

RoBIOS-7 Library Functions

Version 7.3, Apr. 2025 -- RoBIOS is the operating system for the EyeBot controller.

The following libraries are available for programming the EyeBot controller in C/C++ or Python.
Unless noted otherwise, return codes are 0 when successful and non-zero if an error has occurred.

In application source files include: #include "eyebot.h"

Compile application to include RoBIOS library: \$gccarm myfile.c -o myfile.o

- [LCD Output](#)
- [Key Input](#)
- [Camera](#)
- [Image Processing](#)
- [System Functions](#)
- [Timer](#)
- [USB/Serial](#)
- [Audio](#)
- [Distance Sensors](#)
- [Servos and Motors](#)
- [V-Omega Driving Interface](#)
- [Digital and Analog I/O](#)
- [IR Remote Control](#)
- [Radio Communication](#)

- [Multitasking](#)
- [Simulation](#)

LCD Output

```
int LCDPrintf(const char *format, ...); // Print string and arguments on LCD
int LCDSetPrintf(int row, int column, const char *format, ...); // Printf from given position
int LCDClear(void); // Clear the LCD display and display buffers
int LCDSetPos(int row, int column); // Set cursor position in pixels for subsequent printf
int LCDGetPos(int *row, int *column); // Read current cursor position
int LCDSetColor(COLOR fg, COLOR bg); // Set color for subsequent printf
int LCDSetFont(int font, int variation); // Set font for subsequent print operation
int LCDSetFontSize(int fontsize); // Set LCD-font-size (7..18) for subsequent print operation
int LCDSetMode(int mode); // Set LCD Mode (0=default)
```

```
int LCDMenu(char *st1, char *st2, char *st3, char *st4); // Set menu entries for soft buttons
int LCDMenuI(int pos, char *string, COLOR fg, COLOR bg); // Set menu for i-th entry with color [1..4]
```

```
int LCDGetSize(int *x, int *y); // Get LCD resolution in pixels
int LCDPixel(int x, int y, COLOR col); // Set one pixel on LCD
COLOR LCDGetPixel(int x, int y); // Read pixel value from LCD
int LCDLine(int x1, int y1, int x2, int y2, COLOR col); // Draw line
int LCDArea(int x1, int y1, int x2, int y2, COLOR col, int fill); // Draw filled/hollow rectangle
int LCDCircle(int x1, int y1, int size, COLOR col, int fill); // Draw filled/hollow circle
int LCDImageSize(int t); // Define image type for LCD (default QVGA; 0,0; full)
int LCDImageStart(int x, int y, int xs, int ys); // Define image start position and size (default 0,0; max_x, max_y)
int LCDImage(BYTE *img); // Print color image at screen start pos. and size
int LCDImageGray(BYTE *g); // Print gray image [0..255] black..white
int LCDImageBinary(BYTE *b); // Print binary image [0..1] white..black
int LCDRefresh(void); // Refresh LCD output
```

Font Names and Variations:
HELVETICA (default), TIMES, COURIER
NORMAL (default), BOLD

Color Constants (COLOR is data type "int" in RGB order):
RED (0xFF0000), GREEN (0x00FF00), BLUE (0x0000FF), WHITE (0xFFFFFFFF), GRAY (0x808080), BLACK (0)
ORANGE, SILVER, LIGHTGRAY, DARKGRAY, NAVY, CYAN, TEAL, MAGENTA, PURPLE, MAROON, YELLOW, OLIVE

LCD Modes:
LCD_BGCOL_TRANSPARENT, LCD_BGCOL_NOTRANSPARENT, LCD_BGCOL_INVERSE, LCD_BGCOL_NOINVERSE, LCD_FGCOL_INVERSE,
LCD_FGCOL_NOINVERSE, LCD_AUTOREFRESH, LCD_NOAUTOREFRESH, LCD_SCROLLING, LCD_NOSCROLLING, LCD_LINEFEED,
LCD_NOLINEFEED, LCD_SHOWMENU, LCD_HIDEMENU, LCD_LISTMENU, LCD_CLASSICMENU, LCD_FB_ROTATE, LCD_FB_NOROTATION

Keys

```
int KEYGet(void); // Blocking read (and wait) for key press (returns KEY1..KEY4)
int KEYRead(void); // Non-blocking read of key press (returns NOKEY=0 if no key)
int KEYWait(int key); // Wait until specified key has been pressed (use ANYKEY for any key)
int KEYGetXY(int *x, int *y); // Blocking read for touch at any position, returns coordinates
int KEYReadXY(int *x, int *y); // Non-blocking read for touch at any position, returns coordinates
```

Key Constants:
KEY1..KEY4, ANYKEY, NOKEY

Camera

```
int CAMInit(int resolution); // Change camera resolution (will also set IP resolution)
int CAMRelease(void); // Stops camera stream
int CAMGet(BYTE *buf); // Read one color camera image
int CAMGetGray(BYTE *buf); // Read gray scale camera image
```

For the following functions, the Python API differs as in examples:

```
img = CAMGet()
gray = CAMGetGray()
```

Resolution Settings:

QQVGA(160x120), QVGA(320x240), VGA(640x480), CAM1MP(1296x730), CAMHD(1920x1080), CAM5MP(2592x1944), CUSTOM (LCD only)
Variables CAMWIDTH, CAMHEIGHT, CAMPIXELS (=width*height) and CAMSIZE (=3*CAMPixels) will be automatically set,
(BYTE is data type "unsigned char").

Constant sizes in bytes for color images and number of pixels:

QQVGA_SIZE, QVGA_SIZE, VGA_SIZE, CAM1MP_SIZE, CAMHD_SIZE, CAM5MP_SIZE
QQVGA_PIXELS, QVGA_PIXELS, VGA_PIXELS, CAM1MP_PIXELS, CAMHD_PIXELS, CAM5MP_PIXELS

Data Types:

```
typedef BYTE QVGAcol [120][160][3];    typedef BYTE QVGAgray [120][160];
typedef BYTE QVGAcol [240][320][3];    typedef BYTE QVGAgray [240][320];
typedef BYTE VGACol [480][640][3];     typedef BYTE VGAGray [480][640];
typedef BYTE CAM1MPcol[730][1296][3];   typedef BYTE CAM1MPgray[730][1296];
typedef BYTE CAMHDcol [1080][1920][3];  typedef BYTE CAMHDgray [1080][1920];
typedef BYTE CAM5MPcol[1944][2592][3];  typedef BYTE CAM5MPgray[1944][2592];
```

Image Processing

Basic image processing functions using the previously set camera resolution are included in the RoBIOS library. For more complex functions see the OpenCV library.

```
int  IPSetSize(int resolution);           // Set IP resolution using CAM constants (also automatically set by CAMInit)
int  IPReadFile(char *filename, BYTE* img); // Read PNM file, fill/crop if req.; return 3:color, 2:gray, 1:b/w, -1:error
int  IPWriteFile(char *filename, BYTE* img); // Write color PNM file
int  IPWriteFileGray(char *filename, BYTE* gray); // Write gray scale PGM file

void IPLaplace(BYTE* grayIn, BYTE* grayOut); // Laplace edge detection on gray image
void IPSobel(BYTE* grayIn, BYTE* grayOut); // Sobel edge detection on gray image
void IPCol2Gray(BYTE* imgIn, BYTE* grayOut); // Transfer color to gray
void IPGray2Col(BYTE* imgIn, BYTE* colOut); // Transfer gray to color
void IPRGB2Col (BYTE* r, BYTE* g, BYTE* b, BYTE* imgOut); // Transform 3*gray to color
void IPCol2HSI (BYTE* img, BYTE* h, BYTE* s, BYTE* i); // Transform RGB image to HSI
void IPOverlay(BYTE* c1, BYTE* c2, BYTE* cOut); // Overlay c2 onto c1, all color images
void IPOverlayGray(BYTE* g1, BYTE* g2, COLOR col, BYTE* cOut); // Overlay gray image g2 onto g1, using col

COLOR IPPRGB2Col(BYTE r, BYTE g, BYTE b); // PIXEL: RGB to color
void IPPCol2RGB(COLOR col, BYTE* r, BYTE* g, BYTE* b); // PIXEL: color to RGB
void IPPCol2HSI(COLOR c, BYTE* h, BYTE* s, BYTE* i); // PIXEL: RGB to HSI for pixel
BYTE IPPRGB2Hue(BYTE r, BYTE g, BYTE b); // PIXEL: Convert RGB to hue (0 for gray values)
void IPPRGB2HSI(BYTE r, BYTE g, BYTE b, BYTE* h, BYTE* s, BYTE* i); // PIXEL: Convert RGB to hue, sat, int; hue=0 for gray values
```

For the following functions, the Python API differs as in examples:

```
edge = IPLaplace (gray_img)
edge = IPSobel   (gray_img)
gray = IPCol2Gray(col_img)
col = IPGray2Col(gray_img)
[h_gray, s_gray, i_gray] = IPCol2HSI(col_img)
col = IPOverlay (col_source, col_overlay)
col = IPOverlayGray(gray_source, gray_overlay, col_value)
```

System Functions

```
char * OSExecute(char* command); // Execute Linux program in background
int OSVersion(char* buf); // RoBIOS Version
int OSVersionIO(char* buf); // RoBIOS-IO Board Version
int OSMachineSpeed(void); // Speed in MHz
int OSMachineType(void); // Machine type
int OSMachineName(char* buf); // Machine name
int OSMachineID(void); // Machine ID derived from MAC address
```

Timer

```
int OSWait(int n); // Wait for n/1000 sec
TIMER OSAttachTimer(int scale, void (*fct)(void)); // Add fct to 1000Hz/scale timer
int OSDetachTimer(TIMER t); // Remove fct from 1000Hz/scale timer

int OSGetTime(int *hrs,int *mins,int *secs,int *ticks); // Get system time (ticks in 1/1000 sec)
int OSGetCount(void); // Count in 1/1000 sec since system start
```

USB/Serial Communication

```
int SERInit(int interface, int baud,int handshake); // Init communication (see parameters below), interface number as in HDT file
int SERSendChar(int interface, char ch); // Send single character
int SERSend(int interface, char *buf); // Send string (Null terminated)
char SERReceiveChar(int interface); // Receive single character
int SERReceive(int interface, char *buf, int size); // Receive String (Null terminated), returns number of chars received
int SERFlush(int interface); // Flush interface buffers
int SERClose(int interface); // Close Interface
```

Communication Parameters:

Baudrate: 50 .. 230400

Handshake: NONE, RTSCTS

Interface: 0 (serial port), 1..20 (USB devices, names are assigned via [HDT](#) entries)

Audio

```
int AUBeep(void); // Play beep sound
int AUPlay(char* filename); // Play audio sample in background (mp3 or wave)
int AUDone(void); // Check if AUPlay has finished
int AUMicrophone(void); // Return microphone A-to-D sample value
```

Use Analog data functions to record microphone sounds (channel 8).

Distance Sensors

Position Sensitive Devices (PSDs) are using infrared beams to measure distance and need to be calibrated in [HDT](#) to get correct distance readings. LIDAR (Light Detection and Ranging) is a single-axis rotating laser scanner.

```
int PSDGet(int psd); // Read distance value in mm from PSD sensor [1..6]
int PSDGetRaw(int psd); // Read raw value from PSD sensor [1..6]
int LIDARGet(int distance[]); // Measure distances in [mm]; default 360° and 360 points
int LIDARSet(int range, int tilt, int points); // range [1..360°], tilt angle down, number of points
```

PSD Constants:
 PSD_FRONT, PSD_LEFT, PSD_RIGHT, PSD_BACK
 assuming PSD sensors in these directions are connected to ports 1, 2, 3, 4.

LIDAR Constants:
 LIDAR_POINTS Total number of points returned
 LIDAR_RANGE Angular range covered, e.g. 180°

Servos and Motors

Motor and Servo positions can be calibrated through [HDT](#) entries.

```
int SERVOSet(int servo, int angle); // Set servo [1..14] position to [1..255] or power down (0)
int SERVOSetRaw(int servo, int angle); // Set servo [1..14] position bypassing HDT
int SERVORange(int servo, int low, int high); // Set servo [1..14] limits in 1/100 sec
```

```
int MOTORDrive(int motor, int speed); // Set motor [1..4] speed in percent [-100 ..+100]
int MOTORDriveRaw(int motor, int speed); // Set motor [1..4] speed bypassing HDT
int MOTORPID(int motor, int p, int i, int d); // Set motor [1..4] PID controller values [1..255]
int MOTORPIDOff(int motor); // Stop PID control loop
int MOTORSpeed(int motor, int ticks); // Set controlled motor speed in ticks/100 sec
```

```
int ENCODERRead(int quad); // Read quadrature encoder [1..4]
int ENCODERReset(int quad); // Set encoder value to 0 [1..4]
```

V-Omega Driving Interface

This is a high level wheel control for differential driving. It always uses motor 1 (left) and motor 2 (right). Motor spinning directions, motor gearing and vehicle width are set in the [HDT](#) file.

```
int VWSetSpeed(int linSpeed, int angSpeed); // Set fixed linSpeed [mm/s] and [degrees/s]
int VWGetSpeed(int *linSpeed, int *angSpeed); // Read current speeds [mm/s] and [degrees/s]
int VWSetPosition(int x, int y, int phi); // Set robot position to x, y [mm], phi [degrees]
int VWGetPosition(int *x, int *y, int *phi); // Get robot position as x, y [mm], phi [degrees]
```

```
int VWStraight(int dist, int lin_speed); // Drive straight, dist [mm], lin. speed [mm/s]
int VWTurn(int angle, int ang_speed); // Turn on spot, angle [degrees], ang. speed [degrees/s]
int VWCurve(int dist, int angle, int lin_speed); // Drive Curve, dist [mm], angle (orientation change) [degrees], lin. speed [mm/s]
int VWDrive(int dx, int dy, int lin_speed); // Drive x[mm] straight and y[mm] left, x>|y|
int VWRemain(void); // Return remaining drive distance in [mm]
int VWDone(void); // Non-blocking check whether drive is finished (1) or not (0)
int VWWait(void); // Suspend current thread until drive operation has finished
int VWStalled(void); // Returns number of stalled motor [1..2], 3 if both stalled, 0 if none
```

All VW functions return 0 if OK and 1 if error (e.g. destination unreachable)

For the following functions, the Python API differs as in examples:

```
[v,w] = VWGetSpeed()
[x,y,p] = VWGetPosition()
```

Digital and Analog Input/Output

```
int DIGITALSetup(int io, char direction); // Set IO line [1..16] to i-n/o-ut/I-n pull-up/J-n pull-down
int DIGITALRead(int io); // Read and return individual input line [1..16]
int DIGITALReadAll(void); // Read and return all 16 io lines
int DIGITALWrite(int io, int state); // Write individual output [1..16] to 0 or 1
```

```
int ANALOGRead(int channel); // Read analog channel [1..8]
int ANALOGVoltage(void); // Read analog supply voltage in [0.01 Volt]
int ANALOGRecord(int channel, int iterations); // Record analog data (e.g. 8 for microphone) at 1kHz (non-blocking)
int ANALOGTransfer(BYTE* buffer); // Transfer previously recorded data; returns number of bytes
```

Default for digital lines: [1..8] are input with pull-up, [9..16] are output
 Default for analog lines: [0..8] with 0: supply-voltage and 8: microphone
 IO settings are: i: input, o: output, I: input with pull-up res., J: input with pull-down res.

IR Remote Control

These commands allow sending commands to an EyeBot via a standard infrared TV remote (IRTV). IRTV models can be enabled or disabled via a [HDT](#) entry. Supported IRTV models are: Chunghop L960E Learn Remote

```
int IRTVGet(void); // Blocking read of IRTV command
```

```
int IRTVRead(void); // Non-blocking read, return 0 if nothing
int IRTVFlush(void); // Empty IRTV buffers
int IRTVGetStatus(void); // Checks to see if IRTV is activated (1) or off (0)
```

Defined Constants for IRTV buttons are:
IRTV_0 .. IRTV_9, IRTV_RED, IRTV_GREEN, IRTV_YELLOW, IRTV_BLUE,
IRTV_LEFT, IRTV_RIGHT, IRTV_UP, IRTV_DOWN, IRTV_OK, IRTV_POWER

Radio Communication

These functions require a WiFi modules for each robot, one of them (or an external router) in DHCP mode, all others in slave mode.
Radio can be activated/deactivated via a [HDT](#) entry. The names of all participating nodes in a network can also be stored in the [HDT](#) file.

```
int RADIOInit(void); // Start radio communication
int RADIOGetID(void); // Get own radio ID
int RADIOSend(int id, char* buf); // Send string (Null terminated) to ID destination
int RADIOReceiv(int *id_no, char* buf, int size); // Wait for message, then fill in sender ID and data, returns number of chars received
int RADIOCheck(void); // Check if message is waiting: 0 or 1 (non-blocking); -1 if error
int RADIOStatus(int IDlist[]); // Returns number of robots (incl. self) and list of IDs in network
int RADIORelease(void); // Terminate radio communication
```

ID numbers match last byte of robots' IP addresses.

For the following functions, the Python API differs as in examples:

```
[partnerid, buf] = RADIOReceive() # max 1024 Bytes
[total, ids] = RADIOStatus() # max 256 entries
```

Multitasking

For Multitasking, simply use the pthread functions.
A number of multitasking sample programs are included in the demo/MULTI directory.

Simulation *only*

These functions will **only** be available when run in a simulation environment, in order to get ground truth information and to repeat experiments with identical setup.

```
void SIMGetRobot (int id, int *x, int *y, int *z, int *phi);
void SIMSetRobot (int id, int x, int y, int z, int phi);
void SIMGetObject(int id, int *x, int *y, int *z, int *phi);
void SIMSetObject(int id, int x, int y, int z, int phi);
int SIMGetRobotCount()
int SIMGetObjectCount()
```

id=0 means own robot; id numbers run from 1..n

For the following functions, the Python API differs as in examples:

```
[x,y,z,p] = SIMGetRobot(id)
[x,y,z,p] = SIMGetObject(id)
```