# Project Presentation

Dennis Huff, Ari Wahl, Philipp Bodemann

## 6 Axes Roboter
## on Helix Trajectory

# Gliederung

1. Aufgabenstellung/Problemstellung
2. Lösungsansatz
3. Environments
    - 3.1. Aufbau des Environments
    - 3.2. Kinematik des Roboters
4. Lernalgorithmus
5. Ergebnisse
    - 5.1. Testszenarien
    - 5.2. Visualisierung der Lernergebnisse
6. Herausforderungen

# Das Problem

- Der Agent soll ein 3D trajectory (Helix) abfahren (10 cm Höhe, 3 cm Radius)
- Roboterarm mit 6 Gelenken begrenzt auf -180° bis 180°
- pro Schritt: 6 Gelenke unabhängig jeweils 0.1°, 0° oder -0.1° Bewegung
- 729  (3^6) mögliche Actions
- Die Orientierung des Tool Center Points (TCP) soll gleich bleiben

# Unser Lösungsvorschlag

→ deep q-learning network (CNN + insert)

→ primary & target network

→ 3-step-Bootstrapping mit memory replay

→ epsilon decay

→ gymnasium Environment

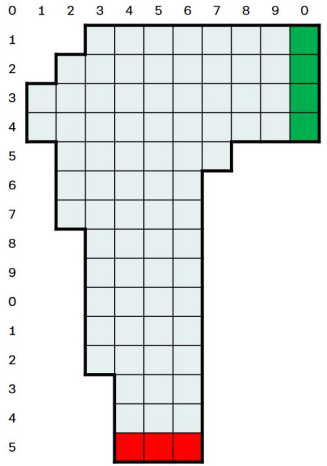
→ extra translation matrix

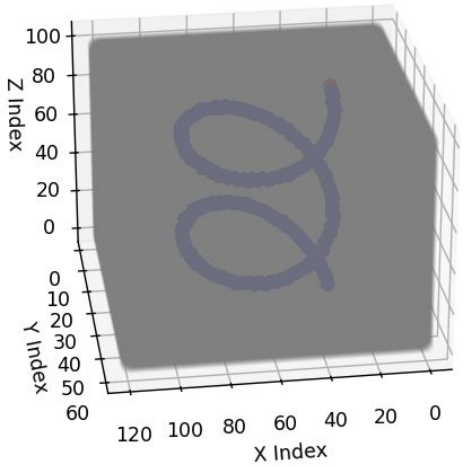
→ reward Funktion

# Das Environment

Idee: Helix im Voxel Space als “Racing Track”

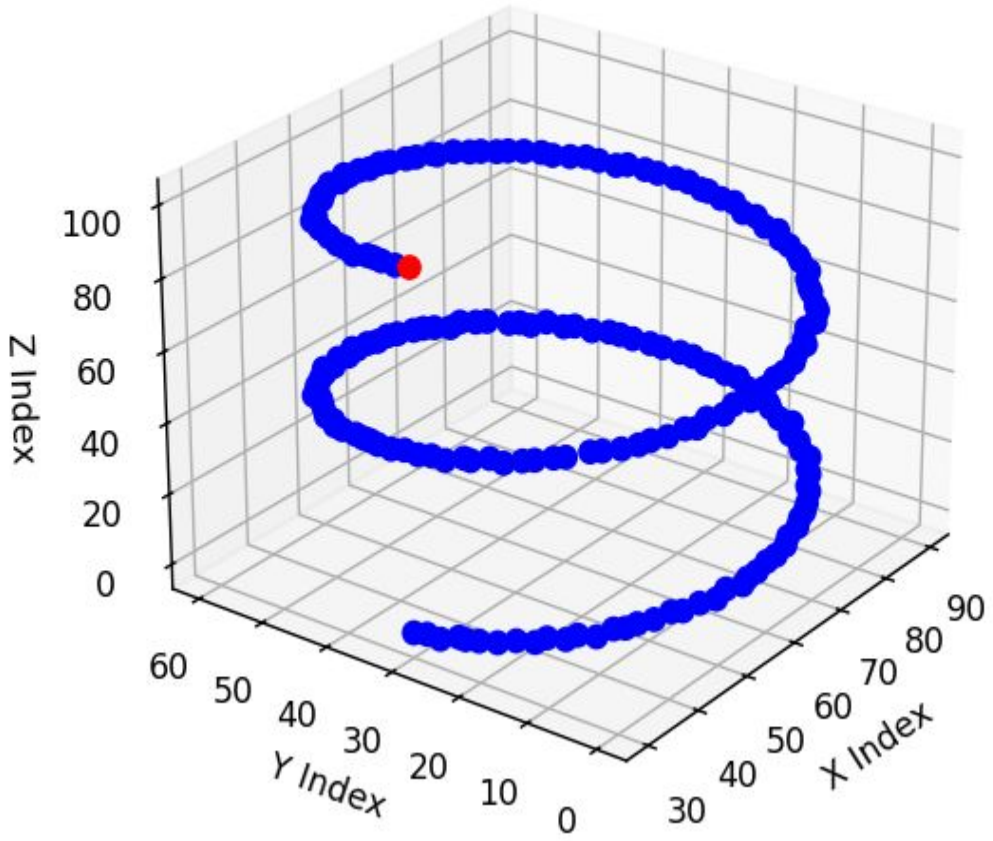
1. Initialisierung Voxel Space → ganzer Voxel Space ist in 1mm^3 Voxels unterteilt
2. Initialisierung der Helix Trajektorie im Voxel Space → Radius 3 cm, Drehungen = 2
3. Voxel Codierung: Helix Voxel = 0, target Voxel = 1, restliche Voxel = -1



# Environment: Helix

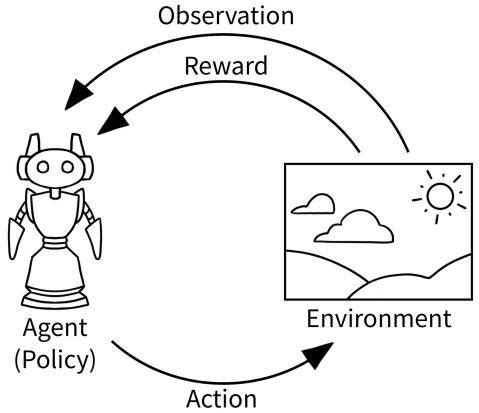
$$\vec{x}(t) = \begin{pmatrix} r \cdot \cos(2\pi\, t) \\ r \cdot \sin(2\pi\, t) \\ h \cdot t + c \end{pmatrix}$$



# Unser Lösungsvorschlag

**gymnasium Environment:**

- stellt API für single agent reinforcement learning environments bereit
- liefert 4 Funktionen: make, reset, step, render
- Env-Klasse → stellt Markov-Entscheidungsprozess (MDP) aus der Reinforcement-Learning-Theorie dar.



# Unser Lösungsvorschlag

**gymnasium Environment:**

reset():

- Rücksetzen der Umgebung
  - reward = Null
- Festlegen der Anfangsbedingungen
  - Anfangswinkel der Gelenke [0°, 0°, 0°, 0°, 0°, 0°]
  - Orientierung von TCP (mit forward kinematics)
  - Position von TCP (mit forward kinematics)
- Rückgabe: Anfangszustand, Infos

# Unser Lösungsvorschlag

**gymnasium Environment:**

step():

- aktualisiert die Umgebung → basierend auf den Aktionen des Agenten
- Berechnung der delta-Winkel → neue Gelenkwinkel
- Vorwärtskinematik → TCP Position und Orientierung
- generiert eine TCP-Beobachtung für den nächsten Schritt
- Belohnung wird berechnet
- Rückgabe:

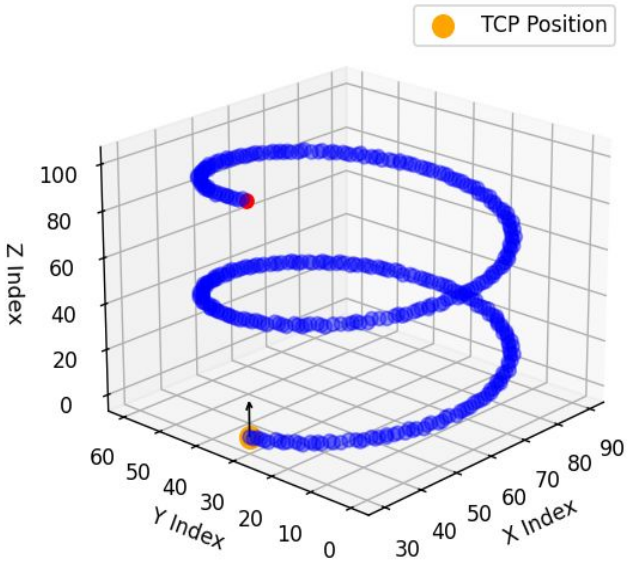
  Zustand, Belohnung, Episode beendet (terminated or truncated), Infos

# Unser Lösungsvorschlag

**gymnasium Environment:**

render():

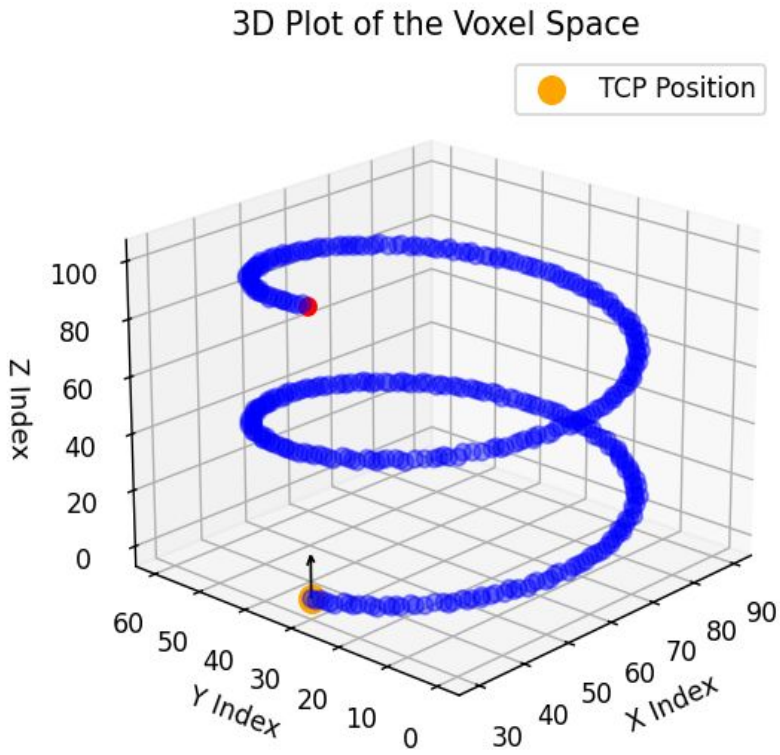
- visualisiert die Helix, aktuelle TCP Position sowie die Orientierung
  - rot = Zielpunkt
  - blau = Voxel der Helix
  - orange = aktuelle Position
  - Pfeil = Orientierung
- Speichern der Figures → GIF



# Unser Lösungsvorschlag

**Reward-Kriterien:**

- TCP auf Helix und Erhöhung in z-Richtung:
  - 25  Punkte
- TCP Abstieg auf z-Koordinate:
  - - 25 Punkte
- Erreichen des Ziels:
  - 1000 Punkte
- Verlassen der Helix:
  - -1 Punkt
  - 0 Punkte (TOL = 2 mm)
- Verlassen des Voxel-Raums:
  - -10 Punkte
- Orientierungs Abweichung und Anstieg auf z:
  - 0 Punkte (Überschreiten der TOL)
  - 5 Punkte (TOL = 10°)



# Kinematic

Devanit-Hartenberg-Matrix:

- Beschreibung kinematischer Kette ein Roboterarms
- Bestimmung der TCP Position und Orientierung

DH-Parameter:

- a [m] (Link length)
- d [m] (Link offset)
- alpha [Rad] (Link Twist)
- theta [Rad] (Joint Angle)

$$
{}^{n-1}T_n = \left[\begin{array}{cccc}
\cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & r_n\cos\theta_n \\
\sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & r_n\sin\theta_n \\
0 & \sin\alpha_n & \cos\alpha_n & d_n \\
\hline
0 & 0 & 0 & 1
\end{array}\right]
$$

# Kinematics

im Projekt verwenden wir einen Roboterarm mit 6-Achsen

um TCP Pos zu bestimmen (Gelenk 1- 6)

→ T = T1*T2*T3*T4*T5*T6

→ T_i entspricht der DH-Transformation zwischen dem i-ten und i+1-ten Glied

$$
{}^{n-1}T_n = \left[\begin{array}{cccc}
\cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & r_n\cos\theta_n \\
\sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & r_n\sin\theta_n \\
0 & \sin\alpha_n & \cos\alpha_n & d_n \\
\hline
0 & 0 & 0 & 1
\end{array}\right] = \left[\begin{array}{ccc|c}
 & R & & T \\
\hline
0 & 0 & 0 & 1
\end{array}\right]
$$

$$
R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}
$$

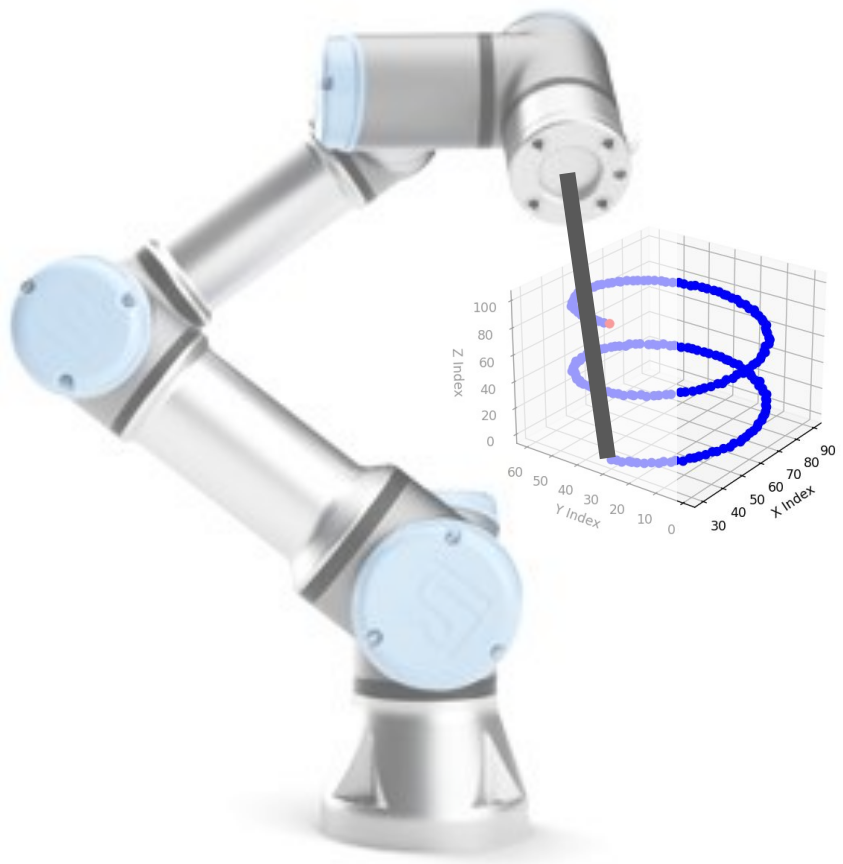
$$
\beta = \operatorname{atan2}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right)
$$

$$
\alpha = \operatorname{atan2}(r_{21}/\cos(\beta), r_{11}/\cos(\beta))
$$

$$
\gamma = \operatorname{atan2}(r_{32}/\cos(\beta), r_{33}/\cos(\beta))
$$

# Helix Anfang im Voxel Space auf Roboter TCP Koordinaten für [0°,0°,0°,0°,0°,0°]

eine Translation mehr…



# Bestandteile unseres Lernalgorithmus

- Deep Q Learning Netzwerk
- Replay Memory
- Agent
- Epsilon Decay
- n-step Bootstrapping
- Reward Funktion
- Hyperparameter Rastersuche

# Lernalgorithmus

1) Deep Q Learning
   - Input: states (from memory)
   - Output: Q(s,a)
   - Primary & Target Netzwerk

# Lernalgorithmus

1) Deep Q-Learning: getestete Netzwerkarchitekturen

- Fully Connected (simpelster Ansatz)

- CNN (kann 3D Daten verarbeiten)

- Multimodale Feature Fusion:  CNN für 3D Daten + Orientierung nur in FC

# Lernalgorithmus: Deep Q-Learning

## Convolutional neural network (CNN)

Was ist CNN?

- Ein CNN ist eine spezielle Art von künstlichen neuronalen Netzwerk, das besonders effektiv bei der Verarbeitung von Daten mit räumlicher Struktur ist

Allgemeine Struktur:

- Konvolutionale Schichten (Convolutional layers)
- Pooling layers
- Vollständig verbundene Schichten (FC → Fully connected Layer)

# Lernalgorithmus

2) Replay-Memory

**- Speichern von Erfahrungen:** Datenbank für Erfahrungen in Form von Tupeln (state, action, reward, next state, done)

**- Zeitliche Korrelationen unterbrechen:** Random Samples in vorgegebener batch_size zum Training des Netzwerks

**- Effiziente Nutzung von Erfahrungen:** Jede Erfahrung kann öfter zum Training verwendet werden

3) Agent

- Ziel: an optimale Q-Funktion annähern (mithilfe des Q-Netzwerks)
- Wahl der Aktion mit dem höchsten geschätzten Q-Value
- Updating des Q-Netzwerks (Feedback basiert → Rewards aus dem Environment)
- exploration and exploitation Balance

4) Epsilon Decay

- **epsilon*=epsilon_decay**
- **epsilon_min**

# Lernalgorithmus

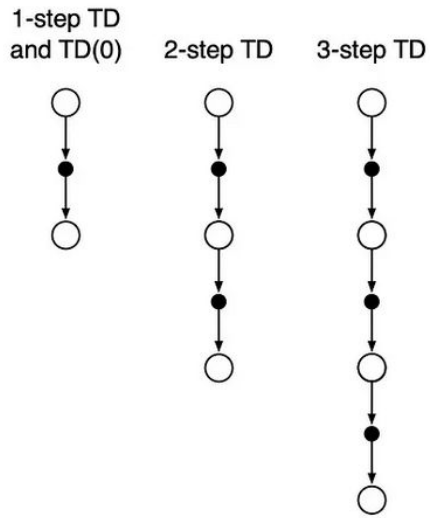
5) n-step Bootstrapping:

- Methode zur Abschätzung von Zustandswerten in RL
- kombiniert die Belohnungen und Zustandswerte über mehrere aufeinanderfolgende Zeitschritte hinweg
- n-Step-Reward wird berechnet mit:
  - γ → Discountfaktor
  - r → reward in each step (t+n)

$$R_{t:t+n} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-1} r_{t+n-1}$$

# Lernalgorithmus: 3-step Bootstrapping

- Übergabeparameter: n=3
- 3-Step-Bootstrapping ist eine spezifische Variante des n-Step-Bootstrappings
- Wertschätzung eines Zustands → Kombination von Belohnungen und Zustandswerte über drei aufeinanderfolgende Schritte
- Berechnung 3-Step-Rückkehr:



# Lernalgorithmus: n-step Bootstrapping

Updating der Q-Values für Deep Q-Learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ R_{t:t+n} + \gamma^n \max_a Q(s_{t+n}, a) - Q(s_t, a_t) \right]$$

alpha = learning rate
R_t:t+n = kumulativer n-step Reward
s_t+n = next n states
s_t = current state
a_t = current action

# Lernalgorithmus

## 7) Reward Funktion

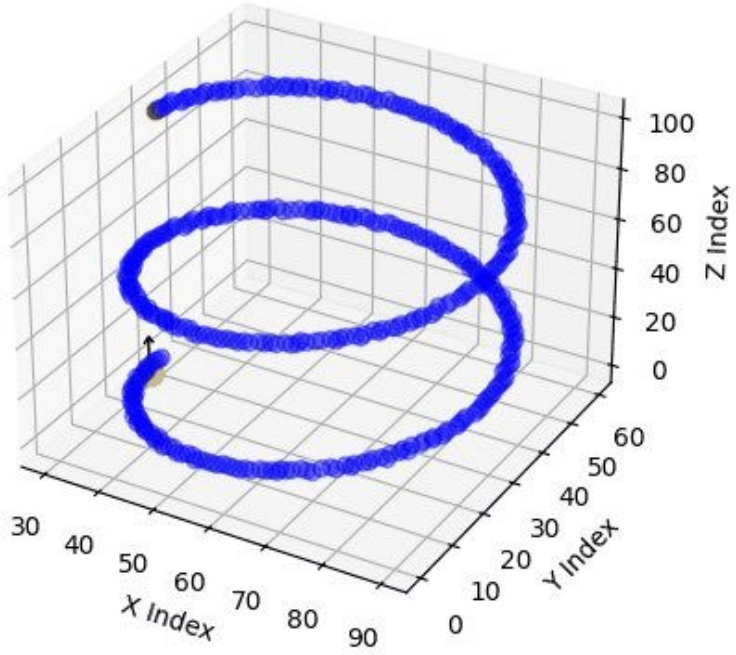
bereits im Environment vorgestellt, wird hier nur Aufgerufen über step()

## 8) Hyperparameter Rastersuche

- um gute (Hyper)Parameter zu Finden, die das Lernen optimieren
- Hyperparameter: Batchsize, Episodenanzahl, Epsilon Decay, Epsilon min

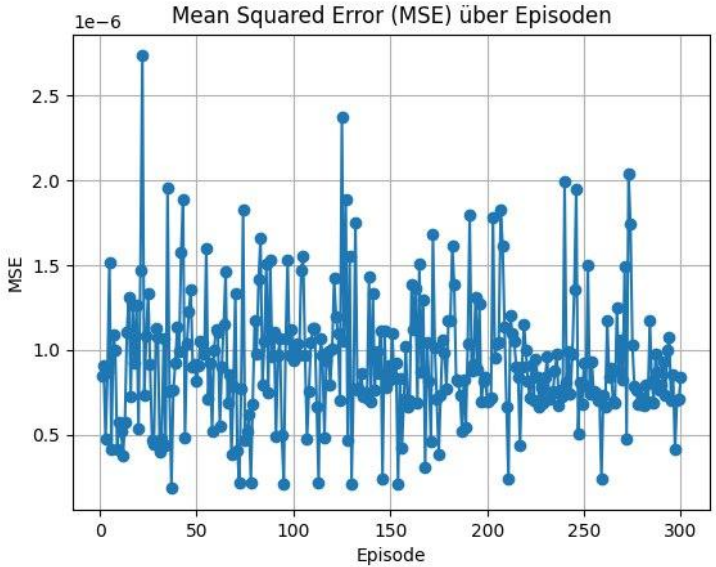
# CNN-Ergebnisse

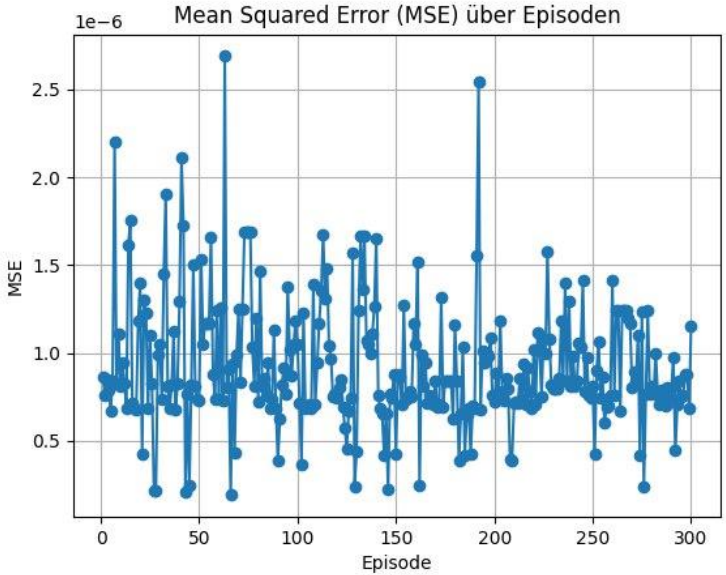
testszenario → parameter variieren wie wirkt sich das auf die Performance aus



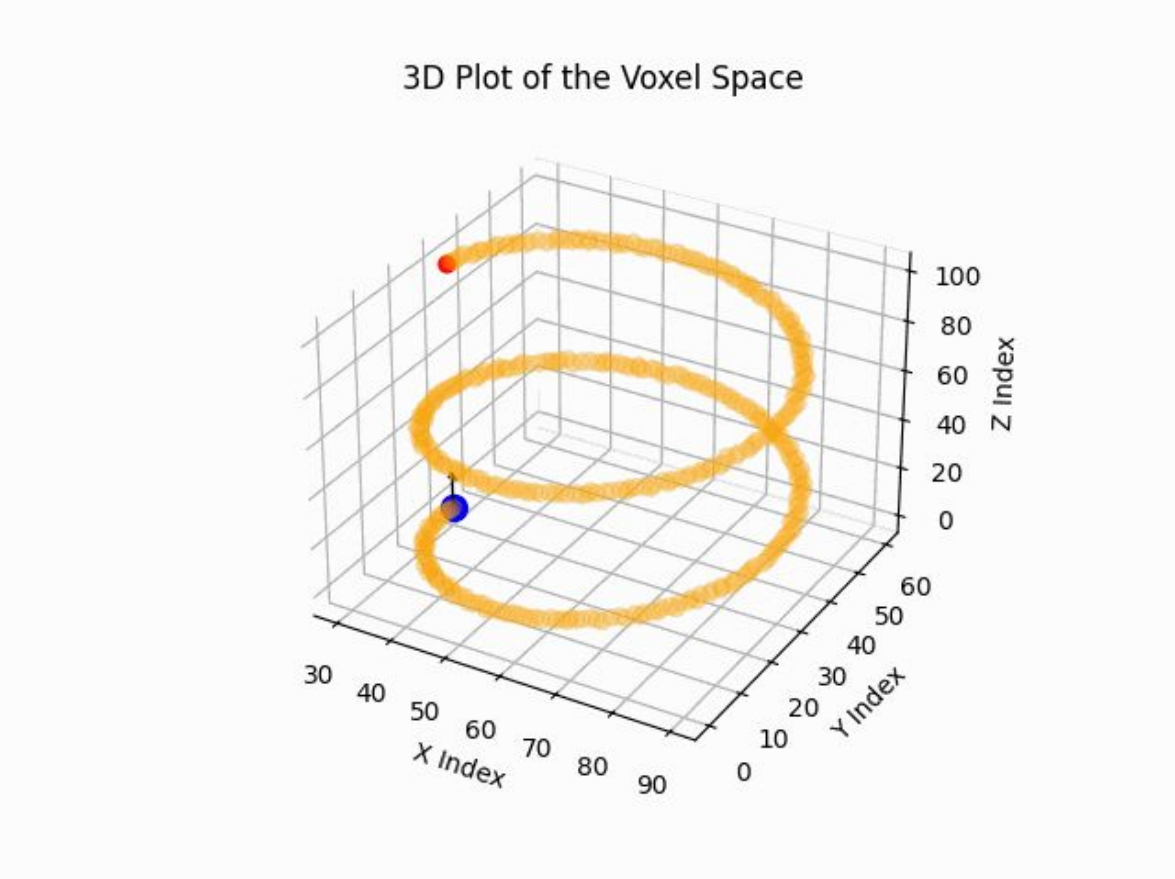
Ideen zur MSE-Reduktion:

- Toleranz verringern
- Toleranz dynamisch verringern über Episoden (Decay)
- Orientierung nicht berücksichtigen
- mehr Episoden
- epsilon anpassen abhängig von letzten 10-20 rewards / steps

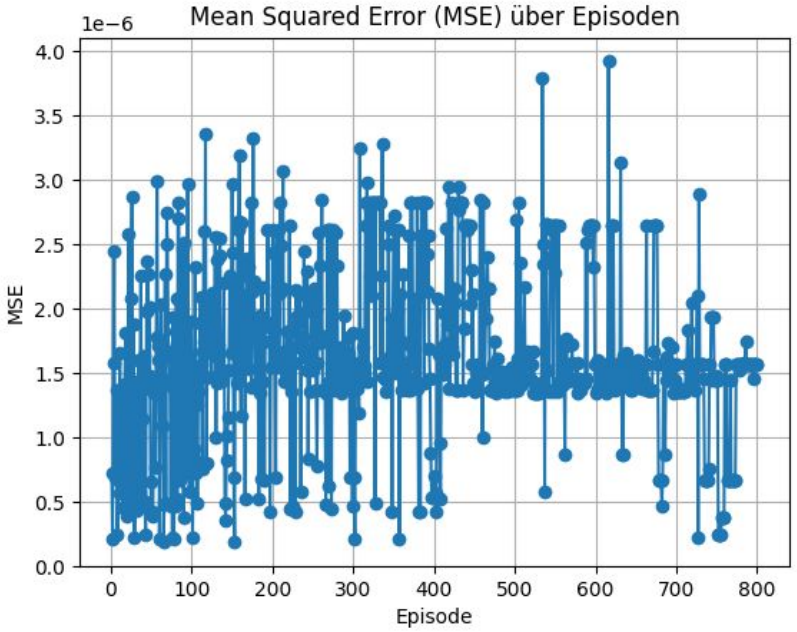


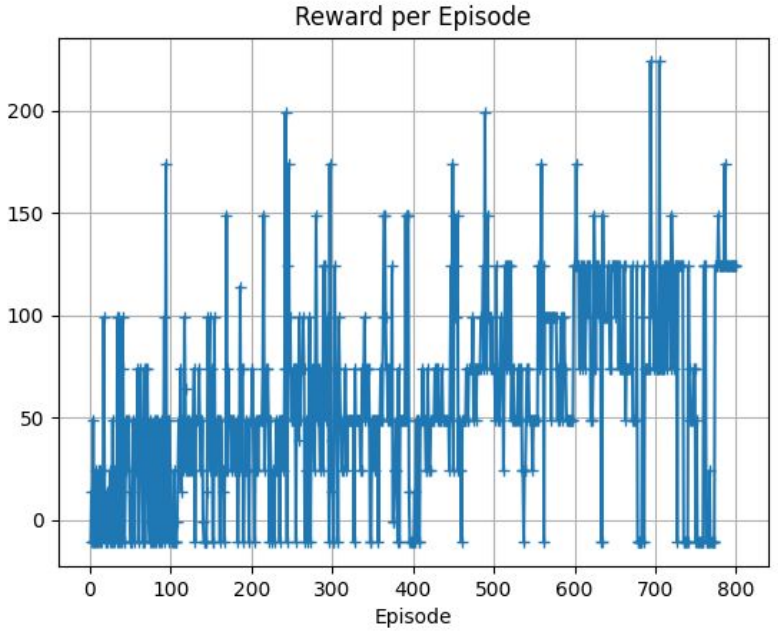



# Multimodal late fusion CNN-Ergebnisse



# Multimodal late fusion CNN-Ergebnisse



# Gelöste Probleme/ Herausforderungen

Motivation: Viel Exploration nicht nur Exploitation im Verhältnis zur Vorlesung

- Wahl der Netzwerkarchitektur (verschiedene Deep Q Learning Netzwerke)
- n-step bootstrapping Implementierung für Deep Q-Learning
- Balance zwischen Bewegung auf der Trajektorie und TCP Orientierung
- Reward-Funktion
- gymnasium Environment (z.B. bzgl. Logging der Ergebnisse)
- Rechenkapazität
- Speichern und Laden (Modell)
- Parametereinstellungen
- Zusätzliche Translation um die Helix am TCP-Start festzulegen