

Objekterkennung mit dem IoT-Bot

Dokumentation

www.siemens.com

SIEMENS

Dauer

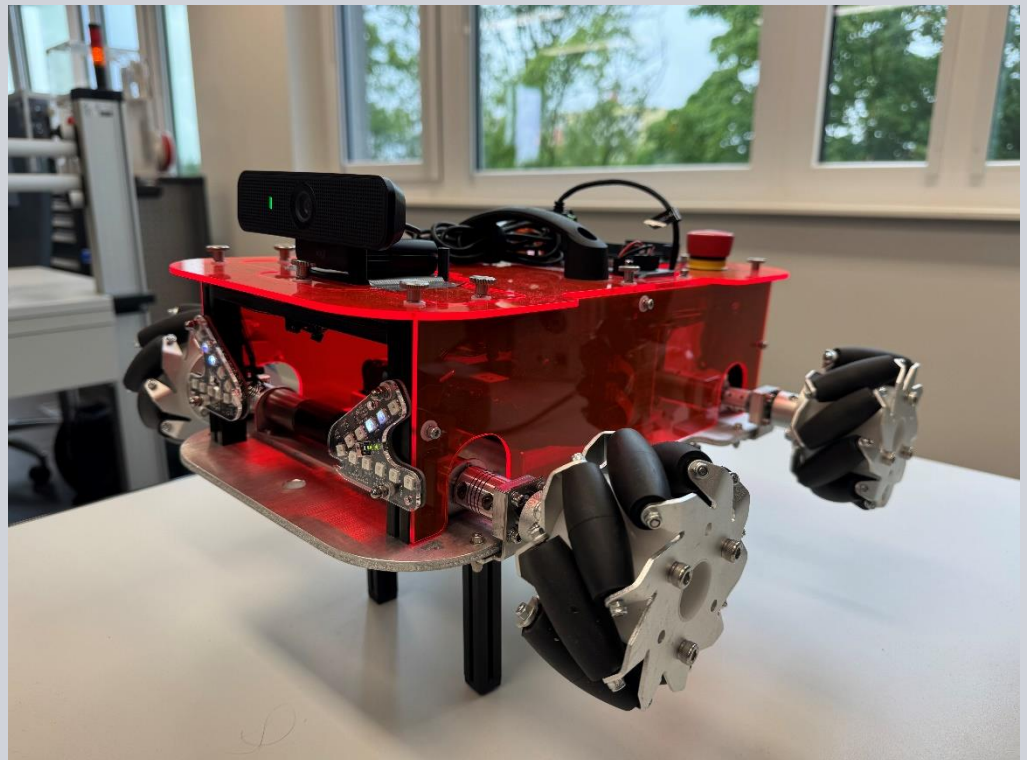
3

Stand

07/2025

Bearbeiter

Manuel Hüls, Fynn Malte Kießler,
Tobias Köhler, Nicolas Stock



Inhaltsverzeichnis

Inhaltsverzeichnis.....	2
1 Projektziel.....	3
2 Kommunikationsschema.....	4
3 Projektdurchführung	5
3.1 Unkompliziert	5
3.2 Herausforderungen	5
4 Reaktion auf Kameradaten:	6
5 Umsetzung in Python:.....	8
6 Datenkommunikation zwischen Node-RED und Python.....	10
7 Projektergebnis.....	13
7.1 Dashboardbasierte Steuerung und Anzeigen:	13
7.2 Objekterkennung:	13
7.3 Dynamische Reaktion basierend auf Kameradaten:.....	13

1 Projektziel



Dashboardbasierte Steuerung und Anzeigen:

- Bewegung des IOT-Bots
- Anzeige verschiedener Daten und des Kamerabilds

Objekterkennung:



- Entwicklung eines Python Skripts zur Erkennung einfarbiger Objekte
- Ausgewertete Datenrückgabe an Nodered
- Anzeige des erkannten Objekts im Dashboard



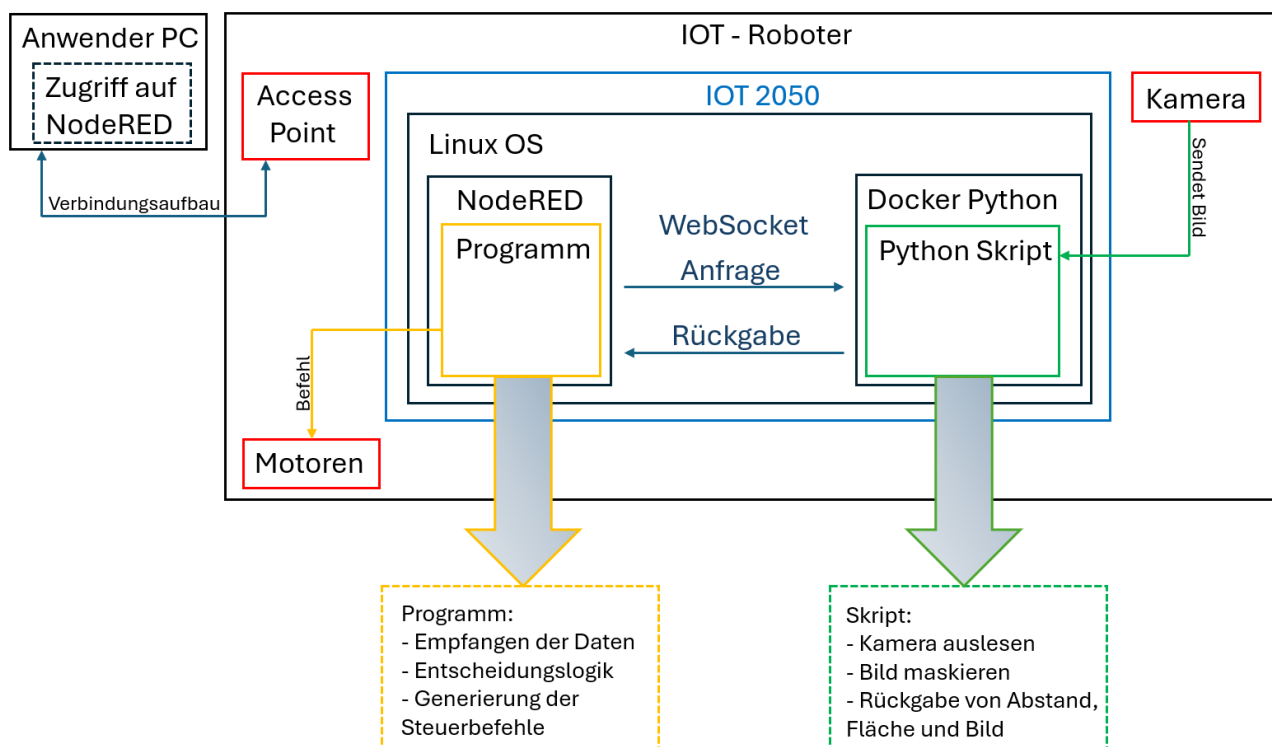
Dynamische Reaktion basierend auf Kameradaten:

- Fahrtrichtungsänderung abgezielt auf rotes Objekt
- Autonomes Verfolgen des Objekts
- Optional: Selbstständiges Fahren auf vorgegebener Linie

Verwendete Links:

<https://chat.siemens.com/chat/5a2a0b68-2511-44e6-91a4-b180d0c86c20>
<https://github.com/tisch017/EduArt-Robotik-Webcam-Pythonskript/tree/main>

2 Kommunikationsschema



3 Projektdurchführung

3.1 Unkompliziert

- Erstellung des Python Skripts
 - ➔ HSV Farbgrenzen konnten leicht ermittelt werden
- Übertragung der Daten vom IOT Roboter an Python mithilfe von WebSocket

Verwendete KI-Tools:

- SiemensGPT
- CoPilot

3.2 Herausforderungen

- Einen geeigneten Kommunikationsmechanismus zwischen Python und Node-RED finden
- Initiale Internetverbindung der IOT 2050
- Node-RED Ansteuerung der Motoren
- Auswertung der empfangenen Kameradaten in Node-RED
- Ingesamter Umgang mit Linux

4 Reaktion auf Kameradaten:

Ablauf der Reaktion auf Kameradaten beim IoT-Roboter

1. Datenerfassung durch die Kamera

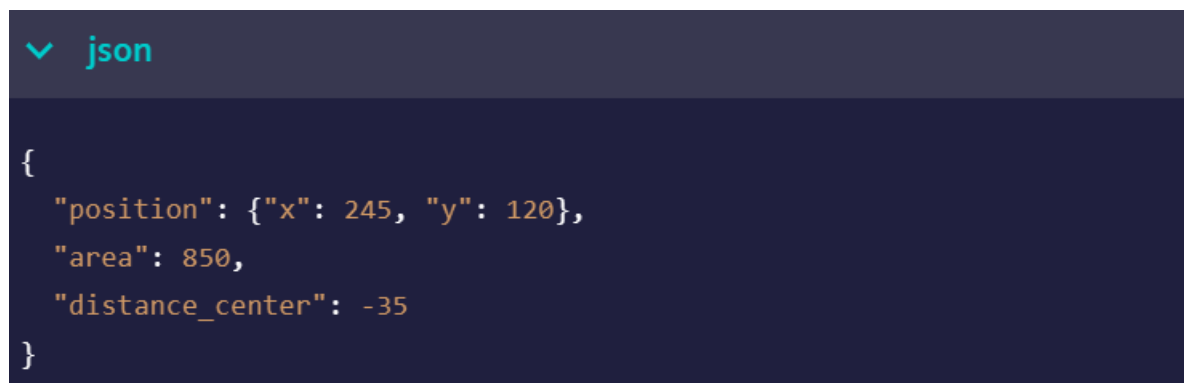
- Der IoT-Roboter erfasst kontinuierlich Kamerabilder.
- Das Python-Skript analysiert das eingehende Videostream-Material Bild für Bild direkt nach gewünschtem Muster oder Zielobjekt (z.B. bestimmte Farben oder Bewegungen).

2. Bildverarbeitung und Auswertung im Python-Skript

- Das Python-Skript nutzt OpenCV zur Verarbeitung der Kameradaten:
 1. Bildumwandlung in das HSV-Farbformat.
 2. Erstellen einer Maske nach den definierten HSV-Farbbereichen.
 3. Identifikation und Lokalisierung des Zielobjektes (Berechnung der Mitte, Position, Flächengröße und Abstand zur Bildmitte).

3. Datenübertragung (WebSocket) vom Python-Skript an Node-RED

- Ergebnisse der Auswertung (z.B. Objektposition, Entfernung zur Bildmitte, Fläche des Objekts) werden als strukturierte Nachricht im JSON-Format übermittelt.
- Kommunikation erfolgt typischerweise via MQTT oder WebSocket:
 - MQTT: Daten werden über einen Broker an Node-RED gesendet und dort empfangen.
 - WebSocket: Direkte permanente Verbindung zwischen Python-Skript und Node-RED ermöglicht Echtzeitübertragung.

A screenshot of a code editor showing a JSON object. The text is as follows:

```
✓ json  
  
{  
  "position": {"x": 245, "y": 120},  
  "area": 850,  
  "distance_center": -35  
}
```

4. Empfangen und Verarbeitung in Node-RED

- Node-RED empfängt Daten über entsprechende Nodes (WebSocket-In-Node).
- Node-RED verarbeitet diese Daten mithilfe von Funktionen oder logischen Nodes, z.B.:
 - Überprüfung der Werte und Einleitung der Aktionen.
 - Entscheidungslogik (z.B. "ist Abstand zur Mitte größer als ein Schwellwert?").

5. Entscheidungen treffen und Steuerdaten generieren

- Aufgrund eingegangener Kameradaten entscheidet Node-RED, ob und welche Aktion der Roboter ausführen soll:
 - z.B. das Objekt zentrieren („Fahr links/rechts“),
 - dem Zielobjekt folgen („Fahr vor/zurück“).
- Node-RED erzeugt entsprechende Steuerbefehle.

6. Senden der Steuerbefehle vom Node-RED an Roboter-Steuersystem zurück

- Ausgewählte Aktionen/Fahrbewegungen werden über dieselbe Kommunikationstechnologie (WebSocket) zurück an den Roboter gesendet.
- Alternativ kann Node-RED über die EXEC-Kommunikation direkt ein weiteres Skript starten und lokal Befehle ausführen.

7. Roboter führt die gewünschte Aktion/Bewegung aus

- Nach Empfang interpretiert das Steuersystem des Roboters den Steuerbefehl und setzt diesen in physische Bewegungen der Aktoren und Motoren um.

5 Umsetzung in Python:

Systemvoraussetzungen

- Python-Installation
- OpenCV (cv2), NumPy, multiprocessing,
- Webcam/Kameramodul
- Systemargumente für Farbbereichsdefinition

Aufgaben des Python-Skripts

1. Farberkennung mittels OpenCV

- Das Skript nutzt eine Webcam zur Erfassung eines kontinuierlichen Videostreams.
- Anhand definierter Farbgrenzen im HSV-Raum (Hue, Saturation, Value) erkennt es gezielt farbliche Objekte innerhalb des Videobilds.
- Es identifiziert die größte Kontur des Farbbereichs und berechnet:
 - Mittelpunkt (X,Y Koordinaten),
 - Fläche der Kontur in Pixel,
 - Abstand zwischen der horizontalen Bildmitte und dem Mittelpunkt des Objekts.

Die Ergebnisse werden unmittelbar visualisiert und in Form eines annotierten Bildes dargestellt.

2. Echtzeit-Kommunikation per Websocket

- Ein Websocket-Server läuft parallel zur Bildverarbeitung innerhalb desselben Python-Skripts.
- Dieser Server ermöglicht eine bidirektionale Echtzeit-Kommunikation zwischen Python und externen Anwendungen (beispielsweise Node-RED).

Kommunikationsvorgang:

- Die externe Anwendung schickt per Websocket Anfragen im JSON-Format an den Python-Server:
 - Anfrage zur Abfrage aktueller Messwerte (Distanz, Fläche, Bild).
 - Anfrage zum Aktualisieren der Parameter für HSV-Farbbereiche und des Flächenschwellwertes.
- Der Python-Websocket-Server reagiert wie folgt:
 - Antwortet auf Abfragen mit aktuellen Messwerten und einem codierten Bild im Base64-Format.
 - Übernimmt neue Konfigurationsparameter dynamisch und bestätigt deren erfolgreiche Übernahme.

Technische Umsetzung

Zur technischen Umsetzung kommen die folgenden Python-Bibliotheken zum Einsatz:

- **OpenCV (cv2)** → Bildverarbeitung, Konturenfindung, Markierung und Visualisierung.
- **NumPy (numpy)** → effiziente Array-Operationen und Darstellung numerischer Daten.

- **Websockets (websockets) + asyncio** → asynchrone Echtzeitkommunikation auf Websocket-Basis.
- **Multiprocessing (multiprocessing)** → parallele Ausführung zweier Hauptkomponenten: Bildverarbeitung und Kommunikation.

Durch das Python-Datenformat **JSON** erfolgt die strukturierte Kommunikation klar definiert und gut strukturiert.

Datenstruktur der Kommunikation

<p>Anfrage zum Erhalten von aktuellen Werten</p> <pre> ▼ json { "request": "get_distance" } </pre>	<p>Antwort des Python-Skripts</p> <pre> ▼ json { "distance_to_center": 15.0, "area": 1024, "image": "<Base64-codiertes Bild>" } </pre>
<p>Anfrage zur Parameteränderung</p> <pre> ▼ json { "request": "set_params", "h_min": 100, "s_min": 150, "v_min": 100, "h_max": 140, "s_max": 255, "v_max": 255, "area_thres": 700 } </pre>	<p>Antwort des Python-Skripts</p> <pre> ▼ json { "status": "parameters_updated" } </pre>

Zusammenfassung der Vorteile und Nutzen

- **Flexibilität:** Adaptierung der Erkennungsparameter über die externe Schnittstelle ohne Unterbrechung des laufenden Prozesses.
- **Echtzeitkommunikation:** Sofortige Bereitstellung aktueller Mess- und Bildinformationen zur Einbindung in Steuer- und Visualisierungssysteme (z.B. Dashboard).
- **Strukturierte Datenaustauschformate:** Kommunikation via JSON erleichtert Integration in verschiedenste Systeme und Applikationen.

Dieses System bietet somit eine robuste, flexible und unmittelbare Einbindung von Python-gesteuerten Bildverarbeitungsprozessen in externe Steuerungs- und Informationssysteme mittels Websocket-Kommunikation.

6 Datenkommunikation zwischen Node-RED und Python

1. HTTP (Hypertext transform protocol)

- **Grundprinzip:**
 - Client-Server-Prinzip: Client sendet eine Anfrage (Request) an den Server, der Server antwortet (Response).
- **Eigenschaften:**
 - Verbindung: synchron, kurzfristig, Anfrage/Antwort basiert
 - Protokoll: zustandslos (jede Anfrage ist unabhängig von anderen)
 - Nachrichtenformat: Textbasiert (z.B.: JSON, XML, HTML)
- **Typische Anwendungsszenarien:**
 - Einmalige Datenübertragungen zwischen IoT-Gerät und Server, Statusupdates
- **Vorteile:**
 - Einfach implementierbar
 - Weit verbreitet, kompatibel mit praktisch allen Umgebungen und Plattformen
- **Nachteile:**
 - Relativ hoher Overhead (z.B.: Header, TCP-Verbindung)
 - Nicht optimal für Echtzeit-Kommunikation oder häufige Statusupdates
 - Mehr Ressourcenverbrauch (besonders bei häufigen Verbindungen)

2. MQTT (Message Queuing Telemetry Transport)

- **Grundprinzip:**
 - Publish/Subscribe-Modell: Geräte (Clients) können Nachrichten unter bestimmten "Topics" senden (Publish) oder abonnieren (Subscribe).
 - Ein zentraler Server (Broker) verwaltet die Nachrichten und verteilt sie an alle abonnierenden Clients.
- **Eigenschaften:**
 - Verbindung: persistent, mit Broker verbunden, asynchron
 - Protokoll: leichtgewichtig, binär
 - Nachrichtenformat: binär, oft JSON in Payload
- **Typische Anwendungsszenarien:**
 - Echtzeit-Datenübertragungen
 - Zustandsmeldungen von Sensoren, IoT-Geräten
 - Steuerungsbefehle in Echtzeitanwendungen (z.B.: Home Automation, Industrie 4.0)
- **Vorteile:**
 - Geringer Bandbreitenverbrauch, ideal für ressourcenarme Umgebungen

- Effiziente und schnelle Kommunikation durch Publish/Subscribe
- Hohe Skalierbarkeit und einfache Wartung
- **Nachteile:**
 - Benötigt zentralen MQTT-Broker
 - Geringfügig komplexer bei der ersten Implementierung im Vergleich zu HTTP

3. EXEC (Ausführung von Befehlen)

- **Grundprinzip:**
 - Direkte Ausführung von Systembefehlen oder Programmen über ein Skript oder eine Schnittstelle (z.B.: Node-RED exec-Node), um direkt auf Systemebene zu agieren.
- **Eigenschaften:**
 - Direkte Schnittstelle zum Betriebssystem
 - Keine spezifische Netzwerkkommunikation, sondern lokale Ausführung auf einem Gerät
 - Führt Programme/Scripts direkt im Betriebssystem aus
- **Typische Anwendungsszenarien:**
 - Starten von lokalen Skripten
 - Lokale Steuerung und Überwachung von Hardware-Komponenten (z.B.: GPIO Pins steuern, lokale Datenverarbeitung, Systemstatus abfragen)
 - Integration von lokal laufenden Programmen in größere Workflows
- **Vorteile:**
 - Maximal flexibel, da jeder unterstützte Kommandozeilenbefehl verwendet werden kann
 - Ideal zur lokalen Systemintegration und Steuerung
- **Nachteile:**
 - Sicherheitsrisiko durch direkte Systemzugriffe (potenziell gefährlich bei unkontrollierten Eingaben)
 - Nur für lokale Aktionen geeignet, keine direkte Remote-Kommunikation wie bei HTTP oder MQTT

4. WebSocket (Bidirektionale Kommunikation)

Funktionsweise:

- WebSocket stellt eine bidirektionale, persistente Kommunikationsverbindung her.
- Node-RED und Python-Skript sind dauerhaft miteinander verbunden und tauschen kontinuierlich Nachrichten aus, ohne jede Nachricht separat anzufragen.

Beispielhafte Anwendung:

- Das Python-Skript sendet kontinuierlich Echtzeit-Daten der Kamera-Erfassung über WebSocket direkt an Node-RED.
- Node-RED sendet Steuerbefehle oder Einstellungen unmittelbar zurück an das Python-Skript in der gleichen Verbindung.

Vorteile:

- Schnelle und persistente bidirektionale Kommunikation.
- Ideal für Echtzeitdaten und interaktive Steuerung.
- Geringe Latenz und wenig Kommunikations-Overhead.

Nachteile:

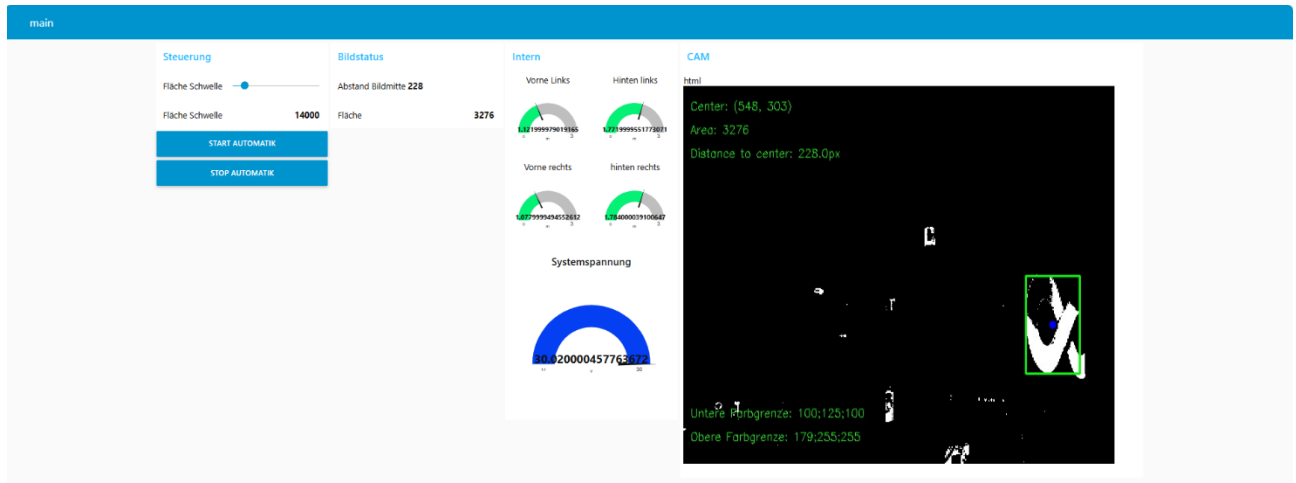
- Geringfügig höherer Initialaufwand zur Implementierung.
- Persistente Verbindung benötigt Ressourcen auf Client- und Serverseite.

➔ Im weiteren Verlauf wurde auf die Implementierung des WebSockets zurückgegriffen, um eine potenzielle Bidirektionale Kommunikation zu ermöglichen.

7 Projektergebnis

7.1 Dashboardbasierte Steuerung und Anzeigen:

- Bewegung des IOT-Bots ✓ umgesetzt
- Anzeige verschiedener Daten und des Kamerabilds ✓ umgesetzt



7.2 Objekterkennung:

- Entwicklung eines Python Skripts zur Erkennung einfarbiger Objekte ✓ umgesetzt
- Ausgewertete Datenrückgabe an Nodered ✓ umgesetzt
- Anzeige des erkannten Objekts im Dashboard ✓ umgesetzt

7.3 Dynamische Reaktion basierend auf Kameradaten:

- Fahrtrichtungsänderung abgezielt auf rotes Objekt ✓ umgesetzt
- Autonomes Verfolgen des Objekts ✓ umgesetzt
- Optional: Selbstständiges Fahren auf vorgegebener Linie ✓ umgesetzt

7.4 Endergebnis Python Programm

