be used if necessary.

2.7 Using POB-Eye's bus from an extension board

To understand how to use a peripheral on the POB-EYE bus we will use the following electric circuit:

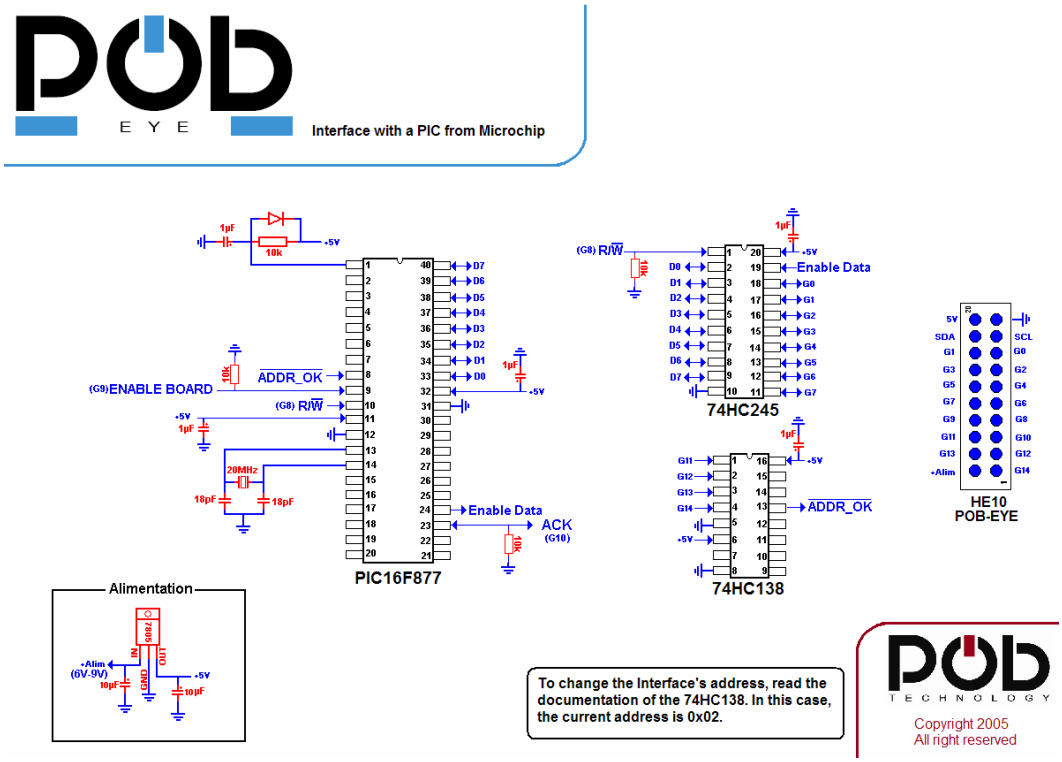


Figure 8 Electronic assemblies for daughter board

This schema helps to understand how the POB-EYE module communicates with an external microcontroller. Using this, could help you in creating your own extensions.

This schema and source code for communication functions are available on the CD-Rom provided with the POB-EYE module.

The source code available is only designed for communication functions between the POB-EYE and the extension card using a PIC16F77 microcontroller.

The electronic circuit is composed of 3 ICs:

74HC138

This IC is used to decode the address from the address bus. When the address is valid, the signal /ADDR_OK is at lower state otherwise when none valid /ADDR_OK is at high state.

In this example, the 74HC138 is configured to work with the 0x02 address which is reserved for POB-Technology, for any further information refer to Figure 10.

74HC245

This IC allows transfer data between POB-EYE and a daughter device. It can be active or not and in this case of OFF-state all the data pins will be put to high impedance. The circuit is active when pin 19 (/ENABLE) is at lower state.

PIC16F877

This IC is the center of the daughter board. Because of its analog inputs and other I/O, the PIC16F877 can add other functionalities to your system. Moreover, the PIC16F877 is a low cost chip and its development tools are free and easy to access.

The names of these signals can be seen on the electric schema

Name	Name on 16F877	Direction	PIN	Description
/ADDR_OK	PORTE 0	Input	8	Indicates that the address is valid
ENABLE_BOARD	PORTE 1	Input	9	Allows POB-EYE to validate the board
Read And Write	PORTE 2	Input	10	Indicates if POB-EYE is in Read or Write mode
ACK	PORTC 4	Input/Output	23	Indicates to POB-EYE that the board is ready
ENABLE_DATA	PORTC 5	Output	24	Validates the data of the 74H245.
	PORTB	Input/Output	33-40	Data transit.

Using POB-Eye's:

All the digital read/write protocols between the POB-EYE and the PIC16F877 are the following:

The card receives data

Wait for:
- Valid address (/ADDR_OK set to 0).
- Board enables (ENABLE_BOARD set to 1).
- POB-EYE in WRITING mode (READ_AND_WRITE set to 0).
- No one using ACK (ACK set to 0).

Configure:
- ACK in output
- PORTB to receive data from the POB-EYE

Set:
- ACK set to 0
- ENABLE_DATA set to 0

Record:
- Data from PORTB

Set:
- ENABLE_DATA set to 1
- ACK set to 1

Wait for:
- Board Enable (ENABLE_BOARD set to 0) by the POB - EYE

Configure:
- ACK set to 0. Little tip to save up time : if ACK becomes an input, PULL-DOWN resistor placed on ACK will act as if ACK was set to 0.

Set:
- ACK set to 0. Warning, do not forget this step otherwise the cards can be damaged.

The card sends data

Wait for:

- Valid address (/ADDR_OK set to 0).
- The board must be enabled (ENABLE_BOARD set to 1).
- POB-EYE set to writing mode (READ_AND_WRITE set to 0).
- No one using ACK (ACK set to 0).

Configure:

- ACK set to output
- PORTB for sending data

Set:

- ACK set to 0
- ENABLE_DATA set to 0
- Data sent to PORTB
- ACK set to 1

Wait for:

- Board enabled (ENABLE_BOARD set to 0)

Set:

- ENABLE_DATA set to 1

Configure:

- ACK set to 0. Little tip to save up time : if ACK becomes an input, the PULL-DOWN resistor placed on ACK will act as if ACK was set to 0.

Set:

- ACK set to 0. Warning, do not forget this step otherwise the cards can be damaged.

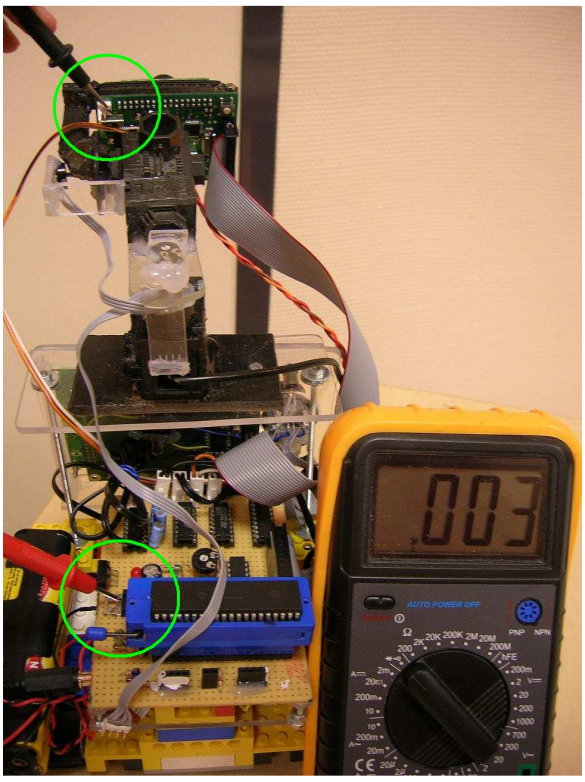
Signals Configuration when adding a daughter board

Nom	I/O	Signal status
/ADDR_OK	Input	
ACK	Input (Becomes an output during communication)	
ENABLE_DATA	Output	High state
PORTB	Input	
ENABLE_BOARD	Input	
READ_AND_WRITE	Input	

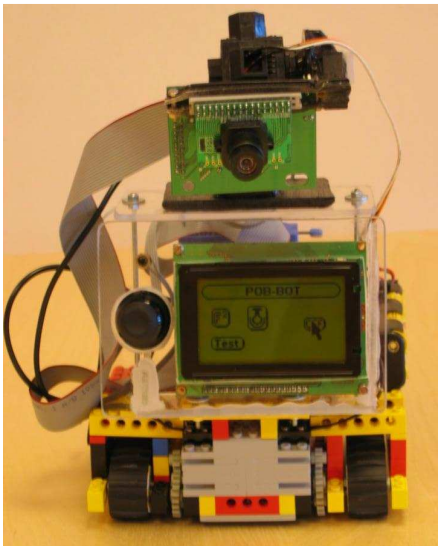
Remarks:

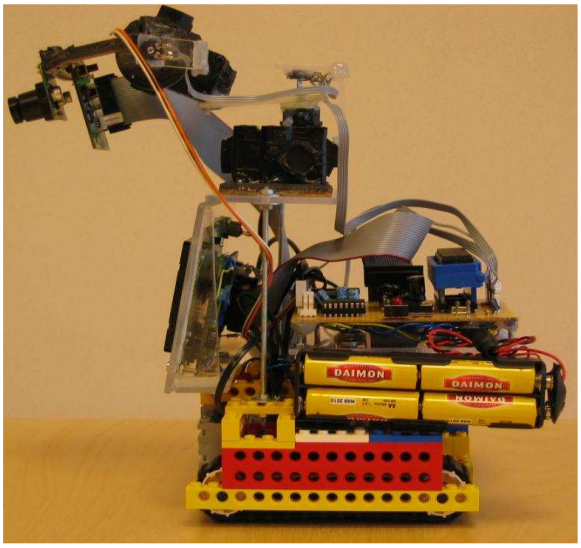
- When a new program is loaded in the POB-EYE, and daughter board is added, it is necessary to unplug and re-plug the whole system for synchronous matter

-When you add daughter board to your POB-EYE, before switching on the supply, you must verify using a multimeter in "diode verification position" that the connection is clearly realized. Simply place one probe to the POB-EYE's GND and the other one to your board's GND. If you have an existing contact, your daughter board is correctly added.



Examples of applications using the POB-EYE bus:





In this example the POB-EYE has 2 daughter boards:

1°) POB-LCD128
POB-LCD128 creates a perfect HMI (Human Machine Interface)

2°) the second daughter board based on figure 10 :

- Gears 4 DC motors (2 for head movements and 2 for traction)
- grabs the position of 4 variable resistor (2 control of the head and 2 for the joystick)
- detects if the pushbutton is pressed.

A video is available on our website.

2.8 Sensor CMOS connector

This connector makes its possible to connect the CMOS OV6620 to the POBEYE module.

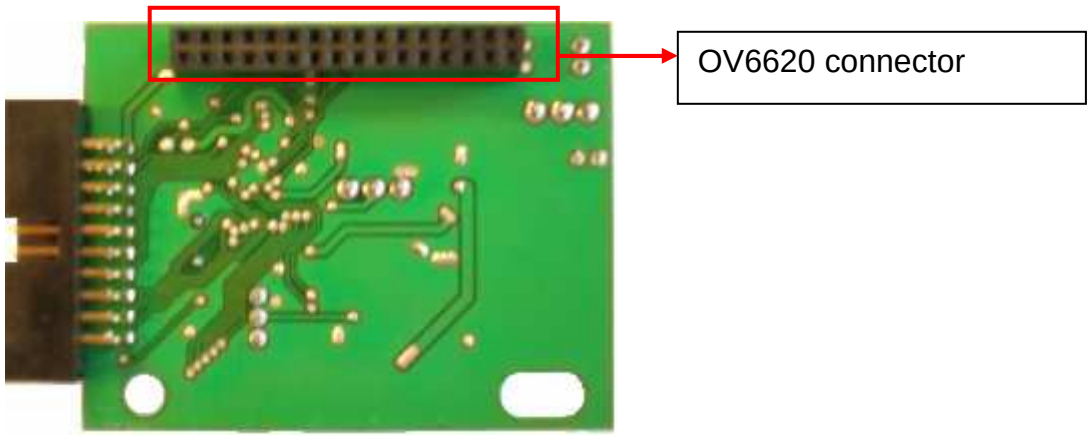


Figure 9 CMOS connector

Warning: The CMOS sensor connects on the left of the connector. To configure the OV6622 register, use the function "*SendToCam*"

2.9 Dimensions of the module

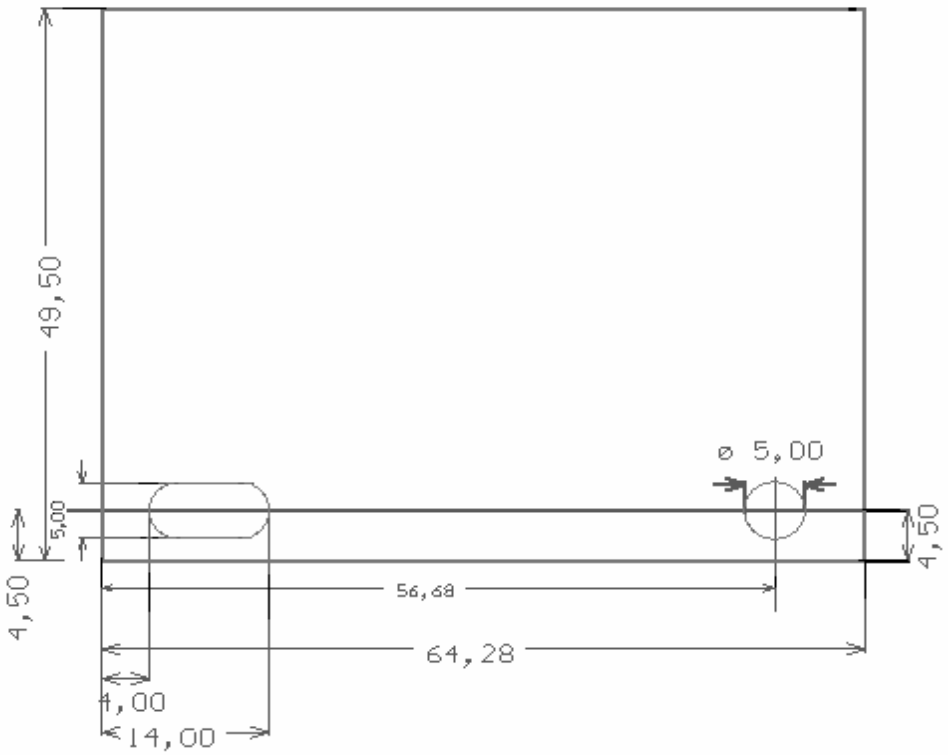
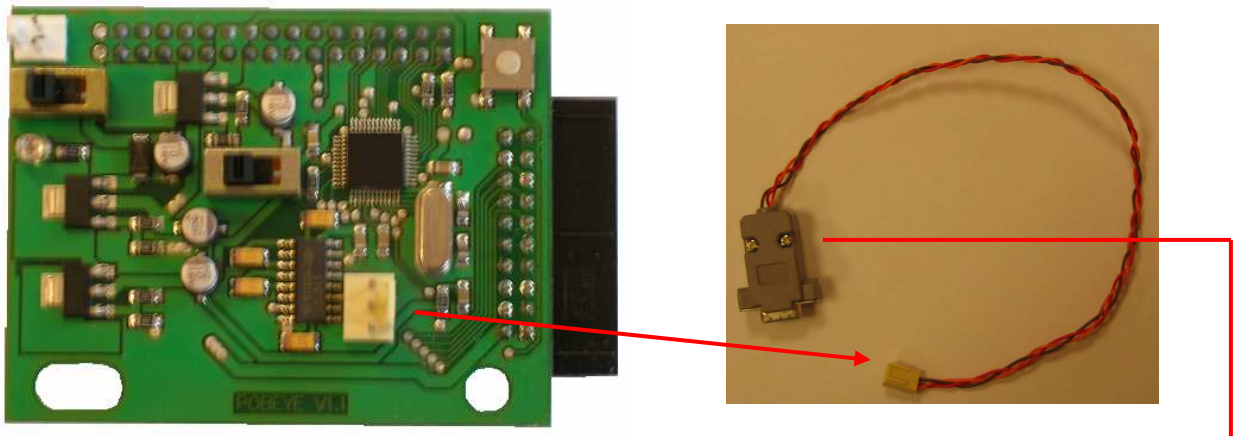


Figure 10 Dimension of the POBEYE

2.10 Connecting POBEYE to a computer

Serial port

It is necessary to connect the supplied cable to POBEYE:

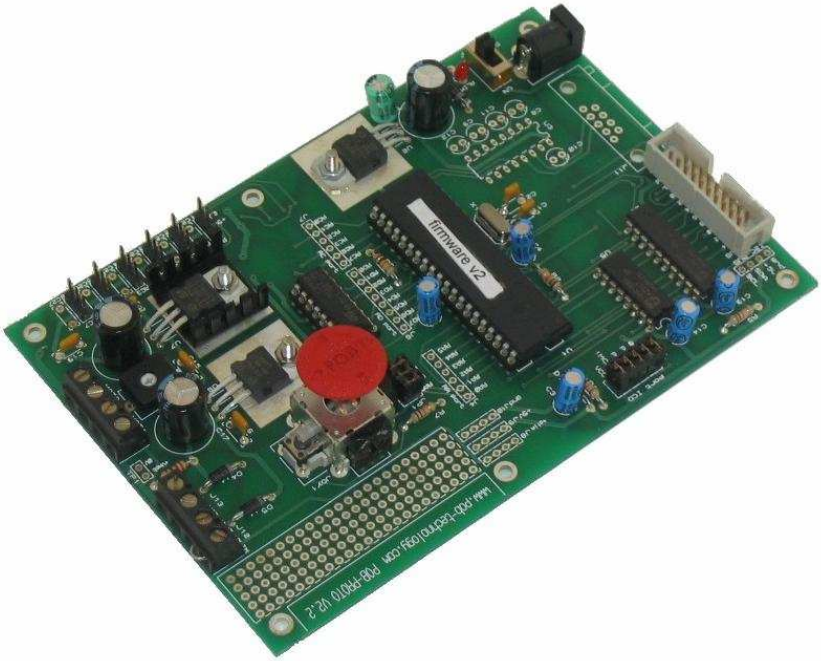


Then link the other end to a serial port available on your computer.

Power supply to POBEYE

Connect the power cable to POBEYE module.

POB-PROTO



Input/Output Board

Manage your sensors with the digital and analog I/O
Bus Pob and I2C

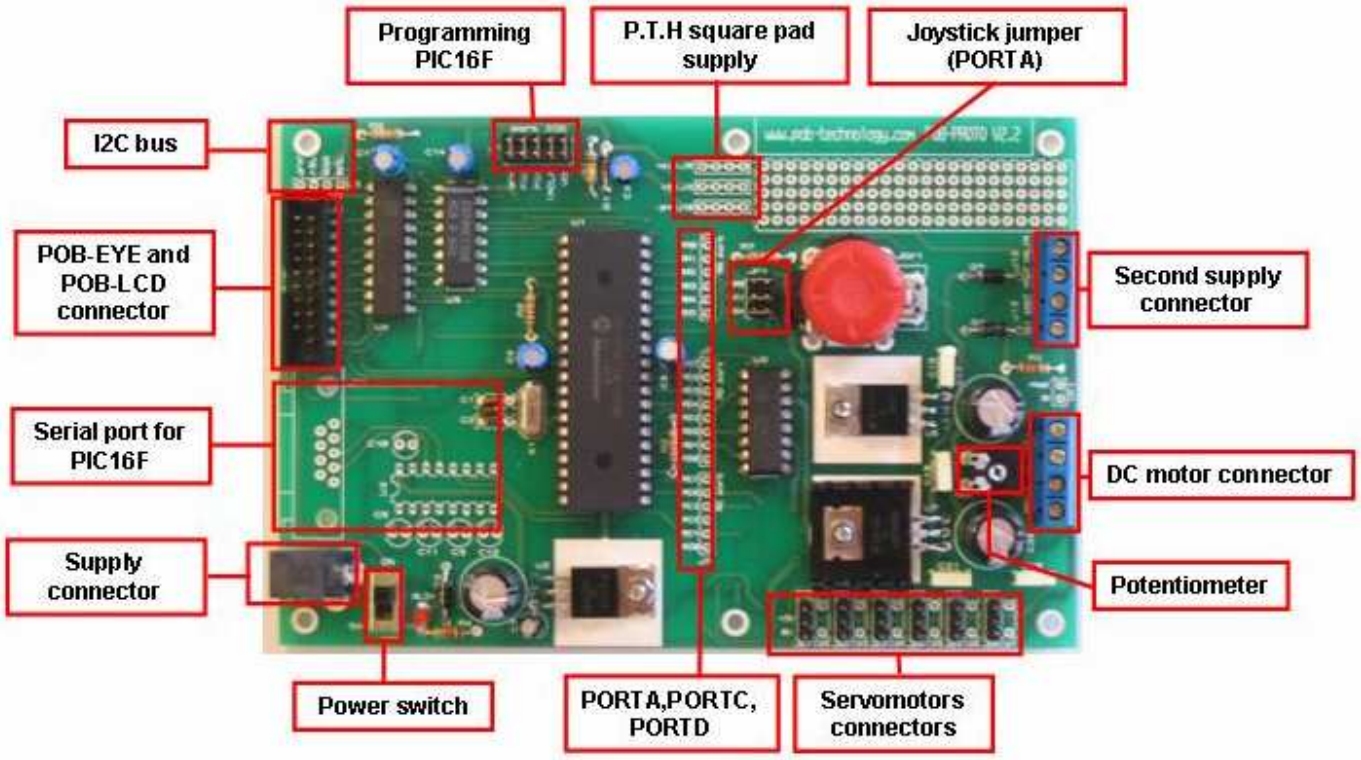
3 POB-PROTO

POB-PROTO was designed for the POB-EYE in order to build a robot very easily. You will find the following:

- 6 connectors to manage servomotors (Futaba)
- 1 analog joystick and its pushbutton.
- An H bridge to gear 2 DC.

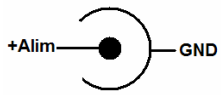
On the CD in POB-PROTO folder you will find examples on how to use the POB-PROTO with all the source code needed.

POB-PROTO description:



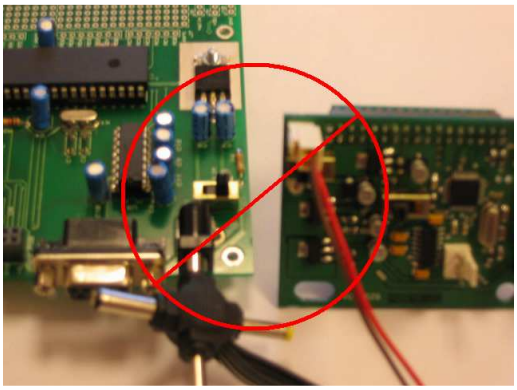
3.1 Board elements description

Board supply:

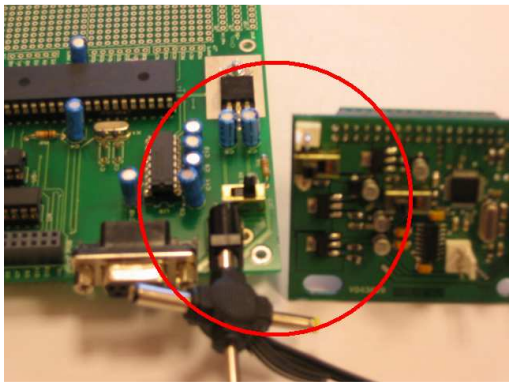


There is a supply connector on the board, make sure to use the right polarity. This connector can then supply the whole system through the POB-bus (on pin 2)

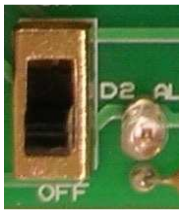
Warning: Make sure you are not using both POB-EYE supply connector and POB-PROTO supply connector.



Do not supply both POB-PROTO and POB-EYE.

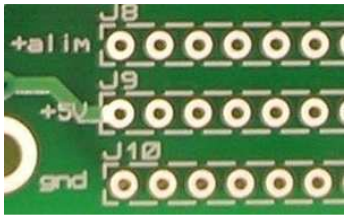


Supply is only on POB-PROTO



You will find a power switch on POB-PROTO that will shut down the whole system.

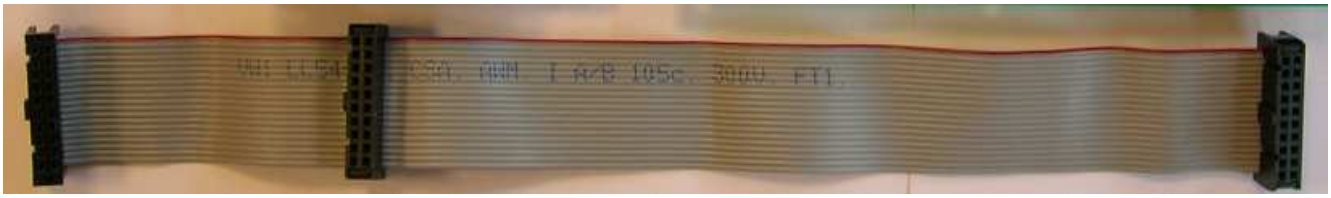
Supply through the P.T.H square pad:



The board has its own P.T.H square pad for supplying other components.

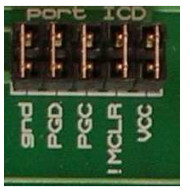
Connection for POB-EYE and POB-PROTO:

The HE10 connector is used to link the POB-EYE to the POB-PROTO.



Warning: If you decide to create your own cable you must do the following: The connectors must be crimped with the head facing the same direction otherwise you might damage your card.

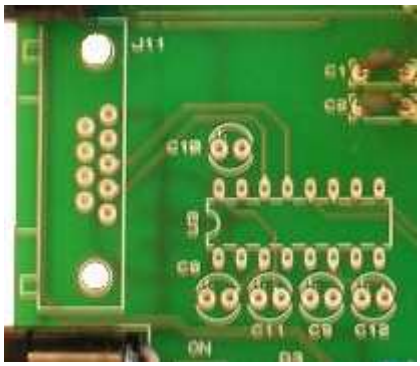
Programming PIC16F877:



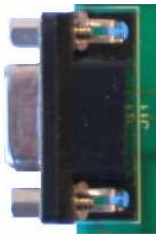
POB-PROTO is supplied with a program connector for the microcontroller. Programming will be done through the ICD2 of the microchip. The user needs to withdraw the jumpers then plug the cable on the lower pins (GND, PGD, PGC, MCLEAR, and VCC).

Once the PIC is programmed, plug back the jumpers. Note that your ICD2 needs its own supply.

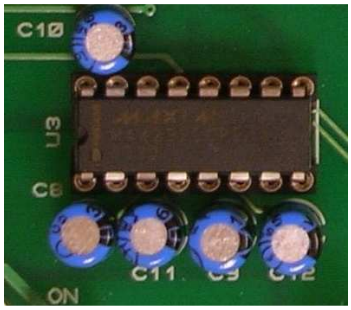
Serial port:



POB-PROTO is supplied with the necessary site to integrate other components to use the PIC16f877 serial port.

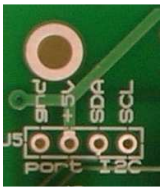


The board has a site for a DB9 connector.



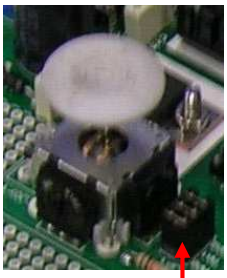
To convert the signal level on the serial connection, POB-PROTO has the site for a MAX232 and its capacitors.

POB-EYE's I2C Bus



On the top right of the POB-PROTO board a layout was designed for the POB-EYE's I2C bus. The signals SDA, SCL, GND and +5V are available.
PULL-UP resistors are already set.

The Joystick



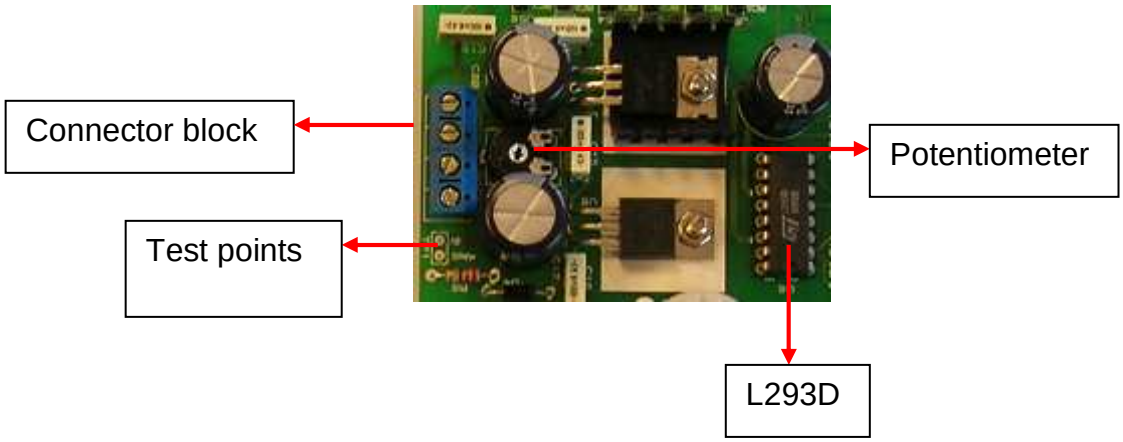
Jumper connections

For example you can use the joystick to control the cursor on the LCD screen. It's working due to its two potentiometer linked to PORTA (RA0 and RA1). Moreover a pushbutton is available on the joystick and linked to RA4.

If the user wants to use PORTA for another application it is possible by removing the jumpers.

If using the joystick you have to use RA3, RA3, and RA5 as analog inputs (see PORTA configuration.)

H Bridge



The H bridge is linked to PORTD (RD0, RD1, RD2, and RD3). It gears the 2 DC motors link to the connector block. To adjust the voltage on the motors, use the potentiometer and if needed you can use a multimeter on the test points.

Caution: The H bridge current cannot exceed 600mA.

The PORTD pin can be configured independently and pins RD4, RD5, RD6, RD7 used for generic usage.

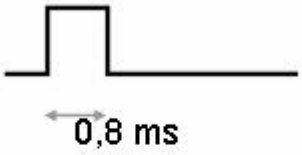
Caution: If you don't need the H bridge simply unplug the L293D from board.

Servomotors connectors

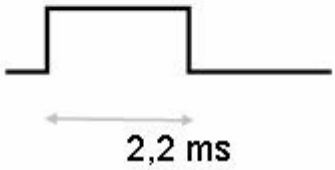
The connectors are plugged to pin RC0, RC1, RC2, RC3, RC6, and RC7 of PORTC.

Because nothing is plugged to these connectors, you can use it for general purpose.

The value to gear servomotors is between 0 and 255.



Value 0 equal to a pulse of 0,8 milliseconds.



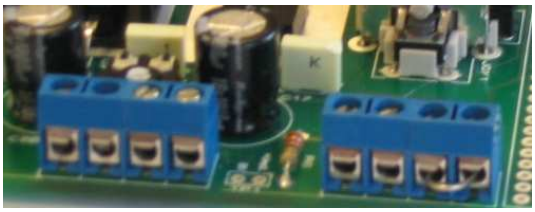
Value 255 equal to a pulse of 2,2 milliseconds..

Warning: If the servomotors require too much current you might shut down the whole system.

To avoid such a problem you need to program the servos movement slowly. For example to move from position 10 to position 200 create a loop incrementing the position 1 per 1.

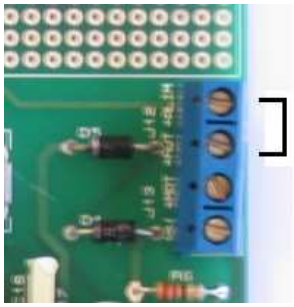
Warning: The tension regulators can be very hot. Never touch, it can burn your self. Never put on it anything, it can destroy it.

Servomotor supply connector:



Because it need too much energy, using CC motors and servomotors can generate noises.
For example, your system can reboot.

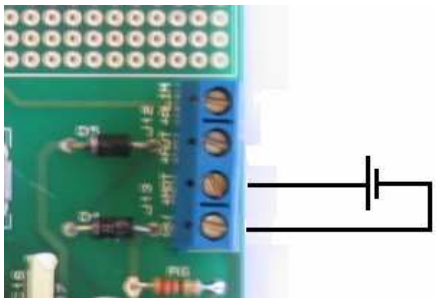
To fix the problem, POB-PROTO board has a connector to add a second alimentation source:



Common supply:

If you want to use the general alimentation, you have to connect a jumper as shown on the left.

The common supply is the default.



Second supply:

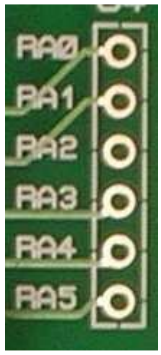
However, if you want to use a second alimentation source, then disconnect the default jumper, and add the source as shown on the left picture. With it, your perturbations disappear.

Cautious: Be careful to respect the schematic, else you should burn your board or create a shortcut which can be make some damages.

PORTA, PORTC and PORTD:

All these ports are already used (connectors, H bridge…) by POB-PROTO. Nevertheless it is possible to unplug the existing material and use the port as wanted.

PORTA



PORTA can work in 2 modes:

- Analog input, except for RA4 which is a digital input of 0 or 255.
- Digital I/O, each pin can be configured independently.

If you are using PORTA for analog inputs, use the function GetPortAnalog with RA4 as a parameter. Note that RA4 is only a digital input taking either the value 0 or 255

In output configuration RA4 works in open-drain, and if needed add a resistor and +5V on RA4.

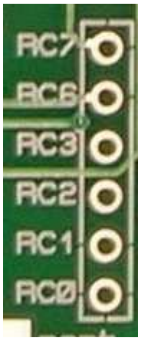
Functions to use PORTA:

UInt8 GetPortA (void)
Return PORTA value.

void SetPortA (UInt8 data)
Set PORTA pins to output.

UInt8 GetPortAnalog (UInt8 RAx)
Return analog value from RAx pin.

PORTC



Each PORTC pin can be configure separately.
You can use RC0, RC1, RC2, RC3, RC6 and RC7 to manage 6 servomotors.

Remarks: If the user wants to reprogram the PIC 16F877, he will then be able to use the serial port on PORTC by adding a MAX232.

Functions to use PORTC:

UInt8 GetPortC (void)
Return PORTC value.

void SetPortC (UInt8 data)
Set PORTC to output.

void SetServoMotor(UInt8 RCx,UInt8 Pos)
Set servomotors position which is connected to RCx

void SwitchOffAllServo(void)
Stop all servomotors control in order to save energy.

void SwitchOnAllServo(void)
Start all servomotors control.

void SwitchOffOneServo(UInt8 Servo)
Stop one particular servomotor's control.

void SwitchOnOneServo(UInt8 Servo)
Start one particular servomotor's control.

PORTD

PORTD is an I/O port; every single pin can be configured separately.

Functions to use the PORTD:

UInt8 GetPortD (void)
Remove value from PORTD

void SetPortD (UInt8 data)
Set PORTD pins to output.

void SendToSapien(UInt8 order)
Send a move order to Robot Sapien.

3.2 PORTA, PORTC and PORTD configuration:

The PIC16F877 program for PORT management is delivered with POB-PROTO.

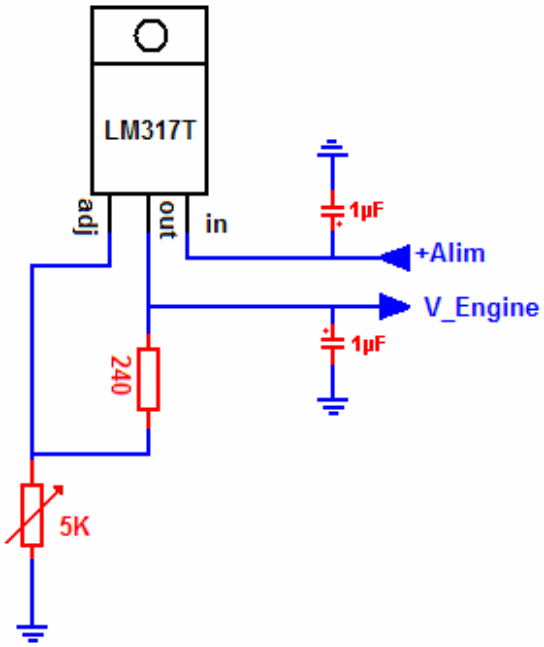
```
PobProto    Proto;

Proto.porta=ALL_PORTA_AS_ANA;
Proto.portc=RC7_AS_DI   | RC6_AS_DI |RC3_AS_DI |RC2_AS_DI|RC1_AS_DI |RC0_AS_SERVO;
Proto.portd=RD7_AS_DI   | RD6_AS_DI |RD5_AS_DI |RD4_AS_DI|RD3_AS_DO |RD2_AS_DO  |RD1_AS_DO  |RD0_AS_DO;

SetPobProto(&Proto);
```

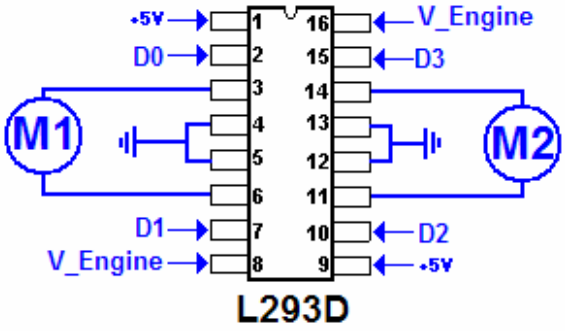
In this example PORTA is set to analog input. RC7, RC6, RC3, RC2, RC1 are configure as inputs and RC0 is configure to manage a servomotor. PORTD, RD7, RD6, RD5, and RD4 are set to digital inputs and RD3, RD2, RD1, RD0 are set to digital outputs.

3.3 Assembly examples with POB-PROTO:

Usually for movements any robots are using 2 DC motors link to a « free » wheel. Motors control requires power electronics.

The assembly needed to POB-PROTO to control DC motors is the following:





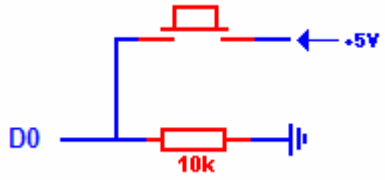
This assembly was done with a L293D (double H bridge). Each motor is managed by 2 microcontroller pins.

Pins D0, D1, D2, and D3 of PORTD are used to control the motors and set to output in your program.

The LM317T is a voltage regulator and with the adjustable resistor it is possible to set the voltage to the wanted value.

Push-button management:

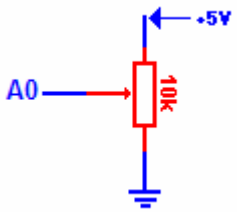
To create your own HMI you need a navigation system control by pushbutton for example. The following assembly shows how to add a pushbutton to the POB-PROTO.



The user needs to configure D0 as an input and read PORTD with the adequate masks to get the correct D0 value.

Analog value acquisition:

This diagram shows how to rely an adjustable resistor to PORTA.

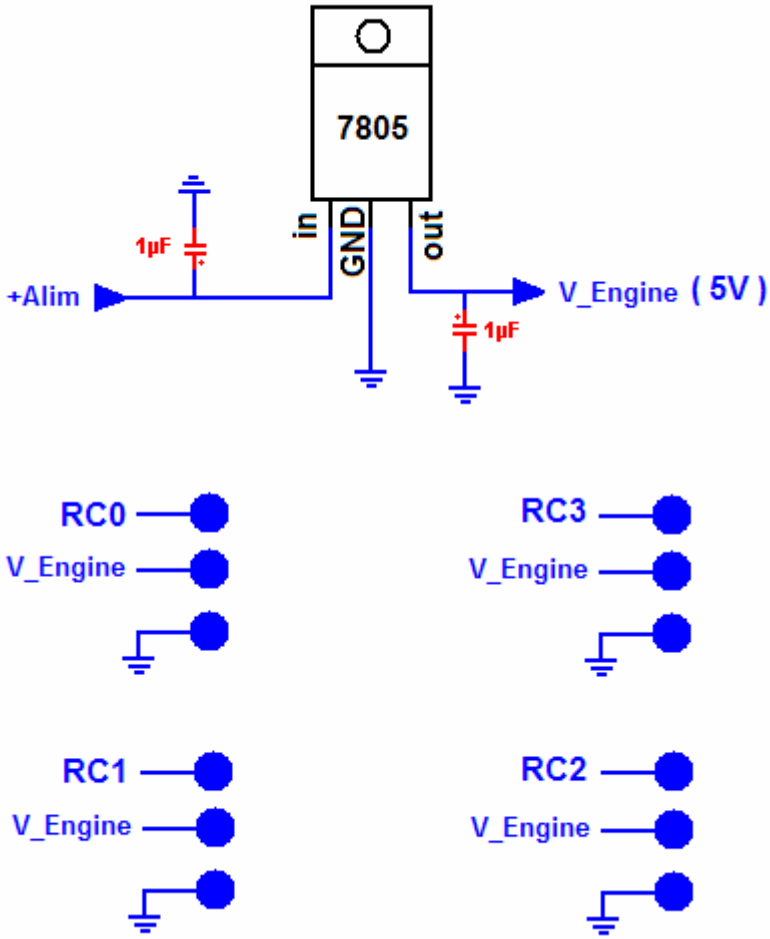


Caution: on PIC16F877, on PORTA, RA0 RA1 RA2 RA3 and RA5 are analog inputs. RA4 is a digital input getting either 0 or 255 when using the GetPortAnalog function.

Servomotors Management:

Only RC0, RC1, RC2, RC3, RC6 and RC7 from PORTC can be used to manage servomotors.

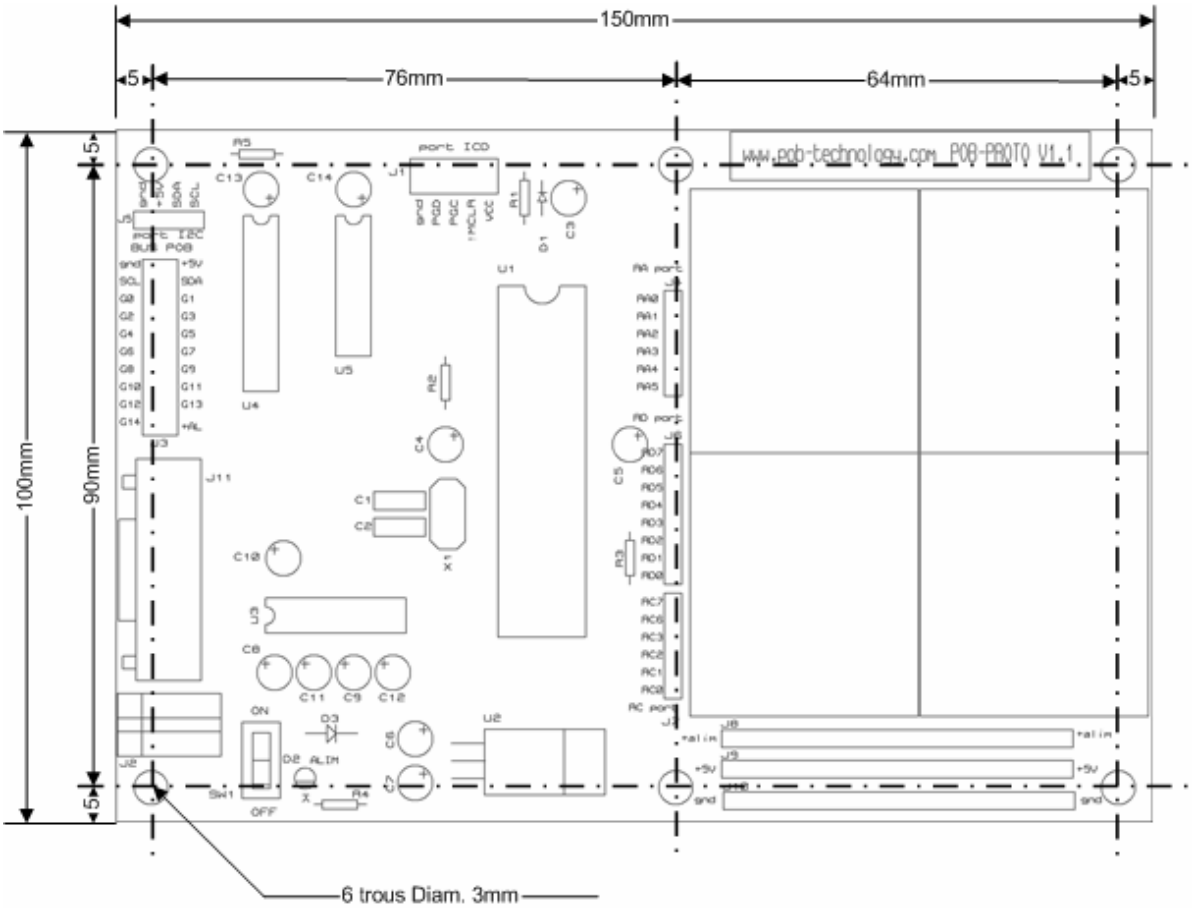
The following diagram shows how to plug a servomotor to the POB-PROTO board.



Caution: To avoid current picks when using the servos, it is advised to use a dedicated regulator to supply the servos.

Remark: The user can configure RC0, RC1, RC2 and RC3 to manage the servos and use RC6 and RC7 as digital Inputs or outputs.

3.4 Mechanical information:



LCD-128

Graphical Screen Module

Watch in real time what is coming from the Pob-Eye
Develop a Graphical User Interface

4 POB-LCD

POB-LCD128 is an extension card for the POBEYE module. POB-LCD128 allows the POB-EYE to display various images. These two modules (POB-EYE and POB-LCD128) work with each other. In fact, POB-LCD128 can display in real-time the images from POB-EYE thus helping the user in developing his own applications.

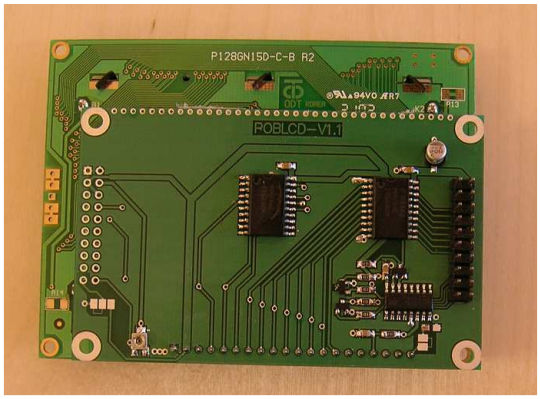
To simplify the drawing on the screen, POB-Technology created the software that will help in integrating images to the application. This software POB-Bitmap is further explained in section POB-TOOLS.

4.1 Hardware description

Technically, le POB-LCD128 is composed of 2 parts :

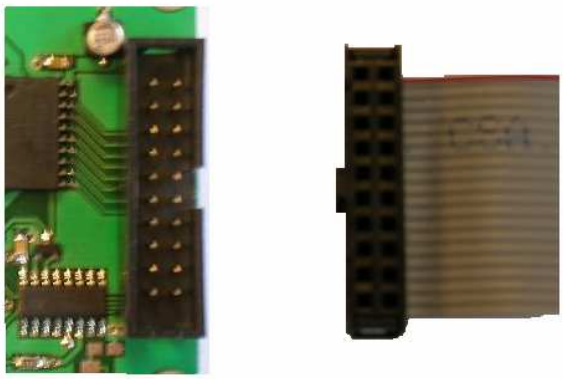
- A LCD of 128 by 64 pixels.

- An electronic interface.

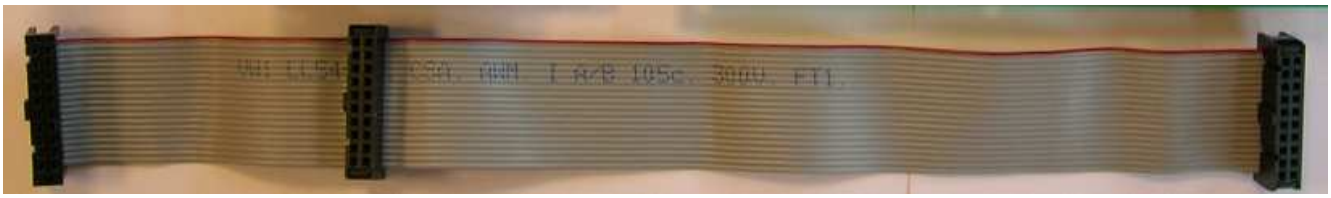


The electronic interface allows POB-LCD128 to work on POB-EYE's bus. It uses the addresses 0 and 1 so that other interface can work on the same bus.

4.2 Connecting POB-LCD128 with POB-EYE



You need to connect the cable supplied by POB-Technology to the POB-LCD128.



Warning: If you decide to create your own cable make sure the connectors are crimped with the head facing the same direction otherwise you might damage your card.

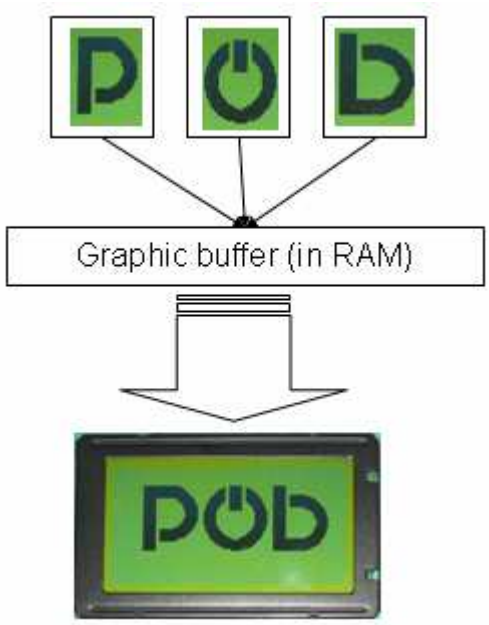
4.3 Drawing with POB-LCD128

The POB-LCD128 is programmed with the library supplied with the module.

Functioning of the graphic buffer

The POB-EYE allows a graphic RAM buffer. All the drawing operations use this buffer, and when finished, the buffer is transferred to the POB-LCD128 memory.

This allows efficient processing of the display and it avoids displaying the pixels one by one on screen in case of processor-intensive calculations on POB-EYE.



With this method any computation will be done directly in POB-EYE's RAM memory. Once the computation is finished, the buffer is sent from RAM memory to the LCD screen.

About pixels

A black and white LCD screen uses one bit to color a pixel. With a byte, the screen displays 8 pixels. The buffer uses 8192 bits (the POB-LCD screen is 128 pixels by 64 pixels) of memory on POB-EYE module.

128*64 pixels = 8129 bits = 1Kbytes

The challenge of this buffer is to have a ratio of "1 bit = 1 pixel". The operations on bits are processor-intensive. The advantage of this "1 bit = 1 pixel" ratio is that it does not use much memory.

In fact, to change a bit value in memory you need to apply a mask. And this mask will be different whether the bit is set to 0 or to 1. This computation can be relatively time consuming because it must be done 8129 times.

That's why using the technique "1 bit = 1 pixel" will need less memory space. Another way of drawing a pixel on the screen is to consider that 1 byte draws 1 pixel. In that case they will be no delay or mask when switching on or off a pixel. Nevertheless, such a technique will take 8 times more memory than the previous one.

128 * 64 pixels = 8129 pixels
1 bit per pixel, buffer is equal to 1Kbytes.
1 byte per pixel, buffer is equal to 8Kbytes.

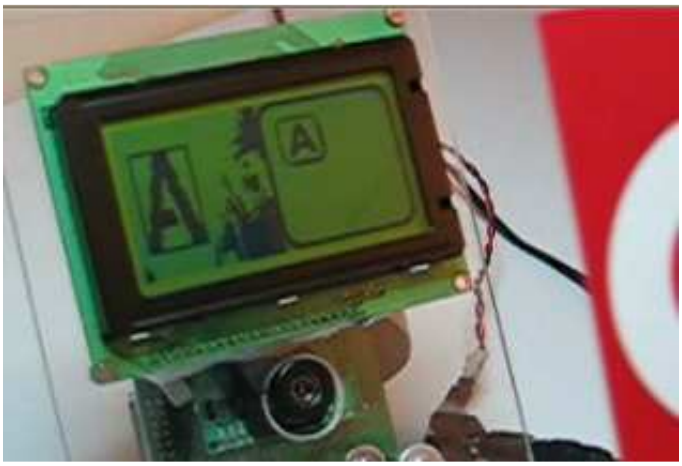
Conclusion:

- Using **1 byte** per pixel allows speeding up the processing of the graphic functions.
- Using **1 bit** per pixel allows saving the POB-EYE memory.

In order to give a free access to the POB-LCD128, a library is given. This library will generate a buffer of 1 bit per pixel or a buffer of 1 byte per pixel.

Screen sharing

Moreover, the library given can either use the whole screen for display or slip the screen in two. The screen-sharing will be of 64 by 64 pixels as shown in the following figure. On one side we have the video in real-time and on the other side the user's interface.



To understand the functions given in the library you can report to the source code of example 4.

Software to program the Pob-Bot

Project managment
Compilers
Upload the programme onto the robot
Manage pictures to screen
Manage forms to recognize
Windows Terminal

POB-TOOLS

POB-COMPILER

POB-LOADER

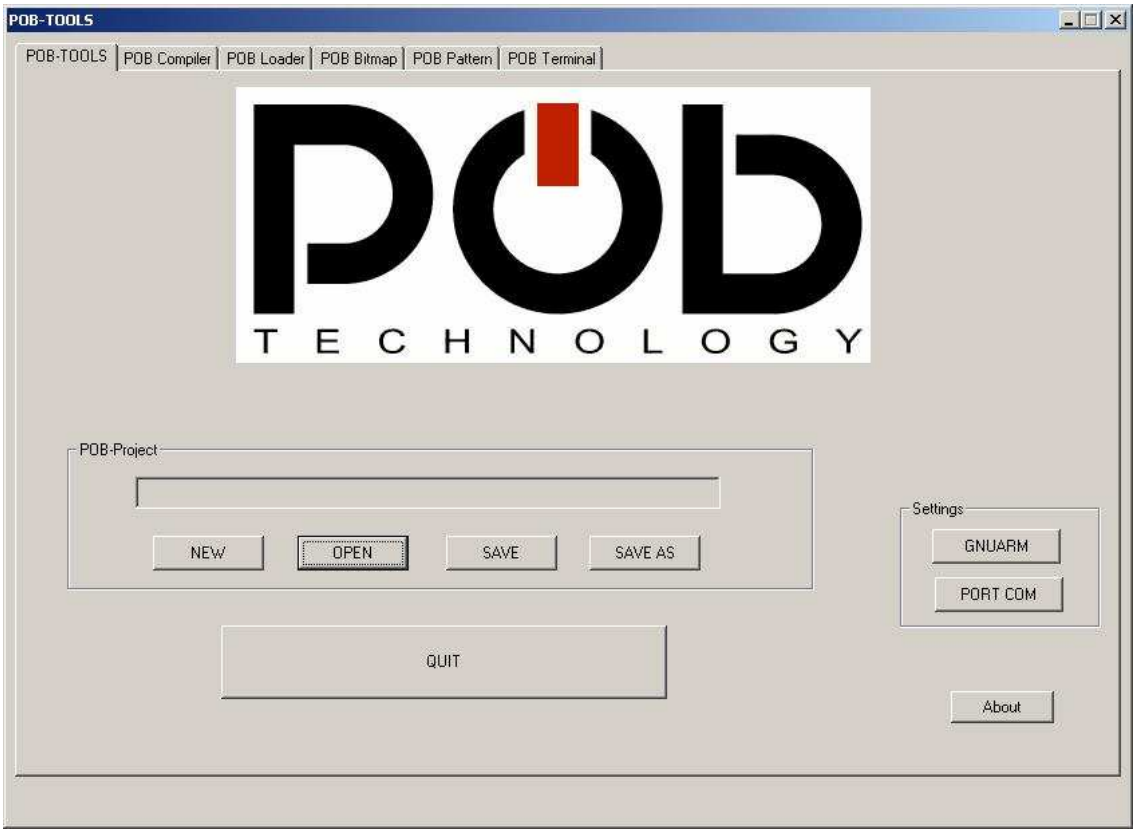
POB-BITMAP

POB-PATTERN

POB-TERMINAL

5 POB-TOOLS

POB-TOOLS are a set of tools for programming, loading and controlling the POBEYE module. This tool is made up of 6 modules:



POB-TOOLS: Allows managing an application project.

POB-Compiler: Enables one-click source file compilation.

POB-Loader: Upload an application to the POBEYE module.

POB-Bitmap: Produce a library of pictures for the POB-LCD128.

POB-Pattern: Create a dictionary of pattern.

POB-Terminal: Help debug your application with the serial port.

5.1 POB-TOOLS installation

On the CD supplied with POBEYE module, you will find two files.

Installation of the GNUARM compiler

The file « **bu-2.15_gcc-3.4.1-c-c++-java_nl-1.12.0_gi-6.0.exe** » is the GNUARM compiler. POB-TOOLS use this compiler to create your application.

To install the GNUARM compiler, click on this file.

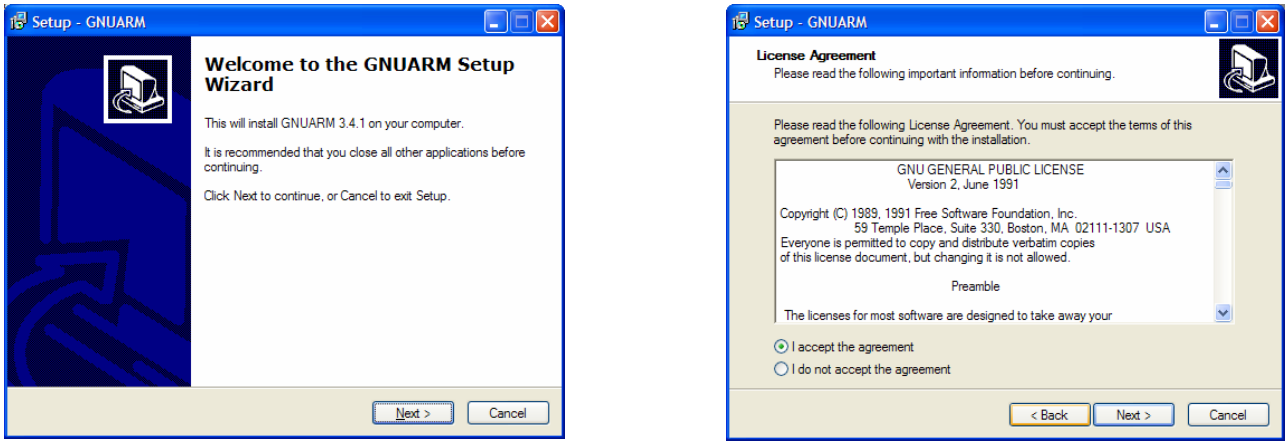


Figure 11 GNUARM introduction and software license: GNU GPL

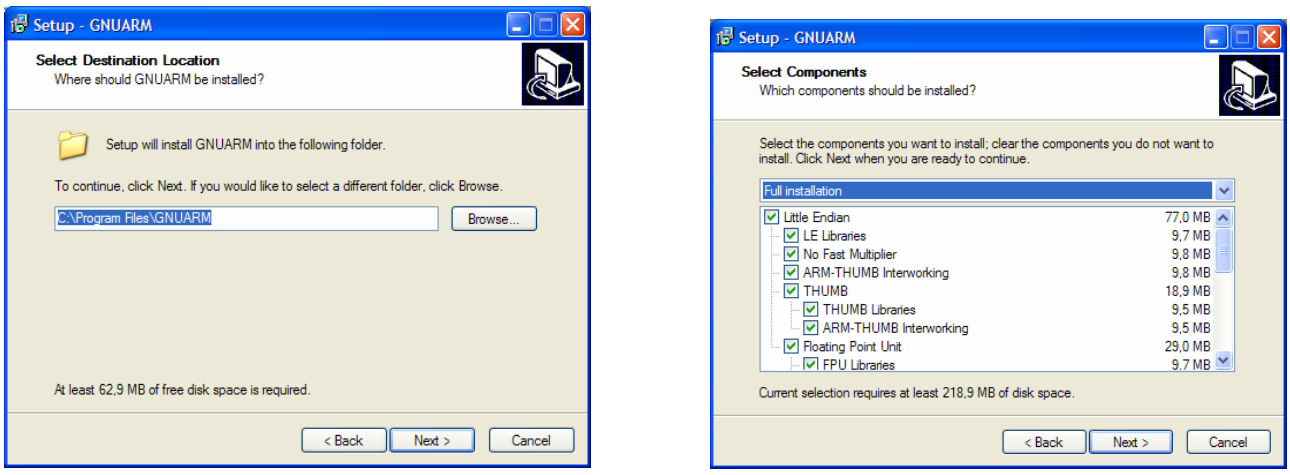
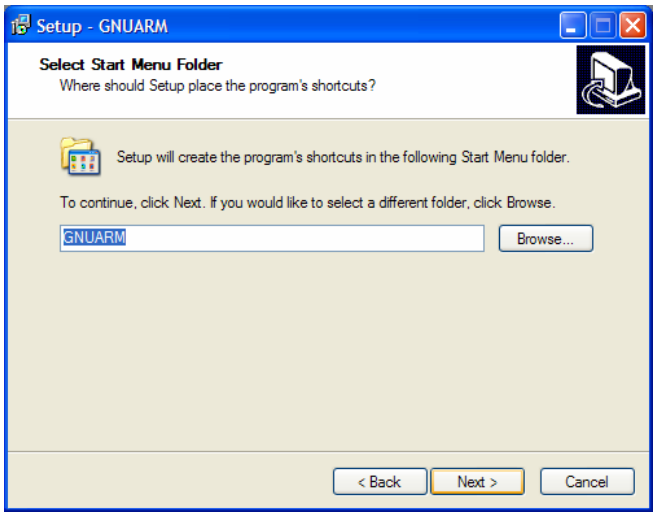


Figure 12 Install repertory and software options (leave everything ticked)



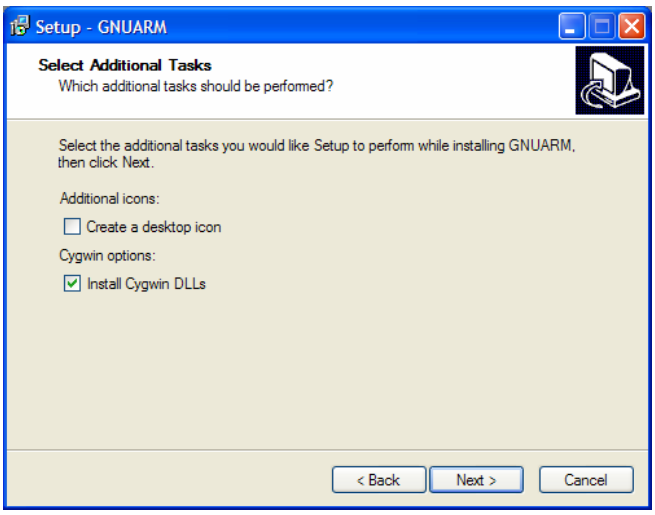
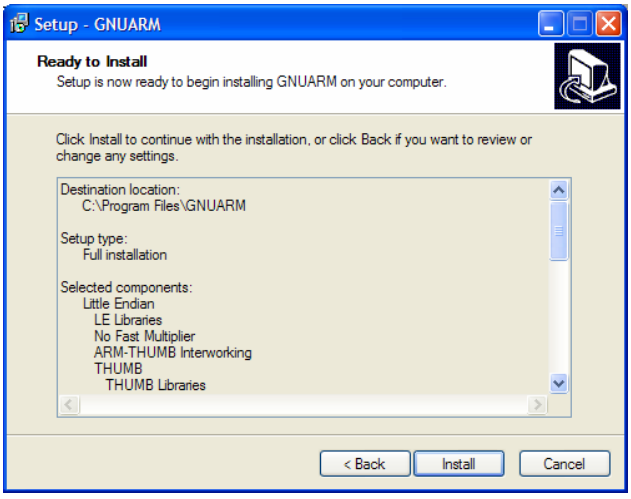


Figure 13 Start Menu and options: Deselect 'Create a desktop icon' and leave 'Install Cygwin DLLs'



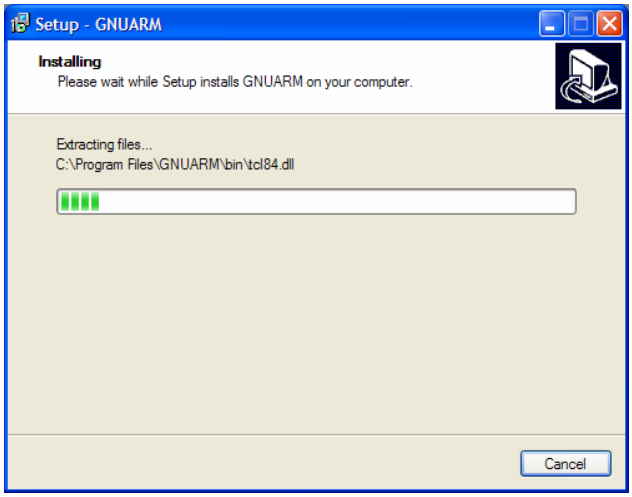
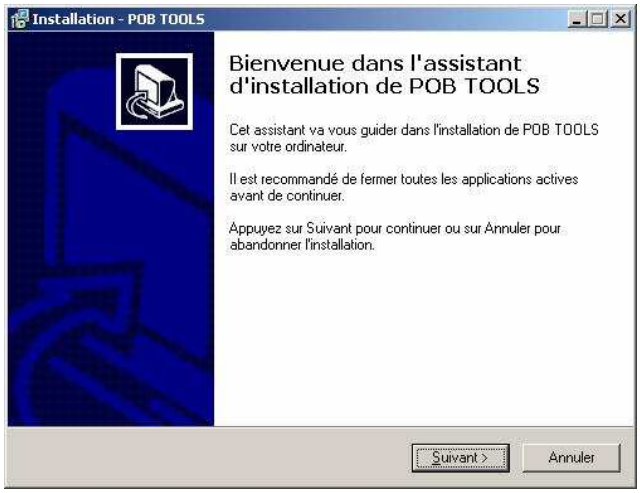
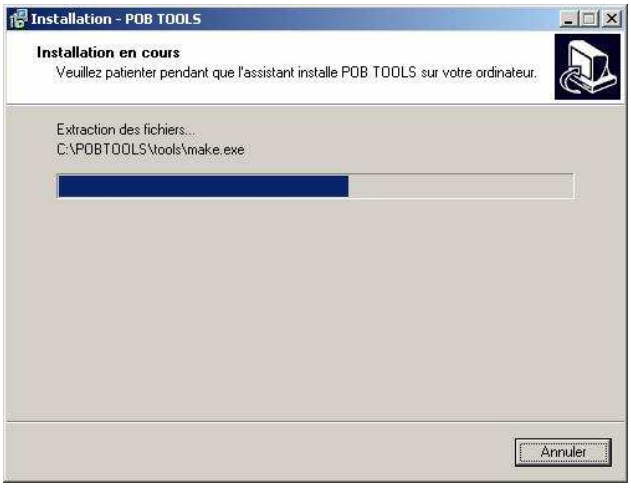


Figure 14 Summary of the components that will be installed and install begin

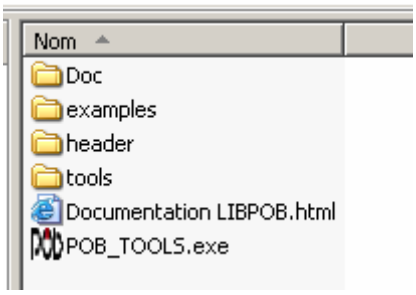
Installation of POB-TOOLS

To install the POB-TOOLS, click on the file "**INSTALL_POBTOOLS.exe**" in the CD and follow the instructions.





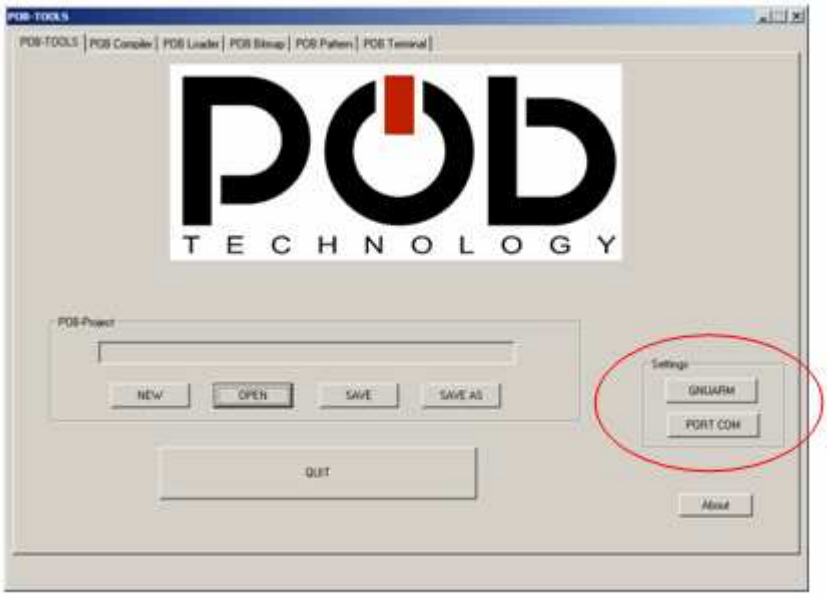
Once installed you will have the following files folders:



- *Doc* : folder with all the help files in html.
- *Examples* : Has some examples for using the POB-EYE and the POB-LCD128. *Header* : folder with all the .h files and the entire library functions *libpob.*
- *tools* : folder with all the POB-TOOLS
- *POB-TOOLS.EXE* : execute the program POB-TOOLS

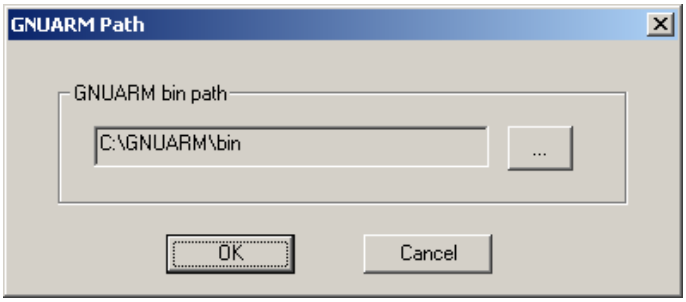
5.2 POB-TOOLS configuration

When POB-TOOLS run for the first time, you have to configure the tools correctly.



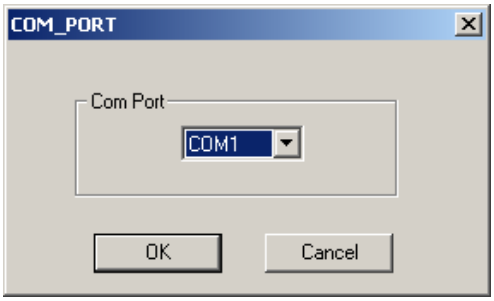
Path to use the GNUARM compiler

You must click on the 'bin' directory of GNUARM. If you have installed it using the default settings, the default path is
C:\Program Files\GNUARM\bin



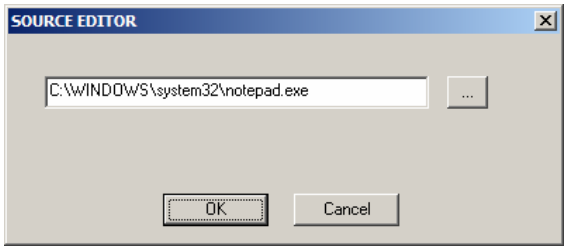
Serial port

Choose the port number on which POBEYE is connected to your computer.



Source code editor

In POB-COMPILER, you can choose the program you want to edit your source code. To declare the path, click on « Source Editor » and look for the executable file.



5.3 Manage a project application

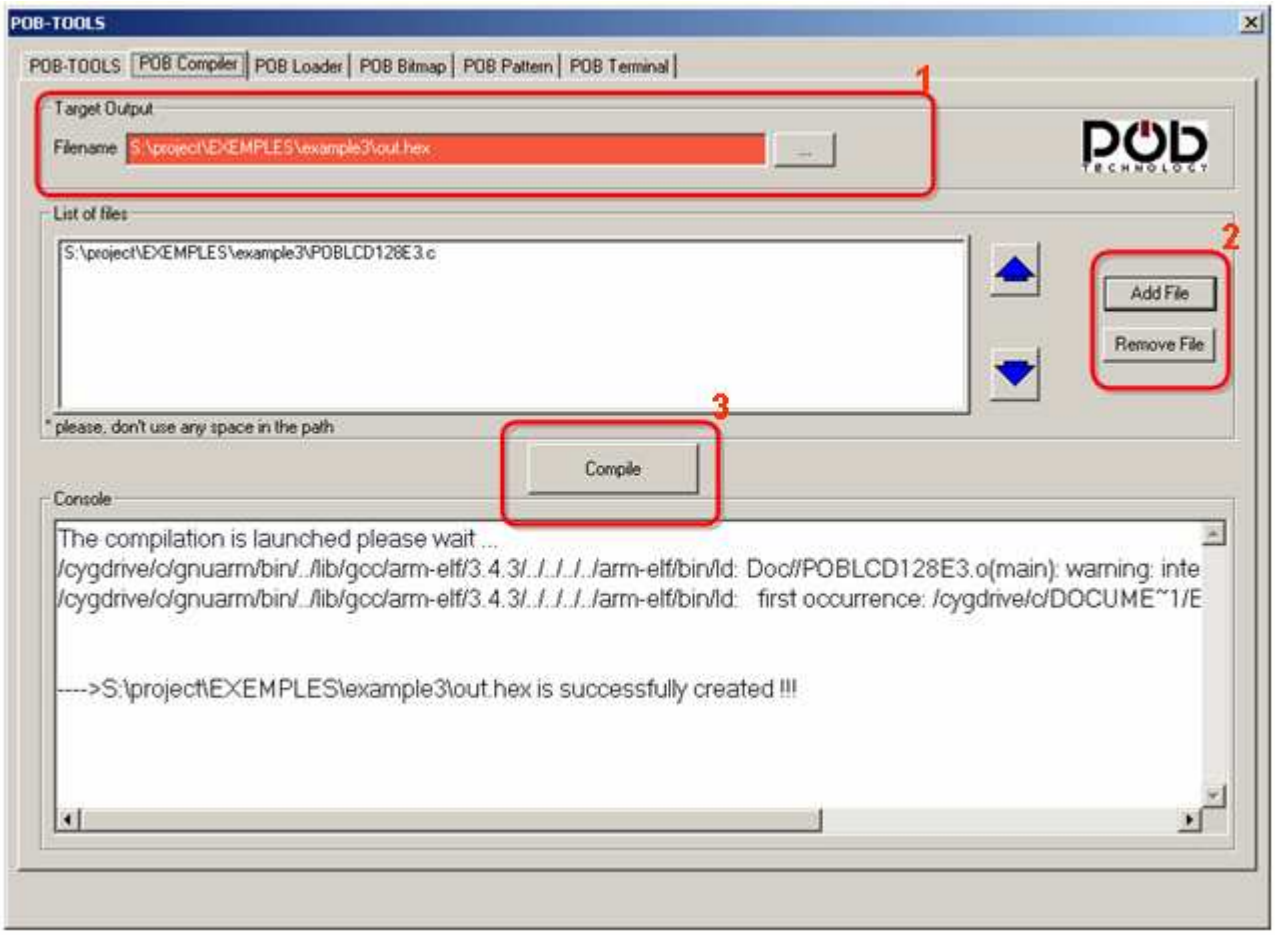
POB-TOOLS are used to manage a project application for the POB-EYE module.

- The "new" button allows the creation of a new project. You must type the name of the new project or select an existing project (extension .pob).
  Warning: if you select an existing file, the content of this file will be overwritten.

- The "open" button allows an existing project to be opened.
- The "save" button allows to save your current work.
- The "save as" button allows to save your work in a new project file.

5.4 POB-Compiler

POB-EYE module is programmed in the C language. The POB-Compiler allows you to create an application written in C for the POB-EYE module. To create an application, follow the 3 following steps:

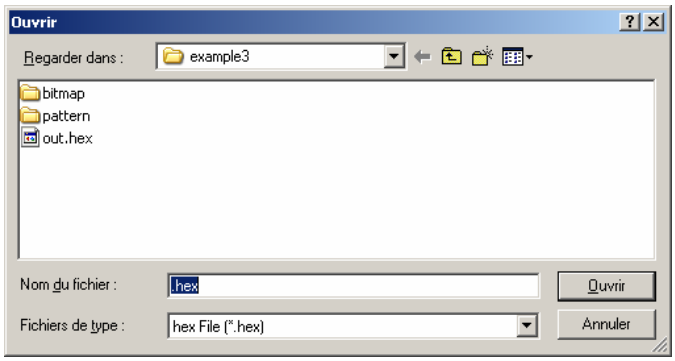


1 – Naming your application:

Click on this button : a dialog box will be displayed.

This dialog box allows setting the name of your application (extension « .hex »).

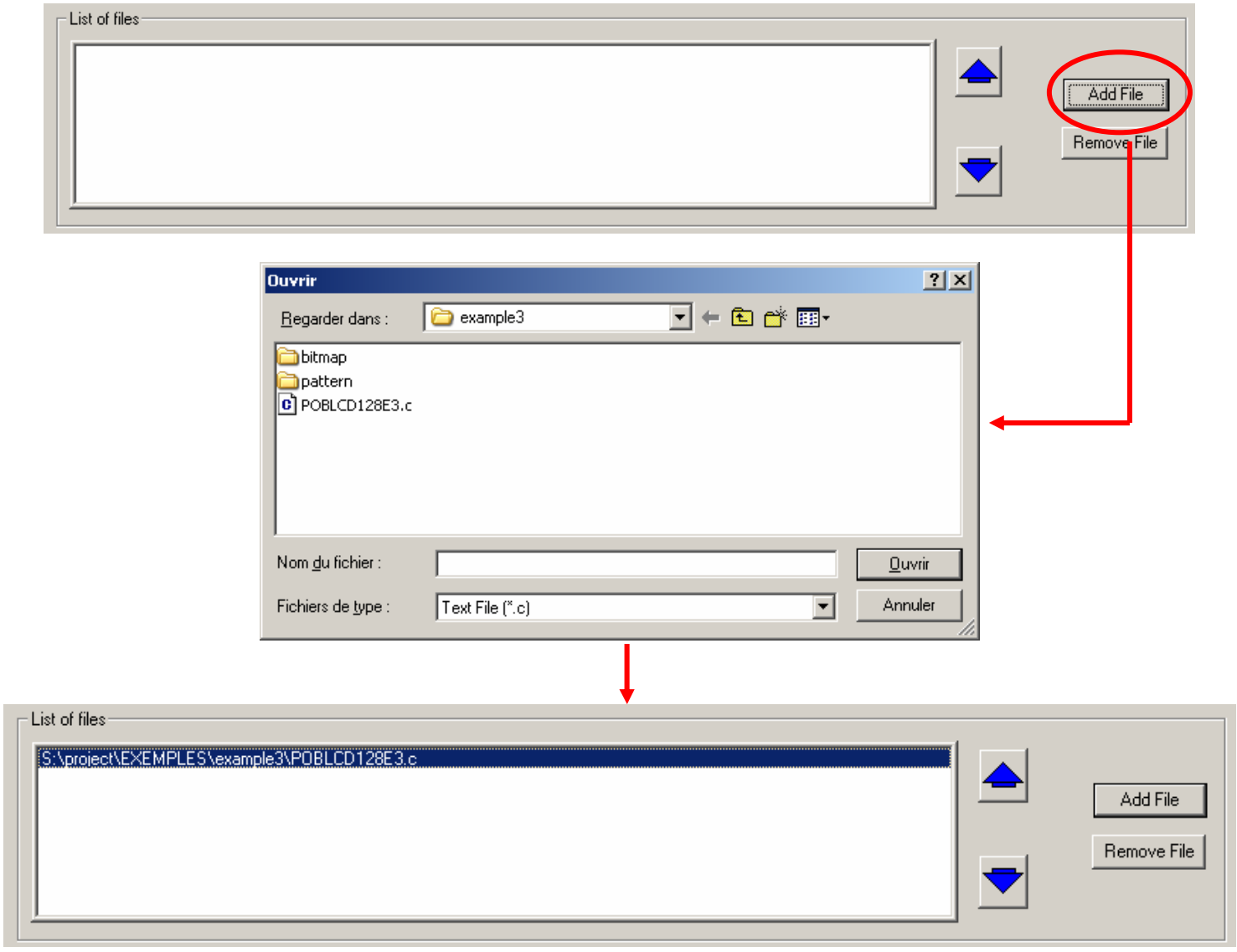
If the file already exists, you only have to select it.



2 – Adding the files to be compiled:

You have to choose the list of source files that will be compiled. To add a file, click on the « **Add File** » button. A dialog box appears and allows you to choose the file that you want to add.

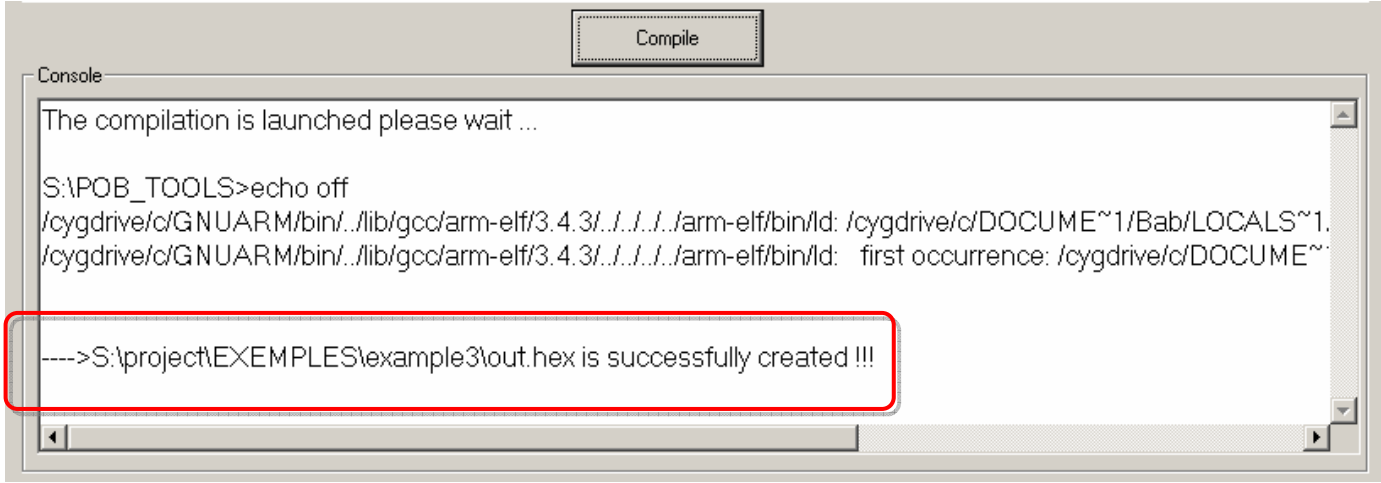
If you want to delete a file on the list, use the « **Remove File** » button. To improve the readability, you can move your files by using the blue arrows.



3 - Compilation:

Click on the « ***Compile*** » button to launch the compilation of your application.

During the compilation, you will see traces being displayed in the window called « **Console** ». These traces correspond to those of the GNUARM compiler. Thus, some "warning" or "error" messages may appear.



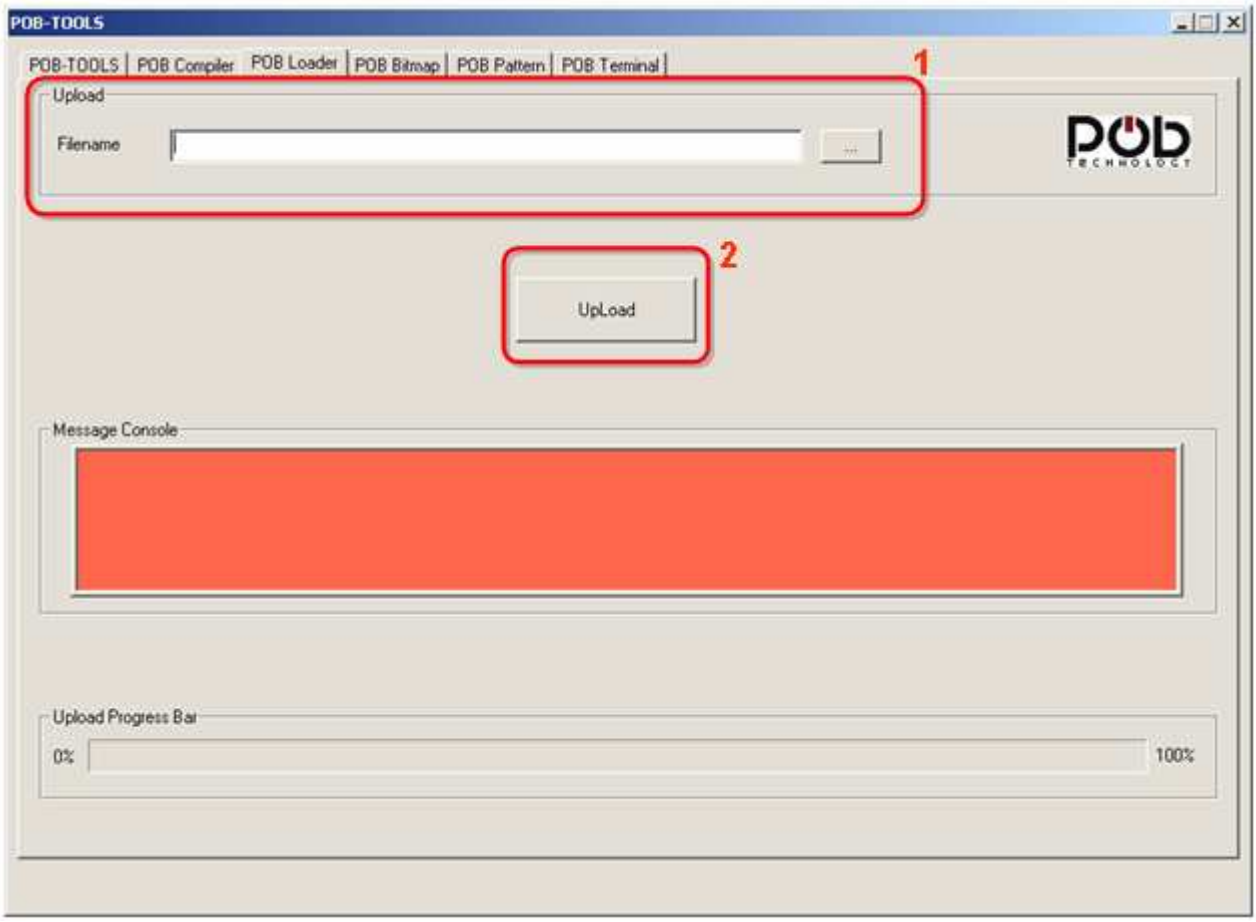
The last line that you will see appearing in the console will tell you if the application was compiled successfully. If the message « **… is successfully created!!!** » appears, your application is ready to be used in the POBEYE module.

If the compilation process failed, the message « **…is not created** » appears. You will then need to correct your application in relation to the message of the compiler.

Remark: The two lines starting with "/cygdrive" are not significant, and do not affect the success of the compilation.

5.5 POB-Loader

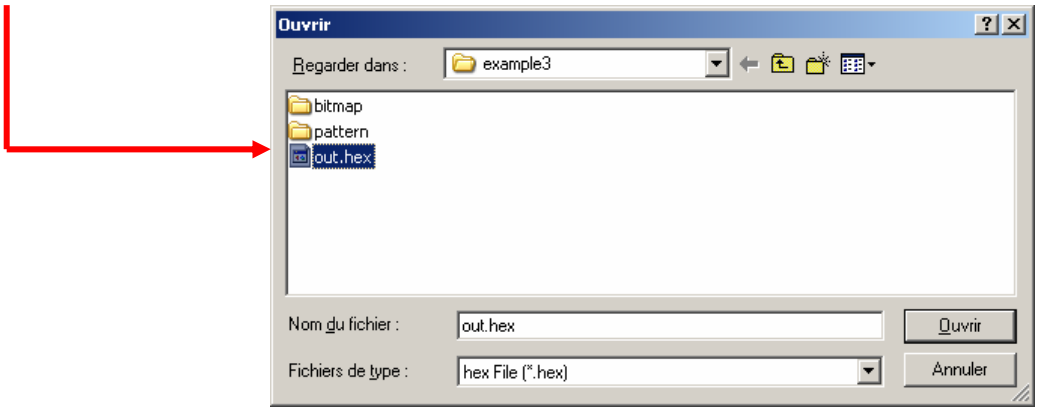
POB-Loader module allows you to load a program (create in the previous module, Pob-compiler) in the POBEYE memory.



Two steps are necessary to load a program.

1 – Selecting an application:

Click on to choose the application.



2 – Program uploads:

Remark: Before loading a program, POBEYE module must be on, in "programming" mode and link to your computer by the serial cable.

Preparing POBEYE module:

- POBEYE module must be link to your computer: see chapter 1.9
- POBEYE module must be in "programming" mode: see chapter 1.3.

To load a program, click on the "**Upload**" button. If the loading proceeds correctly, you should see the progress bar changing.

If this does not work, an error message will appear. You will have to follow the instructions to solve the problem. If the problem persists, do not hesitate to contact the POB-Technology support: support@pob-technology.com

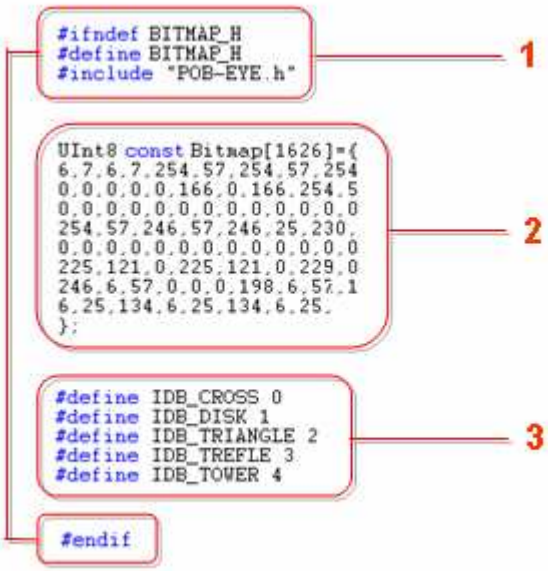
Remark: If the modules are already on, simply switch to programming mode the switch "programming/execution" and press the reset button.

5.6 POB-Bitmap

POB-Bitmap generates graphic resources for the POB-LCD128. POB-LCD allows, for example, real time visualization of what the camera sees or to act as a user interface.

POB-Bitmap generates a file to include in your code by the command "#include name_of_file.h". The file ".h" contains an array of images.

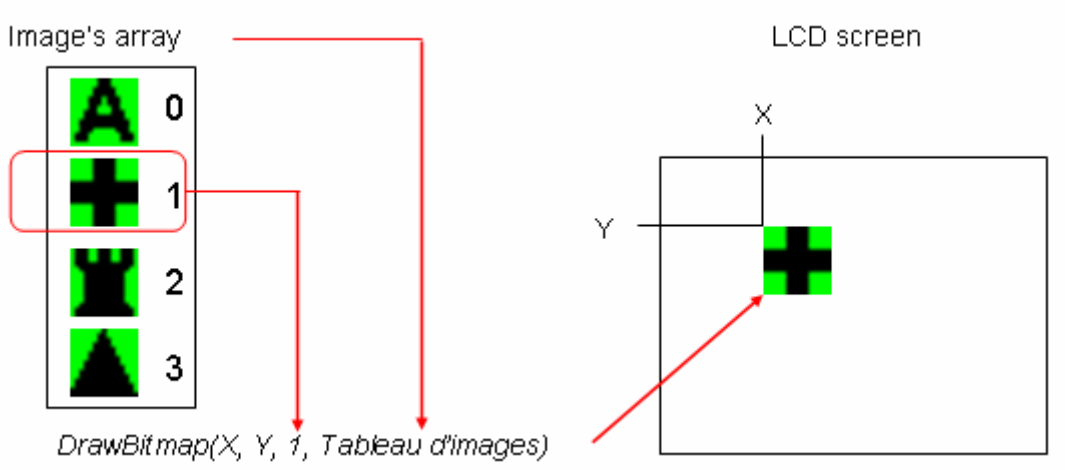
This file contains 3 areas:



- Area 1 will protect the array during compilation.
- Area 2 shows all the data containing all the bitmaps. In this example the array's name is « image » which is re-used by the function *DrawBitmap*.
- Area 3 has all the declared images. .

The graphics resources can be displayed using the library supplied with the POB-Compiler tools. The graphic functions allow you to manage the transparency of the images and to carry out the superposition of images on the LCD screen.

Images are displayed using the "*DrawBitmap*" function:

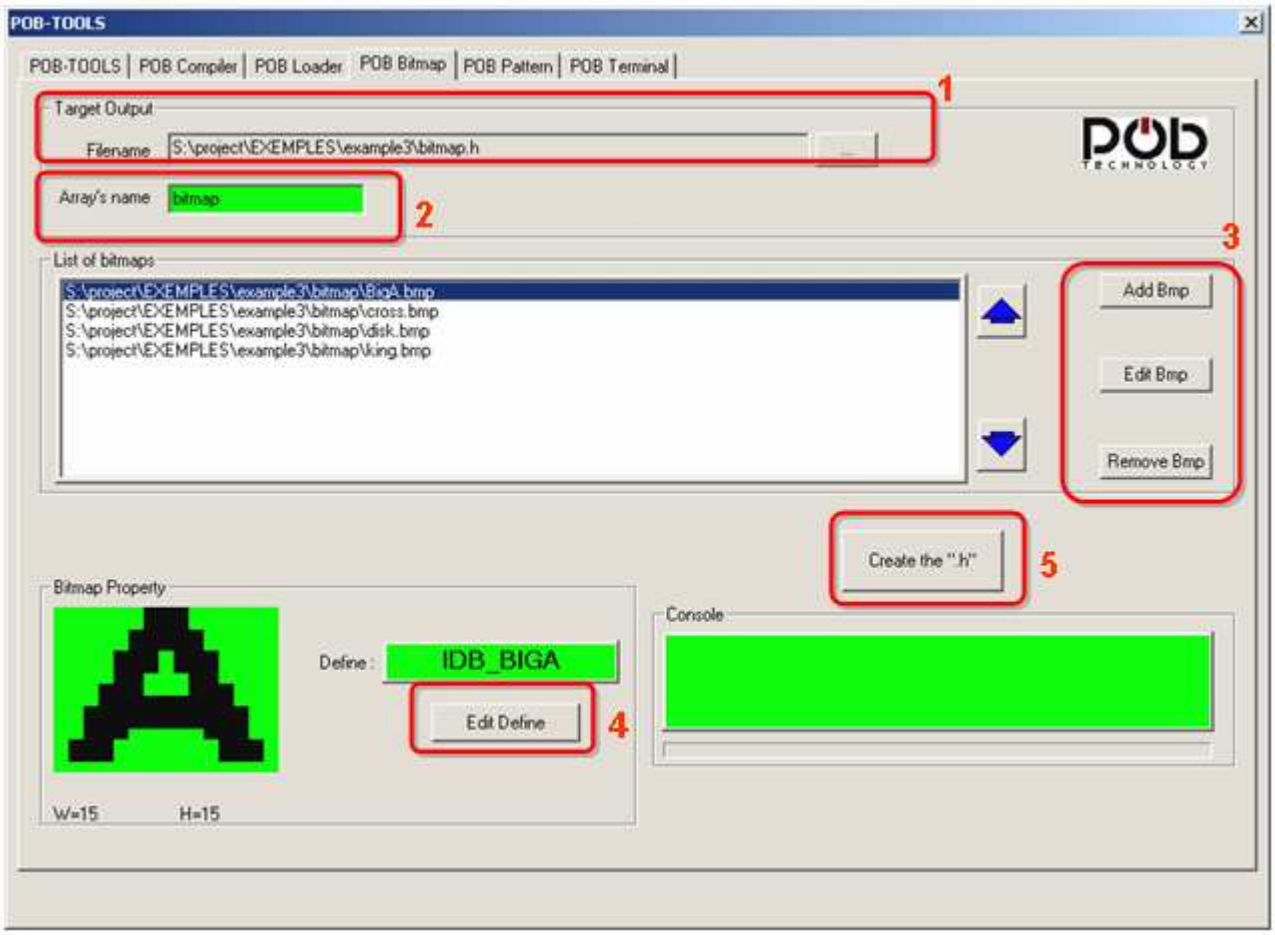


This function uses a number to display the desired image. For more clarity in your code, POB-Bitmap generates in ".h" a series of "#define" from the image's name. The call to "DrawBitmap" can be becomes (if the name of the file is *cross.bmp*):

*DrawBitmap(X, Y, **1**, Tableau);* ➔ *DrawBitmap(X, Y, **IDB_CROSS**, Tableau);*

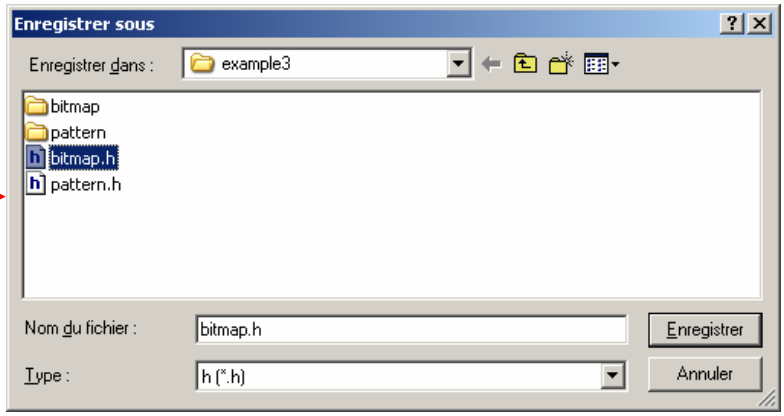
Graphic resources generation:

There are 5 steps to create the file.



1 – Naming the resource file:

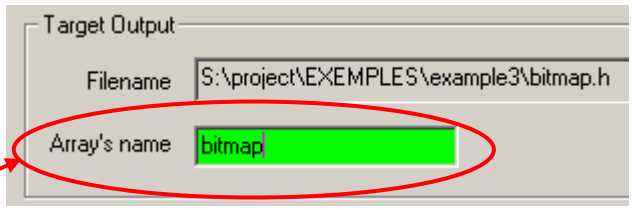
Click on to create your resources file (extension « .h ») or choose an existing file.



Warning: If you choose an existing file, the file content will be lost.

2 – Naming the resources array:

The graphic resources are in a C array.
You can rename this array or simply leave the default name.

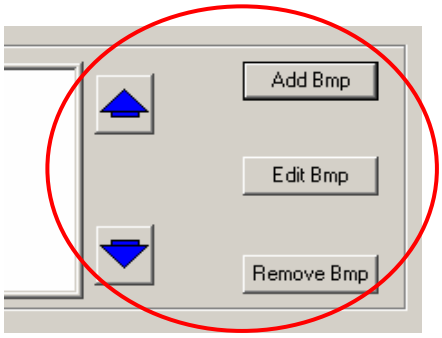



```
UInt8 const Bitmap[1626]={
6,7,6,7,254,57,254,57,254
0,0,0,0,0,166,0,166,254,5
```

3 – Manage image:

To add, edit or remove an image, press the "*Add Bmp*" or "*Edit Bmp*" or "*Remove Bmp*" button.

To move image in the list, use the blue arrows.



Remark: The images put in the library must respect the following format: Bitmap 256 colors, maximum size of 256 per 256 pixels.

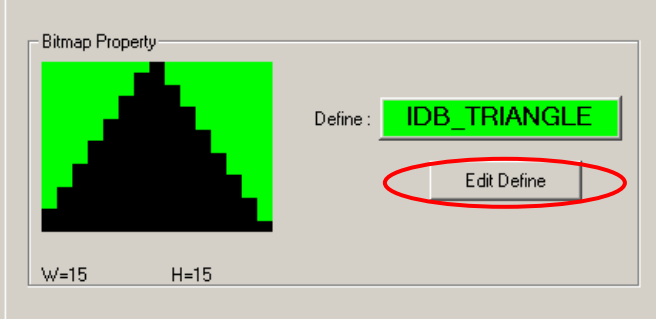
POB-Bitmap uses 3 colors to draw the graphic resources:

- Black: black pixel is drawing on the LCD.
- White: white pixel is drawing on the LCD.
- Green (Red 0, green 255, Blue 0): transparency color (allow stack images).

4 – Edit the resources name:

To locate the graphic resources in the array, POB-TOOLS use "#define".

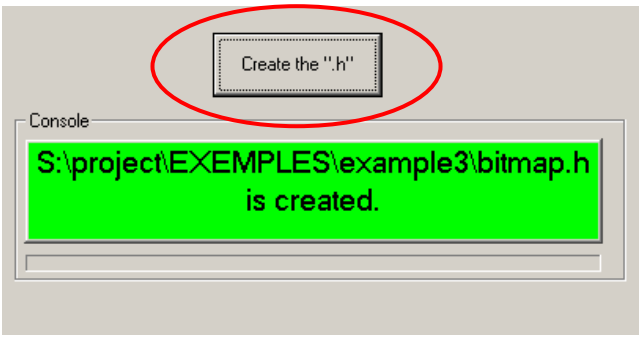
You can edit this "*#define*" with the "*Edit define*" button.



Remark: The graphic functions of the library are using "*#define"*. To find the "define"' name, simply open the resources .h file.

5 – File resources creation:

To create the file « .h », click on the "*create the '.h"*' button.

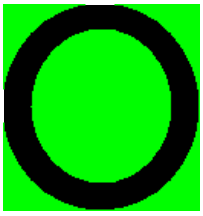


About transparency

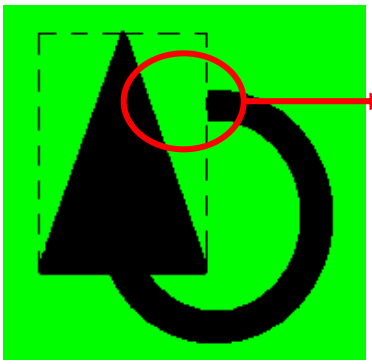
Definition of transparency color:

This color allows a pixel the possibility of not taking shape. Thus, by this means, one keeps what is already drawn.

For example, take's 2 images:

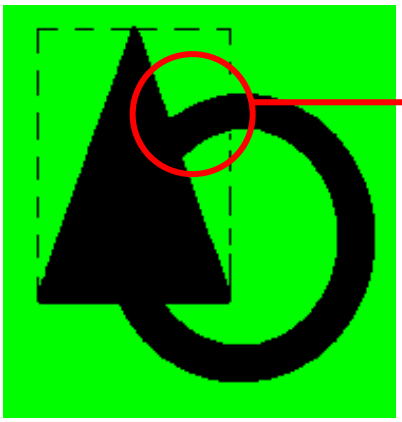


If you draw the triangle on the circle without the management of the color transparency, here is what happens:



The triangle frame erases the circle.

With the transparency management, the superposition of images is possible. Here is the result:



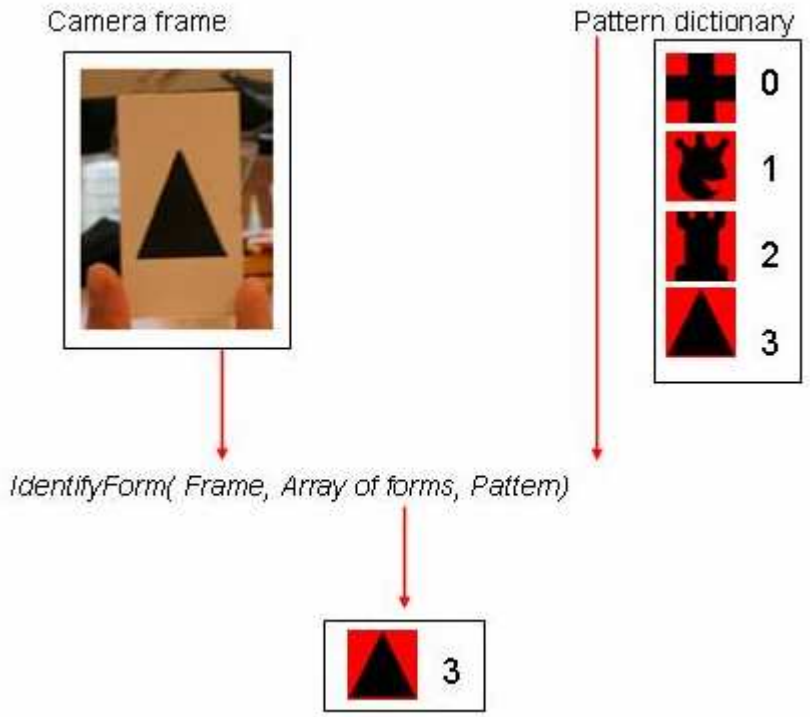
Transparency management allows the part of the circle under the triangle to be shown.

5.7 POB-Pattern

POB-Pattern allows you to create a dictionary of forms. This dictionary allows you to find forms starting from the images of the POB-EYE camera.

POB-Pattern generates a file to include in your code using the command "*#include name_file_h*". This ".h" file contains the table of the forms recognized by the POB-EYE module. The pattern recognition is the carried out using the functions of the library supplied with POB-TOOLS tools.

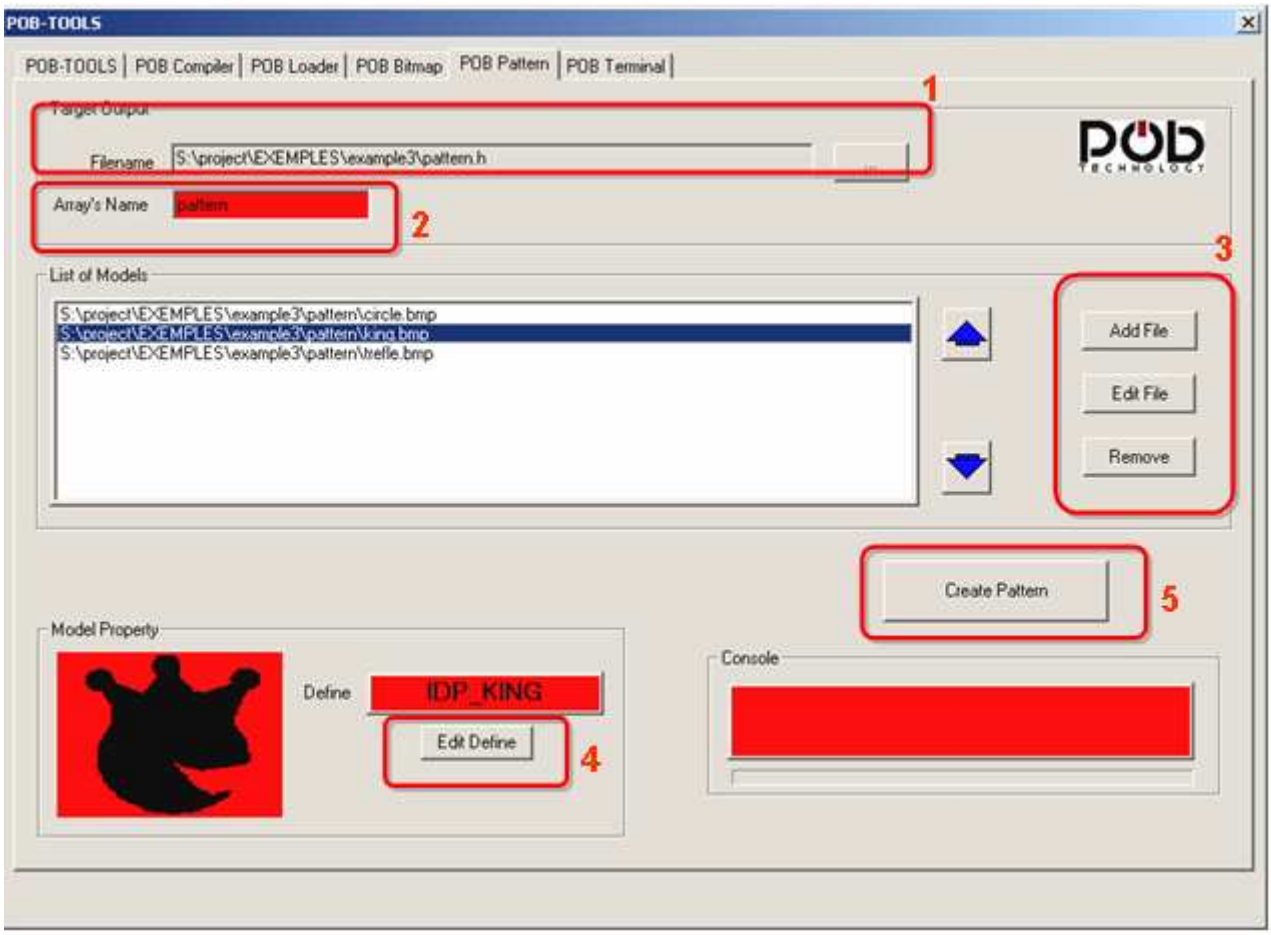
The pattern recognition is carried out using the "*IdentifyForm*" function.



This function uses the image of the camera and the table of forms to recognize an image. As output, the function fills an array with the various forms recognized in the image. For more clarity in your code, you can use the "#define" of the ".h" file instead of numbers to identify the recognized forms.

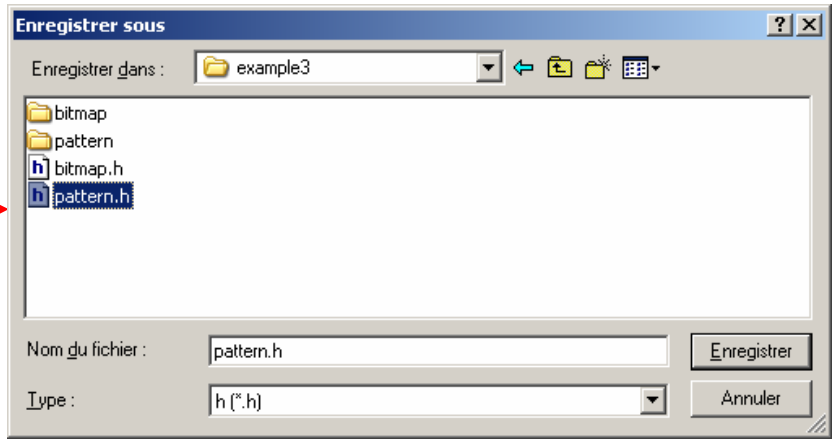
In the previous example, form 3 becomes IDP_TRIANGLE.

Follow the 3 steps to build your pattern file:



1 – Naming the dictionary:

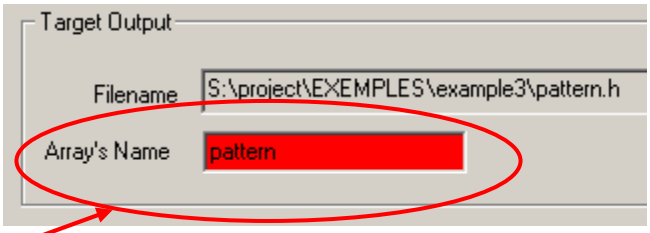
Click on to create your pattern file (extension « .h ») or choose an existing file.



Warning: if you choose an existing file, the file content will be lost.

2 – Naming the pattern array:

The forms of the dictionary are in an array. You can rename this array or leave the default array's name.

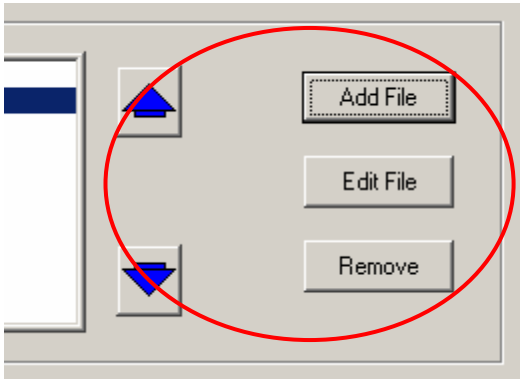



```
UInt8 const Pattern[1135]={
7,0,216,0,102,203,244,202,2,2
,0,166,0,169,0,172,0,172,137,
,130,172,133,172,136,172,0,17
```

3 – Manage images:

For add, edit or remove an image, press the "*Add Bmp*" or "*Edit Bmp*" or "*Remove Bmp*" button.

To move image in the list, use the blue arrows.



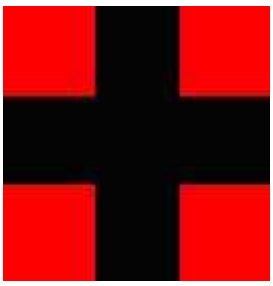
Remark: Image must respect the following format: Bitmap 256 colors, **size of 100 per 100 pixels**.

The forms must be drawn in black with a red background

To obtain the best possible result during the creation of the dictionary, certain conditions should be observed:

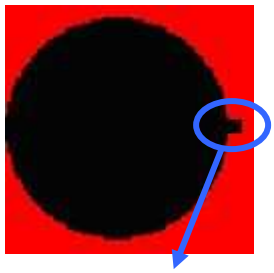
- The drawn image must take the maximum of space within the framework of 100 per 100 pixels.

Drawing not wide enough

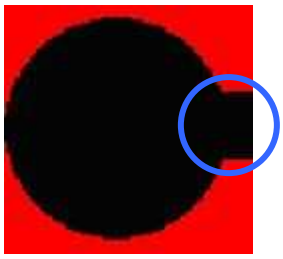


The space totally used

- It is necessary to avoid the small details on the image.



Too small detail



Large enough detail

- The form should not have any empty space.

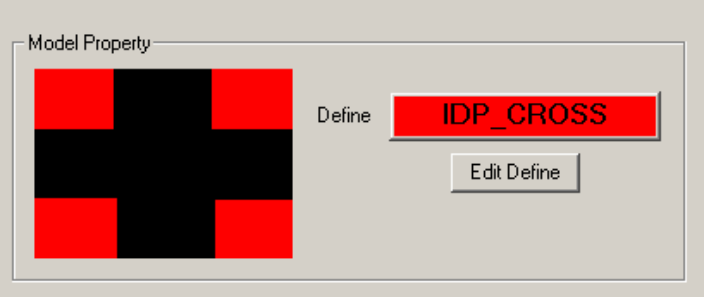
The form is empty!

Part without empty spaces

4 – Naming the forms:

To locate the forms in the dictionary, the library supplied with POB-TOOLS uses "#define".

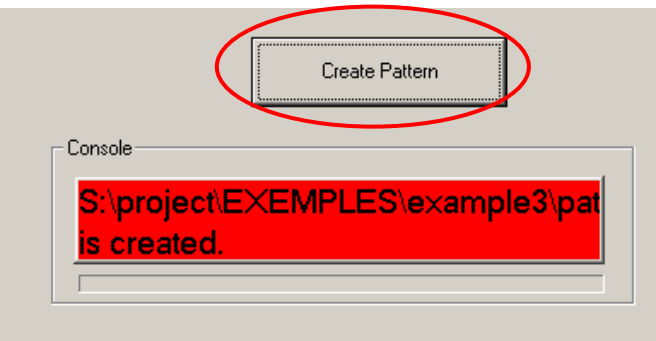
You can edit this "*#define*" with the "*Edit define*" button.



Remark: "#define" is used by the recognition function in your application to identify the recognized form. To find the name of the "define", it is sufficient to open the created ".h" file.

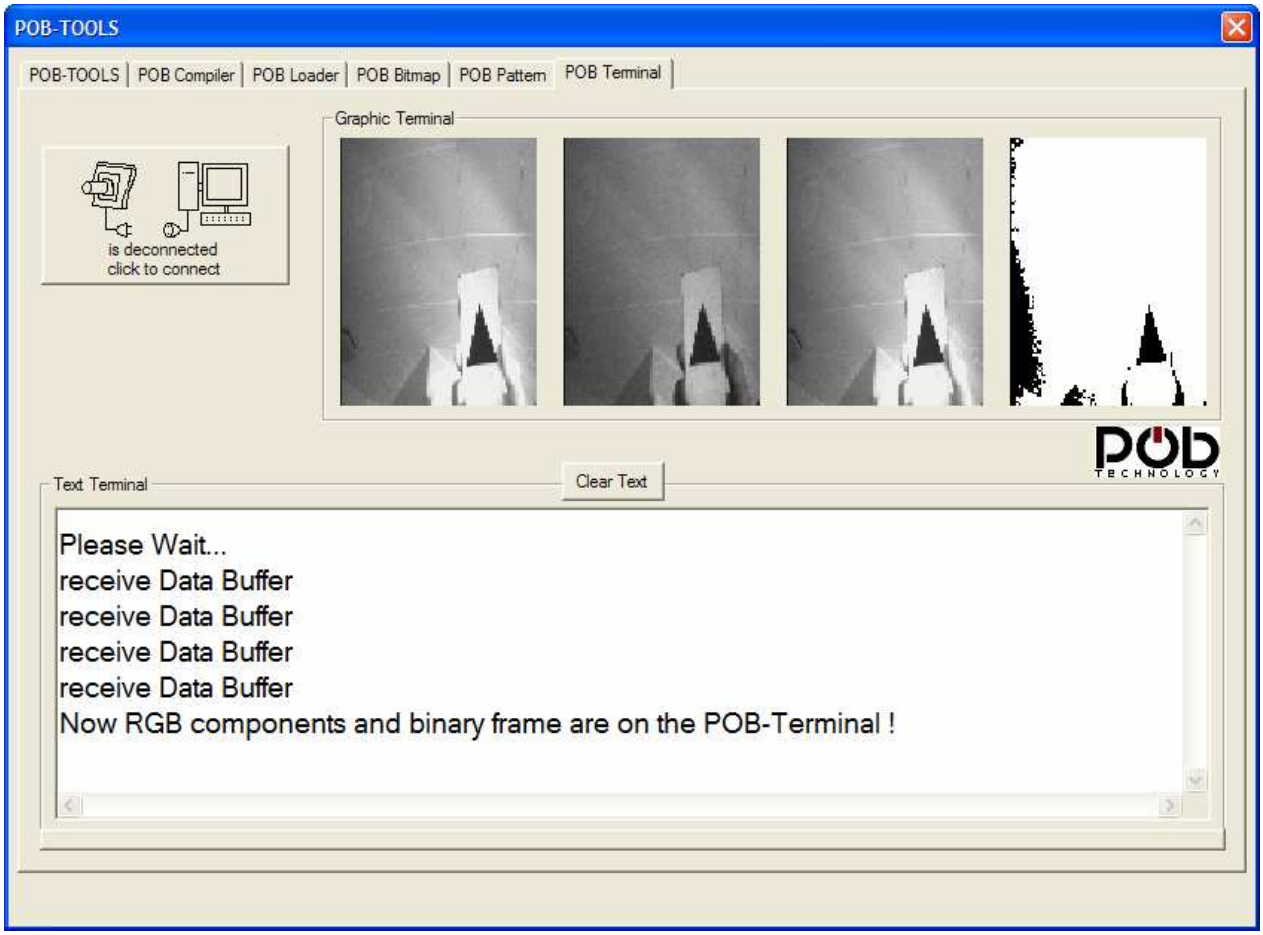
5 – Dictionary creation:

To create the file, click on the "*Create Pattern*" button.



5.8 POB-Terminal

POB-Terminal interface allows you to play the message of the POB-EYE module via the serial port and to display the images captured by the POB-EYE camera. The aim of the POB-Terminal is to facilitate the development of your application.



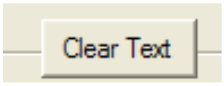
PB-Terminal can display messages from POB-EYE through the serial port as well as images grabbed by the camera. POB-Terminal was design to help you create your applications.

1 – Connecting POB-Terminal:

The "on" button allows you to connect POB-Terminal to the POB-EYE module.

When POB-Terminal is connected, you can disconnect it with the "off" button.

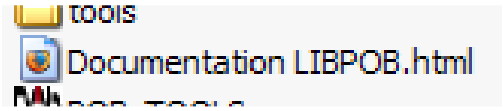
2 – Clearing messages:

You can clear the POB-Terminal messages with the "clear text" button. 

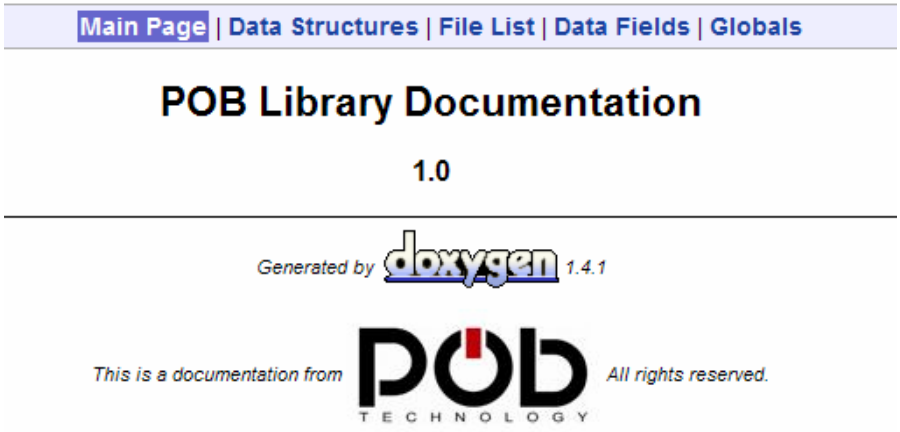
For any further details on « how to display information » on POB-Terminal please refer to the first example given in section 7

6 Library help files

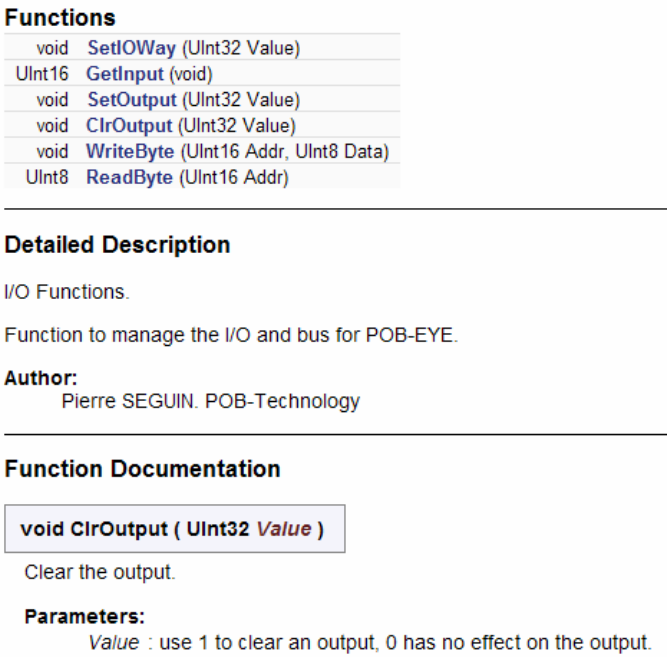
To get the help on how to use the POB-EYE's libraries refer to the file « documentation LIBPOB.html ».



Your web explorer will then display the help page.



From this page you have access to all the function's definitions used in LibPOB library.



Sample Applications

Recognize a form and send it to the POB-Terminal
Print pictures ont the POB-LCD
See in real time what is coming from the POB-Eye

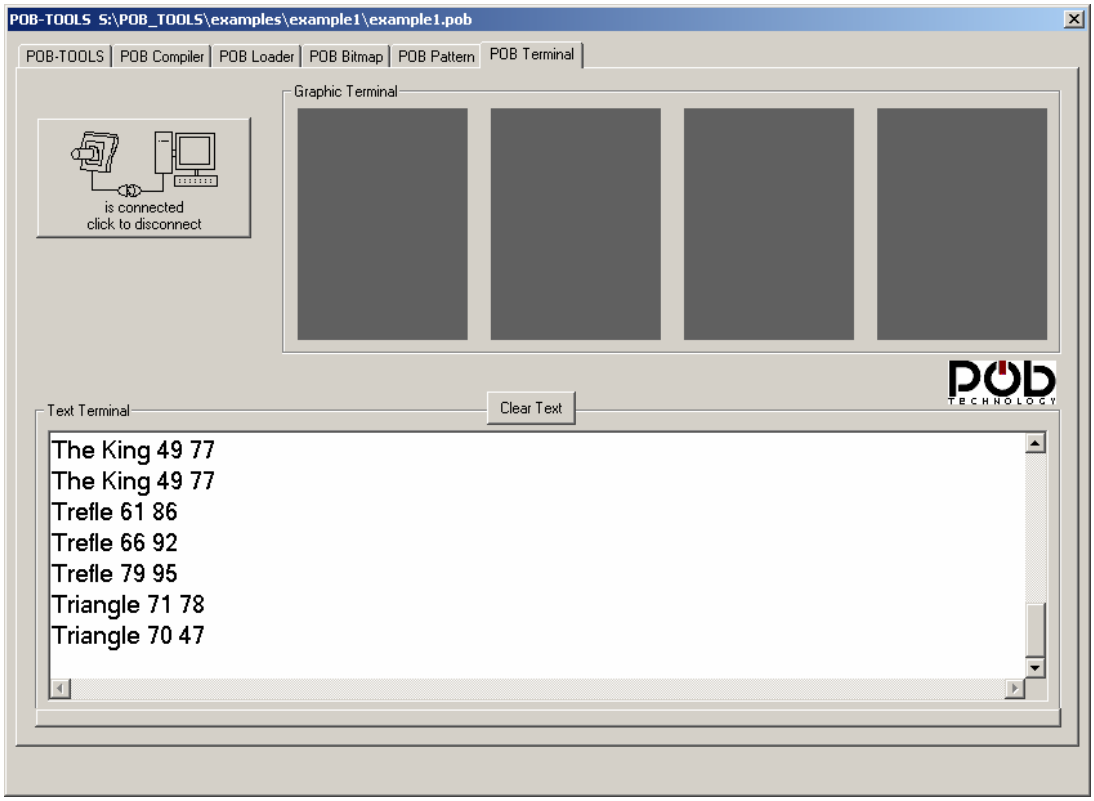
Visit our blog DEVZone for more examples et documents

www.pob-technology.com/blog

7 Sample application

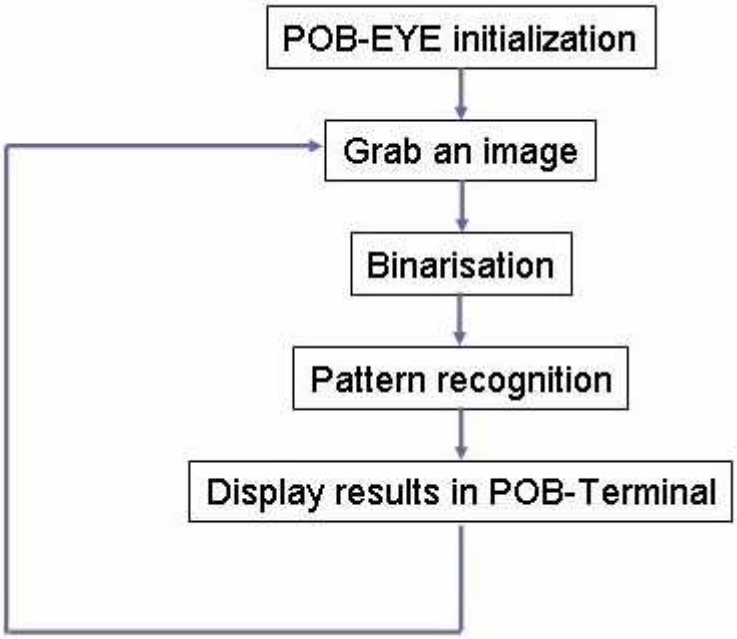
Pattern recognition and POB-Terminal display

This example is given in folder « *example1* ». To open the project in POB-TOOLS, select the file « *example1.pob* ».

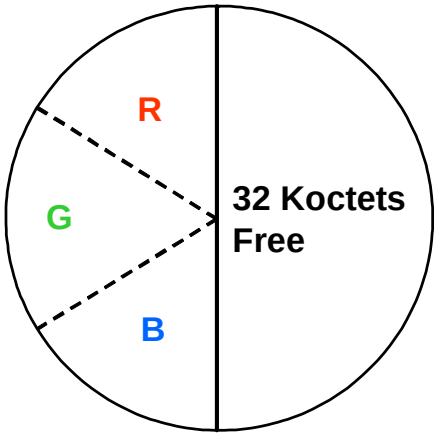


The idea of this program is to identify the patterns showed to the POB-EYE and to display the name and coordinates in POB-Terminal. This program was implemented in C language and uses functions of POB-TOOLS and POB-EYE.

Program algorithm:



Remark: A 32Kbyte was dedicate for the RGB components (88 pixels by 120 pixels = 10560 byte per component). All the operations on video will be done in the memory space.



Program source code:

One file is needed to use all the POB-EYE's (camera, UART, debug on POB-Terminal)

```
#include <pob-eye.h> /* add file for using the POB-EYE */
```

The second file is the pattern. This file is created with POB-TOOLS in the POB-Pattern tab.

```
#include "pattern.h"   /*dictionnary of patterns*/
```

Program's start :

```
int main (void)
{
    UInt8 i=0;

    UInt8 Nb_Identify=0; /* Variable corresponding to the number of patterns */
    Form ListOfForm[MAX_OF_FORM]; /* list of forms*/
    RGBFrame FrameFromCam; /* struct of three pointers on the RGB components */
```

The POB-EYE is initialized using the function « ***InitPOEYE*** » as well as initializing the camera, l'UART and the bus I2C. You have to call this function if you want your module to work properly.

```
    InitPOB-EYE();system initialization*/
```

There is a particular way of working with the camera. This function « *RGBFrame* » contains all the addresses of the RGB components and has to be initialized by the following function « ***GetPointerOnRGBFrame*** ». The address in memory for the RGB components will always stay the same, that's why this function is only called once.

```
    GetPointerOnRGBFrame(&FrameFromCam); /*Get the pointer of the red,green and blue video buffer */


    while(1) /* Main loop */
    {
```

First of all, the function « ***GrabFrameRGB*** » will grab an image from the camera and then will save the image in the dedicated place pointed by « *FrameFromCam* ».

```
GrabFrameRGB(); /* grab the RGB components */
```

For pattern recognition, the image has to be binarized. In the end the 3 RGB components will be the same, a 0 for the white and a 1 for the black

```
BinaryRGBFrame(&FrameFromCam); /* binary the three RGB Buffer */
```

The function « ***IdentifyForm*** » can identify patterns within an image. The parameters needed are : a pointer on the binarized image, an empty array (« *ListOfForm* ») and the pattern dictionary. This function will fill up the array with the patterns and then send the number of the recognized patterns. Please note that the dictionary patterns « *Pattern* » was created using POB-PATTERN.

```
Nb_Identify=IdentifyForm(&FrameFromCam,ListOfForm,Pattern); /* try to identify the forms and make a list of it*/
```

Finally, the information saved in the array « *ListOfForm* » are extracted and then prompt on POB-Terminal. The function « ***PrintTextOnPobTerminal*** » can print text in POB-Terminal.
This function is similar to « *printf* » used in C library.

```
/* parse the list of the form and print result on the Pob-Terminal*/
for( i=0 ; i < Nb_Identify ; i++ )
{
    /* for each patterns we grab its id */
    switch (ListOfForm[i].id)
    {
    /* Display results in POB-Terminal corresponding to the id */
    case IDP_0_CROSS:
    PrintTextOnPobTerminal("Cross %d %d",ListOfForm[i].x,ListOfForm[i].y);
    break;

    case IDP_1_BIGA:
    PrintTextOnPobTerminal("A Big A %d %d",ListOfForm[i].x,ListOfForm[i].y);
    break;

    case IDP_2_KING:
    PrintTextOnPobTerminal("The King %d %d",ListOfForm[i].x,ListOfForm[i].y);
    break;
    case IDP_3_TOWER:
    PrintTextOnPobTerminal("Tower %d %d",ListOfForm[i].x,ListOfForm[i].y);
    break;

    case IDP_4_TREFLE:
    PrintTextOnPobTerminal("Trefle %d %d",ListOfForm[i].x,ListOfForm[i].y);
    break;
```

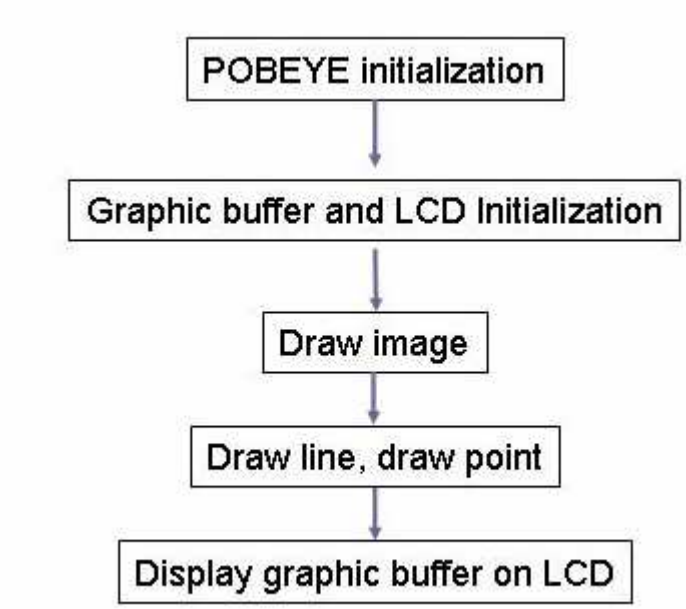
```
                    case IDP_5_TRIANGLE:
                    PrintTextOnPobTerminal("Triangle %d %d",ListOfForm[i].x,ListOfForm[i].y);
                    break;

                    case IDP_6_CIRCLE:
                    PrintTextOnPobTerminal("Circle %d %d",ListOfForm[i].x,ListOfForm[i].y);
                    break;

                    default:
                    break;
                    }
                }
        }
        return 0;
}
```

Display images on POB-LCD128

This example can be found in the folder« *example2* ». To open project in POB-TOOLS choose to open the following file « *example2.pob* ».

This program will display images on the LCD and uses all the graphical functions such a "*draw line, draw point...*"

Program algorithm:



Program source code :

To use POB-EYE's functions and the LCD module include the following functions:

```
#include <pob-eye.h>
```

Then include the image file.

```
#include "Bitmap.h" /*  include bitmap list*/
```

```
int main (void)
{
```

In order to speed up the display on the LCD, a graphic buffer was used. All the operations are done on the buffer and only then the buffer is displayed on the screen.

TO do so you have to declare an array « *LCD_Buffer* » in order to stock the pixels. In this application an array of 128 by 64 and 1 bit per pixel was used. **Caution** it is require indicating the unit : in this case we are using bit. *BITS* are used when working with 1 bit per pixel and *BYTES* when using 1 byte per pixel.

```
UInt8 LCD_Buffer [LCD_WIDTH*LCD_HEIGHT*BITS]; /* where the pixels will be stored*/
```

In addition the buffer, another function is used to store the information on the screen's size.

```
GraphicBuffer ScreenBuffer ; /* buffer screen*/
```

Make sure POB-EYE and POB-LCD are declared for a correct use.

```
InitPOB-EYE(); /* POB-EYE initialization */
InitLCD();  /* LCD initialization*/
```

Finally the graphic buffer is declared using « ***InitGraphicBuffer*** » with the following data : Screen size, Number of bit or byte per pixel and location of the storage array (« *LCD_Buffer* »).

```
InitGraphicBuffer( &ScreenBuffer, LCD_WIDTH,LCD_HEIGHT,ONE_BIT,LCD_Buffer); /* init the Graphic buffer with 128 per 64, one pixel per bit and LCD_Buffer*/
```

For the number of bit per pixel you would like to use you need the following definitions : *ONE_BIT* (for 1 bit per pixel) or *EIGHT_BITS* (for 1 byte per pixel).

Remark : For the graphic buffer we are using the command *BITS* or *BYTES*. And concerning the function InitGraphicBuffer, we are using *ONE_BIT* or *EIGHT_BITS.*

```
ClearGraphicBuffer(&ScreenBuffer); /* clear the graphic buffer */
```

All the image drawing is done by« ***DrawBitmap*** ». The parameters needed are : The X,Y coordinates the number of the image and a pointer towards the graphic buffer.

Please note that the number and the array « *Bitmap* » are created by POB-Tools.

```
DrawBitmap(30,10,IDB_1,Bitmap,&ScreenBuffer); /* image 1 is drawn*/
Draw Bitmap(45,10,IDB_2,Bitmap,&ScreenBuffer); /* image 2 is drawn*/
DrawBitmap(65,10,IDB_3,Bitmap,&ScreenBuffer); /* image 3 is drawn*/
```

To draw a line, use the command « ***DrawLine*** » with the following parameters : Start coordinates Final coordinates and pointer on graphic buffer.

```
DrawLine(10,10,25,30,&ScreenBuffer);
```

Drawing a point is done by « ***PlotAPoint*** ».

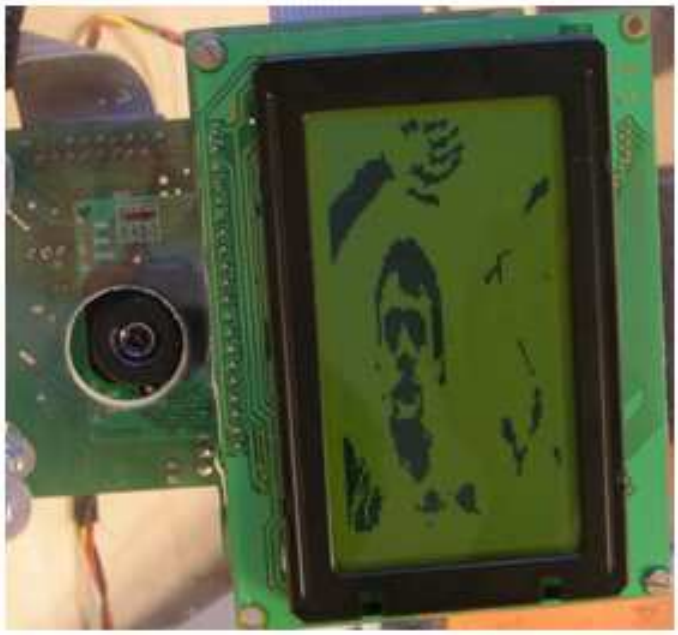
```
PlotAPoint(20,10,&ScreenBuffer);
```

Finally, we can display the graphic buffer on the LCD screen using the function « ***DrawLCD*** » with : Pointer on graphic buffer as a parameter.

```
DrawLCD(&ScreenBuffer); /* Display on screen */

return 0;
}
```

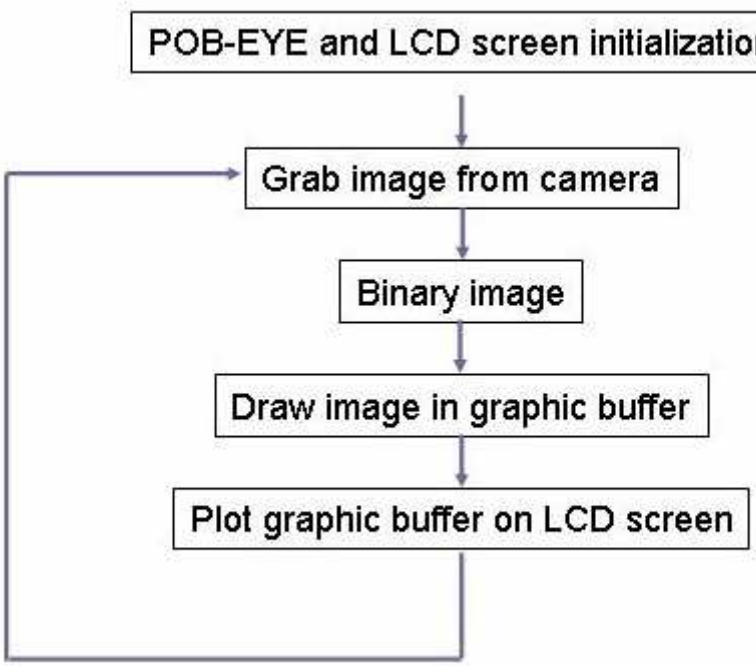
Real-Time display on POB-LCD

This program is using the POB-EYE module and the LCD screen.



This example can be found in folder « *example3* ». To open project select « *example3.pob* » in POB-TOOLS.

Program algorithm:



Program source code:

To use POB-EYE's and POB-LCD's function the following folder has to be included:

```
#include <pob-eye.h>
```

Program beginning:

```
int main (void)
{
        Int16 i=0,j=0,k=0;
```

To display an image on the LCD screen you need two variables: an array to store the pixels and a structure in order to display the graph buffer.

In this application, to avoid lag problems we have to work with 1 Byte per pixel (see chapter on POB-LCD128).

```
        UInt8 LCD_Buffer [LCD_WIDTH*LCD_HEIGHT*BYTES]; where the pixels will be stored*/

        GraphicBuffer LCD_Screen; /* buffer screen*/

        RGBFrame   FrameFromCam; frame of camera*/
```

```
InitPOB-EYE(); /* init POB-EYE module*/
InitLCD(); /* LCD screen ini*/
```

The graph buffer has to be initialized with the screen dimensions and the storage array.

```
/* init the Graphic buffer with 128 per 64, one pixel per bit and LCD_Buffer */
InitGraphicBuffer(&LCD_Screen,LCD_WIDTH,LCD_HEIGHT,EIGHT_BITS,LCD_Buffer);

ClearGraphicBuffer(&LCD_Screen); /* clear the graphic buffer*/
```

Then the address of the RGB components must be initialized.

```
GetPointerOnRGBFrame(&FrameFromCam); /* get the pointer of the red,green and blue video buffer*/
```

Main program loop:

```
while (1)
{
```

Grab an image from the camera:

```
GrabFrameRGB(); /* grab the RGB components*/
```

The LCD screen is in black and white that's why the image from the camera must be binarized.

```
BinaryRGBFrame(&FrameFromCam); /* Binarise RGB components*/
```

To display a binary image on the POB-LCD, fill the array with pixel values from one camera component.

Note that this image is a binary image. All the RGB components are the same therefore you can use either one of them to fill up the array.

```
        for (k=0,i=64;i;i--)
        {
                for (j=0;j<120;j++,k++)
                {
                        LCD_Screen.buffer[k] = FrameFromCam.red[i+(j*88)];

                }
                k+=8;
        }
```

Then display the results on LCD screen

```
        DrawLCD(&LCD_Screen);
    }

    return 0;
}
```

POB-Technology contacts

POB-TECHNOLOGY
4, rue nicéphore niépce
69 680 CHASSIEU
FRANCE

Web: www.pob-technology.com

Mail: contact@pob-technology.com

Phone: +33 (0)4 72 43 02 36
Fax: +33 (0)4 72 66 18 99

Pob-Technology Document – All Right Reserved