Aan de slag met de ESP32-Cam

Binnen de Robotica club worden er heel veel verschillende robots gebouwd en hierbij worden veel verschillende processoren gebruikt. Van oudsher zijn dat de Atmel-AVR en Microchip-PIC chip's. Daarna zijn er een heel scala aan andere processoren bijgekomen, we denken aan de ST, Propeller, ARM en ga zo maar door.
Een aantal jaar terug is de ESP8266-01 module op de markt gekomen. De kracht van deze module is dat het een complete Wifi toepassing is, met hierop een seriele aansluiting. Door middel van AT-commando's is deze module te gebruiken om een willekeurig u-controllerbord van Wifi te voorzien. Op zich een mooie toepassing, maar "onder de motorkap" van deze module schuilt een 32-bits processor met 1 – 4 MB geheugen en dat voor maar een paar Euro!! Hier dromen we van als we over microcontrollers praten, en we gebruiken het "alleen maar" voor communicatie. Het grote nadeel van de ESP8266 serie was dat het programmmeren hiervan niet heel makkelijk ging. Maar voor de IOT toepassingen is het een heel grote doorbraak geweest.
We zijn nu jaren verder, de ontwikkelingen hebben niet stil gestaan. Verschillende ontwikkel platforms hebben de ESP-familie omarmd. Het bedrijf Espressife heeft een opvolger uitgebracht, de ESP32. Tijd om de zaak opnieuw te evalueren.

De ESP32.
De ESP32 bevat een Xtensa dual-core processor met een kloksnelheid van 160 of 240 Mhz, met 512 KiB SRAM, Wifi en Bluetooth en de nodige IO aan boord. Dit is best wel indrukwekkend en dat nog steeds voor maar een paar Euro.
In de afgelopen paar jaar zijn er rond de ESP heel erg veel verschillende bordjes en module's op de markt gekomen. Met hierop 4 – 8 MB geheugen, meer en minder IO-pinnen. Verschillende vormen en standaarden in bordjes. Op basis van de Arduino-layout of de Wemos D1, met of zonder display en vrij recent met een cmos camera module erop. Dat is de **ESP32-CAM**.

Deze module heeft een 2 Mpixel cameramodule, de OV2640-camera , 512 KiB SRAM en 4 MB extra geheugen erop. En dit alles voor rond de 6,50 Euro !!!

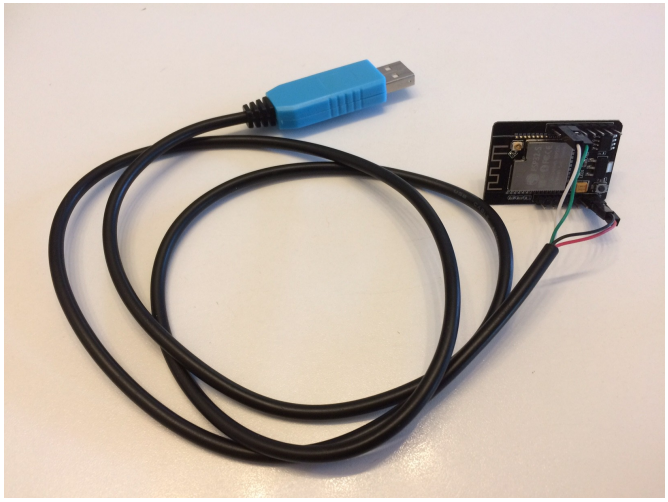
Ontwikkelomgeving.
Van oudsher is er voor de ESP8266 en later de EPS32 door het ontwikkelbedrijf Esperriffs een eigen ontwikkelomgeving op de markt neer gezet. De ESP-SDK. www.espressif.com. Een hele tool-chain met alle toeters en bellen eromheen. Voor de "professionele" ontwikkelaar geen probleem, maar voor de hobbyist voelt het een beetje als met een kanon op een mug schieten. Daarna zijn er door allerlei verschillende partijen verschillende ontwikkelomgevingen opgezet. Een heel mooie is op basis van Visual Studio met hieroverheen PlatformIO.

Zie https://fasani.de/tag/esp32/ voor een opzet voor dit systeem. Dit systeem is nog volop in ontwikkeling en is heel mooi, alleen zelf liep ik zo nu en dan tegen het "probleem" aan dat ik een foutmelding kreeg, die verdween nadat ik PlatformIO opnieuw opgestart had. Dat was best wel even zoeken naar fouten die er niet altijd bleken te zijn. Daarnaast zijn de ESP8266 en ESP32 ook in de Arduino-IDE geimplementeerd. Dit betekent dat we in de voor veel mensen vertrouwde Arduino-omgeving een ESP32 bordje en library's kunnen kiezen en dat we hiermee gewoon aan de slag kunnen. Dan wordt de "moeilijke" ESP32 processor opeens voor veel mensen 'heel makkelijk' bereikbaar.

Hardware ESP32-CAM.
De ESP32-CAM module wordt standaard met een normale lens uitgevoerd. Deze lens heeft een beeldhoek van ongeveer 60 graden. Willen we aan beeldherkenning gaan doen op de wat kortere afstand, lijnvolgen bijvoorbeeld, dan moeten we de module op een vrij grote afstand van de

vloer monteren. Er is ook een Fish-Eye module beschikbaar. Deze heeft een beeldhoek van 120 graden. In dat geval kan de afstand tot de vloer korter worden. Er zijn 2 nadelen/verschillen met de normale camera module. Het beeld is 90 graden gedraaid en de randen van het beeld verlopen van kleur als we de onboard LED aanzetten om het beeld te verlichten. Deze randen worden minder belicht en verlopen hierdoor meer naar grijs.

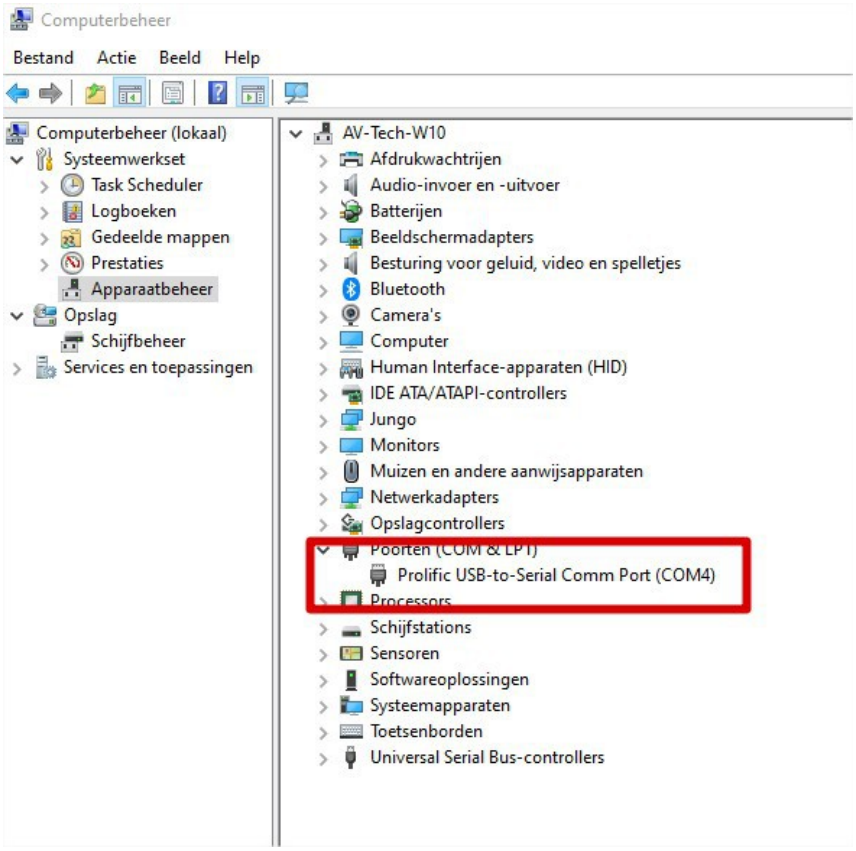
De Esp32-CAM heeft 16 aansluitingen, 3 daarvan zijn GND en er is 1 5V aansluiting die we voor de voeding gebruiken. De 3,3 V aansluiting gebruiken we niet. Ook de uitgaande 3,3/5V aansluiting gebruiken we niet. Voor het programmeren gebruiken we een seriele poort op GPIO1 en GPIO3. En om de module in de programmeer modus te zetten moeten we de GPIO0 aan GND leggen en deze pin kunnen we dan ook (beter) niet gebruiken voor andere toepassingen. GPIO4 is verbonden met de onboard LED. De GPIO12/13 zijn verbonden met de interne SPI aansluiting naar ondermeer de PSRAM en als we deze "verkeerd" gebruiken start de ESP32 module niet op. Het verbinden van een LED aan de 5V via een weerstand naar GPIO12 en 13 levert deze fout op. Zo dunt het wel lekker uit met vrije pinnen. We zouden kunnen uitzoeken of we deze GPIO-pinnen op een andere manier kunnen gebruiken, maar dat heb ik niet gedaan. Dan houden we GPIO-pinnen 2, 14, 15 en 16 over. In theorie kunnen we tot 14 mA sinken met een GPIO-pin. Willen we de ESP32-CAM programmeren dan moeten we een USB-Serieel omzetten op de GPIO1 (TxD) en GPIO3 (RxD) aansluiten. De module heeft GEEN USB aansluiting aan boord.



Aan de slag met de Arduino-IDE.

Als we de normale Arduino-IDE geinstalleerd hebben gaan we naar bestand-voorkeuren en dan bijna onderaan -Meer Board Manager URL's-. Rechts staat een blokje waarmee we een invul scherm kunnen openen waar we aanvullende URL's in kunnen vullen waar vandaan we andere borden kunnen installeren.

Bij [Hulpmiddelen --> Board --> Board beheren] vullen we bij Filter [ESP32] in en we krijgen de librarys die we nogig hebben. Na installatie van de borden gaan we de instellingen voor de ESP32-CAM goed zetten. Bij [Hulpmiddelen --> Board:] kiezen we -ESP32 Wrover Module- Hierna kunnen we voor deze module de volgende instellingen goed zetten:
Upload Speed: "921600"
Falsh Frequency: "80 Mhz"
Flash Mode: "QIO"
Partition Scheme: "Huge APP (3MB No OTA)"
Core Debug Level: "Geen"



Als we het comm-poort nummer van de gebruikte USB naar Serieel omzetten invullen kunnen we een programma op de juiste wijze in onze module zetten. We vinden het COM-poort nummer bij Computerbeheer --> apparaatbeheer. Hier zien we bij Poorten het COM-nummer vermeld. COM4 in onderstaand voorbeeld.

Gaan we naar Voorbeelden --> ESP32 --> Camera --> CameraWebServer, dan kunnen we het voorbeeld programma installeren waarmee we onze ESP32-CAM kunnen testen. In deze sketch moeten we de SSID en wachtwoord van een Wifi netwerk invullen en hiermee verbindt de ESP32-CAM zich met het lokale netwerk en kan je aan de hand van een IP-scan programma of de uitlezing van de seriele debuger

zien op welk IP-adres het beeld van de camera te zien is. We kunnen nu in de webinterface allerlei zaken doen en bekijken. Dit is heel leuk. Een beperking in de eigen programma's is dat deze (nog) niet goed samenwerken met de webinterface in de ESP32. Gebruiken we de webinterface en gaan hier beelden streamen dan hindert dat de werking van het eigen programma. Mijn beperkte programmeer skill's op het gebied van HTML en C weerhouden mij om dit op (korte) termijn op te lossen. In eerste instantie vind ik dat geen probleem. Het zou leuk zijn als hier een oplossing voor komt, maar dat staat bij mij niet bovenaan mijn lijstje van -To Do-

Eigen programma's.
Werkt dit dan kunnen we naar stap-2, next-level, ons eigen programma en hieruit het beeld halen en in onze eigen toepassing gaan gebruiken. Op de GitHub pagina https://github.com/AVTech1/ESP32-Cam-Walker staan de volgende drie programma's waar we naar gaan kijken, te weten: ESP32_beeld.ino, ESP32_lijnsensor.ino en WoF_CAM.ino

ESP32_beeld.ino stuurt een "beeld" wat orgineel 160 * 120 Pixels is in een formaat van 32 * 15 karakters naar de seriele poort. Deze karakters laten zien hoe zwart/grijs/wit een pixel is en zo krijgen we een (vaag) beeld van onze opname.

ESP32_lijnsensor.ino stuurt via de seriele poort 1 lijn als karakters in "grijstinten" of met lijndetectie filter. Deze informatie zou je in een andere microcontroller als lijnsensor-Input kunnen gebruiken.

WoF_CAM.ino is een sketch waarin 2 servo's aangestuurd worden die 4 poten van een Robot vormen.

We behandelen de sketch ESP32_beeld.ino, hierin gaan we het principe behandelen. De sketch bestaat uit verschillende delen, we beginnen met de [void setup()]. Hierin worden alle dingen opgestart en ingesteld die maar 1 keer voorkomen. Hierin komen we het blok [camera_config_t config;] tegen. Hierin staan heel veel toewijzingen en instellingen van de OV2640-camera module. Speciaal zien we de instellingen [config.pixel_format = PIXFORMAT_GRAYSCALE] en [config.frame_size = FRAMESIZE_QQVGA]. We willen eenvoudige data uit onze afbeelding halen en dat doen we als we er een Greyscale afbeelding van maken in het QQVGA formaat van 160 * 120 pixels. Aan het einde van de setup configureren we de onboard-LED door middel van het commando: [ledcSetup(3, 2000, 8);] en [ledcAttachPin(Led0, 3);]. We wijzen de pinnummer Led0 toe aan een willekeurig nummer -3-. Met ledcSetup stellen we deze pin in op 2000 Hz met een 8 bits PWM uitgang. Hierna kunnen we de betreffende uitgang met het commando

[ledcWrite(3,100);] op 100 van de 255 delen "8-bit's intensiteit" aanzetten.
In de [void loop()] roepen we elke halve seconde de routine [camera_beeld();] aan. Hierin zetten we met het commando: [camera_fb_t * s_state = esp_camera_fb_get();] een beeld in het frame-buffer. Dit frame-buffer moet je gewoon als een blok geheugen zien van 160 * 120 byte's met 8-bit's grijstinten. Je kunt het ook zien als een rij van 19.200 (= 160 * 120) byte's. Met dit geheugen cq rij byte's kunnen we nu zelf dingen gaan doen.
In [uint8_t pixel = (s_state->buf[iw + (ih * s_state->width * 2)]);] zijn ih en iw de hoogte en breedte in het beeld-frame waarvan we de pixel-grijswaarde in de variabele [pixel] zetten. Met de variabele [pixel] kunnen we vervolgens eigen zaken gaan doen. In ons voorbeeld gaan we het 160 * 120 pixel bestand omzetten in een 32 * 15 pixel/karakter beeld en sturen dit als ascii-karakters naar de seriele poort.

De andere 2 sketchen zijn uitbreidingen op de sketch ESP32_beeld.ino en hebben verder geen uitleg nodig. In WoF_CAM.ino worden 2 servo's aangestuurd en hiervoor staan de benodigde commentaren in de sketch.

Opmerkingen.
Als we de FishEye camera module gebruiken dan moeten we beseffen dat we best wel breed beeld hebben als we vanaf 20 cm hoogte meten. Een lijn van 1 cm wordt dan heel smal en kunnen we bijna niet meer zien. Dan is het het overwegen waard om veel meer metingen te verrichten en dan bijvoorbeeld per 3 of 5 pixels een gemiddelde te nemen. Dan halen we veel meer gegevens uit onze "lijn".

Conclusie.
De ESP32-CAM is een heel leuke module voor een nog leukere prijs. In de Arduino-IDE is hij heel goed te programmeren. Het "nadeel" zijn de beperkt aantal IO-pinnen, maar dat is eventueel met I2C wel te omzeilen. Omdat ik een Walker wilde bouwen met 2 servo's was het beperkte aantal IO-pinnen voor mij geen probleem.

Ik heb veel plezier beleefd aan deze ontdekkingstocht en hoop dat anderen aan de ESP32-CAM ook veel plezier beleven.

Abraham Vreugdenhil.

Bronvermelding:

Ik heb mijn informatie van ondermeer onderstaande website's afgehaald. Met name de links vermeld bij -Opzetten van ESP32-CAM met demo- bevat veel nuttige informatie om met de ESP32-CAM aan de slag te kunnen gaan. Zeker aan aanrader om te bekijken en ook om te begrijpen hoe bepaalde instellingen voor de Wifi werken.

PlatformIO:
https://fasani.de/tag/esp32/

ESP-SDK:
www.espressif.com

FishEye Lenzen:
https://www.banggood.com/search/esp32-cam.html?from=nav
banggood: esp32-cam

webserver met beeld:
Arduino: Voorbeelden --> ESP32 --> Camera --> CameraWebServer

Opzetten van ESP32-CAM met demo:
https://robotzero.one/esp32-cam-arduino-ide/
https://randomnerdtutorials.com/ [esp32 cam]

ESP32-Cam RC-Tankdemo:
https://github.com/PepeTheFroggie/ESP32CAM_RCTANK

GitHub met de genoemde programma's:
https://github.com/AVTech1/ESP32-Cam-Walker