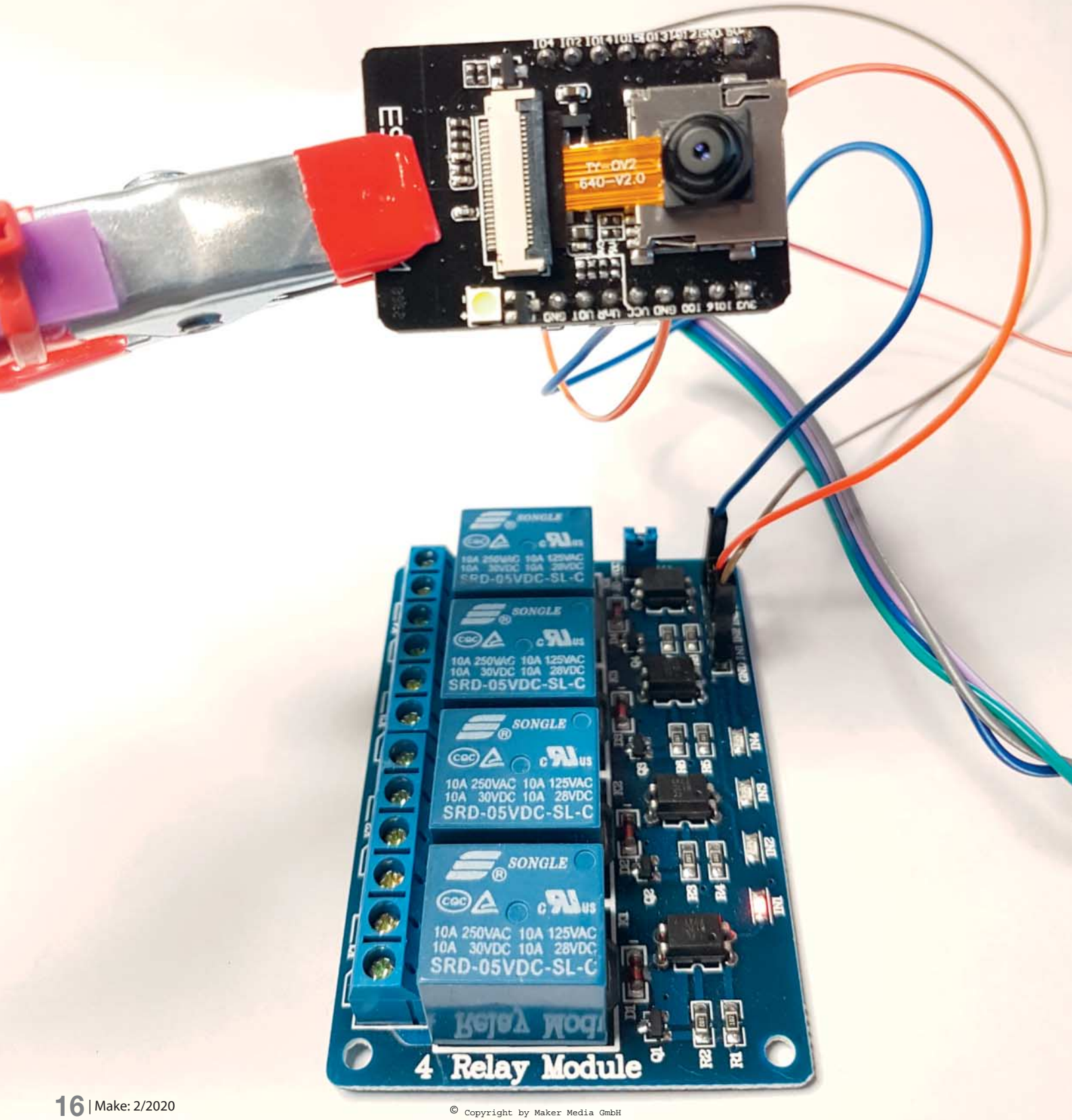
Gesichtssteuerung

Erkennt die ESP32-CAM ein Gesicht, auf das sie trainiert wurde, lassen sich weitere Aktionen auf dem Mikrocontroller starten. Per GPIO kann man beispielsweise ein Relais ansteuern, um einen elektrischen Türöffner zu bedienen.

von Daniel Bachfeld



Im letzten Heft haben wir bereits gezeigt, wie man den offiziellen Beispiel-Sketch „CameraWebServer“ für die Arduino IDE für das eigene WLAN anpasst und wie die Gesichtserkennung arbeitet. Zwar funktioniert das Antrainieren von bestimmten Gesichtern bereits mit diesem Sketch, allerdings gehen die gemerkten Gesichter beim Ausschalten verloren. Zudem kann man mit dem Sketch zunächst nichts steuern und nicht immer will man das Videobild per WLAN irgendwohin streamen. Unser Projekt soll ohne WLAN funktionieren, aber bei einem erkannten Gesicht einen GPIO-Pin steuern und ein daran angeschlossenes Relais schalten. Daran kann man weitere Verbraucher anschließen. Eine Anwendung wäre beispielsweise ein Türöffner, eine Entriegelung oder eine Steuerung im Smart Home für die Beleuchtung.

Für das permanente Speichern der antrainierten Gesichter muss man den Sketch eigentlich nur um drei Zeilen erweitern. Unpraktischerweise will die dem Sketch zugrunde liegende Gesichtserkennungssoftware die Daten jedoch nicht in der standardmäßig für solche Zwecke gedachten Datenpartition im Flash-Speicher des Boards ablegen. Vielmehr erwartet die Software eine spezielle Partition mit dem Namen *fr*. Eine Erklärung zu den normalen Partitionen des ESP32 finden Sie im Kasten „Partitionen“.

Die neue Aufteilung des Speichers soll wie in 1 aussehen. Die dazu passende CSV-Datei stellen wir im Github-Repository dieses Projekts zum Download bereit (siehe Link in der Kurzinfo). Die Datei müssen Sie im Ordner *C:\Users\Benutzername\AppData\Local\Arduino15\packages\esp32\hardware\esp32\1.0.4\tools\partitions* Ihres Windows-PCs speichern – wobei Sie *Benutzername* durch Ihren eigenen ersetzen. Nun müssen Sie die neue Partitionstabelle noch in die Arduino IDE integrieren. Die bereits verfügbaren Partitionstabellen sind mit allen möglichen ESP32-Boards in der Datei boards.txt gekoppelt, die im Ordner *C:\Users\Benutzername\AppData\Local\Arduino15\packages\esp32\hardware\esp32\1.0.4* liegt.

Mit einem Texteditor öffnet man die Datei und sucht nach den Definitionen für das verwendete AI Thinker Board (`esp32cam.name=AI Thinker ESP32-CAM`). Am Ende des Abschnitts für das Board (also vor der Hash-Reihe) fügt man die Zeilen wie in Listing 1 zu sehen ein. Keine Sorge, für Ungeübte haben wir die angepasste Boards-Datei im Github-Repo hinterlegt. Speichern Sie die Datei und starten Sie gegebenenfalls die Arduino IDE neu. Nun sollte unter dem Menüpunkt Werkzeuge beim ausgewählten Board „AI Thinker“ das neue Partitionsschema zu sehen sein 2.

Kurzinfo

- ESP32-Speicher partitionieren
- Bilder speichern
- GPIOs der ESP32-CAM steuern

Checkliste

Zeitaufwand:
1–2 Stunden

Kosten:
20–30 Euro

Programmieren:
Programmieren der Arduino IDE
Umgang mit Github

Alles zum Artikel im Web unter
make-magazin.de/xqf8

Material

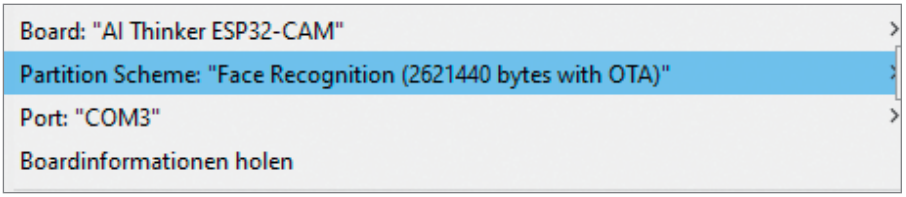
- ESP32-CAM (AI Thinker)
- USB-zu-Seriell-Konverter
- Jumperkabel female/female
- SaintSmart 4 Relay Module
- Stromversorgung

Mehr zum Thema

- Daniel Bachfeld, Intelligente Webcam für 5 Euro, Make 1/20, S. 28
- Daniel Bachfeld, Laserentfernungsmesser, Make 6/19, S. 28

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x5000,
otadata,  data, ota,     0xe000,  0x2000,
app0,     app,  ota_0,   0x10000, 0x280000,
fr,       32,   32,      0x290000, 0xEF000,
eeprom,   data, 0x99,    0x37f000, 0x1000,
spiffs,   data, spiffs,  0x380000, 0x2F000,
```

1 Die neue Partitionstabelle für die ESP32-CAM enthält nun auch einen Eintrag namens fr – für „face recognition“. Darin speichert der neue Sketch die aufgenommenen Bilder.



2 Dank der neuen Board-Definition bekommt die ESP32-CAM eine neue Partitionstabelle.

Listing 1

```
esp32cam.menu.PartitionScheme.partition=Face Recognition (2621440 bytes with OTA)
esp32cam.menu.PartitionScheme.partition.build.partitions=make_partitions
esp32cam.menu.PartitionScheme.partition.upload.maximum_size=2621440
```

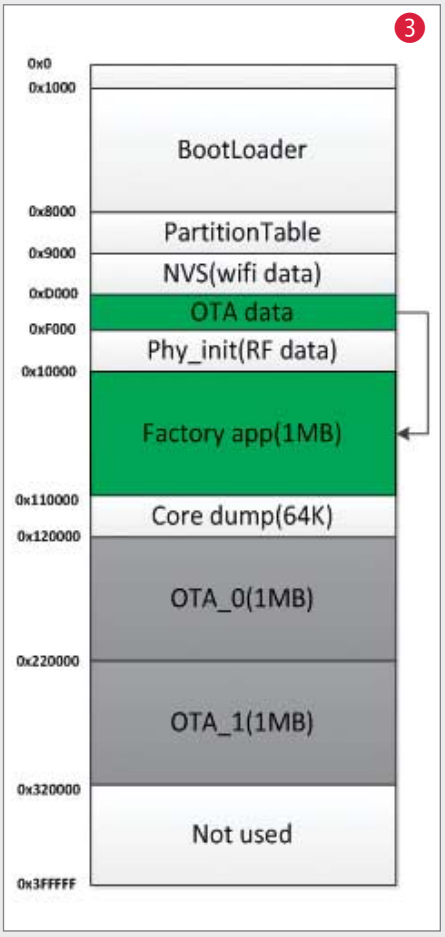
Partitionen

Der ESP32 hat einen eigenen minimalen Flash-Speicher, in dem ein Bootloader abgelegt ist. Dieser wird bei jedem Start oder Reset ausgeführt. Er initialisiert den Zugriff auf den externen (seriell angeschlossenen) zusätzlichen Flash-Speicher (SPI-Flash). Im Bedarfsfall schreibt er einen von außen übertragenen Sketch in den SPI-Flash. Anschließend ruft er den im SPI-Flash immer an der Adresse 0x1000 (hexadezimal) gespeicherten zweiten Bootloader auf. Dieser lädt zuerst die an Adresse 0x8000 gespeicherte Partitionstabelle. Darin sucht der zweite Bootloader nach Partitionen, die Programme (Apps) enthalten.

Üblicherweise haben ESP32-Module mit 4MB Speicher eine Aufteilung in 5 Partitionen, wie in 3 zu sehen. Die Partition *nvs* ist immer vorhanden und enthält wichtige Daten wie MAC-Adressen für WLAN, Bluetooth und den prinzipiell vorhandenen Ethernet-Anschluss. Im Normalfall lädt der Bootloader das in *Factory App* (manchmal auch *app0* genannt) gespeicherte Programm. Die Partitionen *OTA data*, *OTA_01* und *OTA_02* sind Speicherbereiche, um während des Betriebs neue Programme per WLAN herunterzuladen und zu installieren. Man spricht dabei auch von Over the Air Update (OTA) – man kennt das von vielen smarten Geräten daheim, bei denen im laufenden Betrieb neue Firmware installiert wird. Die Partitionen *OTA_01* und *OTA_02* nehmen die neue Firmware auf. Die Partition OTA data (Over the Air data) enthält nur einen Zeiger auf das auszuführende Programm. Der Bootloader startet dann nicht die Factory App, sondern diejenige, auf die der Zeiger verweist. So lässt sich leicht neue Firmware installieren, ohne dass man per USB etwas übertragen muss. Im Fehlerfalle ist sogar ein Rollback möglich.

Grundsätzlich muss man seinen Speicher aber nicht wie hier gezeigt aufteilen. Wer große Programme hat, kann auch nur eine große Factory-App-Partition auswählen. Weitere Vorschläge zum Partitionieren blendet die Arduino IDE zu jedem Board unter dem Punkt „Partition Table“ ein. Das Erstellen der Partitionen übernehmen die Tools von Espressif quasi automagisch, sobald man einen Sketch mit den gewählten Einstellungen auf den ESP32 hochlädt.

Die von Espressif vordefinierten Partitionstabellen findet man als CSV-Dateien unter Windows im Verzeichnis *C:\Users\Benutzername\AppData\Local\Arduino15\packages\esp32\hardware\esp32\1.0.4\tools\partitions*. Sie lassen sich mit jedem Texteditor, Excel oder LibreOffice Calc öffnen und bearbeiten.




```
ESP32-Face | app_httpd.cpp | camera_index.h
#include "esp_camera.h"
#include "img_converters.h"
#include "camera_index.h"
#include "Arduino.h"

#include "fb_gfx.h"
#include "fd_forward.h"
#include "fr_forward.h"
#include "fr_flash.h"

#define ENROLL_CONFIRM_TIMES 5
#define FACE_ID_SAVE_NUMBER 7
```

4 Insgesamt hat die Kamera 7 Plätze für verschiedene Gesichter mit jeweils 5 Aufnahmen.

Sketch anpassen

Sofern Ihre ESP32-CAM noch nicht mit der richtigen Software ausgestattet ist, laden Sie in der Arduino IDE den Beispielsketch CameraWebServer oder den leicht angepassten aus unserem Github. Beide unterscheiden sich nur in der Art der Vergabe der IP-Adresse. Nun müssen wir dem Sketch beibringen, die trainierten Bilder in der Partition *fr* abzulegen.

Wechseln Sie auf den Reiter „app_httpd.cpp“ und fügen Sie, wie in 4 zu sehen, die Zeile

```
#include   fr_flash.h
```

ein. Suchen Sie in diesem Reiter nun nach folgendem Aufruf

```
int8_t left_sample_face =
enroll_face(&id_list,
aligned_face);(&id_list,
aligned_face);
```

und ändern die benutzte Funktion `enroll_face` in `enroll_face_id_to_flash`. Damit legt das Programm die Gesichterbilder im Flash ab. Schließlich suchen Sie noch nach der Funktion `face_id_init` und fügen darunter die neue Zeile

```
read_face_id_from_flash(&id_list);
```

hinzu. Zu Ihrer Sicherheit finden Sie den Sketch („ESP32-Face“) mit den angepassten Zeilen zum Download in unserem Git-hub-Repo.

Nun übersetzt man den Sketch wie gewohnt (Board: AI Thinker) und lädt ihn per USB-zu-seriell-Wandler auf das Board (IO1 auf GND legen!). Die Verkabelung finden Sie in 5. Nach dem Upload trennen Sie die Verbindung von IO1 und GND und resetten die CAM. Anschließend rufen Sie die Weboberfläche der CAM unter der vergebenen IP-Adresse auf.

Training

Stellen Sie die Kameraauflösung auf CIF und starten den Stream. Nun schalten Sie „Face Detection“ und „Face Recognition“ an. Zum Anlernen des Gesichts klicken Sie auf „Enroll Face“ und schauen dabei – ausreichend gute Lichtverhältnisse vorausgesetzt – in die Kamera. Diese umrahmt Ihr Gesicht mit einem hellblauen Kasten und macht 5 Aufnahmen (Sample[1..5]) 6. Damit wäre Ihre Kamera auf Ihr Gesicht trainiert.

Für die weitere Anwendung zur Gesichtssteuerung benötigen wir das WLAN und die Weboberfläche nicht mehr. Laden Sie nun den Sketch „Face-Relais“ aus dem

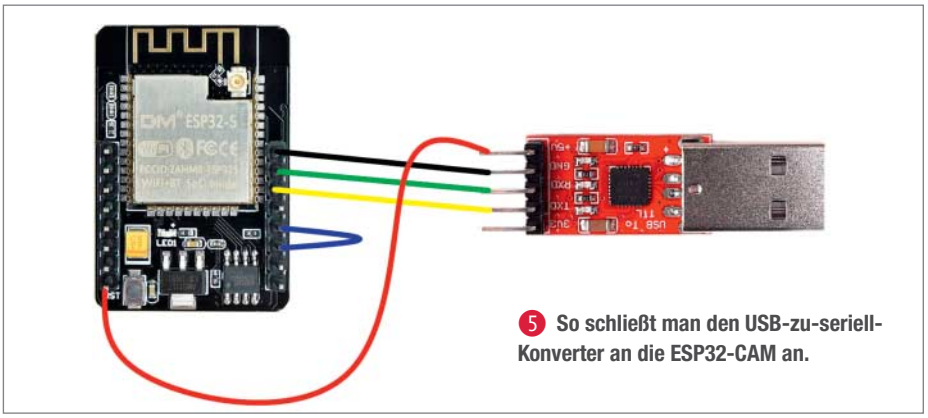
Github-Repo in die Arduino IDE. Fürs Protokoll: Der Sketch ist an das Beispiel des Blogs „Robot Zero One“ angelehnt, das allerdings bei uns nicht funktionierte. Wir haben es so angepasst, dass es auf einer ESP32-CAM vom Typ AI Thinker funktioniert. Es schaltet bei einem erkannten Gesicht den GPIO4 auf logisch High. Praktischerweise hängt an diesem Port auch die weiße LED, die dann leuchtet.

Übersetzen Sie den Sketch und laden ihn auf das Board (IO1 an GND und anschließend wieder abziehen nicht vergessen) und resetten es. Sobald Ihr Gesicht erkannt wird, geht die LED an. Drehen Sie Ihr Gesicht weg, erlischt die LED nach wenigen Sekunden wieder. Für erste Funktionstests reicht das aus. In der Praxis hat man davon jedoch wenig Nutzen und außerdem blendet die LED ziemlich stark. Deshalb steuern wir lieber ein an GPIO2 angeschlossenes Relais an, zum Beispiel das 4-fach-Relais, das man bei so gut wie jedem Gadget-Händler bekommt.

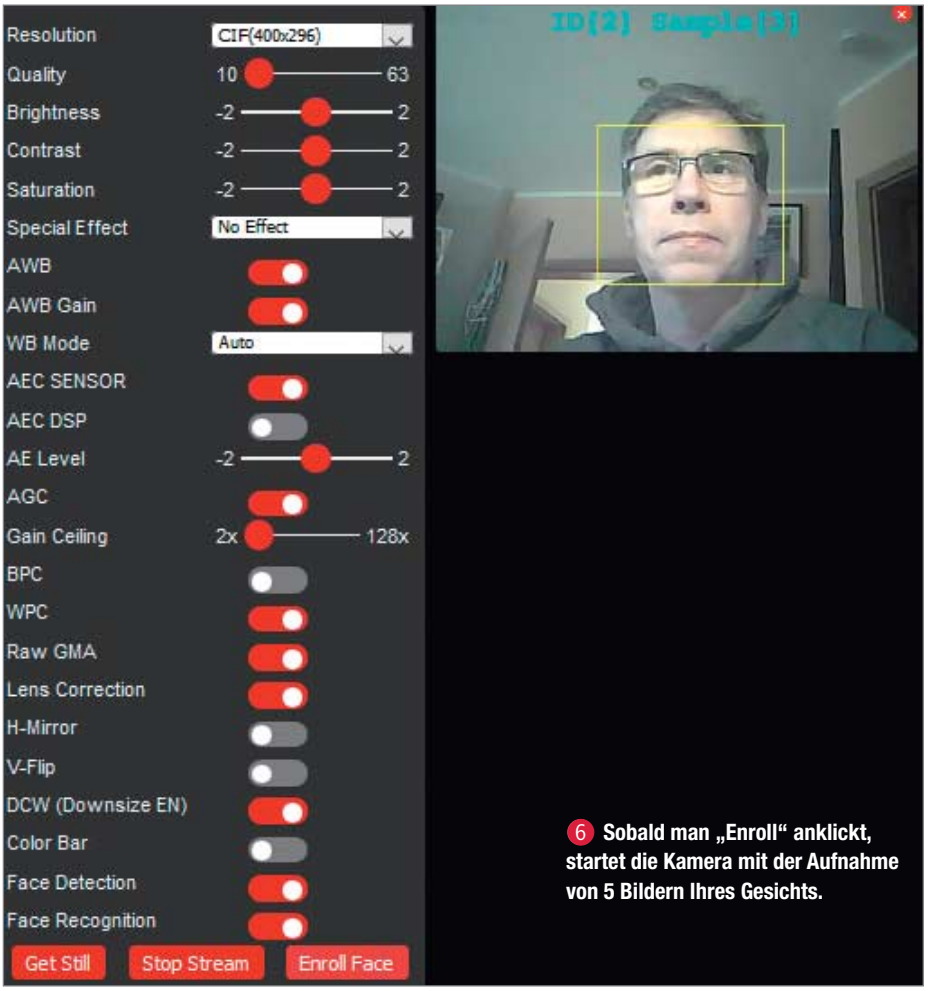
Verkabeln Sie das Relais gemäß 7. Sie können den USB-zu-seriell-Wandler zur Stromversorgung des Boards und des Relais auch weiter angeschlossen lassen. Praktischerweise hat die ESP32-CAM einen Ausgang, um andere Schaltungen mit Strom zu versorgen. Daran können wir das Relais anschließen – im Verkabelungsplan ist das bereits umgesetzt. Standardmäßig ist der Ausgang auf 3,3V eingestellt, was für das Relais ausreicht, um es zu schalten. Wer mit anderen Relais-Modellen doch 5V benötigt, muss auf dem Board der ESP32-CAM einen Jumper umlöten.

Um statt mit GPIO4 den Ausgang mit GPIO2 zu schalten, müssen Sie zunächst in der Zeile am Anfang des Sketches `#define relayPin 4` die 4 in eine 2 ändern, neu kompilieren und erneut den Sketch hochladen. Statt der LED schaltet nun das Relais. Beachten Sie, dass das Relais konstruktionsbedingt dann anzieht, wenn an seinen Eingängen (IN1..4) ein Low-Pegel anliegt. Sie erkennen das unter anderem auch daran, dass die Eingangs-LED leuchtet. Sobald GPIO2 auf High geht, fällt das Relais ab und die LED erlischt. Der Steuerung externer Verbraucher auf der Ausgangsseite des Relais tut das keinen Abbruch. Das Relais hat drei Anschlüsse, die als Umschalter ausgelegt sind. Messen Sie mit einem Ohmmeter durch, welche Anschlüsse bei „GPIO2 high“ offen und welche geschlossen sind.

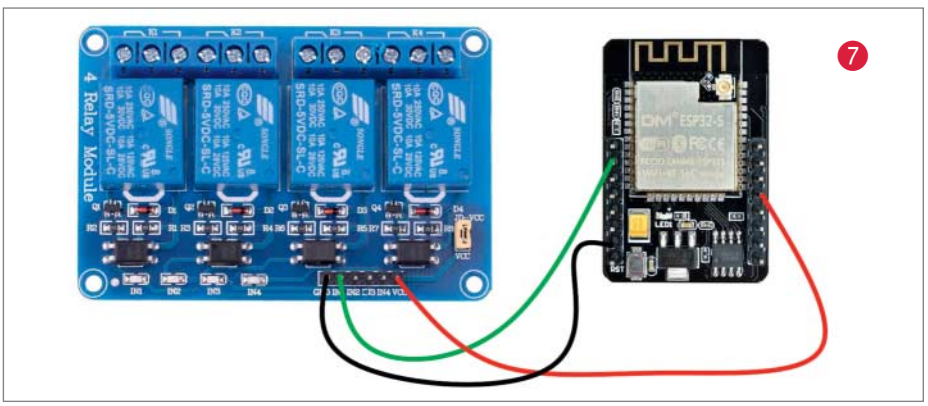
Sie können das hier gezeigte Projekt beispielsweise zur Betätigung eines Türsummers oder zum Aktivieren irgendwelcher Schalter einsetzen. Denkbar ist auch, behinderten Menschen das Steuern von Geräten oder Anlagen auf diesem Weg zu erleichtern. Schreiben Sie uns gerne eine Mail, wie und wo Sie das Projekt eingesetzt haben. *—dab*



5 So schließt man den USB-zu-seriell-Konverter an die ESP32-CAM an.



6 Sobald man „Enroll“ anklickt, startet die Kamera mit der Aufnahme von 5 Bildern Ihres Gesichts.



7