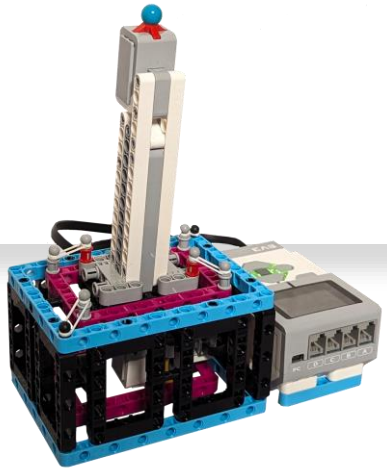
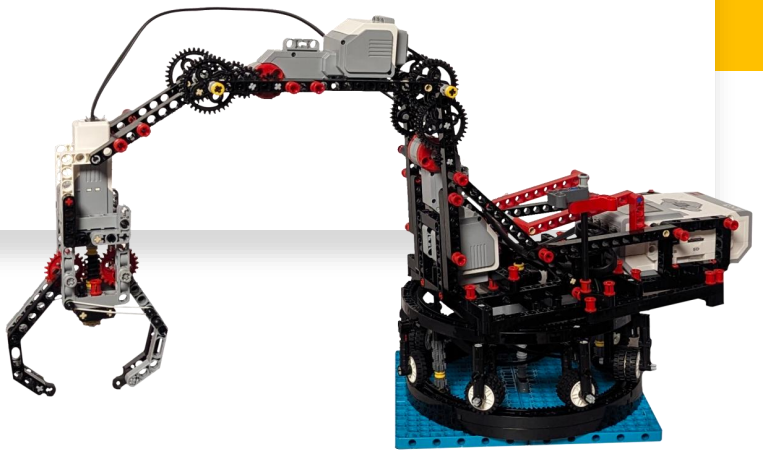
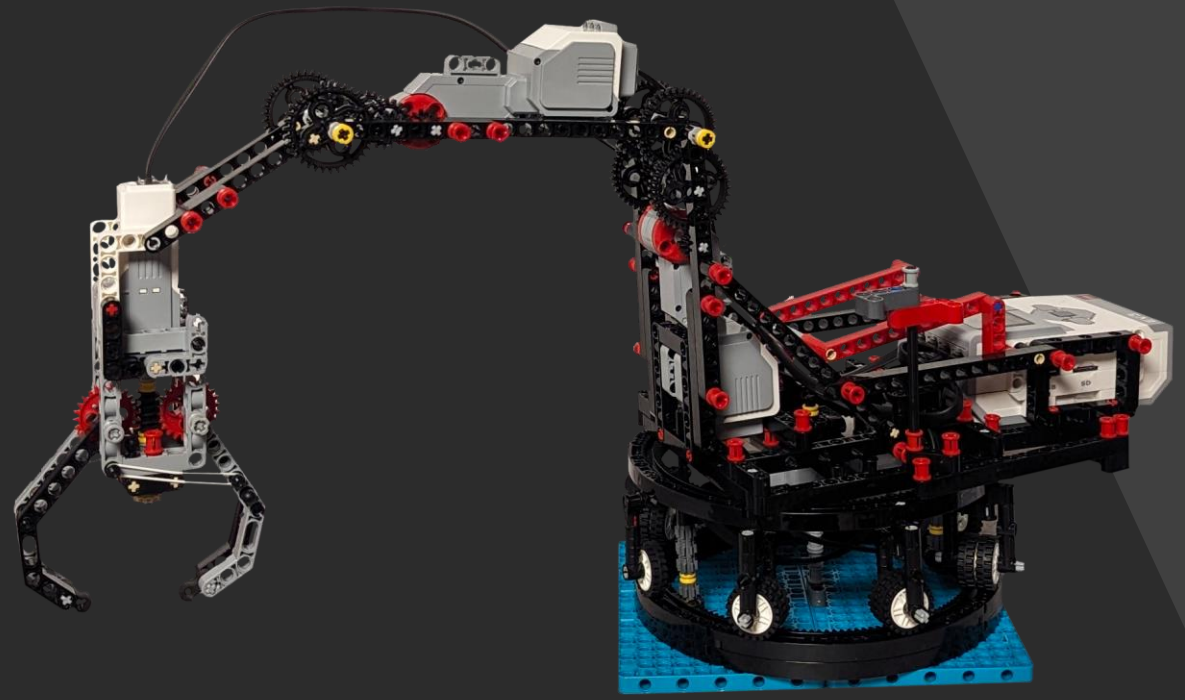
Projektarbeit Lego-Roboterarm

Kevin Hild – Lorenz Windisch





Inhaltsverzeichnis

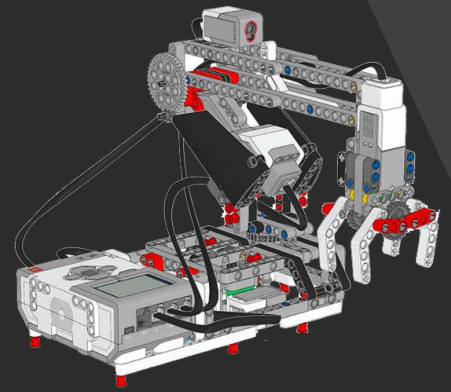


Ideen und Konzepte

Konzept/Idee

Erste Überlegungen

- Roboter-Arm
- EV3 (weil starke Motoren)
- Scratch
- Erkennen von Farben mit Sensor
- automatische Steuerung



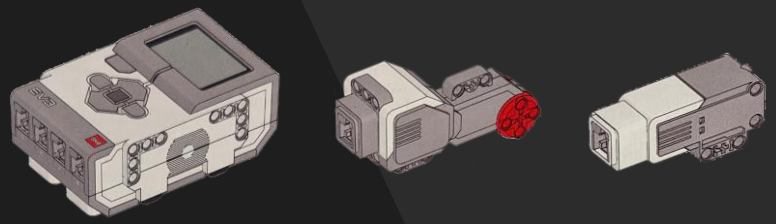
Finale Idee

- Manuelle Steuerung durch Joystick (wiederverwendbar)
- Kein Farbsensor (zu wenig Ports, lieber Steuerung mit Joystick)

Konzept-Aufbau

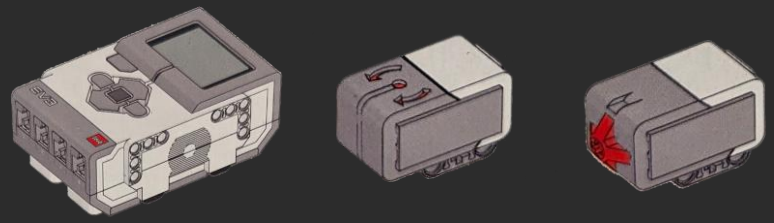
Roboterarm

- 1 Hub [EV3-Roboterarm]
- 2 Motoren für den Arm [B,C]
- 1 Motor für die Zange [D]
- 1 Motor für Plattform [A]



Joystick

- 1 Hub [EV3-Joystick]
- 2 Gyrosensoren [S1,S2]
- 1 Drucksensor [S3]



Probleme und Lösungen

Problemstellen

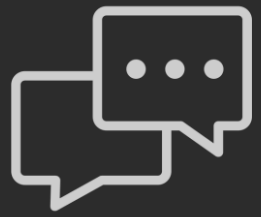
Roboterarm (Hub A)

- Plattform drehen: Motor A
- Roboter-Arm bewegen: Motoren B + C
- Zange: Motor D

Joystick (Hub B)

- Joystick-Aufbau
- Gyrosensoren-Drift S1 + S2

Kommunikation Hub2Hub



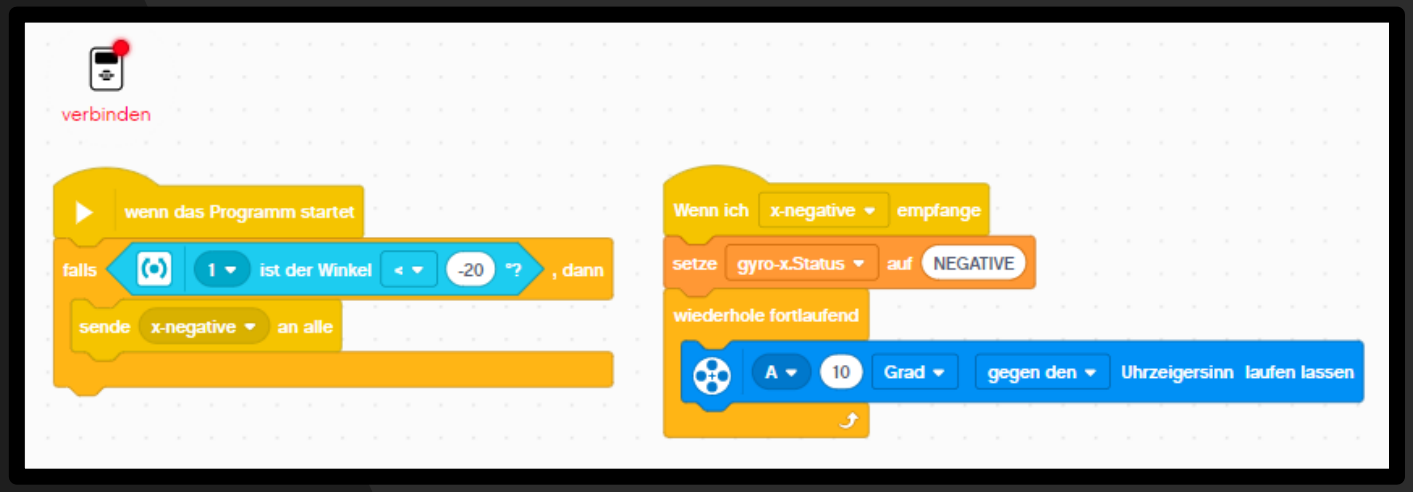
Hub-Kommunikation

Anforderungen:

- Hub B sendet Sensorwerte an Hub A
- Hub A wertet Sensorwerte aus und reagiert

Hub-Kommunikation

Idee: Messages über Scratch 3.0



Problem: Nur innerhalb von einem Programm/Hub möglich, kein Hub2Hub via Bluetooth

Hub-Kommunikation

Lösung: MicroPython

- Bluetooth-Kommunikation zwischen Hubs möglich
- flexible Programmierung möglich
- Wiederverwendbarkeit/Anpassung für andere Projekte

Alternative: LEGO MINDSTORMS LabView

- Bluetooth-Kommunikation wäre möglich
- Einarbeitung nötig
- Nicht so flexibel + wiederverwendbar wie MicroPython

Anforderungen Roboterarm

- Einzelne Komponenten dürfen nicht zu schwer sein
- Motoren müssen Gewicht bewegen und halten können
- Möglichst große Reichweite für den Arm

Plattform

Anforderungen:

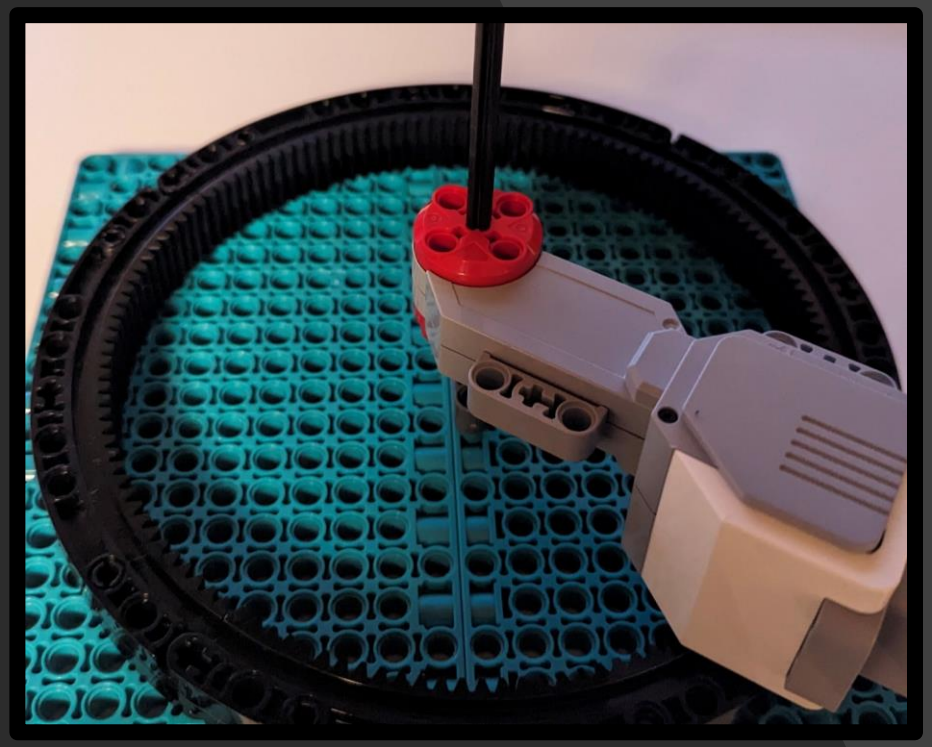
- muss komplettes Gewicht des Roboterarm drehen können
- Motor soll nicht ganze Last halten müssen

Plattform

Version 1:

- Motor dreht Plattform
- Achse vom Motor zur Plattform-Mitte

Problem: viel Last auf Motor



Plattform

Version 2:

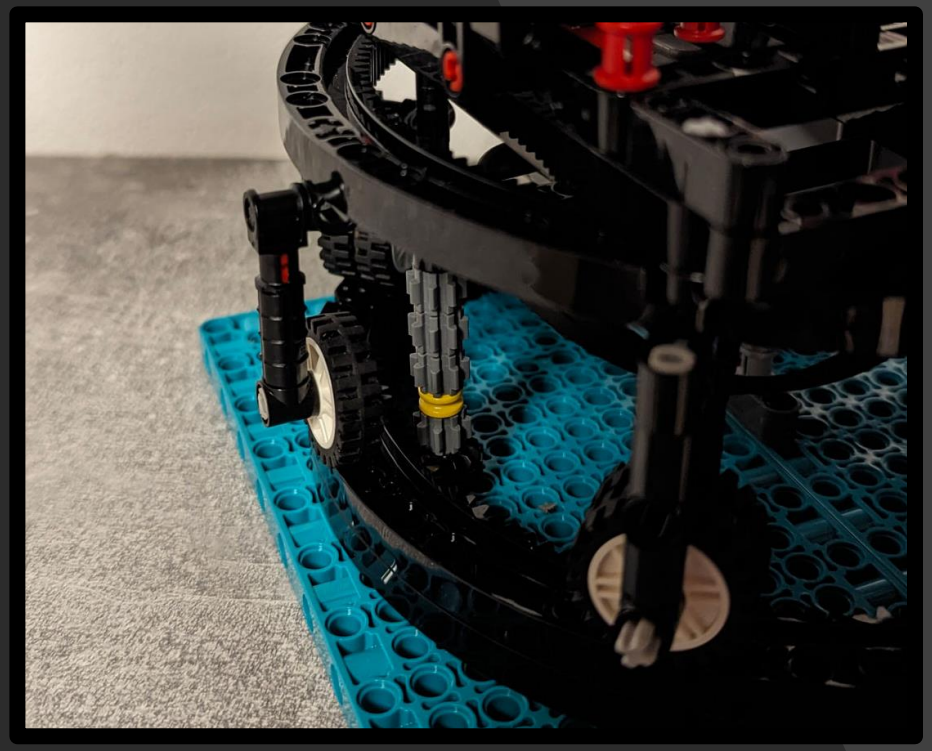
- Motor dreht Zahnrad
- Plattform liegt auf Zahnrad-Kreis mit Rädern

Problem: Räder zu klein -> nicht stabil genug und Reibung

Plattform

Version 2.1:

- wie Lösung 2, aber mit größeren Rädern
- Räder können sich besser bewegen



Roboterarm

Anforderungen:

- muss komplettes Gewicht des Roboterarm halten können
- Motor A muss am meisten Last bewegen + halten

Roboterarm

Version 1:

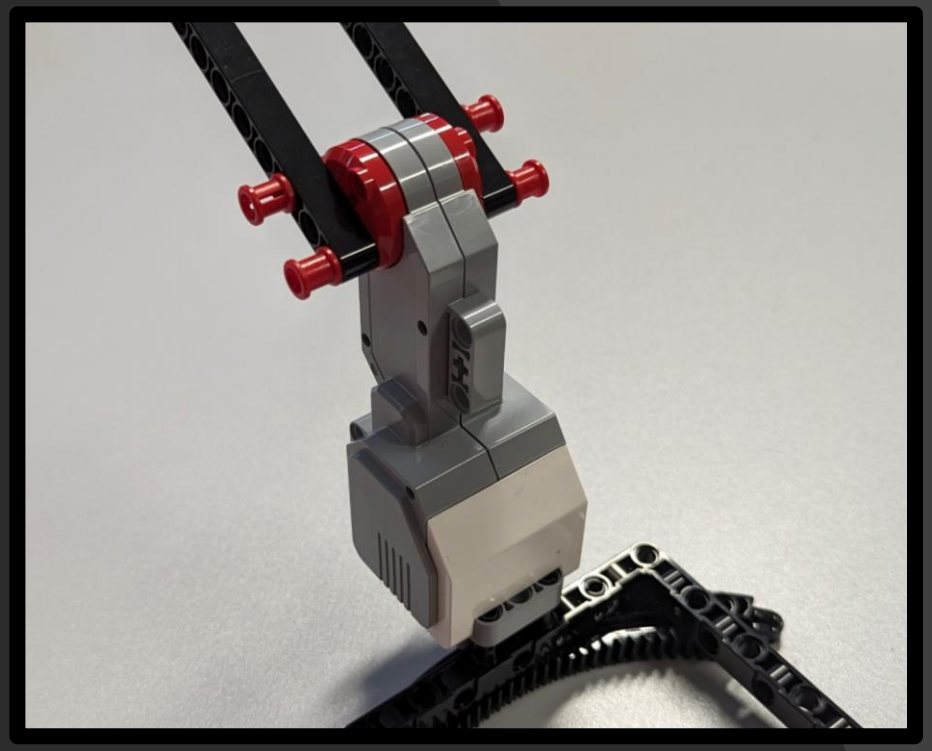
- Nur 1 Basis-Motor

Problem: nicht genug Kraft

Version 2:

- 2 Basis-Motoren

Problem: nicht genügend Ports

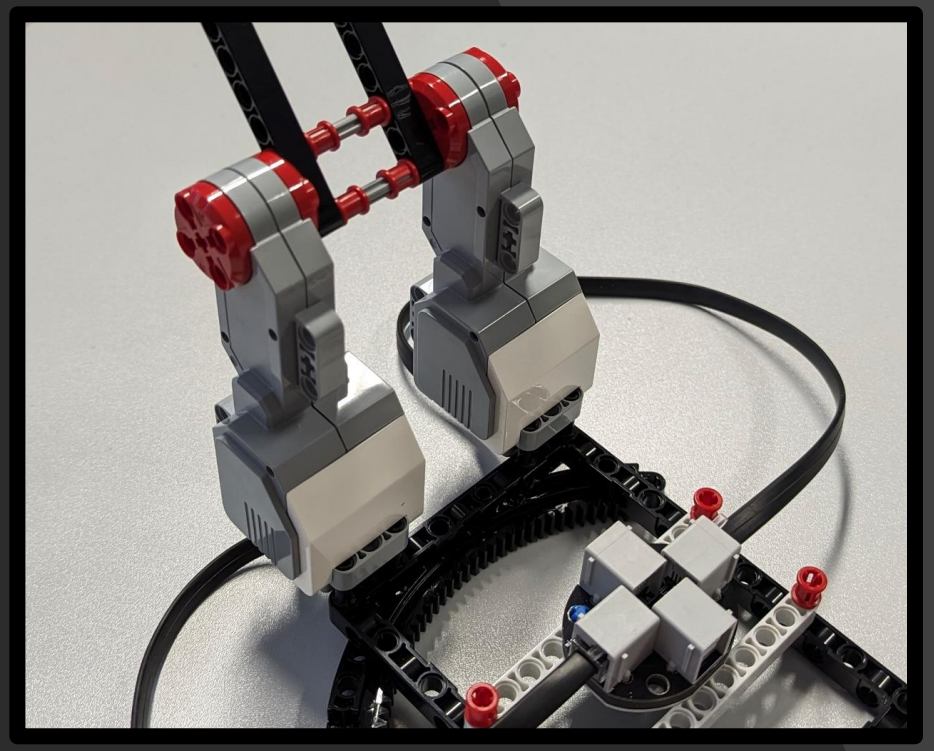


Roboterarm

Version 2.1:

- 2 Basis-Motoren + Multiplexer

Problem: extra Stromzufuhr, nur alte Software



Version 2.2:

- 2 Basis-Motoren + Splitter

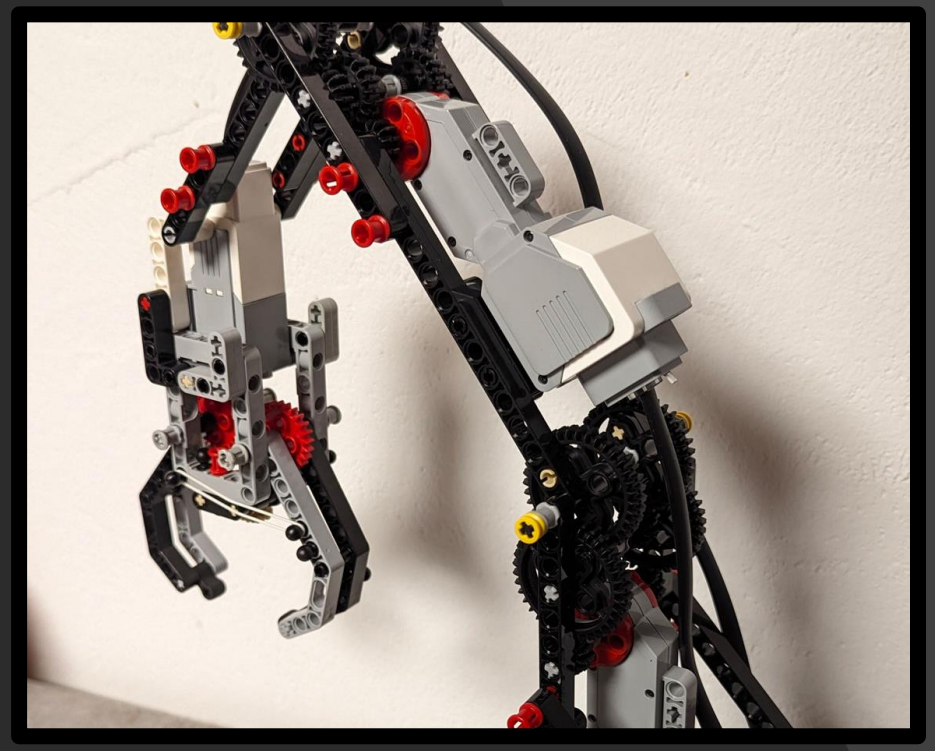
Problem: Splitter funktioniert generell, aber über MicroPython nicht verwendbar

Problem: 2 Motoren stark genug zum Bewegen, aber nicht zum Halten

Roboterarm

Version 3:

- 1 Motor
- Motoren-Übersetzung mit Zahnrädern
- genügend Kraft
- adaptierbar für Motor 2+3



Greifzange

Anforderungen:

- muss Gewicht vom Objekt (Korb) halten, ohne zu öffnen
- Sollte nicht zu schwer sein (muss vom Arm gehalten werden)

Greifzange

Version 1:

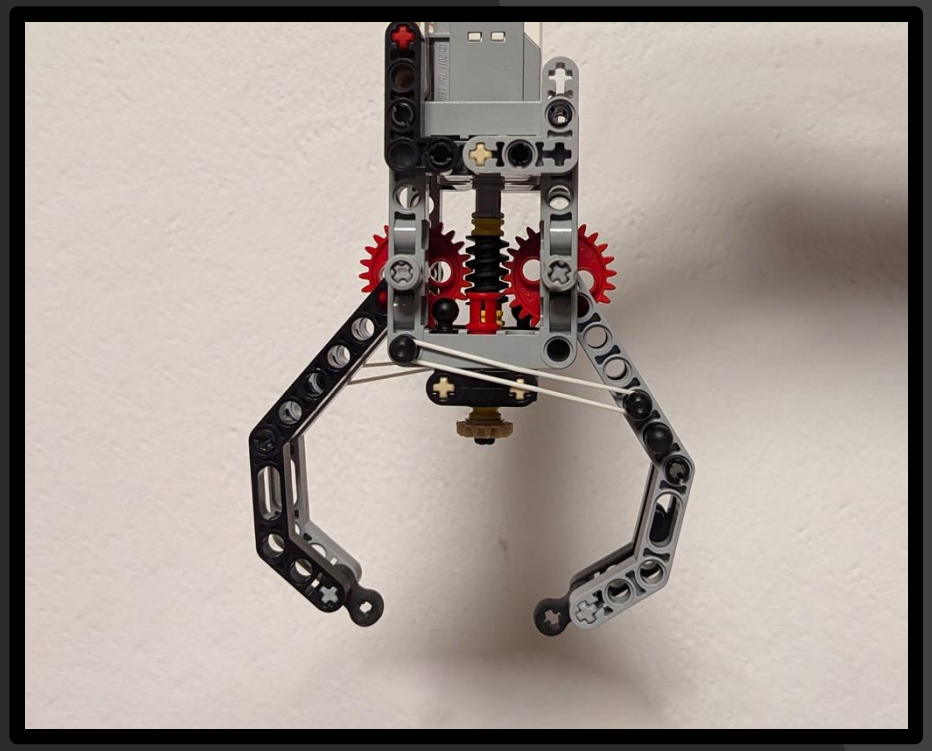
- Greifen nur mit Zahnrädern

Problem: nicht genug Kraft

Greifzange

Version 2:

- Greifen mit größeren Zahnrädern
- Zange durch Gummis geschlossen
- Zange öffnen: Motor
- Zange schließen: Motor + Gummis



Joystick

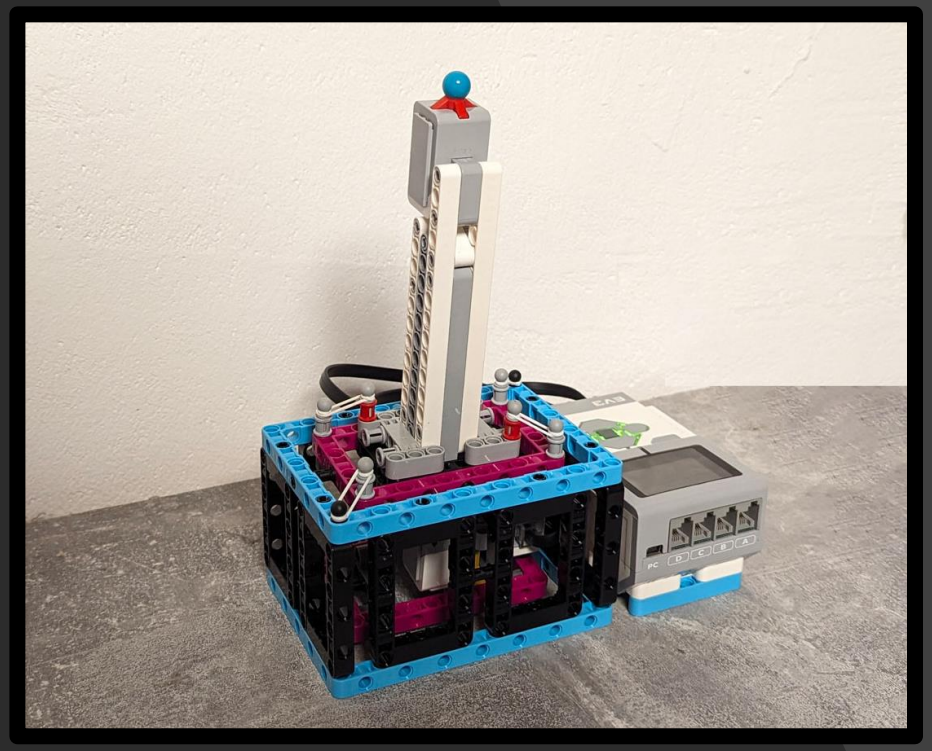
Anforderungen:

- Joystick muss sich in 4 Richtungen bewegen können
- Joystick muss automatisch in neutrale Position zurück
- 2 Gyrosensoren müssen richtig angebracht sein
- Drift der Gyrosensoren muss beachtet werden

Joystick

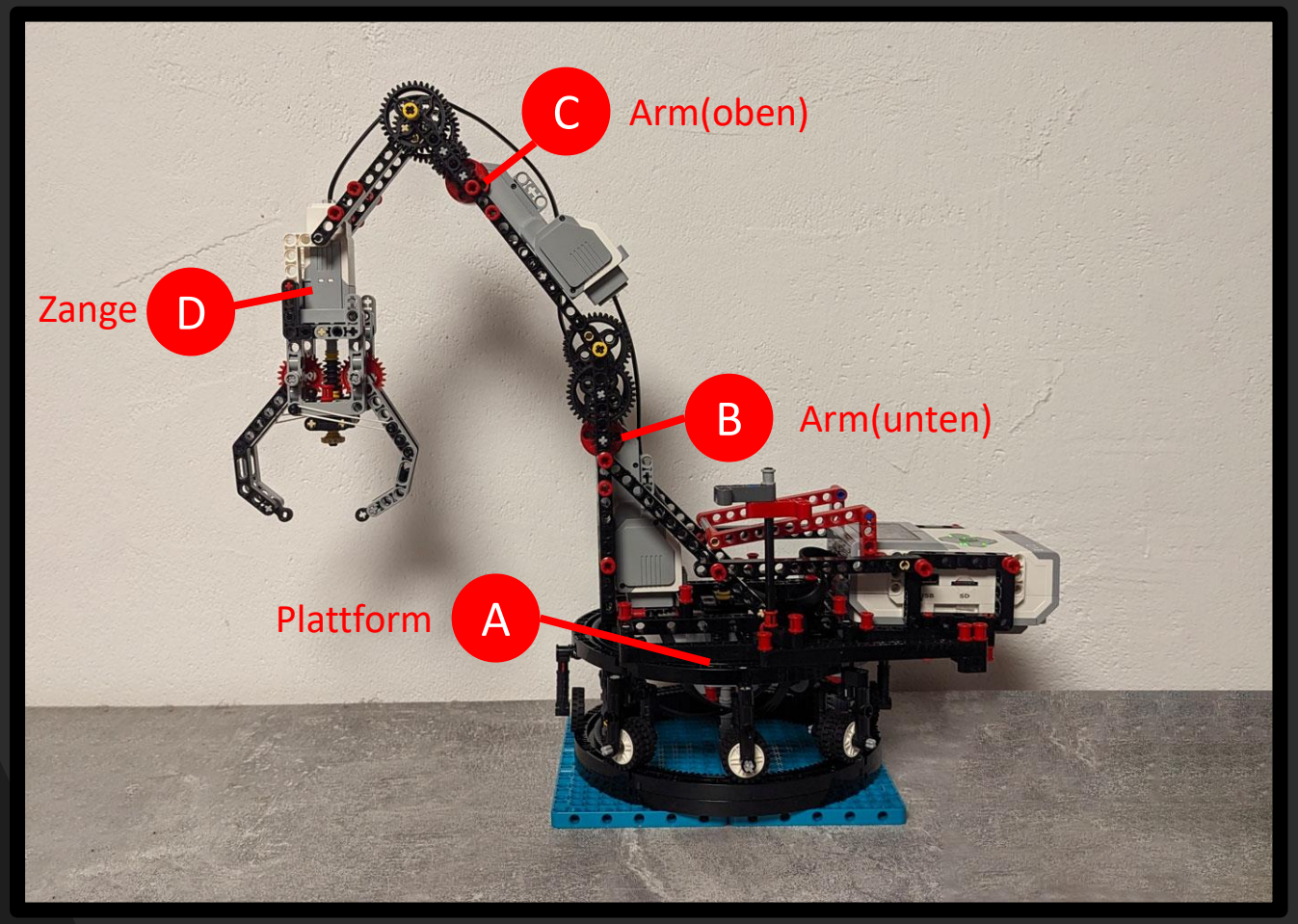
Lösung:

- Joystick mittels Gummis zentriert
- Bewegung in 4 Richtungen möglich
- Drift wird softwareseitig behandelt
  - Lässt sich bei diesem Aufbau nicht vermeiden

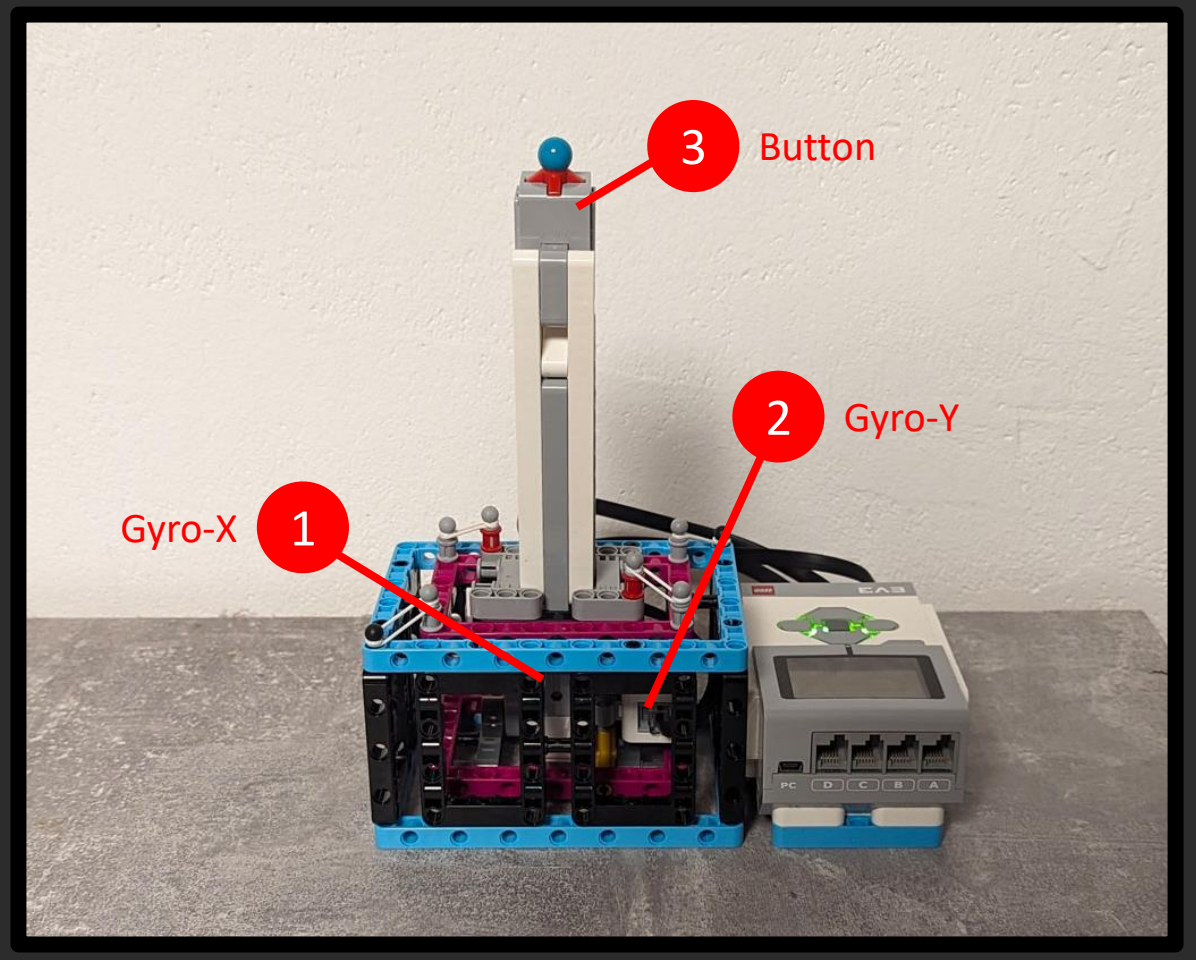


Aussehen und Funktion der Roboter

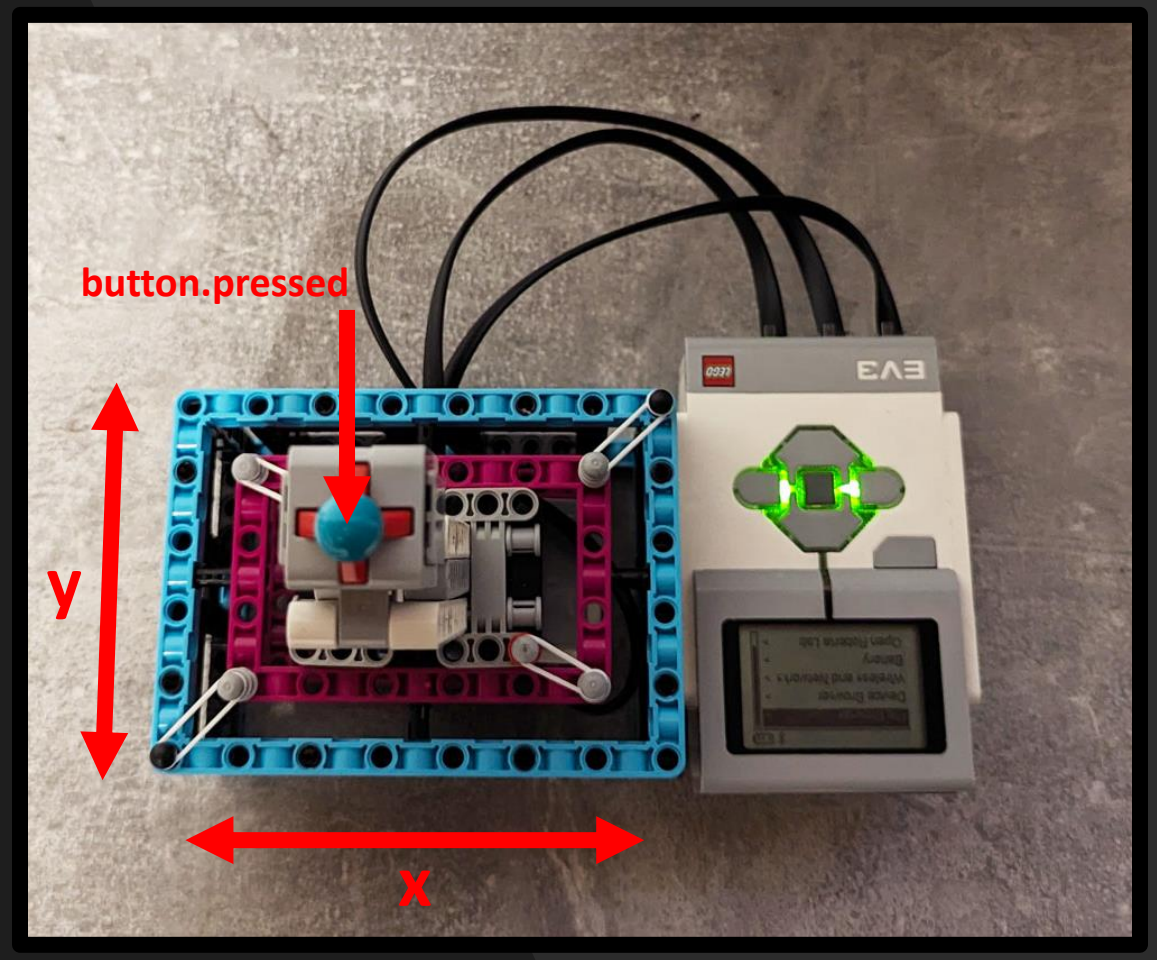
Roboterarm + Motoren



Joystick + Sensoren



Steuerung



Joystick nach links/rechts bewegen

- Plattform links/rechts drehen

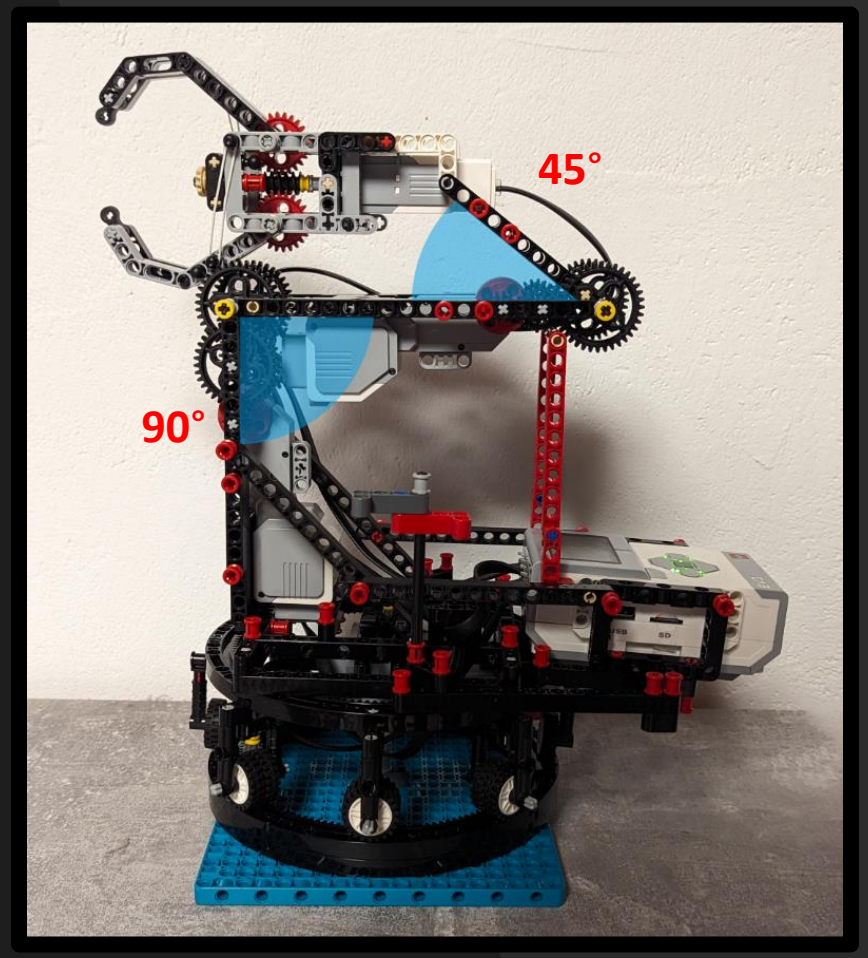
Joystick nach vorne/hinten bewegen

- Roboterarm ausfahren/einfahren

Button drücken/loslassen

- Zange öffnen/schließen

Kalibrierung



Roboterarm

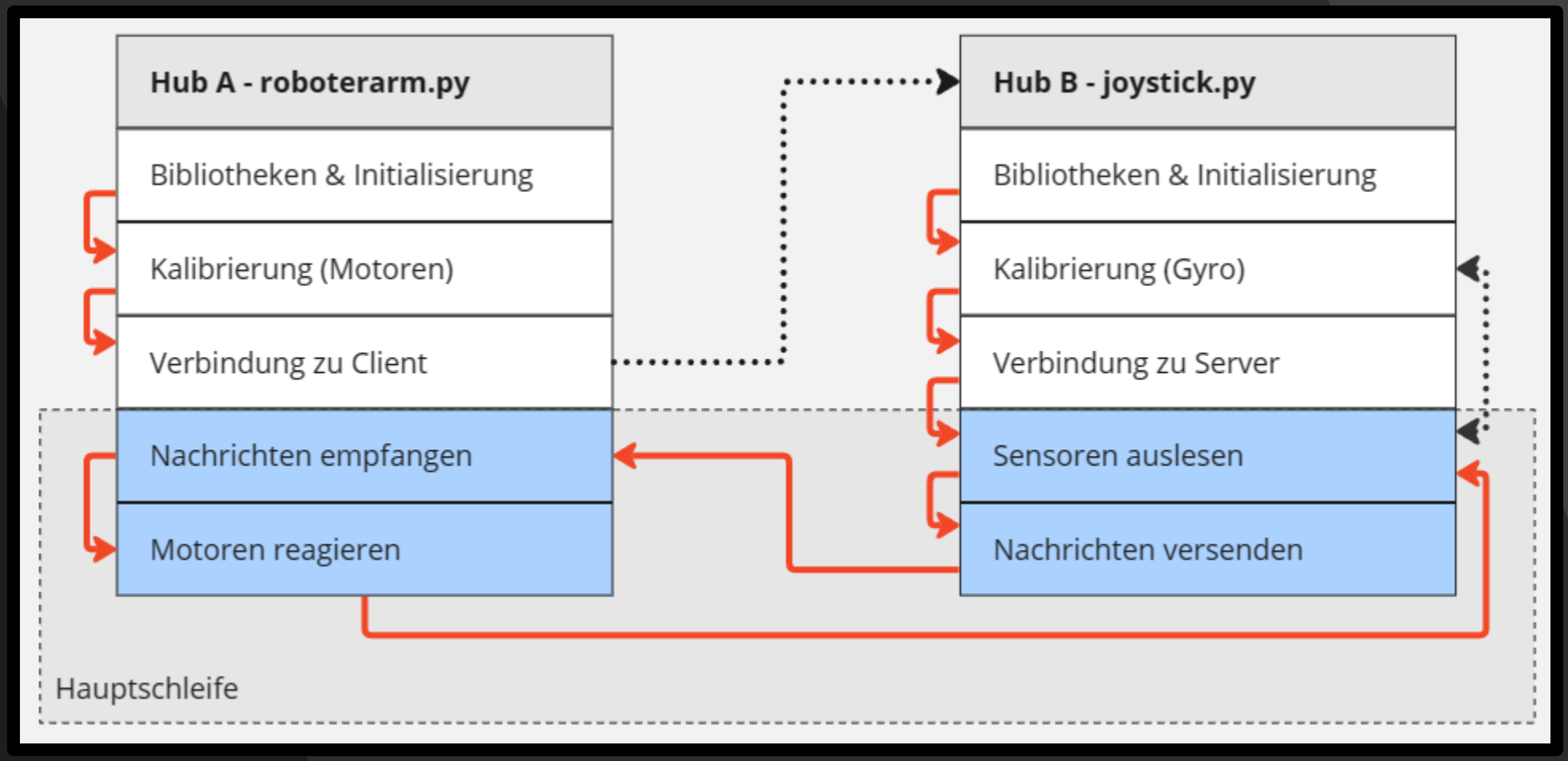
- Ausrichten wie im Bild (Startposition)
- Motoren lassen sich über Programm robotarm_motor.py einzeln steuern

Gyrosensor

- Kalibriert zu Beginn automatisch (kurz Warten)
- Kalibriert über Laufzeit
- Kann manuell kalibriert werden (CENTER)

Code und Logik

Code-Logik


Hub A - roboterarm.py
Bibliotheken & Initialisierung
Kalibrierung (Motoren)
Verbindung zu Client
Nachrichten empfangen
Motoren reagieren
Hub B - joystick.py
Bibliotheken & Initialisierung
Kalibrierung (Gyro)
Verbindung zu Server
Sensoren auslesen
Nachrichten versenden
Hauptschleife

Code für Roboterarm (Hub A) (1/6)

```
from pybricks.messaging import BluetoothMailboxServer, TextMailbox
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import Motor
from pybricks.parameters import Port, Stop

ev3 = EV3Brick()
server = BluetoothMailboxServer()
mbox = TextMailbox('greeting', server)

motor_platform = Motor(Port.A)
motor_arm_1 = Motor(Port.B)
motor_arm_2 = Motor(Port.C)
motor_gripper = Motor(Port.D)
```

Code für Roboterarm (Hub A) (2/6)

```
# Zustände der Sensoren
NEGATIVE = 0
NEUTRAL = 1
POSITIVE = 2

prev_x_state = NEUTRAL
prev_y_state = NEUTRAL
prev_button_state = NEGATIVE
```

Code für Roboterarm (Hub A) (3/6)

```
# Kalibrierung Motoren
ev3.screen.clear()
ev3.screen.print('Starte Kalibrierung...')
print('Kalibrierung...')
motor_platform.reset_angle(0)
motor_arm_1.reset_angle(0)
motor_arm_2.reset_angle(0)
motor_gripper.reset_angle(0)
```

Code für Roboterarm (Hub A) (4/6)

```
# Verbindung zum Client
print('Warte auf Verbindung...')
ev3.screen.clear()
ev3.screen.print('Warte auf Verbindung...')
server.wait_for_connection()
print('Verbunden!')
ev3.screen.clear()
ev3.screen.print('Verbunden!')
```

Code für Roboterarm (Hub A) (5/6)

```
# Funktion zum Steuern des Arms
def move_arm(state):
    speed_motor_1 = 200
    speed_motor_2 = -170
    if state == 'NEGATIVE':
        if motor_arm_1.angle() < 900:
            motor_arm_1.run_target(speed_motor_1, 900, then=Stop.HOLD, wait=False)
        elif motor_arm_1.angle() >= 900 and motor_arm_2.angle() > -700:
            motor_arm_1.run_target(speed_motor_1, 1640, then=Stop.HOLD, wait=False)
            motor_arm_2.run_target(speed_motor_2, -700, then=Stop.HOLD, wait=False)
        elif motor_arm_1.angle() >= 1640 or motor_arm_2.angle() <= -700:
            motor_arm_1.hold()
            motor_arm_2.hold()
```

Code für Roboterarm (Hub A) (6/6)

```
while True:
    mbox.wait()
    message = mbox.read().split('-')
    sensor = message[0]
    state = message[1]

   if sensor == 'X':
        move_platform(state)
    elif sensor == 'Y':
        move_arm(state)
    elif sensor == 'BUTTON':
        move_gripper(state)
```

Code für Joystick (Hub B) (1/5)

```
from pybricks.messaging import BluetoothMailboxClient, TextMailbox
from pybricks.hubs import EV3Brick
from pybricks.parameters import Port
from pybricks.ev3devices import TouchSensor, GyroSensor
from pybricks.tools import wait, StopWatch

ev3 = EV3Brick()
SERVER = 'EV3-Robotarm'
client = BluetoothMailboxClient()
mbox = TextMailbox('greeting', client)

gyro_x = GyroSensor(Port.S1)
gyro_y = GyroSensor(Port.S2)
button = TouchSensor(Port.S3)
```

Code für Joystick (Hub B)(2/5)

```
def calibrateGyro_X():
    while True:
        gyro_minimum_angle, gyro_maximum_angle = 10, -10
        gyro_x_sum = 0
        for _ in range(GYRO_CALIBRATION_LOOP_COUNT):
            gyro_sensor_value = gyro_x.angle()
            gyro_x_sum += gyro_sensor_value
            if gyro_sensor_value > gyro_maximum_angle:
                gyro_maximum_angle = gyro_sensor_value
            if gyro_sensor_value < gyro_minimum_angle:
                gyro_minimum_angle = gyro_sensor_value
            wait(1)
        if gyro_maximum_angle - gyro_minimum_angle < 0.5:
            break
    gyro_x.reset_angle(0)
    gyro_x_offset = gyro_x_sum / GYRO_CALIBRATION_LOOP_COUNT
```

Code für Joystick (Hub B)(3/5)

```
# Wechselt Zustand und sendet an Server
def switchState_and_send(sensor, new_state):
    global prev_x_state, prev_y_state, prev_button_state
    if sensor == "X":
        if new_state != prev_x_state:
            prev_x_state = new_state
            mbox.send("X-"+str(new_state))
            wait(500)
```

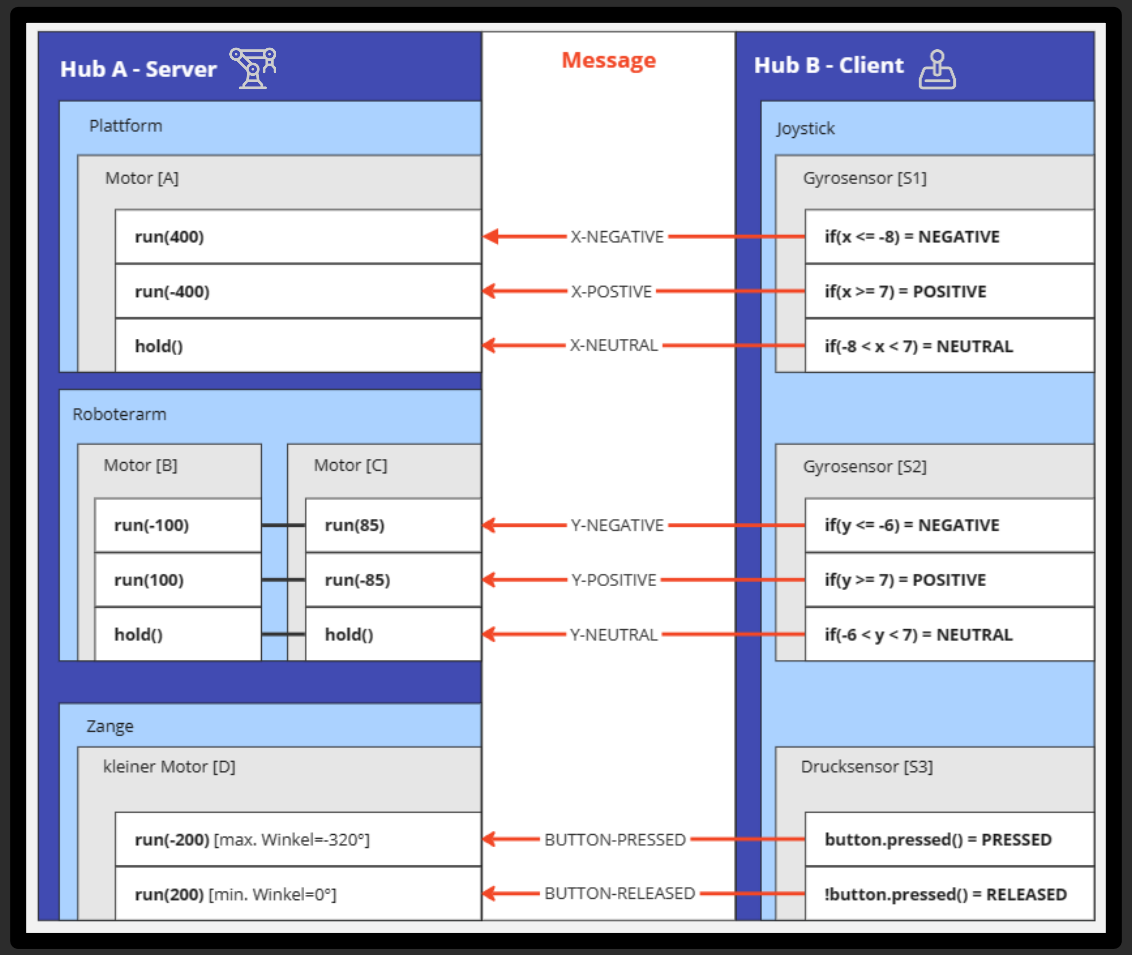
Code für Joystick (Hub B)(4/5)

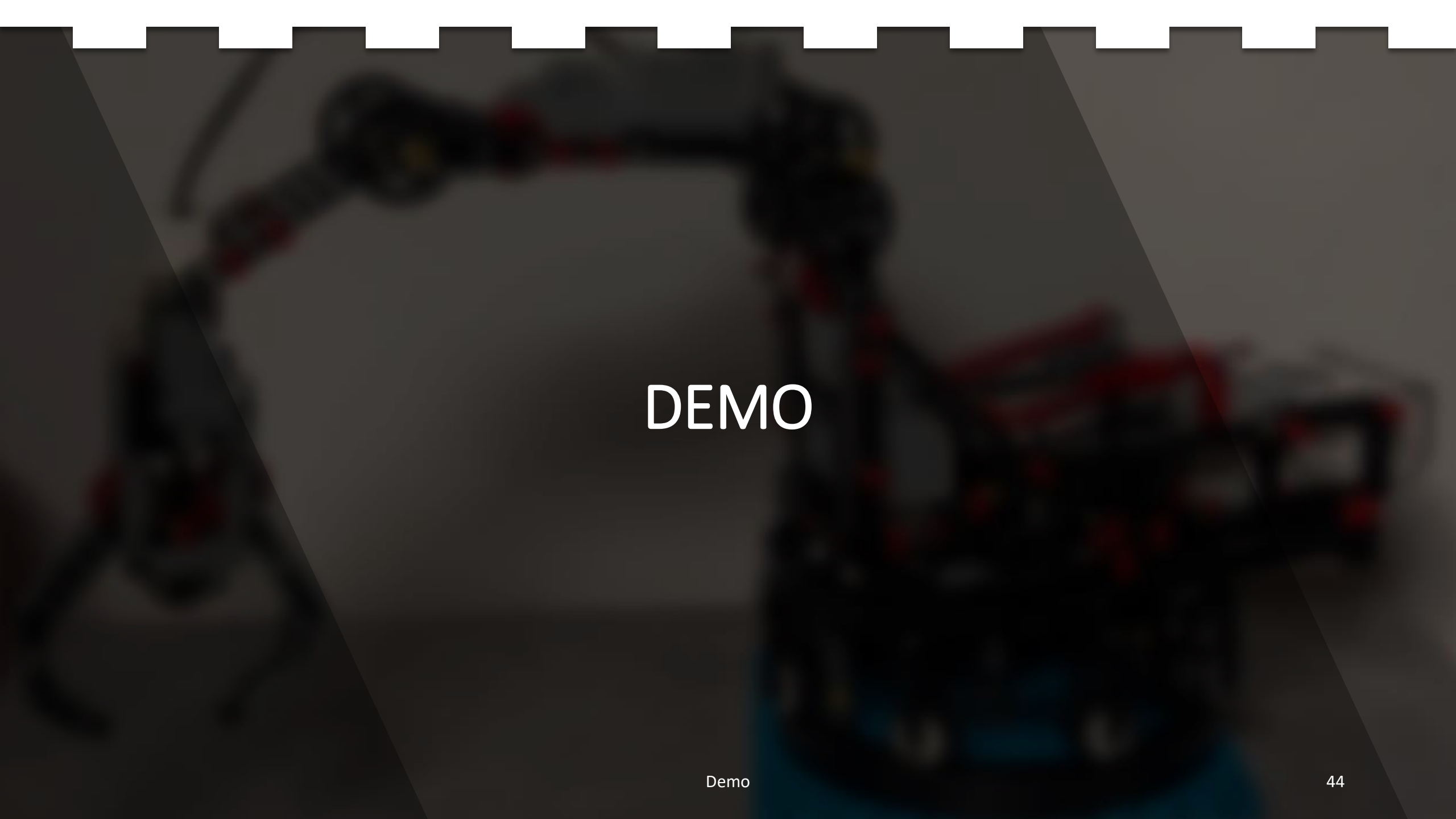
```
# Verbindungsaufbau mit Server
client = BluetoothMailboxClient()
mbox = TextMailbox('greeting', client)


print('Verbindung wird hergestellt...')
ev3.screen.clear()
ev3.screen.print('Verbindung wird hergestellt...')
client.connect(SERVER)
print('Verbunden!')
ev3.screen.clear()
ev3.screen.print('Verbunden!')
```

Code für Joystick (Hub B)(5/5)

```
while True:
    # Regelmäßige Neukalibrierung
     if calibration_timer.time() > CALIBRATION_INTERVAL:
           calibrateGyro_X()
    # Offsetkorrektur
    x_raw = gyro_x.angle() - gyro_x_offset
    # Exponentielle Glättung
    x_filtered = SMOOTHING_FACTOR * x_raw + (1 - SMOOTHING_FACTOR) * x_filtered
    # Re-Calibrate
    if x_filtered <= -10:
        gyro_y.reset_angle(-10)
    # Gyrosensorstatus überprüfen und entsprechende Aktionen ausführen
    if x_filtered <= -8 and prev_x_state != NEGATIVE:
        switchState_and_send("X", "NEGATIVE")
```

Kommunikation Server-Client





DEMO

Ausblick

Roboterarm

Grundstein für Roboterarm und Joystick ist gesetzt

Roboterarm

- Lässt sich in Länge erweitern
- Kraft lässt sich ggf. durch Kraft-Übersetzung erweitern
- Steuerung auch mit anderen Geräten möglich (z. B. Controller)

Joystick

- Vergrößerung des Modells für größere Winkelspannen (weniger Drift)
- 2 Gyro-X-Sensoren und 2 Gyro-Y-Sensoren (weniger Drift)