Hochschule Konstanz
Fakultät Informatik

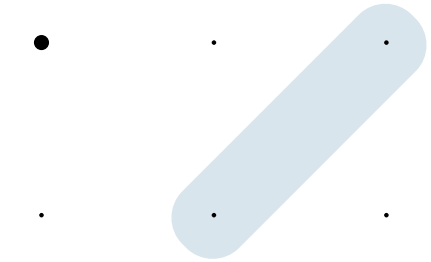
ROS-CoFFE
Continuous Feedback For Exercises with ROS

ROSCon DE 2024 Karlsruhe – Track: Bildung

Florian Drescher, Christopher Gogl
Fakultät Informatik
HTWG Konstanz

Agenda

1. Robotik-Vorlesungen an der HTWG
2. Zielsetzung
3. Umsetzung
4. Beispiel: Auszug aus Aufgabe 3 (Grundlagen der Robotik)
5. Geplante Weiterentwicklung
6. Veröffentlichung auf GitLab



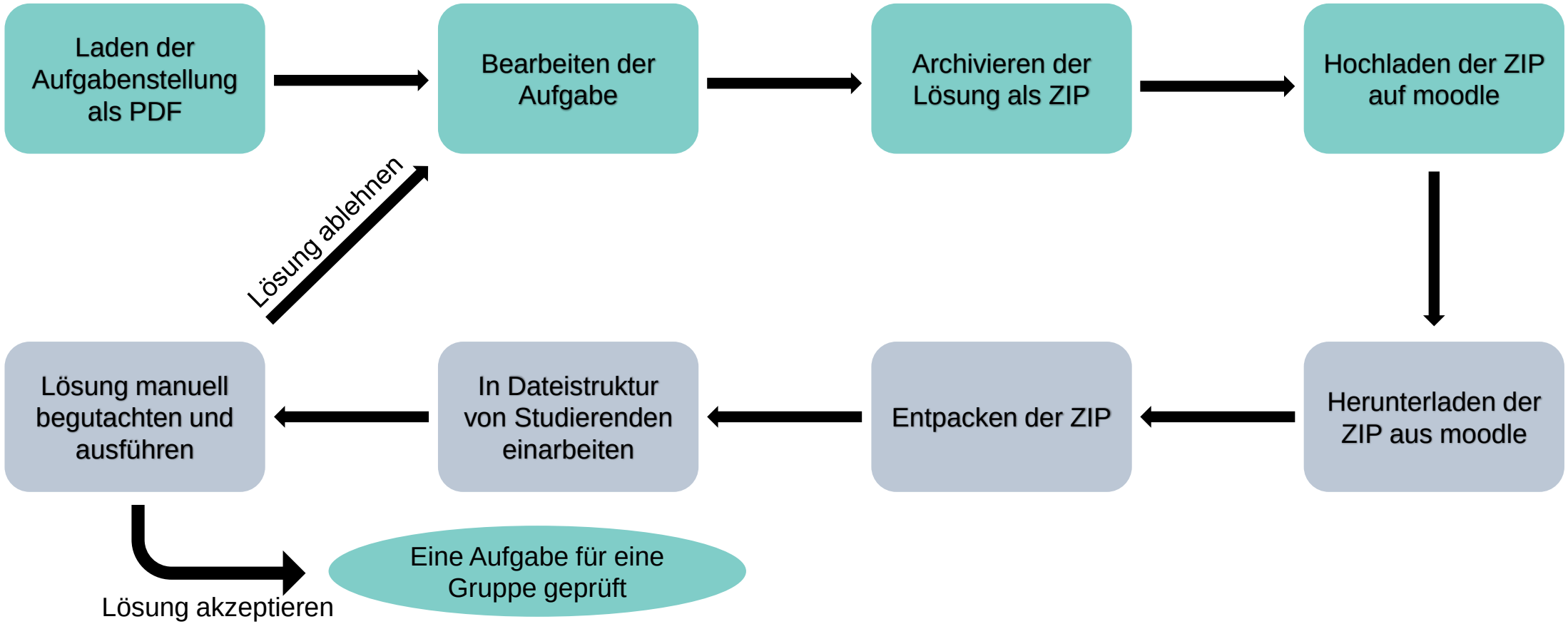
Robotik-Vorlesungen an der HTWG

Robotik-Vorlesungen an der HTWG

- Grundlagen der Robotik
  - Informatik Bachelor
  - Grundlagen Robotik
    - Koordinatensystem
    - Kinematik
    - Lokalisierung
    - Navigation
  - Grundlagen ROS
    - Eigene Implementierungen
    - Publisher und Subscriber
  - Übungsaufgaben zur Vertiefung

- Autonome Roboter
  - Informatik Master
  - Roboter
    - Sensordatenverarbeitung
    - Lokalisierung
    - Navigation
    - Kartierung
    - SLAM
  - Weiterführung ROS
    - bestehende Pakete anwenden
    - Publisher und Subscriber
    - Action Server und Client
  - Übungsaufgaben zur Vertiefung

Bisheriger Workflow



Bisheriger Workflow – Probleme

- Prozess ist sehr zeitaufwändig, da repetitiv
- Prozess bietet Sicherheitsrisiken
- Hoher Betreuungsaufwand
- Studierende müssen gegebenenfalls lange auf Rückmeldung warten
- Prozess verwendet kein ROS 2

Zielsetzung

Ziele

- Hauptziele:
  - Entwicklung neuer Aufgaben auf Basis von ROS 2
  - Schnelleres Feedback für Studierende
  - Weniger Korrekturaufwand für Dozenten

- Ziele für Umsetzung:
  - Studierende bearbeiten Aufgaben in Git-Repos
  - Abgaben sollen über Merge-Requests per GitLab eingereicht werden
  - CI/CD-Pipeline prüft Abgaben
  - Tests in CI-Pipeline dürfen nicht durch Studierende modifizierbar sein

Umsetzung

submission-tests

- Enthält Tests, die nicht von Studierenden modifiziert werden können
- Definiert Docker-Images die für Ausführung von Tests verwendet werden
  - Enthalten die submission tests
  - Images: ros-tester und ros-gazebo-tester

- ros-tester:
  - ros:humble erweitert um pip
  - Enthält submission tests für Aufgaben nur mit ROS
- ros-gazebo-tester:
  - ros:humble erweitert um pip und Gazebo classic
  - Behebt Fehler: „[spawn_entity]: Spawn service failed. Exiting.”
  - Enthält submission tests für Aufgaben mit ROS und Gazebo

exercises-template

- Enthält Aufgaben in Form von READMEs

- In Aufgabenverzeichnissen befinden sich sogenannten local tests
  - Rudimentäre Tests
  - Können von Studierenden erweitert werden
  - Werden auch in Pipeline ausgeführt

- CI-Config: Inkludiert Konfiguration aus Repo „submission-pipeline-config“

- Wird per Helper-Script an Studierenden-Repos ausgerollt

submission-pipeline-config

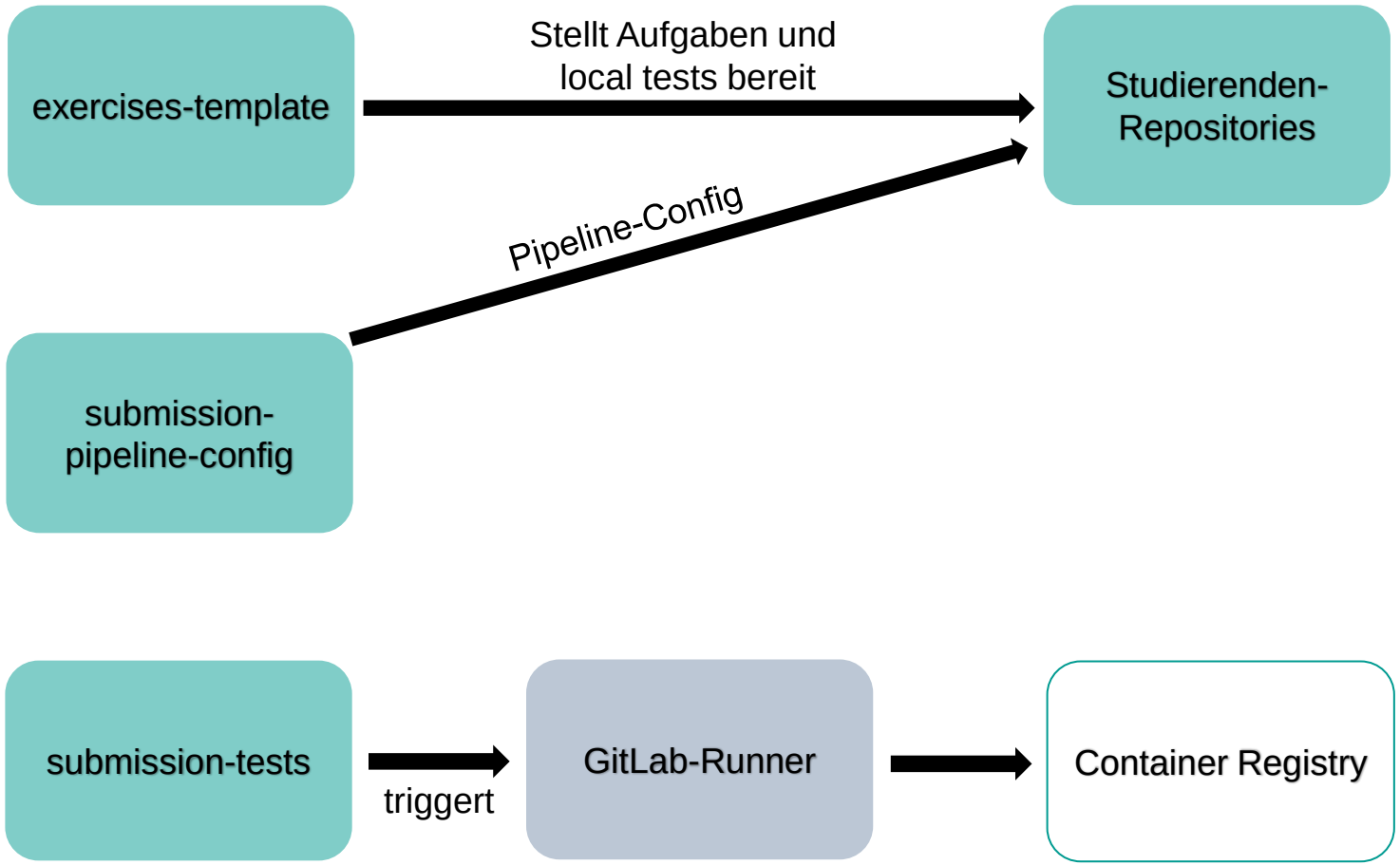
- Konfiguration der Pipeline
- Gesondertes Repository, damit nicht durch Studierende modifizierbar
- Config kann unabhängig von Studierenden-Repositories bearbeitet werden

- Funktionsweise der Pipeline:
  - Stage 1: build
    - Workspace anlegen
    - submission tests in Workspace kopieren
    - Ergebnisse und local tests von Studierenden in Workspace kopieren
    - colcon build
  - Stage 2: test
    - Ausführen von tests für jede Teilaufgabe mit pytest
  - Stage 3: .post
    - Pushen der Test-Ergebnisse in das Repo coffe-results

Architektur

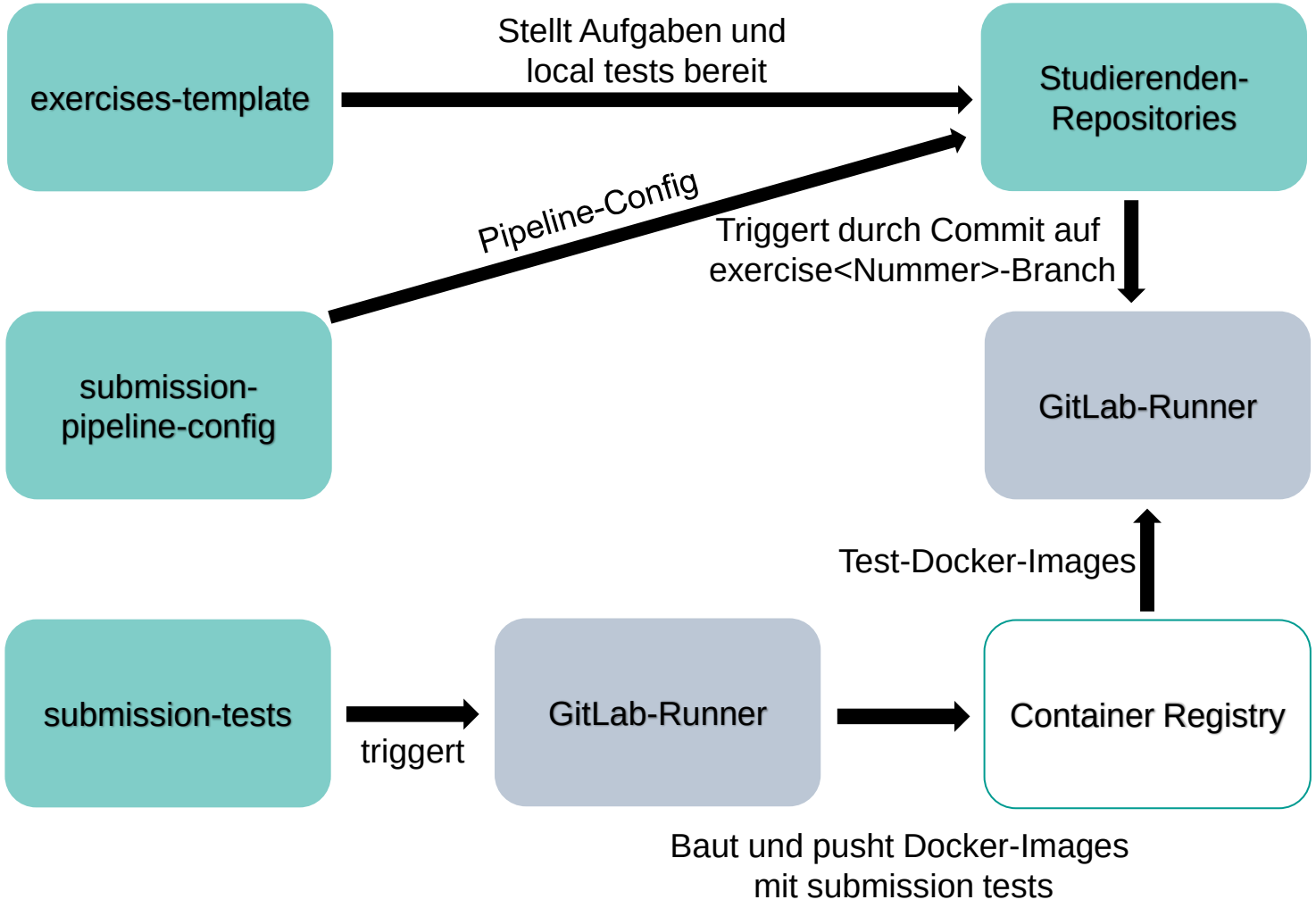
Baut und pusht Docker-Images
mit submission tests

Architektur

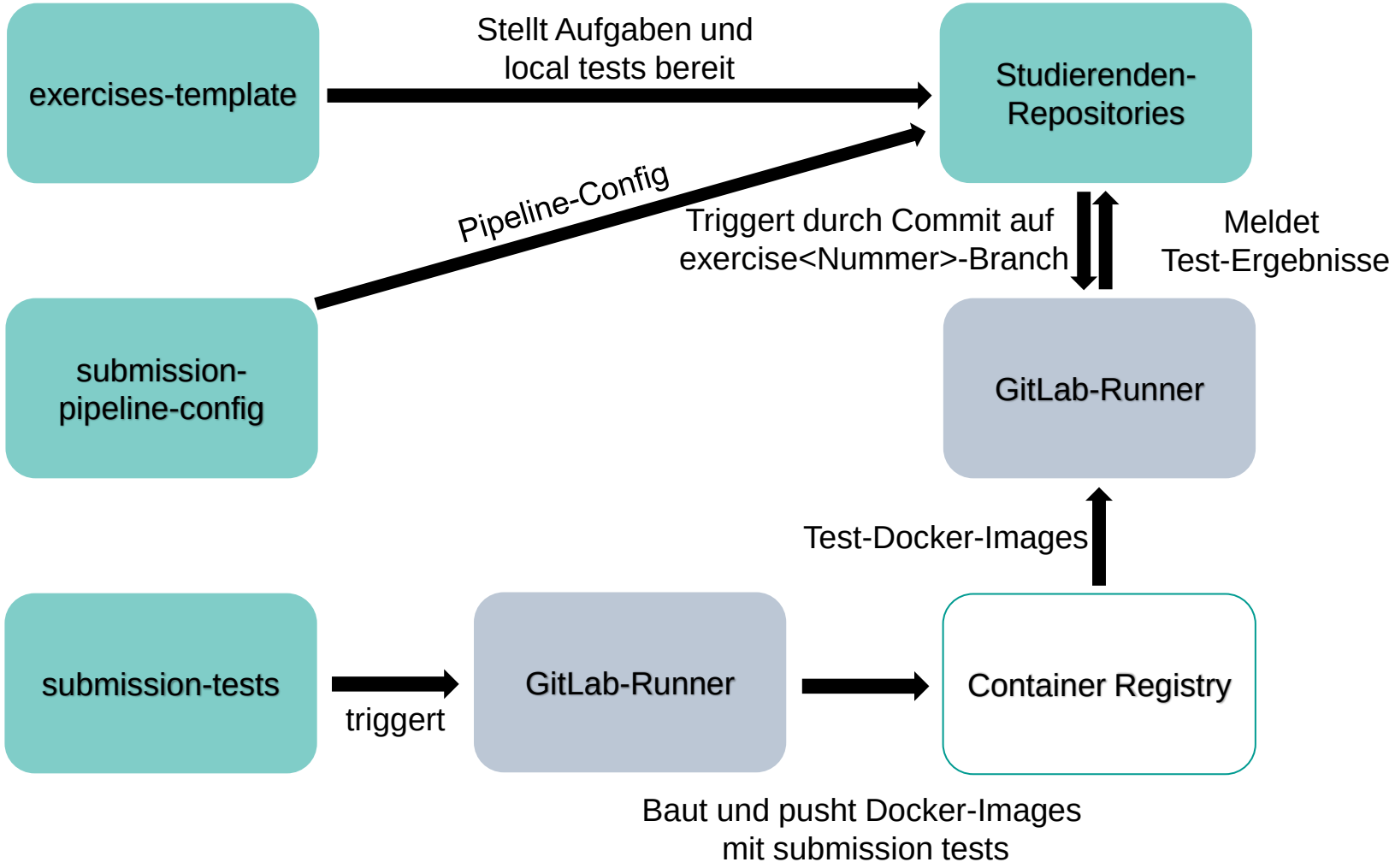


Baut und pusht Docker-Images
mit submission tests

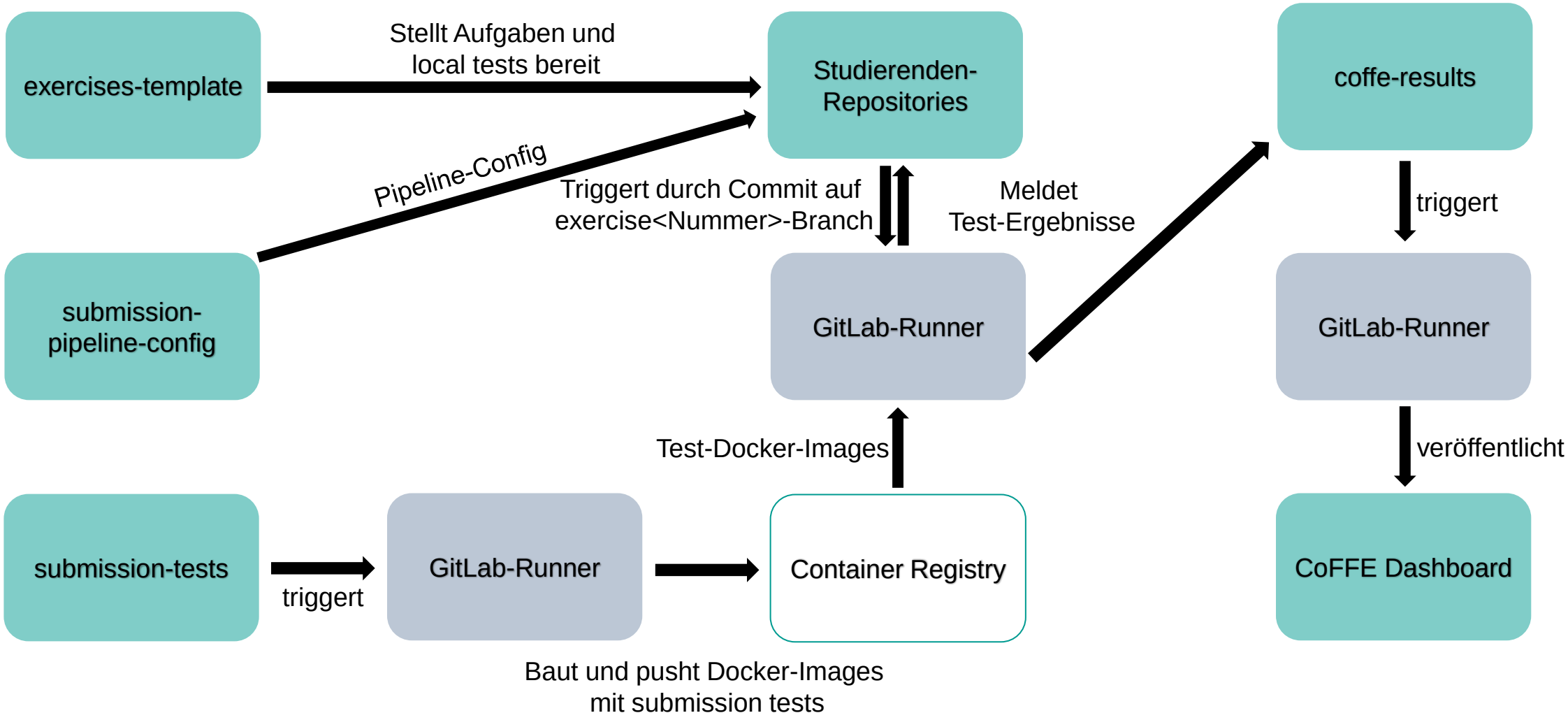
Architektur



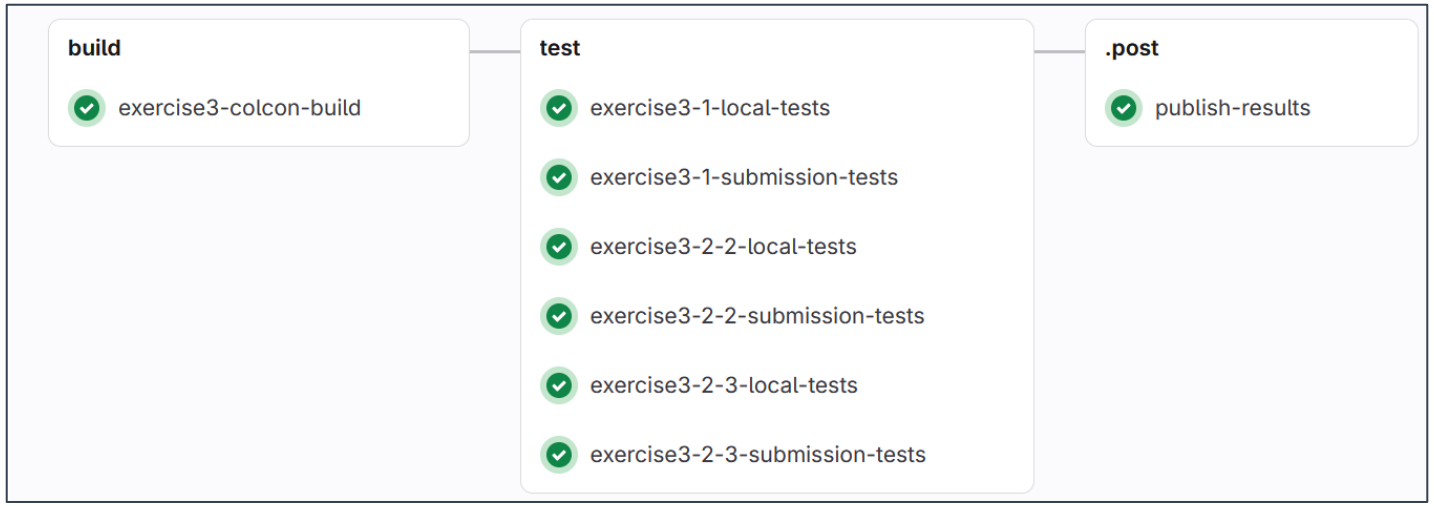
Architektur



Architektur



Ergebnisse im Studierenden-Repository

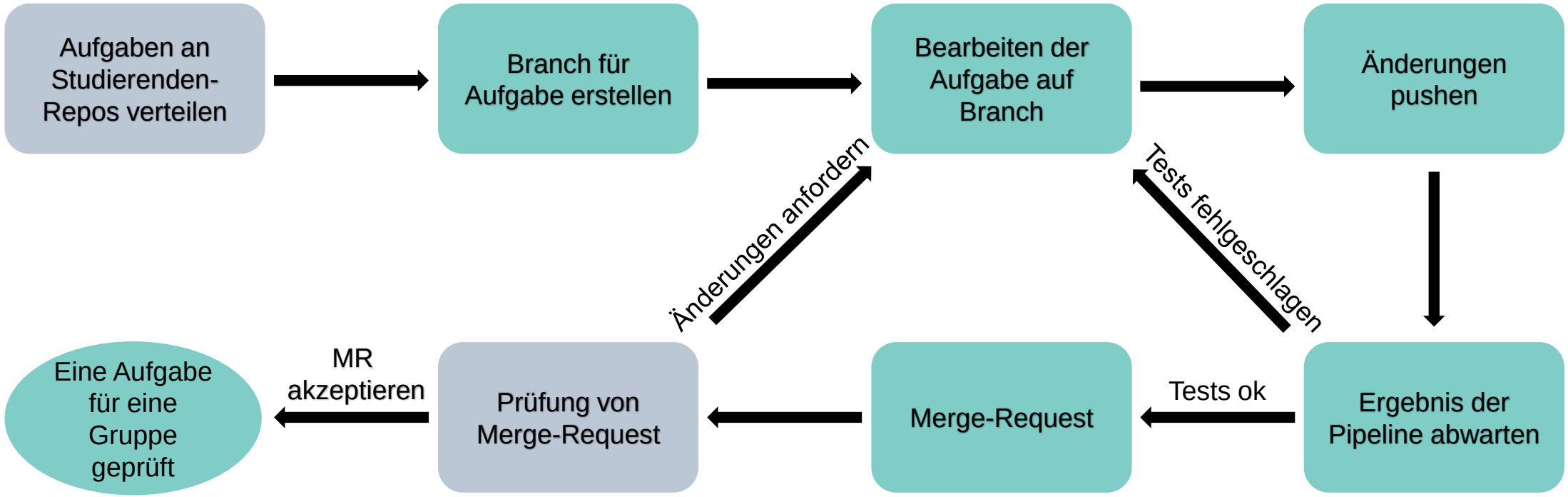


coffe-results / CoFFE Dashboard für Dozenten

CoFFE Dashboard exportieren

Team	exercise1	exercise2	exercise3	exercise4	exercise5
Team01	20/20 3702/3702	18/18 1356/1356	3/3 3/3	5/5 5/5	0/0 0/1
Team02	20/20 3702/3702	18/18 1356/1356	3/3 3/3	5/5 5/5	0/0 0/1
Team03	20/20 3702/3702	18/18 1356/1356	3/3 3/3	5/5 5/5	0/0 1/1
Team04	20/20 3702/3702	18/18 1356/1356	2/3 2/3	5/5 5/5	0/0 1/1
Team05	20/20 3702/3702	18/18 1356/1356	3/3 3/3	5/5 5/5	0/0 0/1
Team06	20/20 3702/3702	18/18 1356/1356	3/3 3/3	5/5 5/5	

Neuer Workflow



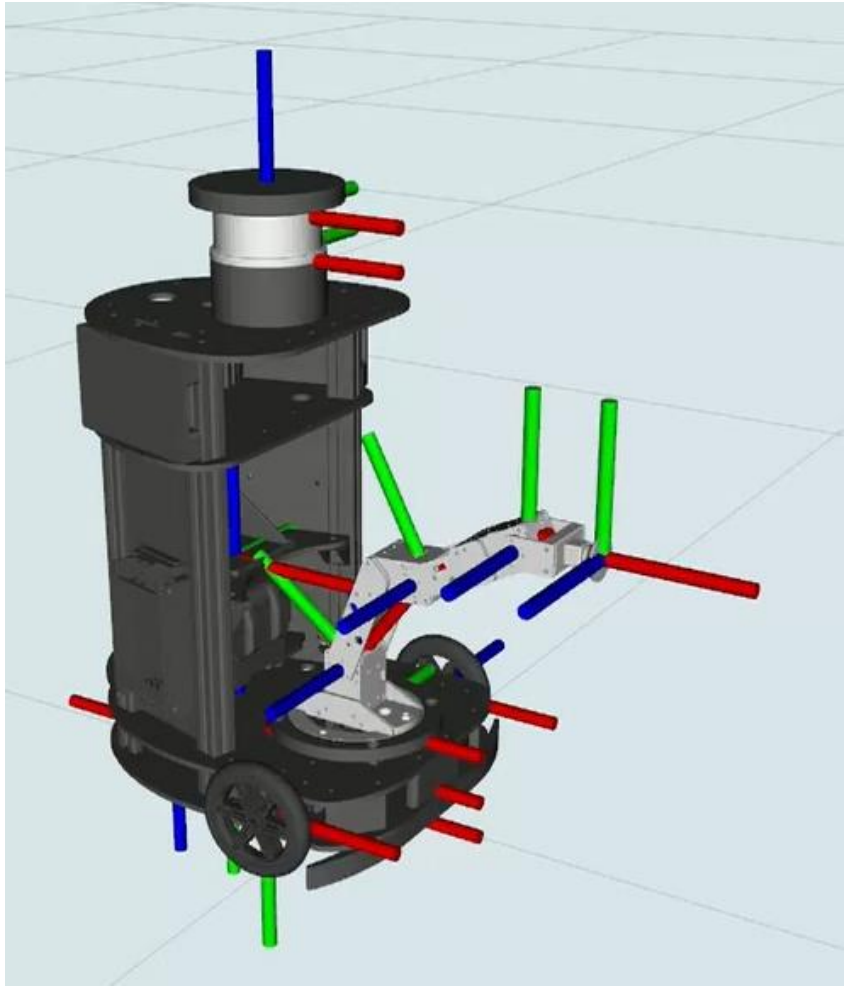
Vorteile

- Übungen verwenden ROS 2
- Reduzierung des Betreuungs- und Zeitaufwands für Dozenten
- Schnelle Feedback-Iterationen für Studierende
- Einfache Erweiterung der Tests
- Schnelle Integration neuer Aufgaben
- In Übungsstunden liegt Fokus mehr auf fachlichen Fragen

Beispiel: Auszug aus Aufgabe 3 (Grundlagen der Robotik)

Aufgabe 3 – Übersicht

- Erstellen von Publisher- und Subscriber-Node
- Ansteuerung eines Gelenkarms
  - Visualisierung von JK_Rollin in Rviz
  - Inverse-Kinematik-Node
  - Kreisbewegungs-Node



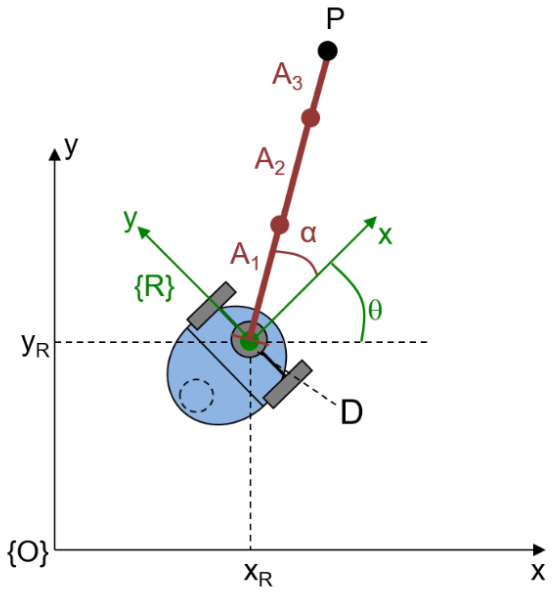
Aufgabe 3 – README.md (Auszug)

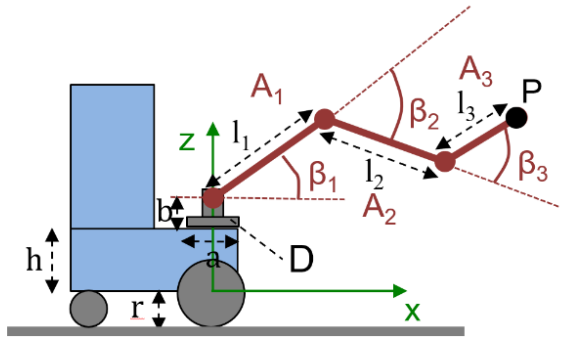
3.2 ROS2 Ansteuerung eines Gelenkarms

Erstellen Sie ein neues Package `my_jk_rollin_kinematics` zur Ansteuerung des Gelenkarms des Roboters JK_Rollin. Erweitern Sie hierfür ihr kinematisches Modell aus Exercise 2 um einen dritten Arm A_3 wie in Übung 2.5. Es müssen also die Gelenkwinkel für `turntable` (α), `shoulder` (β_1), `elbow` (β_2) und `wrist` (β_3) berechnet werden.

Die Abmessungen des Roboters und seine Konfiguration haben sich geändert. Beachten Sie, die geänderten Werte in der Tabelle und den Grafiken und dass der Drehteller nun in der z-Achse des Roboter-KS liegt.

x_R	y_R	θ	l	h	r	a	b	α	l_1	β_1	l_2	β_2	l_3	β_3
2	1	30°	0.0	0.021	0.026	0.12	0.047	40°	0.093	40°	0.082	-60°	0.051	50°





Seitenansicht mit α = 0°

Test-Struktur – Setup

```python
def setUp(self):
    # Create a ROS node for tests
    self.node = rclpy.create_node('test_jk_rollin_draw_circle_node')

    # Generate x, y and z coordinates of points that are expected to be published
    center_r = (0.18, 0, 0.14)
    radius = 0.04
    self.positions_in_circle = 100
    angles = np.linspace(0, 2 * np.pi, self.positions_in_circle)
    self.expected_x = np.zeros(len(angles)) + center_r[0]
    self.expected_y = np.sin(angles) * radius + center_r[1]
    self.expected_z = np.cos(angles) * radius + center_r[2]
```

```python
def test_jk_rollin_draw_circle(self, launch_service, proc_output):
    pose_msgs_rec_buffer: list[PoseStamped] = []

    sub_pose = self.node.create_subscription(
        PoseStamped,
        'tool_pose',
        lambda msg: pose_msgs_rec_buffer.append(msg),
        10
    )
```

Test-Struktur – Datenaufzeichnung

```python
# Messages should be published with 10hz.
# For 100 messages 10s are enough, add some extra time
end_time = time.time() + 30
while time.time() < end_time:
    rclpy.spin_once(self.node, timeout_sec=5)
    # Stop data collection after 100 msgs
    if len(pose_msgs_rec_buffer) == self.positions_in_circle:
        break

# Check if min 100 messages were received
self.assertGreaterEqual(
    len(pose_msgs_rec_buffer), self.positions_in_circle,
    "Not enough messages received. Please make sure that " +
    "messages are published to the topic 'tool_pose' " +
    "with a frequency of 10 Hz by the node 'jk_rollin_draw_circle'.")
```

Test-Struktur – Datenvorverarbeitung

```python
# Use first 100 received messages
pose_msgs_rec_buffer = pose_msgs_rec_buffer[:self.positions_in_circle]

# We may have missed some of the first messages, rotate received messages if necessary
rounded_y = np.round(self.expected_y, 3)
first_received_y = np.round(
    pose_msgs_rec_buffer[0].pose.position.y, 3)
y_indices = np.where(rounded_y == first_received_y)[0]

    ▪
    ▪
    ▪

logging.getLogger().info(
    "Rotating received messages by " + str(matching_offsets[0]))
logging.getLogger().info(
    "First Item before rotating " + str(pose_msgs_rec_buffer[0]))
pose_msgs_rec_buffer = np.roll(pose_msgs_rec_buffer, matching_offsets[0])
logging.getLogger().info(
    "First Item after rotating " + str(pose_msgs_rec_buffer[0]))
```

Test-Struktur – Auswertung

```
# Check for correct points
for i in range(self.positions_in_circle):
    logging.getLogger().info("Compare Point:" + str(i))
    logging.getLogger().info(
        "Receive: " + str(pose_msgs_rec_buffer[i].pose.position.x) + " - " + str(
            pose_msgs_rec_buffer[i].pose.position.y) + " - " +
        str(pose_msgs_rec_buffer[i].pose.position.z))
    logging.getLogger().info(
        "Expected: " + str(self.expected_x[i]) + " - " + str(self.expected_y[i])
        + " - " + str(self.expected_z[i]))
    self.assertAlmostEqual(
        pose_msgs_rec_buffer[i].pose.position.x, self.expected_x[i], 3,
        "position.x did not match expected x")
    self.assertAlmostEqual(
        pose_msgs_rec_buffer[i].pose.position.y, self.expected_y[i], 3,
        "position.y did not match expected y")
    self.assertAlmostEqual(
        pose_msgs_rec_buffer[i].pose.position.z, self.expected_z[i], 3,
        "position.z did not match expected z")
```

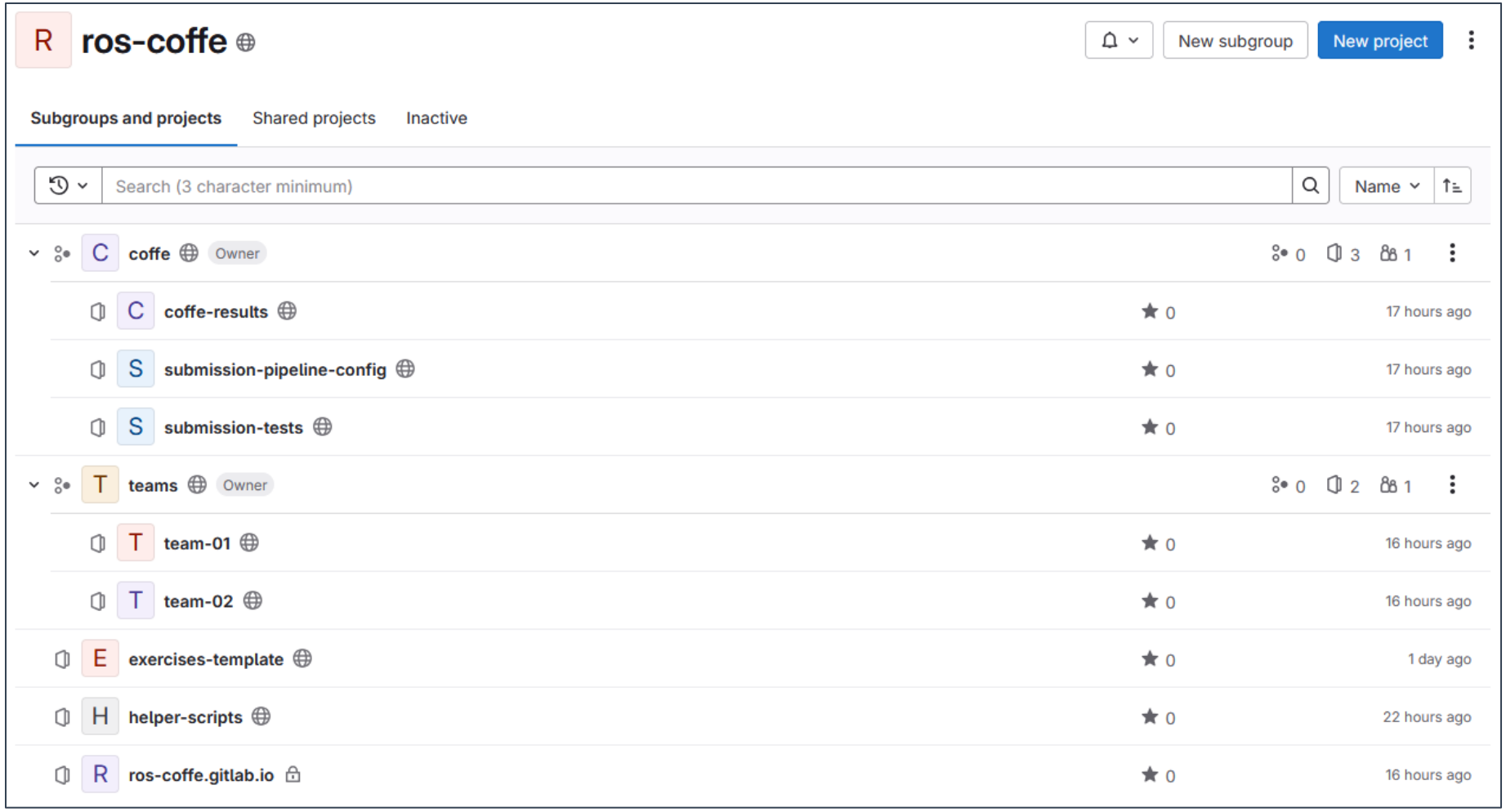
Geplante Weiterentwicklung

Geplante Weiterentwicklung

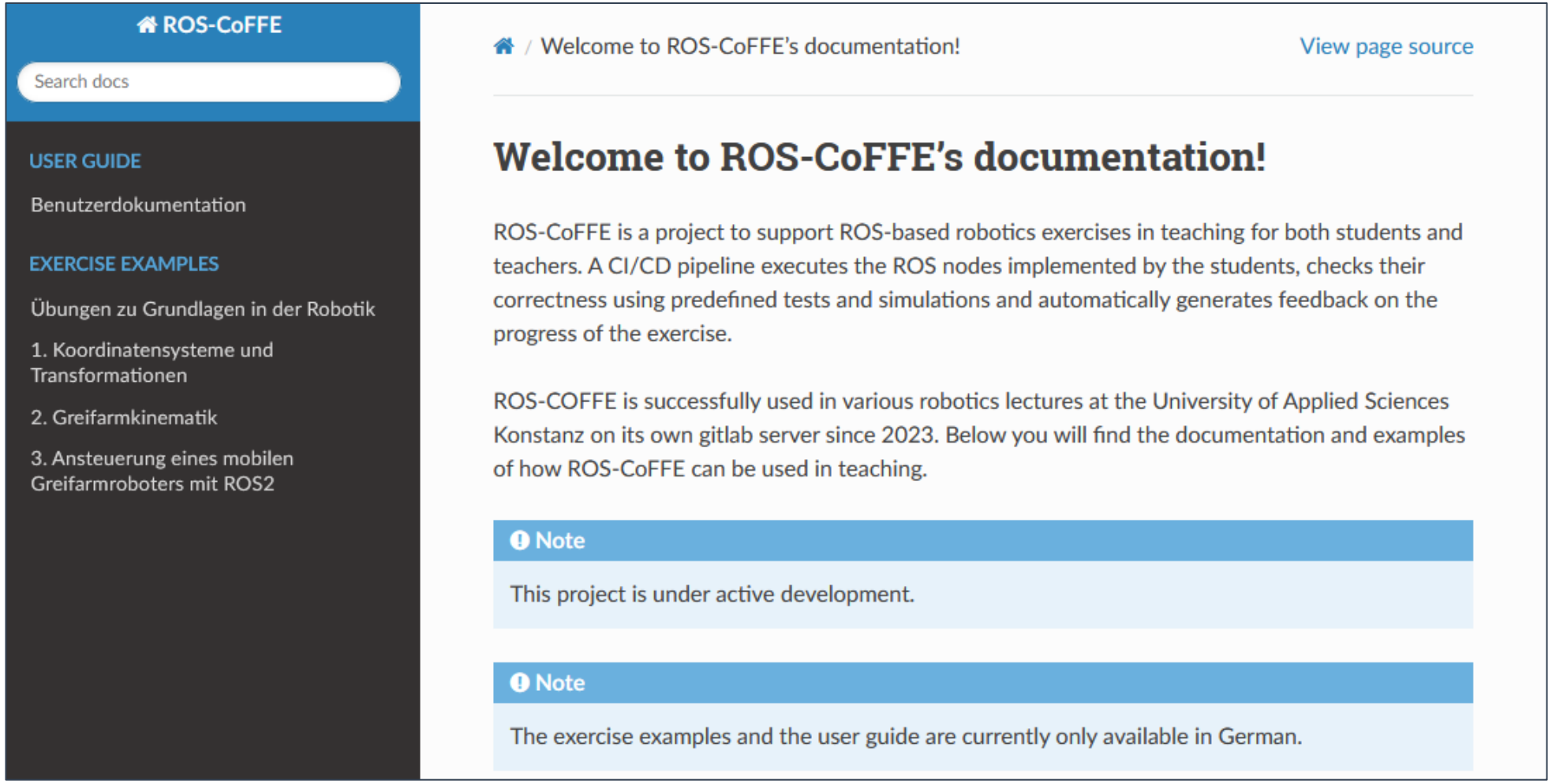
- Reale Roboter JK_Rollin und turtlebot wieder einsatzfähig bekommen
- Upgrade auf Ubuntu 24.04 und ROS2 Jazzy
- Upgrade auf Gazebo-Sim / Gazebo Ignition
- Weitere Verbesserung / Verfeinerung der Aufgabenstellungen und Tests
- Umstellung von Markdown auf RST und Read the Docs
- Eventuell Verwendung von Isaac Sim

Veröffentlichung auf GitLab

gitlab.com/ros-coffe



ros-coffe.gitlab.io



Hochschule Konstanz
Fakultät Informatik

Vielen Dank für Ihre Aufmerksamkeit!