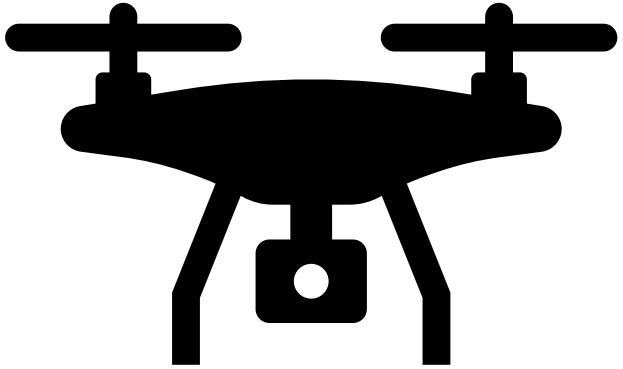
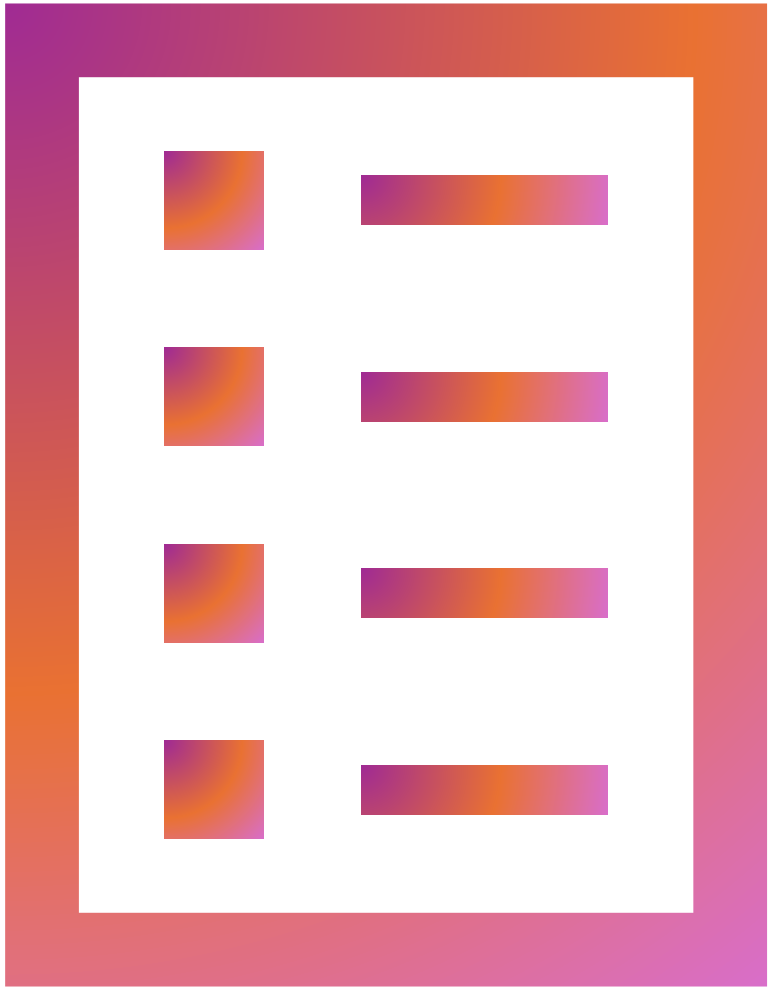


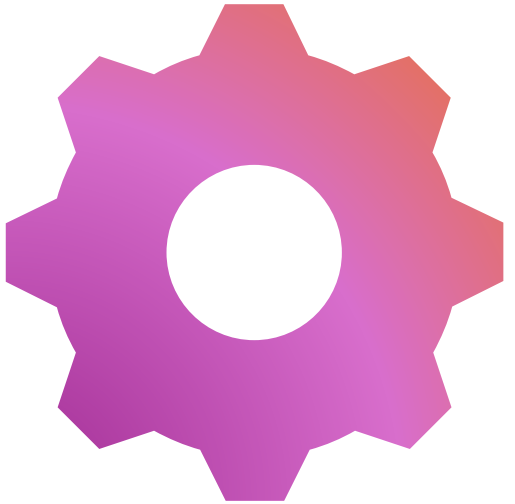
Abschlusspräsentation Gruppe 3

Teilteams Mech, El, IT

Gliederung

- ❑ Mechanik
- ❑ Elektronik
- ❑ Informatik



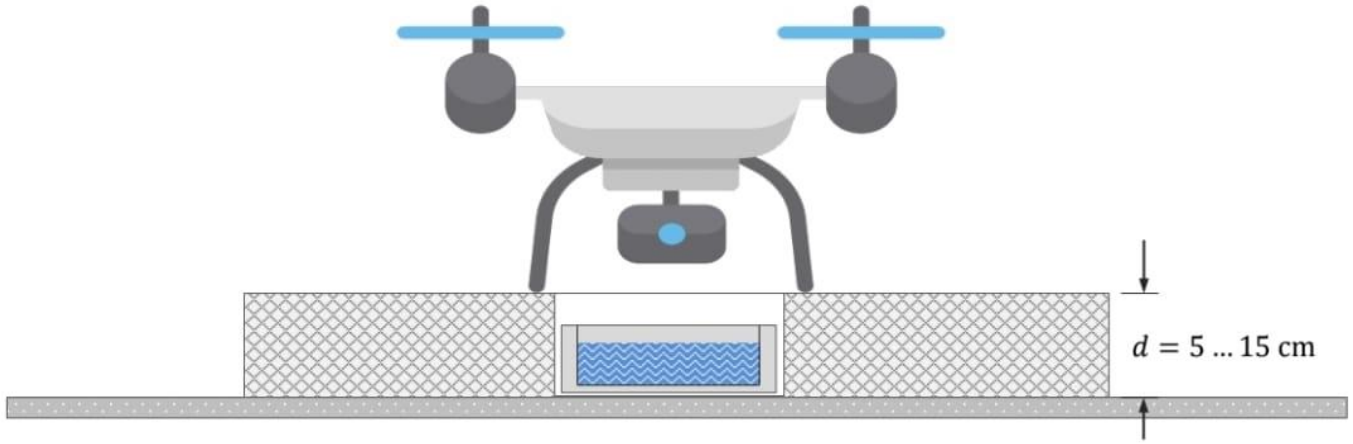


Teilteam Mech

Aufgabenstellung

- Aufnahme der Paylaod (min. 200 ml)
  - Schaumstoffplatte mit bekannter Höhe
  - Wasser wird aus Löschwasserbehälter aufgenomen
- Max. zulässiges Gewicht: 500 g

- Unsere Lösung:
  - Beförderung des Wassers mit einer Pumpe in den Behälter

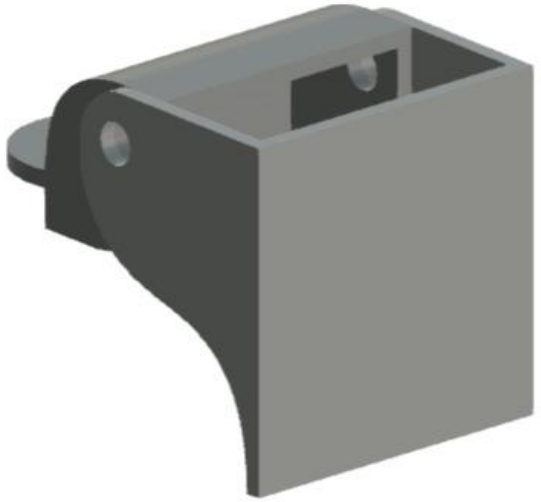


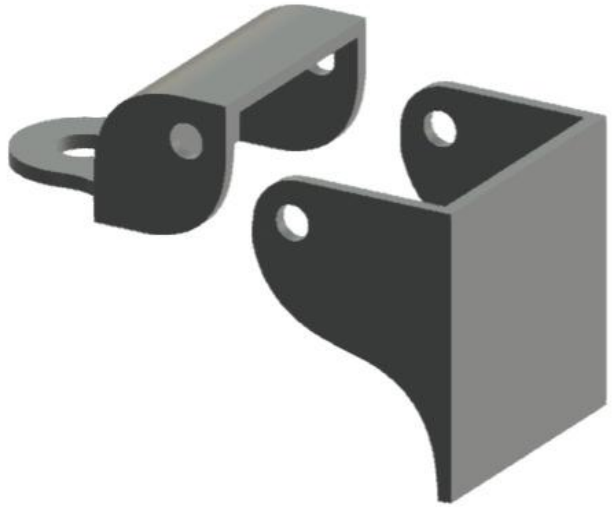
Behälter

- Öffnungsmechanismus: Falltür
- Dichtung mit Haushaltsgummi an Falltür
- Behältnis geschlossen mit zwei Pins
- Öffnung an Deckel für einen Schlauch
- Überlauflöcher am Deckel
- Pumpenhalterung

Weitere Konstruktionen

- Kamerahalterung
- Behälter-Halterung an Drohne





Finales System

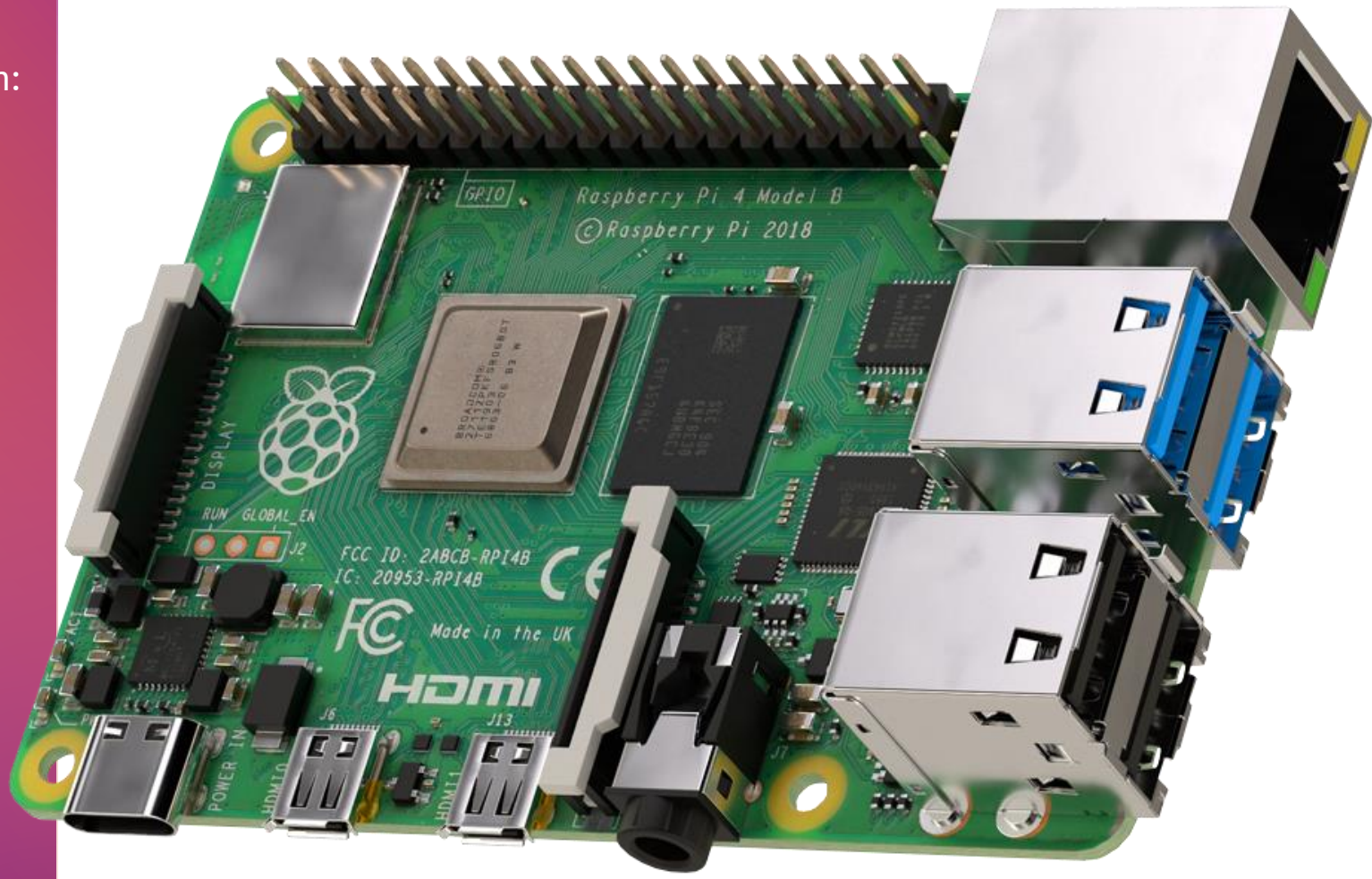
- 12 V Pumpe
  - Durchfluss: 1,5 - 2 L/min
- Wasserbehälter
- Falltür
- Servo zum Herausziehen der Pins
  - Drehmoment 2 kg/cm
  - Geschwindigkeit: 0,09 s/60°
  - Drehwinkel: 180°

Teilteam Elektro

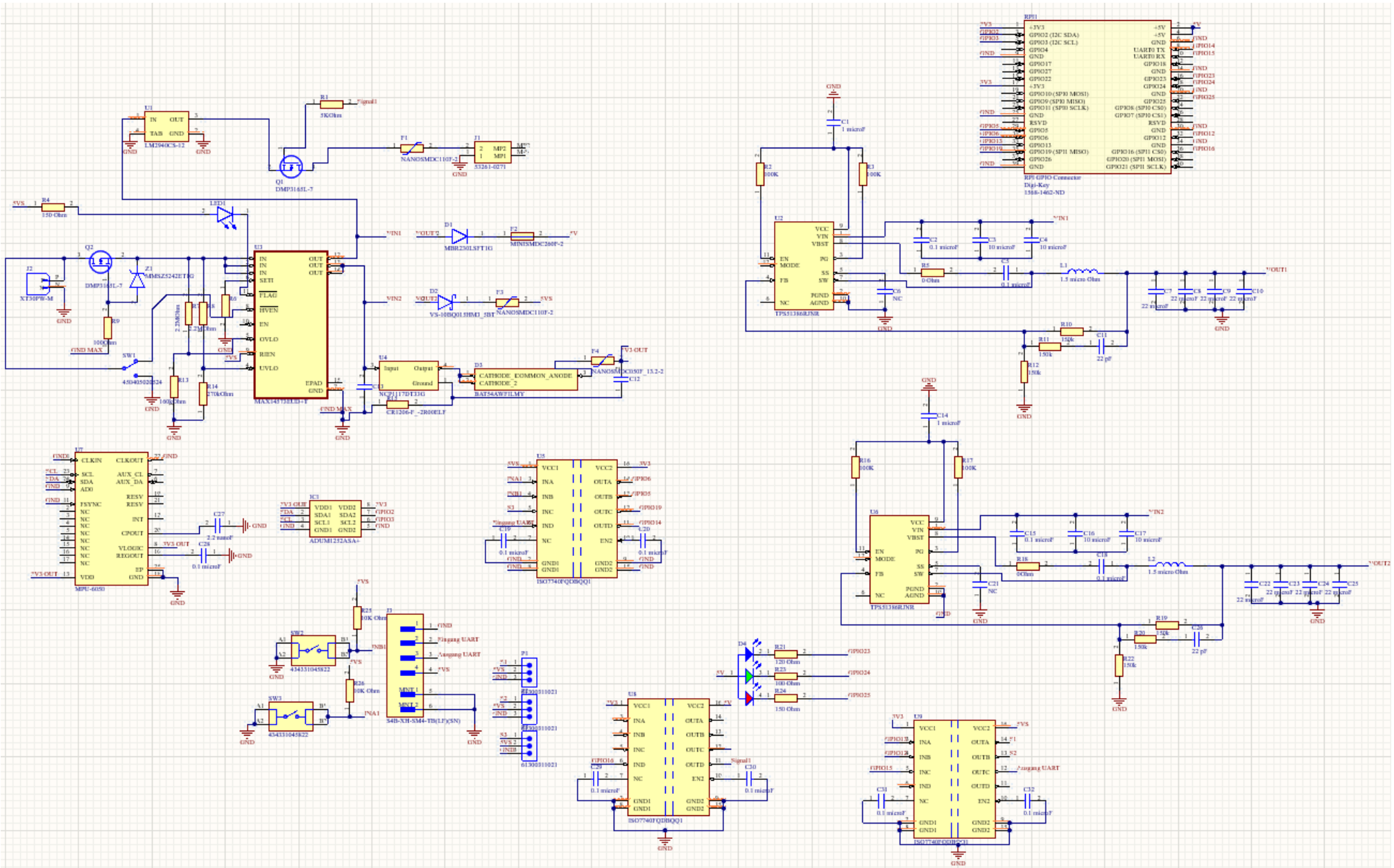
Aufgabenstellung

Das PDB muss folgende Mindestanforderungen erfüllen:

1. Bereitstellung einer Spannungsversorgung für den Raspberry Pi® aus der Drohnen-Batterie(12...18 V (XT30 Stecker) 5 V / 1 A, Wirkungsgrad > 75%)
2. Bereitstellung einer Spannungsversorgung für die benötigte Aktorik/Sensorik (typisch auch 5 V / 1 A)
3. Bereitstellung von (mind. 2) Steckplätzen für den Anschluss von handelsüblichen Modellbau-Servos
4. Bereitstellung von Steckplätzen für die Aktorik und Sensorik
5. Schutz der benötigten GPIO-Signale des Raspberry Pi® durch galvanisch isolierende Signalkoppler.
6. Umfassender Schutz gegen Kurzschlüsse und Verpolung
7. Manuelle Abschaltmöglichkeit aller über das PDB versorgten Verbraucher (Raspberry Pi®, Aktorik) zur Reduzierung des „stand-by“ Stromverbrauchs.
8. Das PDB ist als Aufsteckboard auf die vorhandene Steckerleiste des Raspberry Pi® zu designen.

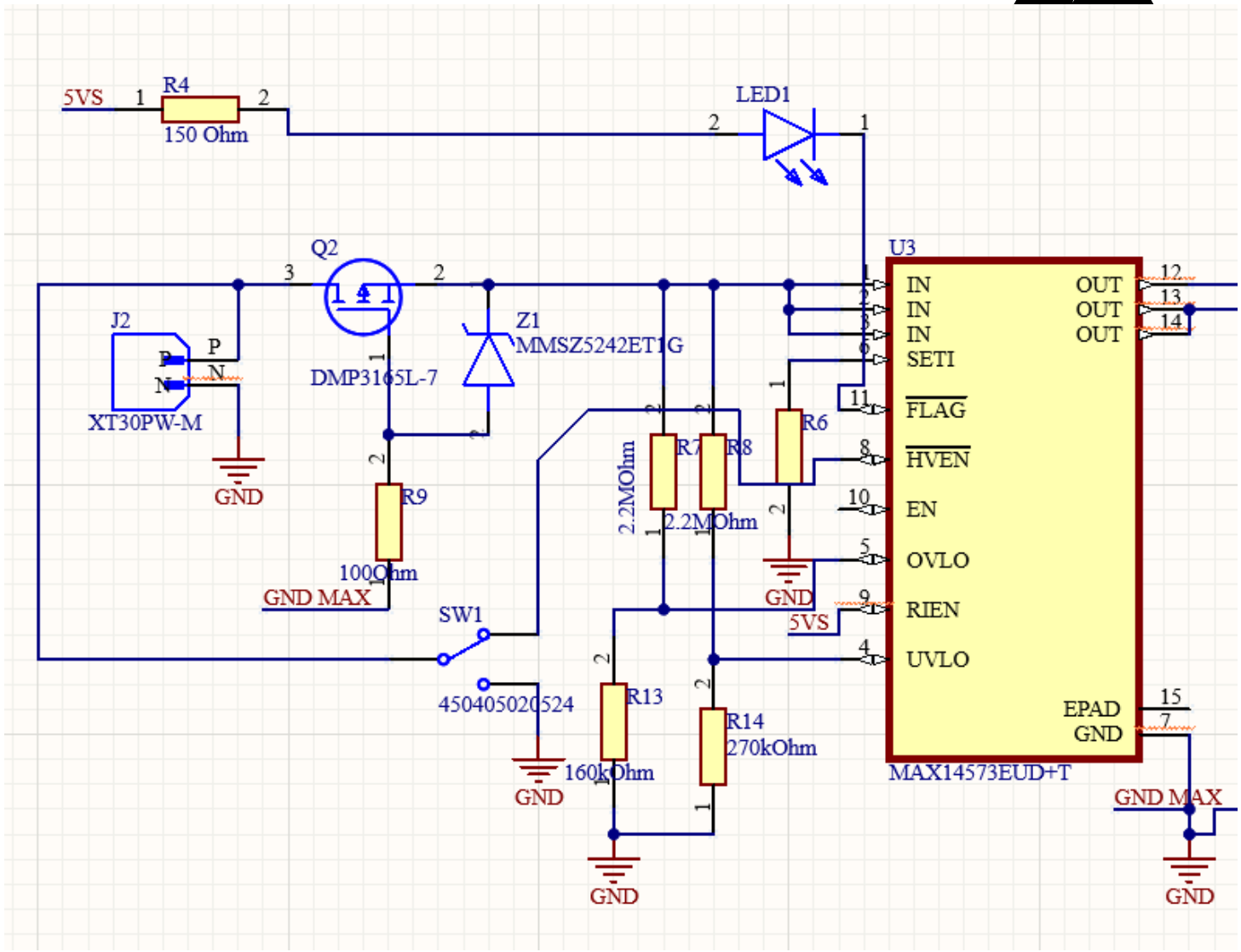


I. Schematic



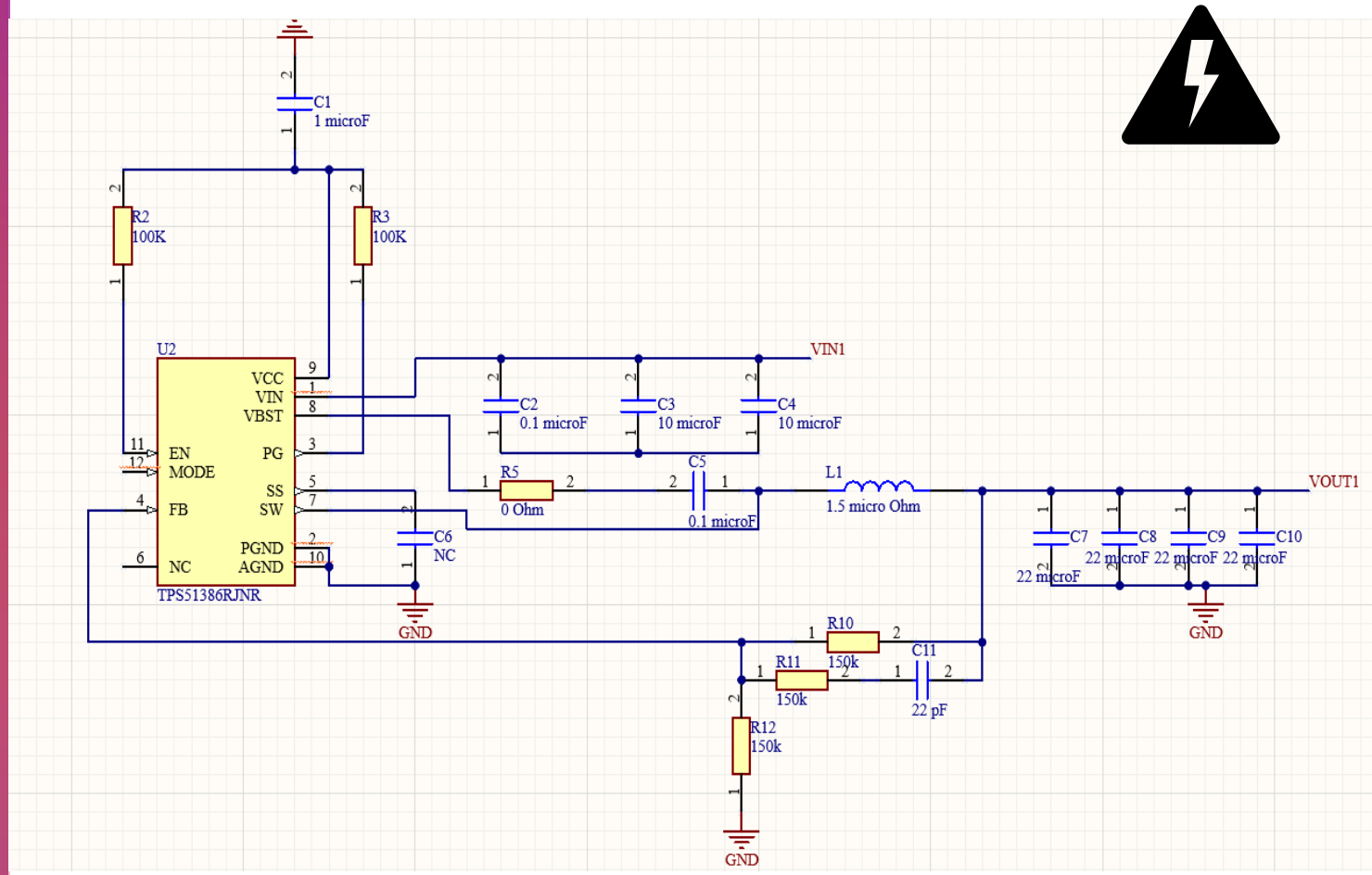
1.Spannungsversorgung für den Raspberry Pi®

- **MAX14573EUD+T** : Schaltungsschutz und Überspannungsunterdrückung
- Verpolungsschutz durch Q2 und Z1.
- Manuelle Abschaltmöglichkeit aller über das PDB durch SW1.



1.Spannungsversorgung für den Raspberry Pi®

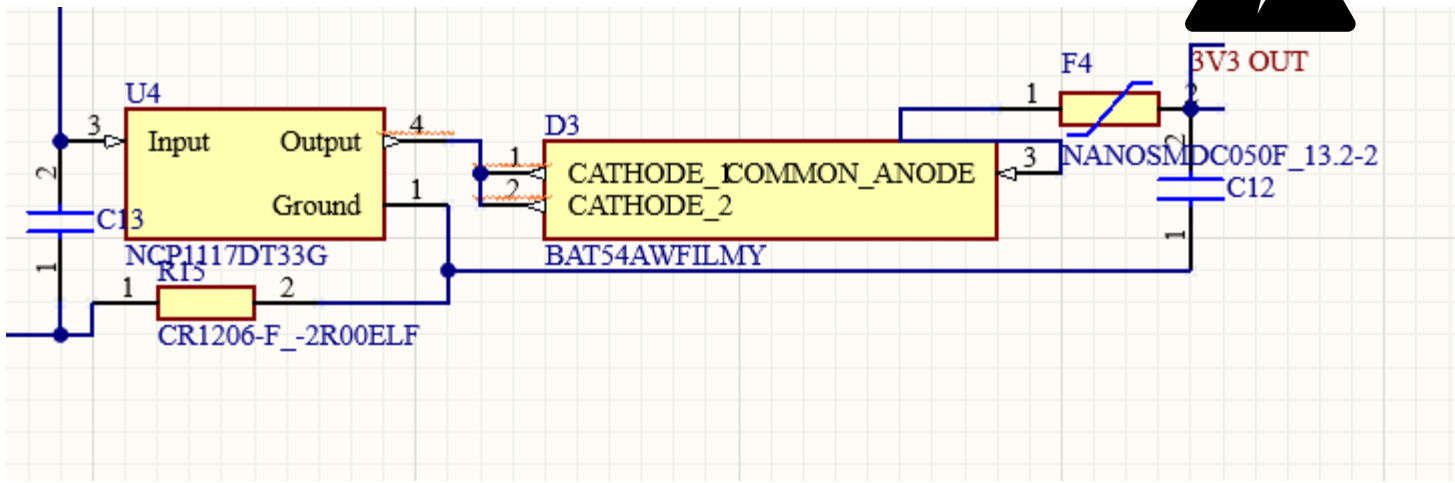
- Spannung Umwandlung 12v zu 5v durch U2 (**TPS51386RJNR**) für den Raspberry Pi und U6 (**TPS51386RJNR**) für die Servo Steckern.

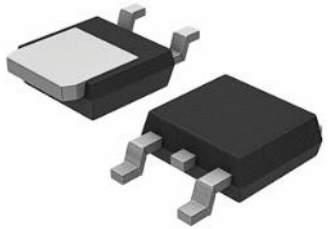


"TPS51386RJNR"

1.Spannungsversorgung für den Raspberry Pi®

- Spannung Umwandlung 12v zu 3,3v durch U4 (**NCP1117DT33G**) für den Raspberry Pi.
- Verpolungsschutz durch Schottky Diode D3.
- Sicherung durch F4.





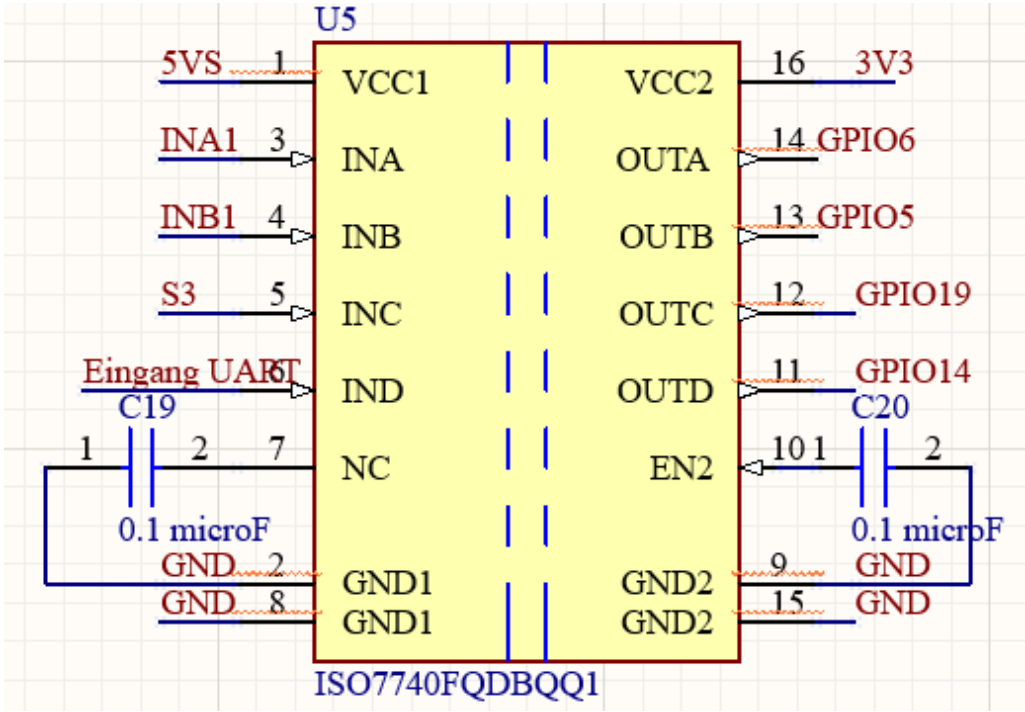
"NCP1117DT33G" (U4)

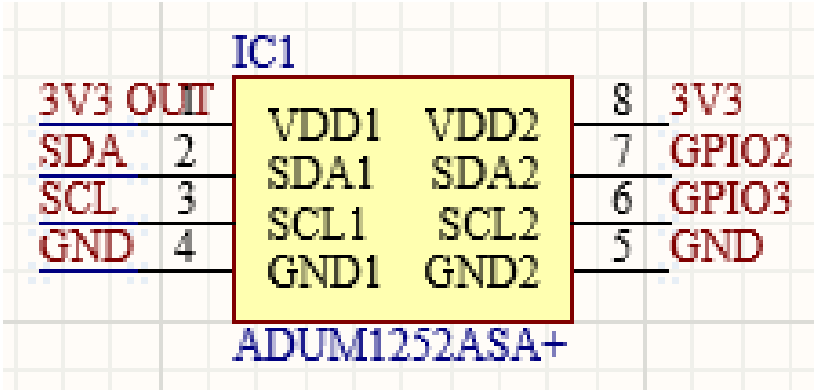
"BAT54AWFILMY" (D3)

"NANOSMDC050F" (F4)

2. Galvanisch isolierende Signalkoppler

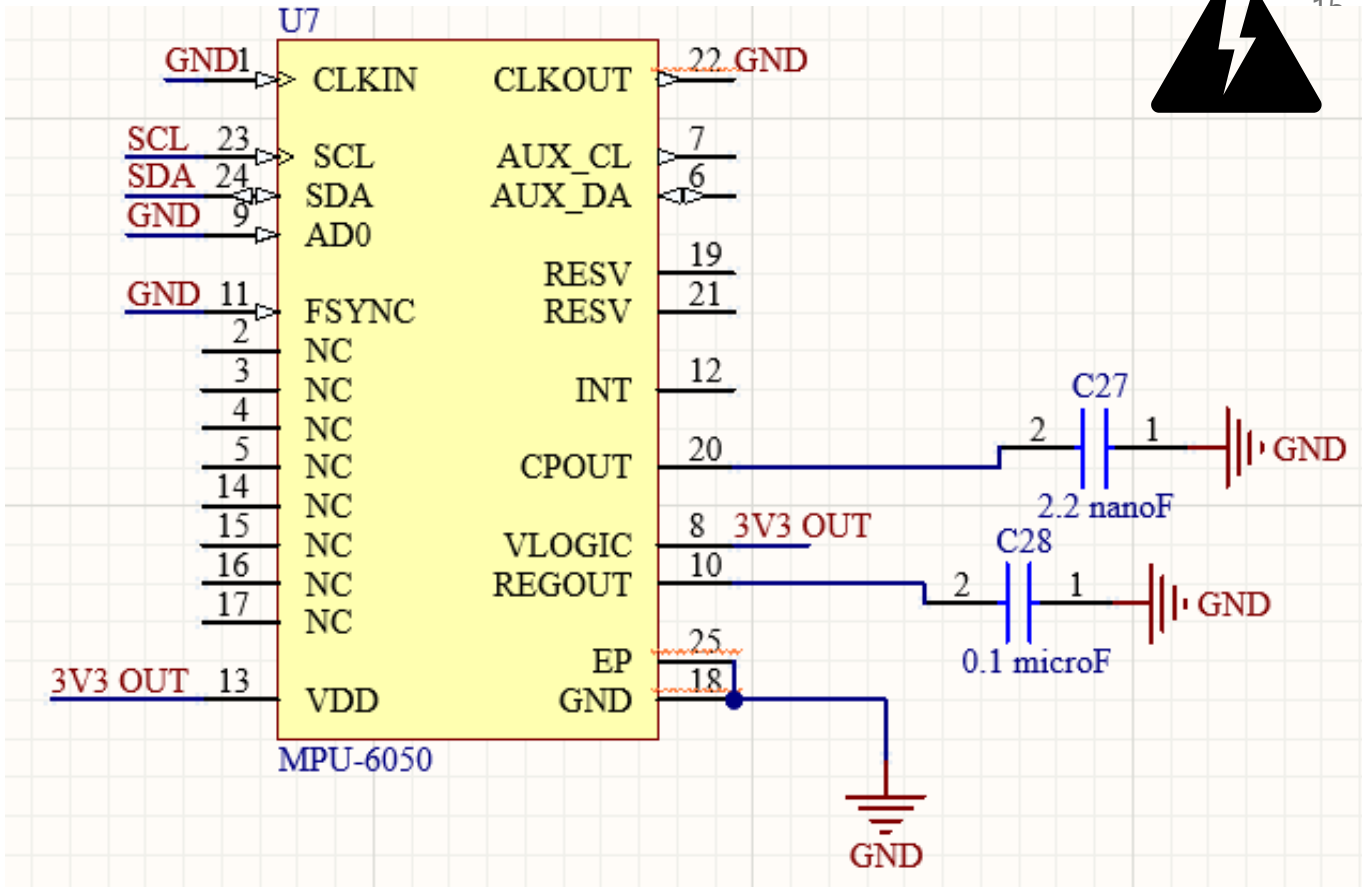
- Schutz der benötigten GPIO-Signale des Raspberry Pi® durch U5, U8, U9 (**ISO7740FQDBQQ1**) und Sicherheit zu gewährleisten und Signalstörungen zu minimieren durch IC1 (**ADUM1252ASA+**)





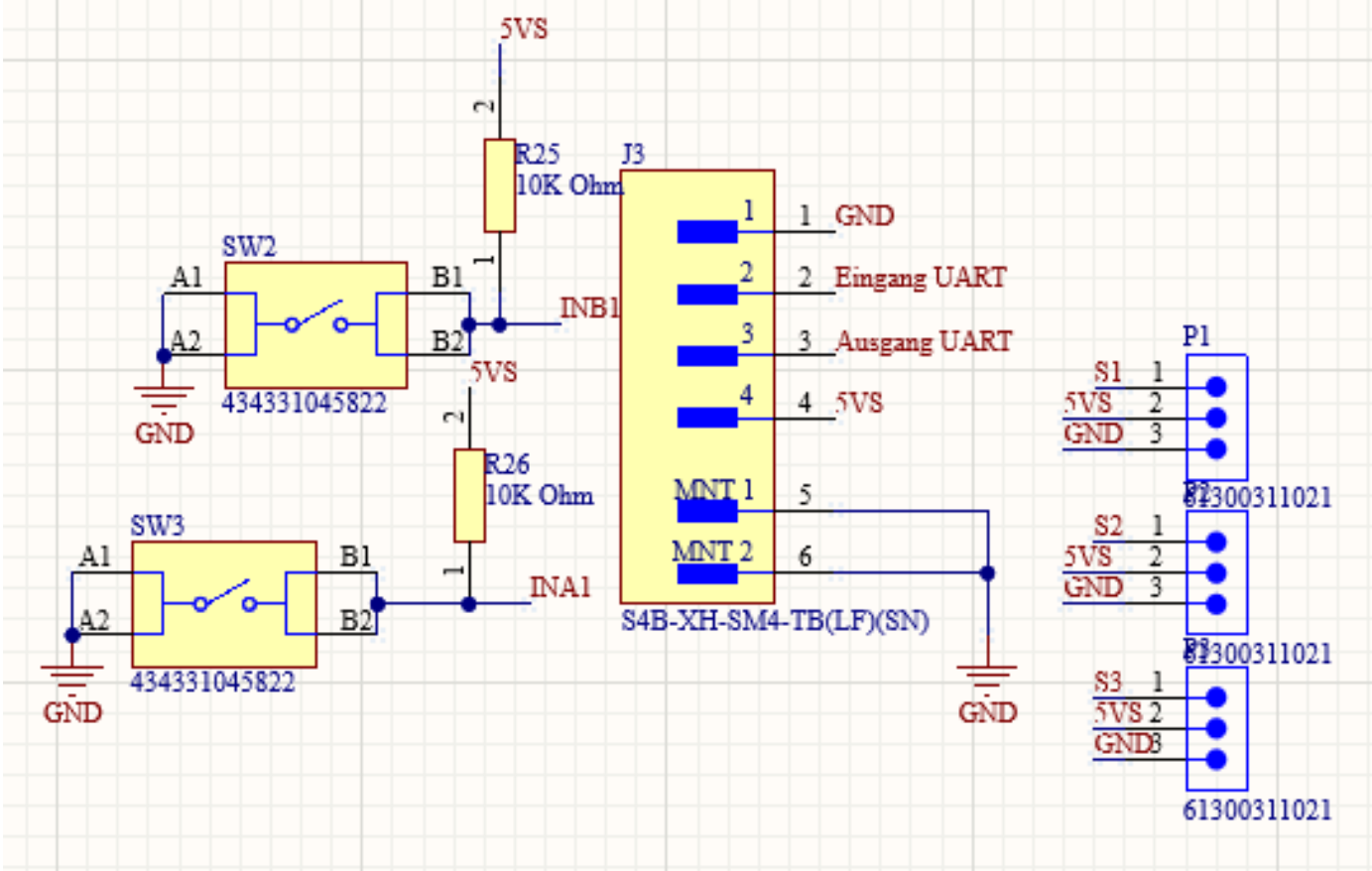
3. Gyroskop

- Um eine präzise Bewegungserfassung und Stabilisierung haben wir für MPU-6050 entschieden.



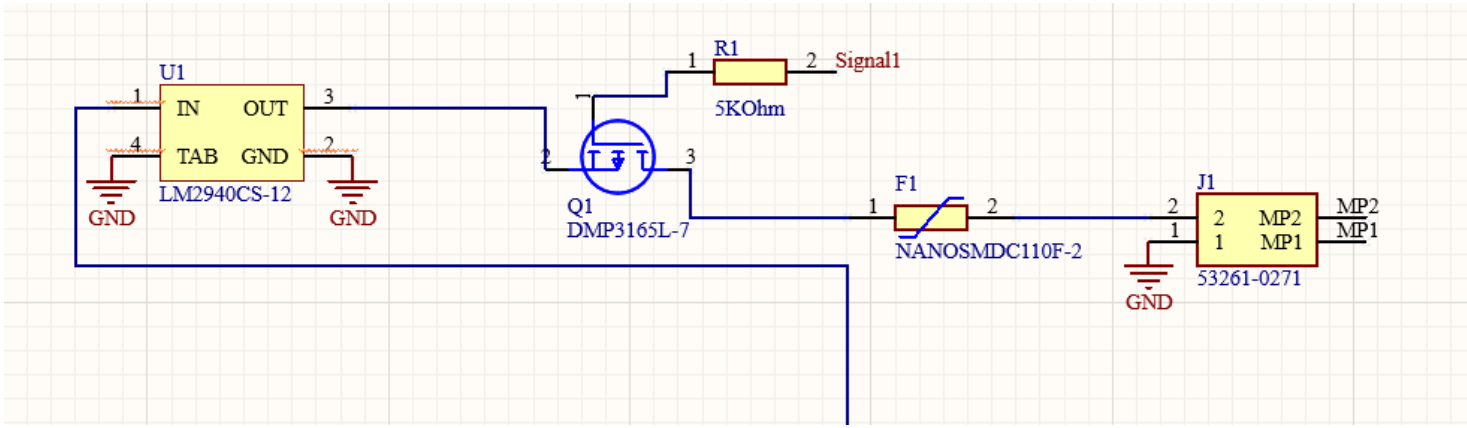
4. Servos und Steckern

- Bereitstellung von 3 Steckplätzen für den Anschluss von handelsüblichen Modellbau-Servos (P1, P2, P3).



5. Spannung und Stromversorgung für die Pumpe

- Spannungsumwandlung zu 12v durch U1 (**LM2940CS-12**) für die Wasserpumpe.
- Verpolungsschutz und Sicherung durch MOSFET „Q1“ (**DMP3165L-7**) und „F1“(**NANOSMDC110F-2**)
- „J1“ ist der Stecker für die Wasserpumpe.



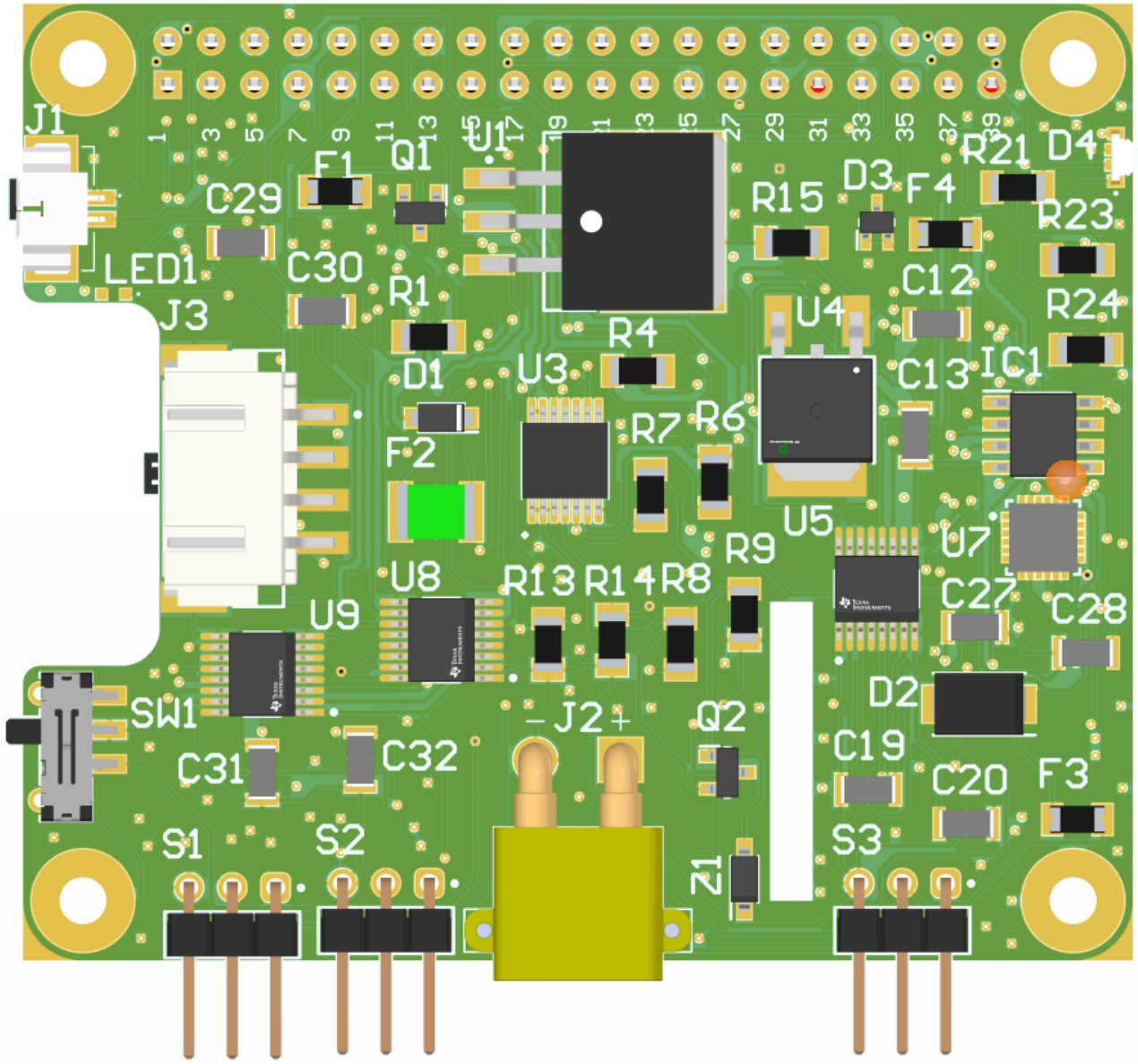
„J1“

„Q1“

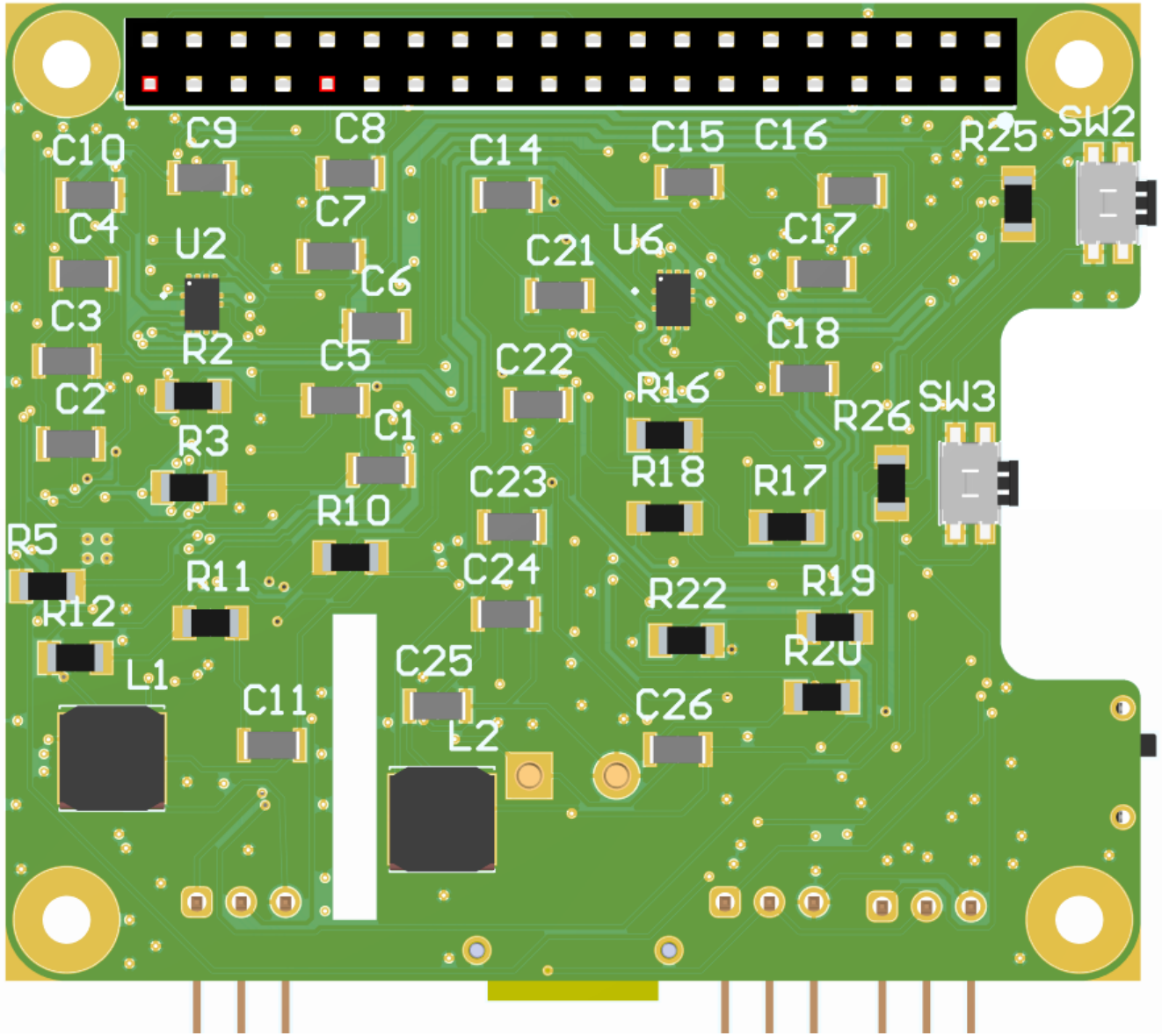
„F1“

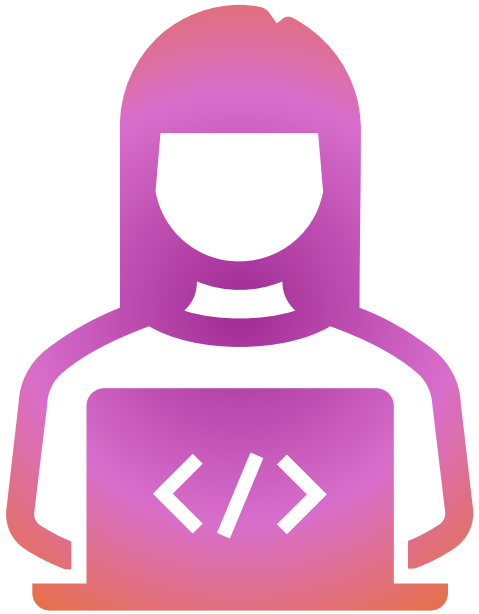
II. Design der Platine

- Wie Sie sehen können, gibt es viele Bauteile und der Platz reicht nicht für alle aus. Daher haben wir beschlossen, einige auf die Rückseite zu verlegen.



- Da haben wir die zwei 5V Regulatoren (U2, U6) platziert weil sie mit viele Widerstände und Kondensatoren verbunden sind, damit wir Platz auf der Vorderseite sparen können.





Teilteam Info

Systemüberblick / Rahmenbedingungen

Rahmenbedingungen :

- Raspberry Pi 4B 8GB, Raspberry Pi Camera 3
- Python als Programmiersprache

Ziele:

- Apriltagerkennung
- Flugbahnberechnung
- Zeitberechnung zum idealen Abwurf
- Aufnahme & Auslösung der Payload
- Restartknopf am Board
- Nutzerdefinierbare Einstellungen
- Anlegen von Logs und Bildern
- Implementierung von:
  - Gyroskop
  - Servoausgänge
  - LED und Knöpfen

Software / Git

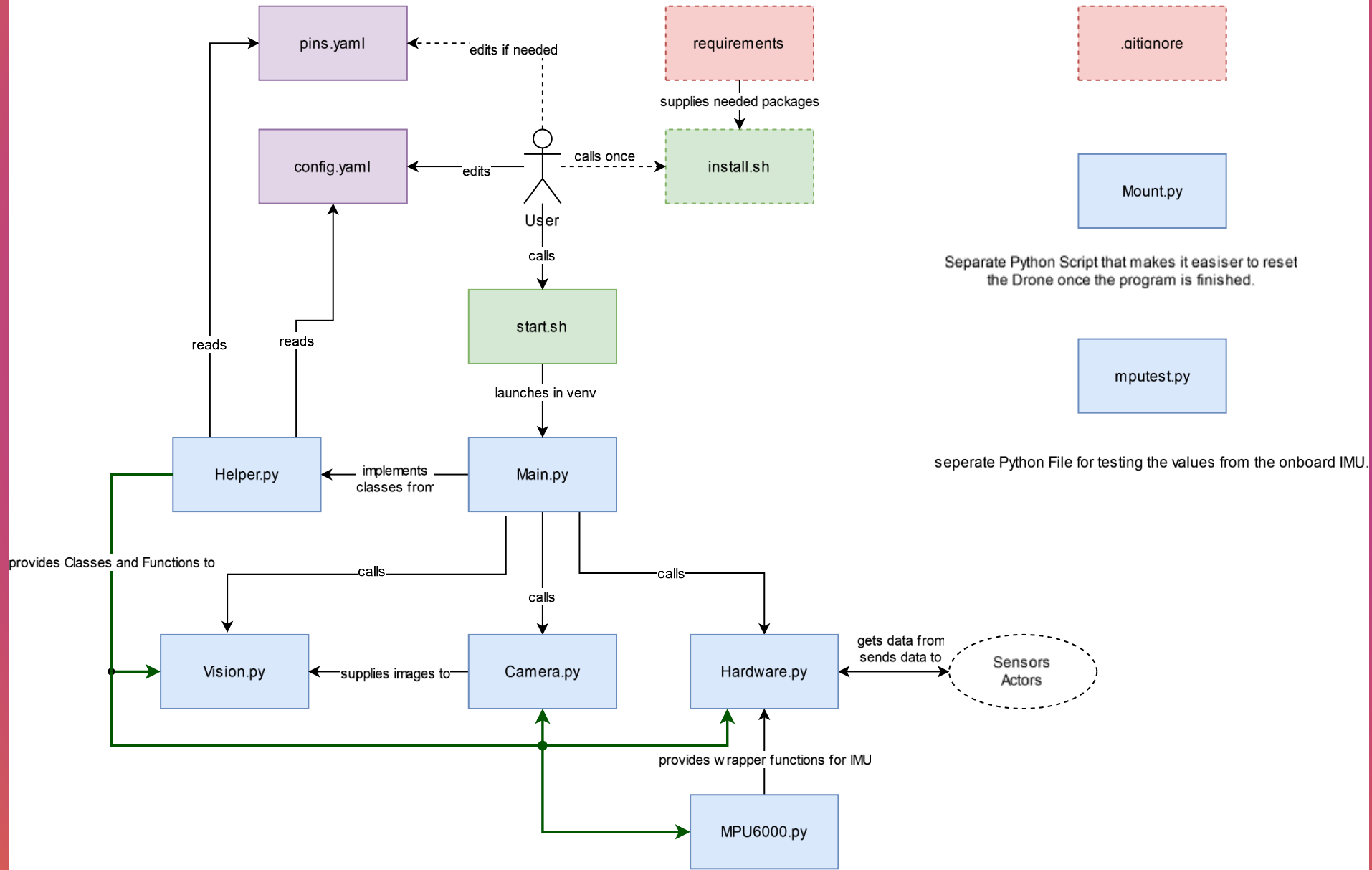
- ✓ Nutzung des Version Control Systems **GIT**
- ✓ Insgesammt **40 Commits** durch **3 Developer**
- ✓ Einzelner Benutzeraccount für jeden Entwickler
- ✓ **3 Branches**

```
./
├── assets
│   ├── logo_cropped.jpg
│   ├── logo.png
│   └── transparent_logo.png
├── electronics
│   └── ...
├── install.sh
├── mechanics
│   └── ...
├── README.md
└── software
    ├── Camera.py
    ├── captures
    ├── config.yaml
    ├── CV2.jpg
    ├── Hardware.py
    ├── Helper.py
    ├── logs
    ├── Main.py
    ├── Mount.py
    ├── MPU6000.py
    ├── mputest.py
    ├── pins.yaml
    ├── __pycache__
    ├── requirements
    ├── SOFTWARE.md
    ├── start.sh
    ├── venv
    └── Vision.py

9 directories, 22 files
```

Functional software diagram

BEWARE: GROSSLY SIMPLIFIED!



Objektorientierung und Datenobjekte

- **Leichte erweiterung möglich auf mehrere Feuer / mehrere Dronen**
- **Definiert in `Helper.py`, importiert in alle .py Dateien**

`Drone:`

`Attribute:`
- Angle
- Speed
- Height
- Active
- Buttons

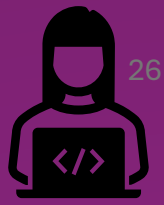
`Methoden:`
`__init__`
Getters, Setters

`Fire:`

`Attribute:`
- Center
- Active
- Height
- Time_to_drop
- ...

`Methoden:`
- **`__init__`**
- **`__str__`**
- **arccalc**

Modularisierung und Kompatibilität



- Importierung der Dateien als Module
- **Separation of concerns**
- Strukturierter Zugriff auf gemeinsame Elemente

Init-Struktur:

```
if __name__=="__main__":
    #empty / debug main function.
    try:
        main()
    except KeyboardInterrupt:
        print("goodbye, cleaning up before I leave...", h.LogLevel.INFO)
        vision_cleanup()
        exit()
else:
    ##Module startup code here.
    setup()
```


```
def main():
    #set the killswitch.
    hw.set_killswitch(restart_exit)
    #Object initialization
    f:Fire = None
    d:Drone = Drone()
    #greet user.
    hw.set_LED(0xFFFFFF)#SET WHITE
    if h.parameters['Main']['coolmode']:
        print(greetstring, h.LogLevel.INFO, "RED")
    else:
        print("Hello World", h.LogLevel.INFO, "MAGENTA")
    polling_thread = start_polling(d, f) # function that calls itself
repeatedly until stopped.
    ##Load water
    hw.acquire_payload()
    ##in-air loop:
    while not exit_flag.is_set():# This is a while True loop adjusted to allow
restart functionality. It exits either because the restart button is pressed
or if a fire was found to be extinguished.
        f = find_fire()
        if f!=None and v.is_in_center(f):
            hw.set_LED(0xD02090)#magenta
```

```
    hw.acquire_payload()
    ##in-air loop:
    while not exit_flag.is_set():# This is a while True loop adjusted to allow
restart functionality. It exits either because the restart button is pressed
or if a fire was found to be extinguished.
        f = find_fire()
        if f!=None and v.is_in_center(f):
            hw.set_LED(0xD02090)#magenta
            print("Fire was found and centered.", h.LogLevel.INFO, "MAGENTA")
            print(f"center at {f.center}", h.LogLevel.INFO)
            break

            ##Found Fire, proceeding
            ##check if extinguishable
        else:
            time.sleep(h.parameters['Main']['fire_polling'])
            continue
            ##no fire found
    #extinguish:
    extinguish(d, f)
    hw.done()
    time.sleep(h.parameters['Main']['afterrun_time'])
    print("made it to EoP, shutting down gracefully..", h.LogLevel.INFO)
    return
```

Computer Vision methoden

- `get_speed()`:
  - Auslesung der Bilddateien (b&w)
  - Pixelshift
  - Geschwindigkeit in mm
  - Mittelwertbildung
- `get_height()`:
  - Errechnung der Apriltag Fläche
  - (Transformation um Drehwinkel)
  - Verhältnis Bildgröße zu Apriltag
    - -> daraus Höhe

`arccalc`

$$h = \frac{l_{Apriltag,physical} * focallength_{mm}}{l_{Apriltag,measured} * l_{pixel}}$$