III

MOTION DETECTION

Now that we got the basics of image capturing, it's time to move to something more involved. This Part III is devoted to motion detection: we want to get notified when something changes in the camera field of view.

Motion detection comes in many shapes and there exists a lot of algorithms: not all of them are suited for real-time execution on the Esp32, though, so we'll take a look at the most basic ones.

Nevertheless, this Part will come handy as a foundation for your own projects.

7

Jpeg decoding digression

At its simplest form, we will define "motion" as a sudden change in the values of a large portion of the pixels of the image.

To count how many pixels changed their value, we first need to *access* the pixels values.

By design, the "EloquentEsp32cam" library only gets the frames out of the Esp32-cam as Jpeg-encoded images.

Why?

Because it is **space-efficient.**

If we were to store the raw RGB pixels of a VGA image (640 x 480), we would need 640 x 480 x 3 = **921600 bytes of RAM**! Even to store the grayscale version we would still need 307200 bytes. It *could* be doable on PSRAM-equipped board, but it will leave few room for other tasks you may want to perform.

A Jpeg-encoded VGA frame at the best quality, on the other hand, usually takes 15-20 Kb (that's a **50x reduction!**). Jpeg encoding fits well for the transmission and storage of images, too, as we saw in the

previous chapters.

The downside is (there must always be a downside) that, when you want to access the raw pixels values, you have to decode the Jpeg. Decoding a jpeg takes a few milliseconds, so we want to optimize this part as much as we can.

Fast Jpeg decoding

It turns out that the Jpeg schema allows you to extract a grayscale version of the image very fast (google for "jpeg DC coefficients"). It is downscaled by a factor of 8, though.

With this optimization, we can decode a VGA frame in 30 milliseconds.

For motion detection this 1/8th version of the image will still work wonder, since we're interested in the portion of the pixels that changed value: the downscaling, in fact, has the added bonus of smoothing the noise out of the image.

```
// 9_Jpeg_Decoding

#include "esp32cam.h"
#include "esp32cam/JpegDecoder.h"

Eloquent::Esp32cam::Cam cam;
Eloquent::Esp32cam::JpegDecoder decoder;

void setup() {
    Serial.begin(115200);
    delay(3000);
    Serial.println("Init");
```

```
    cam.aithinker();
    cam.highQuality();
    cam.vga();

    while (!cam.begin())
        Serial.println(cam.getErrorMessage());
}

void loop() {
  if (!cam.capture()) {
    Serial.println(cam.getErrorMessage());
    return;
  }

  // try to decode image
  if (!decoder.decode(cam)) {
    Serial.println(decoder.getErrorMessage());
    return;
  }

  // we can access the grayscale image at decoder.luma
  // you have access to
  //   - luma.pixels for the uint8_t pixel values
  //   - luma.length for the size of the luma.pixels array
  Serial.print("Luma length: ");
  Serial.println(decoder.luma.length);
  Serial.print("Decoding time: ");
  Serial.print(decoder.getExecutionTimeInMillis());
  Serial.println("ms");
  delay(1000);
}
```

You can do all sort of things with these pixels: in the next chapters, we're going to perform **motion detection.**

8

Image-wise Motion Detection

In most situations, you want to monitor the whole image for something of interest to happen.

You may have your camera pointed in front of a door, or facing to your garden or car… No matter the setting, every change in the image should trigger an alert. We'll call this **image wise motion detection,** as opposed to *region-of-interest* motion detection, which limits the analysis to a small region of the image (next chapter).

There are many terms related to motion detection that you can google if you want to go deeper, like "background modelling", "background subtraction", "foreground detection". The core idea is that you model the background by some algorithm and define *"foreground"* everything that is not background. If this foreground is sufficiently large, you call it *motion*.

Note: this works until you can expect that your scene will be background most of the time and foreground objects will disappear within a few seconds. If an object comes into the scene and stays there for minutes, the algorithm

will continue to trigger until it eventually picks the object up as background!

There are two components that implement motion detection in the "EloquentEsp32Cam" library:

1. the *Detector*: this is like a manager class that counts how many pixels are considered foreground and tells you if motion is detected (when the number of foreground pixels is above a given threshold)
2. the *Algorithm*: this is the model that tells if a pixel is background or foreground

This division is of primary importance because there are a variety of background models and you may want to choose the one that performs best in your specific case. In doing so, the counting procedure will stay the same, so the detector won't change when you change the algorithm.

There are two algorithms implemented in the library:

1. *SimpleChange*: as the name suggests, this model keeps a running average of each pixel's value and tells it is foreground when the difference from its previous value exceeds a threshold you set
2. *SingleGaussian:* this model estimates the mean and variance of each pixel and tells it is foreground when its new value differs from its mean by more than K times its variance (you can set K manually or use the default value)

You are free to implement your own background model class if you're able to, otherwise in most situations '*SimpleChange*' will work pretty well.

A quick note.

```
For motion detection to perform well, I suggest you disable
automatic white balance, otherwise the camera sensor will
artificially alter the pixels and those will be mis-labelled
as foreground
```

Now time to code.

The below snippet builds on the Jpeg decoding example and adds motion detection on the luma component.

```
// 10_Image_Wise_Motion_Detection.ino

/**
 * To optimize memory usage, you have to define
 * at which resolution you will take frames.
 *
 * If you don't have external PSRAM, you should
 * stay below VGA
 */
#define MAX_RESOLUTION_VGA

#include "esp32cam.h"
#include "esp32cam/JpegDecoder.h"
#include "esp32cam/motion/Detector.h"
#include "esp32cam/motion/SimpleChange.h"

Eloquent::Esp32cam::Cam cam;
Eloquent::Esp32cam::JpegDecoder decoder;
Eloquent::Esp32cam::Motion::SimpleChange algorithm;
Eloquent::Esp32cam::Motion::Detector detector(algorithm);

void setup() {
    Serial.begin(115200);
    delay(3000);
    Serial.println("Init");
```

```
cam.aithinker();
cam.highQuality();
cam.vga();
cam.highestSaturation();
/**
 * For motion detection to perform well, I suggest
 * you disable automatic controls, otherwise
 * the camera sensor will artificially alter the
 * pixels and those will be mis-labelled as foreground
 */
cam.disableAutomaticWhiteBalance();
cam.disableAutomaticExposureControl();
cam.disableGainControl();

/**
 * Configure the detector
 */
 /**
  * use the first N frames to train the algorithm
  */
detector.trainFor(30);

/**
 * re-run the training after N frames
 * (at 33 FPS, 33 * 600 = 10 minutes)
 */
detector.retrainAfter(33ULL * 600);

/**
 * detect motion when 20% or more pixels of the frame
 * are labelled as foreground
 */
detector.triggerAbove(0.2);

/**
 * try to remove spurious foreground pixels
```

```
     */
    detector.denoise();

    /**
     * Configure algorithm
     * (each algorithm has its own set of parameters)
     */
     /**
      * label pixel as foreground if its value changed
      * by more than 20 (in a range from 0 to 255)
      */
    algorithm.differBy(20);

    /**
     * when updating the pixel value, how much shall we
     * take into consideration its previous value?
     * The higher this value, the stronger influence
     * the pixel history has w.r.t. its current value
     * The update formula is
     * updated value = a * old value + (1 - a) * new value
     * Where a in in the range 0 - 1 (1 excluded)
     */
    algorithm.smooth(0.9);

    /**
     * when a pixel is labelled as foreground, should we
     * update its value or not?
     * It is updated by default, so if that's what you want,
     * remove the following line.
     */
    algorithm.onlyUpdateBackground();


    while (!cam.begin())
        Serial.println(cam.getErrorMessage());
}
```

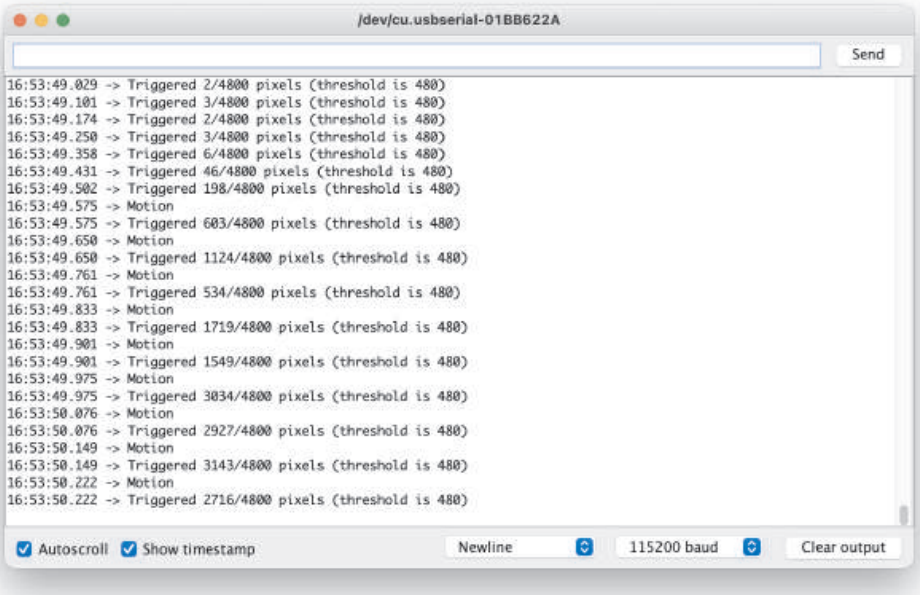
```
void loop() {
    if (!cam.capture()) {
        Serial.println(cam.getErrorMessage());
        return;
    }

    if (!decoder.decode(cam)) {
        Serial.println(decoder.getErrorMessage());
        return;
    }

    /**
     * Update the background model
     * If there's an error, print it
     *
     * Note: while training, `update()` will return False
     * even if it cannot really considered an error.
     * If you want to check if the error is due to training
     or not,
     * you can check for `detector.isTraining()`
     */
    if (!detector.update(decoder.luma)) {
        Serial.println(detector.getErrorMessage());
        return;
    }

    /**
     * Test if motion was detected
     */
    if (detector.triggered()) {
        Serial.println("Motion");
    }

    /**
     * After the call to `triggered()`, you can debug the
     internal
     * status of the detector if you want to find out why it
```

```
    triggered
    * or not
    */
    Serial.println(detector.getTriggerStatus());
}
```

For the moment, you can leave the default configuration as is and upload the sketch.

```
Be sure to point the camera at a static background when it
first boots, otherwise the algorithm won't be able to
correctly model the scene.
```

If you open the Serial Monitor and wave your hand in front of the camera, you will see something like the image below.



Motion detection Serial debug

Now that you got a glimpse on how the detection work, you can test out different configurations for both the detector and the algorithm.

But testing can be time-consuming: every time you update a configuration, you have to re-upload the sketch. **This is not how an expert works.**

Since I'm here to make you an expert, I'll show you a nice way to tune your parameters.

A MIND-BLOWING WAY...

9

Motion Live Video Streaming

If you recall from Chapter 4 "Live Video Streaming", we created an HTTP server that displayed the real-time video feed of the camera into a browser page.
With motion detection we will do the same.

I don't expect you to be able to **find your perfect configuration** by simply looking at the Arduino Serial Monitor and guess what's going on inside the *Detector* and *Algorithm* classes. I want you to **look** at what's happening.

So, without any further awaiting, upload this sketch to your Esp32-cam. You should recognize a few known components (*JpegDecoder, SimpleChange, Detector*) and a new one called "*MotionLiveFeed*". As you may guess, this component will create the HTTP server that displays the live camera feed with a nifty bonus…

```
// 11_Motion_Live_Feed

#define MAX_RESOLUTION_VGA

#include "esp32cam.h"
#include "esp32cam/JpegDecoder.h"
#include "esp32cam/motion/Detector.h"
#include "esp32cam/motion/SimpleChange.h"
#include "esp32cam/http/MotionLiveFeed.h"

// Replace with your WiFi credentials
#define WIFI_SSID "Your SSID"
#define WIFI_PASS "Your password"


Eloquent::Esp32cam::Cam cam;
Eloquent::Esp32cam::JpegDecoder decoder;
Eloquent::Esp32cam::Motion::SimpleChange algorithm;
Eloquent::Esp32cam::Motion::Detector detector(algorithm);
/**
 * Create the MotionLiveFeed HTTP server
 * 80 is the default port for HTTP
 */
Eloquent::Esp32cam::Http::MotionLiveFeed feed(cam, detector,
80);


void setup() {
    Serial.begin(115200);
    delay(3000);
    Serial.println("Init");

    cam.m5wide();
    cam.highQuality();
    cam.vga();
    cam.highestSaturation();
    cam.disableAutomaticWhiteBalance();
```

```
    cam.disableAutomaticExposureControl();
    cam.disableGainControl();

    // See 10_Image_Wise_Motion_Detection for details
    detector.trainFor(30);
    detector.triggerAbove(0.2);
    detector.retrainAfter(33ULL * 600);
    detector.denoise();

    algorithm.differBy(20);
    algorithm.smooth(0.9);
    algorithm.onlyUpdateBackground();

    while (!cam.begin())
        Serial.println(cam.getErrorMessage());

    while (!cam.connect(WIFI_SSID, WIFI_PASS))
        Serial.println(cam.getErrorMessage());

    /**
     * Initialize HTTP server
     */
    while (!feed.begin())
        Serial.println(feed.getErrorMessage());

    cam.viewAt("esp32cam");

    /**
     * Display the IP address of the camera
     */
    Serial.println(feed.getWelcomeMessage());
}


void loop() {
    feed.handle();
}
```

As always, remember to replace your WiFi name and password.

Open the browser at the Esp32-cam IP address (or at http://esp32cam.local) and you will be greeted by something similar to the screeshot below.

Live Motion Feed webpage

Now you have to admit that this is pretty neat. Real-time motion detection debug in the browser. Now this is **expert level**.

You will find the '*detector.getTriggerStatus()*' output at the last line and a couple more:

1. the detector configuration
2. the algorithm configuration

In this way you can take note of their parameters if you find a successful configuration for your own project.

But we closed the previous chapter with a huge problem: **how to quickly change configurations without re-flashing?**

Well, turn out you configure the parameters directly from the URL!

Those parameters you see printed with their values, are also accessible from the URL. Try to visit

```
http://esp32cam.local/?trigger-above=0.2&smooth=0.9&differ-by=20
```

and you will see the changes reflected in the configuration dumps. Now you simply have to alter the URL to try out new configurations until you're satisfied with the results.

A quick side note for self-promotion.

```
Reading how these things work is great, but *seeing* them is
even better. If you didn't purchased the video-content yet,
you should be convinced by now that my content is worth every
single cent, so don't waste time and go grab yours right now
at

https://eloquentarduino.gumroad.com/l/mastering-esp32-cam-pre-sale
```

```
(I will refund you the printed version if you bought it
already, just send me an email at support@eloquentarduino.com)
```