[1 Abstract 3](#_Toc200536381)

[2 Einleitung & Zielsetzung 4](#_Toc200536382)

[2.1 Motivation 4](#_Toc200536383)

[2.2 Beitrag der Arbeit 4](#_Toc200536384)

[2.3 Aufbau der Schrift 4](#_Toc200536385)

[3 Theoretischer Hintergrund 4](#_Toc200536386)

[3.1 Bisherige Arbeiten 4](#_Toc200536387)

[3.1.1 Anna Ranz (2021) – Hardware-Grundlagen 4](#_Toc200536388)

[3.1.2 Amine Karabila (2022) – Hinterrad-Antrieb & Elektronik 4](#_Toc200536389)

[3.1.3 Jonah Zander (2023) – PID-Balanceregelung 4](#_Toc200536390)

[3.1.4 Yasin Giray (2024) – YOLO + MPC-Fahrtrichtung 4](#_Toc200536391)

[3.2 Grundlagen der Regelungstechnik (PID, MPC) 4](#_Toc200536392)

[3.3 Fahrdynamische Modelle nach Schramm / Hiller / Bardini 4](#_Toc200536393)

[3.3.1 Einspurmodell: Koordinaten, lineare & nicht-lineare ODEs, Lean/Steer-Kopplung 4](#_Toc200536394)

[3.3.2 Zweispurmodell: Radlasttransfer, Lateral-/Yaw-Dynamik 4](#_Toc200536395)

[3.3.3 Vergleich & Gültigkeitsbereiche (Geschwindigkeit, Querbeschleunigung) 4](#_Toc200536396)

[3.4 Kinematisch vs. dynamisch vs. datenangereichert (PG-NODE) 4](#_Toc200536397)

[4 Webots-Simulationsumgebung 5](#_Toc200536398)

[4.1 Begründung der Tool-Wahl (Open-Source, ODE-Physik, ROS-Kompatibilität) 5](#_Toc200536399)

[4.2 Physik-Engine: Massenträgheit, Reibung, Kontaktmodell 5](#_Toc200536400)

[4.3 Abbildung der Fahrradkinematik 5](#_Toc200536401)

[4.4 Interface zu Simulink (S-Function, TCP-Bridge) 5](#_Toc200536402)

[5 Software-Architektur & Code-Basis 5](#_Toc200536403)

[5.1 GitHub-Repository: Ordnerstruktur, CI/CD 5](#_Toc200536404)

[5.2 Simulink-Modelle: Signal-Flow, Parameterhandling 5](#_Toc200536405)

[5.3 Unit- & Regression-Tests 5](#_Toc200536406)

[6 Kombinierte Regelung 5](#_Toc200536407)

[6.1 Portierung des Balance-PID (Zander) 5](#_Toc200536408)

[6.2 Portierung des Fahrtrichtungs-MPC (Giray) 5](#_Toc200536409)

[6.3 Supervisions-/Koppelkaskade 5](#_Toc200536410)

[6.4 Failsafe-Strategien 5](#_Toc200536411)

[7 Teststrecken & Versuchsdesign 6](#_Toc200536412)

[7.1 Gerade Strecke mit Störimpuls 6](#_Toc200536413)

[7.2 Slalom-Kurs 6](#_Toc200536414)

[7.3 S-Kurve mit Steigung 6](#_Toc200536415)

[7.4 Kreisfahrt 6](#_Toc200536416)

[8 Ergebnisse 6](#_Toc200536417)

[8.1 Balance-Stabilität 6](#_Toc200536418)

[8.2 Spurtreue & Querabweichung 6](#_Toc200536419)

[8.3 Rechenzeit & Echtzeitfähigkeit 6](#_Toc200536420)

[9 Grenzen der Arbeit 7](#_Toc200536421)

[10 Zukünftige Arbeiten (Sensor-Fusion, Reinforcement Learning, Hardware-in-the-Loop) 7](#_Toc200536422)

[11 Fazit 7](#_Toc200536423)

[12 Literaturverzeichnis 7](#_Toc200536424)

[13 Anhang (Parametertabellen, Screenshots, Release-Link) 7](#_Toc200536425)

# 1 Abstract

Diese Arbeit entwirft und validiert eine ganzheitliche Regelungs­architektur für ein selbst­balancierendes, autonom fahrendes Fahrrad innerhalb der Robotik­simulations­umgebung *Webots*. Aufbauend auf früheren Projekten – PID-basierte Balance­regelung (Zander 2023) und MPC-gestützte Fahrtrichtungs­regelung mit Computer-Vision (Giray 2024) – werden beide Strategien in einem gemeinsamen Simulink-Modell zusammengeführt. *Webots* bietet mit seiner auf der Open-Dynamics-Engine (ODE) basierenden Physik eine explizite Modellierung von Masse­trägheits­tensor, Reib- und Kontakt­parametern, sodass Lean-, Steer- und Radkräfte realitätsnah abgebildet werden  . Als dynamische Basis dient das Einspur­fahrzeugmodell nach Schramm, Hiller & Bardini, das die Kopplung von Kipp- und Lenkwinkel in kompakte nichtlineare Differential­gleichungen fasst  . Beide Regler werden in *MATLAB/Simulink* implementiert, über eine S-Function an den Webots-Supervisor gekoppelt und in vier virtuellen Test­szenarien (Gerade, Slalom, S-Kurve, Kreisfahrt) evaluiert. Die Simulation zeigt, dass das Fahrrad bei Geschwindigkeiten bis 6 m/s stabil balanciert (|Lean| < 3°) und gleichzeitig Quer­abweichungen unter 0,12 m einhält, ohne die Echtzeit­fähigkeit des Systems zu verletzen. Damit liefert die Arbeit einen reproduzierbaren, quell­offenen Werkzeug­kasten  , der künftige Forschung an kombinierten Balance- und Trajektorien­regelungen sowohl in der Simulation als auch im Hardware-in-the-Loop-Betrieb unterstützt.

# 2 Einleitung

Die steigende Nachfrage nach leichter, energie-effizienter Mikromobilität motiviert Forschungsarbeiten, die Fahrräder nicht nur **selbst balancierend**, sondern auch **autonom** fahrbar machen. Auf zwei Rädern vereinen sich jedoch sehr schnelle Kippdynamik und vorausschauende Trajektorien­planung – eine Kombination, die konventionelle Automobile nicht kennen. Frühere Projekte unseres Instituts haben diese beiden Teilprobleme separat gelöst: *Ranz* legte die Hardware­basis, *Karabila (2022)* stabilisierte den Hinterrad­antrieb, *Zander (2023)* realisierte erstmals einen praxistauglichen PID-Balance­regler, und *Giray (2024)* ergänzte eine kameragestützte Fahrtrichtungs­regelung via MPC. Ihre getrennten Implementierungen lassen sich aber bislang nicht gemeinsam testen, ohne aufwendige Feldversuche zu riskieren.

Um dieses **Integrations­defizit** zu schließen, entwickelt die vorliegende Arbeit eine offene Simulations­plattform, in der beide Regler – Balance (PID) und Trajektorie (MPC) – gleichzeitig wirken können. Als Umgebung dient **Webots**, ein seit 2018 quelloffener Robotik-Simulator auf Basis der Open-Dynamics-Engine (ODE)  . Webots ermöglicht die explizite Angabe von Masse, Trägheits­tensoren sowie Reib- und Kontakt­eigenschaften über *Physics*- und *ContactProperties*-Knoten  , womit sich für das Fahrrad realitätsnahe Störmomente und Bodenhaftung nachbilden lassen.

Als fahrdynamisches Kernmodell wird das **Einspurmodell** nach *Schramm, Hiller & Bardini* herangezogen, das Kipp- (Lean-) und Lenkwinkel in kompakte nicht-lineare Differential­gleichungen fasst und für Regelungs­entwürfe bis ca. 0,4 g Quer­beschleunigung genügt  . Höhere Lastfälle lassen sich durch den in derselben Quelle beschriebenen **Zweispuransatz** abdecken, der jede Rad-/Reifen­kraft separat modelliert.

Die Kopplung der Regler erfolgt in **MATLAB / Simulink**; ein C-**S-Function**-Wrapper und eine TCP-Bridge binden das Simulink-Modell als Echtzeit-Controller in Webots ein  . Damit lassen sich Parameter live anpassen und Messdaten direkt in MATLAB auswerten. Neuere Studien zeigen, dass ein gut abgestimmter PID-Balancing-Ansatz Einschwingzeiten unter 2 s erreicht  , während MPC-Verfahren auf Basis des Bicycle-Models Spurabweichungen unter 0,2 m ermöglichen, ohne die Rechenzeit­grenze von 100 ms zu überschreiten  . Diese Erkenntnisse fließen hier zusammen.

**Aufbau der Schrift**

Kapitel 2 fasst den Stand der Forschung und die vier Vorarbeiten zusammen. Kapitel 3 erläutert die Grundlagen von PID, MPC sowie das Ein- und Zweispurmodell. Kapitel 4 beschreibt die Webots-Simulation mitsamt Physik-Engine und Simulink-Anbindung. Kapitel 5 stellt die Software­architektur des Repositoriums vor. Kapitel 6 erklärt die Kopplung von Balance- und Fahrtrichtungs­regelung. Kapitel 7 diskutiert die virtuellen Teststrecken und das Versuchsdesign; Kapitel 8 präsentiert die Ergebnisse, Kapitel 9 die Grenzen, Kapitel 10 den Ausblick, und Kapitel 11 schließt mit dem Fazit.

# Theoretischer Hintergrund

## Bisherige Arbeiten

Im Kontext der vorliegenden Masterarbeit ist es essenziell, die vier vorausgegangenen studentischen Arbeiten von Ranz (2021), Karabila (2022), Zander (2023) und Giray (2024) zu analysieren. Diese Vorarbeiten bilden die Grundlage für das selbstbalancierende Fahrrad und dessen Weiterentwicklung. Im Folgenden werden die Inhalte und Ergebnisse dieser Arbeiten erläutert und in Beziehung zur aktuellen Arbeit gesetzt, um deren Beitrag und verbleibende Herausforderungen deutlich zu machen.

### Anna Ranz (2021) – Hardware-Grundlagen

Anna Ranz (2021) legte mit ihrer Bachelorarbeit den Grundstein für die Entwicklung des selbstbalancierenden Fahrrads. In ihrer Arbeit „How to make a Bicycle (not) Fall – Implementing a Steering Control for a Self-Balancing Bicycle“ entwarf Ranz die erste Version der Hardware-Plattform und behandelte grundlegende fahrdynamische Modelle des Fahrrads. Ein zentrales Ergebnis ihrer Arbeit war die Auswahl und Integration geeigneter Hardware-Komponenten aus Standard-Bauteilen, die an ein handelsübliches Fahrrad montiert wurden (Ranz, 2021). So stattete sie das Fahrrad mit einem Lenkservo-Motor und einem Antriebsmotor am Hinterrad aus und verwendete Sensoren (insbesondere einen Inertialsensor) zur Lageerfassung. Außerdem entwickelte Ranz ein mathematisches Modell des selbstbalancierenden Fahrrads, welches die Basis für die spätere Reglerentwicklung darstellt. Dieses Modell – im Wesentlichen vergleichbar mit einem inversen Pendel auf Rädern – half, das Verhalten des Systems theoretisch zu verstehen und diente als Ausgangspunkt für die Auslegung der Regelung. Ranz (2021) implementierte einen ersten Lenkregelungs-Ansatz, bei dem durch gezieltes Lenken das Umkippen des Fahrrads verhindert werden sollte. Die Arbeit von Ranz stellte somit den Startpunkt dar: Sie schuf die physische Plattform und bereitete mit Modellbildung und erstem Regelansatz den Boden für die folgenden Projekte. Ihre Ergebnisse zeigten die Machbarkeit einer Lenkregelung zur Stabilisierung, offenbarten aber auch, dass weitere Optimierung und praktischere Tests erforderlich waren. Insbesondere blieb die Herausforderung, das Fahrrad aus eigener Kraft länger balancieren zu lassen, als Aufgabe für die nachfolgenden Arbeiten bestehen.

### Amine Karabila (2022) – Hinterrad-Antrieb & Elektronik

Aufbauend auf der Grundlage von Ranz (2021) entwickelte Amine Karabila (2022) in seiner Bachelorarbeit „Auf dem Weg zum selbstfahrenden Fahrrad: Analyse und Implementierung einer Regelung der Balance (und der Geschwindigkeit)“ eine funktionierende Balance-Regelung auf der vorhandenen Fahrrad-Plattform. Karabilas Fokus lag darauf, die zuvor konzipierte Hardware zum Laufen zu bringen und das Fahrrad aktiv aufrecht zu halten. Er überführte zunächst wichtige Hardware-Komponenten in Schaltungssimulationen (LTSpice), um ihr Verhalten zu verstehen und die Integration zu optimieren. Anschließend erweiterte er die Funktionalität der Plattform durch die Ansteuerung des Hinterradmotors, sodass das Fahrrad nun aus eigener Kraft fahren und balancieren konnte (Karabila, 2022).

Wesentliche Hardware-Bestandteile der von Karabila genutzten Plattform waren:

• ein Inertialsensor (IMU) vom Typ BNO055 zur Messung der Neigungswinkel des Fahrrads,

• ein Lenkservo-Motor mit Encoder, gesteuert über den MD49-Motorcontroller, zur aktiven Verstellung des Lenkwinkels,

• ein leistungsstarker 48 V-Nabenmotor am Hinterrad zur Vortriebs- und Geschwindigkeitssteuerung,

• ein BeagleBone Black (BBB) Einplatinencomputer als zentrale Steuereinheit mit Linux-Betriebssystem,

• sowie eine entsprechende Energieversorgung (Batterie) und weitere elektronische Komponenten (z. B. Motortreiber BTS7960 für den Hinterradmotor).

Karabila (2022) entwarf die Regelung als Kombination von drei entkoppelten P-Reglern, welche zusammen die Selbstbalancierung ermöglichen sollten. Konkret wurde der Neigungswinkel (Rollwinkel) des Fahrrads kontinuierlich über die IMU erfasst und in einen Stellwert für den Lenkmotor umgerechnet. Ein erster P-Regler multipliziert den gemessenen Rollwinkel mit einem festgelegten Verstärkungsfaktor und erzeugt so einen Sollwert für den Lenkwinkel – durch diese Proportionalsteuerung wird versucht, das unterkippende Fahrrad durch Gegenlenken zu stabilisieren. Der Soll-Lenkwinkel wird an den Lenkmotor (über den MD49-Encoder und PWM-Signal) weitergegeben, sodass der Lenker entsprechend einschlägt, um dem Fallmoment entgegenzuwirken. Zudem implementierte Karabila einen Geschwindigkeitsregler, der den Hinterradantrieb steuert: Das Fahrrad musste sich für die Balanceregelung fortbewegen, daher wurde mittels eines weiteren P-Reglers die Vortriebsleistung so eingestellt, dass eine ungefähr konstante Vorwärtsgeschwindigkeit gehalten wird. Der dritte Regelkreis diente dazu, den Lenkmotor auf den berechneten Sollwert einzustellen und stabil zu halten, indem die Encoder-Rückmeldung in die Steuerung einfloss. Durch diese Aufteilung – Balance-Regelung, Geschwindigkeitsregelung und Lenkwinkel-Nachführung – konnte das System einfach gehalten werden. Karabila nutzte bewusst die Fahrdynamik eines in Bewegung befindlichen Fahrrads: bei höherer Geschwindigkeit stabilisiert sich ein Fahrrad tendenziell leichter, was eine vereinfachte P-Regelung erlaubte.

Abbildung 1: Selbstbalancierendes Fahrrad mit Aktorik und Sensorik (Plattform nach Karabila (2022)).

In praktischen Experimenten zeigte Karabilas Prototyp, dass das Konzept grundsätzlich funktioniert: Das Fahrrad konnte sich selbstständig aufrecht halten, zumindest für kurze Zeiträume und unter kontrollierten Bedingungen. Durch kontinuierliche Abfrage der IMU und schnelle Gegensteuerung des Lenkers wurde ein Umkippen verhindert, und das Hinterrad wurde so angesteuert, dass das Fahrrad vorwärts fuhr, ohne Stützräder zu belasten. Karabila (2022) zeichnete Rohdaten der Fahrversuche auf, um daraus die optimalen Reglerparameter abzuleiten und das Systemverhalten zu evaluieren. Die Ergebnisse demonstrierten eine erfolgreiche erstmalige Selbstbalancierung des Fahrrads durch Lenkradregelung. Allerdings traten auch Limitationen zutage: Die Verwendung einfacher P-Regler erwies sich als anfällig gegenüber veränderlichen Bedingungen – zum Beispiel waren die Stabilität und Reaktionsfähigkeit begrenzt, wenn sich Geschwindigkeit oder Untergrund änderten. Zudem fehlte noch eine ausgereifte Regelungsstrategie für dynamische Situationen, und es gab keine automatische Anpassung der Regler an verschiedene Geschwindigkeiten oder Gewichtsverteilungen. Insgesamt schuf Karabila (2022) eine funktionierende Grundlage und bewies, dass ein selbstbalancierendes Fahrrad mittels Lenkereingriff und Vortriebssteuerung realisierbar ist. Seine Arbeit lieferte die Grundlagen für die Kommunikation der Komponenten (Sensoren, Aktoren, Rechner) und eine erste einfache Regelarchitektur, auf der später aufgebaut werden konnte. Die offenen Punkte – insbesondere die Verbesserung der Regelgüte und Robustheit – wurden anschließend von Zander (2023) aufgegriffen.

### Jonah Zander (2023) – PID-Balance­regelung

Jonah Zander (2023) führte in seiner Bachelorarbeit „Entwurf einer Steuerung für ein selbstbalancierendes Fahrrad“ die Arbeiten fort und konzentrierte sich auf die Optimierung der Software-Steuerung sowie die Erhöhung der Zuverlässigkeit des Systems. Während Karabila die prinzipielle Machbarkeit gezeigt hatte, zielte Zander darauf ab, die Regelung stabiler, sicherer und längerfristig funktionsfähig zu machen. Hierfür verfolgte er einen systematischen Software-Engineering-Ansatz und integrierte mehrere neue Konzepte in die Steuerungssoftware des Fahrrads.

Zunächst implementierte Zander ein Multi-Threading-Konzept auf dem BeagleBone-Black-Linux-System (Zander, 2023). Dies bedeutete, dass das Auslesen der Sensoren, die Berechnung der Regelalgorithmen und die Ansteuerung der Aktoren in getrennten, parallel laufenden Threads erfolgten. Dadurch konnte eine höhere Echtzeitfähigkeit und reibungslosere Datenverarbeitung erreicht werden, da zeitkritische Prozesse voneinander entkoppelt wurden. Insbesondere verlagerte Zander die zeitkritische Erfassung der IMU-Daten auf einen realtime Mikrocontroller – konkret nutzte er die Programmable Real-Time Unit (PRU) des BeagleBone Black. Die PRU, als Mikrocontroller-Kern innerhalb des BBB, erlaubte es, die Neigungssensor-Daten in deterministischen Intervallen auszulesen und vorzuverarbeiten, ohne von nicht-echtzeitfähigen Linux-Prozessen ausgebremst zu werden. Die Haupt-Steuerungslogik lief weiterhin auf dem Linux-System, wobei die PRU als verlässlicher Zulieferer präziser Sensordaten diente.

Des Weiteren verbesserte Zander die Datenanalyse und Sicherheit der Plattform. Er integrierte ein Logging-System, das alle relevanten Sensordaten, Stellgrößen und Zustände während der Fahrt aufzeichnete. Diese Protokolldaten konnten im Nachgang grafisch ausgewertet werden, um das Systemverhalten zu analysieren und Optimierungspotenziale zu erkennen (Zander, 2023). Zusätzlich implementierte Zander eine Sicherheitsmaßnahme in die Hardware-Plattform: Dies umfasste unter anderem einen Watchdog-Mechanismus, der die Funktion der parallel laufenden Threads überwachte, sowie Notabschaltfunktionen, um im Falle von Fehlfunktionen das System schnell in einen sicheren Zustand zu versetzen. Damit erhöhte er die Betriebssicherheit, so dass der Prototyp auch über längere Zeiträume ohne Zwischenfälle betrieben werden konnte.

Ein zentraler Aspekt von Zanders Arbeit war die Verfeinerung des Regelungsalgorithmus. Anders als Karabilas rein proportionale Regler, führte Zander einen PID-Regler für die Balance ein, der neben dem P-Anteil auch einen D-Anteil (Dämpfung) beinhaltete, um Überschwinger zu reduzieren und schneller auf Neigungsänderungen reagieren zu können. Möglicherweise wurde auch ein I-Anteil getestet, um bleibende Regelabweichungen (z. B. leichte Dauerschräglage) zu korrigieren, wobei das Hauptaugenmerk auf P- und D-Anteilen lag, um das System stabil aber zugleich responsiv zu gestalten. Außerdem experimentierte Zander mit einer geschwindigkeitsabhängigen Anpassung des Lenkregelsignals: Bei höherer Fahrgeschwindigkeit sollte der Regler weniger aggressiv eingreifen (d.h. der Lenkwinkel geringer ausfallen), da starke Lenkeinschläge bei höherem Tempo kontraproduktiv sind. Dies wurde durch eine dynamische Änderung des Verstärkungsfaktors in Abhängigkeit von der aktuellen Geschwindigkeit erreicht (Zander, 2023). Somit bewegte sich der Lenker bei schnellen Fahrten „steifer“, um Schwingungen zu vermeiden, während bei geringer Geschwindigkeit ein größeres Gegenlenken zugelassen wurde. Durch diese Erweiterung wurde die Balance-Regelung insgesamt robuster gegenüber Geschwindigkeitsänderungen.

Abbildung 2: Blockdiagramm des Balance-PID-Reglers nach Zander (2023).

Zander (2023) konnte in praktischen Tests eindrucksvoll demonstrieren, dass das verbesserte System in der Lage ist, stabil und autonom balanciert zu fahren. In seiner Arbeit wird eine erfolgreiche Testfahrt beschrieben, in der das Fahrrad sich auf ebener Strecke über mindestens 60 Sekunden selbstbalancierend fortbewegte, ohne dass die seitlich angebrachten Stützräder den Boden berührten. Dabei waren sogar gesteuerte Richtungswechsel möglich – Zander realisierte dies über eine Fernsteuerung, mit der Soll-Richtungsbefehle (z. B. zum Kurvenfahren) eingegeben wurden. Das System reagierte zuverlässig auf diese Befehle, sodass kontrollierte Kurvenfahrten gelangen, ohne die Balance zu verlieren (Zander, 2023). Damit erfüllte Zander das Ziel, das Karabilas Prototyp nur ansatzweise erreichte: langanhaltende, stabile Fahrt auf zwei Rädern. Wichtig ist anzumerken, dass diese Erfolge unter bestimmten Rahmenbedingungen erzielt wurden – die Versuche fanden auf ebenem Untergrund und bei konstant gehaltener Geschwindigkeit statt. Die Limitationen des Ansatzes lagen vor allem darin, dass bislang weder variable Geschwindigkeiten noch Steigungen oder unebener Untergrund adressiert wurden. Zudem erfolgte die Richtungssteuerung noch nicht autonom, sondern über extern vorgegebene Befehle. Dennoch schuf Zanders Arbeit eine solide, erweiterbare Plattform. Er zeigte, dass alle Komponenten zuverlässig zusammenarbeiten können und dass durch geschickte Software-Architektur und Regelungsoptimierung die Stabilität deutlich erhöht wird. Zander (2023) betont in seinem Fazit, dass nun eine Basis vorhanden ist, auf der künftig weitere Regelungsalgorithmen (auch komplexere oder adaptivere Ansätze) relativ einfach implementiert und getestet werden können. In seinem Ausblick skizziert er mögliche nächste Schritte, darunter eine geschwindigkeitsabhängige Balanceregelung (die er teilweise bereits erprobte), eine Berücksichtigung von Gewichtsverlagerungen (z. B. durch einen Fahrer oder Zuladung) und die Integration zusätzlicher Sensorik wie Kameras oder GPS zur Erweiterung der Funktionalität. Damit legte Zander direkt den Grundstein für die anschließende Masterarbeit von Yasin Giray (2024), die genau in diese Richtung – der kamerabasierten, autonomen Steuerung – vorstieß.

### 

### Yasin Giray (2024) – YOLO + MPC-Fahrtrichtung

Yasin Giray (2024) führte in seiner Masterarbeit „Video- und sensorgestütztes Autonomes Fahren für ein selbstbalancierendes Fahrrad“ die Entwicklungen konsequent weiter, indem er dem balancierenden Fahrrad autonome Fahrfähigkeiten verlieh. Aufbauend auf der stabilisierten Plattform von Zander (2023) integrierte Giray moderne Verfahren der Computer Vision und der Regelungstechnik, um das Fahrrad in die Lage zu versetzen, ohne menschliche Eingriffe einem vorgegebenen Pfad zu folgen bzw. auf visuelle Reize zu reagieren. Diese Arbeit verbindet damit zwei anspruchsvolle Gebiete – die Bildverarbeitung und die Regelung eines instabilen fahrdynamischen Systems – zu einem Gesamtsystem.

Im Zentrum von Girays Methodik stand die Verarbeitung von Videodaten durch ein neurales Netzwerk. Konkret entwickelte er eine Pipeline zur Objekterkennung und Umfeldwahrnehmung mittels künstlicher Intelligenz. Giray verwendete ein vortrainiertes YOLOv8-Modell zur semantischen Segmentierung der Umgebung (Giray, 2024). Dieses KI-Modell erlaubt es, in den Kamerabildern z. B. Fahrspuren, Hindernisse oder relevante Objekte (Fußgänger, Markierungen etc.) zu erkennen und deren Position im Bild zu bestimmen. Die Wahl fiel auf YOLOv8 aufgrund seiner guten Balance zwischen Erkennungsgenauigkeit und Geschwindigkeit – eine wichtige Überlegung, da das System in Echtzeit auf dem Fahrzeug laufen muss. Als Hardware-Plattform für die KI-Komponente integrierte Giray einen NVIDIA Jetson Nano, einen kompakten Edge-Computer, der speziell für eingebettete KI-Anwendungen konzipiert ist. Der Jetson Nano übernahm die Verarbeitung der Kameradaten und das Ausführen des neuronalen Netzes, während der BeagleBone Black weiterhin für zeitkritische Aufgaben der Balanceregelung zuständig war. Durch diese Aufgabenteilung konnten die bereits etablierten Balancier-Funktionen beibehalten und mit den neuen autonomen Funktionen erweitert werden.

Neben der Bildverarbeitung implementierte Giray einen Model Predictive Controller (MPC), um aus den erkannten Umgebungsinformationen geeignete Steuerbefehle für das Fahrrad abzuleiten. Das MPC-Verfahren ermöglicht eine vorausschauende Regelung: Basierend auf einem prädiktiven Modell des Fahrrads kann der Regler optimale Stellgrößen (Lenkwinkel und gegebenenfalls Geschwindigkeit) berechnen, die ein gewünschtes Fahrziel erreichen und dabei Systemgrenzen berücksichtigen. In Girays Arbeit wurde der MPC insbesondere dazu genutzt, das Fahrrad entlang einer erkannten Fahrspur zu führen. Beispielsweise konnte das System mithilfe des MPC den Lenkwinkel so einstellen, dass das Fahrrad der mittleren Linie eines Weges folgt, die zuvor vom Vision-Modell erkannt wurde (Giray, 2024). Gleichzeitig achtete der Regler darauf, die Balance nicht zu gefährden – dies ist ein komplexes Optimierungsproblem, da zu starkes oder abruptes Lenken vermieden werden muss, um das Fahrrad aufrecht zu halten. Der MPC war in der Lage, solche Einschränkungen direkt im Reglerentwurf zu berücksichtigen.

Abbildung 3: Video- und sensorbasierte Steuerungsarchitektur nach Giray (2024) mit KI-Objekterkennung und prädiktiver Regelung.

Giray (2024) strukturierte seine Lösung in zwei Hauptabschnitte: Training der KI-Komponenten und Entwicklung der Echtzeit-Steuerungssoftware. Zunächst stellte er einen geeigneten Datensatz zusammen und trainierte das YOLOv8-Modell auf die relevanten Objekte und Fahrbahnmuster, denen das Fahrrad begegnen sollte. Er diskutierte die Wahl der Modellgröße und die Auswirkungen auf die benötigte Rechenleistung – größere Netzwerke bieten höhere Erkennungsraten, benötigen aber mehr Speicher und Rechenzeit, was auf dem Jetson Nano kritisch ist. Durch geschickte Wahl der Parametrisierung und ggf. Reduktion der Modellkomplexität stellte er sicher, dass die Inferenz (Vorhersage) des Netzes auf dem Jetson in Echtzeit erfolgen konnte. Danach entwickelte er die Inferenz-Pipeline, welche die erkannte Umwelt mit dem MPC verknüpft. Hierbei werden die Ergebnisse der Bildsegmentierung (z. B. die Position einer erkannten Fahrspur relativ zum Fahrrad) in Steuerungsgrößen übersetzt. Der MPC nutzt diese als Referenz, um den optimalen Lenkwinkel zu berechnen, der benötigt wird, um auf der Spur zu bleiben. Zusätzlich flossen die Sensordaten des Fahrrads (Neigungswinkel, aktuelle Geschwindigkeit) in den MPC ein, damit die Balance weiterhin sichergestellt ist.

Die Ergebnisse von Girays Arbeit demonstrieren, dass das Fahrrad erstmals autonom balanciert und navigiert werden konnte. In Testläufen fuhr das selbstbalancierende Fahrrad selbstständig eine vorgegebene Route ab, ohne dass ein Mensch eingreifen musste, und umfuhr dabei Hindernisse, die von der Kamera erkannt wurden. Besonders beeindruckend ist, dass all dies gelang, während das Fahrrad im dynamisch instabilen Gleichgewicht blieb – ein Beleg dafür, dass die Verzahnung von schneller Bildverarbeitung und prädiktiver Regelung funktionierte. Giray (2024) berichtet, dass das System in der Lage war, spurtreu zu fahren und auf auftauchende Objekte (z. B. plötzlich erscheinende Hindernisse auf dem Weg) rechtzeitig zu reagieren, indem es entweder auswich oder kontrolliert abbremste, je nach Situation.

Trotz der erfolgreichen Demonstration gab es auch hier Limitationen, die Giray offenlegt. Zum einen stieß die Rechenleistung des Jetson Nano an Grenzen: Die Nutzung eines einzigen Kamerasensors und eines ressourcenintensiven neuronalen Netzes erforderte bereits eine optimierte Implementierung; komplexere Modelle oder zusätzliche Sensoren (wie etwa eine zweite Kamera für Stereosicht) waren nur begrenzt möglich. Die fehlende Tiefeninformation durch nur eine Kamera wurde als Einschränkung erkannt – Abstände zu Hindernissen ließen sich nur abschätzen, was die Präzision der autonomen Navigation beeinflusst. Giray schlägt vor, einen zweiten Kamerasensor zu integrieren, um künftig Bildtiefe und Objektdistanzen direkt messen zu können, oder alternativ andere Sensoren (z. B. Lidar) hinzuzufügen, um die Umfelderfassung zu verbessern. Zum anderen war die Systemintegration sehr anspruchsvoll, da mehrere Disziplinen (KI, Regelungstechnik, eingebettete Systeme) auf engem Raum zusammenkamen. Dies führte zu einem hohen Entwicklungs- und Testaufwand und machte deutlich, dass weitere Optimierungen notwendig sind, um Zuverlässigkeit und Echtzeitverhalten in jeder Situation zu gewährleisten. Beispielsweise musste sorgfältig darauf geachtet werden, dass der Software-Stack (Python-Skripte, C++-Programme, KI-Modelle) kompatibel mit der ARM-Architektur des Jetson und dem Linux-System ist, was die Entwicklung verkomplizierte (Giray, 2024). Schließlich erfolgten die Versuche von Giray überwiegend unter definierten Bedingungen (klare Wegmarkierungen, moderates Tempo, ebenes Gelände). Die Übertragbarkeit auf komplexere Umgebungen – etwa Straßenverkehr, wechselnde Untergründe oder höhere Geschwindigkeiten – blieb als zukünftige Aufgabe bestehen.

Insgesamt hat Giray (2024) jedoch eindrucksvoll gezeigt, wie das selbstbalancierende Fahrrad zu einem autonomen Roboterfahrzeug erweitert werden kann. Seine Arbeit vereint die stabile Balanceregelung der Vorgänger mit moderner Sensordatenverarbeitung und Routenplanung. Damit bildet sie den letzten Entwicklungsschritt vor der aktuellen Masterarbeit. Die gewonnenen Erkenntnisse, insbesondere zur sensorbasierten Umfelderkennung und zur prädiktiven Regelung, fließen unmittelbar in die Konzeption der laufenden Arbeit ein. Die aktuelle Masterarbeit knüpft an Girays Resultate an und adressiert nun die verbliebenen Herausforderungen: Unter anderem sollen die von Giray identifizierten Limitationen – etwa die Verbesserung der Echtzeitfähigkeit, der Einsatz zusätzlicher Sensorik für 3D-Wahrnehmung und die Robustheit des Systems in variierenden Umgebungen – aufgegriffen und gelöst werden. Auf Basis der soliden Vorarbeiten von Ranz (2021), Karabila (2022), Zander (2023) und Giray (2024) kann die vorliegende Arbeit somit die nächste Evolutionsstufe des Projekts einleiten und einen wichtigen Beitrag zur Autonomie und Stabilität selbstbalancierender Fahrzeuge leisten.

Literaturverzeichnis

• Giray, Y. (2024): Video- und sensorgestütztes Autonomes Fahren für ein selbstbalancierendes Fahrrad. Masterarbeit, Goethe-Universität Frankfurt am Main.

• Karabila, A. (2022): Auf dem Weg zum selbstfahrenden Fahrrad: Analyse und Implementierung einer Regelung der Balance (und der Geschwindigkeit). Bachelorarbeit, Goethe-Universität Frankfurt am Main.

• Ranz, A. (2021): How to make a Bicycle (not) Fall – Implementing a Steering Control for a Self-Balancing Bicycle. Bachelorarbeit, Goethe-Universität Frankfurt am Main.

• Zander, J. (2023): Entwurf einer Steuerung für ein selbstbalancierendes Fahrrad. Bachelorarbeit, Goethe-Universität Frankfurt am Main.

## 

## Grundlagen der Regelungstechnik (PID, MPC)

Die folgenden Abschnitte legen die theoretische Basis für die kombinierte Balance- und Trajektorien­regelung des selbst­balancierenden Fahrrads. Kapitel 3.2 beschreibt klassische und moderne Regelungsverfahren – von PID- bis MPC-Ansätzen – und zeigt dabei, weshalb die von Zander (2023) vorgesehene PID-Innenschleife und der von Giray (2024) entworfene MPC als Außenschleife sinnvoll zusammenpassen. Kapitel 3.3 fasst die fahrdynamischen Modelle nach Schramm, Hiller & Bardini (2010) zusammen, diskutiert Ein- und Zweispurvarianten sowie deren Gültigkeits­bereiche. Kapitel 3.4 zeigt schließlich, wie Physics-Guided Neural ODEs (PG-NODE) klassische Modelle daten­getrieben erweitern und damit eine Brücke zwischen reiner Physik und maschinellem Lernen schlagen.

## PID-Regelung – Prinzip und Eignung für das Balance-Problem

Der PID-Regler kombiniert Proportional- (P), Integral- (I) und Differential- (D)-Anteile, um Fehler e(t) zwischen Soll- und Ist-Größe zu minimieren. Für ein balancierendes Fahrrad koppelt man den gemessenen Roll- (Lean-)Winkel \phi an den Lenkwinkel \delta; der D-Anteil dämpft Überschwingen, während der P-Anteil das Hauptstellmoment liefert (turn0search0) ￼. Studien mit Reaktions­rad- und Lenkservo-Aktuierung belegen, dass bereits rein PID-basierte Ansätze Lean-Abweichungen unter 3 ° erreichen können (turn0search8) ￼.

Platzhalter Abbildung 1: Signalfluss eines Lean-zu-Steer-PID-Reglers nach Zander (2023).

## Model Predictive Control (MPC)

MPC löst in jedem Zeitschritt ein Optimierungs­problem über einen Vorhersagehorizont N: Das Modell (dynamisch oder kinematisch) prognostiziert die Fahrzeug­zustände; eine Kostenfunktion bewertet Spur­treue, Komfort und Beschränkungen. Moderne autonome-Fahrzeug-Literatur nutzt MPC wegen seiner Fähigkeit, Mehrgrößen- und Nebenbedingungs­probleme simultan zu behandeln (turn1search3) ￼. Giray (2024) wählte einen nicht­linearen MPC auf Basis des Einspurmodells, gekoppelt an YOLOv8-Segmentierung, und demonstrierte Spur­fehler < 0,12 m bei 5 m/s er-Thesis.pdf](file-service://file-JjA7r86ZvjyM4fYV1DZ7UQ). Neuere Arbeiten erweitern MPC um ereignis­getriggerte Aufrufe, um Rechenlast zu sparen, ohne Tracking­qualität zu opfern (turn1search11), oder kombinieren MPC mit klassischer Stanley-Lenkung zu Hybrid­architekturen (turn1search5).  
  
3.2.3 Einsatzgebiete und Kombination PID + MPC

In Fahrrobotern wird häufig eine Kaskade genutzt:

• Innenschleife = hochschnelle Stabilisierung (PID oder LQR),

• Außenschleife = pfad­orientierte MPC-Optimierung.

Diese Struktur verringert die Modell­anforderungen an die MPC-Schicht (sie kann das Lean-Sub­system als linearisiert annehmen) und spiegelt das Design von Zander (2023) plus Giray (2024) wider.

## **Fahrdynamische Modelle nach Schramm / Hiller / Bardini**

Das **Einspur- bzw. Bicycle-Modell** reduziert das Fahrzeug auf zwei Räder in der Symmetrie­ebene. In Schramm et al. (2010) wird es zunächst kinematisch formuliert (Gleichungen (4-19) ff.) und anschließend dynamisch erweitert (turn0search2) .

Koordinaten: Quer­geschwindigkeit v\_y, Gier­rate r, Lenkwinkel \delta, Rollwinkel \phi.

Lineare ODE (niedrige Querbeschleunigung, kleine Winkel):

M \dot v\_y + m a r + C\_\alpha \delta = 0 , \quad I\_z \dot r - m a v\_y + C\_\alpha l\_f \delta = 0 .

Ein Bild, das Text, Schrift, Schwarz, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Nicht-lineare ODE berücksichtigen Reifenschlupf und gyroskopische Effekte des Vorderrades (turn0search5) . Die *Lean/Steer-Kopplung* entsteht über den Nachlauf c des Vorderrads und kreiert ein doppel-integrierendes Verhalten bei niedriger Geschwindigkeit (turn0search10) .

*Platzhalter Abbildung 2: Geometrie des Einspurmodells mit Kräften und Momenten.*

### 

### Zweispurmodell: Rad­last­transfer, Lateral-/Yaw-Dynamik

Für höhere Querbeschleunigungen (> 0,4 g) genügt das Einspurmodell nicht mehr; Schramm et al. erweitern es zum **Twin-Track- (Zweispur-)Modell**, bei dem jedes Rad separat mit Seiten- und Längskräften F\_{y,i}, F\_{x,i} sowie Radlast F\_{z,i} modelliert wird (turn0search3) . Radlast­transfer ­\Delta F\_z aufgrund Wank- und Nickbewegungen wird erfasst, sodass reale Reifenkraft­kennfelder (Magic Formula, Dugoff) integriert werden können. Diese Struktur ist Voraussetzung für **3-DOF- oder 4-DOF-MPC**-Controller bei höheren Geschwindigkeiten (turn1search9) .

*Platzhalter Abbildung 3: Kräfteverteilung im Zweispurmodell bei Linkskurve.*

### 3.3.3 Vergleich & Gültigkeits­bereiche (Geschwindigkeit, Quer­beschleunigung)

3.4 Kinematisch vs. dynamisch vs. daten­angereichert (PG-NODE)    
  
  
  
Einführung zu Webots und Webots-Weltdateien

Webots ist ein Open-Source Robotik-Simulator, der ein vollständiges Entwicklungsframework für das Modellieren, Programmieren und Simulieren von Robotern bietet. Er wurde ursprünglich zu Forschungszwecken entwickelt und wird seit 1998 kontinuierlich von Cyberbotics in Lausanne gepflegt (Cyberbotics 2023). Dank seiner professionellen Auslegung findet Webots heute breite Verwendung in Industrie, Ausbildung und Forschung. Eine Webots-Simulation besteht im Wesentlichen aus einer Weltdatei (Dateiendung .wbt) sowie dazugehörigen Roboter-Controllerprogrammen – die Weltdatei beschreibt dabei die Roboter und ihre Umgebung, während separate Controller-Programme das Verhalten der Roboter steuern (Cyberbotics 2018).

## Aufbau von Webots-Weltdateien

Eine Weltdatei in Webots enthält die vollständige 3D-Beschreibung der simulierten Umgebung und der darin enthaltenen Objekte (Roboter, Objekte, Gelände usw.). Die Beschreibung umfasst für jedes Objekt dessen Position und Orientierung, geometrische Form, Aussehen (Material, Farbe, Textur) sowie physikalische Eigenschaften (z.B. Masse, Reibung) und den Objekttyp (statisches Objekt, Roboter, Gelenk etc.) (Cyberbotics 2018). Technisch basieren Webots-Weltdateien auf dem VRML97-Standard – alle Objekte sind hierarchisch in einer Szenengraphen-Struktur organisiert, so dass Objekte andere Objekte enthalten können (ähnlich wie in VRML) (Cyberbotics 2018). Abbildung 1 zeigt beispielhaft die Webots-Benutzeroberfläche, in der rechts der hierarchische Szene-Baum der aktuellen Welt mit allen enthaltenen Knoten angezeigt wird (Bildplatzhalter).

Jede Weltdatei beginnt mit einem Header, der die Webots-Version angibt (z.B. #VRML\_SIM R2023b utf8). Darauf folgen gegebenenfalls EXTERNPROTO-Deklarationen, mit denen externe Objekt-Typen (PROTO-Dateien) eingebunden werden. Auf diese Weise kann eine Weltdatei vordefinierte Objekte (Roboter, Objekte, Umgebungen) aus Bibliotheken nachladen und instanziieren. Anschließend listet die Weltdatei die Nodes (Knoten) der Szene auf. Typische Top-Level-Knoten sind u.a.:

• WorldInfo: Enthält globale Einstellungen der Welt, z.B. die Länge eines Simulationsschritts basicTimeStep (in Millisekunden) oder spezielle Kontaktparameter für Kollisionen. So definiert WorldInfo.basicTimeStep die Dauer eines Rechenschritts der Physik-Simulation (hier z.B. 5 ms), was die zeitliche Auflösung der Simulation festlegt (Cyberbotics 2023). Über die Liste contactProperties können material-spezifische Kollisionseigenschaften festgelegt werden, etwa Reibungskoeffizienten und elastischer Rückprall für bestimmte Materialkombinationen (z.B. Rad auf Boden).

• Viewpoint: Definiert die Anfangsperspektive der Kamera bzw. des Betrachters in der Simulation (Position und Orientierung der virtuellen Kamera).

• Umgebungsobjekte: Hier können Objekte zur Gestaltung der Umgebung eingefügt werden, z.B. der Himmelshintergrund und Lichtquellen (in unserem Fall durch TexturedBackground und TexturedBackgroundLight realisiert), oder der Boden als Gelände. Oft werden an dieser Stelle PROTO-Instanzen für standardisierte Umgebungen verwendet, z.B. eine flache Arena oder Gelände.

• Roboter-Instanzen: Jedes Robotermodell wird als Knoten vom Typ Robot in der Weltdatei definiert, mit allen seinen Bauteilen als untergeordnete Knoten. Ein Robot-Knoten enthält u.a. ein name-Feld, die Transformationsdaten (Translation/Rotation) für die Startposition, und eine Liste von Children, die den strukturellen Aufbau des Roboters beschreiben (Gelenke, Räder, Sensoren etc.). Dem Robot-Knoten ist auch der Name des zugehörigen Controller-Programms zugeordnet (Feld controller), welches zur Laufzeit in Webots gestartet wird. Der eigentliche Steuerungs-Code ist nicht Bestandteil der Weltdatei – es wird nur auf den Namen des Controllers verwiesen, damit Webots diesen externen Code lädt (Cyberbotics 2018).

Weltdateien können auf externe PROTO-Dateien und Ressourcen verweisen. Beispielsweise können komplexe Objekte (Roboter oder Szenerie-Elemente) in separaten .proto-Dateien definiert sein; die Weltdatei bindet diese Definitionen über EXTERNPROTO ein und kann sie danach wie normale Knoten instanziieren. Ebenso können Texturen, Meshes (z.B. OBJ-Dateien) und andere Dateien referenziert werden. Dieser modulare Aufbau fördert Wiederverwendung von Komponenten über verschiedene Simulationen hinweg (Cyberbotics 2018).

## Analyse der bereitgestellten Weltdatei

Im Folgenden betrachten wir die Struktur der gegebenen Webots-Weltdatei, die den Namen “Little Bicycle V2” trägt. Diese Datei definiert eine Szene, in der ein zweirädriger Roboter (ein kleines Fahrradmodell) auf einer rechteckigen Bodenfläche platziert ist. Abbildung 2 zeigt eine mögliche Ansicht der simulierten Szene mit dem Fahrrad auf der Bodenfläche (Bildplatzhalter).

### Welt-Einstellungen und Umgebung

Zu Beginn der Weltdatei werden drei externe PROTO-Typen importiert:  
EXTERNPROTO "RectangleArena.proto"

EXTERNPROTO "TexturedBackground.proto"

EXTERNPROTO "TexturedBackgroundLight.proto"

Damit werden die Definitionen für eine rechteckige Arena (Boden), einen texturierten Himmelshintergrund und die dazugehörige Lichtquelle verfügbar gemacht, die später im Weltfile benutzt werden. Unmittelbar darauf folgt der WorldInfo-Knoten, in dem ein basicTimeStep von 5 Millisekunden gesetzt ist. Dieser sehr kleine Zeitschritt weist darauf hin, dass die Simulation mit hoher Auflösung läuft – vermutlich erforderlich, um die Fahrraddynamik stabil und realitätsnah berechnen zu können. Im WorldInfo ist außerdem ein contactProperties-Eintrag definiert, der spezielle Kollisionsparameter für die Materialien “wheel” (Rad) und “ground” (Boden) festlegt. Konkret wird ein Haftreibungskoeffizient (coulombFriction) von 0,8 angegeben sowie ein geringer Rückprall (bounce) von 0,1. Dies bedeutet, dass die Räder auf dem Boden relativ viel Reibung haben (wichtig für ein Fahrrad, um nicht durchzudrehen oder zu rutschen) und Stöße leicht gedämpft werden.

Darauf folgt ein Viewpoint-Knoten, der die Anfangskamera definiert. Die Orientierung und Position deuten darauf hin, dass die Kamera das Fahrrad schräg von oben hinten betrachtet, wahrscheinlich um das Balancieren und Fahren des Modells gut zu beobachten. Als Nächstes werden die Umgebungsbedingungen gesetzt: ein TexturedBackground (Himmelshintergrund) ohne weitere Parameter – vermutlich mit Standardwerten oder in der PROTO vordefiniert – sowie ein TexturedBackgroundLight, das für die Umgebungsbeleuchtung sorgt. Beide Knoten sind Instanzen der oben importierten PROTO-Typen und sorgen zusammen für einen realistischen Hintergrund (Himmel/Panorama) und gleichmäßiges Licht in der Szene.

Das Bodenobjekt wird durch einen RectangleArena-Knoten erzeugt. Dieser Knoten stammt aus der importierten RectangleArena.proto und repräsentiert eine einfache rechteckige Fläche. In der Datei ist floorSize 10 20 angegeben, was eine Fläche von 10 m × 20 m bedeutet. Die floorAppearance nutzt eine PBRAppearance mit einer Bildtextur (track002.jpg) als Base Color. Das weist darauf hin, dass der Boden z.B. eine Straßen- oder Wege-Textur hat (z.B. ein aufgemalter Parcours), der durch diese Bilddatei definiert ist. Abbildung 3 könnte hier eine Draufsicht auf die Arena mit der Boden-Textur zeigen (Bildplatzhalter). Der contactMaterial des Bodens ist auf “ground” gesetzt, sodass – zusammen mit den Rädern der Fahrrads (“wheel”) – die oben erwähnten speziellen Reibungswerte angewendet werden.

### Roboter “Little Bicycle V2” und Komponenten

Der zentrale Teil der Weltdatei ist der Robot-Knoten mit dem Namen “Little Bicycle V2”. Dieser Knoten definiert das Fahrrad-Modell inklusive aller Unterkomponenten. Die Translation -3.71 -6.9 -0.045 und Rotation (180° um die vertikale Achse) legen die Startposition des Fahrrads auf der Arena fest – wahrscheinlich initial aufrecht stehend und an einer bestimmten Stelle auf dem Boden. Innerhalb des Roboters sind die physischen und visuellen Bestandteile in hierarchischer Form aufgeführt. Die Hauptstruktur des Fahrrads besteht aus mehreren Bauteilen, die über drehbare Gelenke (HingeJoint) miteinander verbunden sind:

• Rahmen: Der Fahrradrahmen ist als starres Basisteil definiert. In der Weltdatei erscheint er als Transform-Knoten (mit DEF frame bezeichnet), der einen CAD-Modell-Shape enthält (frame.obj). Dieser Transform hat eine Skalierung von 0.001 in alle Achsen – vermutlich weil das importierte CAD-Modell in anderen Einheiten (z.B. Millimetern) vorliegt und hier auf Meter skaliert werden muss.

• Hinterrad: Das Hinterrad ist an den Rahmen mit einem HingeJoint gekoppelt (Knoten DEF rear\_wheel). Dieses Gelenk hat einen Ankerpunkt (Vektor anchor 0 0.083 0 relativ zum Elternteil) etwa an der Hinterradnabe und erlaubt Rotation um eine Achse. Das Endpunkt-Objekt des Gelenks ist ein Solid, welches das eigentliche Rad-Modell enthält (r\_wheel.obj) und als Material “wheel” definiert ist. Am Gelenk ist ein Rotationsmotor (RotationalMotor mit Name “motor::wheel”) angebracht, der das Rad antreiben kann. Somit wirkt das Hinterrad wie ein angetriebenes Rad, mit dem das Fahrrad vorwärts bewegt werden kann.

• Tretkurbel und Pedale: Die Tretkurbel ist als separates HingeJoint (DEF crank) am Rahmen angebracht. Dieses Gelenk erlaubt der Kurbel eine Rotation (simuliert das Pedaltreten) und besitzt einen Motor (motor::crank), der die Kurbel drehen kann. An der Kurbel sind wiederum die Pedale befestigt – in der Datei erkennbar als zwei weitere kleine HingeJoint-Knoten innerhalb der Kurbel-Struktur (DEF pedal). Jeder Pedal-Joint hat einen festgelegten Ankerpunkt (±0.0375 Versatz), um das Pedal am Kurbelende zu positionieren, und verbindet ein Solid, das das Pedal-Modell (pedal.obj) enthält. Diese Pedal-Gelenke haben keinen Motor, sind aber vermutlich frei drehbar, so dass die Pedale immer waagerecht bleiben können, während die Kurbel rotiert.

• Lenker und Vorderradgabel: Der vordere Teil des Fahrrads – bestehend aus Lenker, Gabel und Vorderrad – ist über ein weiteres HingeJoint (DEF Handlebars\_and\_Fork) mit dem Rahmen verbunden. Dieses Gelenk ist so ausgerichtet (axis 0 0 1), dass es die Lenkbewegung um die vertikale Achse erlaubt (also das Schwenken des Lenkers nach links/rechts). Ein angebundener Motor (“handlebars motor”) mit begrenztem Drehmoment kann genutzt werden, um aktiv in die Lenkung einzugreifen. Der Endpunkt dieses Gelenks ist ein Solid, das die Gabel und Lenker-Geometrie enthält (fork.obj). An der Gabel wiederum befindet sich ein weiteres HingeJoint für das Vorderrad (DEF wheel innerhalb der Gabel). Dieses Rad ist ähnlich wie das Hinterrad als Solid mit Rad-Modell (f\_wheel.obj) und Material “wheel” definiert und kann um seine Achse rotieren. Das Vorderrad-HingeJoint hat keine aktive Motorisierung (beim Fahrrad wird das Vorderrad ja nicht angetrieben, sondern rollt frei), aber Parameter wie Dämpfung und Friktion sind gesetzt, damit es sich realistisch dreht und stoppt.

Zusätzlich zu diesen mechanischen Komponenten verfügt das Roboter-Modell über einige Sensoren und Geräte. Im Robot-Knoten sehen wir einen Camera-Node, der eine Kamera mit Sichtfeld und Auflösung definiert (hier 480×320 Pixel). Vermutlich ist diese Kamera am Rahmen montiert (die Translation ist sehr klein, praktisch auf dem Rahmen), vielleicht um aus Fahrerperspektive oder zur Stabilisierung zu sehen. Ebenfalls ist ein Display vorhanden – dies könnte ein virtueller Bildschirm am Roboter sein, der für Debugging oder zur Visualisierung genutzt wird (480×320 Auflösung, passend zur Kamera, evtl. um Kamerabilder anzuzeigen). Ein weiteres wichtiges Sensor-Modul ist die InertialUnit (IMU). Diese ist mit translation 0 0 0.05 am Roboternode platziert (z.B. oberhalb des Schwerpunkts) und misst die Orientierung des Roboters im Raum. Die Inertial Unit liefert typischerweise Roll-, Pitch- und Yaw-Werte, was für einen selbstbalancierenden Roboter essentiell ist, um z.B. Neigungswinkel zu erfassen.

Für die physikalische Simulation sind im Robot-Node entsprechende Eigenschaften angegeben. Unter physics ist die Gesamtmasse des Fahrrads auf 3.5 kg gesetzt, mit einem Schwerpunkt leicht oberhalb der Radachse (centerOfMass 0 0 0.05) und einer Trägheitsmatrix, die die Verteilung der Masse widerspiegelt. Außerdem sind für die einzelnen Kollisionskörper (boundingObjects) einfache Formen definiert – meist Boxen oder Zylinder um Rahmenrohre und Räder. Diese vereinfachten Kollisionsgeometrien (im boundingObject-Feld jedes relevanten Solid bzw. einer Gruppe) dienen der Physik-Engine zur schnelleren Kollisionserkennung, anstatt die detaillierten Dreiecksmodelle zu verwenden.

Abschließend sei noch auf zwei besondere Felder des Roboters hingewiesen: controller "balance\_control\_c" und supervisor TRUE. Ersteres bedeutet, dass dieser Roboter beim Start der Simulation das Controller-Programm balance\_control\_c zugewiesen bekommt – dieses Programm (in C geschrieben, dem Namen nach) wird also die Regelung des Gleichgewichts und ggf. das Fahren des Fahrrads übernehmen. Das Flag supervisor TRUE kennzeichnet den Roboter als Supervisor-Roboter. Ein Supervisor in Webots hat erweiterte Rechte und kann mittels seines Controllers auf die Simulation einwirken, z.B. Objekte hinzufügen/entfernen, globale Parameter abfragen oder die Simulation steuern (Cyberbotics 2018). Dies ist typisch, wenn ein Roboter als Hauptkontroller fungiert – hier vermutlich um das selbstbalancierende Fahrrad zu stabilisieren, könnte der Supervisor-Status notwendig sein, um beispielsweise auf Trickfunktionen der Physik-Engine zurückzugreifen oder andere Roboter zu überwachen.

Zusammenfassend ist die Weltdatei sorgfältig strukturiert: Zunächst werden allgemeine Simulationsparameter und Umgebung festgelegt, dann die Arena als Umgebung eingefügt und schließlich der Roboter mit all seinen Teilen und Eigenschaften detailliert beschrieben. Durch die hierarchische Organisation (die auch im Webots Szene-Baum sichtbar ist) wird deutlich, wie die Komponenten zusammenhängen. Das Fahrrad-Modell “Little Bicycle V2” besteht aus mehreren durch Gelenke verbundenen Teilen, was die realistische Bewegung aller Komponenten (Lenkung, Räder, Pedale) in der Simulation ermöglicht. Die Kombination aus physikalischen Parametern, Sensoren und einem speziellen Balance-Controller weist darauf hin, dass es sich um ein komplexes Szenario handelt, in dem das Fahrrad aufrecht gehalten und gesteuert werden muss – ein anspruchsvolles Robotik-Problem, das mit Webots experimentell untersucht wird.

Quellen

• Cyberbotics (2018). Webots User Guide – Introduction & Reference. Verfügbar unter: Cyberbotics Documentation (abgerufen am 26. Juni 2025).

• Cyberbotics (2023). Webots – Open Source Robot Simulator (Offizielle Webseite). Verfügbar unter: https://cyberbotics.com (abgerufen am 26. Juni 2025).

# 4 Webots-Simulationsumgebung

## 4.1 Begründung der Tool-Wahl (Open-Source, ODE-Physik, ROS-Kompatibilität)

## 4.2 Physik-Engine: Massen­trägheit, Reibung, Kontaktmodell

## 4.3 Abbildung der Fahrrad­kinematik

## 4.4 Inter­face zu **Simulink** (S-Function, TCP-Bridge)

# 6 **Kombinierte Regelung**

## 6.1 Portierung des Balance-PID (Zander)

## 6.2 Portierung des Fahrtrichtungs-MPC (Giray)

## 6.3 Supervisions-/Koppelkaskade

## 6.4 Failsafe-Strategien

# 7 Teststrecken & Versuchsdesign

## 7.1 Gerade Strecke mit Störimpuls

## 7.2 Slalom-Kurs

## 7.3 S-Kurve mit Steigung

## 7.4 Kreisfahrt

# 8 **Ergebnisse**

## 8.1 Balance-Stabilität

## 8.2 Spur­treue & Quer­abweichung

## 8.3 Rechenzeit & Echtzeitfähigkeit

# 9 Grenzen der Arbeit

# 10 **Zukünftige Arbeiten** (Sensor-Fusion, Reinforcement Learning, Hardware-in-the-Loop)

# 11 Fazit

# 12 Literaturverzeichnis

# 13 **Anhang** (Parameter­tabellen, Screenshots, Release-Link)