EyeBot 8 User Guide

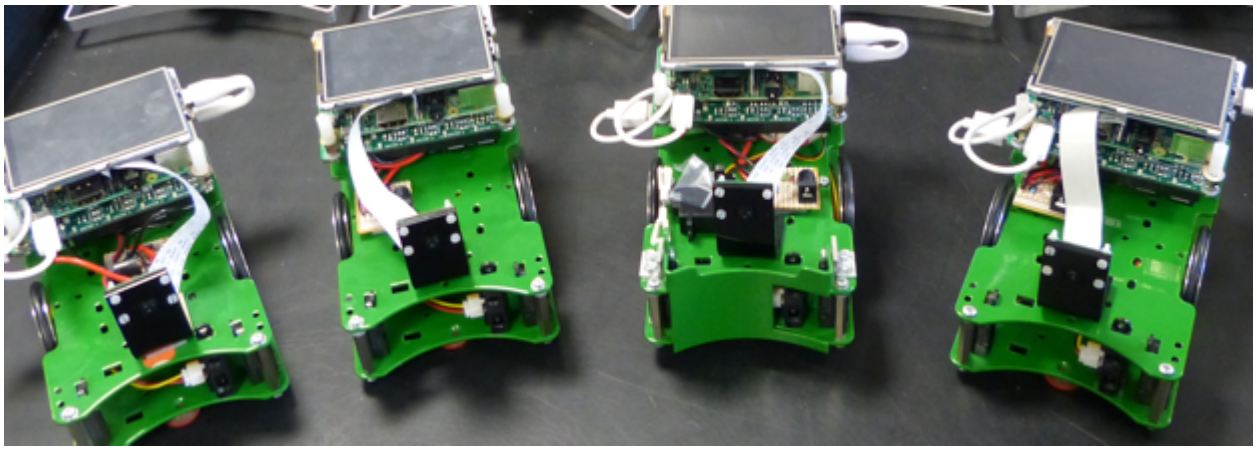
Thomas Bräunl, Marcus Pham, Remi Keat, Daniel Holding
March 5, 2024

EyeBot 8 is the 2024 version of the EyeBot embedded controller for robotics applications. It is based on a Raspberry Pi board with optional LCD display, linked via I/O pins to the EyeBot-8 IO-board, which has hardware and software drivers for motors and digital or analog sensors. Running on top of Raspberry Pi OS, our RoBIOS user interface software provides an extensive robotics library that allows the simple design of robot application programs in C, C++ or Python using the RoBIOS API.

Link https://roblab.org/eyebot/

In the following, we will discuss each of these components separately.

Contents



CHAPTER 1: EyeBot Use Interface

The EyeBot user interface and the RoBIOS library (robot-BIOS) run on a standard Raspberry Pi board. We recommend to use an optional 3.5" Waveshare-Touchscreen-LCD on the Raspberry Pi, however, connecting a monitor via HDMI (requires change of settings) or using Microsoft Remote Desktop will work as well. For applications using vision, the Raspberry Pi camera needs to be connected to the board as well.

In order to drive motors or read sensors, an EyeBot IO-Board is required (see below), which has two USB links (USB-to-USB-micro) to the Raspberry Pi, one for exchanging data and one for supplying power from the IO-Board to the Raspberry Pi and LCD.

The latest Raspberry Pi image for an 8GB SD card is available from:
https://roblab.org/rasp/images/

More frequent updates of the EyeBot software package are available from:
https://roblab.org/rasp/eyebot/

Once an image file has been downloaded and is being run on the Raspberry Pi, EyeBot upgrades can simply be installed on the touch of a button on the controller, as shown later.

The image file has been set by default to use the LCD and to do an autostart of the EyeBot user interface. Both can be deactivated in the Hardware Description Table (HDT file) as shown in a later chapter. So by default, the Raspberry Pi will display the following screen after the initial Linux boot sequence:

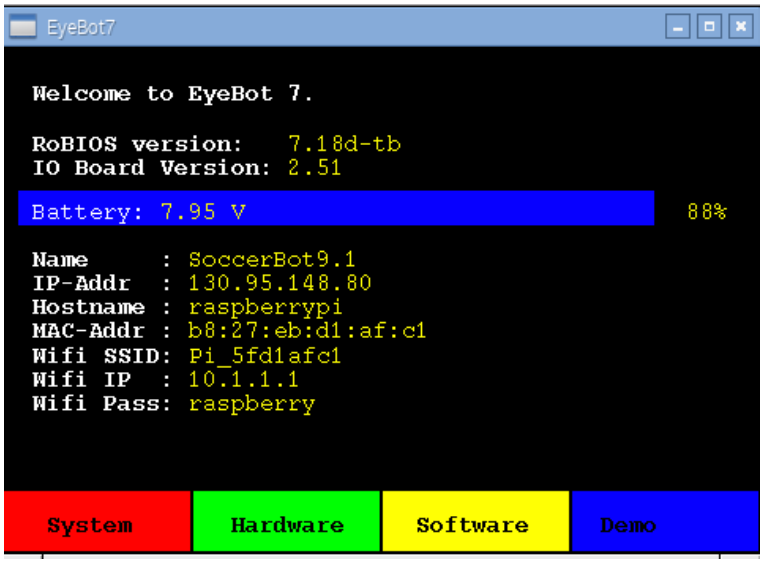


Figure 1.1: EyeBot Start Screen

The four colored areas below on the bottom of the page are soft keys, which can be pressed either by hand or with a stylus to select a menu item. The four top-level menu items are *System, Hardware, Software,* and *Demo*.

For connecting from the outside, you can either use the controller's LAN port or its WIFI connection. When using a LAN cable, the home page lists the IP-Address under

which you can connect. In the above Figure, the controller is connected to a network, however, the stand-alone default IP address is

10.0.0.1

For WIFI connections, there are again two choices: hotspot or network client. The default is “hotspot”, so each controller creates its own local network with default IP address

10.1.1.1 with password **raspberry**

This allows an easy connection from any desktop, laptop, tablet or mobile phone to the controller. Once connected to the controller (either through LAN or WLAN), files can be transferred through “scp” or the more user-friendly “Filezilla”. Login is available either via a console window using “slogin” (Mac, Windows) or “putty” (Windows), or more comprehensive when graphics output is involved using Microsoft Remote Desktop (Mac, Windows, tablets, phones). Standard login name is

pi with password **rasp**

1.1 System Commands

The system page lists a number of performance values of the controller and allows the selection of three system specific submenus.

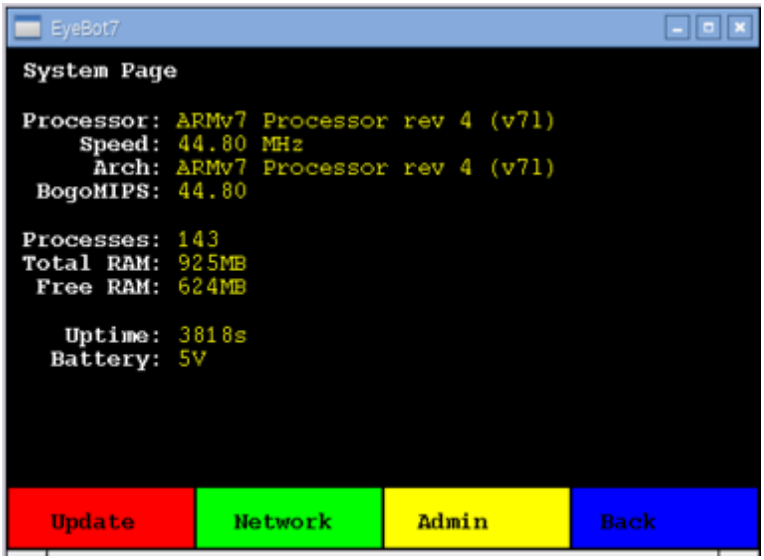


Figure 1.2: System page

1.1.1 Update

The Update page allows an update of either the RoBIOS system or of the HDT file. In case of an RoBIOS update, the controller needs to be connected to the Internet, either via a LAN cable or appropriate WLAN settings. The new RoBIOS system will be downloaded from the Robotics server and installed. The older version will remain in the home directory under the name “eyebot-*year-month-day*”.

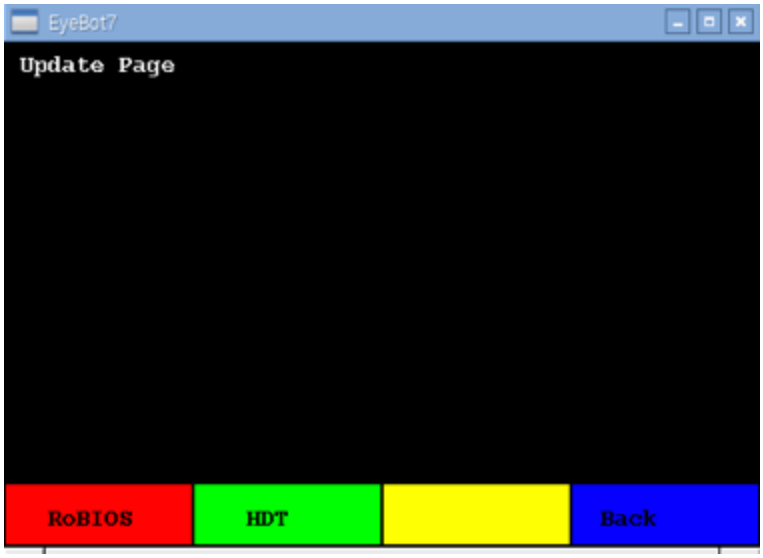


Figure 1.3: Update page

The HDT update will not download new data. Instead it will let the user select a HDT file from a list of existing ones in the directory: eyebot/bin/hdt/

1.1.2 Network

The network page lets the user change the WIFI settings of the controller. By default, the controller acts as a WIFI hotspot with an SSID derived from the Raspberry's MAC address (in this example Pi_5fd1afc1) and default password "raspberry". This is a very convenient method to simplify the first connection to the controller as well as allowing the operation of several controllers in the same room.

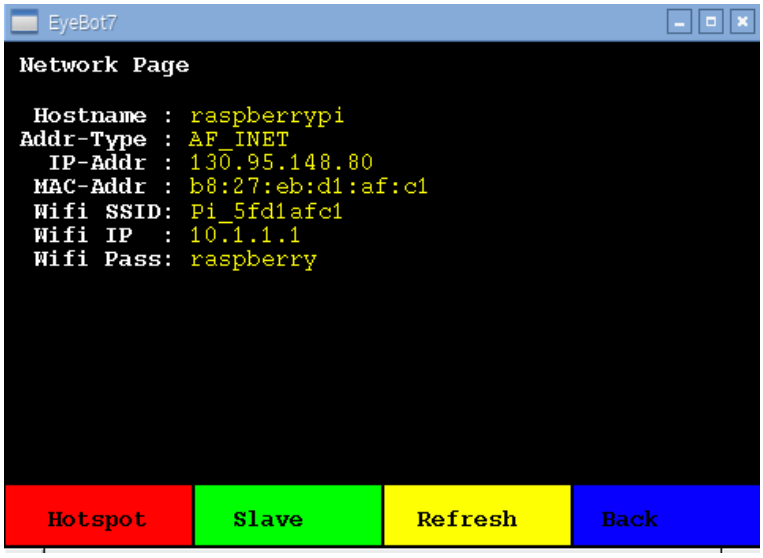


Figure 1.4: Network page

As default IP addresses, we have defined the easy to remember IPs:

10.0.0.1	for a LAN (cable) connection
10.1.1.1	for a WLAN (wireless) connection

The easiest way to connect to the controller in default configuration is using Microsoft's free Remote Desktop App (for Windows, MacOS, iOS, etc.) or an SSH client such as Putty (Windows) or a command shell with command "slogin" (MacOS). After joining the Pi_xxx network (password raspberry), one can simply login to the controller by any of these methods. Default login name is "**pi**", password is "**rasp**".

This network menu allows an easy transition between the controller acting as a WIFI hotspot versus joining an existing WLAN network ("slave"). SSID and password of the slave network can be set in the controller's HDT file (location: eyebot/bin/hdt.txt).

1.1.3 Admin

Admin page allows the swapping between LCD (default) and HDMI-connected displays. It also allows to reboot the controller or to leave the EyeBot system and return to the Linux X interface.

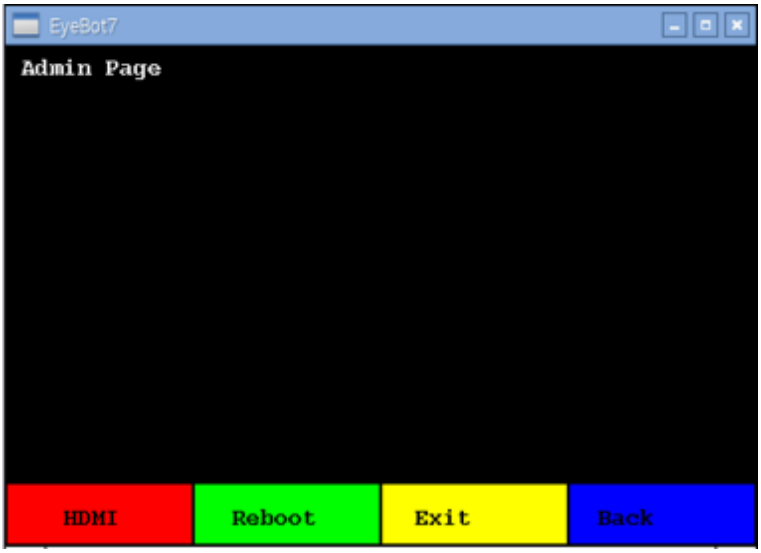


Figure 1.5: Admin page

1.2 Hardware

The hardware page lists all actuators and sensors currently connected to the controller via the EyeBot-IO-Board and specified in the HDT file. In the example below, we have connected 2 motors, 2 encoders, 3 PSD ("position sensitive devices" or infrared distance sensors), 1 infrared TV remote as input device, and 2 servos. All these settings can be further specified and parameterized in the HDT file.

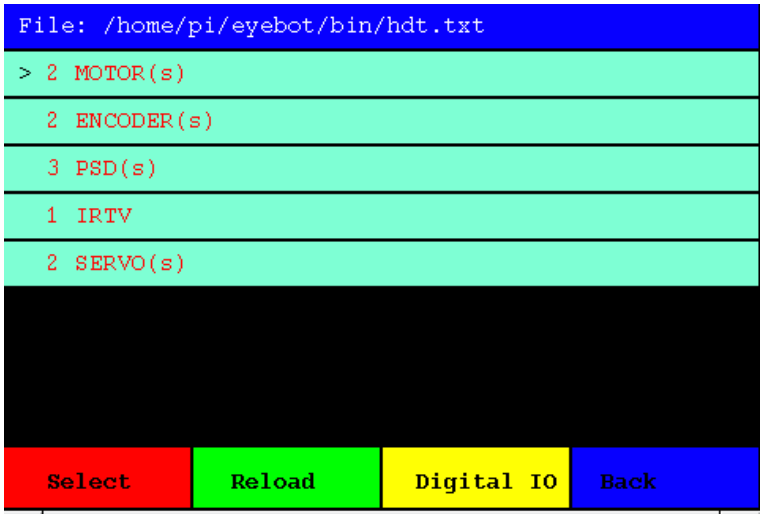


Figure 1.6: HDT page

We can now select any of these devices and test them individually. In this example we have selected the shaft encoders, which are connected to the individual motors. In the following menu, we can now select the encoder we want to try.

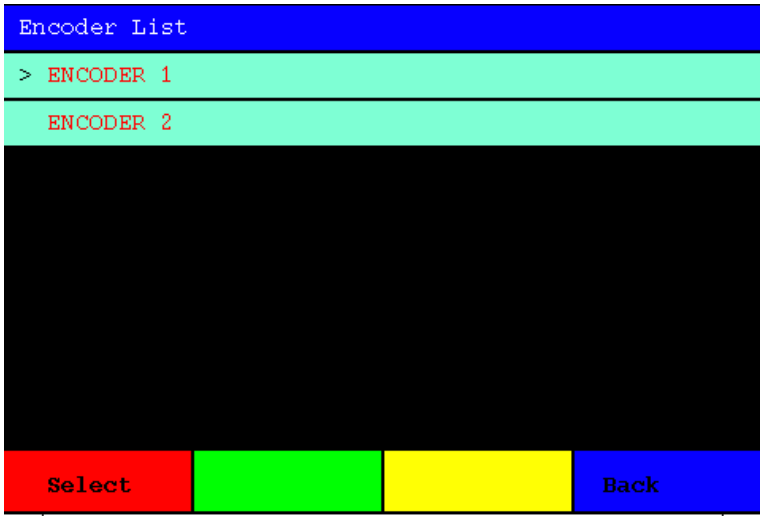


Figure 1.7: Encoder selection from HDT list

We went with “Encoder 1” and have now an encoder-specific test menu, which not only displays the current encoder value, but also calculates the encoder speed and lets us increase or decrease the associated motor speed to see how the encoder values change.

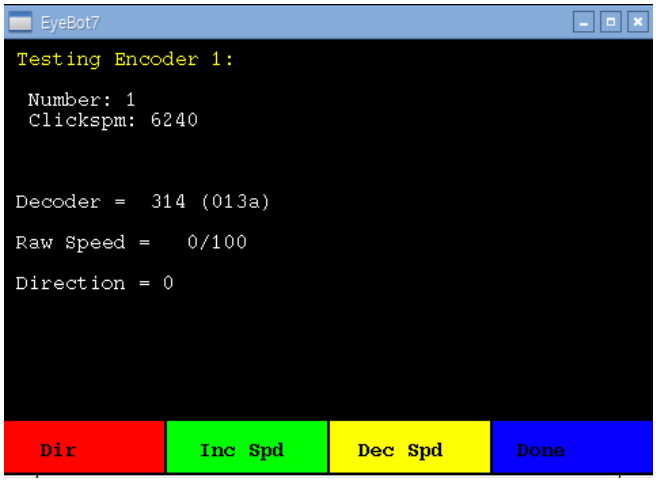


Figure 1.8: Encoder testing page (HDT)

Another choice from the HDT list is testing the PSD (distance sensors) as shown in the image below.

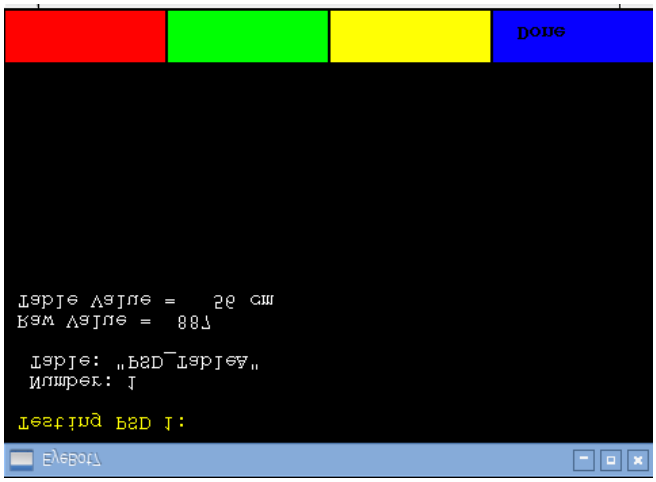


Figure 1.9: PSD testing page (HDT)

Finally, the “Digital IO” sub-menu displays the status of all 16 digital input pins of the controller.

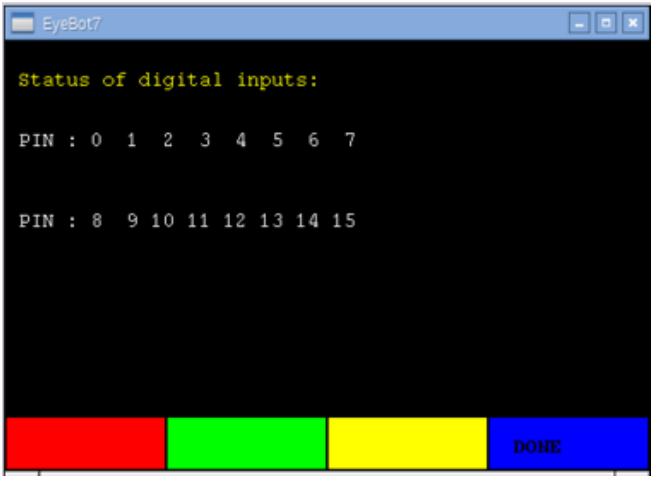


Figure 1.10: Digital IO testing page (HDT)

1.3 Software

User programs can be stored in a special directory with default path:
/home/pi/usr

With this menu, they can be conveniently executed from the user interface. If an executable program under the name of "**startup**" is stored in this directory, then RoBIOS will automatically run this program at boot time as a "turn-key system". This way, a user program can be written that will directly start at power-up of the system, without the need of pressing any buttons.

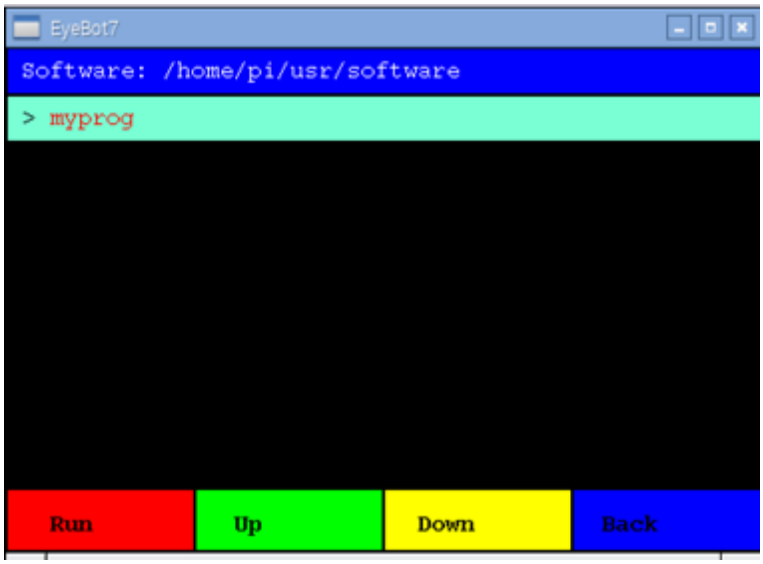


Figure 1.11: User program select page

Programs written in C or C++ can make use of the RoBIOS API (see later Chapter). Individual program files can easily be compiled with the command script:

gccarm myfile.c

This takes care of all required headers and libraries. For larger projects a Makefile is more suitable. A good start is copying the Makefile from one of the demo directories (see below).

1.4 Demo Programs

A number of demo programs are made available to EyeBot users. The menu structure is identical to the software sub-menu, but with a number of sub-directories with various demo programs.

All demo programs are in the path "eyebot/demo" and are complete with all sources and Makefile. All that is required to adapt any of the demo files or create a new one, is to edit the source and then type the Linux command:

make

The system will re-compile all programs in the directory and name the executables "program.demo"; they will then be automatically available from the EyeBot's demo menu.

The list of demo directories is oriented on the RoBIOS functions' API groups (see below).

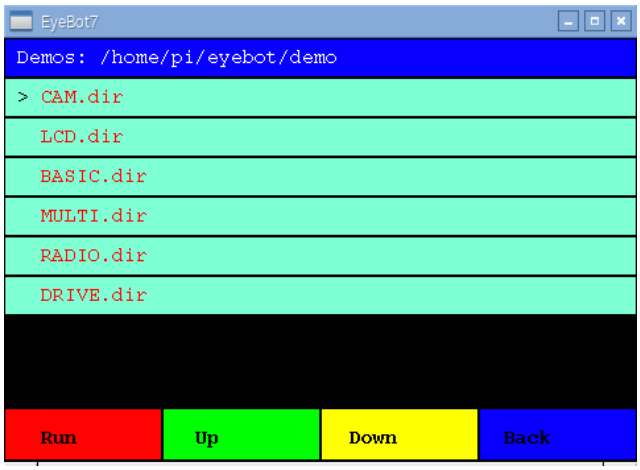


Figure 1.12: Demo program directory select page

Selecting the CAM demo group gives us a number of demo programs to choose from:

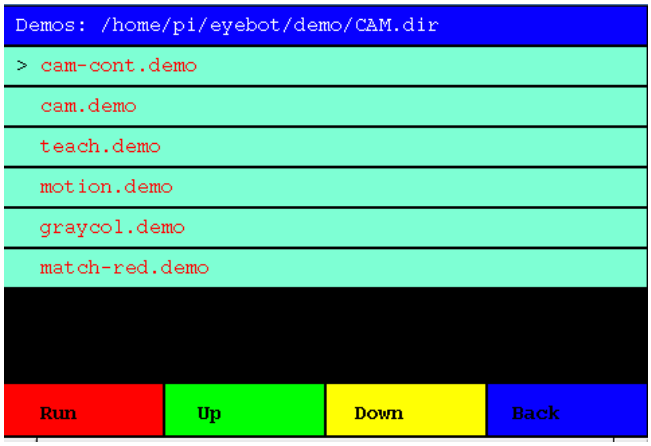


Figure 1.13: Executable demo program selection page

Finally, executing this particular demo program, will run a program that displays the camera image on the screen.

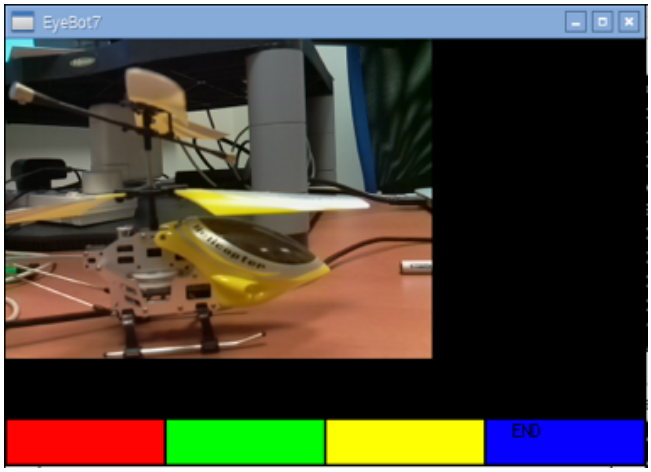


Figure 1.14: Demo program run

CHAPTER 2: RoBIOS Library

The RoBIOS library (Robot Basic Input / Output System) has evolved over many years. It represents a complete, but compact API (application programmer interface) for writing embedded and robotic applications, without having to worry about low-level sensor and actuator details. The latest library version is available from:

https://roblab.org/eyebot/robios.html

Version 7.3, Jan. 2023-- 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

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 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 (0xFFFFFF), 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
```

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

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

```
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 "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 QQVGAcol  BYTE  [120][160][3];    typedef QQVGAgray  BYTE [120][160];
typedef QVGAcol   BYTE  [240][320][3];    typedef QVGAgray   BYTE [240][320];
typedef VGAcol    BYTE  [480][640][3];    typedef VGAgray    BYTE [480][640];
typedef CAM1MPcol BYTE [730][1296][3];    typedef CAM1MPgray BYTE [730][1296];
typedef CAMHDcol  BYTE[1080][1920][3];    typedef CAMHDgray  BYTE[1080][1920];
typedef CAM5MPcol BYTE[1944][2592][3];    typedef CAM5MPgray BYTE[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
```

```
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 dist. in [mm]; default 360° and 360 pots
int LIDARSet(int range, int tilt, int points);  // range [1..360Â°], 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] to pos 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 *linSspeed, 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 1..16 to in/out/In pull-up/Jn 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 RADIOReceive(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* function library.
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)
```

Thomas Bräunl, Remi Keat, Marcus Pham, 1996-2024

CHAPTER 3: Hardware Description Table

The Hardware Description Table or HDT is a text file, stored at path:

eyebot/bin/hdt.txt

with several alternative HDT files available at

eyebot/bin/hdt/

The HDT file has a number of settings, which are mostly self-explanatory. Its basic function is to list all the actuators and sensors that are connected to the EyeBot IO-Board and to calibrate their settings. The HDT also allows to specify the controller's / robot's name, the default display setting, and the default WLAN settings.

Calibration is achieved through the use of lookup tables for motors, servos, and PSD sensors. In these cases, the raw value form the sensor (PSD) is used as a table index, which then reveals the true distance value in mm. Likewise for user motor and servo drive commands, which will be translated through a table lookup into calibrated raw values for the motor/servo controller.

Listed below is the standard version of the HDT file. For specific robots, it needs to be adapted to match the new hardware accordingly.

```
# EYEBOT Name
EYEBOT EyeBot-Standard

# MOTOR Number | TableName
MOTOR 1 Motor_Table
MOTOR 2 Motor_Table
MOTOR 3 Motor_Table
MOTOR 4 Motor_Table

# Servo Number | Low | High |TableName
SERVO 1 0 255 Servo_Table
SERVO 2 0 255 Servo_Table
SERVO 3 0 255 Servo_Table
SERVO 4 0 255 Servo_Table

# ENCODER Number | Clicks per meter
ENCODER 1 3820
ENCODER 2 3820
ENCODER 3 3820
ENCODER 4 3820

# PSD NUMER | TableName
PSD 1 PSD_TableA
PSD 2 PSD_TableA
PSD 3 PSD_TableA
PSD 4 PSD_TableA
PSD 5 PSD_TableA
PSD 6 PSD_TableA

# IRTV Name | Type | Length | tog_mask | inv_mask | mode | bufsize | delay
IRTV "IRTV0" 0 4 0 0 0 4 20

# IRPARAMS Enable | Code | Delay
IRPARAMS 1 786 5

# WIFI Default(0)/Slave(1)/Custom(2)
WIFI 0

# HOTSPOT Network_Name | Password
#HOTSPOT rob rasp
```

```
# SLAVE Network_Name | Password
SLAVE WAMBOT Magic2010

# RoBIOS On(1)/Off(0) | FontSize
ROBIOS 1 10

# USB Number | DeviceName
USB 1 EyeBot
USB 2 GPS

# DISPLAY LCD(0)/HDMI(1) | Fullscreen Off/On (0/1) | Rotation (0/1) | Autorefresh (0/1)
# Note: Change of rotation requires 2 reboots
LCD 0 0 0 0

# RPI RaspPi_Ver(1/2/3)
RPI 3

# DRIVE Wheel distance (90 or 140mm) | Max Motor Speed | Motor1/left dir. (0 = c/w) |
Motor2/right dir. (1=clockwise)
DRIVE 140 262 1 0

#DEMOPATH | Path
DEMOPATH /home/pi/eyebot/demo

#SOFTWAREPATH | Path
SOFTWAREPATH /home/pi/usr

# VOMEGA | Vv | Tv | Vw | Tw (Note if 0,0,0,0 will turn off)
VOMEGA 70 30 70 10

# -------------------- TABLES (Optional) ------------------
# Motor Linearisation Table 101 values: 0 .. 100
TABLE Motor_Table
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
100
END TABLE

# Servo Linearisation Table 256 values: 0 .. 255
TABLE Servo_Table
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
100 101 102 103 104 105 106 107 108 109 110 111      112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191      192 193 194 195 196 197 198 199
200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219
220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
END TABLE

# PSD Sensor Linearisation Table 128 values:
TABLE PSD_TableA
80 80 80 80 80 80 80 80
80 80 80 80 80 80 80 80 80 80 80 77 75 72 70 68 65 61 59 56
54 53 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 33 32
30 29 28 28 28 27 27 26 25 25 25 24 24 23 23 22 22 21 21 21
20 20 19 19 19 19 18 18 17 17 17 16 16 16 16 15 15 15 15 15
14 14 14 14 14 13 13 13 13 13 13 12 12 12 12 11 11 11 11 11
11 11 11 11 11 11 11 11 11 10 10 10 10 10 10 10  9  9  9  9
END TABLE
```

Chapter 4: EyeBot IO-Board V8

This guide provides the user how to use the EyeBot I/O-Board, which plugs directly into a Raspberry Pi controller. It provides a variety of inputs and outputs, as well as sensors for robot control.

Features

- 2 motor driver outputs with voltage supply pins and encoder feedback
- 4 servo outputs (software PWM)
- LCD Display output via digital I/O pins
- 4 PSD sockets (Position Sensitive Devices, positioned at front, back, left, right)
- Hardware on/off switch
- Screw-type power connection
- Mounting point for camera holder (3D-printed)

Design

- The board is designed to be pin-compatible and mounting hole compatible with a Raspberry Pi.
- It takes 7.4V input and generates the 5V supply required by the Raspberry Pi.
- Quadrature encoders only count in one direction; this is compensated in software.

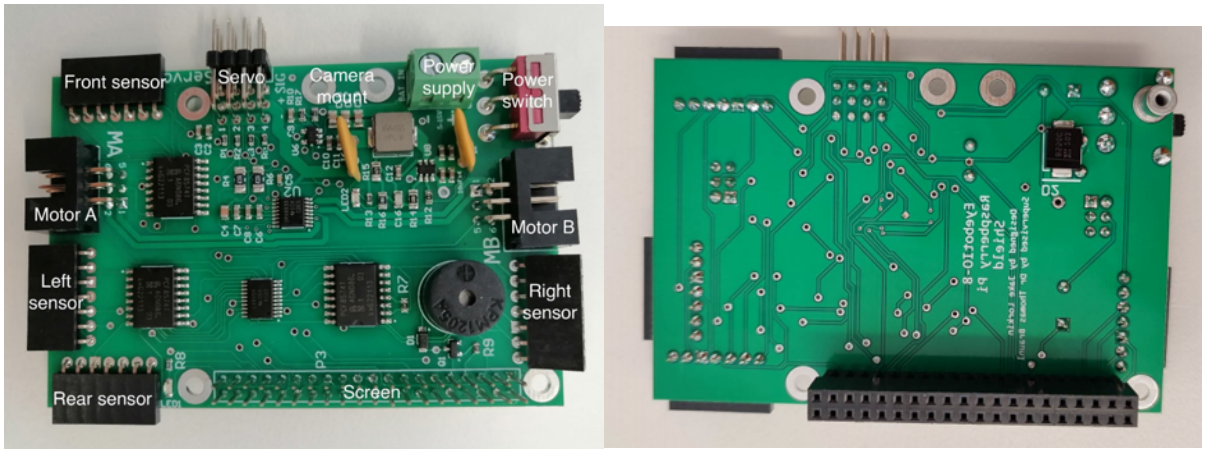


Figure 4.1: Connections on EyeBot-8 board top and bottom

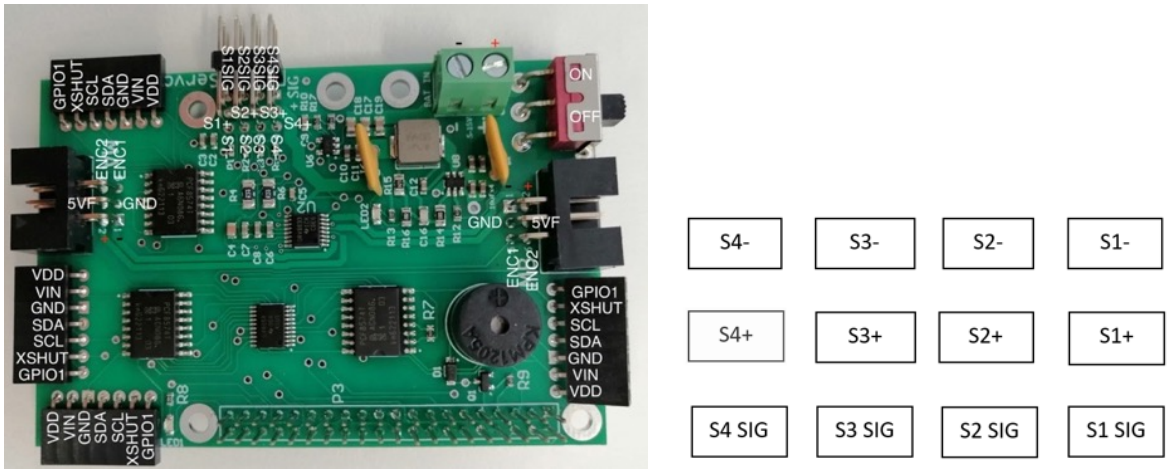


Figure 4.2: Full pin-outs and servo connector pins

Assembly

1. Apply insulation tape on top of USB and Ethernet ports of the Raspberry Pi to insulate it.
2. Insert 8mm hex stand offs for the bottom, as shown in the red circles below.
3. Screw hex standoffs on the top of the raspberry pi board (12mm for the top right corner and 18mm for the rest)
4. Join Raspberry Pi to EyeBot8 IO-shield until they click together
5. Screw 15mm hex stand offs on top of the IO board as shown below.
6. Attach LCD display onto the IO board ensuring that correct alignment of pins as shown below.

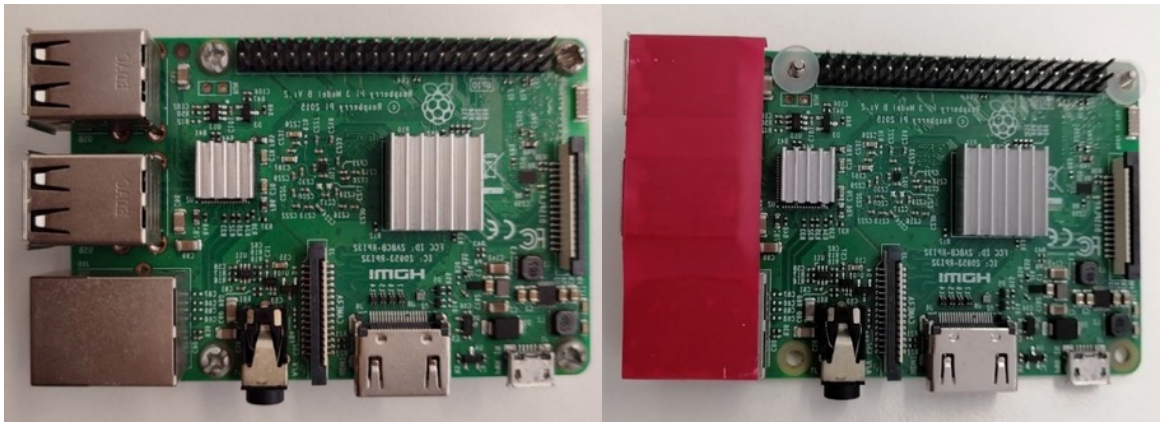


Figure 4.3 Raspberry Pi without and with insulation tape

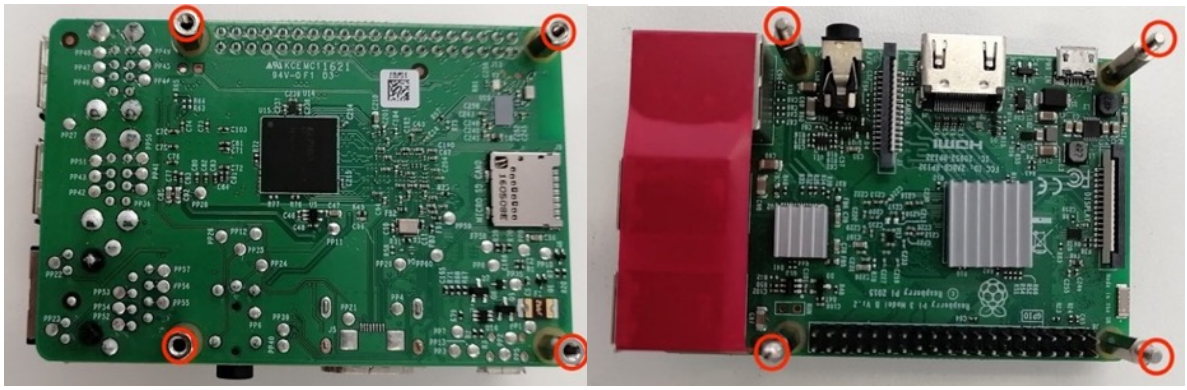


Figure 4.4 Raspberry pi with 8mm hex standoff on back and with hex-stand-off on top

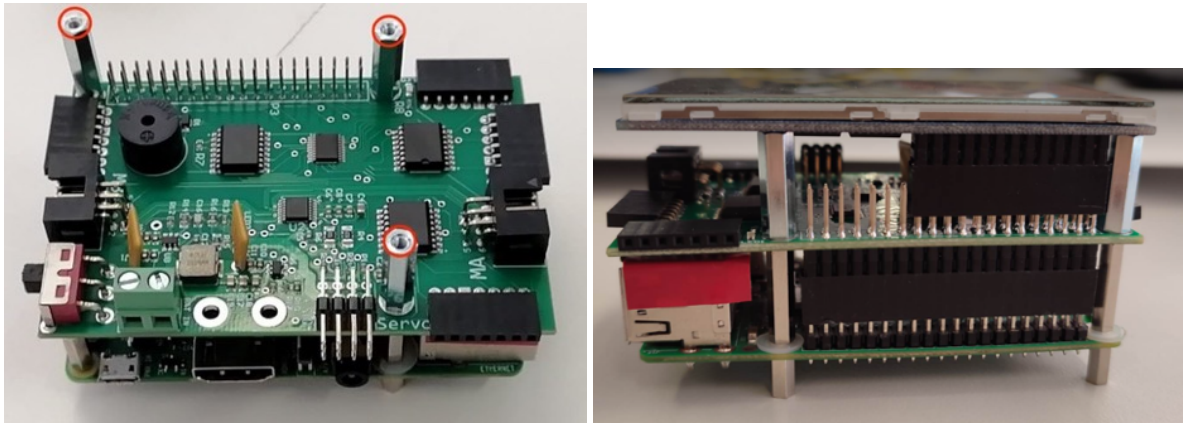


Figure 4.5 Raspberry pi attached to IO board with 15mm hex standoffs on top and with display attached

4.1 Connecting Power

Make sure the power switch is in off-position before starting.

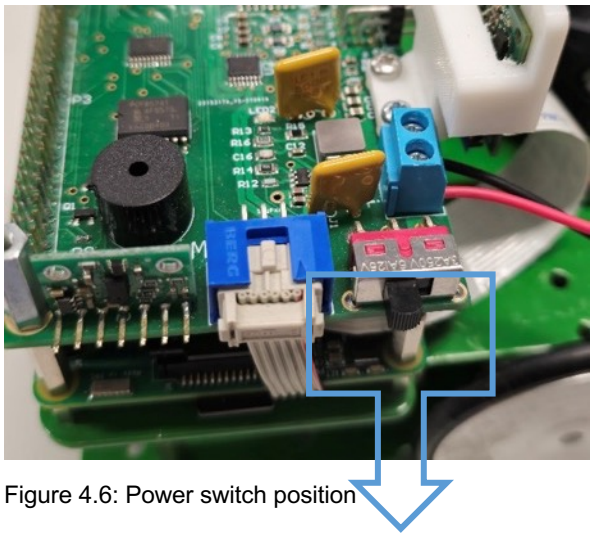


Figure 4.6: Power switch position

The EyeBot-8 board can accept 6–15V DC power. We suggest to use 7.4 V, e.g. from two 18650-Protected batteries in series. When the voltage level drops below 7V, the power outputs of the IO-Board can no longer supply the required voltage, resulting in poor driving performance.

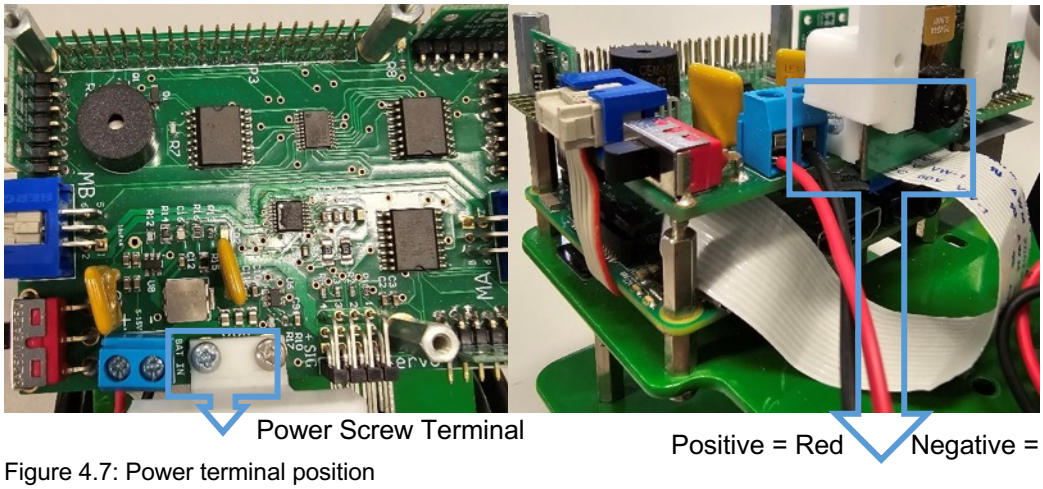


Figure 4.7: Power terminal position

To connect power to the screw terminal:

- Unscrew (anti-clockwise rotation) the two screws to open the jaws of the terminal.
- Connect the positive and negative wires from the power supply to the corresponding positive and negative terminals on the screw terminal, ensure the bare wire is properly inserted into the terminal. See Figure for terminal labels.
- Tighten the two screws to close the jaws of the terminal, fixing the wires in place.
- Add the batteries to the battery holders.
- Turn the EyeBot on by sliding the switch into the “ON” position.

The best way to determine good power supply is to check the blue power LED

top of the EyeBot board or the red power LED on the Raspberry Pi. If the LEDs do not turn on, then check the battery supply and ensure that a solid connection between battery and terminals exists.

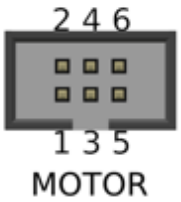
The EyeBot board is equipped with a regulated 5V DC voltage output suitable to power the Raspberry Pi. So only a single power battery connection is required to the IO-Board, which in turn will power the host controller (Raspberry Pi).

4.2 Connecting Motors

The IO-Board offers two motor connections (MA and MB) for left and right motor of a differential drive robot. Both connections feature encoders. Connecting the motor to the board can be done by pushing the 6-pin female header into the corresponding motor header on the IO-Board. The Figure below shows the correct connections of the left/right motors to the matching connectors.

The Board can generate 8 PWMs of 3.3V from 0 – 100% Duty Cycle with a resolution of 8 bits. These are connected to 4 full H-Bridges motor drivers.

Each motor connector also has the supply voltage for an encoder and two inputs for the pulses generated. For motors 1 and 2, the inputs are connected to a built-in quadrature decoder module of the MCU. The decoders output is a counter that increments or decrements according to the direction (clockwise or anti clockwise) of the motor. The decoder calculates direction by utilising the desired drive direction of the EyeBot, and therefore is unable to calculate changes in motor position when turned by hand.



Pin	Description
1	Motor A
2	Motor B
3	GND
4	5VF
5	ENC 1
6	ENC 2

Figure 4.8: EyeBot8 motor and encoder connectors

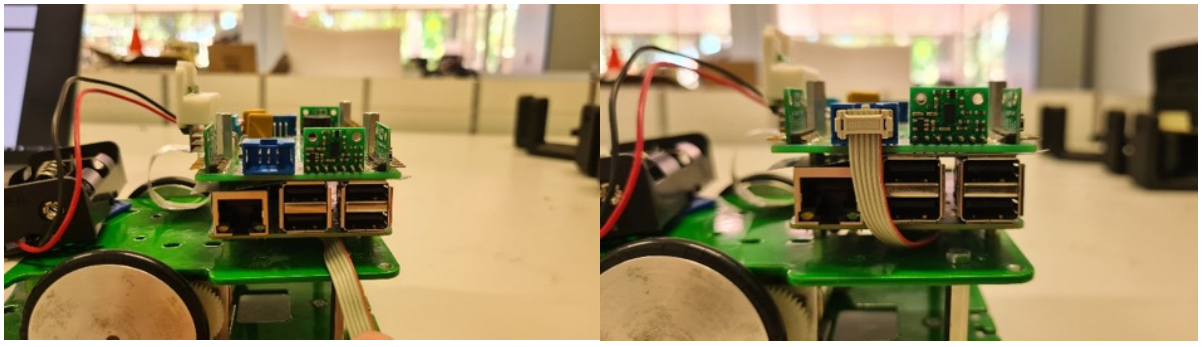


Figure 4.9: Motor port without (left) and with (right) motor connected.

4.3 Connecting PWM / Servo Output

The EyeBot IO-Board has four servo connections, facing towards the front of the EyeBot. Each servo connection consists of 3 pins (PWM, power and ground). The servo can be connected by inserting the female headers from the sensor onto the male connector on the IO-board. The signal wire should be placed on the bottom pin, with positive in the center and negative on the top pin. Ensure the wiring order is correct before powering up the EyeBot.

Each servo output generates an 8 bits PWM in software on the Raspberry Pi at 50Hz frequency with a configurable range around 1ms to 2ms of duty cycle.

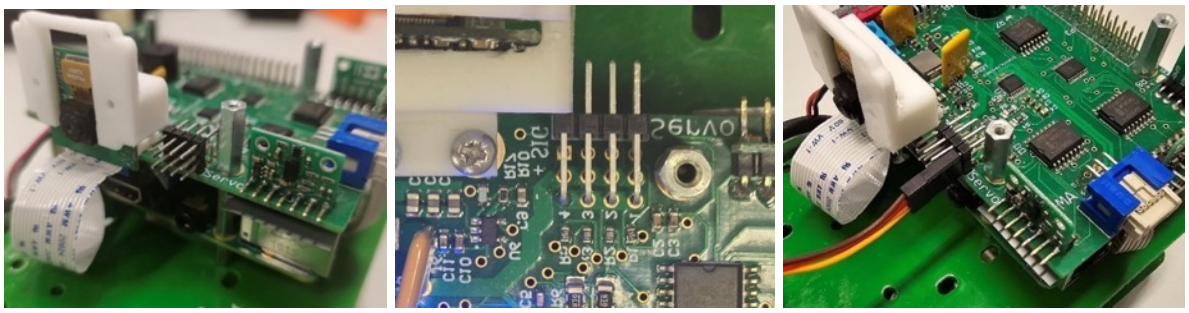


Figure 4.10: Camera mount (white 3D) and servo connector front (top left) and above (top right). connected Servo (bottom) with Brown = Ground, Red = Power and Yellow = Signal

4.4 PSD Sensors

The EyeBot IO-Board has 4 PSD sensor connections, one for each side of the robot. Each PSD port has a 7-pin female header on the IO-Board, which houses the corresponding male connector on the PSD. The sensors to be used are VL53L0X time-of-flight distance sensor from Pololu:

https://www.pololu.com/product/2490

Connector and sensor can be seen in the Figure below.

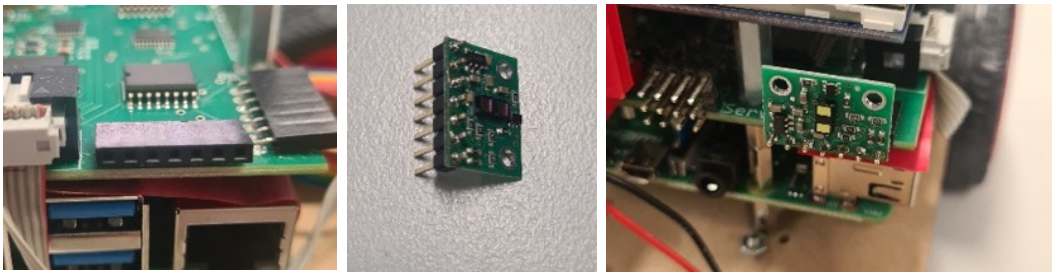


Figure 4.11: PSD Sensor socket (left). PSD Sensor (middle). connected PSD Sensor (right)

4.5 PID Control Software

Each motor has a PID speed controller given the speed set point in encoder ticks per second. The parameters Kp, Ki and Kd can be configured through the API.

```
r_mot  =        r_old[i] + Kp[i]*((pid_error-e_old[i]))
                + Ki[i]*(pid_error+e_old[i])/2
                + Kd[i]*(pid_error - 2*e_old[i] + e_old2[i]);
```

Additionally, a linking value is used to provide straighter driving. This works by adding a value to each sides r_mot depending on the difference between the two wheels velocities. This is only used when no angular velocity is desired, as once turning is introduced the PID control is better able to maintain the desired route.

```
r_mot_right  +=  Klink*(e_right - e_left);
r_mot_left   +=  Klink*(e_right - e_left);
```

4.6 LCD Screen

The LCD screen provides the capability for an EyeBot graphical user interface. To connect the LCD screen, ensure the correct IO pins are connected, as misplacing the screen on the wrong pins will break it. The female connector of the LCD screen is to be placed to the far left of the IO-Board, when looking from the front; the position is wrong if the LCD screen does not line up well with the IO-Board. Figure 6 provides the correct connection layout of the LCD screen.

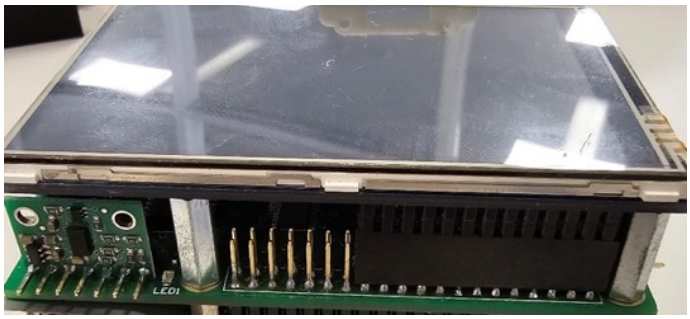


Figure 4.12. LCD Screen Connected to IO-Board

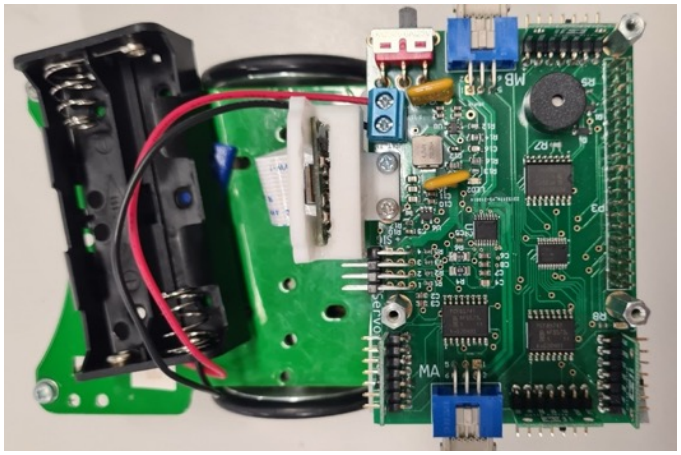


Figure 4.13: EyeBot-8 board mounted on robot without LCD screen

CHAPTER 5: Robot Simulation

In addition to the physical controller and robot design, we have also developed a matching robot simulation system, called "EyeSim". This system runs natively on either Windows, MacOS or Linux and is a great supplementary tool for lab preparation in teaching, as well as in simulation, e.g. when running an application with hundreds of robots or when repeating the same task thousands of times, e.g. for AI applications such as NN or GA.

More information and downloadable file and user manuals for EyeSim can be installed from:
http://roblab.org/eyesim/
http://roblab.org/eyesim/ftp/

See the demo video at: **http://roblab.org/eyesim/videos/EyeSim.mp4**

5.1 Simulation Environment

The simulation environment allows several robots to interact with each other and numerous objects in a common driving area. Environments can be loaded as either World files (Saphira format) or Maze files (Micromouse format). Robots can be selected from a list of "pre-built" EyeBot robots or self-designed robot files.

The programming API for the simulation system is identical to the one for the real EyeBot robots, so every robot application program can run unchanged on either EyeSim or the real EyeBot robots.

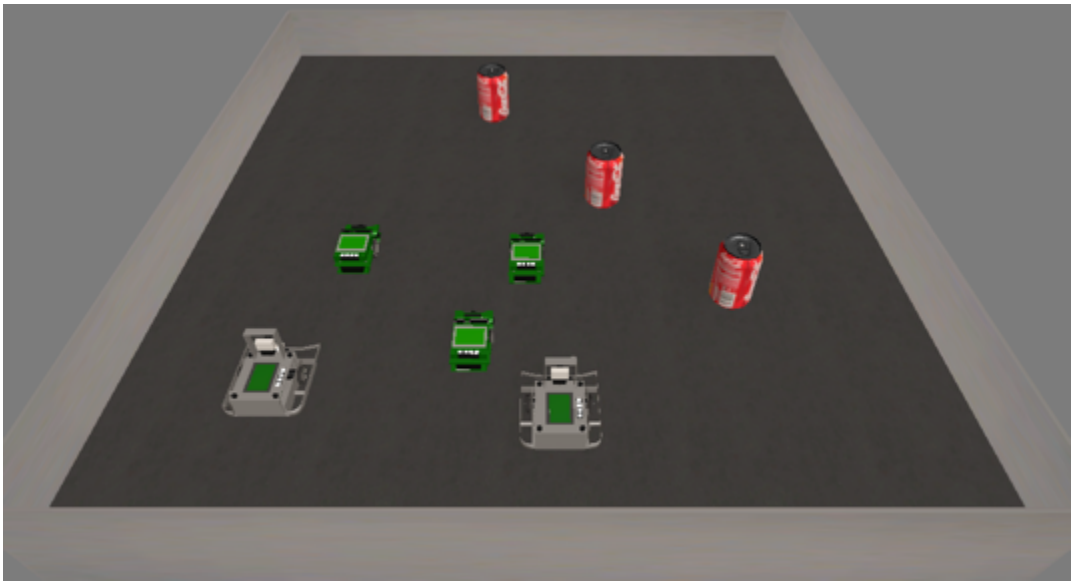


Figure 5.1: Several robots interacting with objects

5.2 Simulation Applications

A shell-script "gccsim" is being provided to compile robot programs into binaries that can be loaded in the simulation system. The standard procedure is to start the simulator by double-clicking its icon, then either loading a prepared world (or maze) scene, or manually inserting robots and objects.

Then the compiled robot program (default suffix ".x") can be started, e.g. with; **./myprog.x**
This program will then automatically connect to the simulation program and exchange actuator/sensor data with in in a similar way that a robot application program would do on a controller linked to a real robot.

Figure 5.2 shows an example application that uses color histograms to detect a colored object in the scene.

Figure 5.2: Robot closing in on Coke can; robot display with superimposed color histogram

CHAPTER 6: Building a Robot

We will now use the controller and IO-Board together with motors and sensors to build a complete driving robot. There are many different design choices, including the conversion of a model car, but we usually use two independent drive motors in a “differential drive” configuration, which lets the robot drive forwards, backwards and turn on the spot.

There is also a huge variety and significant price differences in motors and sensors. The components listed below are therefore only one possible choice.

Components

- Raspberry Pi controller with SD memory card
- Raspberry Pi camera with cable
- LCD Touch Screen Module Waveshare 3.5 inch
- EyeBot-8 IO-Board
- 1–4 Pololu distance sensors, VL53L0X Time-of-Flight Distance Sensor
- 2 Motors at 6V with encapsulated encoders and gears
  e.g. Faulhaber-2230 or DX-431572
- 2 Wheels
- 1 Slider for front surface contact
- Optional servos, e.g. for camera movement or actuators (e.g. ball kicker, etc.)
- Rechargeable battery, 7.4V, preferable 18650-Protected
- Mounting plate and brackets out of aluminum or 3D-printed
- Camera holder, 3D-printed

6.1 Robot Design

EyeBot8-I/O board, Raspberry Pi and LCD are mounted as a stack on top of the robot platform. All motors (DC motors, servos, etc.) and sensor (encoders, infrared PSD distance sensors, etc.) simply plug into the EyeBot-8 controller, no soldering is required.

The following images show different versions of putting together Raspberry Pi controller and the EyeBot-8 IO-Board with motors and sensors to build a complete robot.

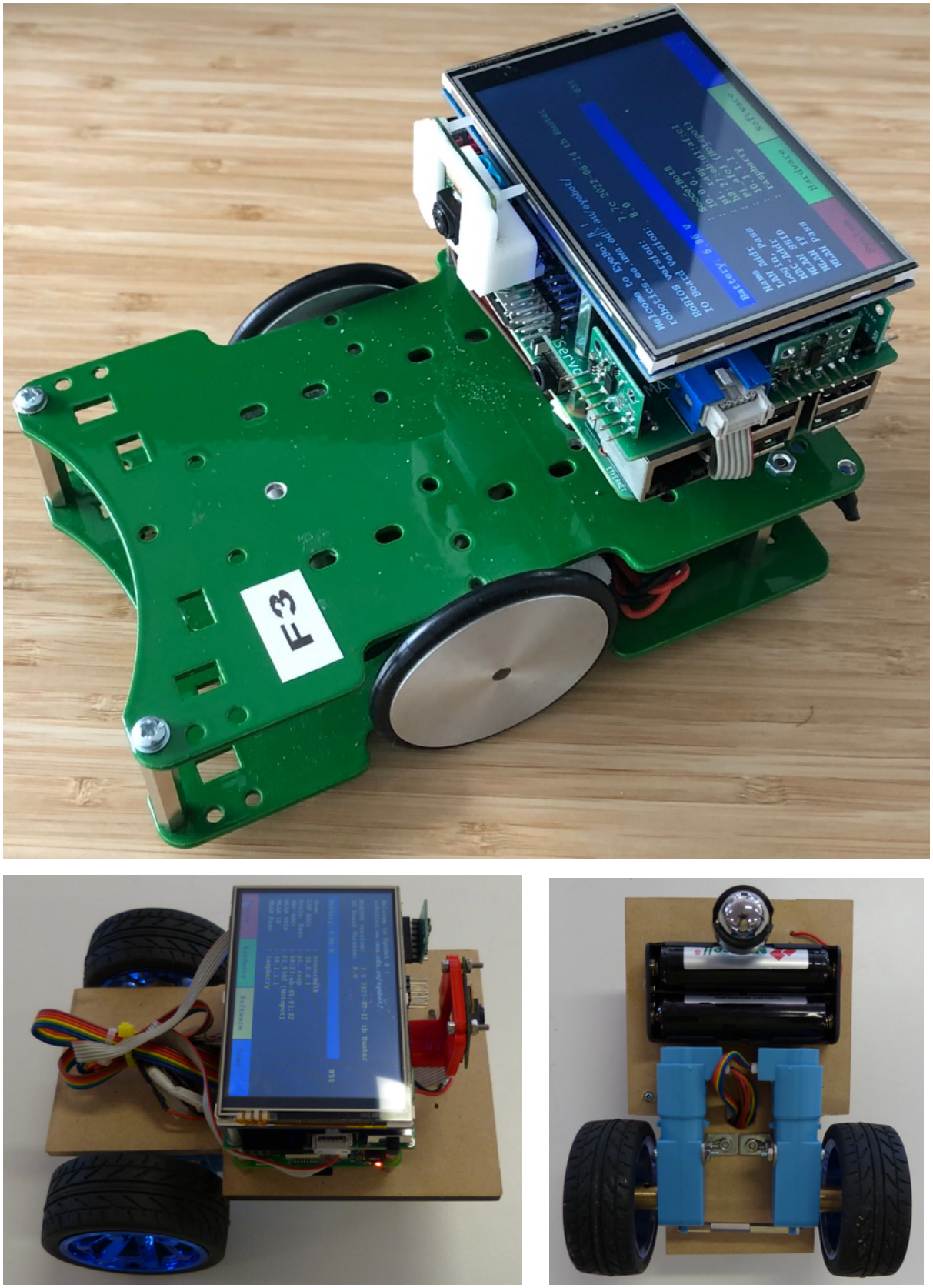


Figure 6.1: SoccerBot and EyeCart mobile robots with EyeBot-8 board

For more details, please consult these books:

- **Embedded Robotics**, Springer-Verlag 2022
  https://www.amazon.com/Embedded-Robotics-Autonomous-Vehicles-Raspberry/dp/9811608032
  https://www.springerprofessional.de/embedded-robotics/20247878

- **Mobile Robot Programming**, Springer-Verlag 2023
  https://www.amazon.com/Mobile-Robot-Programming-Adventures-Python/dp/3031327969
  https://www.springerprofessional.de/en/mobile-robot-programming/25929512