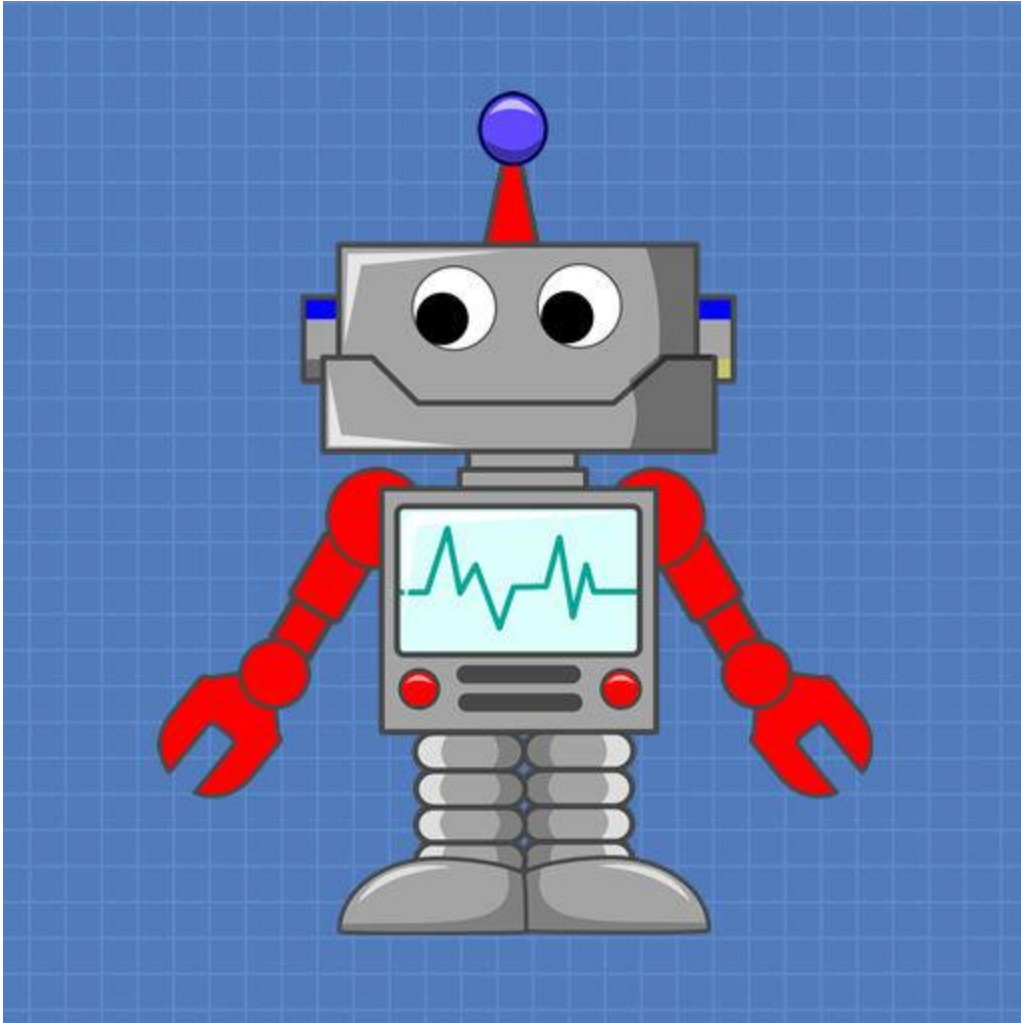


# Arduino GIGA Display Shield



DroneBot Workshop Tutorial

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Today, we will take an in-depth look at the new Arduino GIGA Display Shield. This useful peripheral adds a touchscreen GUI to your Arduino GIGA board, along with a microphone, IMU, and a handy connection for a video camera.



## Introduction

We have already looked at the Arduino GIGA board, an incredibly powerful microcontroller in the same format as the classic Arduino Mega 2560. Today we will look at an upgrade for the GIGA, a “shield” that is actually more of a “reverse shield” (this will make more sense as you read on, trust me!).

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



This new board will allow you to construct great-looking user interfaces for your projects. It makes a powerful combination when added to the GIGA's already impressive list of features.

<https://dronebotworkshop.com>

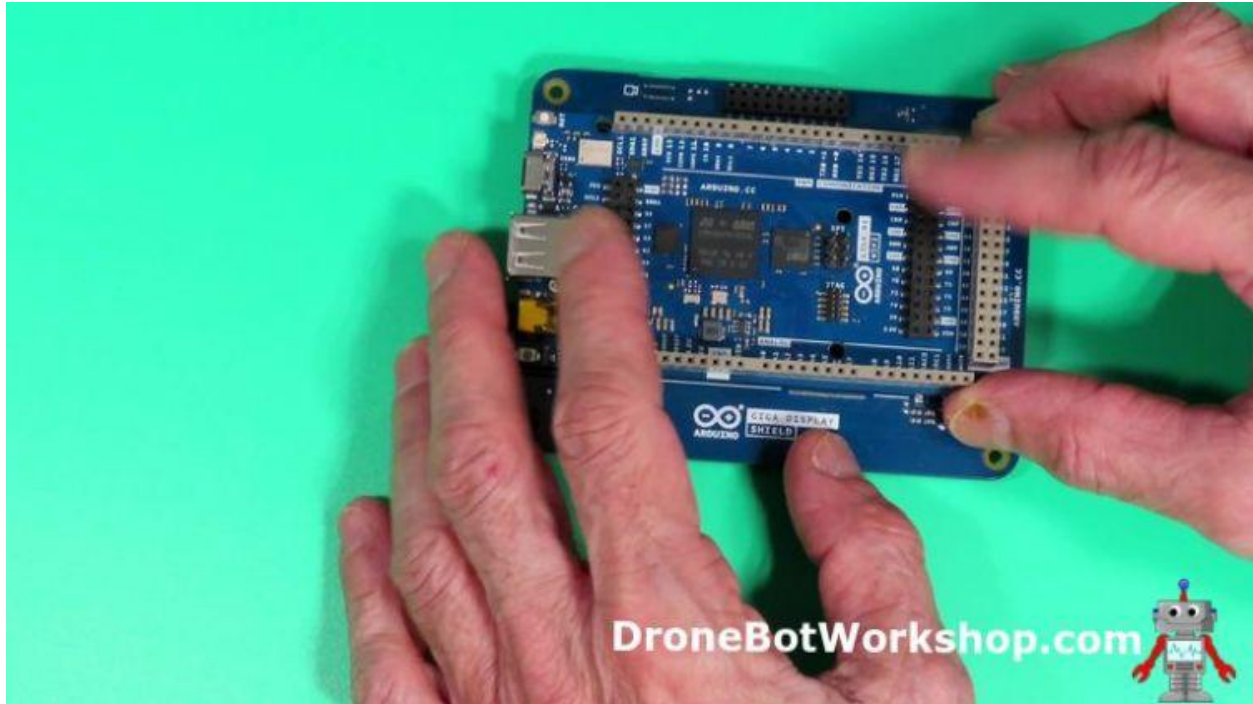
## Arduino GIGA Display Shield

The GIGA Display Shield is a 480 x 800 touchscreen display that snaps onto the back of the Arduino GIGA.



There is a silk-screened outline on the back of the display board to assist you in orienting the GIGA correctly. This is a different mounting arrangement from that used by “regular” shields that mount on the female Dupont connectors on the top of the board. You can still mount conventional shields on the board with the display installed.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



In addition to the display, the GIGA Display Shield also has an onboard MEMS microphone. This allows for the construction of point-of-entry terminals and audio analysis tools.

The board also has an IMU capable of sensing motion and acceleration. You can use this to adjust display orientation or even to build a game.

<https://dronebotworkshop.com>



An onboard RGB LED, as is an extension for the SPI camera connector, is also included. The latter is compatible with a number of Arducam camera modules.

## Getting Started

Let's get started with your new display!

Before starting working with the display, we must ensure that our system is set up correctly for the Arduino GIGA WiFi boards. We must also install several libraries to utilize all the display features fully.

## Setting up the Arduino GIGA WiFi

We have covered working with the Arduino GIGA WiFi board in the [video](#) and [article](#) dedicated to it, so I'll refer you to that if you want some additional information.



Otherwise, installing the Arduino GIGA board manager is the only step you need to take. Search for “GIGA” in the Arduino IDE Boards Manager window and install the resulting file. The board manager will also install many example sketches, a few of which we will use later.

If you are on Linux, you may encounter an error when uploading to the GIGA board. This known bug can be resolved by creating a simple text file. You will find all the details in [the previous GIGA article](#).

## Installing the Libraries

In addition to the files installed by the board manager, we will need to install some libraries. Lots of libraries. Nine, to be exact!

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

But don't worry, it's not as much work as you might expect, as one of the libraries will also install seven more. So, you only need to perform two installations if you execute them correctly.

All the libraries can be installed using the Library Manager in the Arduino IDE. Open it up and search for the "Arduino\_GigaDisplay" library. You will get a list of items; install the first one.

You will also be prompted to install some other libraries upon which the Arduino\_GigaDisplay" library is dependent. In total, you will be installing the following libraries:

- Arduino\_GigaDisplay
- Adafruit BusIO
- Adafruit GFX Library
- ArduinoGraphics
- Arduino\_BMI270\_BMM150
- Arduino\_GigaDisplayTouch
- Arduino\_GigaDisplay\_GFX
- lvgl

Note that you may already have some of these libraries installed in your IDE; only the missing ones will be installed. Ensure that any existing installed libraries are updated to the most recent version.

Once you have everything installed, we can start testing the display shield!

## Testing the IMU

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

The Bosch BMI270 6 Axis IMU (Inertial Measurement Unit) is well suited for use on the GIGA display, as it can provide precise acceleration and angular rate measurement. It also has on-chip motion-triggered interrupt features.



We can use the IMU to control the display orientation to behave like a phone or tablet. It could also be very useful if you want to design a game or create an interactive controller.

To measure acceleration, we can use the `Arduino_BMI270_BMM150` library. It provides a very simple method of getting acceleration data for all three of the axis measurements.

Here is a sketch that will provide this data:

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
1  /*
2   Arduino GIGA Display - IMU Acceleration Demo
3   gigadisplay-imu-test.ino
4   Demonstrates operation of builtin IMU
5   Displays acceleration on serial monitor
6   Requires rduino_BMI270_BMM150 library
7   Code provided by Arduino
8
9   DroneBot Workshop 2023
10  https://dronebotworkshop.com
11 */
12
13 // Include libraries
14 #include "Arduino_BMI270_BMM150.h"
15
16 // Define IMU object on second I2C bus
17 BoschSensorClass imu(Wire1);
18
19 void setup() {
20   // Start serial monitor
21   Serial.begin(115200);
22
23   // Start IMU
24   imu.begin();
25 }
26
27 void loop() {
28   // Variables for X, Y and Z axis
```

<https://dronebotworkshop.com>

```
29  float x, y, z;
30  if (imu.accelerationAvailable()) {
31      // Read acceleration values
32      imu.readAcceleration(x, y, z);
33
34      // Print to serial monitor
35      Serial.print(x);
36      Serial.print('\t');
37      Serial.print(y);
38      Serial.print('\t');
39      Serial.println(z);
40  }
41 }
```

We start the sketch by including the library and then defining an IMU object. Note that we use the Wire1 interface, the second I2C bus.

In Setup, we start the Serial Monitor and also start the IMU.

In the Loop, we simply define variables for the X, Y, and Z axis and then read and print the values.

Load the sketch and move the display around while observing the serial monitor readings. You should see that they correlate to the movements you are making.



## Working with the RGB LED

The display module also has a surface-mount RGB LED, which can be controlled through the `Arduino_GigaDisplay` library.

Here is a sketch that demonstrates its operation:

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
1  /*
2   Arduino GIGA Display - RGB LED Test
3   gigadisplay-rgb.ino
4   Demonstrates operation of onboard RGB LED
5   Requires Arduino_GigaDisplay Library
6
7   DroneBot Workshop 2023
8   https://dronebotworkshop.com
9  */
10
11 // Include library
12 #include <Arduino_GigaDisplay.h>
13
14 // Create RGB object
15 GigaDisplayRGB rgb;
16
17 void setup() {
18   // Initialize the RGB object
19   rgb.begin();
20 }
21
22 void loop() {
23
24   // Turn on red pixel
25   rgb.on(255, 0, 0);
26   delay(1000);
27   // Turn off all pixels
28   rgb.off();
```

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
29  delay(1000);
30
31  // Turn on green pixel
32  rgb.on(0, 255, 0);
33  delay(1000);
34  // Turn off all pixels
35  rgb.off();
36  delay(1000);
37
38  // Turn on blue pixel
39  rgb.on(0, 0, 255);
40  delay(1000);
41  // Turn off all pixels
42  rgb.off();
43  delay(1000);
44
45  // Turn on all pixels
46  rgb.on(255, 255, 255);
47  delay(1000);
48  // Turn off all pixels
49  rgb.off();
50  delay(1000);
51 }
```

This is a very simple sketch that uses the `Arduino_GigaDisplay` library. We include the library and then define an RGB object. In `Setup`, we simply initialize that object.

The `Loop` is where we manipulate the three segments of the display, and it is pretty self-explanatory.

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

We can turn the display on and pass it RGB values to select the color. Or we can turn the display off.

Otherwise, the sketch is really just like an advanced “blink” sketch, turning the LED on and off every second in a different color.

The RGB LED would make a good status display for your project.

## Microphone with Level Display

Another peripheral included with the Arduino GIGA Display Module is a microphone. This is an MP34DT06JTR MEMS microphone module, an omnidirectional capacitive-sensing device with low noise and high sensitivity.

The output of the microphone is PDM or Pulse Density Modulation. Arduino has a PDM Library for working with devices like this.

Until now, we have demonstrated the GIGA Display components by showing their output on the serial monitor. We could continue this trend with the microphone, displaying its output level on the monitor, but that’s a bit boring. And as we have a display to evaluate, why not use it instead of the serial monitor?

The following sketch, using code provided by Arduino, will do the trick. It will display the microphone output level on a bargraph on the display.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
1  /*
2   Arduino GIGA Display - Microphone Demo
3   gigadisplay-microphone.ino
4   Demonstrates operation of builtin microphone
5   Displays audio level on bargraph
6   Requires PDM, Arduino_H7_Video and lvgl library
7   Code provided by Arduino
8
9   DroneBot Workshop 2023
10  https://dronebotworkshop.com
11 */
12
13 // Include libraries
14 #include <PDM.h>
15 #include "Arduino_H7_Video.h"
16 #include "lvgl.h"
17
18 // Create display object
19 Arduino_H7_Video Display(800, 480, GigaDisplayShield);
20
21 // Slider
22 static void set_slider_val(void *bar, int32_t val) {
23     lv_bar_set_value((lv_obj_t *)bar, val, LV_ANIM_ON);
24 }
25
26 // Default number of output channels
27 static const char channels = 1;
28
```

<https://dronebotworkshop.com>

```
29 // Default PCM output frequency
30 static const int frequency = 16000;
31
32 // Buffer to read samples into, each sample is 16-bits
33 short sampleBuffer[512];
34
35 // Number of audio samples read
36 volatile int samplesRead;
37
38 lv_obj_t *obj;
39 lv_anim_t a;
40 int micValue;
41
42 void setup() {
43     // Initialize Display
44     Display.begin();
45
46     // Setup callback
47     PDM.onReceive(onPDMdata);
48
49     // Start PDM
50     if (!PDM.begin(channels, frequency)) {
51         Serial.println("Failed to start PDM!");
52         while (1)
53             ;
54     }
55
56     // Create the bar
```

```
57  obj = lv_bar_create(lv_scr_act());
58  lv_obj_set_size(obj, 600, 50);
59  lv_obj_center(obj);
60  lv_bar_set_value(obj, 500, LV_ANIM_OFF);
61
62  // Create the animation for the bar
63  lv_anim_init(&a);
64  lv_anim_set_exec_cb(&a, set_slider_val);
65  lv_anim_set_time(&a, 300);
66  lv_anim_set_playback_time(&a, 300);
67  lv_anim_set_var(&a, obj);
68 }
69
70 void loop() {
71
72  // Wait for samples to be read
73  if (samplesRead) {
74
75    // Print samples to the serial monitor or plotter
76    for (int i = 0; i < samplesRead; i++) {
77      Serial.println(sampleBuffer[i]);
78      micValue = sampleBuffer[i];
79      micValue = micValue / 10; // Adjust divisor for sensitivity
80      if (micValue > 500) {
81        micValue = 500;
82      }
83      lv_anim_set_values(&a, 0, micValue);
84      lv_anim_start(&a);
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
85     }
86
87     // Clear the read count
88     samplesRead = 0;
89     delay(10);
90 }
91 lv_timer_handler();
92 }
93
94 // Callback function to process the data from the PDM microphone.
95
96 void onPDMdata() {
97     // Query the number of available bytes
98     int bytesAvailable = PDM.available();
99
100    // Read into the sample buffer
101    PDM.read(sampleBuffer, bytesAvailable);
102
103    // 16-bit, 2 bytes per sample
104    samplesRead = bytesAvailable / 2;
105 }
```

The code includes the PDM Library for working with digital audio and the Arduino H7 video library. The LVGL library is also included to provide the graphics (we will learn more about LVGL in a bit).

This is filled by a function that sets the slider value.

We then define some audio parameters; then we move to the Setup.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

In Setup, we initialize the display. We then set the PDM object to have a callback function, which will be called every time that audio is received. After that, we start the PDM.

We also set up both the bargraph and the animation for the bar.

In the Loop, we check whether any samples have been read. If they have, we create an array with the samples. We then read that array, one element at a time, and use the value to operate the bargraph.

We then clear the samples, add a short delay, and wait again.

Note that the final line in the Loop, a call to the *lv\_timer\_handler*, is required for LVGL. We will see more of that later.

The *onPDMData* function is the callback handler, called whenever data is available. It reads the data into a sample buffer, 2 bytes at a time, as it is 16-bit.



<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

You'll see a bargraph on the display when you load up the sketch. Make some noise, and you should see the bar move. You can adjust the divisor on line 79 (current value is 10) to increase or decrease the sensitivity.

# Using the Display for Graphics

So, now that we have looked at the included peripherals, it's time to focus on the display itself. And by that, I mean we need to learn how to place graphics, images, and text onto the screen.

There are three graphics libraries that you can use to work with the display:

- ArduinoGraphics Library
- GFX Library
- LVGL Library & Framework

## Graphics 1 – ArduinoGraphics Library

The [ArduinoGraphics library](#) allows you to draw and write on the display with “graphical primitives,” basic shapes like circles and rectangles.

This library has syntax similar to the [Processing 3](#) GUI design tool.

## ArduinoGraphics Simple Demo

To illustrate just how easy it is to use the ArduinoGraphics library, I have put together a very simple demonstration. Here is the sketch:

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
1  /*
2   Arduino GIGA Display - ArduinoGraphics Demo
3   gigadisplay-arduinographics.ino
4   Demonstrates ArduinoGraphics library functions
5   Uses Arduino_H7_Video and ArduinoGraphics libraries
6
7   DroneBot Workshop 2023
8   https://dronebotworkshop.com
9  */
10
11 // Include required libraries
12 #include "Arduino_H7_Video.h"
13 #include "ArduinoGraphics.h"
14
15 // Create a display object
16 Arduino_H7_Video Display(800, 480, GigaDisplayShield);
17
18 void setup() {
19
20   // Initialize display
21   Display.begin();
22
23   // Draw a red rectangle that covers the entire display
24   Display.beginDraw();
25   Display.clear();
26   Display.noStroke();
27   Display.fill(255, 0, 0);
28   Display.rect(0, 0, Display.width(), Display.height());
```

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
29
30 // Draw a 200px blue circle at coordinates 200,200
31 Display.fill(0, 0, 255);
32 Display.circle(200, 200, 200);
33
34 // Draw a 300px green circle at coordinates 500,300
35 Display.fill(0, 255, 0);
36 Display.circle(500, 300, 300);
37
38 // Finish drawing
39 Display.endDraw();
40 }
41
42 void loop() {
43 }
```

The sketch starts by including the two libraries we need. The *Arduino\_H7\_Video* library is the library that drives the GIGA Display.

We then create an object to represent the display.

From here on, everything is done in the Setup routine. We start by initializing the display and then begin our drawing.

After that, we draw a rectangle that fits the entire display; this is a method of changing the background color.

The line “Display.fill(255, 0, 0)” sets the color. You can change the RGB values if you want a different color. Currently, it is set to red. The *Display.rect* command draws a

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

rectangle with bottom left coordinates at 0,0, and the size of the rectangle which is equal to the display screen size.

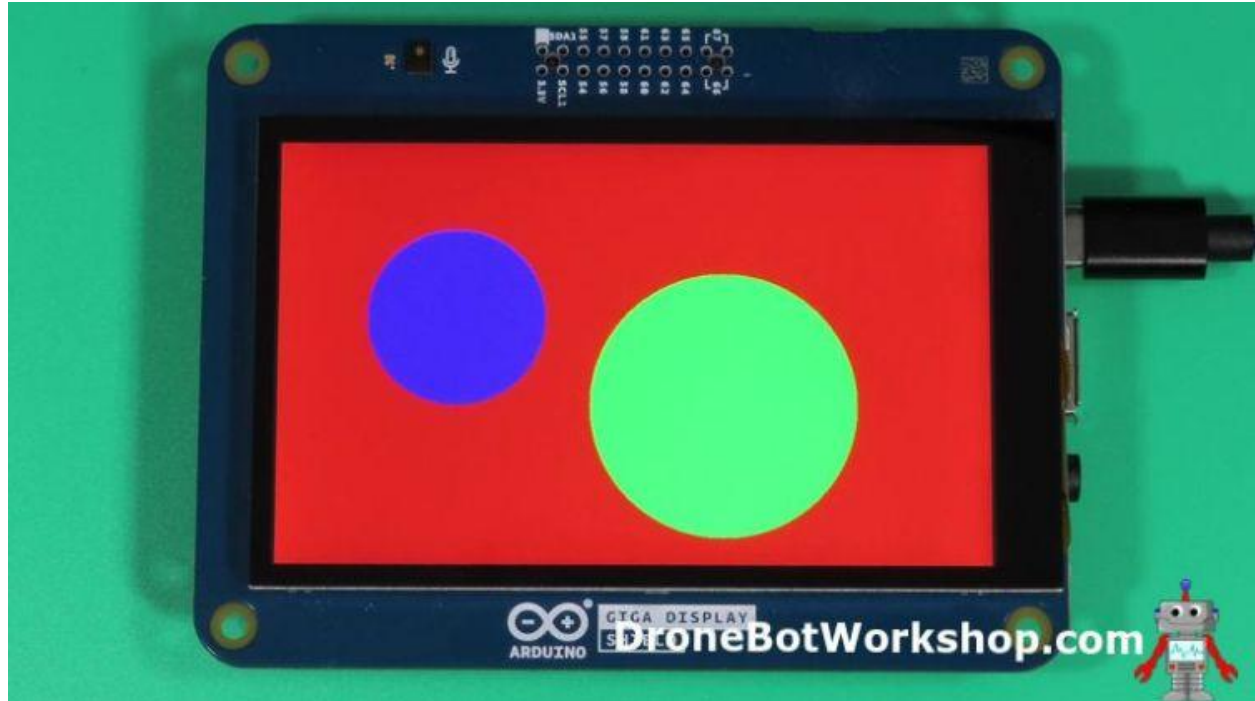
Next is a circle. All that is required here is to fill the display with the desired color, which, in this case, is blue. We then use a *Display.circle* to create our circle, specifying the diameter and the location of the circle's center.

After that, we repeat the circle process, this time with a 300px green circle at coordinates 500,300.

Finally, we finish drawing with *Display.endDraw()*. This will complete the job and print on the display.

That finishes the program; there is nothing in the Loop.

Here is what the final result is.



<https://dronebotworkshop.com>

## Displaying Images with ArduinoGraphics

Another thing we can do with the ArduinoGraphics library is to display images.

We will use the ArduinoLogo example installed when you added the GIGA Boards Manager. You will find it at this location:

*File/Examples/Examples for Arduino Giga R1/Arduino\_H7\_Video/ArduinoLogo*

```
1 /*
2   ArduinoLogo
3
4   created 17 Apr 2023
5   by Leonardo Cavagnis
6 */
7
8 #include "Arduino_H7_Video.h"
9 #include "ArduinoGraphics.h"
10
11 #include "img_arduino.h"
12 // Alternatively, any raw RGB565 image can be included on demand using this macro
13 // Online image converter: https://lvgl.io/tools/imageconverter (Output format:
14 //   Binary RGB565)
15 /*
16 #define INCBIN_PREFIX
17 #include "incbin.h"
18 INCBIN(test, "/home/user/Downloads/test.bin");
19 */
20
21 Arduino_H7_Video Display(800, 480, GigaDisplayShield);
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
21//Arduino_H7_Video Display(1024, 768, USBCVideo);
22
23Image img_arduinologo(ENCODING_RGB16, (uint8_t *) texture_raw, 300, 300);
24
25void setup() {
26  Display.begin();
27
28  Display.beginDraw();
29  Display.image(img_arduinologo, (Display.width() - img_arduinologo.width())/2,
30  (Display.height() - img_arduinologo.height())/2);
31  Display.endDraw();
32}
33
34void loop() { }
```

The sketch is pretty simple; it loads the same two libraries we used earlier. It also includes the “*img\_arduinologo.h*” file; this is the actual Arduino logo graphic converted to a C file.

The file *incbin.h* is also included but isn’t actually used to display the Arduino Logo. It’s used when displaying binary graphics files, which we will do in a moment when we substitute the Arduino logo for an image of our own.

Once again, we set up a display object. We also set up an image object, pointing to the “*img\_arduinologo.h*” file.

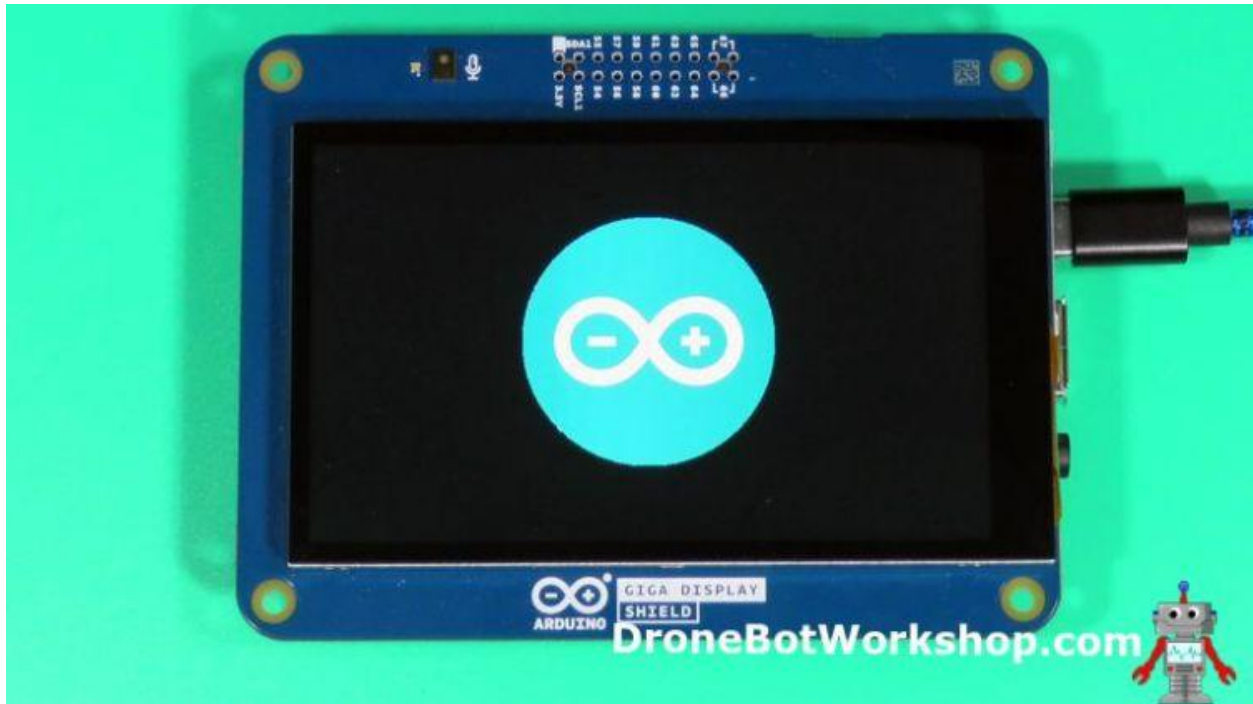
In Setup, the display is started, and we begin the drawing. We then use the *Display.image* command to display the image, providing the image, its width and height, and the position we want to place it.

We finish the same way we did before, with *Display.endDraw()*.

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

The results are shown here:



## Displaying Your Own Image

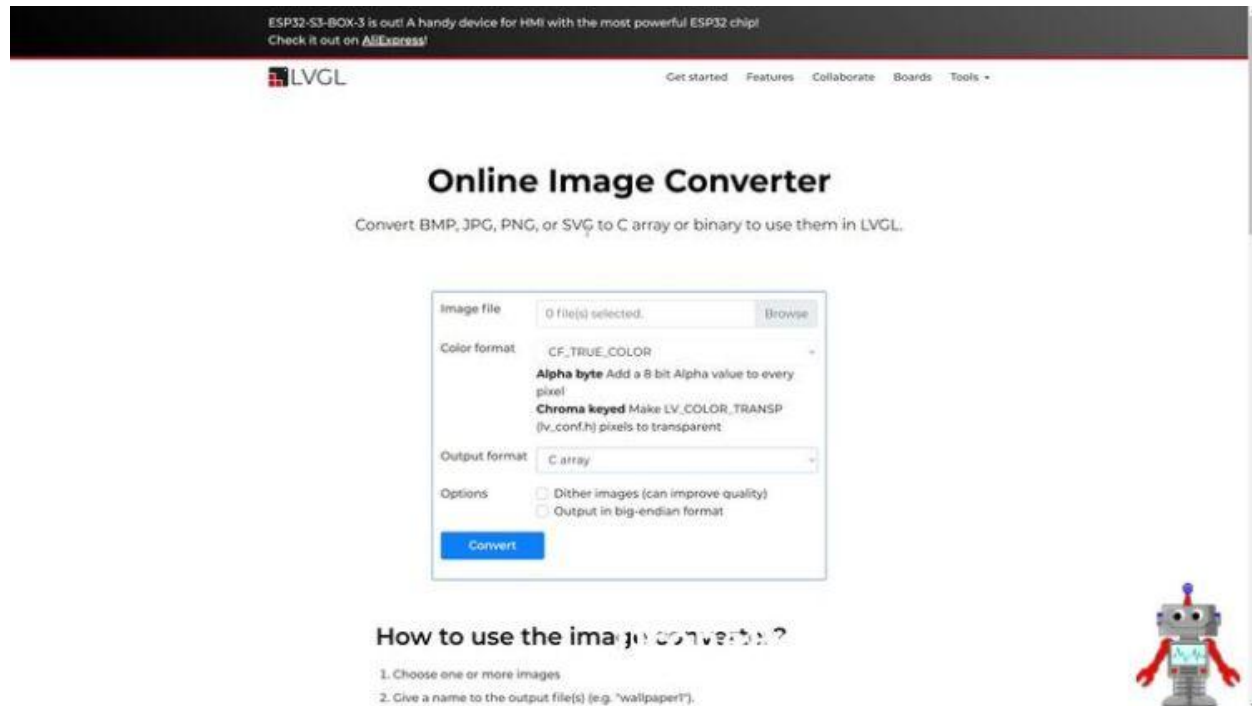
Displaying the Arduino logo is fine, but I'm sure you would rather display your own images. The sketch we just used can be modified to do that. You'll note some instructions in the remarks at the beginning of the code.

Before you run the next sketch, you must prepare your image. It will need to be correctly sized to fit onto the display, and it will also have to be converted into a binary format.

You can use any photo editor to resize the image if it's necessary; the screen is 480×800. Make note of the actual image size, as you will need that in the code.

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



To convert it to a binary format, you can use the [LVGL Graphics Converter](#). Although it is provided for use with LVGL, it can also be used to convert graphics for other libraries.

Use the graphics converter as follows:

- Use the *Browse* button to select your BMP, JPG, PNG, or SVG graphic file.
- Leave the *Color Format* at *CF\_TRUE\_COLOR*.
- Change the *Output Format* to *Binary RGB565*.
- Click the *Convert* button. You'll get an image to download to your local drive.

The resulting file will be a binary file with a ".bin" extension. Ensure you note where it has been saved, as you'll need the file path when you create the code.

The *ArduinoLogo* that we just looked at has instructions for using your own file. Here is a sketch written using those examples:

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
1 /*
2   Arduino GIGA Display - Display Image
3   gigadisplay-display-image.ino
4   Demonstrates operation of onboard RGB LED
5   Requires Arduino_H7_Video and ArduinoGraphics Libraries
6   Requires incbin.h by Dale Weiler
7   Code example "ArduinoLogo" by Leonardo Cavagnis
8
9   DroneBot Workshop 2023
10  https://dronebotworkshop.com
11 */
12
13 // Include required libraries
14 #include "Arduino_H7_Video.h"
15 #include "ArduinoGraphics.h"
16
17 // Any raw RGB565 image can be included on demand using this macro
18 // Online image converter: https://lvgl.io/tools/imageconverter (Output format:
19 // Binary RGB565)
20 // Download *.bin file and note file path
21
22 #define INCBIN_PREFIX
23 #include "incbin.h"
24 // Replace with path to your *.bin file
25 INCBIN(test, "/home/dronebotworkshop/Downloads/robot-redarms-358x422.bin");
26
27 // Create display object
28 Arduino_H7_Video Display(800, 480, GigaDisplayShield);
```

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
29 // Define image variable with parameters (edit size parameter as required)
30 Image img_robot(ENCODING_RGB16, (uint8_t *) testData, 358, 422);
31
32 void setup() {
33   // Start display
34   Display.begin();
35
36   // Draw image in center of display
37   Display.beginDraw();
38   Display.image(img_robot, (Display.width() - img_robot.width())/2,
39   (Display.height() - img_robot.height())/2);
40   Display.endDraw();
41 }
42
43 void loop() { }
```

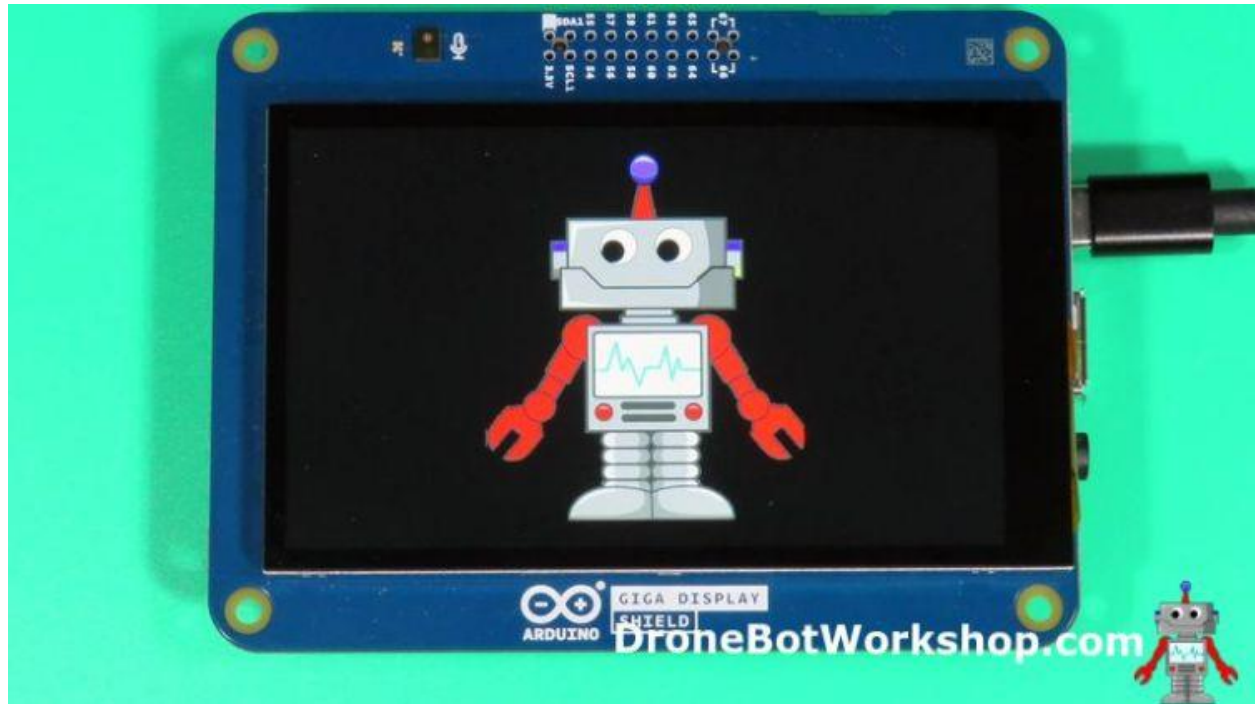
Note that you will still need the *cabin.h* file; however, you no longer need the Arduino logo file.

The method of calling the binary file is a bit different from the previous example; it's actually very simple. Here is what you need to change to use it with your graphic file:

- On line 24, you must change the file's location to your downloaded file location.
- On line 32, change the size dimensions of the image to match your image.

You can also change the coordinates used by *Display.image* to move the image away from the display center.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



If all is well, you should be rewarded with a display of your custom image!

<https://dronebotworkshop.com>

## Graphics 2 – GFX Library

Another method of manipulating the display is to use the popular GFX Graphics Library. This graphics library by Adafruit is a highly popular and versatile tool for creating graphical interfaces and displays for microcontroller-based projects.

Initially, the GFX Graphics Library only supported Adafruit displays; however, it has now been expanded to work with a wide range of displays from multiple manufacturers, including OLEDs, TFTs, and LCDs.

### Running the GFX Graphics Demo

A great way to become familiar with the GFX Graphics Library is to run the demo sketch. This is a standard demonstration that cycles the display through a number of shapes and functions. We have used this sketch with other displays, so it's a great way to see how to adapt other GFX codes to the Arduino GIGA Display Shield.

You can grab the sketch from the following location:

*[File/Examples/Examples From Custom Libraries/Arduino\\_GigaDisplay\\_GFX/Demo](#)*

```
1 #include "Arduino_GigaDisplay_GFX.h"
2
3 GigaDisplay_GFX tft;
4
5 #define GC9A01A_CYAN    0x07FF
6 #define GC9A01A_RED     0xf800
7 #define GC9A01A_BLUE    0x001F
8 #define GC9A01A_GREEN   0x07E0
9 #define GC9A01A_MAGENTA 0xF81F
10 #define GC9A01A_WHITE   0xffff
11 #define GC9A01A_BLACK   0x0000
12 #define GC9A01A_YELLOW  0xFFE0
13
14 void setup() {
15     Serial.begin(9600);
16     Serial.println("GC9A01A Test!");
17
18     tft.begin();
19
20     Serial.println(F("Benchmark          Time (microseconds)"));
21     delay(10);
22     Serial.print(F("Screen fill          "));
23     Serial.println(testFillScreen());
24     delay(500);
25
26     Serial.print(F("Text                      "));
27     Serial.println(testText());
28     delay(3000);
```

```
29
30 Serial.print(F("Lines          "));
31 Serial.println(testLines(GC9A01A_CYAN));
32 delay(500);
33
34 Serial.print(F("Horiz/Vert Lines  "));
35 Serial.println(testFastLines(GC9A01A_RED, GC9A01A_BLUE));
36 delay(500);
37
38 Serial.print(F("Rectangles (outline)  "));
39 Serial.println(testRects(GC9A01A_GREEN));
40 delay(500);
41
42 Serial.print(F("Rectangles (filled)    "));
43 Serial.println(testFilledRects(GC9A01A_YELLOW, GC9A01A_MAGENTA));
44 delay(500);
45
46 Serial.print(F("Circles (filled)      "));
47 Serial.println(testFilledCircles(10, GC9A01A_MAGENTA));
48
49 Serial.print(F("Circles (outline)        "));
50 Serial.println(testCircles(10, GC9A01A_WHITE));
51 delay(500);
52
53 Serial.print(F("Triangles (outline)      "));
54 Serial.println(testTriangles());
55 delay(500);
56
```

```
57 Serial.print(F("Triangles (filled)  "));
58 Serial.println(testFilledTriangles());
59 delay(500);
60
61 Serial.print(F("Rounded rects (outline)  "));
62 Serial.println(testRoundRects());
63 delay(500);
64
65 Serial.print(F("Rounded rects (filled)  "));
66 Serial.println(testFilledRoundRects());
67 delay(500);
68
69 Serial.println(F("Done!"));
70 }
71
72 void loop(void) {
73   for (uint8_t rotation = 0; rotation < 4; rotation++) {
74     tft.setRotation(rotation);
75     testText();
76     delay(1000);
77   }
78 }
79
80 unsigned long testFillScreen() {
81   unsigned long start = micros();
82   tft.fillScreen(GC9A01A_BLACK);
83   yield();
84   tft.fillScreen(GC9A01A_RED);
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
85  yield();
86  tft.fillScreen(GC9A01A_GREEN);
87  yield();
88  tft.fillScreen(GC9A01A_BLUE);
89  yield();
90  tft.fillScreen(GC9A01A_BLACK);
91  yield();
92  return micros() - start;
93 }
94
95 unsigned long testText() {
96  tft.fillScreen(GC9A01A_BLACK);
97  unsigned long start = micros();
98  tft.setCursor(0, 0);
99  tft.setTextColor(GC9A01A_WHITE);  tft.setTextSize(1);
100  tft.println("Hello World!");
101  tft.setTextColor(GC9A01A_YELLOW); tft.setTextSize(2);
102  tft.println(1234.56);
103  tft.setTextColor(GC9A01A_RED);   tft.setTextSize(3);
104  tft.println(0xDEADBEEF, HEX);
105  tft.println();
106  tft.setTextColor(GC9A01A_GREEN);
107  tft.setTextSize(5);
108  tft.println("Groop");
109  tft.setTextSize(2);
110  tft.println("I implore thee,");
111  tft.setTextSize(1);
112  tft.println("my foonting turlingdromes.");
```

<https://dronebotworkshop.com>

```
113  tft.println("And hooptiously drangle me");
114  tft.println("with crinkly bindlewurdles,");
115  tft.println("Or I will rend thee");
116  tft.println("in the gobberwarts");
117  tft.println("with my blurglecruncheon,");
118  tft.println("see if I don't!");
119  return micros() - start;
120 }
121
122 unsigned long testLines(uint16_t color) {
123     unsigned long start, t;
124     int          x1, y1, x2, y2,
125                w = tft.width(),
126                h = tft.height();
127
128     tft.fillScreen(GC9A01A_BLACK);
129     yield();
130
131     x1 = y1 = 0;
132     y2   = h - 1;
133     start = micros();
134     for (x2 = 0; x2 < w; x2 += 6) tft.drawLine(x1, y1, x2, y2, color);
135     x2    = w - 1;
136     for (y2 = 0; y2 < h; y2 += 6) tft.drawLine(x1, y1, x2, y2, color);
137     t     = micros() - start; // fillScreen doesn't count against timing
138
139     yield();
140     tft.fillScreen(GC9A01A_BLACK);
```

```
141  yield();
142
143  x1    = w - 1;
144  y1    = 0;
145  y2    = h - 1;
146  start = micros();
147  for (x2 = 0; x2 < w; x2 += 6) tft.drawLine(x1, y1, x2, y2, color);
148  x2    = 0;
149  for (y2 = 0; y2 < h; y2 += 6) tft.drawLine(x1, y1, x2, y2, color);
150  t     += micros() - start;
151
152  yield();
153  tft.fillScreen(GC9A01A_BLACK);
154  yield();
155
156  x1    = 0;
157  y1    = h - 1;
158  y2    = 0;
159  start = micros();
160  for (x2 = 0; x2 < w; x2 += 6) tft.drawLine(x1, y1, x2, y2, color);
161  x2    = w - 1;
162  for (y2 = 0; y2 < h; y2 += 6) tft.drawLine(x1, y1, x2, y2, color);
163  t     += micros() - start;
164
165  yield();
166  tft.fillScreen(GC9A01A_BLACK);
167  yield();
168
```

```
169  x1      = w - 1;
170  y1      = h - 1;
171  y2      = 0;
172  start = micros();
173  for (x2 = 0; x2 < w; x2 += 6) tft.drawLine(x1, y1, x2, y2, color);
174  x2      = 0;
175  for (y2 = 0; y2 < h; y2 += 6) tft.drawLine(x1, y1, x2, y2, color);
176
177  yield();
178  return micros() - start;
179 }
180
181 unsigned long testFastLines(uint16_t color1, uint16_t color2) {
182  unsigned long start;
183  int          x, y, w = tft.width(), h = tft.height();
184
185  tft.fillScreen(GC9A01A_BLACK);
186  start = micros();
187  for (y = 0; y < h; y += 5) tft.drawFastHLine(0, y, w, color1);
188  for (x = 0; x < w; x += 5) tft.drawFastVLine(x, 0, h, color2);
189
190  return micros() - start;
191 }
192
193 unsigned long testRects(uint16_t color) {
194  unsigned long start;
195  int          n, i, i2,
196
197              cx = tft.width() / 2,
```

```
197         cy = tft.height() / 2;
198
199     tft.fillScreen(GC9A01A_BLACK);
200     n = min(tft.width(), tft.height());
201     start = micros();
202     for (i = 2; i < n; i += 6) {
203         i2 = i / 2;
204         tft.drawRect(cx - i2, cy - i2, i, i, color);
205     }
206
207     return micros() - start;
208 }
209
210 unsigned long testFilledRects(uint16_t color1, uint16_t color2) {
211     unsigned long start, t = 0;
212     int          n, i, i2,
213
214         cx = tft.width() / 2 - 1,
215         cy = tft.height() / 2 - 1;
216
217     tft.fillScreen(GC9A01A_BLACK);
218     n = min(tft.width(), tft.height());
219     for (i = n; i > 0; i -= 6) {
220         i2 = i / 2;
221         start = micros();
222         tft.fillRect(cx - i2, cy - i2, i, i, color1);
223         t += micros() - start;
224         // Outlines are not included in timing results
225         tft.drawRect(cx - i2, cy - i2, i, i, color2);
```

```
225     yield();
226 }
227
228 return t;
229 }
230
231 unsigned long testFilledCircles(uint8_t radius, uint16_t color) {
232     unsigned long start;
233     int x, y, w = tft.width(), h = tft.height(), r2 = radius * 2;
234
235     tft.fillScreen(GC9A01A_BLACK);
236     start = micros();
237     for (x = radius; x < w; x += r2) {
238         for (y = radius; y < h; y += r2) {
239             tft.fillCircle(x, y, radius, color);
240         }
241     }
242
243     return micros() - start;
244 }
245
246 unsigned long testCircles(uint8_t radius, uint16_t color) {
247     unsigned long start;
248     int x, y, r2 = radius * 2,
249         w = tft.width() + radius,
250         h = tft.height() + radius;
251
252     // Screen is not cleared for this one -- this is
```

```
253 // intentional and does not affect the reported time.
254 start = micros();
255 for (x = 0; x < w; x += r2) {
256     for (y = 0; y < h; y += r2) {
257         tft.drawCircle(x, y, radius, color);
258     }
259 }
260
261 return micros() - start;
262 }
263
264 unsigned long testTriangles() {
265     unsigned long start;
266     int          n, i, cx = tft.width() / 2 - 1,
267             cy = tft.height() / 2 - 1;
268
269     tft.fillScreen(GC9A01A_BLACK);
270     n = min(cx, cy);
271     start = micros();
272     for (i = 0; i < n; i += 5) {
273         tft.drawTriangle(
274             cx      , cy - i, // peak
275             cx - i, cy + i, // bottom left
276             cx + i, cy + i, // bottom right
277             tft.color565(i, i, i));
278     }
279
280     return micros() - start;
```

```
281 }
282
283 unsigned long testFilledTriangles() {
284     unsigned long start, t = 0;
285     int          i, cx = tft.width() / 2 - 1,
286                cy = tft.height() / 2 - 1;
287
288     tft.fillScreen(GC9A01A_BLACK);
289     start = micros();
290     for (i = min(cx, cy); i > 10; i -= 5) {
291         start = micros();
292         tft.fillTriangle(cx, cy - i, cx - i, cy + i, cx + i, cy + i,
293             tft.color565(0, i * 10, i * 10));
294         t += micros() - start;
295         tft.drawTriangle(cx, cy - i, cx - i, cy + i, cx + i, cy + i,
296             tft.color565(i * 10, i * 10, 0));
297         yield();
298     }
299
300     return t;
301 }
302
303 unsigned long testRoundRects() {
304     unsigned long start;
305     int          w, i, i2,
306                cx = tft.width() / 2 - 1,
307                cy = tft.height() / 2 - 1;
308
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
309  tft.fillScreen(GC9A01A_BLACK);
310  w    = min(tft.width(), tft.height());
311  start = micros();
312  for (i = 0; i < w; i += 6) {
313      i2 = i / 2;
314      tft.drawRoundRect(cx - i2, cy - i2, i, i, i / 8, tft.color565(i, 0, 0));
315  }
316
317  return micros() - start;
318 }
319
320 unsigned long testFilledRoundRects() {
321     unsigned long start;
322     int          i, i2,
323             cx = tft.width() / 2 - 1,
324             cy = tft.height() / 2 - 1;
325
326     tft.fillScreen(GC9A01A_BLACK);
327     start = micros();
328     for (i = min(tft.width(), tft.height()); i > 20; i -= 6) {
329         i2 = i / 2;
330         tft.fillRoundRect(cx - i2, cy - i2, i, i, i / 8, tft.color565(0, i, 0));
331         yield();
332     }
333
334     return micros() - start;
335 }
```

The sketch uses the [Arduino\\_GigaDisplay\\_GFX Library](#).

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

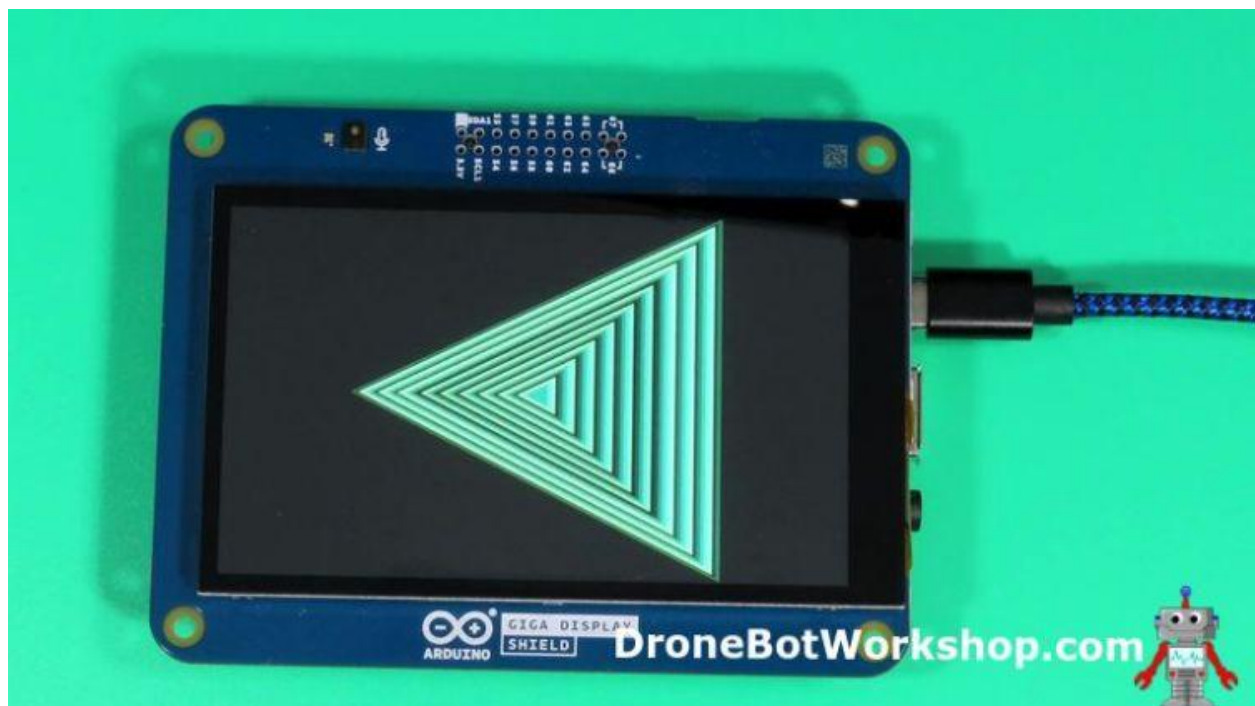
We start by defining an object to represent the display, followed by a number of color definitions.

Then we move to Setup. Here, we start the serial monitor and the display; then, we call a number of functions. These functions, defined at the bottom of the sketch, cycle the display through various patterns and sequences.

You can examine each function to see the commands used to create the display pattern.

After running through Setup, we finish in the Loop. Here, we display some test text (Vogon poetry from The Hitchhiker's Guide to the Galaxy) in different fonts. The display will cycle through this text in different orientations until it is reset or powered off.

Load it up and watch it work. Then, try modifying the code and observe the results.



<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

You can learn more about working with this library in the Arduino [GIGA Display Shield GFX Guide](#).

## Graphics 3 – LVGL

The [Light and Versatile Graphics Library](#), or LVGL, is more than just a library. This is a complete graphics framework that can be used to create some very sophisticated displays.

LVGL has over 30 built-in widgets, a layout manager, and a style system. It also integrates with [SquareLine Studio](#), a professional GUI design application with a drag-and-drop editor.

### LVGL Demo

Once again, we will run a sketch demonstrating this graphics library's capabilities. This demo produces an interface screen divided into quadrants. Each section has different controls – an image, a working slider, some buttons and checkboxes, and a bargraph.

You'll find the demonstration sketch at this location in the Arduino IDE:

*[File/Examples/Examples for Arduino Giga R1/Arduino\\_H7\\_Video/LVGLDemo](#)*

```
1  /*
2    LVGLDemo
3
4    created 17 Apr 2023
5    by Leonardo Cavagnis
6  */
7
8  #include "Arduino_H7_Video.h"
9  #include "Arduino_GigaDisplayTouch.h"
10
11 #include "lvgl.h"
12
13 Arduino_H7_Video      Display(800, 480, GigaDisplayShield); /* Arduino_H7_Video
14 Display(1024, 768, USBCVideo); */
15 Arduino_GigaDisplayTouch  TouchDetector;
16
17 /* Button click event callback */
18 static void btn_event_cb(lv_event_t * e) {
19     static uint32_t cnt = 1;
20     lv_obj_t * btn = lv_event_get_target(e);
21     lv_obj_t * label = lv_obj_get_child(btn, 0);
22     lv_label_set_text_fmt(label, "%LV_PRIu32, cnt);
23     cnt++;
24 }
25
26 /* Slider update value handler */
27 static void set_slider_val(void * bar, int32_t val) {
28     lv_bar_set_value((lv_obj_t *)bar, val, LV_ANIM_ON);
29 }
```

```
29
30 void setup() {
31     Serial.begin(115200);
32
33     Display.begin();
34     TouchDetector.begin();
35
36     /* Create a container with grid 2x2 */
37     static lv_coord_t col_dsc[] = {370, 370, LV_GRID_TEMPLATE_LAST};
38     static lv_coord_t row_dsc[] = {215, 215, LV_GRID_TEMPLATE_LAST};
39     lv_obj_t * cont = lv_obj_create(lv_scr_act());
40     lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);
41     lv_obj_set_size(cont, Display.width(), Display.height());
42     lv_obj_set_style_bg_color(cont, lv_color_hex(0x03989e), LV_PART_MAIN);
43     lv_obj_center(cont);
44
45     lv_obj_t * label;
46     lv_obj_t * obj;
47
48     /* [0;0] - Image */
49     obj = lv_obj_create(cont);
50     lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, 0, 1,
51                          LV_GRID_ALIGN_STRETCH, 0, 1);
52
53     LV_IMG_DECLARE(img_arduinoologo);
54     lv_obj_t * img1 = lv_img_create(obj);
55     lv_img_set_src(img1, &img_arduinoologo);
56     lv_obj_align(img1, LV_ALIGN_CENTER, 0, 0);
```

```
57  lv_obj_set_size(img1, 200, 150);
58
59  /* [1;0] - Checkboxes and button */
60  obj = lv_obj_create(cont);
61  lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, 1, 1,
62                      LV_GRID_ALIGN_STRETCH, 0, 1);
63  lv_obj_set_flex_flow(obj, LV_FLEX_FLOW_COLUMN);
64
65  lv_obj_t * cb;
66  cb = lv_checkbox_create(obj);
67  lv_checkbox_set_text(cb, "Apple");
68
69  cb = lv_checkbox_create(obj);
70  lv_checkbox_set_text(cb, "Banana");
71  lv_obj_add_state(cb, LV_STATE_CHECKED);
72
73  static lv_style_t style_radio;
74  static lv_style_t style_radio_chk;
75  lv_style_init(&style_radio);
76  lv_style_set_radius(&style_radio, LV_RADIUS_CIRCLE);
77  lv_style_init(&style_radio_chk);
78  lv_style_set_bg_img_src(&style_radio_chk, NULL);
79
80  cb = lv_checkbox_create(obj);
81  lv_checkbox_set_text(cb, "Lemon");
82  lv_obj_add_flag(cb, LV_OBJ_FLAG_EVENT_BUBBLE);
83  lv_obj_add_style(cb, &style_radio, LV_PART_INDICATOR);
84  lv_obj_add_style(cb, &style_radio_chk, LV_PART_INDICATOR | LV_STATE_CHECKED);
```

```
85
86  cb = lv_checkbox_create(obj);
87  lv_checkbox_set_text(cb, "Melon");
88  lv_obj_add_flag(cb, LV_OBJ_FLAG_EVENT_BUBBLE);
89  lv_obj_add_style(cb, &style_radio, LV_PART_INDICATOR);
90  lv_obj_add_style(cb, &style_radio_chk, LV_PART_INDICATOR | LV_STATE_CHECKED);
91  lv_obj_add_state(cb, LV_STATE_CHECKED);
92
93  lv_obj_t * btn = lv_btn_create(obj);
94  lv_obj_set_size(btn, 100, 40);
95  lv_obj_center(btn);
96  lv_obj_add_event_cb(btn, btn_event_cb, LV_EVENT_CLICKED, NULL);
97
98  label = lv_label_create(btn);
99  lv_label_set_text(label, "Click me!");
100 lv_obj_center(label);
101
102 /* [0;1] - Slider */
103 obj = lv_obj_create(cont);
104 lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, 0, 1,
105                      LV_GRID_ALIGN_STRETCH, 1, 1);
106
107 lv_obj_t * slider = lv_slider_create(obj);
108 lv_slider_set_value(slider, 75, LV_ANIM_OFF);
109 lv_obj_center(slider);
110 label = lv_label_create(obj);
111 lv_label_set_text(label, "Drag me!");
112 lv_obj_align_to(label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
113
114  /* [1;1] - Bar */
115  obj = lv_obj_create(cont);
116  lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, 1, 1,
117                      LV_GRID_ALIGN_STRETCH, 1, 1);
118
119  lv_obj_t * bar = lv_bar_create(obj);
120  lv_obj_set_size(bar, 200, 20);
121  lv_obj_center(bar);
122  lv_bar_set_value(bar, 70, LV_ANIM_OFF);
123
124  lv_anim_t a;
125  lv_anim_init(&a);
126  lv_anim_set_exec_cb(&a, set_slider_val);
127  lv_anim_set_time(&a, 3000);
128  lv_anim_set_playback_time(&a, 3000);
129  lv_anim_set_var(&a, bar);
130  lv_anim_set_values(&a, 0, 100);
131  lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
132  lv_anim_start(&a);
133}
134
135void loop() {
136  /* Feed LVGL engine */
137  lv_timer_handler();
138}
```

The sketch makes use of three libraries:

- **Arduino\_H7\_Video** – The library for the GIGA Display Shield.

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

- **Arduino\_GigaDisplayTouch** – The touchscreen library
- **lvgl** – The LVGL Library

We start by defining both the display and the touch detector.

The next bit of code is a callback function, called whenever the button control is clicked.

The function that follows sets the value of the slider control.

We then move into Setup, where most of the code is executed. After starting the serial monitor, display and touch detector, we define a container with a 2×2 grid. This is the container for our graphics display.

Next, we specify the objects that go into each segment of the display grid.

- Grid 0,0 – An Arduino logo image is displayed here.
- Grid 1,0 – A couple of checkboxes, some radio buttons, and a pushbutton.
- Grid 0,1 – A working slider control.
- Grid 1,1 – An animated bargraph.

You can examine the LVGL code statements to see how these objects are created.

The Loop only contains one function, but it is an important one. The call to *lv\_timer\_handler()* is performed periodically to keep the graphics engine functioning. You will need to include this in any code you write that uses e LVGL.

Load the code to the GIGA and observe the display. You should have a professional looking interface with working controls.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



## LVGL with IMU

We can combine the display with the IMU readings to perform special effects. In this example, we will use the IMU to keep the display level, even when it is tilted.

You'll find this example in the Arduino IDE at *[File/Examples/Examples From Custom Libraries/Arduino\\_GigaDisplay/lvgl/imu\\_orientation](#)*

<https://dronebotworkshop.com>

```
1 #include "Arduino_BMI270_BMM150.h"
2 #include "Arduino_H7_Video.h"
3 #include "lvgl.h"
4
5 Arduino_H7_Video      Display(800, 480, GigaDisplayShield); /* Arduino_H7_Video
   Display(1024, 768, USBCVideo); */
6
7
8   BoschSensorClass imu(Wire1);
9
10  LV_IMG_DECLARE(img_arduino);
11  lv_obj_t * img;
12
13  void setup() {
14      Serial.begin(115200);
15
16      Display.begin();
17      imu.begin();
18
19      img = lv_img_create(lv_scr_act());
20      lv_img_set_src(img, &img_arduino);
21      lv_obj_align(img, LV_ALIGN_CENTER, 0, 0);
22      lv_img_set_pivot(img, (img_arduino.header.w)/2, (img_arduino.header.h)/2);
23      /* Rotate around the center of the image */
24  }
25
26  uint8_t rotation = 0;
27
28  void loop() {
29      float x, y, z;
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
29  if (imu.accelerationAvailable()) {
30      imu.readAcceleration(x, y, z);
31      if ( z < 0.8 && z > -0.8) {
32          if (x < -0.8) {
33              rotation = 0;
34          } else if (x > 0.8) {
35              rotation = 2;
36          } else if (y < -0.8) {
37              rotation = 1;
38          } else if (y > 0.8) {
39              rotation = 3;
40          }
41          int16_t rot_angle = 900 - atan(x / y) * 180.0 / M_PI * 10;
42          lv_img_set_angle(img, rot_angle);
43      }
44  }
45  lv_timer_handler();
}
```

The sketch includes a second file, the Arduino logo image we are displaying.

Three libraries are used, including the library for the IMU. After including the libraries, we create objects to represent the display, IMU, and the image itself.

In Setup, we start the serial monitor, display, and IMU. We then define the image and set its position and orientation.

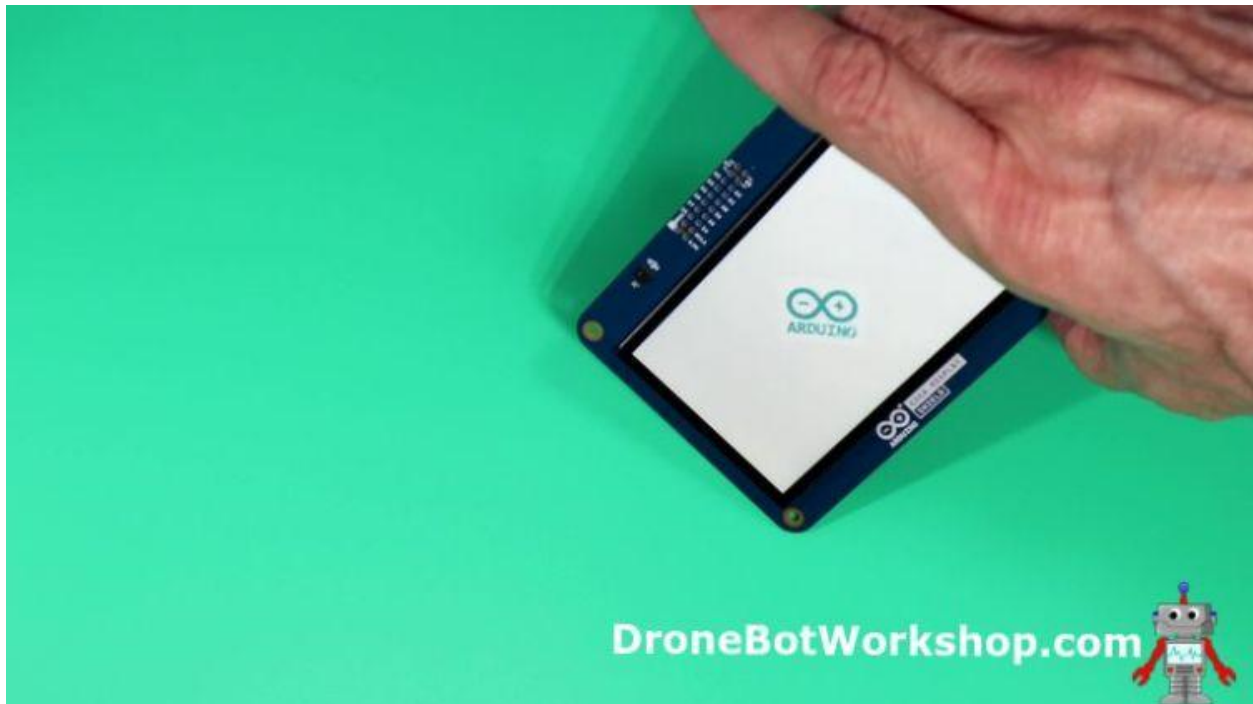
In the Loop, we get the IMU values and determine the rotation value for the x, y, and z-axis. We then use those values to calculate the correct image orientation.

Once again, the *lv\_timer\_handler()* is called to keep the LVGL graphics operational.

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Load the sketch to the Arduino GIGA and observe the image. Now try rotating the display. The image should track the rotation and steady itself.



This technique has applications for both interfaces and games.

## Using the Touch Screen

The Arduino GIGA Display Shield employs a Goodix GT911 capacitive touch controller. This provides up to five concurrent touchpoints, allowing for advanced controls such as gestures.



The touch screen can operate in either a polled or interrupt-driven mode. We will look at code samples for each mode.

## Touch Screen – Polling Mode

In polling mode, we continuously check the touch screen to see if it has been touched. If it has, we grab up to five sets of coordinates.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Open the sample code at *File/Examples/Examples From Custom Libraries/Arduino\_GigaDisplayTouch/Touch\_Polling*

```
1  /*
2   Touch_Polling
3
4   created 03 May 2023
5   by Leonardo Cavagnis
6  */
7
8  #include "Arduino_GigaDisplayTouch.h"
9
10 Arduino_GigaDisplayTouch touchDetector;
11
12 void setup() {
13   Serial.begin(115200);
14   while(!Serial) {}
15
16   if (touchDetector.begin()) {
17     Serial.print("Touch controller init - OK");
18   } else {
19     Serial.print("Touch controller init - FAILED");
20     while(1) ;
21   }
22 }
23
24 void loop() {
25   uint8_t contacts;
26   GDTpoint_t points[5];
```

<https://dronebotworkshop.com>

```
27
28 contacts = touchDetector.getTouchPoints(points);
29
30 if (contacts > 0) {
31     Serial.print("Contacts: ");
32     Serial.println(contacts);
33
34     for (uint8_t i = 0; i < contacts; i++) {
35         Serial.print(points[i].x);
36         Serial.print(" ");
37         Serial.println(points[i].y);
38     }
39 }
40
41 delay(1);
42 }
```

As you can see, this is pretty straightforward code. It uses the *Arduino\_GigaDisplayTouch* library, which is used to create an object representing the touch detector.

In Setup, we start the serial monitor and detector.

In the Loop, we define an array to represent up to five contact coordinates. Then, we simply use the *touchDetector.getTouchPoints* function to populate that array if there are indeed points to record.

After that, we cycle through the array, displaying the coordinates on the serial monitor.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Load the sketch and give it a try. See what happens when you press more than one point.



## Touch Screen – Interrupt Mode

Another method of using the touch screen is interrupt mode. In this mode, an interrupt is generated every time the screen is touched.

This method has many advantages over polling mode, especially in a large sketch where polling could produce erratic results.

Our example is located at *[File/Examples/Examples From Custom Libraries/Arduino\\_GigaDisplayTouch/Touch\\_IRQ](#)*

```
1  /*
2   Touch_IRQ
3
4   created 03 May 2023
5   by Leonardo Cavagnis
6  */
7
8  #include "Arduino_GigaDisplayTouch.h"
9
10 Arduino_GigaDisplayTouch touchDetector;
11
12 void gigaTouchHandler(uint8_t contacts, GDTpoint_t* points) {
13   Serial.print("Contacts: ");
14   Serial.println(contacts);
15
16   if (contacts > 0) {
17     /* First touch point */
18     Serial.print(points[0].x);
19     Serial.print(" ");
20     Serial.println(points[0].y);
21   }
22 }
23
24 void setup() {
25   Serial.begin(115200);
26   while(!Serial) {}
27
28   if (touchDetector.begin()) {
```

```
29     Serial.println("Touch controller init - OK");
30 } else {
31     Serial.println("Touch controller init - FAILED");
32     while(1) ;
33 }
34
35 touchDetector.onDetect(gigaTouchHandler);
36 }
37
38 void loop() { }
```

The sketch starts off identically to the polling example. One difference is that there is a function defined as a “callback function.” This is called every time that a touch has been detected. In the callback, you will grab the coordinates to be used in your code.

The *gigaTouchHandler* function is our callback. In this demo, it just prints the number of touchpoints plus the coordinates of the first one.

Since the handler contains the bulk of the code, Setup is pretty simple. We start the serial monitor and touch display, then set the callback function. And there is no code in the Loop at all!

Load the sketch and go through the same exercise you did for the last sketch. The results will be similar. Note that only the first coordinate is printed when multiple points are touched; this is done to reduce the time the sketch spends inside the callback function.



## Using the Video Camera


The Arduino GIGA Display Shield has an extension for the existing Arducam-compatible camera connector. This allows you to mount a camera facing the display operator.

There are currently four Arducam cameras that can be used with the display:

### HM01B0


## Arduino GIGA Display Camera

### Arducam HM01B0



- Himax 0.1MP HM01B0 Monochrome
- 320 x 320 effective pixels
- 60.0° FOV (Field of View)

DroneBotWorkshop.com



## HM0360

# Arduino GIGA Display Camera

## Arducam HM0360



- Himax 0.3MP HM0360 Color
- 640 x 480 effective pixels
- 64.0° FOV (Field of View)

DroneBotWorkshop.com



## GC2145

# Arduino GIGA Display Camera

## Arducam GC2145



- **GalaxyCore 2MP GC2145 Color**
- **1616 x 1232 effective pixels**
- **80.0° FOV (Field of View)**


**DroneBotWorkshop.com**



## OV7675


# Arduino GIGA Display Camera

## Arducam OV7675



- OmniVision 0.3MP OV7675 sensor
- 640 x 480 effective pixels
- 63.9° FOV (Field of View)

DroneBotWorkshop.com



To demonstrate the camera, we can use the sketch found here:

*File/Examples/Examples from Custom*

*Libraries/Arduino\_GigaDisplay/camera/display\_camera*

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
1 #include "arducam_dvp.h"
2 #include "Arduino_H7_Video.h"
3 #include "dsi.h"
4 #include "SDRAM.h"
5
6 // This example only works with Greyscale cameras (due to the palette +
  resize&rotate algo)
7
8 #define ARDUCAM_CAMERA_HM01B0
9
10 #ifndef ARDUCAM_CAMERA_HM01B0
11 #include "Himax_HM01B0/himax.h"
12
13 HM01B0 himax;
14 Camera cam(himax);
15
16 #define IMAGE_MODE CAMERA_GRAYSCALE
17 #elif defined(ARDUCAM_CAMERA_HM0360)
18 #include "Himax_HM0360/hm0360.h"
19
20 HM0360 himax;
21 Camera cam(himax);
22
23 #define IMAGE_MODE CAMERA_GRAYSCALE
24 #elif defined(ARDUCAM_CAMERA_OV767X)
25 #include "OV7670/ov767x.h"
26
27 // OV7670 ov767x;
28 OV7675 ov767x;
29 Camera cam(ov767x);
30
31 #define IMAGE_MODE CAMERA_RGB565
32 #error "Unsupported camera (at the moment :) )"
33
34 #elif defined(ARDUCAM_CAMERA_GC2145)
35 #include "GC2145/gc2145.h"
36
37 GC2145 galaxyCore;
```

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
29 Camera cam(galaxyCore);
30 #define IMAGE_MODE CAMERA_RGB565
31 #error "Unsupported camera (at the moment :) )"
32 #endif
33
34 // The buffer used to capture the frame
35 FrameBuffer fb;
36 // The buffer used to rotate and resize the frame
37 FrameBuffer outfb;
38 // The buffer used to rotate and resize the frame
39 Arduino_H7_Video_Display(800, 480, GigaDisplayShield);
40
41 void blinkLED(uint32_t count = 0xFFFFFFFF)
42 {
43     pinMode(LED_BUILTIN, OUTPUT);
44     while (count--) {
45         digitalWrite(LED_BUILTIN, LOW); // turn the LED on (HIGH is the voltage level)
46         delay(50);                      // wait for a second
47         digitalWrite(LED_BUILTIN, HIGH); // turn the LED off by making the voltage LOW
48         delay(50);                      // wait for a second
49     }
50 }
51
52 uint32_t palette[256];
53
54 void setup() {
55     // Init the cam QVGA, 30FPS
56     if (!cam.begin(CAMERA_R320x240, IMAGE_MODE, 30)) {
```

<https://dronebotworkshop.com>

```
57     blinkLED();
58 }
59
60 // Setup the palette to convert 8 bit greyscale to 32bit greyscale
61 for (int i = 0; i < 256; i++) {
62     palette[i] = 0xFF000000 | (i << 16) | (i << 8) | i;
63 }
64
65 Display.begin();
66 dsi_configeCLUT((uint32_t*)palette);
67
68 outfb.setBuffer((uint8_t*)SDRAM.malloc(1024*1024));
69
70 // clear the display (gives a nice black background)
71 dsi_lcdClear(0);
72 dsi_drawCurrentFrameBuffer();
73 dsi_lcdClear(0);
74 dsi_drawCurrentFrameBuffer();
75 }
76
77 void loop() {
78
79     // Grab frame and write to another framebuffer
80     if (cam.grabFrame(fb, 3000) == 0) {
81
82         // double the resolution and transpose (rotate by 90 degrees) in the same step
83         // this only works if the camera feed is 320x240 and the area where we want to
84         // display is 640x480
85         for (int i = 0; i < 320; i++) {
```

```
85     for (int j = 0; j < 240; j++) {
86         ((uint8_t*)outfb.getBuffer())[j * 2 + (i * 2) * 480] =
87         ((uint8_t*)fb.getBuffer())[i + j * 320];
88         ((uint8_t*)outfb.getBuffer())[j * 2 + (i * 2) * 480 + 1] =
89         ((uint8_t*)fb.getBuffer())[i + j * 320];
90         ((uint8_t*)outfb.getBuffer())[j * 2 + (i * 2 + 1) * 480] =
91         ((uint8_t*)fb.getBuffer())[i + j * 320];
92         ((uint8_t*)outfb.getBuffer())[j * 2 + (i * 2 + 1) * 480 + 1] =
93         ((uint8_t*)fb.getBuffer())[i + j * 320];
94     }
95     dsi_lcdDrawImage((void*)outfb.getBuffer(), (void*)dsi_getCurrentFrameBuffer(),
96     480, 640, DMA2D_INPUT_L8);
97     dsi_drawCurrentFrameBuffer();
98 } else {
99     blinkLED(20);
100 }
101 }
```

The sketch is written for the Arducam HM01B0 camera but can be modified to use one of the other three cameras. To modify the sketch for a different camera, change line 8 as follows:

- **HM01B0:** `#define ARDUCAM_CAMERA_HM01B0`
- **HM0360:** `#define ARDUCAM_CAMERA_HM0360`
- **GC2145:** `#define ARDUCAM_CAMERA_GC2145`
- **OV7675:** `#define ARDUCAM_CAMERA_OV767x`

The code creates a buffer and dumps frames of video into the buffer. These are then moved to a second display buffer and then shown on the display.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

When I tried the sketch with an Arducam OV7675 it worked, but the video I displayed was upside down and mirror imaged (I can see why the mirroring would be desirable, but the upside down is an error). I tried two cameras with the same results.



Nonetheless, the code does work and illustrates how you can incorporate a camera into your projects that use the Arduino GIGA Display Shield.

<https://dronebotworkshop.com>

## Conclusion

The Arduino GIGA Display Shield is undoubtedly a high-performance display. When coupled with the powerful Arduino GIGA R1, it can be used to create some very sophisticated designs. Using LVGL, you can easily create a GUI rivaling commercial devices.

The downside of this display is, of course, its price. While it isn't too expensive considering its features, it can only be used with the relatively expensive Arduino GIGA R1. Similar LCD solutions that use the Mega 2560 are much less expensive, but they are also not as powerful.

If you need a high-performance display and can justify the price tag, then the Arduino GIGA Display Shield certainly fits the bill. It's easy to use and even leaves the standard Arduino GIGA connectors available for additional shields.

All in all, it's an impressive device!