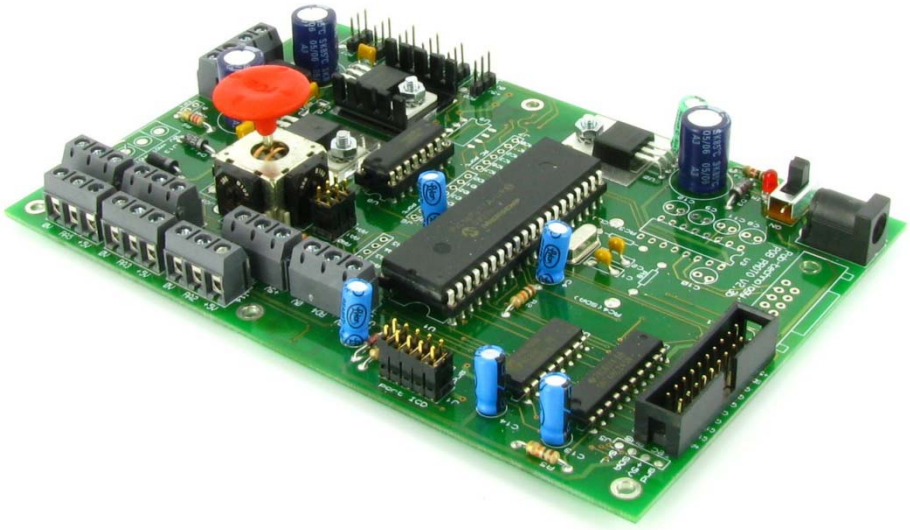
POB TECHNOLOGY

CODE EXAMPLES
POB-EYE & POB-PROTO



August 2009 | POB Technology

SUMMARY

Informations **3**

1. Presentation **4**

1.1 POB Library 4

1.2 Required tools & equipment 4

2. POB-Eye examples **5**

2.1 Hello World 5

2.2 Communicate with POB-Eye 8

2.3 Display image from camera on LCD screen 9

2.4 Forms recognition 10

2.5 Forms recognition on LCD screen 14

3. POB-Proto examples **17**

3.1 POB-Proto initialization 17

3.2 Read the value of the Joystick 18

3.3 Move the bot with the Joystick 21

LEGALS INFORMATIONS

Company	**POB-Technology**
Type	SARL
SIRET number	483 594 386 000 28
APE	742C

CONTACTS

Adress	**POB-Technology** **11, avenue Albert Einstein** **69 100 VILLEURBANNE** **France**
Email	contact@pob-technology.com
Phone number	+33 (0)4 72 43 02 36
Fax	+33 (0)4 83 07 50 89

DOCUMENT MANAGEMENT

File name	**Ezamples_for_POB_bot_en.pdf**
Creation date	03/08/2009
Author	Alisson Foucault
Last update	03/08/2009

1. GENERALITE

1.1 POB LIBRARY

POB programs are developed from a library specific to our products, facilitating their implementation. Most of the functions needed for programming your robot include this library. You will find all necessary information in the API documentation of your Pob-tools directory. It is important to familiarize yourself with the library POB to optimize your use of POB-bot, which is why we strongly encourage you to do it through these examples.

1.2 REQUIRED TOOLS & EQUIPMENT

To achieve the examples in this document, you will need the equipment below :

- A POB-Bot and its serial cable
- POB-Tools
- Paint or another image processing software
- Tera-term
- Printable forms for recognition by your robot

Picture 1.2.1 - POB-Eye

Picture 1.2.2 - POB-Proto

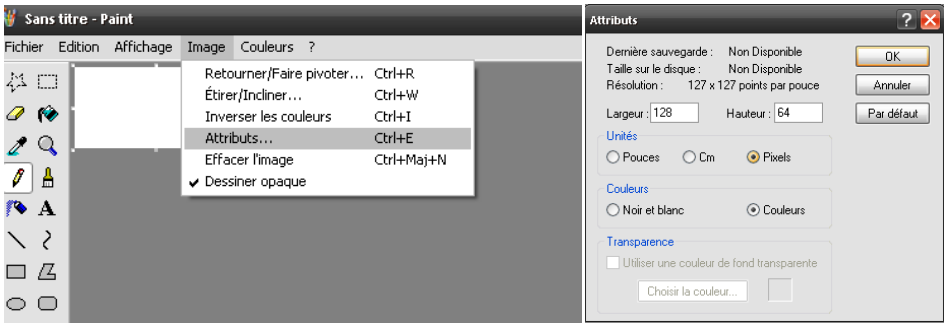
2. POB-EYE EXAMPLES

2. 1 HELLO WORLD

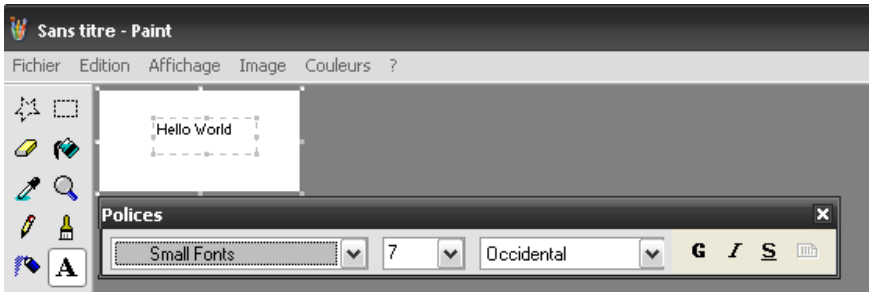
The first of all, the Hello World allows here to display an image on the LCD screen of the robot POB.

A. CREATE PICTURE

The POB LCD screen is able to displaying 256 colors bitmap with a resolution of 128x64 pixels. In order to proceed easy as possible to create this file, it is recommended to use Paint. To resize your file once the software started, click Image, then Attributes. Represent the width and height of your pixels (respectively 128 and 64), and the click OK.



Now use the text editor and select your working area to put the comentary of your choice, here « Hello world". We strongly recommend you to use the police « Small Font », in size 7, to optimize the display on the LCD screen of the robot.



Once the file finished, save it to bitmap format (.bmp), 256 color (instead 24bits by default).

⚠ Warning, do not leave spaces in the name of your bitmap !

Thereafter you will need to create a file Bitmap.h included in your code. We will see how in Part C about compilation.

B. CODE

These are some very basic guidelines. As a fisrt step, you must include the library <pob-eye.h> and the Bitmap,h file you will create later. Now you have access to POB library, see their utility.

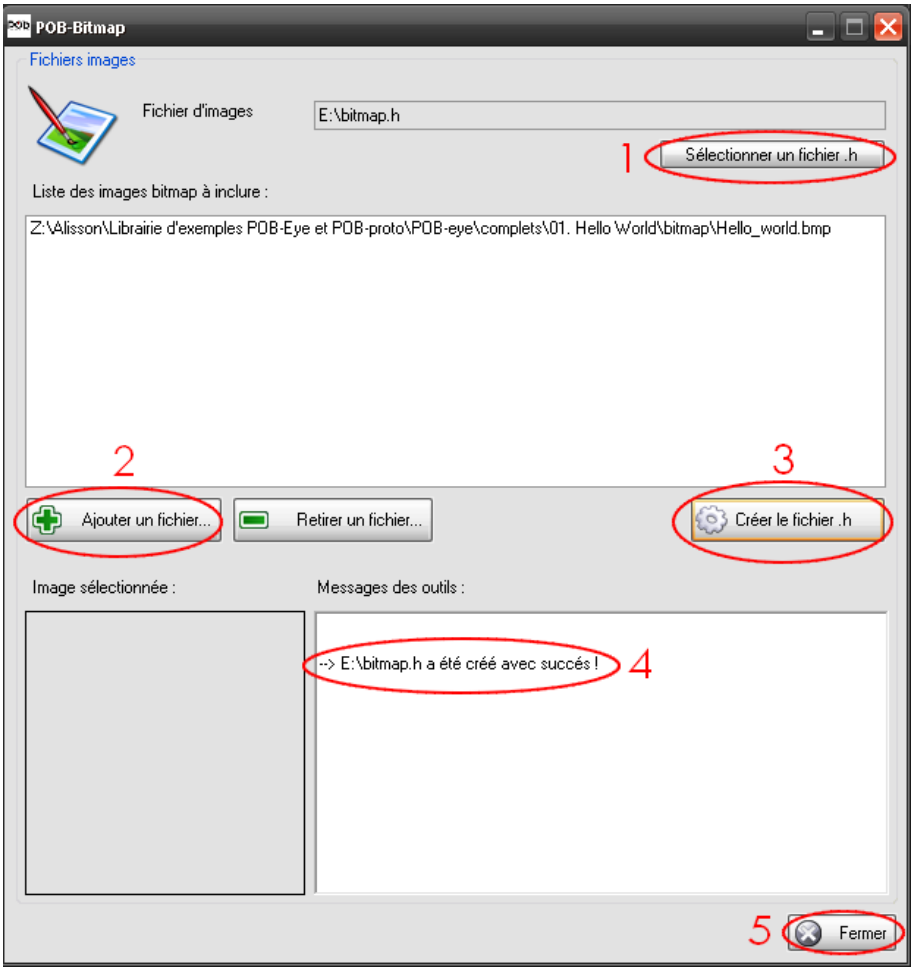
```
#include <pob-eye.h>
#include "Bitmap.h"  // include bitmap list

int main (void)
{
        //Initialize Buffer
        UInt8 LCD_Buffer [LCD_WIDTH*LCD_HEIGHT*BITS] ;
        GraphicBuffer ScreenBuffer ;

        //Initialize System
        InitPOBEYE();
        InitLCD();

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

        // Clear the graphic buffer
        ClearGraphicBuffer(&ScreenBuffer);

        //Draw the picture on the buffer
        DrawBitmap(0,0,IDB_HELLO_WORLD,bitmap,&ScreenBuffer);

        //Draw buffer in the LCD screen.
        DrawLCD(&ScreenBuffer);

        return 0;
}
```

This function is enough to display the bitmap hello_world.bmp on the LCD screen of your robot. You can also establish a basic form of communication via serial port between your POB-bot and your PC, using the example hello_world_text, which returns the string “Hello World” in a terminal (like Tera Term).

C. LOADING

Once your file .c registered, it's time to compile. To do this, open POB-Tools and create a new project (example: Hello_world.pobtools).

At Bitmap include file .h, click on *details*, *select include file*, and create one you call *bitmap.h*. (1) Then add your bitmap "hello_world.bmp" (2), and create the file. (3). A message confirming the creation of your file appears (4). You now have your Bitmap include file. Now, the program.



To compile your program, still in POB-tools, in output .hex, click on *details*, *select .hex file,* create *hello_world.hex*, add your .c file and compile. Your program is ready to be loaded on the POB-Bot. It remains to connect it to your computer with the serial cable, select options in the POB-Tools corresponding to your COM port, and download your program into your POB-bot in programming mode.

A reset later, the POB-bot displays on its screen the famous "Hello World!"

2.2 COMMUNICATE WITH POB-EYE

Now we will see how to establish communication between the POB-Bot and the user, with a computer and a free terminal emulator: Tera Term (http://ttssh2.sourceforge.jp/). We will therefore create a program ping_pong.c to send and receive information byte by byte, in the form of an echo text sent to the POB-Eye.

A. CODE

This really simple program does a few lines of code.

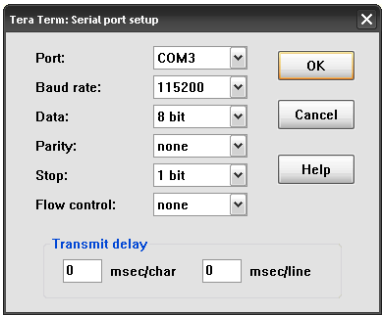
```
#include <pob-eye.h>

int main(void)
{
	//Initialize byte value
	UInt8 byte = 0;

	// Initialization of the POB-Eye and the Serial
	InitPOBEYE();
	InitUART((UInt16)(BR_115200|NO_PARITY|ONE_STOP_BIT|LENGTH_8_BIT));

	while (1)
	{
		//Get the byte from the UART
		byte = GetByteFromUART();

		//Send the byte to the UART
		SendByteToUART(byte);
	}
	return 0;
}
```

As always, we start by initializing the systems used, the POB-Eye and the serial port. Then the robot fetches each byte sent to the serial port to send it in turn, creating a basic form of communication through the serial port.

B. LOAD AND EXECUTE

In the same way as Example 1, compile and load your program into your bot with POB-Tools. You can start Tera Term. This requires a small setup at the launch. Select Serial Mode, and select the COM port for your connection with the POB-Bot. In Setup, Serial Port, set the transfer rate to 115 200 and confirm.

Each of the characters you enter in Tera Term terminal is returned by the POB-Bot.



2.3 DISPLAY IMAGE FROM CAMERA ON LCD SCREEN

In this example, we will capture the camera image of the POB-Eye and print it on the LCD screen.

CODE & TRANSFERT

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

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

	//LCD buffer, 128*64 pixels, 1 byte per pixel
	UInt8 LCD_Buffer [LCD_WIDTH*LCD_HEIGHT*BYTES];

	// Graphic buffer
	GraphicBuffer LCD_Screen;

	// Frame of camera
	RGBFrame FrameFromCam;

	//Initialize POB-Eye and LCD screen
	InitPOBEYE();
	InitLCD();

	// Init the graphic buffer : 128 per 64, 1 bytes per pixel and LCD_Buffer
	InitGraphicBuffer(&LCD_Screen,LCD_WIDTH,LCD_HEIGHT,EIGHT_BITS,LCD_Buffer);

	// Clear the graphic buffer
	ClearGraphicBuffer(&LCD_Screen);

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

	while (1)
	{
		// Grab the RGB components
		GrabRGBFrame();

		// Binary the RGB buffer
		BinaryRGBFrame(&FrameFromCam);

		// Put the pixel from the red component witch is binary, to the lcd buffer
		for (k=0,i=63 ; i ; i--)
		{
			for (j=0;j<120;j++,k++)
				LCD_Screen.buffer[k] = FrameFromCam.red[i+(j*88)];
			k+=8;
		}
		// draw the buffer on the screen
		DrawLCD(&LCD_Screen);
	}
	return 0;
}
```

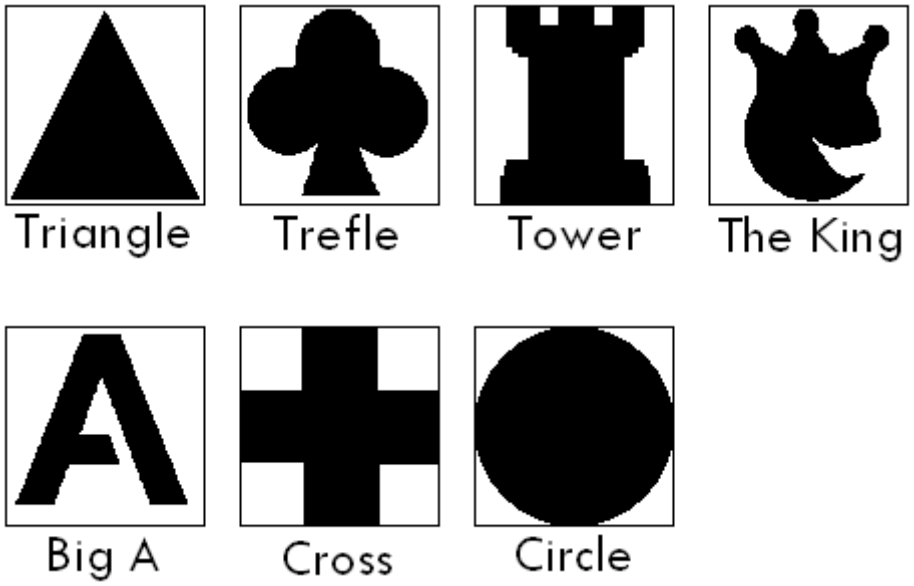
After initialization of all components and all the variables used, the image is sent from the camera in binary form on the LCD screen that stores and shape. The program is loaded in the same way that mentioned above.

2.4 FORMS RECOGNITION

This example allows us to use the camera of the POB-Eye to recognize shapes previously defined by the user.

FORMS

You must obtain or develop your own distinctive dictionary of forms for the robot. The program provides an example using the following forms:

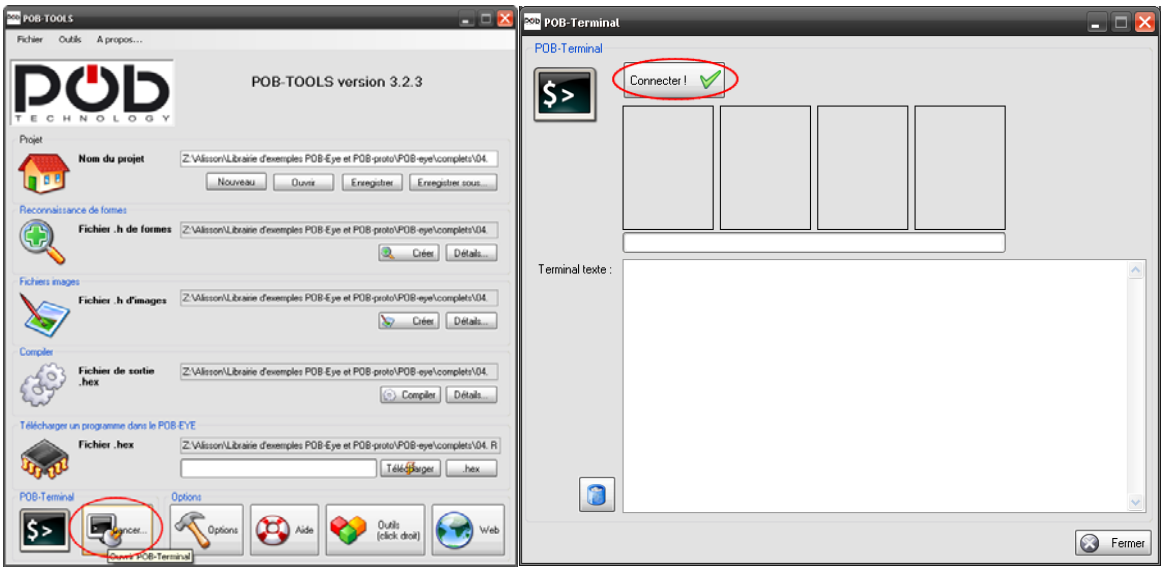


Like the images to display on a screen, the form to be recorded should be a bitmap. The optimal resolution is 100x100 pixels. To improve the recognition, use a white background behind your black forms. Similarly, some forms, such as circle, tend to recur with increased contrast on the camera image. If you want accurate detection, avoid these kinds of forms.

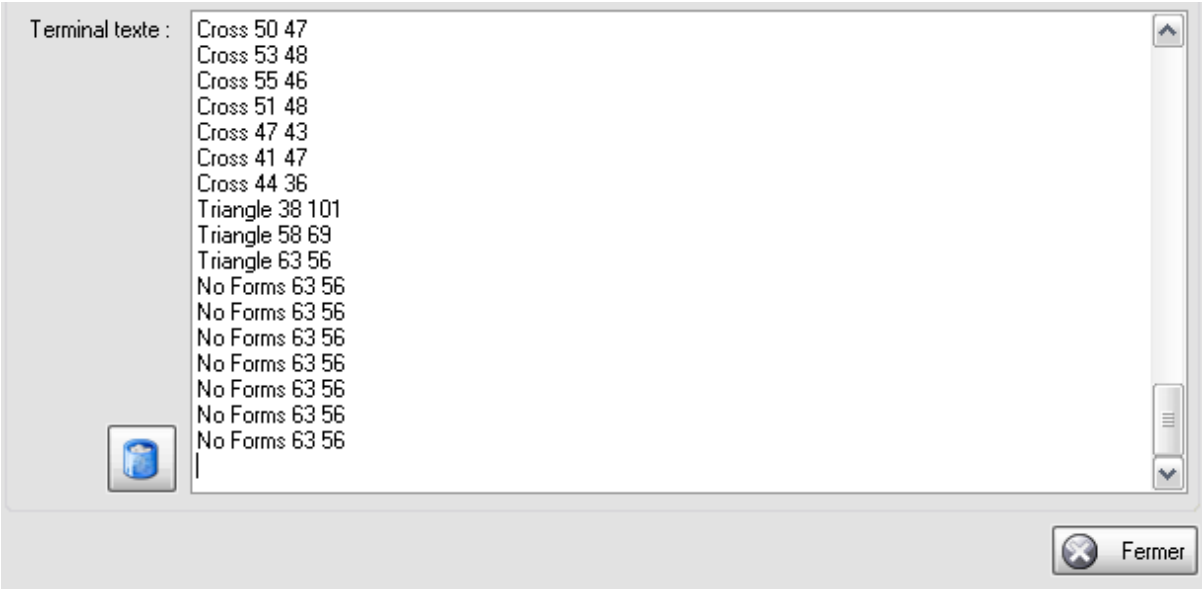
Once all your forms created, it only remains for you to create the Pattern include file with Pob-tools. Run the software. At *Pattern include file*, click *Details*, then *select include file*, and create a new file (example: pattern.h). Add all your shape bitmaps, and then click to create the file. This file will be included in your program.

DISPLAY

To allow the user to perform when the robot recognizes a form, the program prints the name of the shape and its position on the terminal Pob-tools. When you start your robot in execution mode, leave it connected to your computer with the serial port. Then, in Pob-tools, click on *Run POB Terminal*, then *Connect !*



The recognized forms are displayed on the terminal.



You can also, as in the example, using bitmap to display the name or form of recognized forms on the LCD screen.

CODE

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

// Include bitmap list and dictionnary of forms
#include "Bitmap.h"
#include "pattern.h"

int main (void)
{
	UInt8 i=0,Nb_Identify=0;

	// List of form
	Form ListOfForm[MAX_OF_FORM];

	// Struct of three pointers on the RGB components
	RGBFrame FrameFromCam;

	// System and LCD screen initialization
	InitPOBEYE();
	InitLCD();

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

	//The pixels will be stocked in the LCD buffer
	UInt8 LCD_Buffer [LCD_WIDTH*LCD_HEIGHT*BITS] ;
	GraphicBuffer ScreenBuffer ;

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

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

	while(1)
	{
		// grab the RGB components
		GrabRGBFrame();

		// Binary the three RGB Buffer
		BinaryRGBFrame(&FrameFromCam);

		// Try to identify the forms and make a list of it
		Nb_Identify=IdentifyForm(&FrameFromCam,ListOfForm,pattern);

		// Parse the list of the form and print result on the Pob-Terminal and the LCD Screen
		for (i=0;i<Nb_Identify;i++)
		{
			switch (ListOfForm[i].id)
			{
			case IDP_0_CROSS:
				// Draw bitmap on the buffer and the LCD screen
				DrawBitmap(0,0,IDB_CROSS,bitmap,&ScreenBuffer);
				DrawLCD(&ScreenBuffer);
				// Print text on the terminal
				PrintTextOnPobTerminal("Cross %d %d",ListOfForm[i].x,ListOfForm[i].y);
			break;
```

```
                        case IDP_1_BIGA:
                                DrawBitmap(0,0,IDB_BIGA,bitmap,&ScreenBuffer);
                                DrawLCD(&ScreenBuffer);
                                PrintTextOnPobTerminal("A Big A %d %d",ListOfForm[i].x,ListOfForm[i].y);
                        break;

                        case IDP_2_KING:
                                DrawBitmap(0,0,IDB_KING,bitmap,&ScreenBuffer);
                                DrawLCD(&ScreenBuffer);
                                PrintTextOnPobTerminal("The King %d %d",ListOfForm[i].x,ListOfForm[i].y);
                        break;

                        case IDP_3_TOWER:
                                DrawBitmap(0,0,IDB_TOWER,bitmap,&ScreenBuffer);
                                DrawLCD(&ScreenBuffer);
                                PrintTextOnPobTerminal("Tower %d %d",ListOfForm[i].x,ListOfForm[i].y);
                        break;

                        case IDP_4_TREFLE:
                                DrawBitmap(0,0,IDB_BIGA,bitmap,&ScreenBuffer);
                                DrawLCD(&ScreenBuffer);
                                PrintTextOnPobTerminal("Trefle %d %d",ListOfForm[i].x,ListOfForm[i].y);
                        break;

                        case IDP_5_TRIANGLE:
                                DrawBitmap(0,0,IDB_TRIANGLE,bitmap,&ScreenBuffer);
                                DrawLCD(&ScreenBuffer);
                                PrintTextOnPobTerminal("Triangle %d %d",ListOfForm[i].x,ListOfForm[i].y);
                        break;
                        default:
                        break;
                        }
                }
        if (Nb_Identify == 0)
                {
                DrawBitmap(0,0,IDB_NOFORMS,bitmap,&ScreenBuffer);
                DrawLCD(&ScreenBuffer);
                PrintTextOnPobTerminal("No Forms %d %d",ListOfForm[i].x,ListOfForm[i].y);
                }
        }
        return 0;
}
```

The program begins as usual by a full initialization of all the cards and sensors used. In the loop, we use the function IdentifyForm to compare the received image by the camera with the shapes to recognize. If a form is identified, its name is printed on the terminal, and drawn on the LCD screen with corresponding bitmaps files. (Bitmap.h)

2.5 FORMS RECOGNITION ON LCD SCREEN

This program is designed not only to project the images collected by the camera on the LCD screen, but also to proceed to the real-time analysis of form, a combination of examples 2.3 and 2.4. For this reason, we do not detail the methods, which are the same, but we will give you the code in this example.

CODE

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

//bitmap and form list
#include "bitmap.h"
#include "pattern.h"


#define BUFFER_WIDTH64
#define BUFFER_HEIGHT          64

//function to draw picture on both screen, when a form is recognize
static void DrawWhatIsKnown(Form What);
//function to draw rectangle arround the form known
static UInt8 DrawRect (Rect Rectangle,const GraphicBuffer *Target);


UInt8 LCD_Left_Buffer [BUFFER_WIDTH*BUFFER_HEIGHT*BYTES];  // array to store the left lcd screen, one byte per pixel
UInt8 LCD_Right_Buffer [BUFFER_WIDTH*BUFFER_HEIGHT*BITS];  // array to store the right lcd screen one bit per pixel.

GraphicBuffer LCD_Left ;  // left graphic buffer
GraphicBuffer LCD_Right ; // right graphic buffer
UInt8 X_Pic=5,Y_Pic=5;    // x,y to draw pictures of form known on the right LCD screen

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

        // LCD Buffer
        UInt8 LCD_Buffer[BUFFER_WIDTH*2*BUFFER_HEIGHT*BITS];
        // Number of form
        Int16 Nb_Identify=0 ;
        // List of forms
        Form  ListOfForm[MAX_OF_FORM];
        // RGB Frame of camera
        RGBFrame FrameFromCam ;

        //Initialize POB-EYE and LCD Screen
        InitPOBEYE();
        InitLCD();

        // Init Left and Right screens buffer
        InitGraphicBuffer( &LCD_Left,BUFFER_WIDTH,BUFFER_HEIGHT,EIGHT_BITS,LCD_Left_Buffer);
        InitGraphicBuffer( &LCD_Right,BUFFER_WIDTH,BUFFER_HEIGHT,ONE_BIT,LCD_Right_Buffer);

        // Clear the Left and Right screen
        ClearGraphicBuffer(&LCD_Left);
        ClearGraphicBuffer(&LCD_Right);

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

```
        while(1)
        {
                // Init the x,y of the first picture
                X_Pic=8;
                Y_Pic=8;

                // draw a frame in right buffer
                DrawBitmap(0,0,IDB_7_FRAME,bitmap,&LCD_Right);

                // Grab the RGB components
                GrabRGBFrame();

                // Binary the three RGB Buffer
                BinaryRGBFrame(&FrameFromCam);

                // draw red component in left buffer
                DrawComponentInABufferVideo(FrameFromCam.red, &LCD_Left );

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

                // Parse the list of the form
                for(i=0;i<Nb_Identify;i++)
                        DrawWhatIsKnown(ListOfForm[i]);

                // Draw the twis LCDs screens
                DrawLeftLCD(&LCD_Left);                 //the left for the real time video
                DrawRightLCD(&LCD_Right);       //the right for the result
        }
        return 0;
}
```

Two functions called in the main are not part of the library: DrawWhatIsKnow and DrawRect as follows.

```
void DrawWhatIsKnown (Form What)
{
        Rect Target;

        // draw the frame
        DrawBitmap(X_Pic,Y_Pic,IDB_8_LITTLE_FRAME,bitmap,&LCD_Right);

        // the picture of what is known in the frame
        DrawBitmap(X_Pic+3,Y_Pic+3,What.id -1,bitmap,&LCD_Right);

        //manage the (x,y) To draw the next known form on the right lcd screen
        X_Pic+=28;
        If ((X_Pic+28)>BUFFER_WIDTH)
        {
                Y_Pic+=26;
                X_Pic=8;
        }

        // draw a rectangle arround the known form on the left lcd screen

        Target.x=What.x - (What.width/2);               // manage the upper left corner of the form
        Target.y=What.y - (What.height/2);
        Target.width=What.width;
        Target.height=What.height;

        DrawRect(Target,&LCD_Left);                     // Draw the rectangle arround the form known.
}
```

```
UInt8 DrawRect (Rect Rectangle,const GraphicBuffer *Target)
{
	UInt8 i=0,X=0,Y=0,W=2,H=2;
	UInt16 tmp=0,SizeOfBuffer=0;

	SizeOfBuffer=Target->width*Target->height;

	/*****************************************************************/
	/* we have to manage a ratio between the video buffer of the frame	*/
	/* from the cam (88*120 pixels), and the LCD screen (64*64 pixels)	*/
	/*****************************************************************/

	X=(Rectangle.x*186)>>8;
	Y=(Rectangle.y*135)>>8;
	W+=((Rectangle.width)*186)>>8;
	H+=((Rectangle.height)*135)>>8;


	//draw the 2 horizontals lines
	for (i=0;i<W;i++)
	{
		tmp=X+Y*Target->width+i;
		if (tmp>(SizeOfBuffer)) return 0;
		Target->buffer[tmp]=0xFF;

		tmp+=H*Target->width;
		if (tmp>(SizeOfBuffer)) return 0;
		Target->buffer[tmp]=0xFF;
	}

	//draw the 2 verticals lines
	for (i=0;i<H;i++)
	{
		tmp=X;
		tmp+=((i+Y)*Target->width);
		if (tmp>(SizeOfBuffer)) return 0;
		Target->buffer[tmp]=0xFF;

		tmp+=W;
		if (tmp>(SizeOfBuffer)) return 0;
		Target->buffer[tmp]=0xFF;
	}

	/*****************************************************************/
	/* Drawing Horizontal line or vertical with a "for" instruction */
	/* it's more faster than using the DrawLine function		*/
	/*****************************************************************/

	return 1;
}
```

3. POB-PROTO LIBRARY

After an overview of the functions performed on the POB-Eye, here is a brief list of examples for POB-Proto.

3.1 POB-PROTO INITIALIZATION

The first example, and not least, is the initialization of the POB-Proto, which is not part of the library. To facilitate its use we recommend creating a .c containing the necessary functions that you attach to each of your programs using the POB-Proto.

CODE

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

//Function to initialize the POB-PROTO board
void InitPobProto (void)
{
	// struct to set the pob-proto
	PobProto	Proto;

	//to get the position of the analogic joystick, you have to set the PORTA as analogic input
	Proto.porta=ALL_PORTA_AS_ANA;

	//all pin of PORTC are configured to manage servomotors
	Proto.portc=RC7_AS_SERVO	| RC6_AS_SERVO |RC3_AS_SERVO |RC2_AS_SERVO|RC1_AS_SERVO |RC0_AS_SERVO;

	//RD0 RD1 RD2 RD3 are configured as digitals output to gear DC motor, RD4 RD5 RD6 RD7 are configured as digitals input
	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;

	//set the pob proto
	SetPobProto(&Proto);
}

int main(void)
{
	//init POB-EYE and Serial
	InitPOBEYE();
	InitUART((UInt16)(BR_115200|NO_PARITY|ONE_STOP_BIT|LENGTH_8_BIT));

	//pob-proto init
	InitPobProto ();

	return 0;

}
```

The function InitPobProto has ports A, C and D. The port A received the position of the analog joystick. Port C is dedicated to servomotors. The Port D includes digital inputs and outputs.

3.2 READ THE VALUE OF THE JOYSTICK

Now that the POB-Proto is initialized, we can discover the possible exploitation through some examples. Start with a program to read the position of the joystick and display it.
The joystick moves in two directions: horizontal and vertical. Function PrintTextOnPobLCD is used to display values in real time, and the axis to which they correspond.

CODE

Begin by providing you with the code of the function PrintTextOnPobLCD. We recommend you to save it into a file .c to re use later.

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

#include "math.h"
#include "bitmap.h"

//array for display the graphic interface
static GraphicBuffer        LCD_Buffer_Video;
static UInt8                LCD_Buffer[LCD_WIDTH*LCD_HEIGHT*BITS];
static UInt8                ASCII_Buffer[LCD_WIDTH*LCD_HEIGHT*BITS]; // Buffer to stock the ascii table in bitmap format

// Set the bitmap  picture of the first 127 ASCII Char in ASCII_GRAPHIC_BUFFER
// ASCII_GRAPHIC_BUFFER will be used to get the pictures of characters to display on the LCD
void InitAsciiBuffer()
{
        GraphicBuffer ASCII_GRAPHIC_BUFFER;
        InitGraphicBuffer(&ASCII_GRAPHIC_BUFFER,LCD_WIDTH,LCD_HEIGHT,ONE_BIT,ASCII_Buffer);
        ClearGraphicBuffer(&ASCII_GRAPHIC_BUFFER);
        DrawBitmap(0,0, IDB_ASCII,bitmap,&ASCII_GRAPHIC_BUFFER); //bitmap is an array built by pob-tools
}

void PrintTextOnPobLCD(int row, int col, char *string, UInt8 *Screen_Buffer)
{
        int i,j,k=0; // i = ascii char starting buffer; j = 0 to 7 (8 times 8 points)

while(*string)
        {
                // starting point in the ASCII_Buffer of the char to display,128 (16charx8lines of pixels) is a complete row of text
                i = (string[0]/16)*128 + (string[0]%16);
        // display char on Screen_Buffer, 8 bytes
        for (j=0; j<8; j++)
                {
                  // 8 intergers to define a bitmap of a char
                  Screen_Buffer[col+row*128+k*16]=ASCII_Buffer[i+j*16];
                  k++;
                }
        string++;
        k=0;
        // 1 byte right
        col+=1;
        if (col%16==0)
                {
                // Manages end of line and go to the next line on the left
                row++;
                col=0;
                        // Go to the top of the screen if the bottom is reached
                        if (row%8 == 0)  row=0;
                }
        }
}
```

PrintTextOnPobLCD function requires the conversion from bitmap format to a .h. The picture, like the include bitmap.h, are both provided. This function accepts the following arguments:

• row: Line of the first character on the LCD screen (about 1 to 8)

• Pass: Column of the first character on the LCD screen (about 1 to 16)

• string: string to display. Use the sprintf function to convert a string from integers and decimals.

• Screen_Buffer: Buffer to view.

Ascii_Buffer is a buffer of 127 characters in the ASCII table. Each character is 8 pixels by 8, and it can display 16 characters per line, and 8 per column.

The function InitPobProto will also be used to recover the value of the joystick, and we do not detail here because it was the subject of the previous example. The rest of the function works as follows:

As a first step, we initialize variables and systems, as always. Then performs a cleaning of the screen. Variables store the value of horizontal and vertical axes, and then passed string to be displayed with PrintTextOnPobLCD. They are finally displayed with some additional information to clarify their position. You get a real-time display of coordinates for each joystick axis. If there is a pressure on the joystick, a message appears for one second.

The code for this function is available below. The coordinates thus recovered can then be used, as will be done in the next example.

```
#include "pob-eye.h"
#include "pad.h"

void InitPobProto (void);
void IntroPOB(void);

// external functions declarations (for PrintTextOnPobLCD)
extern void InitAsciiBuffer();
extern void PrintTextOnPobLCD(int row, int col, char *string, UInt8 *Screen_Buffer);

int	main(void)
{
	//variable declaration (128 is the center of the LCD screen)
	int top_bottom_axe = 128;
	int right_left_axe = 128;
	int button = 0;
	char top_bottom[10];
	char right_left[10];

	//This buffers will stock the pixels to display,
	 GraphicBuffer			LCD_Buffer_Video;
	UInt8				LCD_Buffer[LCD_WIDTH*LCD_HEIGHT*BITS];
	// Buffer to stock the ascii table in bitmap format
	UInt8				 ASCII_Buffer[LCD_WIDTH*LCD_HEIGHT*BITS];

	//Initialize POB-EYE (lib), POB-LCD (lib) and POB-PROTO(source code at end of this file)
	InitPOBEYE();
	InitLCD();
	InitPobProto();
```

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

	//Clear and draw the buffer to make clean the screen
	ClearGraphicBuffer(&LCD_Buffer_Video);
	DrawLCD(&LCD_Buffer_Video);

	// Init Ascii buffer, use to write in the LCD screen with PrintTextOnPobLCD function
	InitAsciiBuffer();

	while (1)
	{
		//get the values of pad
		top_bottom_axe = GetPortAnalog(UP_DOWN_AXE);
		right_left_axe = GetPortAnalog(RIGHT_LEFT_AXE);
		button = GetPortAnalog(BUTTON);

		if (button >=100)
			{
			ClearGraphicBuffer(&LCD_Buffer_Video);
			//convert it from int to char*
			sprintf(top_bottom, "%d", top_bottom_axe);
			sprintf(right_left, "%d", right_left_axe);

			//Display values
			PrintTextOnPobLCD(1, 2, "Top", LCD_Buffer);
			PrintTextOnPobLCD(2, 1, "Bottom", LCD_Buffer);
			PrintTextOnPobLCD(5, 3, top_bottom, LCD_Buffer);
			PrintTextOnPobLCD(1, 8, "Right", LCD_Buffer);
			PrintTextOnPobLCD(2, 8, "Left", LCD_Buffer);
			PrintTextOnPobLCD(5, 8, right_left, LCD_Buffer);
			DrawLCD(&LCD_Buffer_Video);
			}
		// If button is pressed
		if (button < 100)
			{
			ClearGraphicBuffer(&LCD_Buffer_Video);
			DrawLCD(&LCD_Buffer_Video);
			PrintTextOnPobLCD(4, 1, "Button pressed", LCD_Buffer);
			DrawLCD(&LCD_Buffer_Video);
			Wait(1000000);
			ClearGraphicBuffer(&LCD_Buffer_Video);
			DrawLCD(&LCD_Buffer_Video);
			}
	}
 return 0;
}
```

3.3 MOVE THE BOT WITH THE JOYSTICK

When you know the position of the joystick, its coordinates can find various utilities, including the control of servomotors. Below : two examples of manipulation of servomotors by Joystick. The first allows you to move the servo controlling the head of the POB-Bot, the second will move your robot, depending on the position of the pad.

HEAD INCLINATION OF THE POB-BOT

With the analog joystick, you can tilt as you wish the head of your POB-bot. The head is articulated theory of 180 degrees, but its provision limits this angle a few dregrees less. Use 30th to 140th degree. Like in the previous example, PrintTextOnPobTerminal and InitPobProto will be used.

```
#include <string.h>
#include "pob-eye.h"
#include "pad.h"

// external functions declarations (for PrintTextOnPobLCD)
extern void InitAsciiBuffer();
extern void PrintTextOnPobLCD(int row, int col, char *string, UInt8 *Screen_Buffer);

int	main(void)
{
	//variable declaration (128 is the center of the LCD screen)
	int	top_bottom_axe = 128;
	int	right_left_axe = 128;
	char	top_bottom[10];
	char	right_left[10];
	int	position = 130;


	//This buffers will stock the pixels to display,
	 GraphicBuffer  LCD_Buffer_Video;
	UInt8			LCD_Buffer[LCD_WIDTH*LCD_HEIGHT*BITS];
	// Buffer to stock the ascii table in bitmap format
	UInt8			ASCII_Buffer[LCD_WIDTH*LCD_HEIGHT*BITS];

	//Initialize POB-EYE (lib), POB-LCD (lib) and POB-PROTO(file Functions.c)
	InitPOBEYE();
	InitLCD();
	InitPobProto();
	SwitchOnAllServo();

	// Initialize the Graphic buffer with 128 per 64, one pixel per bit and LCD_Buffer
	InitGraphicBuffer( &LCD_Buffer_Video, LCD_WIDTH,LCD_HEIGHT,ONE_BIT,LCD_Buffer);

	//Clear and draw the buffer to make clean the screen
	ClearGraphicBuffer(&LCD_Buffer_Video);
	DrawLCD(&LCD_Buffer_Video);
```

```
        while (1)
        {
                //get the values of pad
                top_bottom_axe = GetPortAnalog(UP_DOWN_AXE);
                right_left_axe = GetPortAnalog(RIGHT_LEFT_AXE);

                if (top_bottom_axe > 200)
                {
                        if (position >= 30)
                                position= position - 15;
                        SetServoMotor(0, position);
                }
                else if (top_bottom_axe < 100)
                {
                        if (position <=140)
                                position = position + 15;
                        SetServoMotor(0, position);
                }
                else if (right_left_axe >200)
                        MoveBot(RIGHT);
                else if (right_left_axe < 100)
                        MoveBot(LEFT);
                else
                        MoveBot(STOP);

                //convert it from int to char*
                sprintf(top_bottom, "%d", top_bottom_axe);
                sprintf(right_left, "%d", right_left_axe);

                //display it
                PrintTextOnPobLCD(1, 2, "Top", LCD_Buffer);
                PrintTextOnPobLCD(2, 1, "Bottom", LCD_Buffer);
                PrintTextOnPobLCD(5, 3, top_bottom, LCD_Buffer);
                PrintTextOnPobLCD(1, 8, "Right", LCD_Buffer);
                PrintTextOnPobLCD(2, 8, "Left", LCD_Buffer);
                PrintTextOnPobLCD(5, 8, right_left, LCD_Buffer);
                DrawLCD(&LCD_Buffer_Video);
        }
return 0;

}
```

A simple addition of conditions relating to the position of the sensor allows to move the servo on one axis.

MOVE OF THE POB-BOT

In the same way, the joystick can also be used for directional control of the robot. We just have to use the position of the joystick to activate the servomotors of the tank. As you can see, only a few functions calls change, the rest of the program is the same.

```
while (1)
{
        //get the values of pad
        top_bottom_axe = GetPortAnalog(UP_DOWN_AXE);
        right_left_axe = GetPortAnalog(RIGHT_LEFT_AXE);

        if (top_bottom_axe > 200)
                MoveBot(RUN);
        else if (top_bottom_axe < 100)
                MoveBot(BACK);
        else if (right_left_axe >200)
                MoveBot(RIGHT);
        else if (right_left_axe < 100)
                MoveBot(LEFT);
        else
                MoveBot(STOP);

         //convert it from int to char*
        sprintf(top_bottom, "%d", top_bottom_axe);
        sprintf(right_left, "%d", right_left_axe);

        //display it
        PrintTextOnPobLCD(1, 2, "Top", LCD_Buffer);
        PrintTextOnPobLCD(2, 1, "Bottom", LCD_Buffer);
        PrintTextOnPobLCD(5, 3, top_bottom, LCD_Buffer);
        PrintTextOnPobLCD(1, 8, "Right", LCD_Buffer);
        PrintTextOnPobLCD(2, 8, "Left", LCD_Buffer);
        PrintTextOnPobLCD(5, 8, right_left, LCD_Buffer);
        DrawLCD(&LCD_Buffer_Video);

}
```

CONTACT POB-TECHNOLOGY

POB-TECHNOLOGY
11, avenue Albert Einstein
69 100 VILLEURBANNE
France

Website	http://www.pob-technology.com/
Email	contact@pob-technology.com
Phone number	+33 (0)4 72 43 02 36
Fax	+33 (0)4 83 07 50 89