

# EyeBot 7 User Guide

Thomas Bräunl, Marcus Pham, Franco Hidalgo, Remi Keat, Hendra Wahyu  
August 29, 2018

EyeBot 7 is the 2017 version of the EyeBot embedded controller for robotics applications. It is now based on a Raspberry Pi board with optional LCD display, linked via USB to the EyeBot7-IO board, which has hardware and software drivers for motors and digital or analog sensors. The board runs Raspian Linux with the RoBIOS user interface software on top and provides an extensive robotics library that allows the simple design of robot application programs in C using the RoBIOS API.

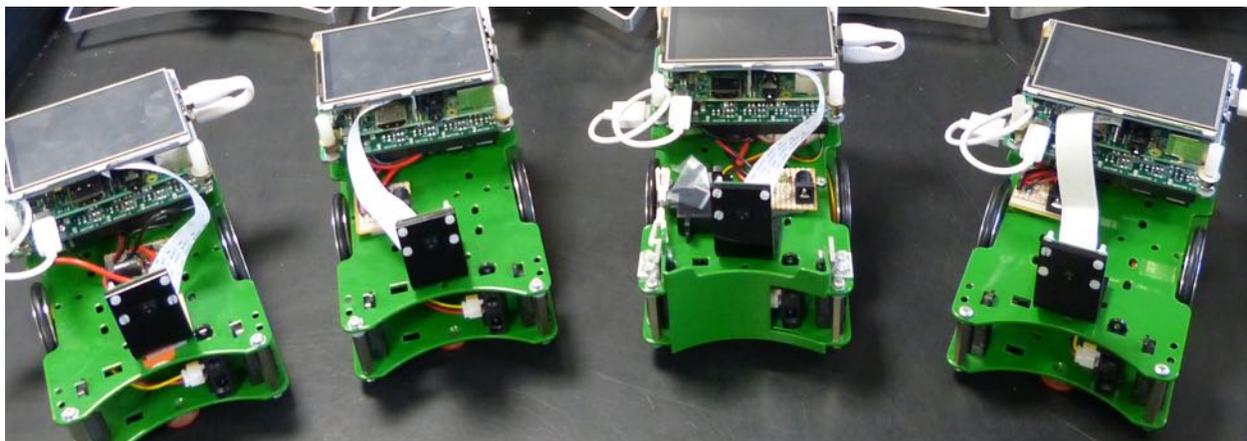
## Link

<http://robotics.ee.uwa.edu.au/eyebot7/>

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

## Contents

1. EyeBot User Interface
2. RoBIOS Library
3. Hardware Description Table
4. EyeBot IO-Board
5. Building a Robot



# 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:  
<http://robotics.ee.uwa.edu.au/rasp/images-pi3/>

More frequent updates of the EyeBot software package are available from:  
<http://robotics.ee.uwa.edu.au/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:

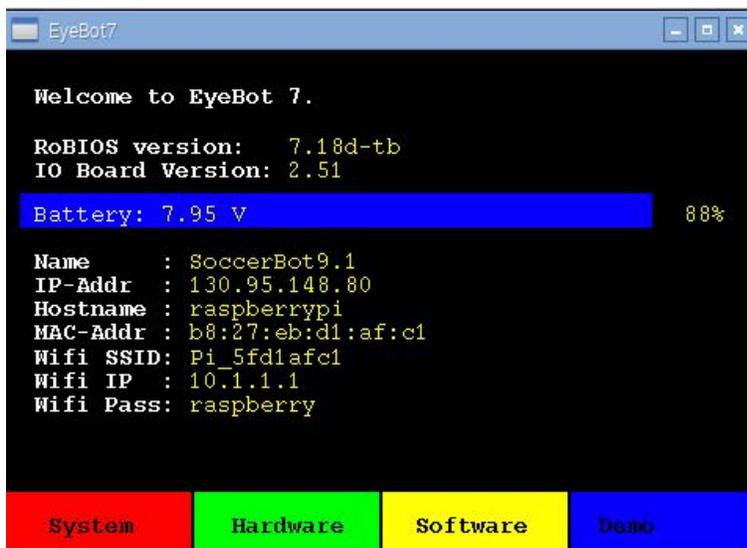


Figure 1.1: EyeBot Start Screen

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 and 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 and password  
raspberry

## 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.

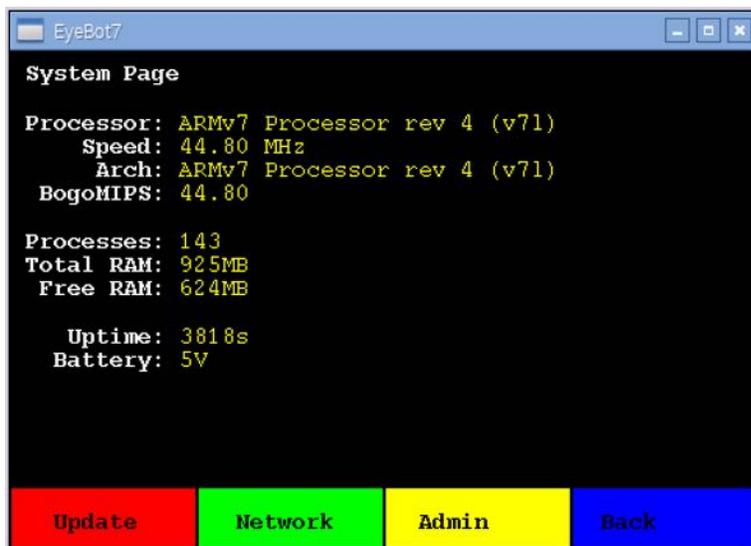


Figure 1.2: System page

### 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".

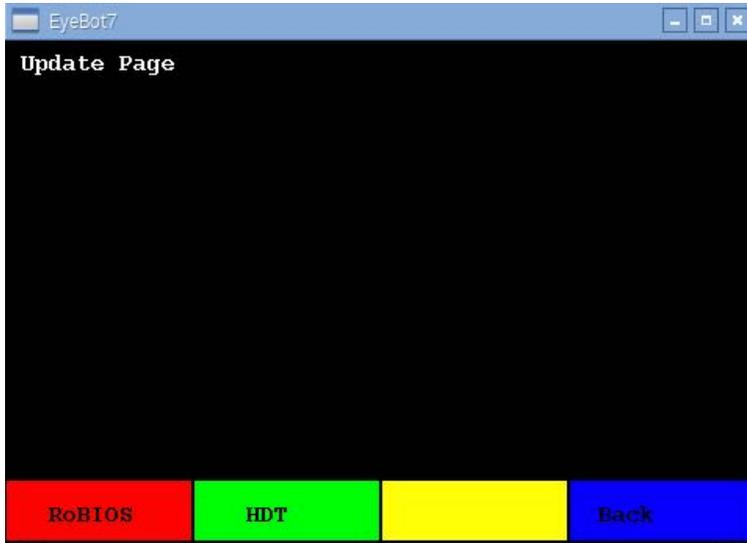


Figure 1.3: Update page

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 "raspberrry". 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.

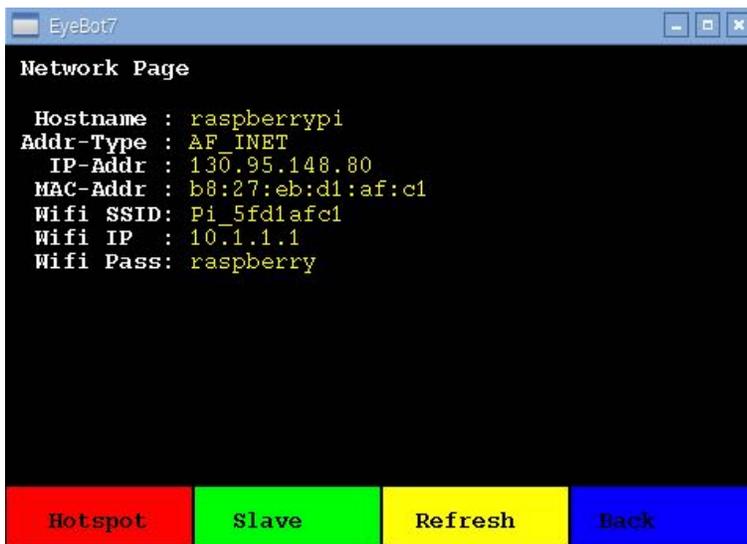


Figure 1.4: Network page

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. The default login name is “pi”, password is “raspberrry”.

This network menu then 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.

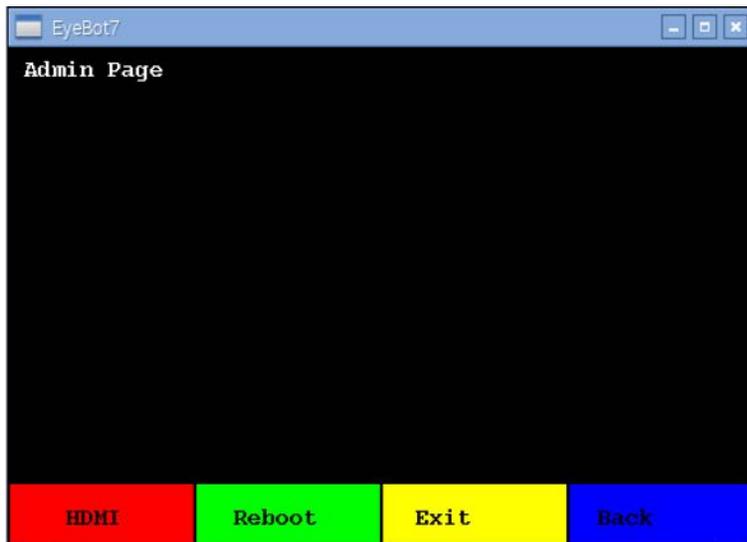


Figure 1.5: Admin page

## 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.

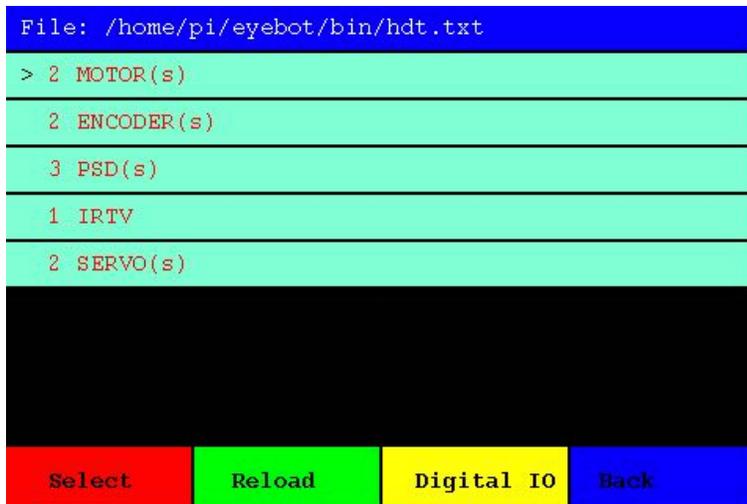


Figure 1.6: HDT page

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.

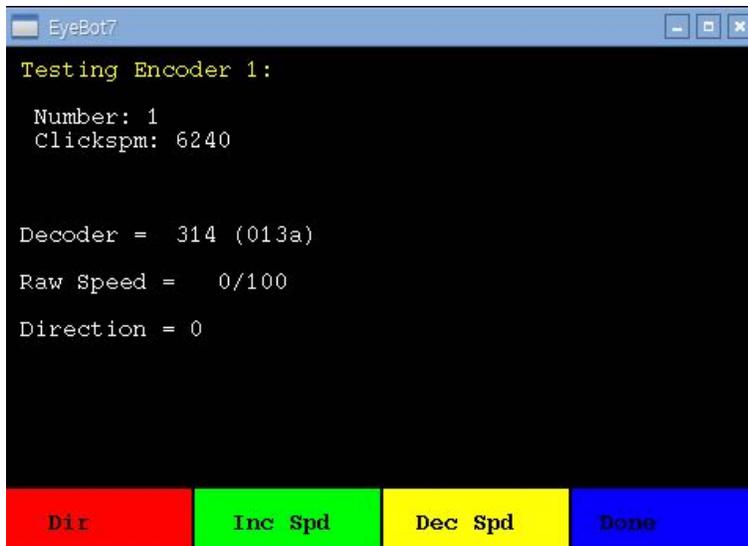


Figure 1.8: Encoder testing page (HDT)

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

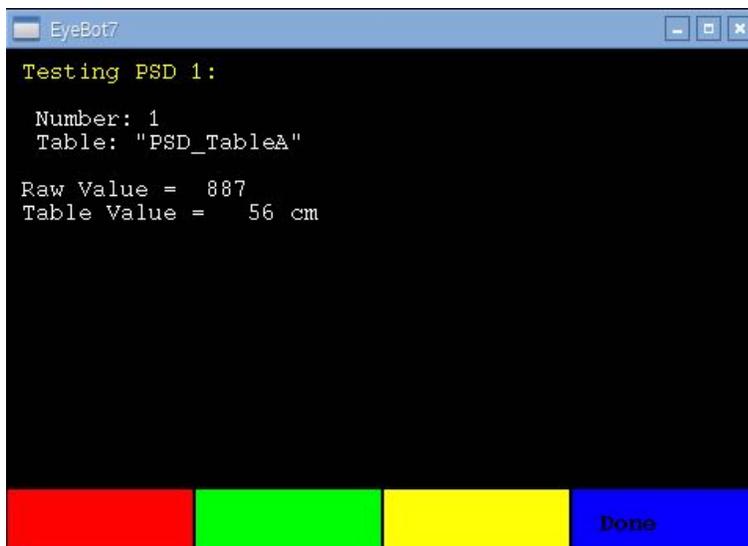


Figure 1.9: PSD testing page (HDT)

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

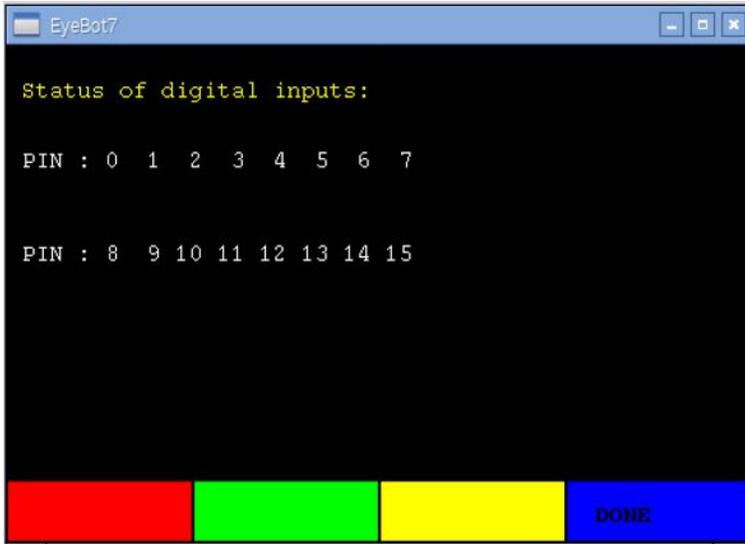


Figure 1.10: Digital IO testing page (HDT)

### 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.

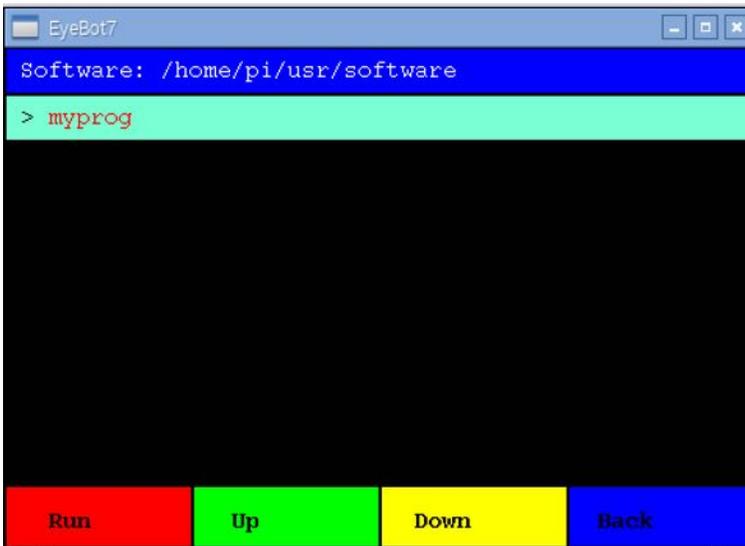


Figure 1.11: User program select page

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).

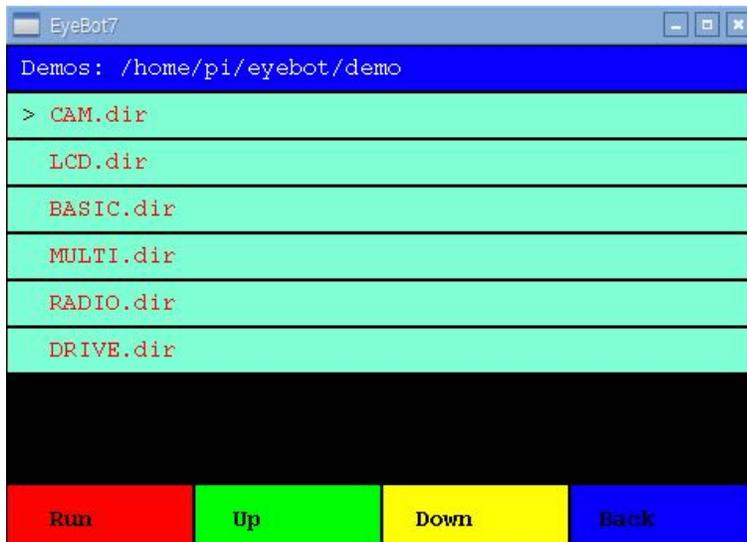


Figure 1.12: Demo program directory select page

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

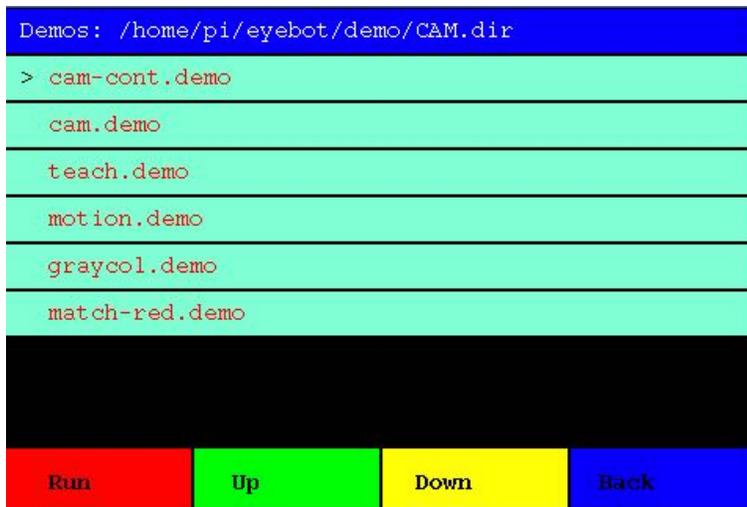


Figure 1.13: Executable demo program select page

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

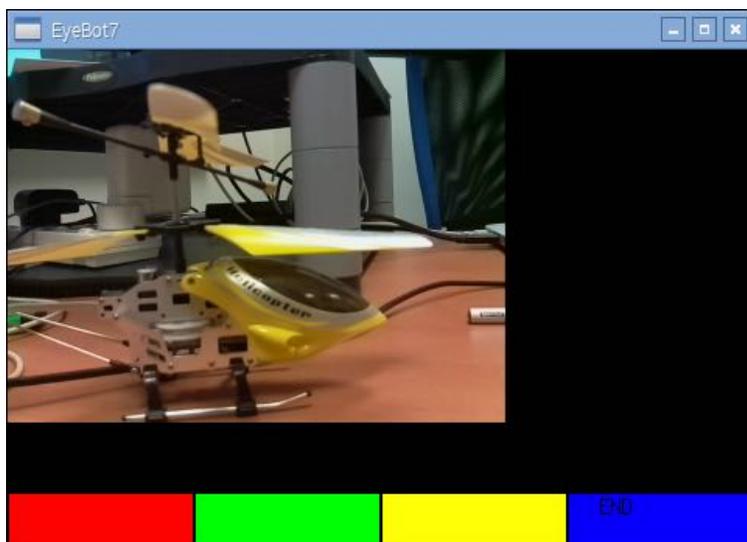


Figure 1.14: Demo program run

# 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:

<http://robotics.ee.uwa.edu.au/eyebot7/Robios7.html>

Version 7.0, June 2016-- RoBIOS is the operating system for the EyeBot controller. The following libraries are available for programming the EyeBot controller in C or C++. 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 (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 (us 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
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 QVGAgray 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 const.
int IPReadFile(char *filename, BYTE* img); // Read PNM file, fill/crop if req.
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 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
void IPPRGB2HSI(BYTE r, BYTE g, BYTE b, BYTE* h, BYTE* s, BYTE* i); // PIXEL: Convert RGB to HSI
```

---

## 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, interface number as in HDT
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, ret. number of chars received
bool SERCheck(int interface); // Non-blocking check if character is waiting
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]; returns 0 if OK

```

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] pos. 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 [degr/s]
int VWCurve(int dist, int angle, int lin_speed); // Drive Curve, dist [mm], angle [deg], l.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-block. check if drive is finished (1) or not (0)
int VWWait(void); // Suspend thread until drive operation has finished
int VWStalled(void); // Returns number of stalled motor [1..2], 3 if both
All VW functions return 0 if OK and 1 if error (e.g. destination unreachable)
```

---

## 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 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 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, BYTE* buf); // Send string (Null terminated) to ID destination
int RADIOReceive(int *id_no, BYTE* buf, int size); // Read bytes from ID source, returns #chars rec.
int RADIOCheck(void); // Non-blocking check whether message is waiting (0/1)
int RADIOStatus(int IDlist[]); // Returns number of robots (excl. self) and net list
int RADIORelease(void); // Terminate radio communication
ID numbers match last byte of robots' IP addresses.
```

---

## 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 SIMGetPose(int *x, int *y, int *phi);
void SIMSetPose(int x, int y, int phi);
void SIMGetObject(int id, int *x, int *y, int *phi);
void SIMSetObject(int id, int x, int y, int phi);
```

---

*Thomas Bräunl, Remi Keat, Marcus Pham, 1996-2017*

# 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 from 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

---

This guideline is to provide a user with details on how to use the EyeBot I/O USB expansion board version 2.63, short called EyeBot IO-Board. It offers a simple and economical way to extend the capabilities of a main controller, such as the Raspberry Pi, Beagle Board, etc., and adds a number of additional digital and analog I/O lines as well as hardware and software device drivers to interface with low-level hardware. The EyeBot IO-Board allows various types of inputs (digital, analog and encoders) and outputs (digital, RC, PWM and power PWM). The board features a powerful, high performance 8/16-bit AVR ATxmega128A1U microcontroller and uses USART micro USB port for serial communication with the host controller.

The EyeBot 7 board is usually directly linked to a Raspberry Pi or similar controller, but for testing you can directly link it to a PC or Mac via USB. The connection commands are:

- Windows: Putty to COMx (find out COM port from device manager)
- Mac: screen /dev/cu.usbmodem1421 (or similar /dev port)

## Features

---

- High performance and low power 8/16-bit Atmel XMEGA128 A1U Microcontroller
- 14 PWM/servo outputs (8 are available to use, 6 are reserved for other use).
- 4 H-Bridge Motor drivers with configurable voltage supply pins and encoder feedback
- 16 digital I/O pins
- 8 Analog Input pins (ADC IN channels) and 6 additional analog pins (reserved for PSD)
- USART micro USB port
- 2 x 5VDC power supply outputs (1 USB 2.0 and 1 screw-type terminal)

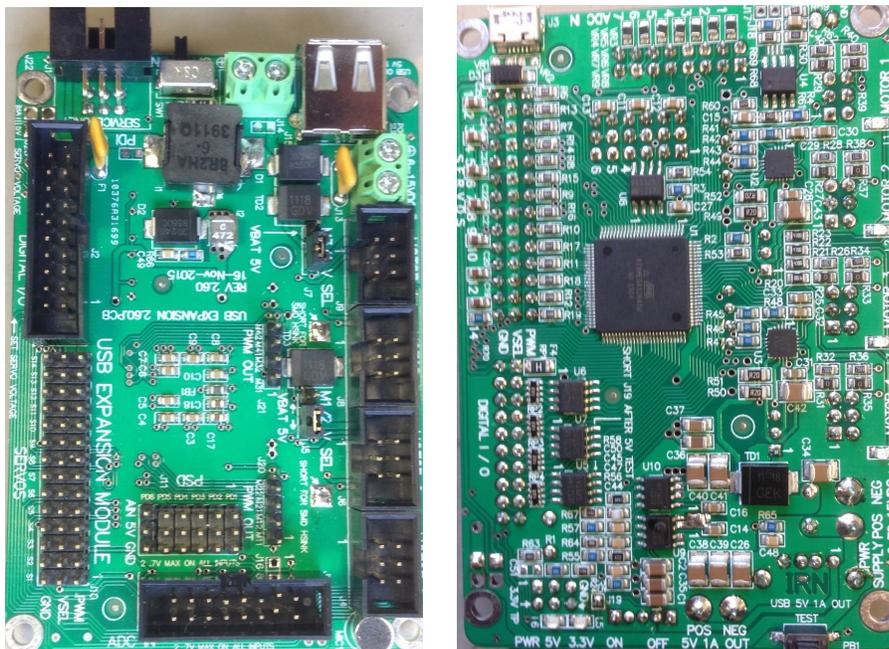


Figure 4.1: EyeBot7 IO-Board

# 4.1. Hardware

## Connecting Power

The USB expansion module can accept 6 – 15VDC power supply. Choosing the correct battery involves many factors such as voltage and amps requirements. When the voltage is below 6 VDC, the microcontroller will work but all 5VDC outputs will be disabled. We therefore recommend to use a 7.2V battery of sufficient capacity. Below are steps to wire and connect a battery to USB Expansion Module:

- Before starting, please ensure that slider switch is in “OFF” position (see picture below)

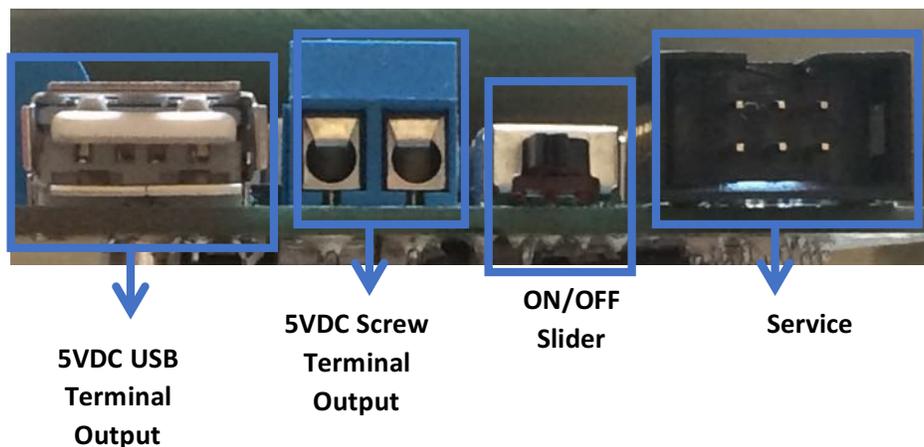


Figure 4.2: Power Connections

- The IO-Board is powered by a screw-type terminal, which is located between MOTOR 4 socket and the USB port.

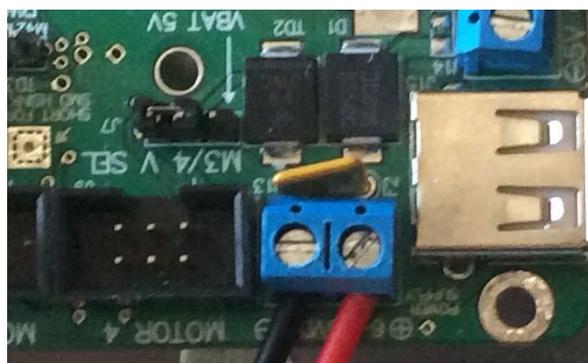


Figure 4.3: Power Supply Terminal Input

- Unscrew (anticlockwise) the screw from the terminal.
- Connect red wire to positive terminal (“+”) and black wire to negative terminal (“-“) by inserting bare copper to each hole in the terminal. (For different color-code wire, please refers to manufacturer guide or contact battery supplier).
- Screw in clockwise direction so the metal plate will hold and clamp both wires down until it is tight.
- Connect the connector to the battery.
- Turn the slider switch to “ON” position.

It is also possible to use the micro USB connector to supply 5VDC to USB expansion on, but this will only energize the microcontroller but, no the regulated 5V outputs.

The best way to determine good power supply is to check the LEDs indicator located at the backside of USB expansion module next to switch slider. Two LEDs (marked as 3.3V and 5V) must both turn “ON” (solid bright).

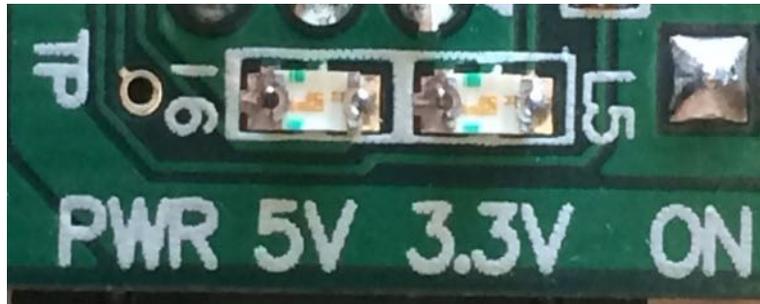


Figure 4.4: Power LED Indicator

If this does not occur with the batteries supply connected, please check the battery supply and ensure that a solid connection between battery and terminals exists. When powering from USB 2.0, you may need to unplug and re-plug the connector to your PC.

**NOTE: There are 2 important things to remember when checking the power supply: polarity and voltage level.**

## Power Supply Output

The IO-Board is equipped with a regulated 5VDC voltage output suitable to power up an embedded host computers, such as the Raspberry Pi (see Figure 4.1). So only a single power battery connection is required to the IO-Board, which in turn will power the host controller (Raspberry Pi). There are two ways to provide the 5VDC power output:

- **USB Type A:**  
This is commonly used to connect with a Raspberry Pi or similar controllers. There is no power button on the RPi, so it will start working instantly once power is connected and can only be turned off again by physically removing the power cable or turning off the IO-Board.

- Screw-type terminal:  
The screw-type terminal setup in IO-Board is similar with terminal power supply (see procedure 1 on how to install a cable). This can be used to power up additional peripherals, if required.

## Connecting Motors

The IO-Board offers 4 motor connectors (M1 to M4, each with dual encoders), available to be used to run small motors up to 12VDC. To connect the motor to the board:

- Connect a 6-pin female header to any of the motor headers. These PCB-mount male pin headers are located next to the power supply voltage input (marked as “MOTOR x” while x indicates a number).

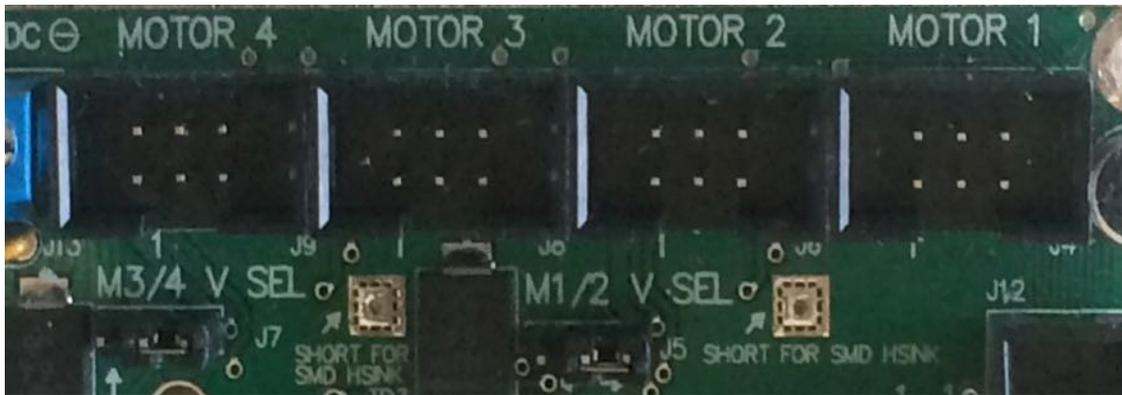


Figure 4.5: PCB Motor Ports and voltage select

## Selecting Motor Voltage Level

- Insert a shunt pin header plug to the voltage select 3 pin header in front of the motor sockets to select the power source to the motors between **5V** or directly to the battery (e.g. **7.2V**). See details in Figure 4.6.

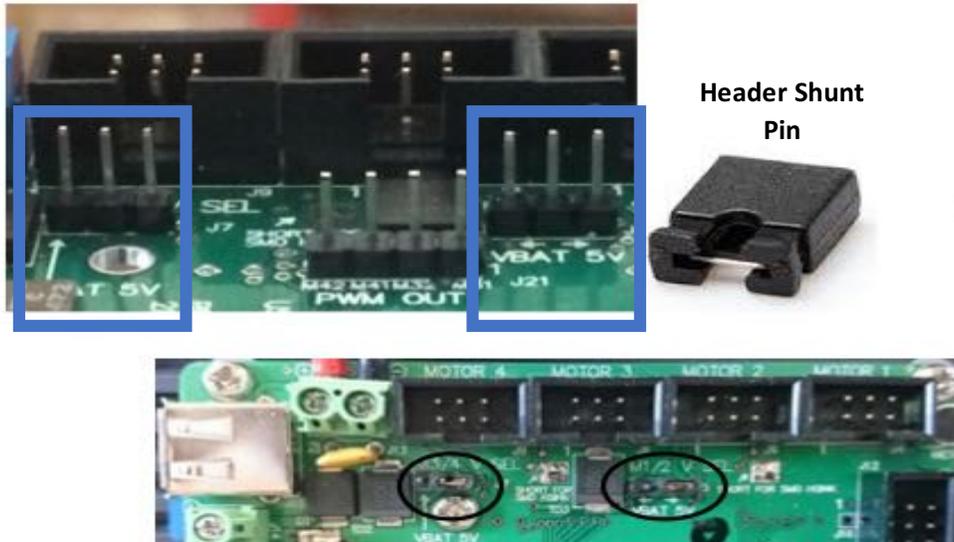


Figure 4.6: Shunt pin header on pin 1 and pin 2 for battery supply or to pin 2 and 3 for 5VDC supply.

## Connecting PWM / Servo Output

PWM / Servo pins are set up as output signal pins to control servo motors (Figure 7). It can drive 8 servos, each connector splits into 3 rows (PWM for signal output, VSel at 5V and GND). The S9 to S14 ports are reserved for other purposes by default, but can be used if not all motors and encoders are required. Each row is to be connected separately starting from “S1” (right hand side) and all the way to the left.

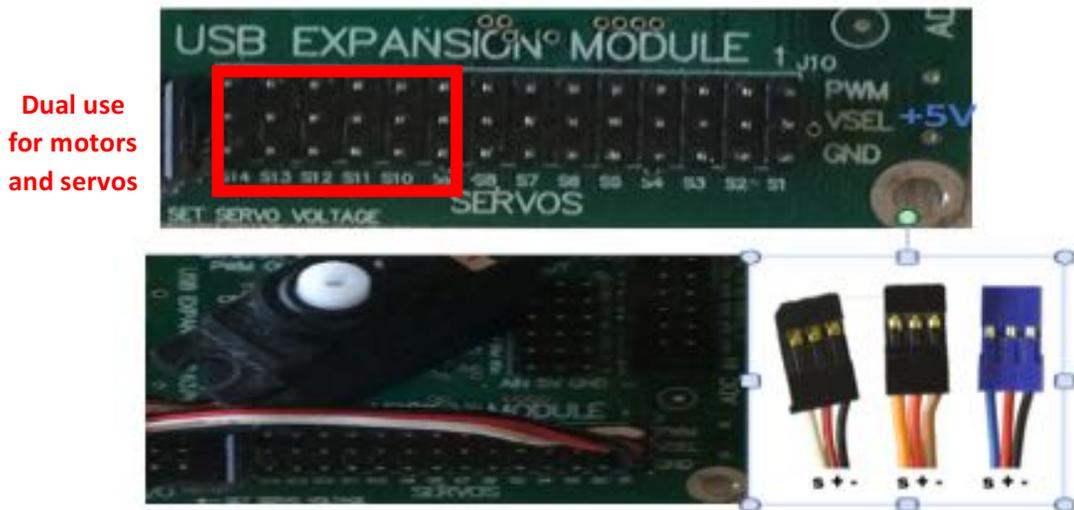


Figure 4.7: Servo ports include 3 rows (PWM, Vsel, GND)

**Note: The Servo Cable colors can vary by manufacturer and model, therefore you should use the lead markings as illustrated above. If color differs, please consult with manufacturer’s cable color code guide before starting connecting.**

## Connecting ADC / PSD Input

The IO-Board's Analog Input is located on the middle right of the board. The first pin header is ADC (Analog to Digital Input) which are housed within a socket and of 2 rows: bottoms are GND and tops are the analog signals (see Figure 7). Pin 1 is 3.3V and pin 2 is 5V. All top rows can be used as analog channels where the first channel starts from pin 3 (analog Channel 1) up to pin 15 (analog Channel 7).

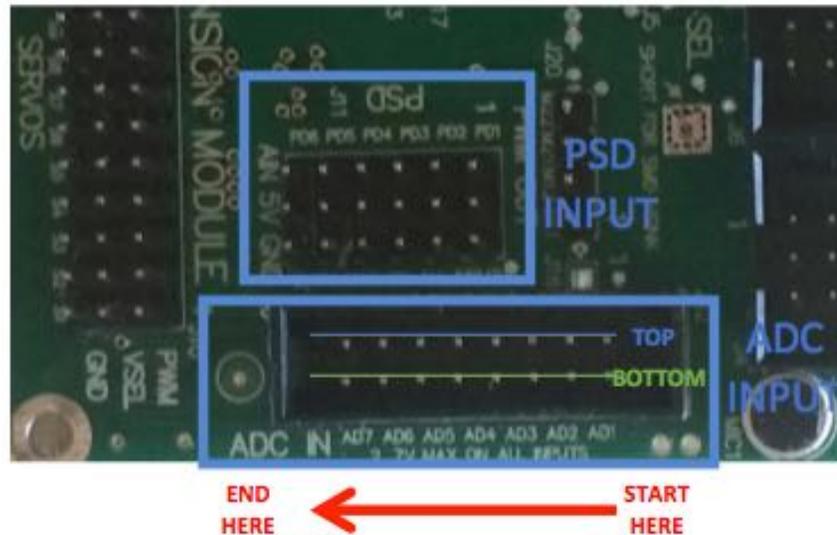


Figure 4.8: IOBoard 7 - Analog ADC Input pins and 6 PSD Input pins.

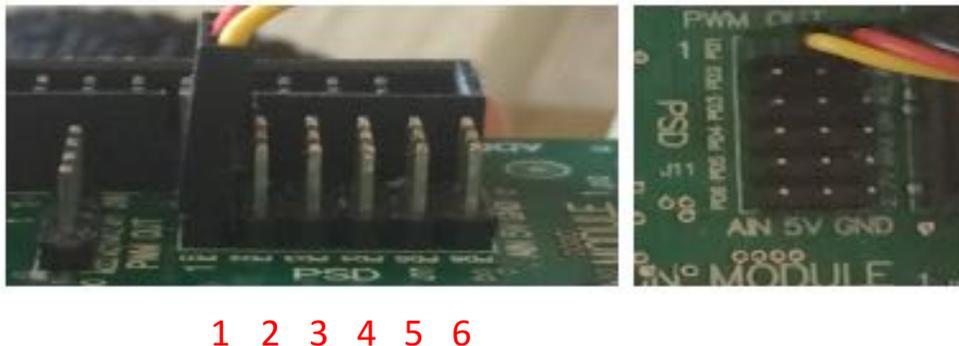


Figure 4.9: PSD connection from side and top view. Always start from PD1 when connecting PSD sensor

The second pin header is dedicated to to PSD sensors. It can connect to up to 6 PSD sensors (Figure 4.9). Similar to the servo connection, PSD input pins are separated into 3 rows (AIN for signal, 5V and GND). Connection starts from the right (PD1) to the left (PD6). **Note: If cable color differs, please check the Manufacturer's Guide.**

## Connecting Digital I/O Ports

The IO-Board has 16 configurable digital Input/Output pins. They are located next to the servo/output pin headers labeled as “Digital I/O”. It is a 20-pins port, fitted in 2 rows of 10 male 2.54mm headers. Pin numbers for the GPIO port start from the right and are split with the top row taking the odd-number and the bottom row with even number. Each pin can be configured as either an input or an output. The digital voltages are 0 and 3.3V and the port also contains pins for additional 5V (pin 2) or 3.3V (pin 1) supplies as well as ground (pin 19 and pin 20) (see Figure 4.10).



Figure 4.10: Digital I/O Port. These ports are designed so that a user can write and read data as a Byte.

## Communication

The IO-Board uses a Universal Synchronous Asynchronous Receiver Transmitter (USART) serial bus to provide a simple two wire serial interface. As the purpose of the module, USART is used to connect to a miniature - sized CPU such as Raspberry Pi or PC with a USB port. The IO-Board receives commands through this port to control the board and provide responses when queried. A full list of the commands is presented in Section 4.5.



Figure 4.11: Micro USB to communicate with a host controller (e.g. Raspübery Pi)

## 4.2. Software

### Mac OS / Linux System Administration

Most Linux distributions are open source (except for Mac OS), user friendly, and with a graphical user interface (GUI) that provides an easy way to perform common tasks. In Linux, the kernel is the heart of an operating system and handles the communication between the user and the IOBoard. Below are the procedures to actuate controls and peripherals, to read inputs or to configure the read / write value in Digital I/O (In this procedure steps, we are using Ubuntu 14.0.4):

- Turn on the power supply switch on the EyeBot IO-Board
- Plug the micro USB to a PC (Raspberry Pi, Laptop, Desktop, etc.)
- Open a new terminal
- On the terminal, type: **>> screen /dev/ttyACM0**
- Once you hit ↵, a new blank session terminal will be opened for you. You can start typing here to configure or simulate peripherals connected to the IO-Board according to the list in Section 4.5.
- Try typing: **h ↵** to display the command list available.

**Description:** /dev/ttyACM0 is a USB communication device (CDC) of sub-type “abstract control model” (ACM). For a complete reference of screen commands, go [here](#).

- **To exit** a screen command: Ctrl – A – k

### Windows

Windows does not come with a serial port terminal hence we will need to use a serial terminal such as Putty. Putty can be downloaded here (<http://www.putty.org/>). The steps to enable a connection to Raspberry Pi are:

- Identify the port number by going to:  
On the desktop. Right click on “This PC” → Properties.  
On System → Device manager (left menu)  
Under “Ports (COM & LPT) check for:  
**EyeBot\_M7(COMXX)** where XX is the PORT number  
**Note: If it is listed under unknown devices you need to install the driver for the board that can be found here:**  
[http://robotics.ee.uwa.edu.au/eyebot7/EyeBot-IO/EyeBot\\_M7.inf](http://robotics.ee.uwa.edu.au/eyebot7/EyeBot-IO/EyeBot_M7.inf)  
The same directory also contains the latest HEX (“binary”) files with the EyeBot-IO firmware.
- Open Putty:  
In “Category”, select: “Serial”
  - Type the corresponding Port number and leave the baudrate as default (57'600)
  - Click OpenA new terminal will appear where you can type the commands.
  - Try typing: **h ↵** to display the command list available.

## 4.3. IO-Command Examples

Once the board is connected and a serial terminal is opened you should be able to send commands and query the EyeBot IO-Board.

An easy way to test if the connection is alive is sending: `h ↵` to retrieve the list of commands accepted by the IO-Board. The following table shows different command sequences to familiarize with the IO-Board. If you are using the robot cars of the lab the board will be already connect to two motors, one servo and one PSD sensor.

Table 1: IO-Board command examples

Command	Description
<b>LED Test (LEDs on the IO-Board)</b>	
l 1 1	Turn LED 1 on
l 1 0	Turn LED 1 OFF
<b>Motor</b>	
m1 50	Turn motor 1 on a 50% Duty cycle power supply
m1 100	Turn motor 1 at maximum power
m1 0	Turn motor 1 off
<b>Encoder</b>	
e1	Return the value of a counter associated with the encoder on motor 1. Try rotating the motor clockwise and anticlockwise and repeating the command to observe the variation of this value.
<b>Servo</b>	
s1 128	Turn servo 1 to central position
s1 0	Turn servo 1 to anticlockwise limit
s1 256	Turn servo 1 to clockwise limit

Command	Description
<b>PSD</b>	
p1	Will return the digitalized PSD voltage value. This is a number between 0 and 4096 that represents a voltage between 0 and 5 volts, which reflects the distance between sensor and the nearest object. Try the sensor at different distances to observe a trend
<b>Motor with PID-Control</b>	
M1 50	The motor speed will be controlled by a PID at 50% of the maximum speed. Try braking the motor carefully with your finger to feel how the power of the motor is incremented by the controller to keep a constant velocity.
m1 0	Turn off the motor and PID control

## 4.4. IO Connections

The EyeBot M7 **I/O Board** was created to allow the Raspberry Pi or similar host controllers to interface with low-level hardware, such as motors, servos, and position sensing devices (PSDs). The I/O board also supplies power to the Raspberry Pi and LCD display module when powered from an external source greater than 6 volts. Alternatively, it can run directly from a 5V USB supply (micro USB connector).

### Hardware Design

The I/O board was designed around the Raspberry Pi and its display with respect to board size and the locations of mounting holes. An ATxmega128A1U is used to interface up to 4 motors with 8 encoders, 14 servos, 6 PSDs, 6 ADC inputs, 16 digital I/O pins, and 4 LEDs. The ATxmega communicates with a master controller over a USB connection which emulates a serial connection.

#### Board Overview:

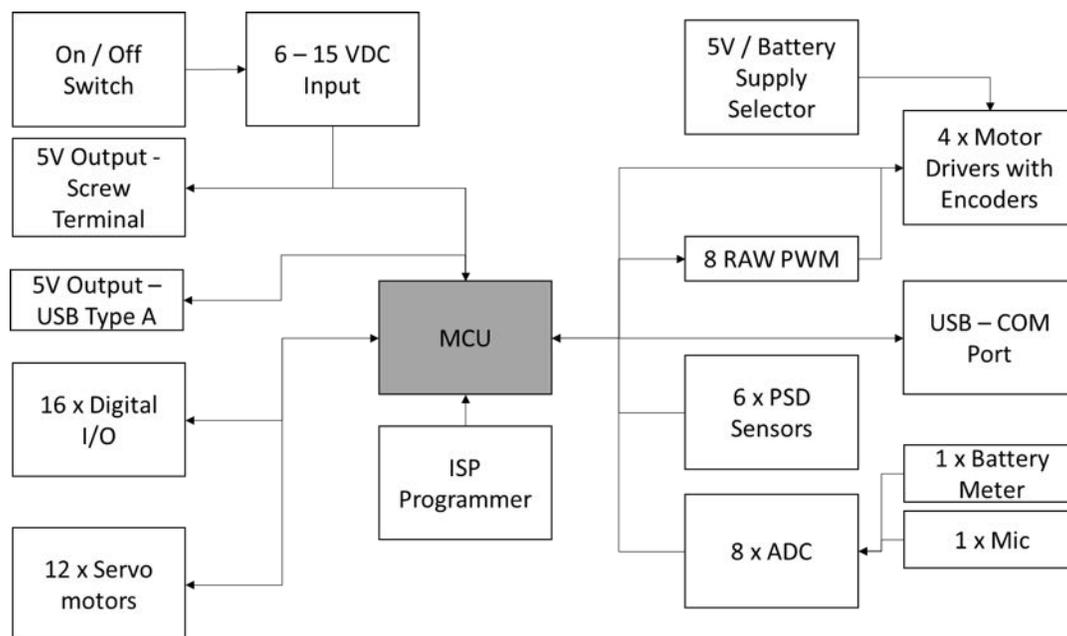


Figure 4.12: EyeBot7 IO-Board block diagram

#### Digital I/O:

There are 16 digital I/O pins that can be configurable, written or read independently or as a Byte. The digital voltages are 0 and 3.3 V and the port also contains pins for additional 5V or 3.3V supplies and ground. The board also contains 4 LEDs on one of the sides that can be controlled via API.

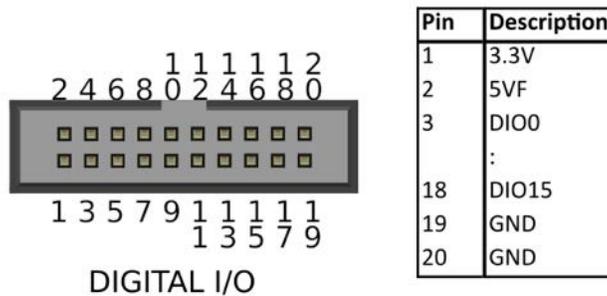


Figure 4.13: EyeBot7 digital IO

### Analog Inputs (ADC) and PSDs:

There are 8 ADC channels from which two are already wired to (1) the power supply, to work as a battery meter, and (2) the microphone, to have the option to react to sound. The other 6 channels are available on the board next to a ground rail to ease connections. The ADC works with a resolution of 16 bits for signals between 0 – 3.3 V. Additionally, there are 6 PSD channels with the same characteristics of the analog input, but with 3-pin headers (Signal – Vcc – GND) suitable for common PSD distance sensors.

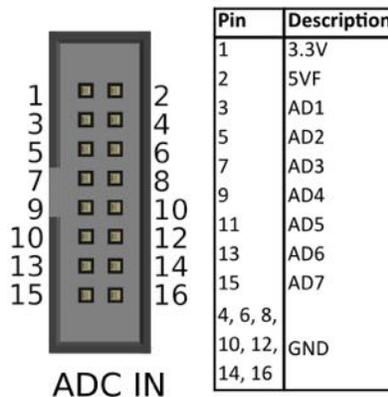


Figure 4.14: EyeBot7 Analog input

### Motors:

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. A jumper selects the power supply for the drivers switching between battery voltage (up to 9 V) or 5 V from the voltage regulator. The maximum current handled for each driver is 1 A.

Each motor connector also has the supply voltage for a dual-quadrature 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. Motors 3 and 4 decoders are software implemented and can detect pulses up to 10kHz. The decoders output is a counter that increments or decrements according to the direction (clockwise or anti clockwise) of the motor.

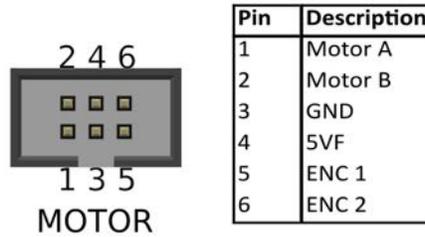


Figure 4.15: EyeBot7 motor and encoder connectors

### Servo motors:

There are 12 RC Servo motors connectors available on the board. Each of them generates 8 bits PWM, 50Hz frequency with a configurable range around 1ms to 2ms of duty cycle.

### PID Controller:

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]);
```

### VW Controller:

There is a navigation controller based on velocity and angular velocity in order to drive a two motors differential vehicle.

## Hardware Connections for Wheeled Vehicle

Component	IOBoard
Motor Left	M1
Motor Right	M2
PSD Left	PSD1
PSD Center	PSD2
PSD Right	PSD3
Magnet	DIO0
Camera Servo	Servo1

Figure 4.18: EyeBot7 connection order for robot vehicle

# Software

---

Application software is written using Atmel Studio and utilizes the Atmel Software Framework (ASF), which provides access to libraries covering nearly every aspect of the CPU. Atmel Studio provides the necessary tools to organize a project, and build it into an "ATMEL Solution". This build process also generates the an "*application.hex*" file, which we will need to keep track of when uploading using AVRDUDE. In order to set a machine up for effective programming, the following items are required:

## Drivers for EyeBot-IO board, Dragon Programmer, and USB upload mode

These can be installed by using the **libusb** driver install wizard, located at <http://sourceforge.net/projects/libusb-win32/files/libusb-win32-releases/> (choose the latest version)

## Programmer Hardware:

### AVR Dragon

An AVR Dragon board is required to install software on the EyeBot-IO board:  
<http://www.atmel.com/tools/avrdragon.aspx>

This board can be purchased from various sources, e.g.

Element14 <http://au.element14.com/microchip/atavrdragon/in-system-debugger-programmer/dp/1455088>

RS <http://au.rs-online.com/web/p/products/6962566/>

Digikey <https://www.digikey.com.au/products/en?mpart=ATAVRDRAGON>

## Programmer Software:

### AVRDUDE

Can be located from <http://download.savannah.gnu.org/releases/avrdude/> (latest version)

### Flip

Can be located at <http://www.atmel.com/tools/flip.aspx>

### Atmel Studio (optional)

Can be downloaded from <http://www.atmel.com/microsite/atmel-studio/>

### Bootloader

Standard *bootloader.hex* file from site <http://robotics.ee.uwa.edu.au/eyebot7/>

# Software Upload

---

There are two program memory sections to program in the IOBoard:

- The bootloader section which is small program executed when the board starts (needed to allow the USB programming)
- The application section where the main program is stored.

The bootloader can only be written by an ISP/PDI programmer (and other advanced programming modes) but, it can be programmed through serial communication such as USB.

The application can be uploaded through ISP/PDI programmers and serial communication (USB). For the last option to work, the bootloader must be already installed and the microcontroller has to boot in a DFU mode (Device Firmware Update).

Besides the program memory there are some internal fuses in the microcontroller which set a series of parameters like the frequency of the CPU, writing protection and internal modules configuration, etc. These fuses are programmed only once and can only be modified through ISP Programming.

In the laboratory, the AVR Dragon Board is available for ISP programming. The software to manage the programming is AVRDUDE.

The programs the we are going to flash to the memories are:

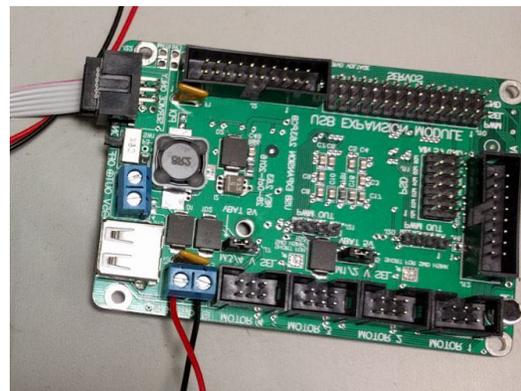
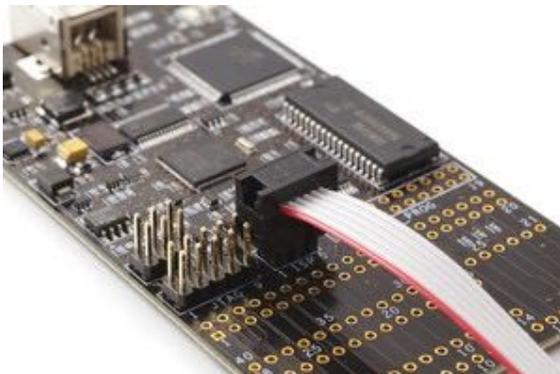
- *USBExpansionBootloader.hex* - For the bootloader
- *USBExpansionBoard.hex* – For the application program

Hex files and drivers for windows available in:

<http://robotics.ee.uwa.edu.au/eyebot7/EyeBot-IO/>

### Setting a board from scratch – Dragon Board (Bootloader + Program)

In this part both memory sections will be written using the AVR Dragon Board. A 6-way ribbon cable with 3x2 connectors on the ends is used. The Dragon Board has 6 pins labeled as “ISP” next to “JTAG” that are connected to the “Service Only” port on the IOBoard. The pin numbers must match to perform the uploading (pin 1 on the IOBoard connected to pin 1 on the Programmer, etc.)



The IOBoard needs an external power supply, connecting the micro USB port to the computer is the easiest way to obtained it. Make sure the switch is ON and you see at least one light on, on the IOBoard.

Connect the Dragon Board to your PC. Open a “Command prompt” or cmd in Windows or a terminal in MAC/Linux. Use the following commands to upload the codes. The \*.hex files have

to be in the same folder as AVRDUDE to work, otherwise you will have to indicate the full path for each file.

1. Bootloader:  
**avrdude -c dragon\_pdi -p x128a1u -D -U boot:w:USBExpanderBootloader.hex**
2. Application program:  
**avrdude -c dragon\_pdi -p x128a1u -U application:w:USBExpansionBoard.hex**

#### **Notes/Troubleshoots:**

To use the Dragon Programmer with the AVRDUDE it should be installed as an “Atmel USB Device” ◇ AVRDragon. If it’s installed as another device:

1. Install libusb drivers
  - Run Inf-wizard.exe
  - Select AVRDRAGON when devices are listed
  - At the end, it creates a temporal folder with the drivers
2. Uninstall the previous drivers for AVRDragon In Windows (tick the box for “remove software”)
3. After a few seconds the device should appear as a “Atmel USB Device” if not:
  - Update the driver manually with files created
  - Go back to 1

#### **Uploading application data via USB (Application program only):**

1. Connect the IOBoard to your computer though the micro USB port.
2. Turn the IOBoard on
3. Make sure the board appears as a serial communication board. COM port, for Windows.
4. Set the board to ‘upload mode’ by sending the letter ‘u’ + enter (carriage return) to the board. You can use a serial terminal such as Putty or Hyperterminal. After that, the serial communication will be lost until rebooting since the board is now in ‘upload mode’ or DFU mode.
5. The board should be recognized as an “Atmel USB Device”, if not, install the required drivers.
6. Use the command line:  
**avrdude -c flip2 -p x128a1u -D -U application:w:USBExpansionBoard.hex**
7. Turn the board off and back on.

#### **Notes/Troubleshoots:**

- When you upload an application via Dragon using AVRStudio, it usually wipes the bootloader, so you will need to set the board via AVRDUDE and Dragon Programmer.
- If the program is uploaded but is not running when resetting, make sure the fuses are set as follows:
  - FUSEBYTE0 = 0xFF
  - FUSEBYTE1 = 0x00
  - FUSEBYTE2 = 0xFF
  - FUSEBYTE4 = 0xFF
  - FUSEBYTE5 = 0xF7

### 'AVRDUDE write fuse4:'

```
avrdude -c dragon_pdi -p x128a1u -U fuse4:w:0xff:m
```

### 'AVRDUDE terminal mode:'

```
avrdude -c dragon_pdi -p x128a1u -t
```

```
d fuse4 --> reads the value
```

```
w fuse4 0 0xff --> writes the value 0xff
```

### AVRDUDE parameters:

**-c <programmer>** tells AVRDUDE which programmer to use. Flip2 for DFU and dragon\_pdi for other usage

**-p <partid>** tells AVRDUDE which part is to be programmed, in our case we only used the x128a1u chip

**-Dvvv** tells AVRDUDE to not overwrite the wrong memory and print verbose output

**-U <fileflags>** tells AVRDUDE which file to use and what to do with it (read/write)

### Raspberry Pi - IOBoard low-level terminal:

In order to test the IOBoard in low level through the Raspberry Pi:

1. On Mac or Linux:  
screen /dev/ttyACM0  
Then you can use the list of commands on:  
<http://robotics.ee.uwa.edu.au/eyebot7/IO7.html>
2. On Windows:  
Run "putty" to establish a serial connection to the board  
Then you can use the list of commands on:  
<http://robotics.ee.uwa.edu.au/eyebot7/IO7.html>

## 4.5. EyeBot7 IO Interface Commands

NOTE: <void> means that the command has no arguments

All commands can be sent in ASCII as listed or in binary code (command+0x80).

Command	Description	Parameters
h	help: print list of commands including descriptions and parameters of all command interfaces	h <void>
O	Output: text output in verbose or short mode parameter is [0 ...1] Ex. O 0 (turn off verbose mode)	O <verbose>
E	demo: drives a robot along a path and print relevant sensor data	E <void>
m	motor: set motor speed (uncontrolled) from 1 to 4 motors and speed value from -100 to +100, where 0 is at stop position. Ex: >> m 1 50, >> m 2 -100	m <motor [1..4]> <speed>
M	motor: set motor speed (controlled) from 1 to 4 motors and speed value is defined as ticks per 1/100 second (signed short integer) from -128 to +127 to enable PID controller. 0 is at stop position. (see back page for ex) Ex: >> m 1 50, >> m 2 -100	M <motor [1..4]> <ticks>
d	PID: set PID controller speed from 1 to 4 motors and PID value is defined as byte [0 ... 255]. Ex: >> p 2 50 100 255	d <motor [1..4]> <p> <i> <d>
s	servo: set servo [1 ... 8] and angle of motion as byte [0 ... 255] which translated to [0 ... 180] Ex: >> s 2 200	s <servo [1..8]> <position>
S	servo: set servo [1 ... 8] and constrict lower and upper angle of motion as byte [0 ... 255] which translated to [0 ... 180] Ex: >> S 2 50 200	s <servo> <lower> <top> lower: 0 – 255 top: 0 – 255
p	PSD: read PSD distance sensor [1 ... 6]. Value returns as 2 bytes, analog	p <sensor [1..6]>
e	Encoder: print encoder value from each motor pin socket header [1 ... 4]. Returns 4 bytes in binary.	e <motor [1..4]>
v	Version: print a version number Binary Return: 4 bytes [characters]	v <void>
V	Print supply voltage level Binary Return: 2 bytes [Volt/100]	V <void>
i	Input: read ONLY a single input pin from Digital I/O port [3 ... 18]	i <port>
I	Input: read all input ports from Digital I/O port (start from pin 3 to pin 18) which is translated as [0 ... 15] and return 2 bytes	I <void>

Command	Description	Parameters
o	Output: set a single pin in Digital I/O port as an input [0] or output [1] Ex: >> o 7 1, o 7 0	o <port [0..15]> <bit>
c	Config: set input or output in Digital I/O port configuration (start from pin 3 to pin 18) which is translated as [0 ... 15] State: <ol style="list-style-type: none"> <li>1. i – input</li> <li>2. o – output</li> <li>3. I – in pull up</li> <li>4. J – in pull down</li> </ol> By default pin 3 to 9 [0 ... 7] are set as an input pull up while pin 10 to 18 [8 ... 15] are set as an output  Ex: >> c 0 o, c 7 J	C <port [0..15]> <state>
a	Analog: read analog input channel (start from pin 3 to pin 15 [0 ... 7], ONLY top rows / odd number pins, while bottom rows, except pin 2, are reserved for Ground connection).  a 0 or pin 1 is wired to power supply 3.3VDC and a 1 or pin 2 is wired to 5VDC which works as battery meter. a 8 is microphone input. Returns 2 bytes [volt/100]	a <port [0..15]>
r	Record: record analog input sequence  <ul style="list-style-type: none"> <li>• Port [0..8] (0 is supply volt., 8 is microphone)</li> <li>• Port [9..14] for sensors PSD1 .. PSD6</li> <li>• Iterations [1..2048] at 1kHz</li> <li>•</li> </ul> Ex.: r 8 1000 record for 1sec microphone input	r <port> <iterations>
R	Read: read recorded analog values transfers previously recorded byte values  Ex.: R	R <void>
l	Led (small L): set led status [1 ... 4] ON (1) and OFF (0) independent to any input / output.	l <led [1..4]> <bit[0..1]>
t	Time: print 100Hz timer counter which return 4 bytes	t <void>
T	Timer: set timer frequency [1 ... 100] in 10Hz Ex: >> T 100 (equivalent to 1000Hz)	T <timer [1..100]>
u	Upload: set I/O module into USB-upload mode for software upgrade	u <void>

Command	Description	Parameters
w	<p>Drive function: initialize v-omega driving to use 2 motors, M1 and M2, at the same time with speed in ticks per second (unsigned integer).</p> <ol style="list-style-type: none"> <li>1. Speed: tick per second (2 bytes) [0 ... 65,535]</li> <li>2. Base width in mm [1 ... 65,535].</li> <li>3. Max speed in cm/s [1 ... 65,535]</li> <li>4. Direction [1...4] where: <ul style="list-style-type: none"> <li>➔ 1 = M1 counter and M2 clockwise</li> <li>➔ 2 = M1 clockwise and M2 counter</li> <li>➔ 3 = M1 clockwise and M2 clockwise</li> <li>➔ 4 = M1 counter and M2 counter</li> </ul> </li> </ol> <p>Ex: w 20 75 200 1 (for 20 ticks / sec, 75 mm base, max speed 2m/s, direction counter/clock)</p>	w <tick> <base> <speed> <dir>
W	Drive function: stop v-omega driving	W <void>
A	<p>Drive function: change v-omega 4 PID parameters</p> <ul style="list-style-type: none"> <li>• v: typically set around 7 (parameter 70) [1/10]</li> <li>• t: typically set around 0.2 (parameter 20) [1/100]</li> </ul> <p>Ex: A 70 30 70 10 (for vv = 7, tv = 0.3, vw = 7, tw = 0.1)</p>	A <vv> <tv> <vw> <tw>
x	<p>Drive function: set linear and angular speed where:</p> <ul style="list-style-type: none"> <li>• linear: [-32,678...+32,767] in mm/s (2 bytes)</li> <li>• angular: [-128...+127] in rad/sec (1 byte) where -128 is 180 right and +128 is 180 left</li> </ul> <p>Ex:</p> <p>x 100 0 (drive with speed 0.1 mm/s for 1 meter/second straight)</p> <p>x -32678 127 (drive backward for speed 32.678 m/s and curve left for 180<sup>o</sup>/s)</p>	x <linear sp><angular sp>
X	Drive function: get and print vehicle speed to console where linear speed is printed as [cm/s] - 2 bytes and angular speed as [0.1 rad/s] – 1 byte. Read 3 bytes	X <void>
q	Drive function: get vehicle in x, y and phi axis where x and y are defined in mm [-32,768...32767] and phi in [100 rad/s] [-314...314]. Return 6 bytes. By default x = 0, y = 0 and phi = 0.	q <void>

Command	Description	Parameters
Q	<p>Drive function: set vehicle in x, y and phi axis where x and y are defined in mm [-32,768...32767] and phi in [100 rad/s] [-314...314]. Return 6 bytes. By default x = 0, y = 0 and phi = 0.</p> <p>Ex:            &gt;&gt; Q 200 100 157            (x = 0.2 m, y= 0.1m and phi = 90 degrees to the left, clockwise)</p> <p>Q 32767 -32767 -314            (x = 32.767m, y= -32.767m and phi = 360 degrees to the right, counter-clockwise)</p>	Q <x> <y> <phi>
y	<p>Drive function: driving straight with speed in mm/s (0...65,535) and distance in mm [-32,768...32,767]            Total bytes: 4 bytes</p> <p>Ex.            y 100 -1000 (speed = 100mm/s and -1000 = 1 meter backward)</p> <p>y 65535 -32767 (speed = 65.535 m/s with distance going backward to 32.768 m)</p>	y <speed> <distance>
Y	<p>Drive function: turn vehicle on the spot with angle [-3276.8...3276.7] in radian with angular speed [0...6553.5] rad/s where positive angle to the left (counter – clockwise) and negative angle to the right (clockwise)</p> <p>Ex:            Y 1000 -6284 (turn in 100 rad/s with twice full clockwise rotation (720 degrees))</p> <p>Y 65535 32768 (turn with speed 6553.5 rad/s with full counter-clockwise rotation (360 degree))</p>	Y <angular speed><angle>
C	<p>Drive function: driving vehicle curving forward / backward with speed in mm/s [0...65535] (2 bytes), distance in mm [-32768...32767] (right...left) 2 bytes and angle [-3276.8...3276.8] radian</p> <p>Ex:            C 1000 2071 1000 (drive forward with 1 m/s for 2.071 meter and slightly skewed to the left for 90 degrees)</p> <p>C 65535 -32768 -32768 (drive backward with speed 65.535 m/s for distance 32.768 m while curving 360 degrees to the right)</p>	C <speed> <distance> <angle>
z	<p>Drive function: a function to check how far in mm or radians the distance remaining (0 if reached), 2 bytes</p>	z <void>

Command	Description	Parameters
Z	Drive function: a function to check if the drive function has already completed (done or stalled). This will return 0 if drive function is achieved and return 1 otherwise	Z <void>
L	Check if the one of the motor is stalled. It will return stall bit flags for motors [1...4] Ex. Return value: 0b1111 = all motors stalled, 0b1101: only motor 3 stalled	L <void>

# 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, MacOC 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://robotics.ee.uwa.edu.au/eyesim/>  
<http://robotics.ee.uwa.edu.au/eyesim/ftp/>

## 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.

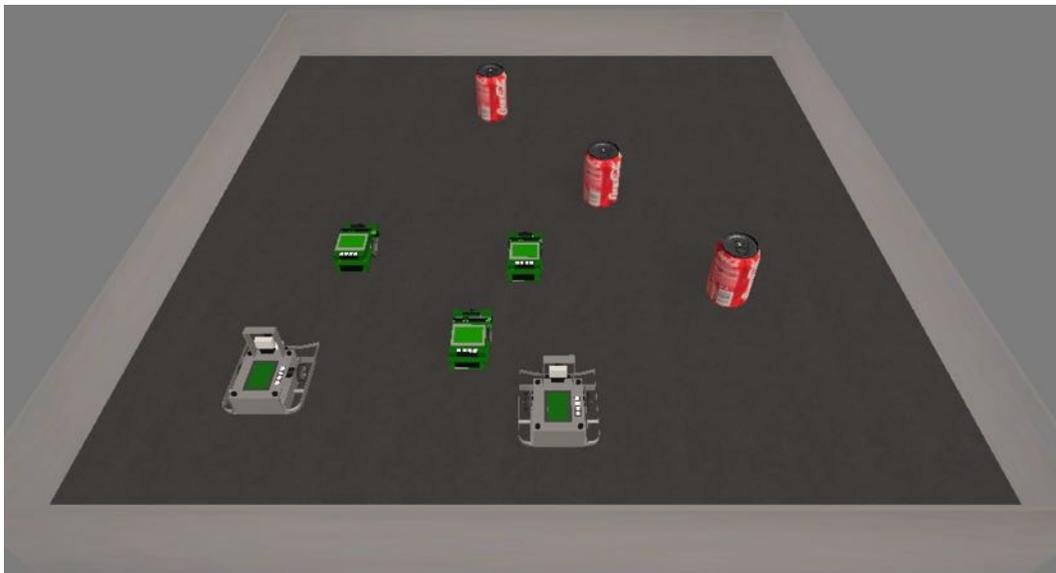


Figure 5.1: Several robots interacting with objects

## 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 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
  - EyeBot7 IO-Board (connected by 2 USB/micro-USB cables)
  - 3 PSD infrared distance sensors with cable, e.g. Sharp GP2Y0A41SK0F
  - 2 Motors 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, 7.2V
  - Mounting plate and brackets out of aluminum or 3D-printed plastic
- 

### 6.1 Robot Design

The following images show different versions of putting together Raspberry Pi controller and the EyeBot7 IO-Board with motors and sensors to build a complete robot.

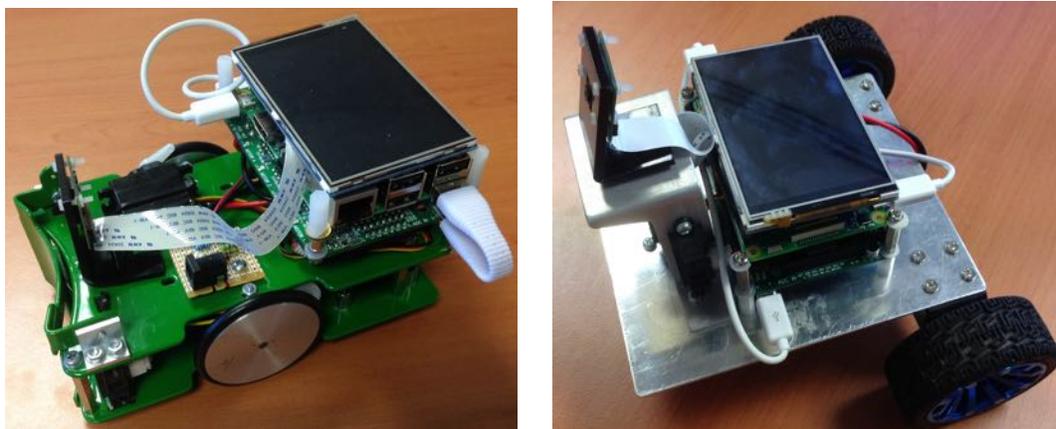


Figure 6.1: SoccerBot and EyeCart mobile robots

EyeBot7-I/O board, Raspberry Pi and LCD are mounted as a tack on top of the robot platform. All motors (DC motors, servos, etc.) and sensor (encoders, infrared PSD distance sensors, etc.) simply plug into the EyeBot7 controller, no soldering is required.

For more details please consult the book "Embedded Robotics", Springer-Verlag, by Thomas Bräunl.

<https://www.amazon.com/Embedded-Robotics-Mobile-Applications-Systems/dp/3540705333/>

<http://www.springer.com/us/book/9783540343196>