WORKSHOP-TERMIN 3

Drohnenprogrammierung und Automatisiertes Fliegen

22.09.2022

Organisatorisches

- Patrik Golec
  - patrik.golec@fh-kufstein.ac.at
- wissenschaftlicher Mitarbeiter
  - Masterstudent WCIS
- Projekt DIH West
  - Schulungsreihe im Rahmen des Projekts

Organisatorisches

- 12.04.: VM & Python Basics
- 10.05.: ROS & ROS Python
- **22.09.: Drohnenschwärme & -programmierung**

Agenda

- VM-Check
- Recap ROS
- Drohnen & Drohnenschwärme
- Drohnenlabor-Setup
  - Hardware
  - Software (Crazyswarm)
- Programmierung mit Crazyswarm
  - praktische Beispiele
- Drohnenlabor-Besichtigung
- Ausblick Automatisiertes Fliegen

Ziele

- Grundlagen Drohnenschwärme verstehen
- Funktionsweise Drohnenlabor-Setup verstehen
  - Hardware
  - Software (ROS + Crazyswarm)
- Einfache Manöver mittels Python umsetzen
- Potenzial und zukünftige (zivile) Einsatzszenarien erkennen

VM-Check

- Vorkonfigurierte Ubuntu 20.04-VM
  - Oracle VirtualBox
- Starten
  - ggf. aktualisieren
  - ggf. Sicherungspunkt wiederherstellen
- ROS-Installation prüfen

```
> rosversion -d
```

ROS RECAP

ROS RECAP

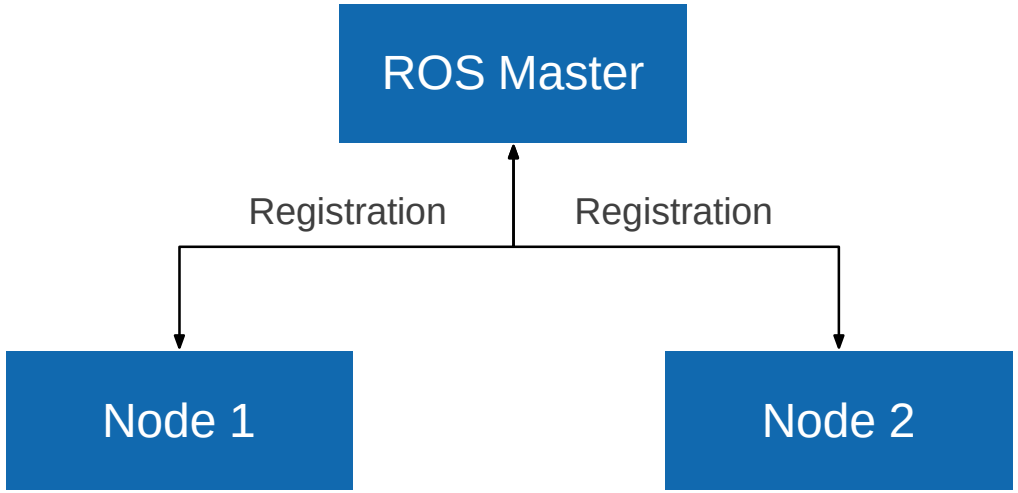
- Robot Operating System (ROS)
  - Meta-Betriebssystem
  - Middleware für Roboterprogrammierung, „Unterbau“
- Komponenten
  - Kommunikationsschicht
  - Werkzeuge & Tools
  - Algorithmen
  - Ecosystem

ROS Prinzipien

- **Peer to peer**
  - ROS-Programme kommunizieren über API (ROS messages, services, etc.)
- **Verteilt (Verteiltes System)**
  - ROS-Programme können über mehrere Maschinen verteilt laufen
- **Multi-lingual**
  - ROS Module können in beliebigen Programmiersprachen entwickelt warden, solange es eine Clientbibliothek gibt (C++, **Python**, Java, MATLAB, etc.)
- **Leichtgewichtig**
  - Wrapper für Bibliotheken
- **Kostenlos und open-source**
  - die meiste ROS-Software ist frei und quelloffen

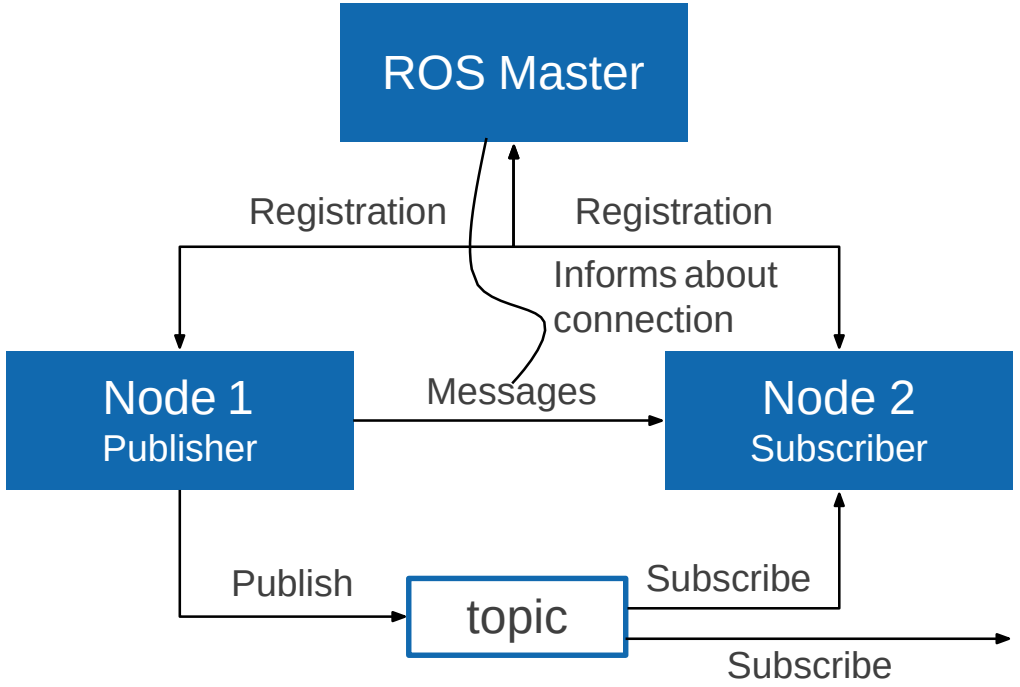
ROS Kommunikation

- Master
  - ermöglicht Verbindungsaufbau zwischen Nodes (Lokalisierung, Kommunikation)
- Nodes (Prozesse)
  - ausführbares Programm
  - unabhängig von anderen Nodes



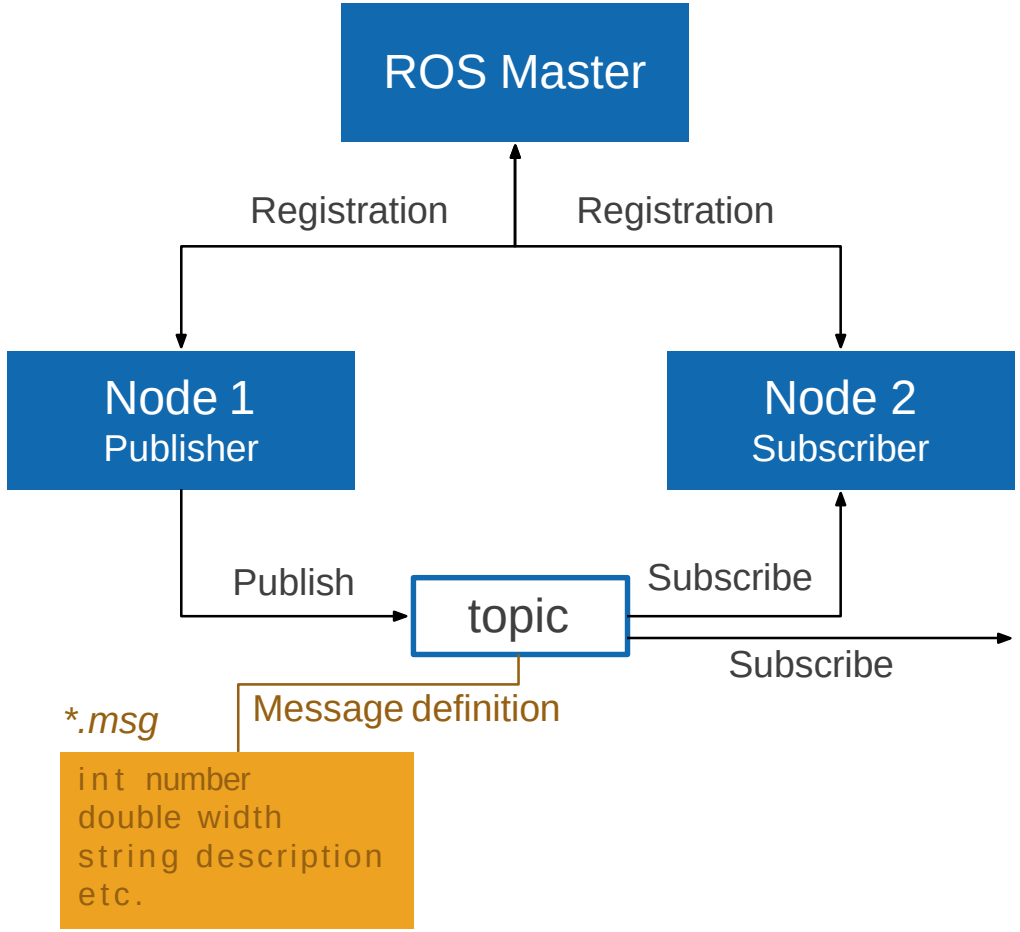
ROS Topics

- Nodes kommunizieren über Topics
  - können Topic subscriben oder publishen (Publisher-Subscriber-Muster)
  - für gewöhnlich 1 Publisher, mehrere Subscriber
- Topic ist ein kontinuierlicher Datenstrom



ROS Messages

- werden innerhalb von Topics von Nodes untereinander versendet
- Datenstruktur definiert Typ eines Topics
- Verschachtelte Strukturen von Integern, Floats, Booleans, Strings usw.
- definiert über **.msg* Dateien



ROS Entwicklung

- ROS Build-System
  - Arbeitsbereich
  - Werkzeug: catkin

Hier arbeiten

src

Der Quellbereich (*source space*) enthält den Quellcode. Hier kann man den Quellcode für die Pakete, die man erstellen möchte, klonen, erstellen und bearbeiten.

Nicht berühren

build

Im *build*-Bereich wird CMake aufgerufen, um die Pakete im Quellbereich zu bauen. Cache-Informationen und andere Zwischendateien werden hier aufbewahrt.

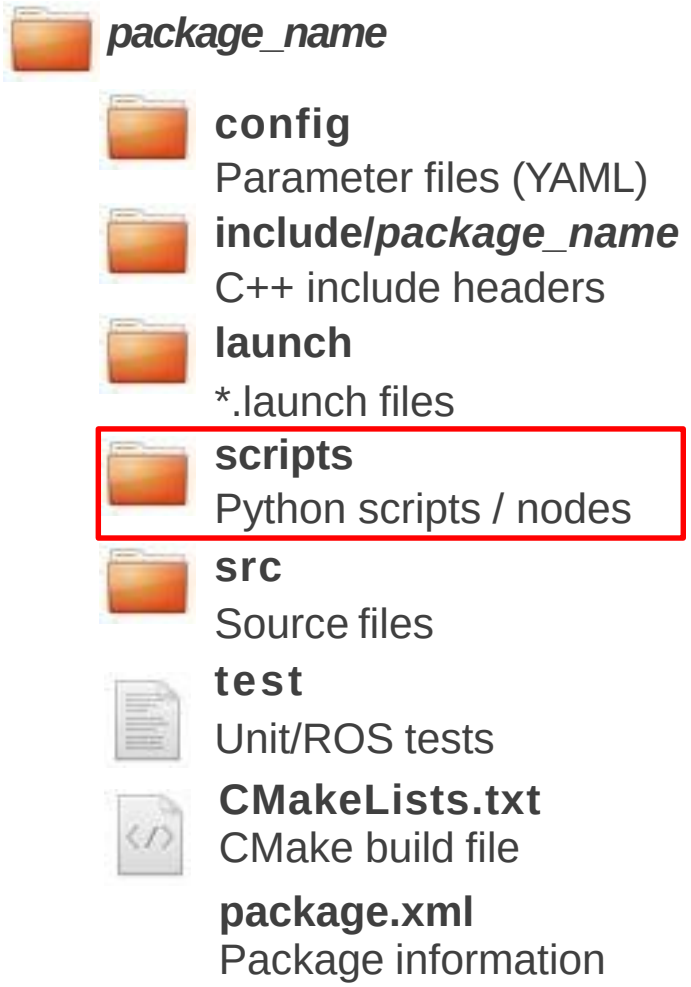
Nicht berühren

devel

Der Entwicklungsbereich (*development space - devel*) ist der Bereich, in dem die gebauten Pakete platziert werden (bevor sie installiert werden).

ROS Packages

- ROS-Software in Paketen organisiert
- enthalten Quellcode, Launch-Dateien, Konfigurationsdateien, Message-Definitionen, Daten und Dokumentation



Drohnen

Drohne

- unbemanntes Luftfahrzeug bzw. Roboter – Unmanned Aerial Vehicle (UAV)
- Steuerung eigenständig bzw. fernbedient
  - Embedded Computer (Mikrocontroller/SoC)
  - Bodenbasierte Fernsteuerung
- Kommunikationssystem zw. Steuerung und UAV
  - Funk, Bluetooth, Wifi (2.4 GHz)

Drohnenbauteile

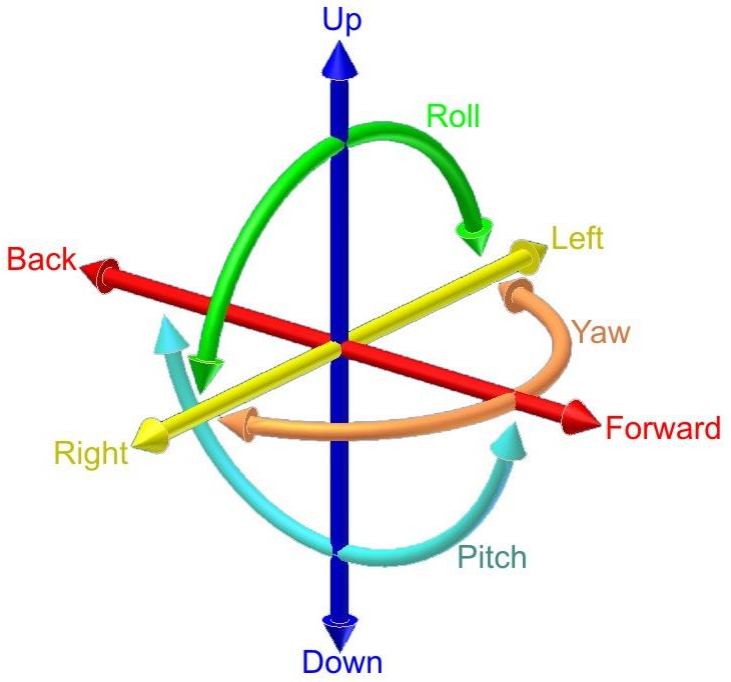
- Akku
- Funksender/-empfänger
- Flight Controller
  - eigener Mikroprozessor
- Inertial Measurement Unit (IMU) - Sensorik
  - Drehratensensoren (Kreisel / Gyroskop)
  - Beschleunigungssensoren (Accelerometer)
  - (optional) Navigation (Magnetometer oder GPS)
- Luftdrucksensor (Barometer)
- Motoren & Rotoren
- (Kamera)

Quadrotor / Quadcopter

- Luftfahrzeug / Drohne mit 4 Rotoren
- Drehgeschwindigkeit pro Rotor unabhängig
- Rotoren bewirken Auftrieb und Drehmoment (Torque)
- instabil, benötigt Sensorik für stabiles Flugverhalten

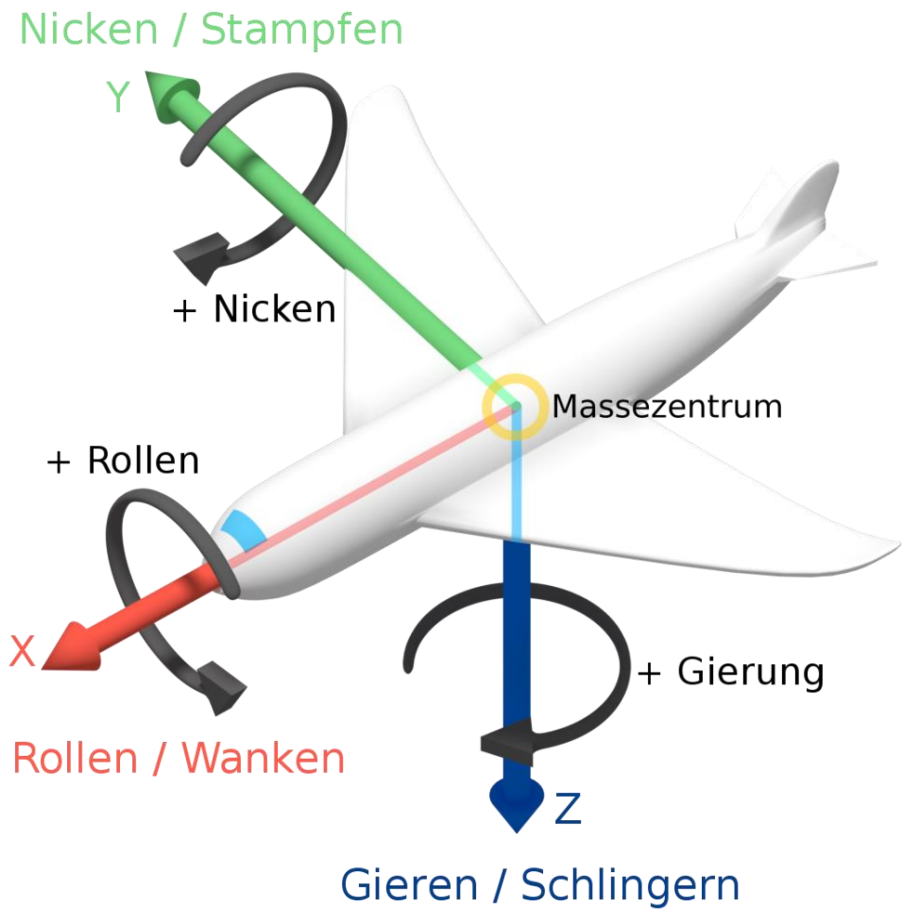
Bewegung im Raum

- 6 Freiheitsgrade
- 3-Dimensionaler Raum
- Bewegung eines Körpers im freien Raum
- X, Y, Z-Achsen (vorwärts/rückwärts, seitwärts, hoch/runter)
- Rotationsachsen
  - Roll, Pitch, Yaw
  - Längs-, Quer- und Hochachse



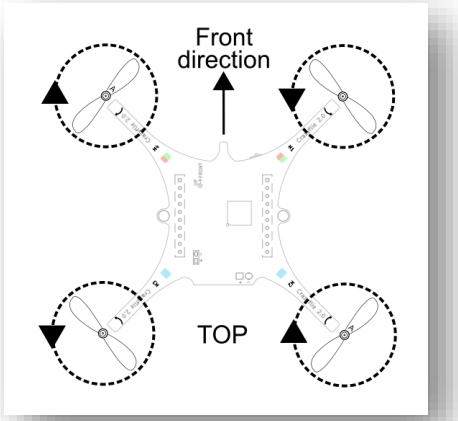
6 Freiheitsgrade (6 DoF)

Rotationsachsen (Flugzeug)

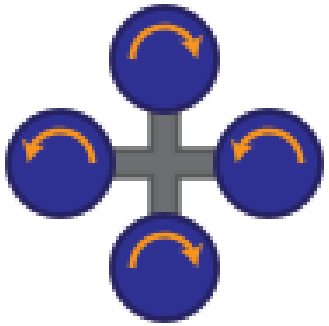


Quadcopter Flugverhalten

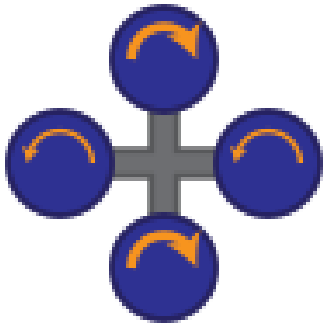
- Quadcopter hat Kontrolle über 4 DoF
  - Z-Achse (Höhe)
  - Rollen, Nicken, Gieren (Roll, Pitch, Yaw)
- Rotoren
  - 2 Rotoren drehen im Uhrzeigersinn (CW)
  - 2 Rotoren drehen gegen Uhrzeigersinn (CCW)
- Auftrieb durch Erhöhung oder Verringerung der Motordrehzahl
- Vortrieb durch Neigung der Rotorebene
- Rollen, Nicken (Roll, Pitch) durch Änderung des Schubzentrums
- Gieren (Yaw) durch Änderung des Drehmoments



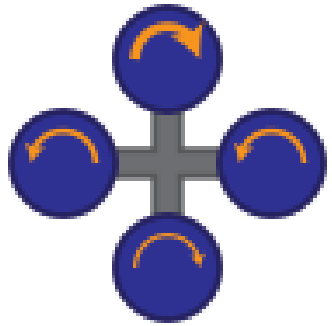
Quadcopter Flugverhalten



Ein Quadrotor schwebt (Hover) oder ändert seine Höhe, indem er auf alle vier Rotoren den gleichen Schub ausübt.



Ein Quadrotor passt sein Gieren (Yaw) an, indem er mehr Schub auf die in eine Richtung rotierenden Rotoren ausübt.



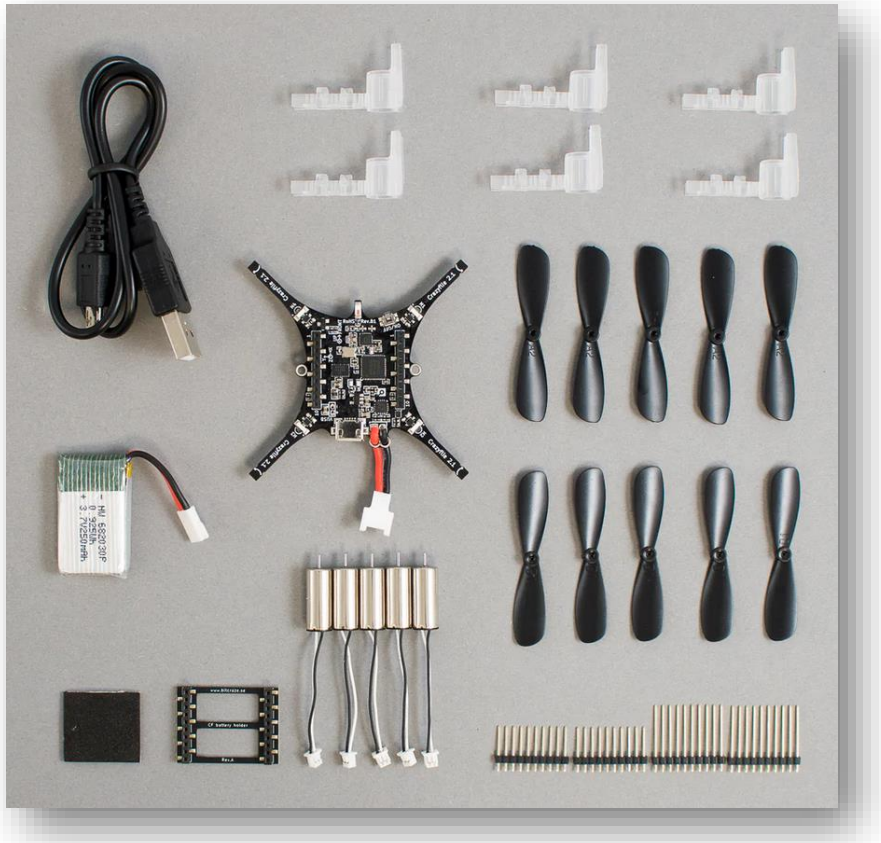
Ein Quadrotor passt sein Nicken (Pitch) oder Rollen (Roll) an, indem er mehr Schub auf einen Rotor (oder zwei benachbarte Rotoren) und weniger Schub auf die gegenüberliegenden Rotoren ausübt.

Bitcraze Crazyflie 2.1

- Nano Quadcopter von Bitcraze
- für Indoor-Einsatz, Forschung
- 27g Gewicht, passt in Handfläche
- Open-Source Firmware
  - Python API für Steuerung
- Kosten: ~230 €
- Flugzeit: 6-7 Minuten
- Ladezeit: 35-40 Minuten
- Maximal empfohlenes Nutzlastgewicht: 15 g

Bitcraze Crazyflie 2.1

- Kommunikation über USB-Funk-Dongle (2.4 GHz) oder Bluetooth Low Energy (Mobilgerät)
  - CrazyRadio
- Einfach zusammenzubauen
- Bauteile:
  - Control Board (SoC)
  - LiPo-Batterie
  - 4 Motoren + Rotorblätter
  - Plastikbeine



Drohnenschwärme

Autonomie

- Was bedeutet Autonomie oder autonomer Betrieb?
- Selbsbestimmung, Selbstverwaltung oder Entscheidungs- bzw. Handlungsfreiheit

- Autonome Drohne
  - **erlaubt keinen Eingriff in den Flugeinsatz und Steuerung der Drohne**
  - befindet sich aber in Modus, in dem die Drohne **selbstständig agiert**
  - impliziert künstliche Intelligenz

Automatisierung

- Was bedeutet Automatisierung?
- Übernahme von Prozessen und Vorgängen durch Maschinen oder Roboter

- Autonome bzw. Automatisierte Flugmanöver
  - Drohnen bzw. ihre Systeme führen Aufgaben **selbstständig** durch
  - Kontrolle durch Mensch (**Befehlserteilung**) notwendig
  - Eingriff in Flugeinsatz / Steuerung möglich

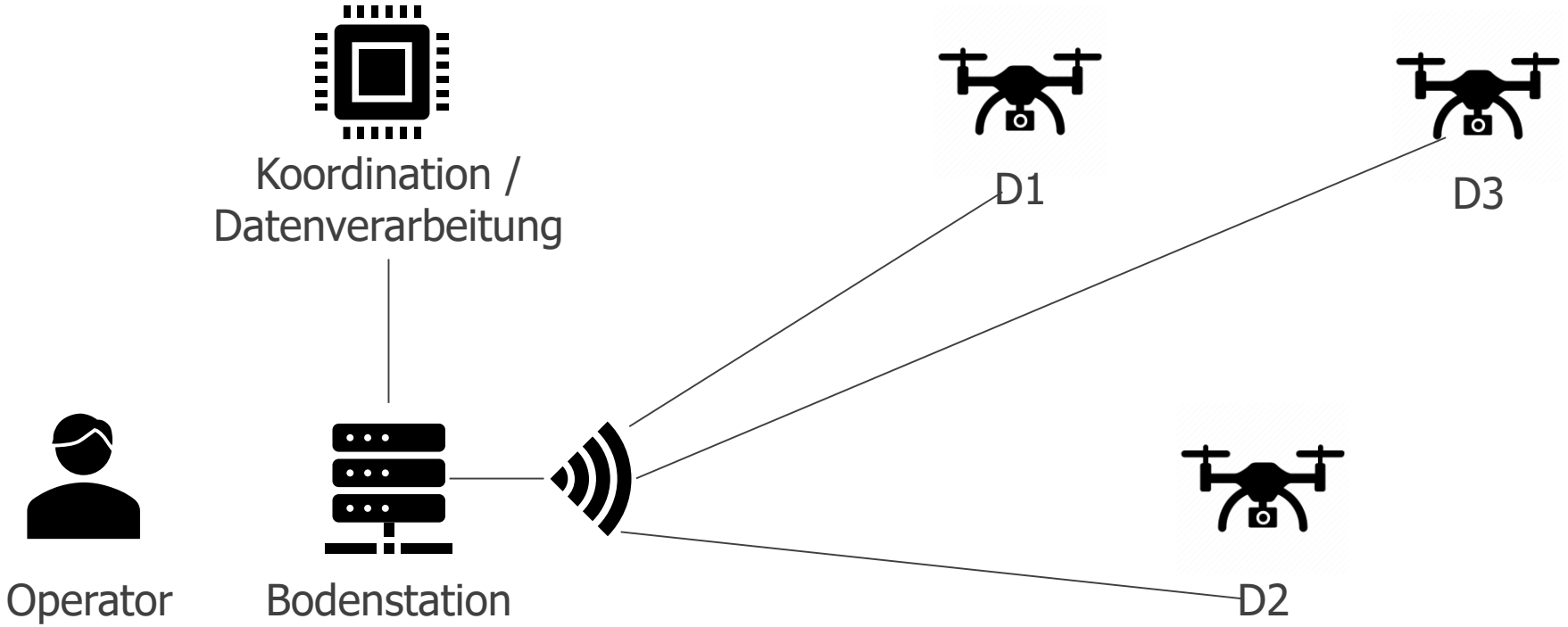
- waypoint navigation, trajectory following mittels GPS

Drohnenschwärme

- Was ist ein Drohnenschwarm?
- "Ein Drohnenschwarm (Unmanned Aerial Vehicles, UAVs) ist eine Gruppe von Drohnen, die zusammenarbeiten, um ein bestimmtes Ziel zu erreichen"
- jede Drohne hat Datenerfassungs- und –verarbeitungsaufgaben
  - setzt Rechenkapazität auf Drohne voraus
  - weitere, rechenintensivere Verarbeitung auf Bodenstation

Quelle: https://www.researchgate.net/publication/336282175_Swarms_of_Unmanned_Aerial_Vehicles_-_A_Survey

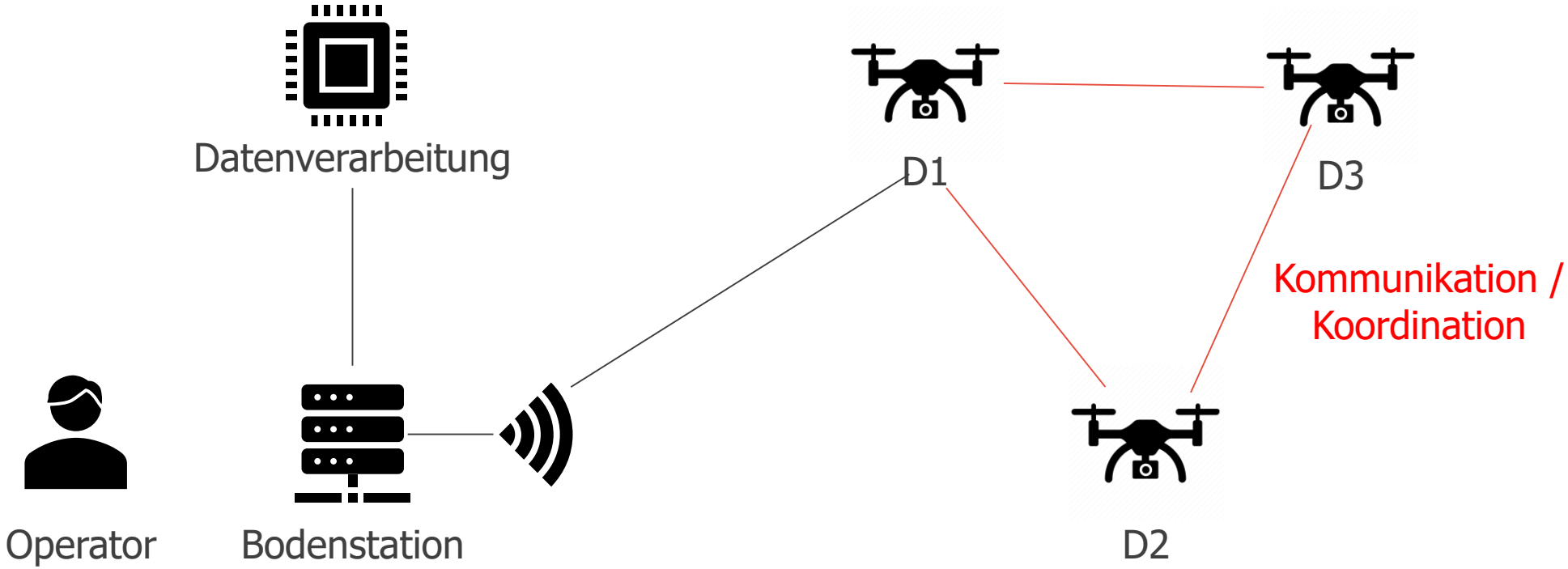
Beispiel Drohnenschwarm 1



Semi-autonomer Drohnenschwarm

- Nicht autonom, semi-autonom
- Bodenstation (GCS) dient als zentrale Steuerung
- Drohnen treffen keine Entscheidungen,
  auch keine unabhängigen
- Keine Kommunikation zwischen Drohnen
- Keine Koordination zwischen Drohnen
- jede Drohne arbeitet für sich

Beispiel Drohnenschwarm 2



Autonomer Drohnenschwarm

- Schwarm an sich autonom
- Bodenstation dient der Datenverarbeitung
- Drohnen treffen Entscheidungen
- Kommunikation & Koordination zwischen Drohnen
- Drohne kennt ihre Position relativ zu anderen im Schwarm

Autonomer Drohnenschwarm

- Kommunikation
  - Zuverlässigkeit des Kanals
- Perzeption (Umgebungswarnehmung)
  - Robustheit gegen Kollisionen
- Planung & Entscheidungsfindung
- Künstliche Intelligenz

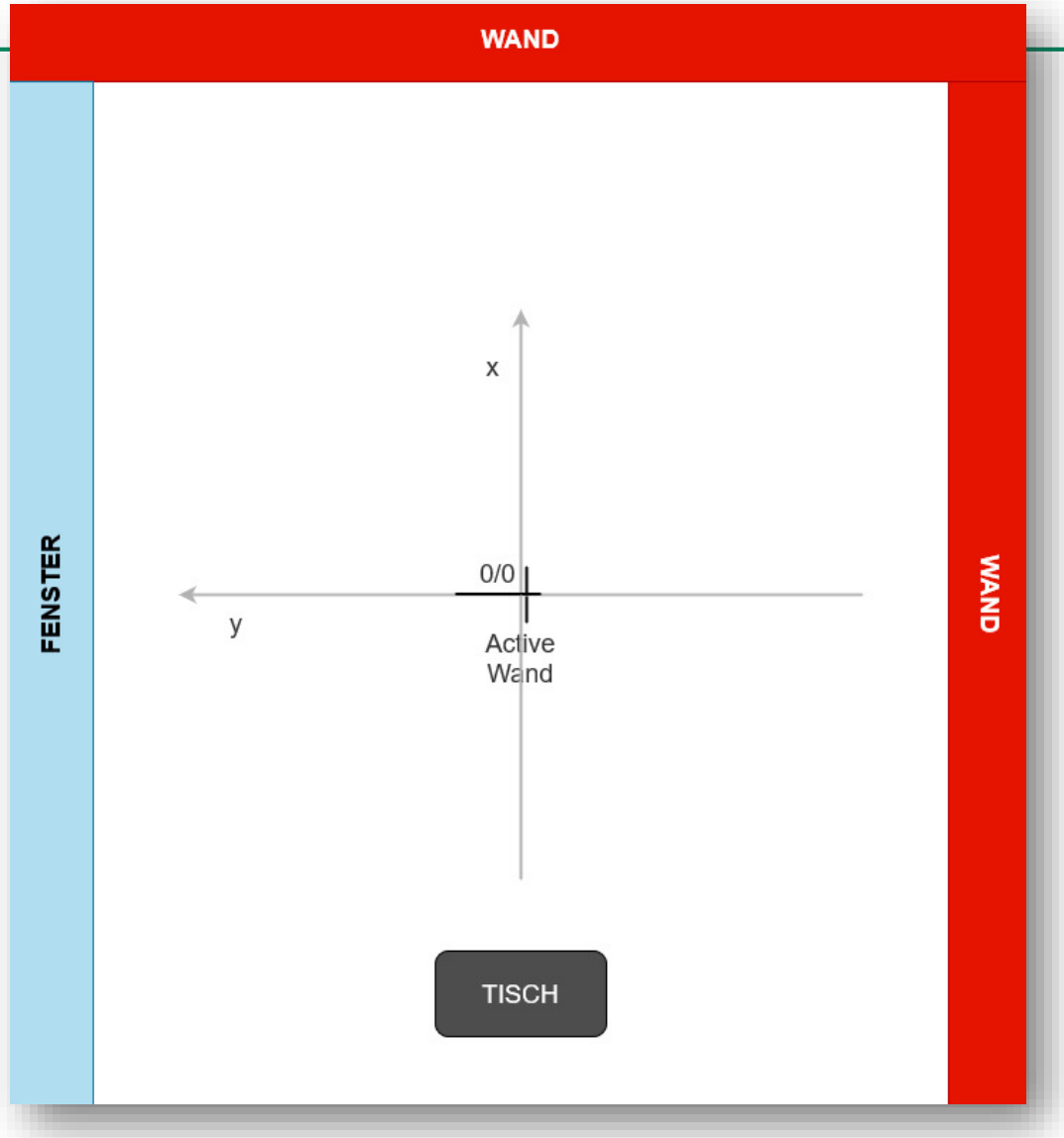
Drohnenschwärme Drohnenlabor

- „Drohnenschwärme“ in Drohnenlabor entsprechen erstem Schwarm-Typ
- https://youtu.be/Rx6lF6HphxA

Drohnenlabor

- Vicon Motion-Capture-System für Lokalisierung
  - 6 Infrarotkameras
  - Control Center-Software auf Laptop
- Bodenstation für Steuerung
  - Ubuntu-Laptop mit ROS
  - **Crazyswarm ROS-Package**
    - **Drohnenprogrammierung**

Drohnenlabor – Raum (2D)



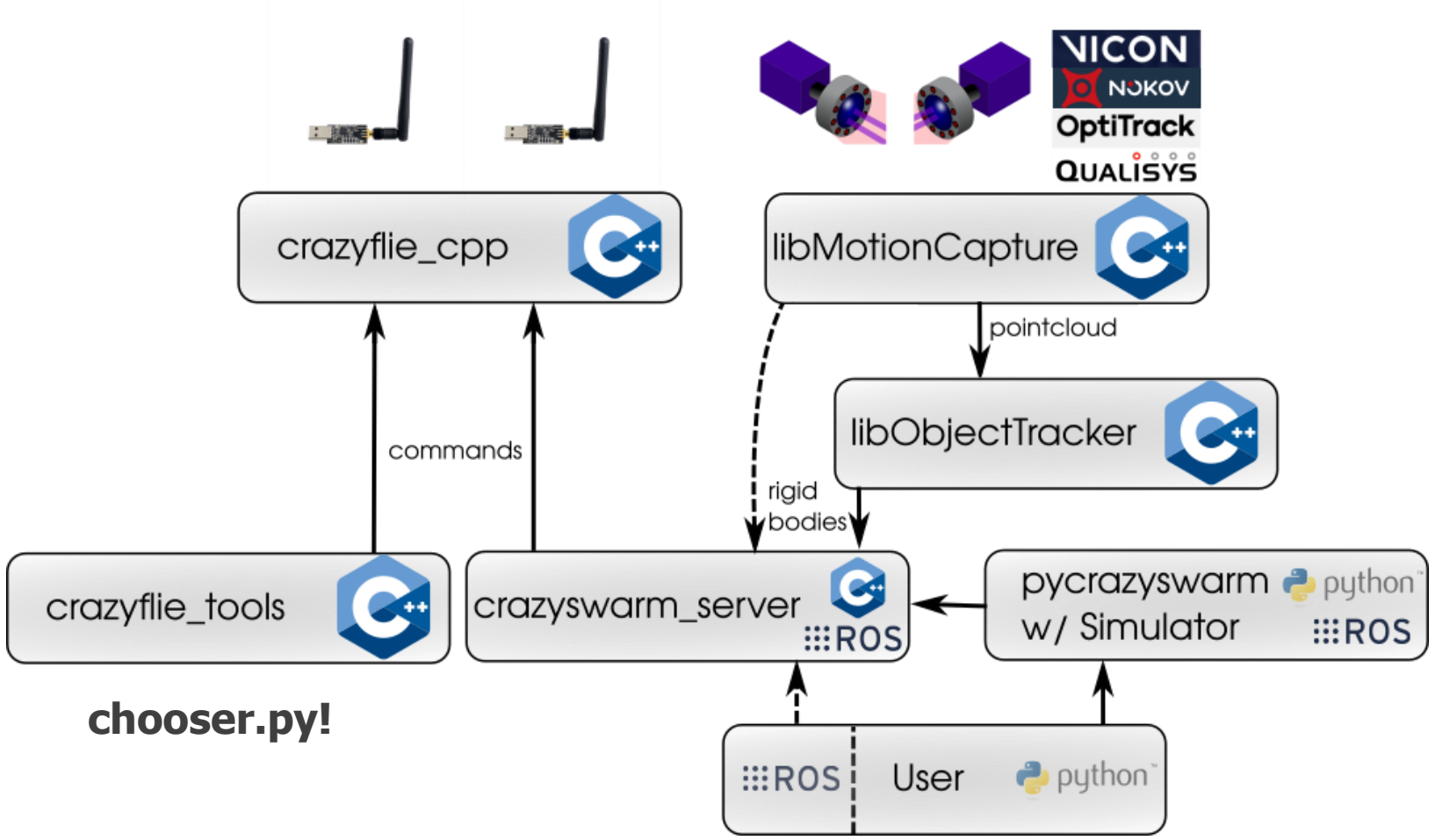
Drohnenlabor

- https://drohnenlabor.web.fh-kufstein.ac.at
- https://fh-dronelab.pages.web.fh-kufstein.ac.at/dronelab-docs/

Crazyswarm

- auf ROS-aufbauende Software für Crazyflie-Steuerung
- ermöglicht Flug in engen, synchronisierten Formationen
- verwendet Motion-Capture-Systeme zur Lokalisierung
- Flug von bis zu 49 CF-Drohnen möglich
- Single-Marker-Tracking möglich
- Python-Bibliothek für Flugmanöverprogrammierung (heute wichtig!)
- Simulator enthalten (heute wichtig!)

Crazyswarm


VICON
NOKOV
OptiTrack
QUALISYS
crazyflie_cpp
libMotionCapture
pointcloud
libObjectTracker
commands
rigid
bodies
crazyflie_tools
crazyswarm_server
ROS
pycrazyswarm
w/ Simulator
python
ROS
chooser.py!
ROS
User
python

Tool: chooser.py

- erlaubt Aus/Abwahl zu verwendender Drohnen
- erlaubt Ausschalten, Neustarten
- erlaubt Anzeigen von Batteriestatus
- erlaubt flashen von MCU (Firmware-Updates)

pyCrazyswarm Library

- https://crazyswarm.readthedocs.io/en/latest/api.html
- liegt unter */ros_ws/crazyswarm/scripts*
- Bietet Abstraktion der Drohnen- und Schwarmsteuerung
- 3 wichtige Klassen:
  - **Crazyflie** – Objekt, welches einzelne Drohne repräsentiert
  - **CrazyflieServer** – Objekt, welches Aussenden von Befehlen an alle Drohnen ermöglicht (Broadcast)
  - **TimeHelper** – Objekt, welches zeitabhängige Funktionen bereitstellt

pyCrazyswarm Library - Simulator

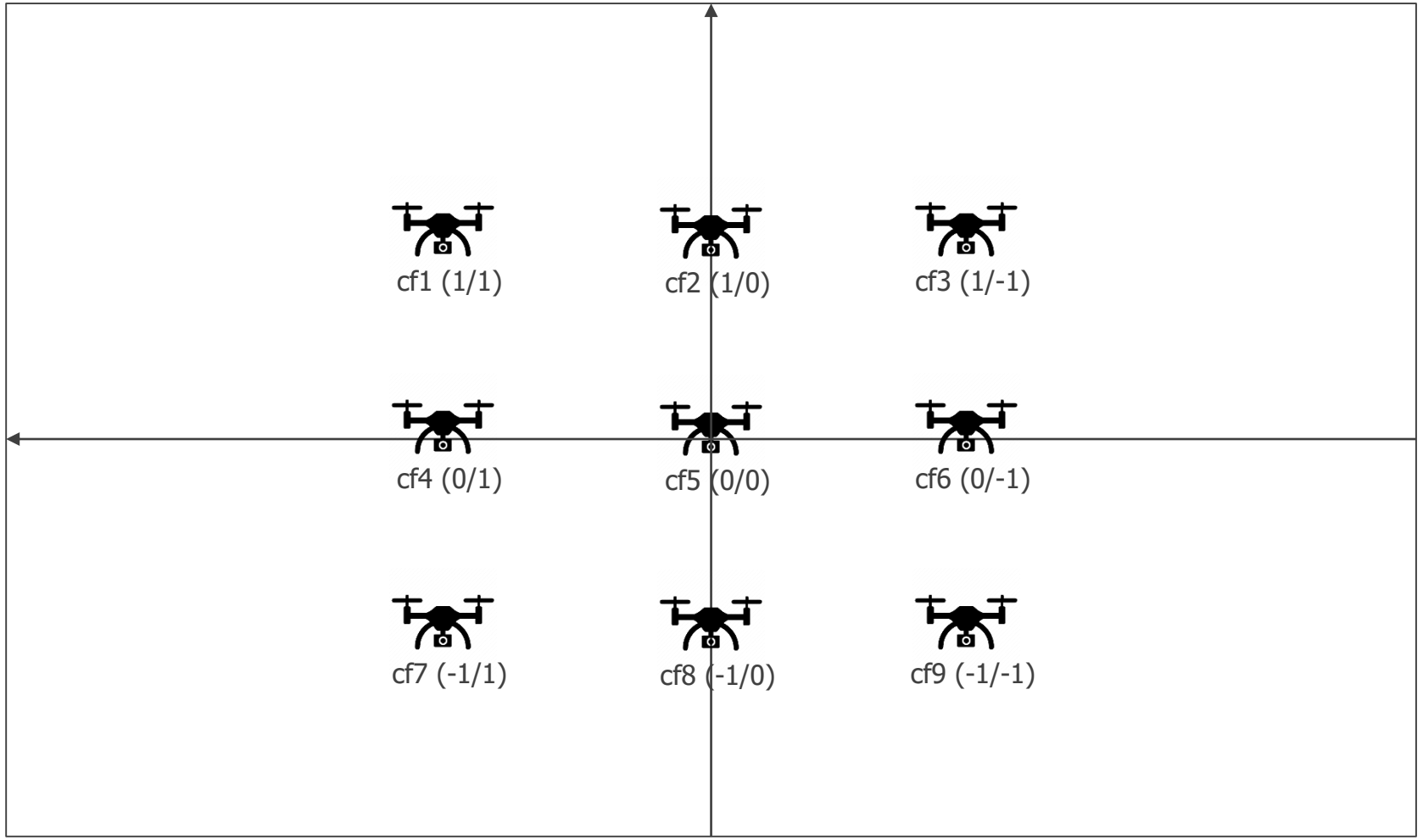
- Programmierkontext ist bei physischer Hardware und Simulation derselbe
- Korrekte Python-Scripts laufen auf physischer Hardware und in Simulation
- Simulation durch setzen von Flag aufrufbar
- Ausführung auf physischer Hardware (benötigt ROS Master etc.):

```
> python3 hover.py
```

- Ausführung in Simulator (kein ROS Master):

```
> python3 hover.py --sim
```

Drohnenpositionen - Visualisierung



Wichtige Aspekte bei Flugmanövern (Drohnenlabor)

- Gieren bzw. Rotation nicht möglich
- Bewegung entlang X/Y/Z-Achsen möglich
- Manöver nicht länger als 30 Sekunden
- für unsere Manöver maximal +/-2 Meter in alle Richtungen (X/Y/Z)
- Maximale Annäherung: seitwärts 50cm, übereinander 1 Meter (Downwash)
- initiales Abheben und abschließendes Landen mind. 2 Sekunden pro Meter Höhe
- nach jeder Bewegung muss timeHelper aufgerufen werden!

Skriptbeispiel 1: Grundlagen

```
1   from pycrazyswarm import *
2   import numpy as np
3
4   if __name__ == "__main__":
5       swarm = Crazyswarm()
6       timeHelper = swarm.timeHelper
7       allcfs = swarm.allcfs
8
```

Crazyswarm-Objekt initialisieren
TimeHelper und CrazyflieServer (allcfs) in Variablen speichern

```
9       allcfs.takeoff(targetHeight=1.0, duration=2.0)
10      timeHelper.sleep(2.0)
```

Über Server takeoff-Befehl mit Höhe und Dauer an alle cfs aussenden (Broadcast)
timeHelper muss aufgerufen werden und dieselbe Dauer wie der vorherige Befehl schlafen!

```
16      cfs.land(targetHeight=0.02, duration=2.0)
17      timeHelper.sleep(2.0)
```

Über Server land-Befehl mit Höhe und Dauer an alle cfs aussenden (Broadcast)

Skriptbeispiel 1: Grundlagen

```
14      cfs = swarm.allcfs
15      cf1 = swarm.allcfs.crazyflies[0]
16      cf2 = swarm.allcfs.crazyflies[1]
17
```

Einzelne CFs aus Array selektieren

```
for cf in allcfs.crazyflies:
    pos = np.array(cf.initialPosition) + np.array([0, 0, 2.0])
    cf.goTo(pos, 0, 1.0)
```

über CFs iterieren (for-loop)
Neue Position, relativ zu initialer Position einer einzelnen CF, berechnen
mit goTo-Funktion bestimmte Position an CF senden

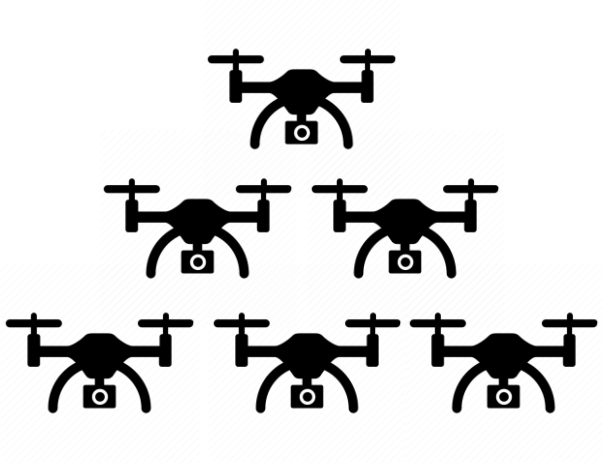
timeHelper: Außerhalb der Schleife schlafen!

Skriptbeispiel 2: Hovering

```
3   from pycrazyswarm import Crazyswarm
4
5   HOVER_DURATION = 5.0
6   TARGET_HEIGHT=1.5
7
8   if __name__ == "__main__":
9       swarm = Crazyswarm()
10      timeHelper = swarm.timeHelper
11      cfs = swarm.allcfs
12
13      cfs.takeoff(targetHeight=TARGET_HEIGHT, duration=2.0*TARGET_HEIGHT)
14      timeHelper.sleep(2.0*TARGET_HEIGHT + HOVER_DURATION)
15      cfs.land(targetHeight=0.02, duration=TARGET_HEIGHT*2.0)
16      timeHelper.sleep(TARGET_HEIGHT*2.0)      You, now • Uncommitted changes
17
```

Crazyswarm Aufgabe 1: Dreieck

- Drohnen 1, 2, 3, 4, 5, 6 (chooser.py!)
- Dreieck in der Z-Achse bilden (übereinander)



Crazyswarm Aufgabe 2: Quadrat fliegen

- Drohnen 1, 8, 9 (chooser.py!)
- Jede Drohne soll, relativ zu ihrer Ausgangsposition:
  - 1 Meter in positive Y-Richtung
  - 1 Meter in positive X-Richtung
  - 1 Meter in negative Y-Richtung
  - 1 Meter in negative X-Richtung (ursprüngliche Position erreicht)
  - auf ursprünglicher Position landen

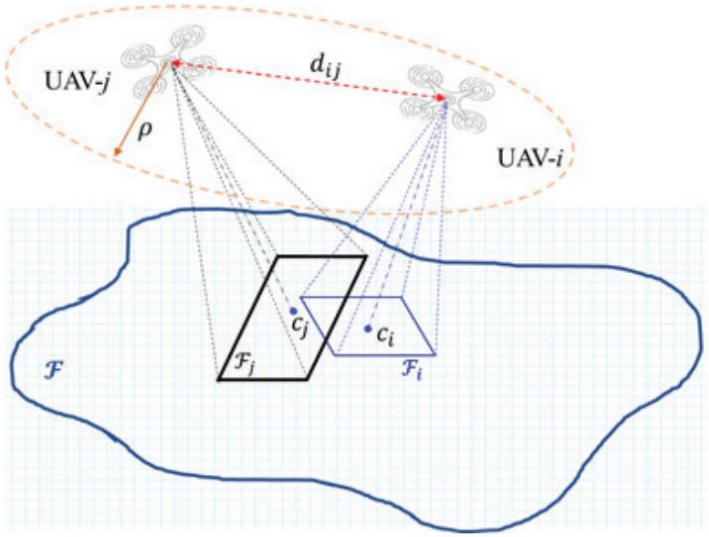
Ausblick - Einsatzszenarien autonomer Drohnen und -schwärme

- Gelände-/Grundstücküberwachung
  - Einsparen von Sicherheitspersonal
  - Effizientere Überwachung von Gelände/Bereich
- Katastrophenhilfe
  - Gefahrenguterkennung
  - Bekämpfung von Brandherden
- Umweltkartierung (Environmental Mapping)
  - Erkundung von Höhlen (Archäologie)
  - Flüsse und Überschwemmungen
  - Gebäudervermessung

Ausblick - Einsatzszenarien autonomer Drohnen und -schwärme

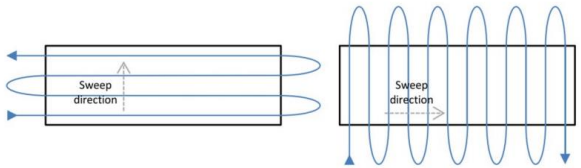
- Vermisstensuche / Search and Rescue
  - Effizientes Abfliegen großer Areale
  - Erkennung vermisster Personen
  - auch Rehkitzsuche
- Drone Delivery
  - Logistik
  - Amazon Prime Air

FH-Projekt - Area Coverage (3D CPP)



Goal:
scan pre-defined area with a (swarm of) UAV agent(s)

Traditional Approaches:
Calculating a predefined trajectory and executing it on the agent:



This is problematic when the environment or the agent changes (e.g UAV crashes, battary status), since recalculating is computational expensive.

Solution:
A deep Reinforcement Learning model suggest an action for every Agent at each timestep