

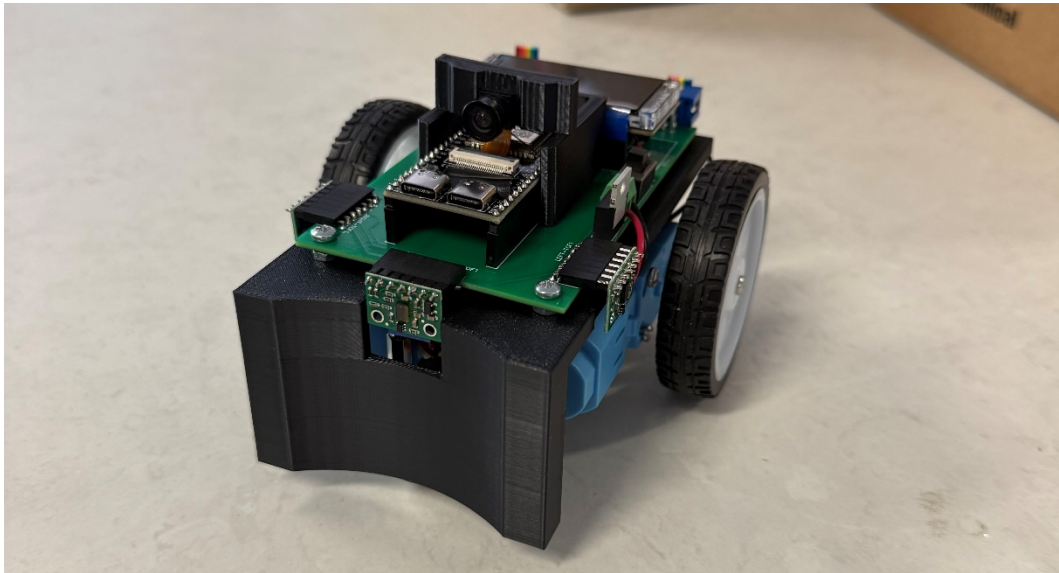
EyeBot-32 Manual

Designed by Parham Bahrami, supervised by Thomas Bräunl

Contact: 23426998@student.uwa.edu.au

Thomas.Braunl@UWA.edu.au

<https://roblab.org/eyebot/>



EyeBot-32 is a classroom-ready mobile robot built around the ESP32-S3. It combines a colour camera, three time-of-flight distance sensors, two encoder-driven motors, and a bright TFT so you can *see, think, and move* in real time. The robot is programmed with PlatformIO in C with a tiny API (RoBIOS-style) that hides hardware complexity and lets you focus on autonomy: follow walls, track lines, detect edges, and navigate.

What it is:

- Controller: TTGO T-Display S3
- Camera: OV2640 @ QQVGA for fast vision labs
- Distance sensors: 3× VL53L0X (front/left/right)
- Drive: 2× DC gearmotors + encoders
- UI: 1.9" TFT + two buttons
- Power: dual 18650 protected batteries

These EyeBots have been designed as part of a final year student project – we hope that you find it useful for your learning in ‘Embedded Systems’!

PlatformIO Library Installation Guide:

Steps to install the EyeBot library:

1. Download the zip file of the EyeBot library and the VL53L0X library.
2. Extract both these libraries into your PlatformIO project's library folder.
3. Ensure your .ini file contains the below:

```
[env:lilygo-t-display-s3]

platform = espressif32

board = lilygo-t-display-s3

framework = arduino

monitor_speed = 115200

monitor_filters = esp32_exception_decoder


lib_deps =

    Bodmer/TFT_eSPI

    Wire

    esp32-camera


build_flags =

    -Llib/eyebot/src

    -leyebot
```

Take note of the following:

1. If you use your own TFT library, you can remove the TFT_eSPI entry.
2. You will need to define the T-Display S3 as your board inside of the TFT library, as usual.
3. If you save the library as part of your PlatformIO global libraries instead of a project specific library, you will need to change the first build flag to be the path of your library. For example:

-L/Users/yourname/.platformio/lib/eyebot/src

Quick Start Guide

EyeBot is equipped with the brand new RoBIOS API, which contains 77 accessible functions which range from driving, button presses, camera usage, image processing and screen writing. These simplify how you program the robot by taking away the brunt work of repetitive tasks such as taking a camera image or driving straight and allow you to build code mostly based on logical building blocks as opposed to hardware integration.

What is RoBIOS?

RoBIOS is a small, stable Application Programming Interface (API) that hides the hardware details so students can write portable robot code fast. On EyeBot-32, we expose the old RoBIOS-style on top of Arduino/FreeRTOS:

- LCD* – text and simple graphics
- CAM* – grab a camera frame (RGB565 colour or grayscale)
- IP* – image processing (Sobel, Laplace, morph ops, grayscale/colour convert)
- VW* – “vehicle” motion primitives (straight/turn/curve, set speed)
- MOTOR* – direct wheel control and PID tuning
- KEY* – read the two onboard buttons
- PSD* – read front/left/right distance sensors in mm

You still write Arduino-style programs with `setup()`/`loop()`, but most labs can be solved just calling these functions. You do not need to know the internals of the functions to do well in the labs, only their inputs and outputs and what they do.

The header for the library is called “`eyebot.h`”. You will need to add this to your library declarations at the top of your `main.cpp` file.

In order to control the robot, we program the TTGO using the library which sends and receives various signals to control the robot’s actions.

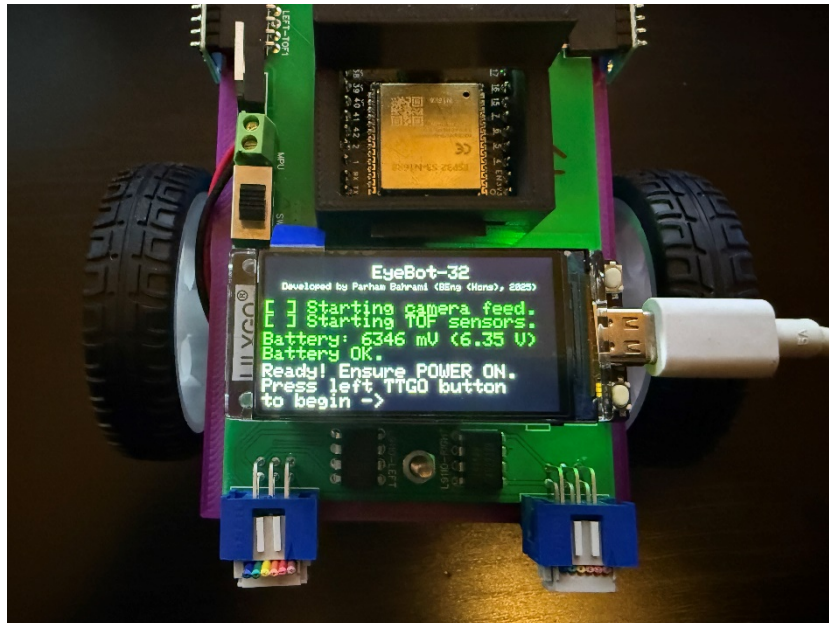
When your sketch boots, as long as you have some RoBIOS function in your code, eyebot.h:

- Powers the board rails and backlight, configures buttons and encoders, and brings the motors to a safe OFF state.
- Initialises SPI (camera), I²C (ToF sensors), and the TFT display; runs a camera sanity check so you see if the feed is live.
- Performs a battery sanity gate with on-screen guidance; you confirm with the left button before anything can move. **If your battery is not on, you cannot run your code.**
- Sets up mutexes/tasks so later drive and sensing functions can run concurrently without you thinking about threads.
- Allows a single button press to begin your code.

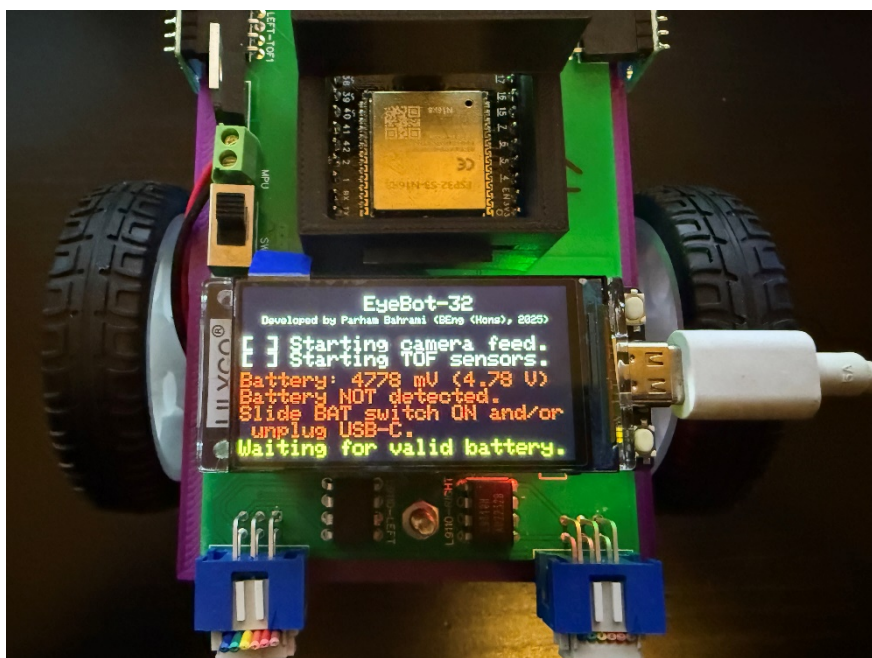
This sequence keeps fleets safe in a lab and makes setup deterministic across many units.

```
#include "eyebot.h"
void setup() {
  // your code here
}
void loop() {
  // your code here
}
```

To program the robot, simply plug it in using a USB-C cable and upload your code as usual.



Please pay attention to the boot screen which accompanies the initialisation function. On this boot screen, the TTGO will check it has an established connection between the camera module and the distance sensors. If it cannot establish this connection, a warning will come up and the TTGO will enter an infinite loop, requiring you to fix the issue and then restart the TTGO. Please ask a facilitator for help if the issue isn't obvious, but the biggest issues are missing distance sensor or an unplugged camera.



Additionally, the TTGO will be powered on when it is plugged in via USB-C, as well as when it is powered on through the battery switch located above the TTGO, to its left. You may flash code onto the TTGO without powering on the board, however the EyeBot library will prevent

you from executing your code if the battery hasn't been turned on. This is to prevent you from power surging your TTGO in the case that your motors are running in your code without the battery. The TTGO on its own is not equipped to power the whole circuit. If you see the red warning saying that the battery is not detected, simply turn on the TTGO's power and please unplug your USB-C cable. There is no need to reset the TTGO after you turn the battery power on. Another indication that your battery power is not working correctly is the bright red LED underneath the TTGO.

Example Code:

Below is an example of how you would drive towards a wall and stop in time. This is done as a standard C-program.

```
1  #include "eyebot.h"
2
3  int main()
4  { VWSetSpeed(100,0); /* drive */
5    while (PSDGet(PSD_FRONT) > 200) /* wait */ ;
6    VWSetSpeed(0,0); /* stop */
7  }
```

When coding the same in PlatformIO we don't need a 'while'-loop as it is already part of the 'loop' function.

```
1  #include "eyebot.h"
2
3  void setup()
4  { VWSetSpeed(100,0); } // drive
5
6  void loop(){
7  { if (PSDGet(PSD_FRONT)<200) VWSetSpeed(0,0); } // stop
```

More example code can be found in the following textbook:

Braunl, T. (2023). *Mobile Robot Programming : Adventures in Python and C* (Second edition). Springer. <https://doi.org/10.1007/978-3-031-32797-1>

RoBIOS Library:

Unless stated otherwise, all non-void functions return 0 on success and -1 on failure.

Camera:

```
int CAMInit(int resolution);           // Prepare camera; resolution must be QQVGA.
int CAMGet565(uint16_t* gray565);      // Capture one frame into RGB565 buffer (QQVGA).
int CAMGet565Gray();                   // Capture grayscale encoded as RGB565
int CAMRelease();                      // Tear down camera (no-op/safe). 0 on success.
int CAMGet(BYTE* img);                 // Capture RGB888 (QQVGA*3 bytes) into img.
int CAMGetGray(BYTE* gimg);            // Capture 8-bit grayscale (QQVGA) into gimg.
BYTE* RGB888ToRGB565(const BYTE* rgb888); // Convert QQVGA RGB888 → RGB565
```

PSD: (Position Sensitive Devices; time-of-flight distance sensors):

```
int PSDGet(int psd);                  // Read VL53L0X distance in mm (1=front,2=left,3=right)
```

V-Omega Driving Interface: (high-level interface for differential driving)

```
int VWSetSpeed(int linSpeed, int angSpeed); // Start background drive: linear (mm/s) &
                                              angular (deg/s)
int VWGetSpeed(double* leftOut, double* rightOut); // Get filtered wheel speeds (mm/s)
int VWSetPosition(int x, int y, int phi); // Set odometry (mm, mm, deg)
int VWGetPosition(int* x, int* y, int* phi); // Get odometry (mm, mm, deg)
int VWRemain(); // Remaining straight-line distance (mm) or -1
                // if not distance-driving.
int VWWait(); // Block until current distance/turn motion
              // completes
int VWDone(); // Driving state flag: 1 while driving, 0 when
              // idle/done.
int VWStraight(int dist, int lin_speed); // Non-blocking straight drive (mm at mm/s)
int VWTurn(int angle, int ang_speed); // Blocking on-spot turn (deg at deg/s); returns
                                       // when turn finishes.
int VWCurve(int dist, int angle, int lin_speed); // Non-blocking arc dist mm, angle deg, mm/s).
int VWDrive(int dx, int dy, int lin_speed); // Non-blocking curve, dx,dy offset; mm/s.
int VWStalled(); // 0=none, 1=left, 2=right, 3=both stalled
                 // (cmd>0 but speed≈0).
int VWGoTo(int dx, int dy, int lin_speed); // Turn toward (dx,dy) then drive straight
int VWStop(); // Stop all motors and kills any VW task.
```

Motor Control: Low-level motor control and encoders

```
int MOTORDrive(int motor, int speed); // Direct wheel: motor 1=left, 2=right; speed -100..100
int MOTORPID(int motor, int p, int i, int d); // Set per-wheel PID gains for speed control.
int MOTORPIDOff(int motor); // Disable/clear PID for wheel (safe no-op)
```

LCD:

```
int LCDPrintf(const char* format, ...); // printf at current cursor.
int LCDSetPrintf(int row,int column,const char*,...); // Move cursor (row,col in px) then printf text.
int LCDClear(); // Clear screen to black.
int LCDSetPos(int row, int column); // Set cursor (row=y, column=x) in pixels.
int LCDGetPos(int *row, int *column); // Get cursor (row=y, column=x). 0 on success.
int LCDSetColor(COLOR fg, COLOR bg); // Set text colors (COLOR = 0xRRGGBB).
int LCDSetFont(int font, int variation); // Select TFT_eSPI font index.
int LCDSetFontSize(int fontsize); // Set text scale factor.
int LCDSetMode(int mode); // Reserved/unsupported (returns -1).
int LCDGetSize(int *x, int *y); // Get display width & height in pixels. 0 on success.

int LCDPixel(int x, int y, COLOR col); // Draw one pixel (0xRRGGBB).
COLOR LCDGetPixel(int x, int y); // Read one pixel color (if supported by panel/driver).

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 Rectangle
int LCDCircle(int x,int y,int radius,COLOR col,int fill); // Circle (fill=1 solid, 0 outline).
int LCDImageStart(int x,int y,int xs,int ys); // Define target area for subsequent image draws.

int LCDImage(BYTE *img); // Draw RGB888 image (QQVGA) into area set by LCDImageStart().
int LCDImage565(const uint16_t* buf); // Draw 160x120 RGB565 image at (0,0).
int LCDImageGray(BYTE *g); // Draw 8-bit grayscale image into current area.
int LCDImageBinary(BYTE *b, uint8_t threshold=128); // Draw binary (>=thr white, else black) from 8-bit gray.

int LCDRefresh(void); // No-op; TFT refreshes as drawn.
```

Note – you may still use the TFT functions, and an instance of the screen is already initialised as an extern variable in the library, called tft. Note not all of the functions have an operation, they are designed to replicate exactly the old EyeBot. Please note the difference between the different LCDImage functions.

Buttons:

```
int KEYGet(); // Wait for a key press; returns bitmask (KEY_LEFT|KEY_RIGHT) after debounce.
int KEYRead(); // Instantaneous key read
int KEYWait(int wantedMask); // Block until any of wanted buttons are pressed
int KEYReadEdge(); // Rising-edge bitmask since last call.
```

These functions use the two buttons on the TTGO. Note that KEY_LEFT and KEY_RIGHT have been defined in the library and can be used with these functions.

Image Processing:

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

```
void IPLaplace(BYTE* grayIn, BYTE* grayOut);           // 3x3 Laplacian edge magnitude on 8-bit
                                                         gray (0..255).
void IPSobel(BYTE* grayIn, BYTE* grayOut);             // True Sobel gradient magnitude on 8-bit
                                                         gray (0..255).
void IPCol2Gray(BYTE* imgIn, BYTE* grayOut);           // RGB888 → grayscale (luma) per pixel.
void IPGray2Col(BYTE* imgIn, BYTE* colOut);            // Grayscale → RGB888 (replicate Y into
                                                         R=G=B).
void IPRGB2Col(BYTE* r,BYTE* g,BYTE* b,BYTE* imgOut);  // Pack planar R/G/B channels into
                                                         interleaved RGB888.
void IPCol2HSI(BYTE* img,BYTE* h,BYTE* s,BYTE* i);     // RGB888 → H,S,I (each 0..255) per pixel.
void IPOverlay(BYTE* c1,BYTE* c2,BYTE* cOut);          // Overlay RGB888 c2 on c1 (pure black in
                                                         c2 = transparent).
void IPOverlayGray(BYTE* g1,BYTE* g2,BYTE r,BYTE g,BYTE b,BYTE* cOut); // Colorize mask g2 over gray
                                                         g1 with (r,g,b).
void IPPCol2RGB(COLOR col,BYTE* r,BYTE* g,BYTE* b);    // Split 0xRRGGBB into 8-bit R,G,B.
void IPPCol2HSI(BYTE* imgIn,BYTE* H,BYTE* S,BYTE* I);  // RGB888 → H,S,I (per pixel), pointer alias
                                                         of IPCol2HSI.
void IPPRGB2HSI(BYTE r,BYTE g,BYTE b,BYTE* H,BYTE* S,BYTE* I); // One-pixel RGB → H,S,I (0..255).
void IPRGB565toHSI(uint16_t* src,int w,int h,
                    uint8_t* H,uint8_t* S,uint8_t* I8); // RGB565 image → H,S,I planes.
void IPRGB565toGray(uint16_t* src,int w,int h,uint8_t* gray); // RGB565 → 8-bit grayscale.
void IP565Adjust(uint16_t* buf,size_t pixels,
                 int bright,float contrast,bool swapBytes); // In-place RGB565 brightness/contrast
void IP565Brightness(uint16_t* buf,size_t pixels,
                    int bright,bool swapBytes);         // RGB565 brightness only.
void IP565Contrast(uint16_t* buf,size_t pixels,
                  float contrast,bool swapBytes);        // RGB565 contrast only.
void IPOpen(uint8_t* binaryImg);                       // 3x3 morphological opening on binary
                                                         (0/1) image, in-place.
void IPAdjust(BYTE* rgb,size_t pixels,
              int bright,float contrast);               // In-place RGB888 brightness/contrast.
void IPBrightness(BYTE* rgb,size_t pixels,int bright);  // RGB888 brightness only.
void IPContrast(BYTE* rgb,size_t pixels,float contrast); // RGB888 contrast only.
```