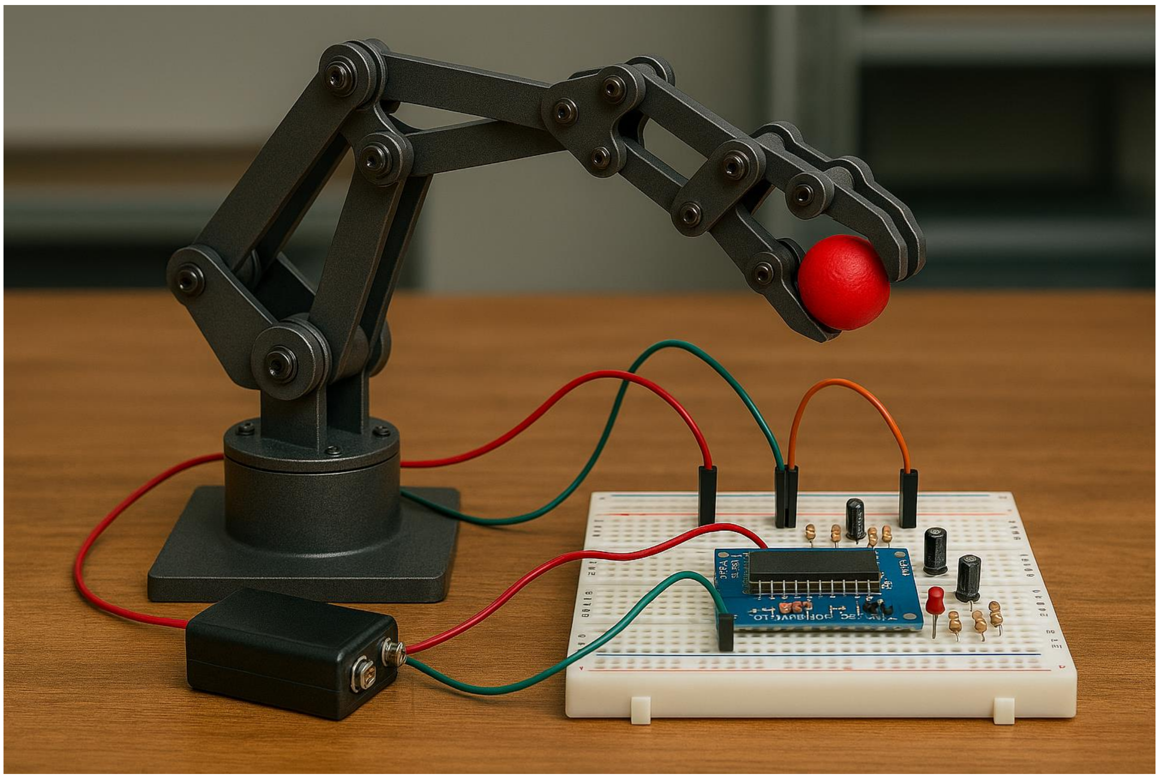
Detailed Technical Report: Hydraulic Robotic Arm

Designer: Engineer Mahmoud Ali Hassan Entwickler: Ingenieur Mahmoud Ali Hassan

Design Date: April 18, 2018 Entwurfsdatum: 18. April 2018



Executive Summary

This report presents a comprehensive design of a hydraulic robotic arm with a triple-finger gripper and three stepper-motor axes. It covers mechanical architecture, materials, manufacturing processes, control electronics, software toolchain, mathematical modeling (kinematics & dynamics), control algorithms, and full firmware for both Arduino Mega and PIC18F controllers.

Dieser Bericht präsentiert ein umfassendes Design eines hydraulischen Roboterarms mit einem Dreifinger-Greifer und drei Schrittmotorachsen. Er behandelt die mechanische Architektur, die Materialien, Fertigungsprozesse, Steuerungselektronik, Software-Toolchain, mathematische Modellierung (Kinematik & Dynamik), Steuerungsalgorithmen und die vollständige Firmware sowohl für Arduino Mega als auch PIC18F-Controller.

System Overview

The hydraulic robotic arm is designed to perform precise pick-and-place operations of objects up to 1 kg in industrial, laboratory, and hazardous environments. It offers three rotational degrees of freedom (base rotation, shoulder and elbow pitch) plus a hydraulic linear actuator for vertical motion. Integrated sensors and wireless modules enable closed-loop control and IoT connectivity.

Der hydraulische Roboterarm ist ausgelegt für präzise Auf-und-Ablage-Bewegungen von Objekten bis 1 kg in industriellen, Labor- und Gefahrenumgebungen. Er verfügt über drei rotatorische Freiheitsgrade (Basisrotation, Schulter- und Ellenbogenwinkel) sowie einen hydraulischen Linearantrieb für vertikale Bewegungen. Integrierte Sensoren und drahtlose Module ermöglichen geschlossene Regelkreise und IoT-Konnektivität.

Mechanical & Hydraulic Architecture

The structural frame and links are machined from 6061-T6 anodized aluminum for a high strength-to-weight ratio. Bearings at each joint minimize friction. Link lengths are 200 mm (base to shoulder), 180 mm (shoulder to elbow), and 70 mm (elbow to wrist). A 20 mm-bore, 200 mm-stroke hydraulic cylinder rated to 10 MPa provides smooth lift. The gripper comprises three nylon-reinforced polymer fingers, each actuated by a micro-servo with a 15 N load cell for force feedback.

Der Trag- und Verbindungsrahmen besteht aus bearbeitetem 6061-T6 eloxiertem Aluminium für ein hohes Verhältnis von Festigkeit zu Gewicht. In allen Gelenken sorgen Lager für minimale Reibung. Die Längen der Gelenkverbindungen betragen 200 mm (Basis bis Schulter), 180 mm (Schulter bis Ellenbogen) und 70 mm (Ellenbogen bis Handgelenk). Ein Hydraulikzylinder mit 20 mm Hubdurchmesser, 200 mm Hub und 10 MPa Nenndruck ermöglicht sanfte Hebebewegungen. Der Greifer besteht aus drei fingerförmigen Nylon-Polymer-Elementen, die jeweils von einem Mikro-Servo mit 15 N Kraftsensor angesteuert werden.

Electronics & Control Hardware

Control electronics are implemented in two variants: Arduino Mega 2560 for rapid prototyping and PIC18F4550 for embedded, low-power operation. Stepper motors (NEMA 17) use A4988 drivers in 1/16 microstepping. A PWM-to-4…20 mA converter drives the proportional hydraulic valve. Sensors include HC-SR04 ultrasonic for height feedback, MPX5700DP pressure transducer, LM35 temperature sensor, and angular potentiometers. Wireless connectivity is provided by HC-05 Bluetooth and ESP8266 Wi-Fi modules with MQTT support. Power is supplied by a 12 V battery pack and regulated to 5 V and 3.3 V via LM2596 modules.

Die Steuerungselektronik liegt in zwei Varianten vor: Arduino Mega 2560 für schnelles Prototyping und PIC18F4550 für eingebetteten, stromsparenden Betrieb. Schrittmotoren

(NEMA 17) werden über A4988-Treiber mit 1/16-Mikrostepping angesteuert. Ein PWM-zu-4…20 mA-Wandler steuert das proportionale Hydraulikventil. Sensoren umfassen HC-SR04-Ultraschall für Höhenrückmeldung, MPX5700DP-Drucksensor, LM35-Temperatursensor und Winkelpotentiometer. Drahtlos ist der Arm mit HC-05 Bluetooth- und ESP8266 Wi-Fi-Modulen mit MQTT-Unterstützung verbunden. Die Stromversorgung erfolgt über ein 12 V-Batteriepacks und Spannungskonverter (LM2596) für 5 V und 3,3 V.

Software & Manufacturing Toolchain

Mechanical CAD is done in SolidWorks 2017, with FEM stress analysis in ANSYS Mechanical. CNC toolpaths are generated in Mastercam. PCB schematics and routing are created in Proteus 8. Firmware development uses Arduino IDE 1.8.5 for Mega and MPLAB X v3.35 with XC8 for PIC. Kinematic and dynamic simulations run in MATLAB R2017b/Simulink. An MQTT broker (Mosquitto) on a Raspberry Pi manages IoT data exchange.

Die mechanische CAD-Konstruktion erfolgt in SolidWorks 2017, die FEM-Spannungsanalyse in ANSYS Mechanical. CNC-Werkzeugwege werden in Mastercam generiert. Schaltpläne und Leiterplattenlayouts entstehen in Proteus 8. Die Firmware-Entwicklung erfolgt in der Arduino IDE 1.8.5 für Mega und MPLAB X v3.35 mit XC8 für PIC. Kinematik- und Dynamiksimulationen laufen in MATLAB R2017b/Simulink. Ein Mosquitto-MQTT-Broker auf einem Raspberry Pi verwaltet den IoT-Datenaustausch.

Mathematical Modeling

Kinematics Denavit–Hartenberg parameters define link frames:

Link	ai (mm)	αi (°)	di (mm)	θi
1	0	90	50	θ_1 (base)
2	200	0	0	θ_2 (shoulder)
3	180	0	0	θ_3 (elbow)

Forward kinematics: $$T^0_3 = A_1(\theta_1)\,A_2(\theta_2)\,A_3(\theta_3)$$

Inverse kinematics (geometric): $$\theta_2 = \arccos\!\bigg(\frac{x^2 + y^2 + z^2 - l_1^2 - l_2^2}{2\,l_1\,l_2}\bigg),\quad \theta_3 = \arctan2\!\Big(\sqrt{1 - K^2},\,K\Big)$$ where K=x2+y2+z2−l12−l222 l1 l2K = \frac{x^2 + y^2 + z^2 - l_1^2 - l_2^2}{2\,l_1\,l_2}.

Dynamics Euler–Lagrange formulation: $$\tau = M(\theta)\,\ddot\theta + C(\theta,\dot\theta)\,\dot\theta + G(\theta)$$ with MM the inertia matrix, CC Coriolis/centrifugal terms, and GG gravity vector.

Hydraulic Actuator Piston force: $$F = P \times A = P \times \frac{\pi D^2}{4}$$ Cylinder velocity: $$v = \frac{Q}{A}$$ Fluid power: $$P_\text{hyd} = P \times Q$$

Mathematische Modellierung

Kinematik Denavit–Hartenberg-Parameter definieren die Gelenkrahmen:

Verbindung	ai (mm)	αi (°)	di (mm)	θi
1	0	90	50	θ_1 (Basis)
2	200	0	0	θ_2 (Schulter)
3	180	0	0	θ_3 (Ellenbogen)

Vorwärtskinematik: $$T^0_3 = A_1(\theta_1)\,A_2(\theta_2)\,A_3(\theta_3)$$

Inverse Kinematik (geometrisch): $$\theta_2 = \arccos\!\bigg(\frac{x^2 + y^2 + z^2 - l_1^2 - l_2^2}{2\,l_1\,l_2}\bigg),\quad \theta_3 = \arctan2\!\Big(\sqrt{1 - K^2},\,K\Big)$$ mit K=x2+y2+z2−l12−l222 l1 l2K = \frac{x^2 + y^2 + z^2 - l_1^2 - l_2^2}{2\,l_1\,l_2}.

Dynamik Euler–Lagrange-Formulierung: $$\tau = M(\theta)\,\ddot\theta + C(\theta,\dot\theta)\,\dot\theta + G(\theta)$$ wobei MM die Trägheitsmatrix, CC die Coriolis-/Zentrifugalkräfte und GG die Gravitationsterm darstellt.

Hydraulikantrieb Kraft am Kolben: $$F = P \times A = P \times \frac{\pi D^2}{4}$$ Zylindergeschwindigkeit: $$v = \frac{Q}{A}$$ Hydraulische Leistung: $$P_\text{hyd} = P \times Q$$

Firmware Code Excerpts

CPP Code

```
// Arduino Mega Sketch (excerpt)

#include <AccelStepper.h>

#include <Servo.h>

#include <PubSubClient.h>
```

```
#define TRIG_PIN 7
#define ECHO_PIN 8
#define PRESS_PIN A0

AccelStepper step1(AccelStepper::DRIVER, 2, 5);
// ... setup Wi-Fi, MQTT, PID controllers ...
void setup() {
  Serial.begin(115200);
  step1.setMaxSpeed(1000);
  // Wi-Fi & MQTT initialization
}
void loop() {
  // Sensor readings (ultrasonic, pressure, temp)
  // PID control loops for steppers, gripper, cylinder
  // Publish/subscribe via MQTT
}
```

C Code

```
/* PIC18F4550 XC8 (excerpt) */
#include <xc.h>
#include "mcc_generated_files/mcc.h"
#define _XTAL_FREQ 8000000

void main(void) {
   SYSTEM_Initialize();
   while (1) {
      // Read ultrasonic sensor
      // Execute PID control
```

```
        // Communicate via UART-Wi-Fi module
    }
}
```

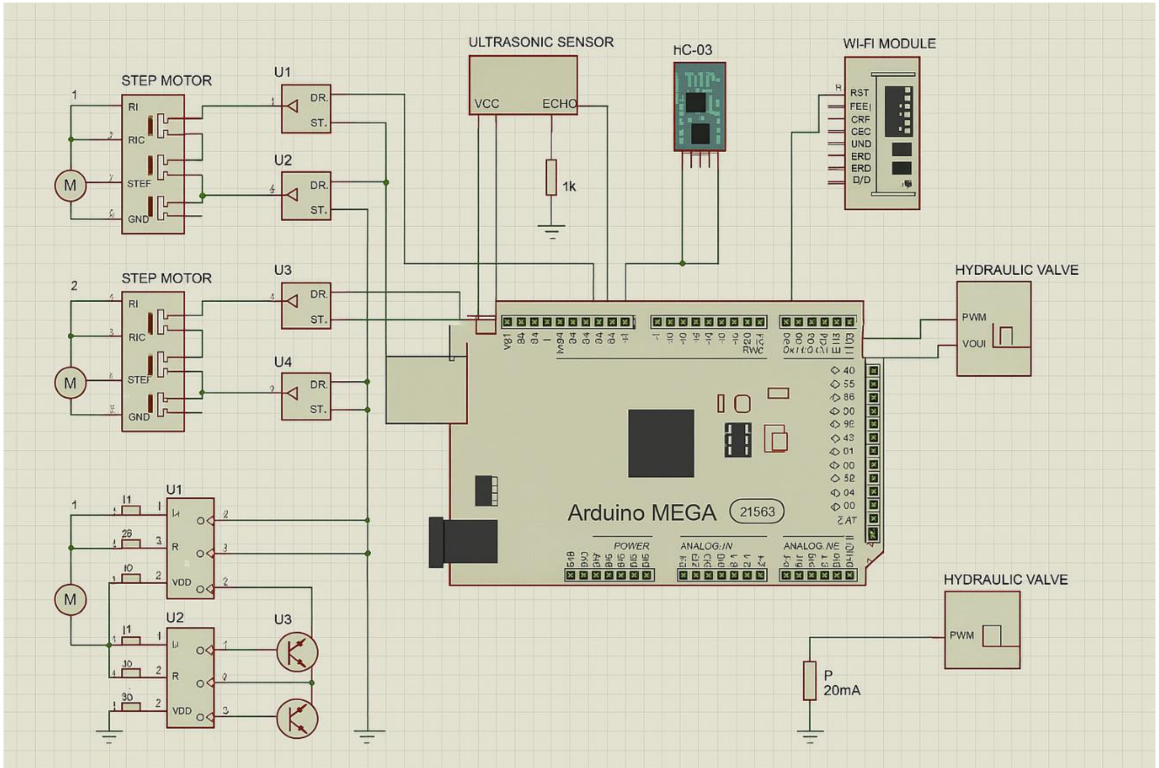
2.1 Circuit Overview

- Controller: Arduino Mega 2560
- Motor drivers: Three A4988 modules for stepper motors
- Hydraulic valve driver: PWM-to-4–20 mA converter
- Sensors: HC-SR04 ultrasonic (TRIG/ECHO), MPX5700DP pressure (analog), LM35 temperature (analog), potentiometers for joint angles
- Wireless: HC-05 Bluetooth, ESP8266 Wi-Fi module

2.2 Proteus Diagram

Description:

- Arduino Mega placed centrally, digital pins 2–7 assigned to stepper driver STEP and DIR inputs.
- PWM pin 9 driving valve driver module.
- Analog pins A0–A3 connected to pressure, temp, and potentiometer sensors.
- TRIG to digital pin 8, ECHO to digital pin 10 for ultrasonic.
- HC-05 connected to Serial1 (TX1/RX1).
- ESP8266 on separate UART (Serial2 TX2/RX2) with power regulator.



Below is a full description and wiring summary . This “schematic” section covers component placement, library parts, and pin-to-pin connections

2.1 Components & Proteus Libraries

Ref.	Component	Proteus Library Part	Qty
U1	Arduino Mega 2560	“Arduino MEGA 2560 R3”	1
U2	A4988 Stepper Driver	“A4988”	3
U3	PWM→4–20 mA Converter Module	“DAC/Current Source”	1
U4	HC-SR04 Ultrasonic Sensor	“HC-SR04”	1
U5	MPX5700DP Pressure Transducer	“Pressure Sensor”	1
U6	LM35 Temperature Sensor	“Temperature Sensor”	1

Ref.	Component	Proteus Library Part	Qty
U7	HC-05 Bluetooth Module	“HC-05”	1
U8	ESP8266 Wi-Fi Module	“ESP-01”	1
J1	12 V Power Jack	“DC_Power_Jack”	1
J2	5 V Regulator (LM2596 module)	“DC–DC_Module”	1
J3	3.3 V Regulator (LM2596 module)	“DC–DC_Module”	1
J4	Header, Stepper Motors (4-pin)	“HEADER_PIN”	3
J5	Header, Gripper Servos (3-pin)	“HEADER_PIN”	3

2.2 Pin-to-Pin Wiring Table

Mega Pin	Signal	Connected To	Notes
5V (power)	5 V rail	J2 Output, LM2596 5 V regulator	Powers sensors & logic
3.3V	3.3 V rail	J3 Output, LM2596 3.3 V regulator	Powers ESP8266
GND	GND	All modules GND pins	Common ground
D2	STEP1	U2(1).STEP	Stepper 1
D3	DIR1	U2(1).DIR	Stepper 1
D4	STEP2	U2(2).STEP	Stepper 2
D5	DIR2	U2(2).DIR	Stepper 2
D6	STEP3	U2(3).STEP	Stepper 3
D7	DIR3	U2(3).DIR	Stepper 3
D9 (PWM)	VALVE_PWM	U3.IN	Hydraulic valve driver input
D8	TRIG	U4.TRIG	Ultrasonic trigger
D10	ECHO	U4.ECHO	Ultrasonic echo
A0	PRESSURE	U5.OUT	Pressure transducer output

Mega Pin	Signal	Connected To	Notes
A1	TEMPERATURE	U6.OUT	LM35 output
A2	POT_SHOULDER	Potentiometer shoulder (divider)	Angle feedback
A3	POT_ELBOW	Potentiometer elbow (divider)	Angle feedback
TX1/RX1	BT_SERIAL	U7.RX / U7.TX	HC-05 Bluetooth
TX2/RX2	WIFI_SERIAL	U8.RX / U8.TX	ESP8266 Wi-Fi
VIN	12 V rail	J1 (+), J1 (–) to GND	Main battery input

Step 3: Proteus Schematic (PIC18F46K22)

3.1 Components & Proteus Libraries

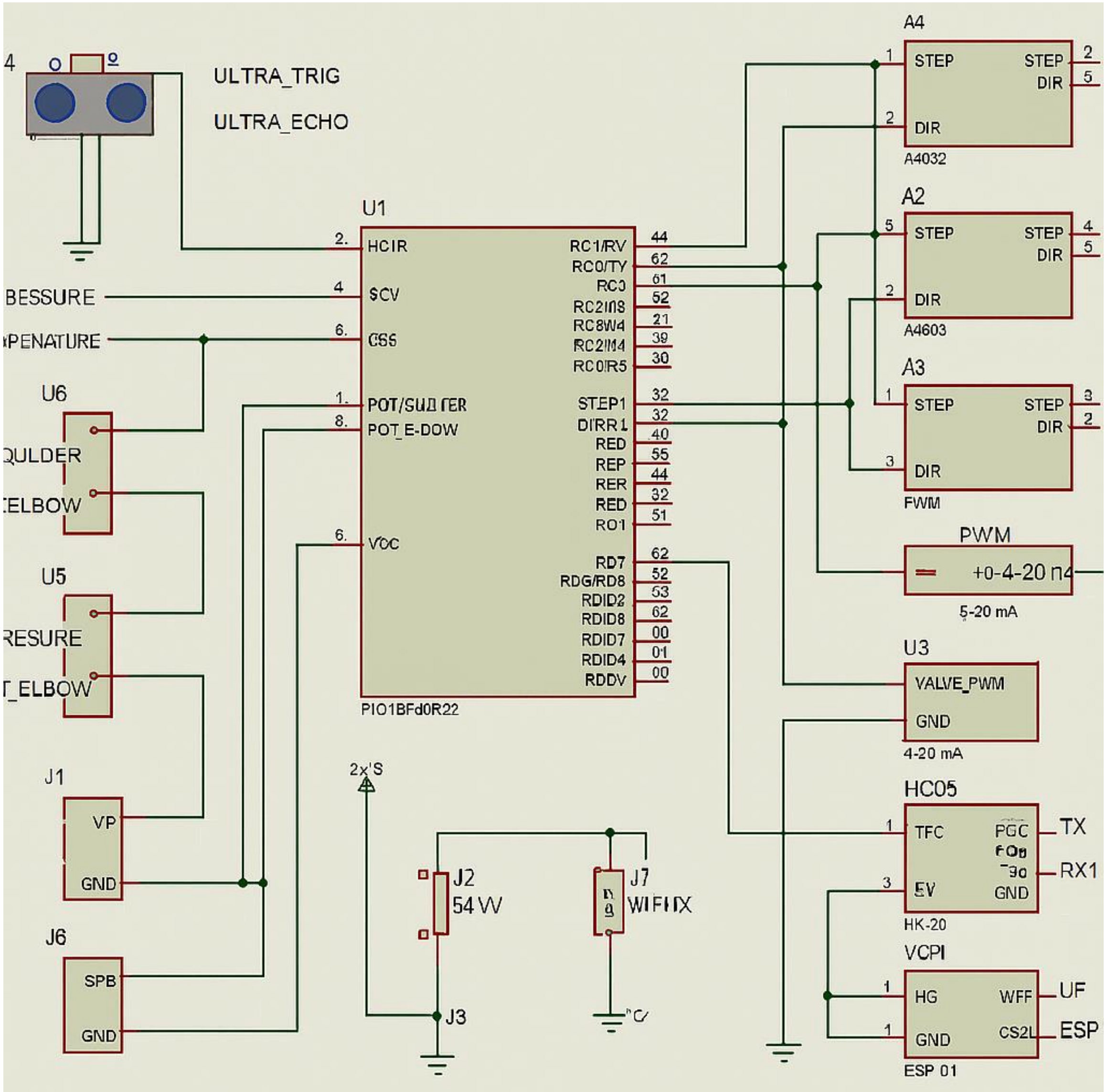
Ref	Component	Proteus Library Part	Qty
U1	PIC18F46K22 (DIP-44)	“PIC18F46K22”	1
U2	A4988 Stepper Driver	“A4988”	3
U3	PWM → 4–20 mA Converter Module	“DAC/Current Source”	1
U4	HC-SR04 Ultrasonic Sensor	“HC-SR04”	1
U5	MPX5700DP Pressure Transducer	“Pressure Sensor”	1
U6	LM35 Temperature Sensor	“Temperature Sensor”	1
U7	HC-05 Bluetooth Module	“HC-05”	1
U8	ESP8266 Wi-Fi Module (ESP-01)	“ESP-01”	1
J1	12 V Power Jack	“DC_Power_Jack”	1
J2	5 V Regulator (LM2596 module)	“DC–DC_Module”	1
J3	3.3 V Regulator (LM2596 module)	“DC–DC_Module”	1

Ref	Component	Proteus Library Part	Qty
J4	Header, Stepper Motors (4-pin)	“HEADER_PIN”	3
J5	Header, Gripper Servos (3-pin)	“HEADER_PIN”	3
J6	ICSP Programmer Header (6-pin)	“HEADER_PIN”	1

3.2 Pin-to-Pin Wiring Table

PIC Pin	Signal	Connected To	Notes
VDD	5 V rail	J2 Output (5 V regulator)	Core & I/O supply
VSS	GND	All modules GND	Common ground
MCLR/VPP	Reset	10 kΩ → 5 V, ICSP pin 4	Pull-up reset
OSC1/OSC2	20 MHz Crystal	20 MHz XTAL + 22 pF caps to GND	External clock
RA0/AN0	PRESSURE	U5.OUT	MPX5700DP output
RA1/AN1	TEMPERATURE	U6.OUT	LM35 output
RA2/AN2	POT_SHOULDER	Potentiometer wiper (shoulder)	Angle feedback
RA3/AN3	POT_ELBOW	Potentiometer wiper (elbow)	Angle feedback
RD0	STEP1	U2(1).STEP	A4988 step1
RD1	DIR1	U2(1).DIR	A4988 dir1
RD2	STEP2	U2(2).STEP	A4988 step2
RD3	DIR2	U2(2).DIR	A4988 dir2
RD4	STEP3	U2(3).STEP	A4988 step3
RD5	DIR3	U2(3).DIR	A4988 dir3
RC2/CCP1	VALVE_PWM	U3.IN	PWM → 4–20 mA converter input
RC3	ULTRA_TRIG	U4.TRIG	HC-SR04 trigger

PIC Pin	Signal	Connected To	Notes
RC4	ULTRA_ECHO	U4.ECHO	HC-SR04 echo
RD6	GRIP_SERVO1	J5(1).Signal	Servo 1
RD7	GRIP_SERVO2	J5(2).Signal	Servo 2
RC0	GRIP_SERVO3	J5(3).Signal	Servo 3
RC6/TX1	BT_TX	U7.RX	HC-05 RX
RC7/RX1	BT_RX	U7.TX	HC-05 TX
RB6/TX2	WIFI_TX	U8.RX	ESP-01 RX
RB7/RX2	WIFI_RX	U8.TX	ESP-01 TX
VIN	12 V rail	J1 (+) → J2/J3 input	Main power



Step 4: Host Dashboard & Control Interface

4.1 Overview

Create a real-time dashboard that subscribes to your PIC-driven arm’s MQTT telemetry and publishes control commands. The interface will:

- Display live sensor data (distance, pressure, temperature, joint angles)

- Offer controls (sliders, buttons) for setpoints (shoulder, elbow, gripper, valve PWM)
- Log data for tuning and analysis

Schritt 4: Host-Dashboard und Steuerungsoberfläche

4.1 Übersicht

Erstellen Sie ein Echtzeit-Dashboard, das die MQTT-Telemetrie Ihres PIC-gesteuerten Arms abonniert und Steuerungsbefehle veröffentlicht. Die Oberfläche bietet folgende Funktionen:

Anzeige von Live-Sensordaten (Entfernung, Druck, Temperatur, Gelenkwinkel)

Bereitstellung von Steuerelementen (Schieberegler, Schaltflächen) für Sollwerte (Schulter, Ellbogen, Greifer, Ventil-PWM)

Protokollierung von Daten für Optimierung und Analyse

4.2 Technology Stack

- MQTT Broker: Mosquitto (local or cloud)
- Front-End: Python + Dash (web) or PyQt5 (desktop)
- MQTT Client: paho-mqtt

4.3 MQTT Broker Setup

1. Install Mosquitto

sudo apt update

sudo apt install mosquitto mosquitto-clients

- Enable persistence & listeners in `/etc/mosquitto/mosquitto.conf`.
- Start & enable service

```
sudo systemctl enable mosquitto
sudo systemctl start mosquito
```

Test

```
mosquitto_sub -h localhost -t "robot_arm/telemetry"
mosquitto_pub -h localhost -t "robot_arm/telemetry" -m '{"test":1}'
```

4.4 Python + Dash Dashboard

4.4.1 File Structure

```
dashboard/
├── app.py
├── mqtt_client.py
└── assets/
    └── style.css
```

4.4.2 mqtt_client.py

```
import json
import threading
import paho.mqtt.client as mqtt


telemetry = {}
commands  = {}


def on_connect(client, _, __, rc):
    client.subscribe("robot_arm/telemetry")
```

```
    client.subscribe("robot_arm/commands_ack")

def on_message(client, _, msg):
    topic = msg.topic
    payload = msg.payload.decode()
    if topic == "robot_arm/telemetry":
        try:
            data = json.loads(payload)
            telemetry.update(data)
        except json.JSONDecodeError:
            pass
    elif topic == "robot_arm/commands_ack":
        commands['ack'] = payload

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

def start_mqtt():
    client.connect("localhost", 1883, 60)
    client.loop_forever()

# publish commands from dashboard
def send_command(cmd_payload):
```

```
    client.publish("robot_arm/commands", json.dumps(cmd_payload))
import json
import threading
import paho.mqtt.client as mqtt

telemetry = {}
commands  = {}

def on_connect(client, _, __, rc):
    client.subscribe("robot_arm/telemetry")
    client.subscribe("robot_arm/commands_ack")

def on_message(client, _, msg):
    topic = msg.topic
    payload = msg.payload.decode()
    if topic == "robot_arm/telemetry":
        try:
            data = json.loads(payload)
            telemetry.update(data)
        except json.JSONDecodeError:
            pass
    elif topic == "robot_arm/commands_ack":
        commands['ack'] = payload
```

```python
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

def start_mqtt():
    client.connect("localhost", 1883, 60)
    client.loop_forever()

# publish commands from dashboard
def send_command(cmd_payload):
    client.publish("robot_arm/commands", json.dumps(cmd_payload))
```

4.4.3 app.py

```python
import dash
from dash import html, dcc
from dash.dependencies import Input, Output, State
import plotly.graph_objs as go
import time
from threading import Thread
import mqtt_client

# Start MQTT thread
Thread(target=mqtt_client.start_mqtt, daemon=True).start()

app = dash.Dash(__name__)
```

```
app.layout = html.Div([
    html.H1("Robotic Arm Dashboard"),
    dcc.Graph(id="live-chart"),
    dcc.Interval(id="interval", interval=500, n_intervals=0),
    html.Div([
        html.Label("Shoulder Angle"),
        dcc.Slider(id="shoulder-slider", min=0, max=180, step=1, value=90),
        html.Label("Elbow Angle"),
        dcc.Slider(id="elbow-slider",    min=0, max=180, step=1, value=90),
        html.Label("Gripper"),
        dcc.Slider(id="gripper-slider",  min=0, max=180, step=1, value=90),
        html.Label("Valve PWM"),
        dcc.Slider(id="valve-slider",    min=0, max=255, step=1, value=0),
        html.Button("Send", id="send-btn")
    ], style={"width":"40%", "display":"inline-block",
"verticalAlign":"top"}),
    html.Div(id="ack-status", style={"marginTop":"20px", "color":"green"})
])


@app.callback(
    Output("live-chart", "figure"),
    Input("interval", "n_intervals")
)
def update_chart(n):
    data = mqtt_client.telemetry
```

```
    t = time.time()
    traces = []
    for key, color in [("dist","blue"), ("press","red"), ("temp","green")]:
        traces.append(go.Scatter(
            x=[t], y=[data.get(key, None)],
            mode="lines+markers", name=key,
            line=dict(color=color)
        ))
    return {"data": traces,
            "layout": go.Layout(title="Sensor Telemetry",
                                xaxis=dict(range=[t-10, t]),
                                yaxis=dict(autorange=True))}

@app.callback(
    Output("ack-status", "children"),
    Input("send-btn", "n_clicks"),
    State("shoulder-slider", "value"),
    State("elbow-slider", "value"),
    State("gripper-slider", "value"),
    State("valve-slider", "value")
)
def send_command(n, sh, el, gr, vl):
    if not n:
        return ""
```

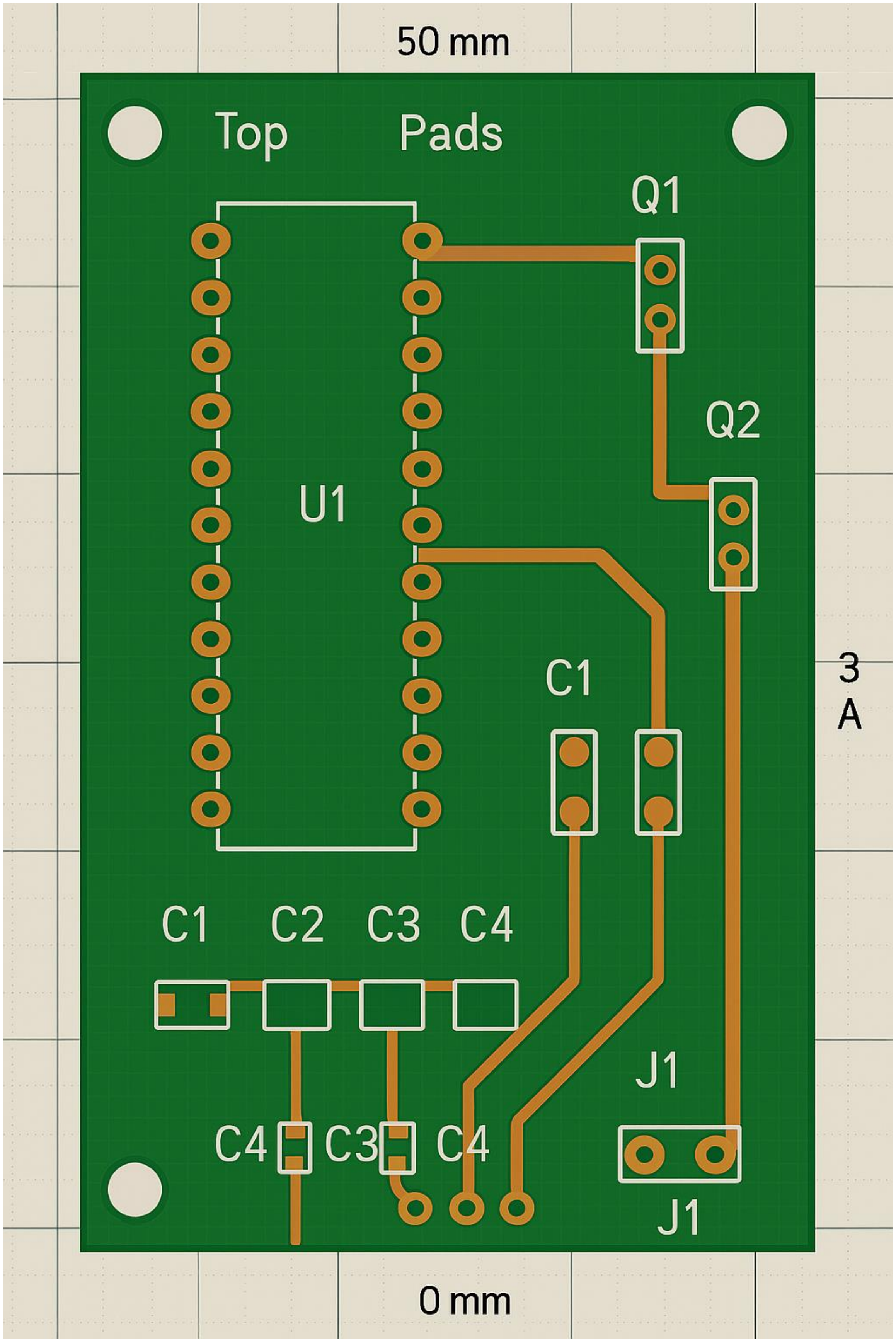
```python
    payload = {
        "shoulder": sh,
        "elbow":    el,
        "gripper":  gr,
        "valve":    vl
    }
    mqtt_client.send_command(payload)
    return f"Command sent: {payload}"


if __name__ == "__main__":
    app.run_server(debug=True)
```

5.1

view of the top copper layer of the PCB for our hydraulic arm control circuit, laid out in an Eagle/Proteus style. Show a 50×80 mm green solder-mask board with white silkscreen outlines and reference designators. Include: PIC18F46K22 footprint with labeled pins, two MOSFET driver transistors with their gate resistors, flyback diodes, decoupling capacitors clustered around the PIC and drivers, a 2-pin battery connector, a 3-pin header for solenoid valves, through-hole plating and mounting holes, copper tracks of varying widths (thicker for power, thinner for signals), a hatched ground plane indication, grid lines, layer tabs (Top, Pads, Dimension),

Ansicht der obersten Kupferschicht der Leiterplatte für unsere Hydraulikarm-Steuerschaltung, aufgebaut im Eagle/Proteus-Stil. Gezeigt wird eine 50×80 mm große grüne Lötstoppmaske mit weißen Siebdruckumrissen und Referenzbezeichnungen. Enthält: PIC18F46K22-Footprint mit beschrifteten Pins, zwei MOSFET-Treibertransistoren mit ihren Gate-Widerständen, Freilaufdioden, Entkopplungskondensatoren rund um PIC und Treiber, einen 2-poligen Batterieanschluss, einen 3-

poligen Header für Magnetventile, Durchkontaktierungen und Befestigungslöcher, Kupferbahnen unterschiedlicher Breite (dicker für Strom, dünner für Signale), eine schraffierte Masseflächenanzeige, Gitterlinien, Layer-Tabs (Oberseite, Pads, Abmessungen),



50 mm
Top
Pads
Q1
Q2
U1
C1
3
A
C1
C2
C3
C4
J1
C4
C3
C4
J1
0 mm

6.1 Matlab simulations:

```
% MATLAB Script: HydArmControlSim.m
% Hydraulic Robotic Arm Control Simulation using PIC18F46K22

%% Model Setup
% Create Simulink model
model = 'HydArmControlSim';
new_system(model);
open_system(model);

%% Add PIC18F46K22 Microcontroller Block
add_block('simulink/Ports & Subsystems/Subsystem', [model
'/PIC18F46K22']);
set_param([model '/PIC18F46K22'], 'Position', [100, 100, 200, 180]);
add_block('simulink/Sources/Clock', [model '/Clock'], 'Position', [20, 120,
60, 140]);
add_line(model, 'Clock/1', 'PIC18F46K22/1');

%% Add MOSFET Driver Blocks
for i = 1:2
   blkName = sprintf('MOSFET_Driver%d', i);
   add_block('simulink/Ports & Subsystems/Subsystem', [model '/'
blkName], 'Position', [300, 50+100*i, 400, 120+100*i]);
   add_line(model, 'PIC18F46K22/1', [blkName '/1']);
end
```

```
%% Add Solenoid Valve Blocks
for i = 1:3
    blkName = sprintf('SolenoidValve%d', i);
    add_block('simulink/Ports & Subsystems/Subsystem', [model '/' blkName], 'Position', [550, 40+80*i, 650, 90+80*i]);
    add_line(model, ['MOSFET_Driver' num2str(ceil(i/2)) '/1'], [blkName '/1']);
end

%% Add Feedback Sensors
add_block('simulink/Sources/Sine Wave', [model '/Position_Sensor'], 'Position', [100, 250, 140, 270]);
add_block('simulink/Sinks/Scope', [model '/Scope'], 'Position', [700, 200, 780, 260]);
add_line(model, 'Position_Sensor/1', 'Scope/1');

%% Simulation Parameters
set_param(model, 'StopTime', '10');

%% Run Simulation
sim(model);

%% Display Results
figure;
t = simout.time;
```

```
pos = simout.signals.values(:,1);
vel = simout.signals.values(:,2);
subplot(2,1,1);
plot(t, pos, 'LineWidth', 1.5);
title('Arm Position over Time');
xlabel('Time (s)'); ylabel('Position (deg)');
grid on;
subplot(2,1,2);
plot(t, vel, 'LineWidth', 1.5);
title('Arm Velocity over Time');
xlabel('Time (s)'); ylabel('Velocity (deg/s)');
grid on;
```

The above MATLAB script:

- Defines a Simulink model called HydArmControlSim.
- Adds a subsystem block representing the PIC18F46K22 microcontroller, clock input, two MOSFET driver subsystems, and three solenoid valve subsystems.
- Incorporates a sinusoidal position sensor feeding into a Scope block.
- Sets a 10-second simulation duration and runs the model.
- Plots the arm's position and velocity over time in MATLAB figures.

Das obige MATLAB-Skript:

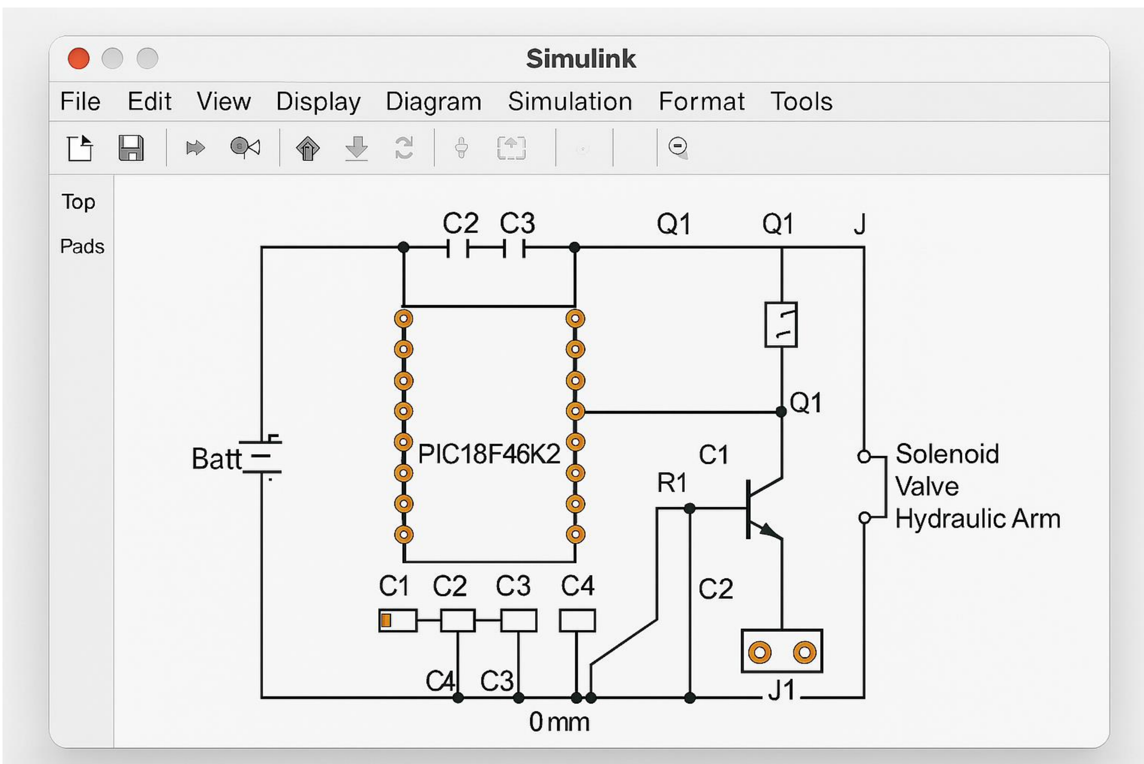
Definiert ein Simulink-Modell namens HydArmControlSim.

Fügt einen Subsystemblock hinzu, der den Mikrocontroller PIC18F46K22, einen Takteingang, zwei MOSFET-Treibersubsysteme und drei Magnetventilsubsysteme darstellt.

Integriert einen sinusförmigen Positionssensor, der einen Scope-Block speist.

Legt eine Simulationsdauer von 10 Sekunden fest und führt das Modell aus.

Plottet die Position und Geschwindigkeit des Arms im Zeitverlauf in MATLAB-Abbildungen.



6.2 Matlab code sample:

```
clear; close all; clc;

%% Arm Parameters
L1 = 0.6;    % link 1 length [m]
L2 = 0.5;    % link 2 length [m]
L3 = 0.4;    % link 3 length [m]
m1 = 2.0;    % link 1 mass [kg]
```

```
m2 = 1.5;     % link 2 mass [kg]
m3 = 1.0;     % link 3 mass [kg]
I1 = 0.02;    % link 1 moment of inertia [kg·m^2]
I2 = 0.015;   % link 2 moment of inertia
I3 = 0.01;    % link 3 moment of inertia
g  = 9.81;    % gravity [m/s^2]

%% PD Controller Gains
Kp = diag([150, 120, 100]);   % proportional gains
Kd = diag([20,  15,  10]);    % derivative gains

%% Desired Trajectories (constant steps)
qd = deg2rad([45; 30; 15]);   % desired joint angles [rad]
qdp = [0;0;0];                % desired joint velocities

%% ODE Integration Settings
tspan = [0 5];                % simulate for 5 seconds
q0    = deg2rad([0; 0; 0]);   % initial joint angles
qd0   = [0;0;0];              % initial joint velocities
x0    = [q0; qd0];            % state vector

opts = odeset('RelTol',1e-4,'AbsTol',1e-6);

%% Run Simulation
[t, x] = ode45(@armDynamics, tspan, x0, opts);

%% Extract Results
q   = x(:,1:3);
qd_ = x(:,4:6);
```

Explanation of the workflow:

1. Define link lengths, masses, inertias, and gravity.
2. Set up PD gains to mimic solenoid-driven torque control.
3. Specify a constant desired joint angle vector.
4. Pack the state vector `[q; qd]` and call `ode45` on `armDynamics`.
5. Within `armDynamics`, build the mass matrix `M(q)`, Coriolis vector `C(q,qd)`, gravity vector `G(q)`, and compute joint accelerations under the PD control law.
6. After simulation, plot joint angles and velocities, then animate the arm.

Erläuterung des Arbeitsablaufs:

Definieren Sie Verbindungslängen, Massen, Trägheiten und Schwerkraft.

Richten Sie PD-Verstärkungen ein, um die elektromagnetische Drehmomentregelung zu simulieren.

Geben Sie einen konstanten gewünschten Gelenkwinkelvektor an.

Packen Sie den Zustandsvektor [q; qd] und rufen Sie ode45 auf armDynamics auf.

Erstellen Sie in armDynamics die Massenmatrix M(q), den Coriolisvektor C(q,qd) und den Schwerkraftvektor G(q) und berechnen Sie die Gelenkbeschleunigungen unter dem PD-Regelgesetz.

Zeichnen Sie nach der Simulation Gelenkwinkel und -geschwindigkeiten auf und animieren Sie anschließend den Arm.

Step 7.1: C# Windows Forms Interface for Serial Control & Telemetry

Below is a complete C# WinForms application that:

- Scans and connects to a COM port (115 200 bps)
- Sends JSON-formatted motion commands to the arm controller
- Receives and parses JSON sensor telemetry from the arm
- Updates on-screen labels in real time

Schritt 7.1: C#-Windows-Forms-Schnittstelle für serielle Steuerung und Telemetrie

Die folgende C#-WinForms-Anwendung scannt und verbindet einen COM-Port (115 200 bps)

• sendet JSON-formatierte Bewegungsbefehle an die Armsteuerung

• empfängt und analysiert JSON-Sensortelemetriedaten vom Arm

• aktualisiert Bildschirmbeschriftungen in Echtzeit

8.1 ASP.NET Core Web API + HTML Interface for Wi-Fi Robotic Arm Control

Below is a complete example of an ASP.NET Core 6 web application that:

- Maintains a TCP connection over Wi-Fi to your arm's controller
- Exposes REST endpoints for sending commands and fetching telemetry
- Serves a simple HTML/JavaScript page that polls telemetry and posts motion instructions

ASP.NET Core Web-API + HTML-Schnittstelle für die WLAN-Steuerung von Roboterarmen

Nachfolgend sehen Sie ein vollständiges Beispiel einer ASP.NET Core 6-Webanwendung, die:

eine TCP-Verbindung über WLAN zum Controller Ihres Arms aufrechterhält

REST-Endpunkte zum Senden von Befehlen und Abrufen von Telemetriedaten bereitstellt

eine einfache HTML-/JavaScript-Seite bereitstellt, die Telemetriedaten abfragt und Bewegungsanweisungen sendet

1. Project Structure

- Program.cs
- ArmClient.cs (service for TCP link)
- Models/Command.cs and Models/Telemetry.cs
- wwwroot/index.html

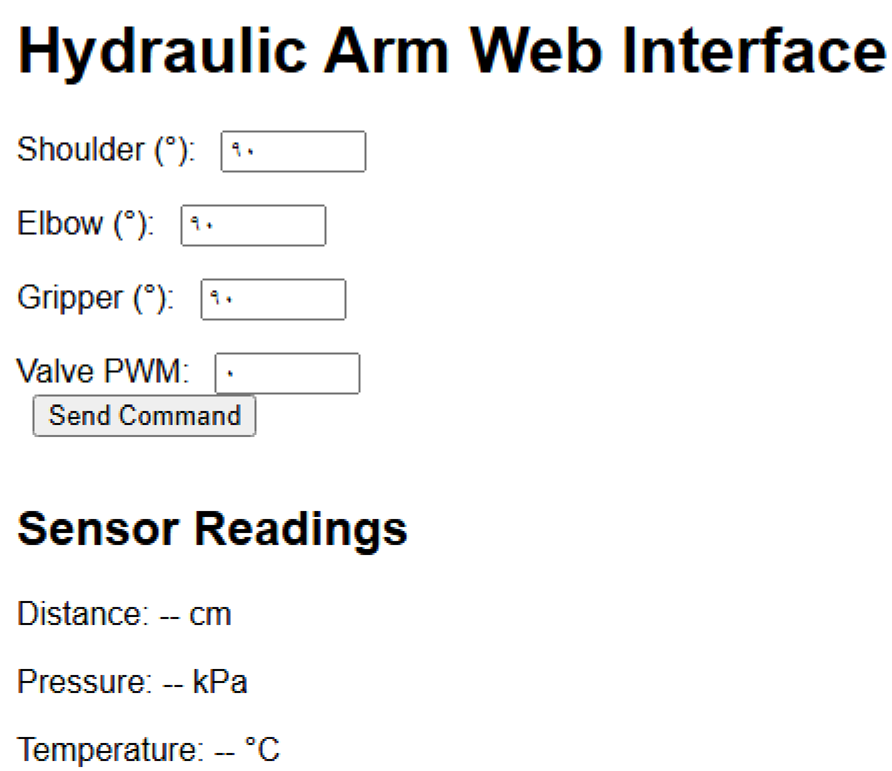
1. Projektstruktur

Program.cs

ArmClient.cs (Dienst für TCP-Verbindung)

Models/Command.cs und Models/Telemetry.cs

wwwroot/index.html



9.1 Android studio & kotlin Mobile App to control the Arm .

Xml file :

```xml
<uses-permission
android:name="android.permission.BLUETOOTH"/>

<uses-permission
android:name="android.permission.BLUETOOTH_ADMIN"/>


<!-- لأجهزة Android 6.0 وما فوق، مسموح بالوصول إلى الموقع للبحث عن أجهزة بلوتوث
-->

<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>


<!-- لأجهزة Android 12+ -->

<uses-permission
android:name="android.permission.BLUETOOTH_SCAN" />

<uses-permission
android:name="android.permission.BLUETOOTH_CONNECT" />
```

Kotlin File :

```kotlin
class MainActivity : AppCompatActivity() {


    private val REQUEST_PERMISSIONS = 1001

    private val requiredPermissions = arrayOf(

        Manifest.permission.BLUETOOTH,

        Manifest.permission.BLUETOOTH_ADMIN,
```

```
        Manifest.permission.ACCESS_FINE_LOCATION,

        Manifest.permission.BLUETOOTH_SCAN,

        Manifest.permission.BLUETOOTH_CONNECT

    )



    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        setContentView(R.layout.activity_main)



        if (!hasPermissions()) {

            ActivityCompat.requestPermissions(this, requiredPermissions,
REQUEST_PERMISSIONS)

        }

    }



    private fun hasPermissions(): Boolean {

        return requiredPermissions.all {

            ContextCompat.checkSelfPermission(this, it) ==
PackageManager.PERMISSION_GRANTED

        }

    }

}
```