

```
const action = btn.dataset.action;
if (!action) {
  console.error("No data-action on button:", btn);
  return;
}

console.log(buttonId, "→ action =", action);
...
}
```

 Code kopieren

## TL;DR

`action === undefined` means the browser cannot find `data-action` on that element.  
Not Flask. Not UART. Just HTML ↔ JS wiring.

If you paste **one button's HTML line** here, I'll point to the exact character that's wrong 🧐

[new chat](#)

All good 😊

New chat started — what do you want to work on now?



connect a sonar radar to raspberry pi

Got it 👍

I'll assume you mean a **sonar / ultrasonic distance sensor** (often called "sonar radar"), like **HC-SR04** or a **Waveshare ultrasonic radar module**, connected to a **Raspberry Pi**. I'll start with the **most common case (HC-SR04)** and note variations after.



## 1 What you need (typical setup)

- Raspberry Pi (any model with GPIO)
- Ultrasonic sensor (HC-SR04 or similar)
- 2 resistors (for voltage divider!)
  - 1 kΩ + 2 kΩ (or similar)
- Jumper wires

### ⚠ Important

The HC-SR04 ECHO pin outputs 5 V → Raspberry Pi GPIO is 3.3 V max → you must use a voltage divider.

## 2 Wiring (HC-SR04 → Raspberry Pi)

### Sensor pins

SCSS

VCC → 5V

TRIG → GPIO

ECHO → GPIO (via voltage divider)

GND → GND

 Code kopieren

### Example wiring

HC-SR04	Raspberry Pi
VCC	Pin 2 (5V)
GND	Pin 6 (GND)
TRIG	GPIO23 (Pin 16)
ECHO	GPIO24 (Pin 18) <b>via divider</b>

### Voltage divider (ECHO → GPIO)

lua

ECHO ---- 1kΩ ----+----- GPIO24  
                          |  
                          2kΩ



|  
GND

 Code kopieren

This drops 5 V → ~3.3 V safely.

---

### 3 Enable GPIO (if not already)

```
bash
```

```
sudo raspi-config
```

 Code kopieren

Enable:

- Interface → GPIO (usually already on)
- 

### 4 Python code (distance measurement)

```
python
```

```
import RPi.GPIO as GPIO
import time
```

```
TRIG = 23
ECHO = 24
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
```

```
GPIO.output(TRIG, False)
time.sleep(2)
```

```
def get_distance():
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    start = time.time()
    while GPIO.input(ECHO) == 0:
        start = time.time()
```



```
while GPIO.input(ECHO) == 1:
    end = time.time()

duration = end - start
distance = (duration * 34300) / 2 # cm
return round(distance, 2)

try:
    while True:
        dist = get_distance()
        print(f"Distance: {dist} cm")
        time.sleep(0.5)

except KeyboardInterrupt:
    GPIO.cleanup()
```

 Code kopieren

## 5 Common problems (and fixes)

Problem	Cause	Fix
Always 0 cm	Echo voltage too high	Use divider
Random values	Floating echo	Short wires
No output	Wrong GPIO numbering	Use BCM mode
Freezing loop	No echo received	Add timeout

## Timeout-safe version (important for robots)

python

```
timeout = time.time() + 0.04
while GPIO.input(ECHO) == 0:
    if time.time() > timeout:
        return None
```

 Code kopieren

## 6 "Radar" style (servo + sonar)

If you meant rotating radar:



- Mount sonar on a **servo**
- Sweep 0–180°
- Measure distance per angle
- Plot in Python / web UI

I can give you:

- Servo wiring
- Sweep code

Plot distance per angle



Aanmelden

Gratis registreren



## 7 If you're NOT using HC-SR04

Tell me:

- Exact **sensor model**
- UART / I2C / GPIO?
- Standalone Pi or robot (UGV, car, rover)?

I'll tailor the wiring + code exactly to your hardware 🙌



can raspberry pi run uart en gpio at the same time?

Yes 😊 — **absolutely.**

A Raspberry Pi can run **UART** and **GPIO** at the same time without any problem.

This is a very common setup in robots (UART for motor controller, GPIO for sonar, buttons, LEDs, etc.).

## How this works (conceptually)

- **UART** = a *dedicated hardware peripheral* (TX/RX pins)
- **GPIO** = general-purpose digital pins
- They are **independent subsystems** inside the Pi

Using one does **not block or interfere** with the other.



## Typical real-world example (like yours)

- **UART**
  - `/dev/serial0`
  - Motor controller / UGV base
- **GPIO**
  - Ultrasonic sensor (TRIG / ECHO)
  - Servo
  - Buttons

All running simultaneously ✔

## Important things to check (this is where people get tripped up)

### 1 Pin conflicts (MOST IMPORTANT)

Make sure you don't reuse UART pins as GPIO.

Default UART pins:

Function	GPIO	Physical pin
TX	GPIO14	Pin 8
RX	GPIO15	Pin 10

✗ Don't use GPIO14/15 for sonar, buttons, etc.

### 2 Disable serial console (if using UART for devices)

If UART is used for hardware (UGV, Arduino, etc.):

```
bash
```

 Code kopieren

```
sudo raspi-config
```

- Interface Options → Serial
  - Login shell over serial → **NO**
  - Enable serial hardware → **YES**

Reboot after.



### 3 Python can handle both at once

Example (UART + GPIO together):

python

 Code kopieren

```
import serial
import RPi.GPIO as GPIO
import time

# UART
ser = serial.Serial('/dev/serial0', 115200, timeout=0.1)

# GPIO sonar
TRIG = 23
ECHO = 24

GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

while True:
    ser.write(b'FORWARD\n')    # UART command
    GPIO.output(TRIG, True)    # GPIO pulse
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    time.sleep(0.1)
```

Works fine ✓

### 4 Performance & timing notes (robot-specific)

- UART runs in hardware → very reliable
- GPIO timing:
  - HC-SR04 works fine
  - Servo PWM works fine
- Avoid long **blocking loops** on GPIO
- Use **timeouts** on sonar echo



## When problems DO happen

They're usually due to:

- ✗ Reusing GPIO14 / GPIO15
  - ✗ Forgetting to disable serial console
  - ✗ Power issues (5V sag when motors run)
  - ✗ Blocking code (while loops without timeout)
- 

## TL;DR

- ✓ UART + GPIO at the same time → YES
- ✓ Common and safe
- ✓ Just avoid pin conflicts
- ✓ Disable serial console
- ✓ Watch your power rails

If you want, tell me:

- Pi model
- Which UART device
- Which GPIO devices

I can double-check your exact pinout so nothing collides 🛠️🚀



if on raspberry pi i run flask and use uart to control the ugv01 how and where do i have to create the stop command if the radar detects an object within 30 cm?

Great question — this is **exactly the right architecture question** 👍

You're thinking in the right direction.

Short answer first, then the clean design.

**The STOP command must be triggered in the backend (Python), not in Flask routes and not in JavaScript.**



The radar (sonar) and UART live on the **same authority level**: the Raspberry Pi backend.

---

## 🔑 Core principle (VERY IMPORTANT)

- Flask = web interface (user intent)
- Radar (sonar) = safety sensor (hard override)
- UART control = actuator (motors)

👉 Safety logic always overrides user commands

So:

- JS should *never* decide to stop the robot for safety
  - Flask routes should *never* block waiting for sensors
  - A **background safety loop** should enforce STOP
- 

## ✅ Correct architecture (clean & safe)

SCSS

📋 Code kopieren

Browser (buttons)

↓ HTTP

Flask routes → desired motion

↓

Motion controller (state)

↓

UART → UGV01 motor controller

↑

Radar safety thread (STOP override)

---

## 1 Create a shared motion state

This is the key.

python

📋 Code kopieren

```
from threading import Lock
```

```
motion_state = {
    "command": "stop",
```



```
"blocked": False    # radar override
}

state_lock = Lock()
```

---

## 2 Flask routes only SET intent (never drive directly)

python

 Code kopieren

```
@app.route('/action', methods=['POST'])
def action():
    action = request.json.get('action')

    with state_lock:
        motion_state["command"] = action

    return '', 204
```

✗ No UART writes here

✗ No sleeps

✗ No loops

---

## 3 Radar safety thread (this is where STOP happens)

This runs continuously in the background.

python

 Code kopieren

```
import threading
import time

STOP_DISTANCE_CM = 30

def radar_monitor():
    while True:
        dist = get_distance_cm() # your sonar function

        with state_lock:
            if dist is not None and dist < STOP_DISTANCE_CM:
                motion_state["blocked"] = True
```



```
        else:
            motion_state["blocked"] = False

        time.sleep(0.05) # 20 Hz
```

- 🔥 This is the authority for safety
- 🔥 This is where "object within 30 cm" is detected

---

## 4 Motion controller thread (only place that talks to UART)

python

 Code kopieren

```
def motion_controller():
    last_sent = None

    while True:
        with state_lock:
            if motion_state["blocked"]:
                cmd = "stop"
            else:
                cmd = motion_state["command"]

        if cmd != last_sent:
            send_uart_command(cmd)
            last_sent = cmd

        time.sleep(0.05)
```

- ✓ Sends STOP immediately
- ✓ Ignores user commands when blocked
- ✓ Prevents UART spam

---

## 5 Start background threads BEFORE Flask

python

 Code kopieren

```
if __name__ == '__main__':
    threading.Thread(target=radar_monitor, daemon=True).start()
    threading.Thread(target=motion_controller, daemon=True).start()
```



```
app.run(host='0.0.0.0', port=5000, threaded=True)
```

---

## 6 What `send_uart_command()` looks like

python

 Code kopieren

```
def send_uart_command(cmd):  
    if cmd == "forward":  
        ser.write(CMD_FORWARD)  
    elif cmd == "backward":  
        ser.write(CMD_BACKWARD)  
    elif cmd == "left":  
        ser.write(CMD_LEFT)  
    elif cmd == "right":  
        ser.write(CMD_RIGHT)  
    elif cmd == "stop":  
        ser.write(CMD_STOP)
```

---

## ✗ What NOT to do (common mistakes)

- ✗ Put radar logic in JavaScript
- ✗ Put radar loop inside a Flask route
- ✗ Call STOP from JS
- ✗ `while True` inside a route
- ✗ Let user override safety

---

## Why this design is solid

- Flask stays responsive
- Radar works even if browser disconnects
- UART is controlled from one place
- STOP is deterministic and immediate
- Easy to extend (speed, priorities, modes)

---

## Extra safety (highly recommended)



## Watchdog STOP if no commands received

python


 Code kopieren

```
last_cmd_time = time.time()
```

```
# In Flask route
```

```
last_cmd_time = time.time()
```

python

 Code kopieren

```
if time.time() - last_cmd_time > 0.5:  
    send_uart_command("stop")
```

---

### ❏ Final answer (plain language)

Create the STOP command in a background Python thread that monitors the radar. That thread must override Flask/user commands and directly control UART via a motion controller.

If you want, I can:

- Adapt this exactly for **UGV01 protocol**
- Add **direction-aware stopping** (only block forward)
- Show a **full working file**
- Add a **radar visualization in Flask**

Just tell me 👍

