ESP32-CAM as a programmable camera research platform

Henry Dietz, Dillon Abney, Paul Eberhart, Nick Santini, William Davis, Elisabeth Wilson, and Michael McKenzie; University of Kentucky; Lexington, Kentucky

Abstract

Experimenting with custom-programming of cameras can be difficult. Most consumer cameras are protected to prevent users from reprogramming them. Industrial cameras can be flexibly controlled by an external computer, but are generally not standalone programmable devices. However, various inexpensive camera modules, designed largely to be used for building IoT (Internet of Things) devices, combine extensive programmability with a camera in a compact, low-power, module. One of the smallest and least expensive, the ESP32-CAM module, combines a 2MP Omnivision OV2640 camera with a dual-core 32-bit processor, 802.11 WiFi and BlueTooth as well as wired I/O interfaces, a microSD slot, low power modes, etc., all supported by the Arduino programming environment and a rich collection of open source libraries. Why not use it for programmable camera research?

This paper describes how the ESP32-CAM had to be adapted to enable use in a variety of experimental cameras. For example, some of these cameras do not use the lens screwed and glued onto the OV2640, and replacing this lens revealed a number of issues ranging from spectral response to adjustment of lens corrections. There are numerous strange interactions between different functions that end-up sharing the same I/O pins, so work-arounds were needed. It also was necessary to devise ways to handle various higher-level issues such as implementation of a live view and synchronization across cameras. However, the key problems have been resolved with open source software and hardware designs described here.

Introduction

A wide range of digital cameras are readily available, but very few make it easy to conduct experimental camera research requiring custom programming.

The most obvious options would be high-end consumer cameras, especially mirrorless interchangeable-lens cameras using MFT, APS-C, or full-frame sensors. Although most consumer cameras employ a variety of mechanisms to discourage users from reprogramming them, some open-source efforts have managed to reverse-engineer ways to support 3rd-party programming. Magic Lantern[1] was successful in adding programmability and various features to several Canon EOS DSLRs and the original EOS M mirrorless, but porting to new models is difficult, and none of the latest models are supported. The OpenMemories[2] project enables 3rd-party software to run in the protected Android camera app environment that Sony created for their PlayMemories apps. Sony supported that app environment in a wide range of models, from digital video cameras to the high-end full-frame 42MP A7r II, but discontinued support of Android apps in all new cameras immediately after OpenMemories became viable.

Cell phones and compact consumer cameras also would be obvious alternatives using smaller sensors. Most cell phones offer excellent programmable camera support via either Android or IOS, very similar to the API developed for the research platform Frankencamera[3][4], but the cost is high for the image quality delivered, and the cameras are essentially unalterable sealed subsystems. Compact cameras are much cheaper while offering somewhat better image quality, but only the reverse-engineered open source Canon Hack Development Kit (CHDK)[5] offers the desired level of programmability. At this writing, CHDK supports 160 different low-end Canons, including most PowerShot models and a few EOS M series mirrorless bodies; for more than a decade, CHDK has been one of the most viable approaches for experiments involving reprogramming of cameras.

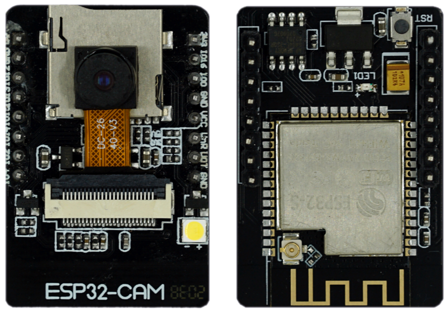


Figure 1. *ESP32-CAM actual size: front and rear views*

Cameras designed to be tethered to a computer do not offer programmability directly, but can be intelligently controlled by a host computer program. Although various interfaces are available, most such cameras use USB, often leveraging the driverless UVC (USB video class) protocol[6]. Consumer-oriented tethered cameras are generally marketed as webcams, whereas those marketed as industrial or machine vision cameras tend to have more robust physical construction, accept C-mount interchangeable lenses, and may offer additional features. Due to the limited market for industrial cameras, cost tends to be higher than for consumer cameras with similar specifications.

There have been a few commercial cameras designed for programmability, but they tend to be relatively expensive. The DevCAM[7] is a very promising open source design incorporating an FPGA to support research use of up to six MIPI sensor modules, but it also is relatively expensive per camera.

The alternative explored in the current work leverages the large market that has developed for inexpensive camera modules intended to be used for building IoT (Internet of Things) devices. These camera modules often are paired with fairly substantial microcontrollers, and supported as platforms for both hobbyist/experimenter use and IoT product development. On the hobbyist/experimenter-oriented high end are boards providing a full Linux environment, such as NVIDIA's Jetson[8] and various

Table 1: Potential Camera Platform support of Key Attributes and Features

Attribute or Feature	Mirrorless	Webcam	CHDK PowerShot	AI-Thinker ESP32-CAM
Image quality	$\geq$20MP,$\geq$12bpp	$\geq$0.3MP,$\geq$8bpp	20MP, 12bpp	2MP (1600$\times$1200), 10bpp
Exposure control	Extensive	Basic	Extensive	Basic plus some features
Interchange lens	Yes	Some models	No	Simple modification
Sensor size	$\geq$17.3$\times$13mm	$\geq$2.4$\times$1.8mm	$\sim$6$\times$4.5mm	3.59$\times$2.684mm
Near infrared (NIR)	Hard mod	Some models	Very hard mod	Simple mod
Wired connectivity	USB	USB	USB	UART, SPI, & I2C
Wireless	WiFi	No	Some models	WiFi & Bluetooth (with BLE)
Tethered control	Proprietary	Yes, UVC	Yes, CHDK PTP	Yes, programmable
Autonomous operation	Very limited	No	Yes	Yes
Programmable display	No	No	LCD	Options via connectivity
Programming support	No	No	CHDK C & Lua	Arduino & Espressif C/C++
Processor	Various ARM	?	Dual 80MHz ARM	Dual 240MHz Xtensa
Usable main memory	Varies	None	Several MB	520KB SRAM & 4MB PSRAM
Flash memory	SD card	No	SD card	4MB Flash & TF card
Power management	Minimal	No	Minimal	Modes from 310mA to 6μA
Sensor inputs	Camera UI	No	Camera UI	9 I/O pins; ADC, I2C, & SPI
Control outputs	Flash sync.	No	No	9 I/O pins; PWM, I2C, & SPI
Real time sync support	Remote	Some models	RTC, USB detect	RTC, programmable sync
Ease of embedding	Very hard	Moderate	Hard	Easy: 27x40.5x4.5mm board
Cost	$\geq$ \$500	\$8 – \$150	$\geq$ \$100	$\sim$ \$7

Raspberry Pi models[9]. On the low end are the various ESP32-CAM[10][11] versions, such as the AI-Thinker version used in the projects discussed here. Despite being one of the least expensive modules at around \$7, the ESP32-CAM combines a 2MP Omnivision OV2640[12] camera with substantial computing facilities centering on a dual-core 32-bit processor supported by the Arduino programming environment and a multitude of libraries. For example, a demonstration application implements a WiFi camera web server with real-time face recognition.

Since 1999, our research group has constructed many programmable cameras and camera arrays[13]. Table 1 gives a compact comparison of relevant features of some camera systems that we have used to construct programmable camera research platforms. From the table, it is not surprising that most of our earlier efforts used Canon PowerShot cameras under CHDK. However, the combination of lower resolution for the OV2640 camera, relatively powerful computing hardware resources, and extensive library support makes the ESP32-CAM capable of implementing significantly more sophisticated custom programming.

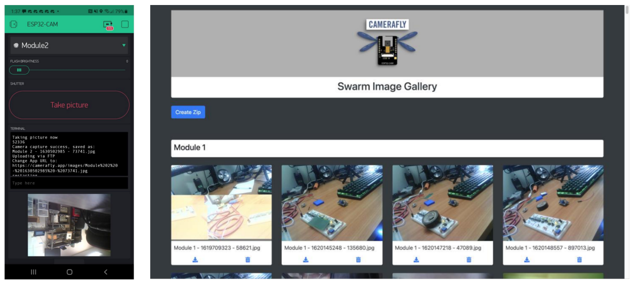
The remainder of this paper demonstrates the suitability of ESP32-CAM for such research use by exploring three very different programmable camera systems our group has constructed since 2019: KameraflY, KISS-E, and Lafodis.

KameraflY Wireless Multicamera

The concept behind KameraflY was to create a very flexible synchronized wireless multicamera that would allow a swarm of stand-alone camera modules to be freely positioned for panoramas, 3D capture, Matrix-style time-frozen fly-bys, etc. The system was to be as inexpensive as possible, with a target cost of under \$25 per stand-alone camera. Yet, it was to be able to support tightly synchronized capture using dozens of cameras. In fact, this description is essentially the charge that was given to the undergraduate Senior Design team of Nick Santini, William Davis, Elisabeth Wilson, and Michael McKenzie – shown in Figure 2 with the working system they built[14].

Figure 2. KameraflY Senior Project Team and system demonstration

Precise synchronization is critical for multicamera applications, so the team built a LED synchronization tester and experimented. Synchronization proved easier than expected, with WiFi synchronization of the cameras to 10ms – shorter than the electronic rolling shutter traversal time of an OV2640. The swarm cameras were programmed to work with the Blynk IoT smartphone app[15] for capture to a TF card, image upload via FTP,


ESP32-CAM
Module2
FLASH BRIGHTNESS
SHUTTER
Take picture
TERMINAL
Type here
CAMERAFLY
Swarm Image Gallery
Create Zip
Module 1
Module 1 - 1619709323 - 58621.jpg
Module 1 - 1620145248 - 135680.jpg
Module 1 - 1620147218 - 47089.jpg
Module 1 - 1620148557 - 897013.jpg

```
Taking picture now
52336
Camera capture success, saved as:
Module 2 - 1630502985 - 73741.jpg
Uploading via FTP
Change App URL to:
https://camerafly.app/images/Module%202%20
-%201630502985%20-%2073741.jpg
```

Figure 3. *KameraflY control app and ftp image gallery*

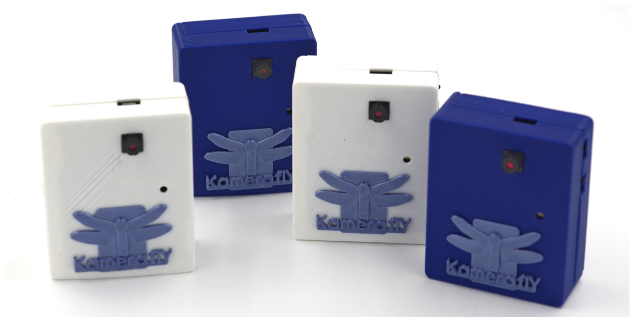


Figure 4. *Swarm of four stand-alone KameraflY modules*

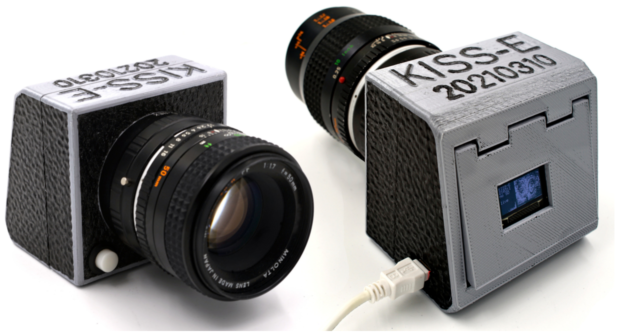


Figure 5. *KISS-E front and rear views*

and live streaming of video. A PHP web app allows browsing the images collected from the multicamera. The two apps are shown in Figure 3.

Although bare ESP32-CAM can be used with the software, making a stand-alone KameraflY module requires a bit more. Making self-contained units able to run off either USB power or an internal rechargeable battery was perhaps the largest challenge, especially since battery safety was a priority. The 3D-printed housing and custom board the team made neatly packaged the ESP32-CAM with a charging module and a BP-70A 3.7v Lithium-Ion battery – a battery used in many Canon PowerShot models. The KameraflY modules shown in Figure 4 actually came-in well under budget, at a total cost of $13.43 each.

KISS-E Interchangeable-Lens Camera

While KameraflY modules are standalone cameras, they are designed to be programmed to be either fully autonomous or remote controlled via WiFi. For example, there is no shutter button and the camera has no display to show images nor even camera status. In contrast, KISS-E, shown in Figure 5, fully supports tethered control via either USB or WiFi, but also implements the basic features needed to directly operate the camera. However, the most important aspect of KISS-E is that it is designed to accept any of a wide range of interchangeable lenses. KISS-E stands for Kentucky's Interchangeable-lens Small Sensor – E mount; the OV2640 lens is removed and any lens adaptable to Sony E mount may be used instead.

Why Sony E mount? Most small-sensor cameras with interchangeable lenses use C mount. However, Sony's mirrorless cameras have made the larger-diameter E mount the most popular target for adapting lenses, so inexpensive adapters for almost any lens (including C mount) are widely available. Compared to full-frame 36×24mm Sony cameras, the tiny OV2640 sensor imposes a crop factor of approximately 9.7; that makes it hard to get a wide angle view, but a 50mm lens essentially becomes 485mm, giving very long effective focal lengths and high magnifications with cheap and compact lenses. This is somewhat problematic for hand-held use, because any camera shake is greatly magnified, but there is a standard tripod 1/4-20 screw thread on the base of KISS-E.

The electronics, wiring, and 3D-printed parts of KISS-E are shown in Figure 6. Total parts cost was about $14. Unlike KameraflY modules, there is no custom board in KISS-E. Instead, the 3D-printed KISS-E body incorporates structures to mechanically hold the ESP32-CAM, FT232RL, SSD1306 boardlets and shutter button, which additionally can be secured using hot glue. In fact, 3D printers even can print custom circuit boards – they simply cannot print wiring. Thus, combining 3D-printed mechanical mountings with electrical connections implemented by wire wrap directly to boardlet pins provides remarkable design flexibility. Note that the rear OLED mount is a printed-assembled hinged door that can be pivoted up for waist-level live view and also serves as an access to the camera internals, including the TF card. The small rectangular object on the 3D model plate is a tool for removing/inserting a TF card.

The software written for KISS-E is very flexible, allowing a mix of WiFi, USB, and manual control. Most significantly, KISS-E fully supports real-time live view. Live view is a feature digital cameras generally provide, but there is actually some complexity in managing the OV2640 to provide a fast frame rate live view stream while allowing on-demand image captures at a higher resolution. KISS-E can deliver the live view stream via WiFi and also via the built-in rear OLED display.

The SSD1306 OLED display is bright and costs just $3, but it is tiny, only 128×64 pixels resolution, and each pixel is strictly on or off with no colors nor gray levels. Despite those restrictions, we were able to write special software to produce the live display seen in Figure 7. Of course, bigger and better displays are available, but in addition to cost, better displays would have implied significantly more compute overhead, lowering camera responsiveness. The OLED display shows:

- KISS-E version ("210716" in Figure 7)
- Percentage of available TF card space used... or, in Figure 7, the fact that no TF card was present
- The last part of the current WiFi address (.21) and a single character indicating WiFi status ("-" means on, no request being processed)
- The current camera state ("live" meaning showing live view)

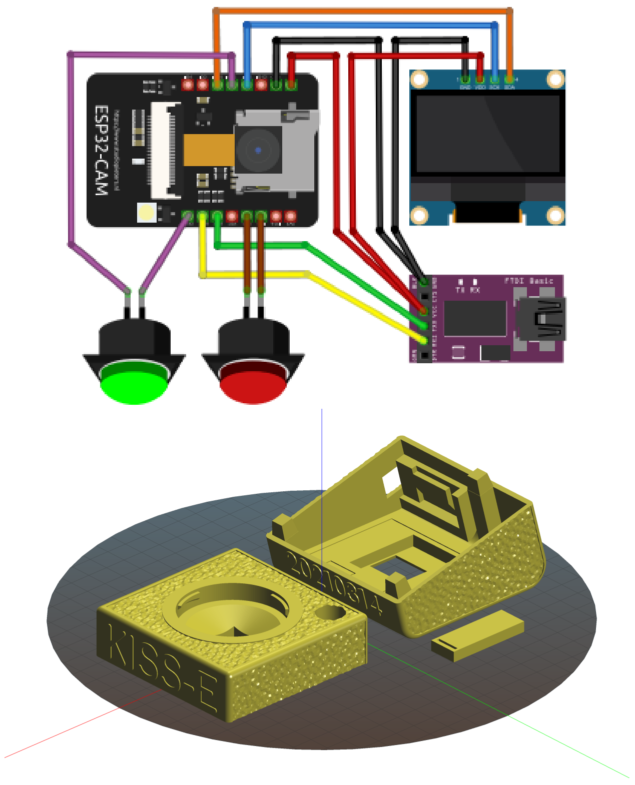


Figure 6. *KISS-E component wiring and 3D printed parts*

- A live histogram of pixel values
- An 80×60 pixel live view image produced by a special rendering algorithm that combines error distribution with edge enhancement to make the scene as recognizable as possible
- A new type of focus quality indicator (below the live view): the gap between two horizontal lines gets smaller as manual focus becomes sharper

The focus quality indicator is particularly important because the poor quality of the live view makes it insufficient to check focus, but focus is critical with typical lenses due to the very thin depth of field. Empirically, this line-gap focus aid based on contrast detection in the center of the frame is very effective.

A complication is that, not only does removing the (fixed focus) native lens from the OV2640 make it possible to use a different lens that requires manual focusing, but it also has the side effect of removing the NIR-blocking coating that is on the native lens. Without that filtration, the camera responds strongly to a full spectrum of light starting in the NIR region. Figure 8 shows the classic NIR-dominated colors of an outdoor scene shot without an NIR-blocking filter. Although an appropriate filter can be fitted to the front of a lens, large NIR filters are not cheap; the best alternative was thus to 3D-print a holder that can use $8 \times 8mm$ 650nm NIR-cut filters which cost under $1, as shown in Figure 8. These filters work well, but it is significant that the holder covers the edges of the filter glass, because they otherwise caused strange flare/tinting of the image.

There are a number of other minor issues worth noting in the construction of KISS-E:

- Although KISS-E provides a USB interface that can be used for providing power, ESP32-CAM programming, or tethered control, it currently cannot support the driverless UVC protocol to behave as a webcam
- The ESP32-CAM as mounted in KISS-E relies on the antenna built-into the boardlet, which easily can suffer interference if wire-wrap wires are routed over the antenna
- There are actually at least two different versions of the OV2640 supplied with the ESP32-CAM. Although functionality is very similar, the sensor is rotated 90° in one relative to the other
- Removing the standard lens from the OV2640's plastic

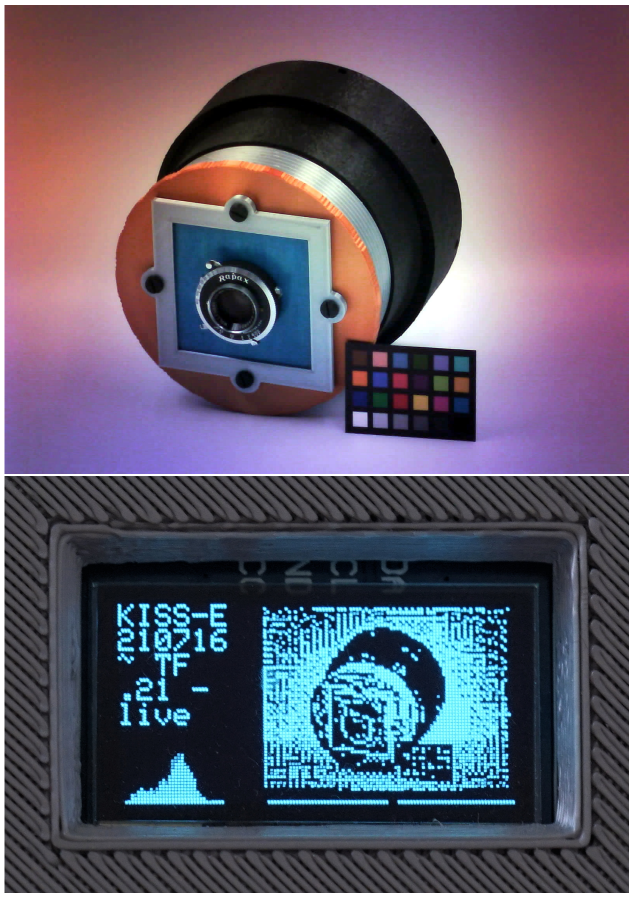


Figure 7. *KISS-E capture of Lafodis and corresponding OLED live view*

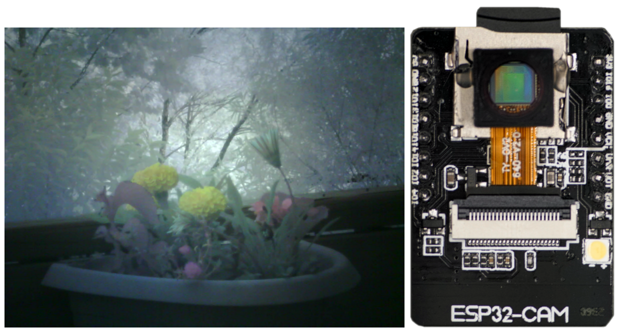


Figure 8. *KISS-E image captured without NIR filter, $8 \times 8mm$ filter mount*

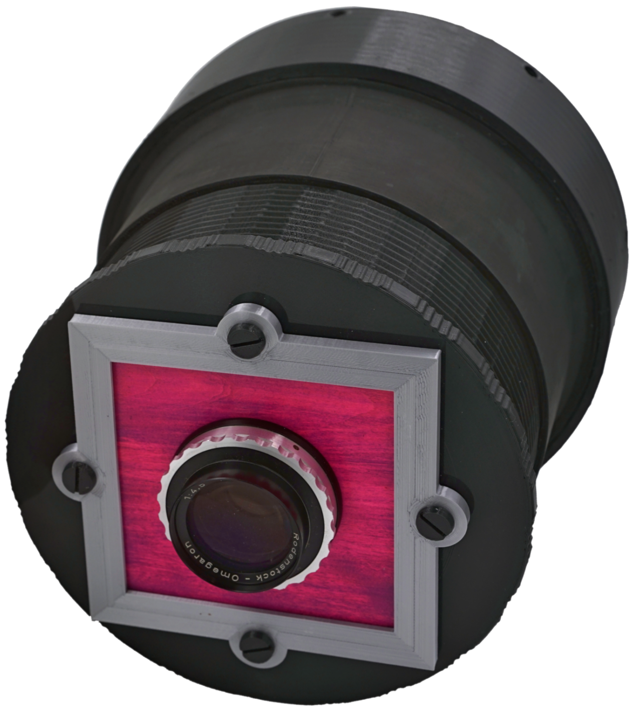


Figure 9. *Lafodis large-format wireless IoT camera*

screw mount is complicated by the fact that it is glued in place
- The software is more convoluted than one might expect due to multiple functions sharing I/O pins

In sum, KISS-E is a very poor replacement for a Sony E-mount camera, but it is highly functional and fully programmable.

Lafodis Scanning Camera

Lafodis (LArge FOrmat DIgital Scanning) is a large-format camera that can scan a 160mm diameter lens-projected image to resolutions of approximately 2.6GP. Originally intended purely as a test platform for research investigating novel scan orderings[16], it became clear that such a device could be useful to others. Lafodis has thus far evolved through four prototypes with the design goal of making it easy for others to build this sub-$50 camera to capture $\geq$500MP images from a 4x5" sensor area. The third prototype is shown in Figure 9.

Unlike KameraflY and KISS-E, Lafodis does not just use the ESP32-CAM as a programmable camera, but also as a microcontroller to control the physical scanning. The ESP32-CAM is mounted on a moving platform via a traceless 3D-printed circuit board mount, and it orchestrates the rotational and radial movement of that platform using a pair of 28BYJ-48 steppers with ULN2003 driver boardlets. The stepper and driver boardlet combinations cost just $3 each because they are a standard part widely used in HVAC, vending machines, etc. There are 32 full steps per rotation of the motor, but internal gears drop the speed by a factor of $32/9 \times 22/11 \times 26/9 \times 31/10 = 63.68395$, giving approximately 2037.8864 full steps per rotation of the drive shaft. This gives the steppers substantial torque even when stepping at a peak rate of over 1000 steps/second. Each step is done using

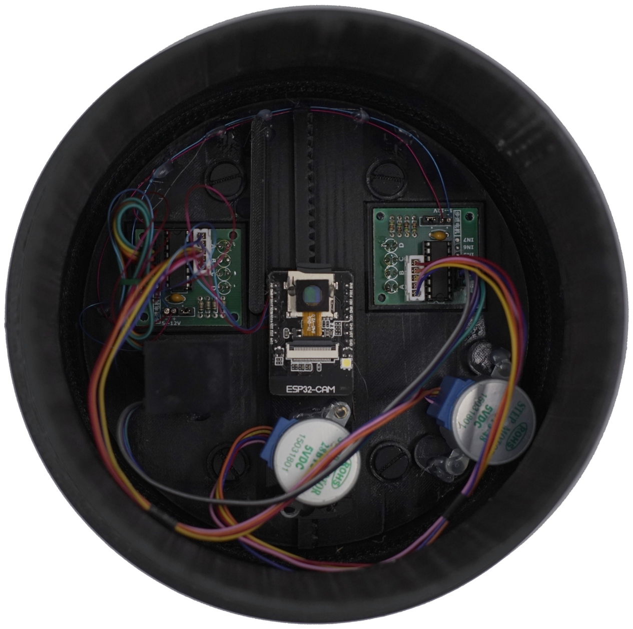


Figure 10. *Inside Lafodis: electronics and Herringbone gear drive*

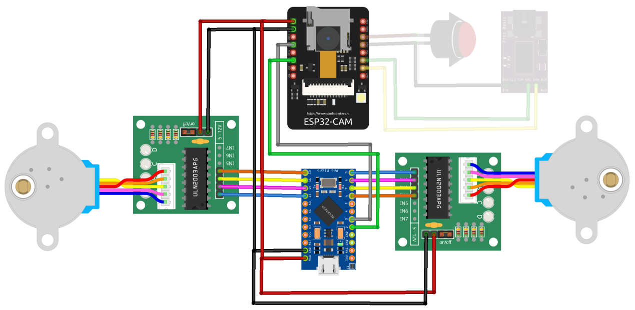


Figure 11. *Circuit design for 3rd and 4th Lafodis prototypes*

four control signals to select which coils should be energized, and the ULN2003 driver simply provides Darlington transistor pairs to switch power.

In the first two Lafodis prototypes, the 8 stepper control signals were directly taken from 8 I/O pins on the ESP32-CAM. The catch is that the various restrictions on how pins can be used meant that only 7 appropriate digital outputs were available on boardlet pins – a wire had to be soldered to the pad normally driving the red LED on the back of the ESP32-CAM boardlet for the eighth. Although that was fully functional, we felt that the soldering needed could discourage others from building Lafodis, so the third and fourth Lafodis prototypes use an entirely different method: I2C communication with a second microcontroller dedicated to stepper control.

The second microcontroller is an Arduino Pro Micro (actually, a clone which cost us $1.25; pricing is now closer to $7), and it conveniently also provides a USB connector that can be used to externally power Lafodis. The wiring diagram is shown in Figure 11, and again all components are mechanically held by 3D-printed mounts and electrically connected using standard cables or wire wrap – without needing a custom board. This I2C solution is very scalable; not only does the second microcontroller provide many more I/O pins, but the I2C network can be used to connect up to 128 devices. I2C devices we have used in other experimental systems range from an AMG8833 thermal camera to an MPU-9250 9-axis inertial measurement unit – as well as displays like the SSD1306 OLED used in KISS-E. The only negative issue we found in using I2C for Lafodis is that the electrically noisy environment within the camera tended to cause (rare) transmission errors between the ESP32-CAM and Pro Micro. These errors were essentially eliminated by building a simple higher-level protocol on top of the I2C transport to detect and correct errors.

Thanks to the I2C offloading, the third and fourth Lafodis prototypes leave enough pins open on the ESP32-CAM to allow for USB reprogramming of the processor by adding the circuitry shown as a ghosted image in Figure 11. However, the ESP32-CAM also allows for OTA (over the air) reprogramming via WiFi, and all our Lafodis software takes advantage of that. The only complication is that some chip resources must be reserved to support OTA, so the maximum application code size is reduced to 1.9MB.

Conclusion

The construction of programmable cameras for research purposes requires an appropriate camera platform. The one-line summary of the findings in this paper is that cameras designed for IoT applications generally, and the ESP32-CAM in particular, are a good match to typical research system requirements. A 2018 paper[13] summarized the key requirements for multicamera research systems as:

- Programmable camera modules: it is difficult to find a better-supported programming environment than the combination of Espressif ESP32-specific libraries and the Arduino infrastructure; there are over 50 ESP32-specific example applications in the Arduino programming environment (and dozens more generic Arduino application examples that also can run on ESP32-CAM)
- Synchronization of local clocks: in addition to a multitude of hardwired mechanisms, ESP32-CAM can use NTP to synchronize clocks with internet time references
- Local storage and processing: compared to the resolution of the camera, the RAM and non-volatile storage available are quite large, and the combination of the OV2640's ability to do JPEG and other processing reduces load on the dual-core 32-bit 240MHz Tensilica Xtensa LX6 ESP32 processor
- Physical mounting and alignment: in addition to the designs detailed here, there are 259 open source 3D-printed designs on Thingiverse involving use of ESP32-CAM
- Live view: although not directly provided by ESP32-CAM, we developed reasonable support for live view in the KISS-E project – and now it can be used as a library for other projects
- Fault tolerance: ESP32-CAM is cheap enough to have multiple spares

Beyond those concerns, KameraflY, KISS-E, and Lafodis all clearly demonstrate extensive support for connectivity – especially wireless – which makes many research tasks easier. Finally, Lafodis shows that the ESP32 is also a good general-purpose microcontroller, able to implement a wide range of real-time control and sensing tasks in addition to managing an OV2640 camera.

All the work discussed in this paper, and additional materials, will be linked into:

`http://aggregate.org/DIT/ESP32CAM`

Note that the authors have no affiliation with any entities involved in producing nor selling any of the various versions of the ESP32-CAM, nor does our research group speak for the University of Kentucky in advocating use of this platform.

References

[1] Magic Lantern, *http://www.magiclantern.fm/* (accessed 1/28/2022)

[2] OpenMemories, *https://github.com/ma1co/OpenMemories-Tweak* (accessed 1/28/2022)

[3] Marc Levoy. "Experimental platforms for computational photography," IEEE Computer Graphics and Applications 30, no. 5, pp. 81-87, 2010

[4] Andrew Adams, et al. "The Frankencamera: an experimental platform for computational photography," Commun. ACM 55, 11 (November 2012), DOI: 10.1145/2366316.2366339, pp. 90-98, 2012

[5] Canon Hack Development Kit (CHDK), *http://chdk.wikia.com/wiki/CHDK* (accessed 1/28/2022)

[6] Hans van Antwerpen et al. "Universal Serial Bus Device Class Definition for Video Devices," USB Implementers Forum, Inc., revision 1.5, August 9, 2012

[7] Dominique Meyer, Meher Birlangi, and Falko Kuester. "DevCAM: An open-source multi-camera development system for embedded vision," Electronic Imaging 2021, Image Sensors and Systems 2021, January 20, 2021

[8] NVIDIA, "Advanced AI Embedded Systems," *https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/*, (accessed 1/28/2022)

[9] "Introducing the Raspberry Pi Cameras," *https://www.raspberrypi.com/documentation/accessories/camera.html*, (accessed 1/28/2022)

[10] Sara Santos, "ESP32 Camera Dev Boards Review and Comparison (Best ESP32-CAM)," *https://makeradvisor.com/esp32-camera-cam-boards-review-comparison/*, (accessed 1/28/2022)

[11] Random Nerd Tutorials. "ESP32-CAM Camera Boards: Pin and GPIOs Assignment Guide," *https://randomnerdtutorials.com/esp32-cam-camera-pin-gpios/*, (accessed 1/28/2022)

[12] OmniVision. "OV2640 Color CMOS UXGA (2.0 MegaPixel) CameraChip^{TM} with OmniPixel2^{TM} Technology," Version 1.6, February 28, 2006

[13] Henry Dietz, Clark Demaree, Paul Eberhart, Chelsea Kuball, and Jong Yeu Wu. "Lessons from design, construction, and use of various multicameras," Electronic Imaging 2018, Photography, Mobile, and Immersive Imaging, pp. 182-1-182-10(10), 2018

[14] Nick Santini et al, "KameraflY ESP32-CAM Camera Swarm System," *https://github.com/npsantini/KameraflY-ESP32-CAM-Camera-Swarm-System*, (accessed 1/28/2022)

[15] *https://blynk.io/*, (accessed 1/28/2022)

[16] Henry Dietz and Paul Eberhart. "An Ultra-Low-Cost Large-Format Wireless IoT Camera," Electronic Imaging, Imaging Sensors and Systems, pp. 70-1-70-7(7), 2021 DOI: 10.2352/ISSN.2470-1173.2021.7.ISS-070