ECEN 4213
Embedded Computer System Design

Final Project: Web-based Remote Robot Control

Instructor: Dr. Weihua Sheng, weihua.sheng@okstate.edu

TA: Zhanjie Chen, zhanjie.chen@okstate.edu

Claudia Pauyac, cpauyac@okstate.edu

Fall 2025

I. Lab 4 Due Date and Submission

II. Final Project Introduction

III. Final Project Due Date and Submission

Due date

- Lab demonstration:
  - ✓ no later than 7: 20 pm, October 28, 2025 (Tuesday Session)
  - ✓ no later than 7: 20 pm, October 29, 2025 (Wednesday Session)
  - ✓ no later than 5: 20 pm, October 31, 2025 (Friday Session)

- Lab report:
  - ✓ no later than 11: 59 pm, October 28, 2025 (Tuesday Session)
  - ✓ no later than 11: 59 pm, October 29, 2025 (Wednesday Session)
  - ✓ no later than 11: 59 pm, October 31, 2025 (Friday Session)

Submission (in a ZIP file)

- Lab report (Word or PDF file) must include supplemental questions, screenshots of your result, pictures of the circuit
- Your code

I. Lab 4 Due Date and Submission

II. Final Project Introduction

III. Final Project Due Date and Submission

II. Final Project Introduction

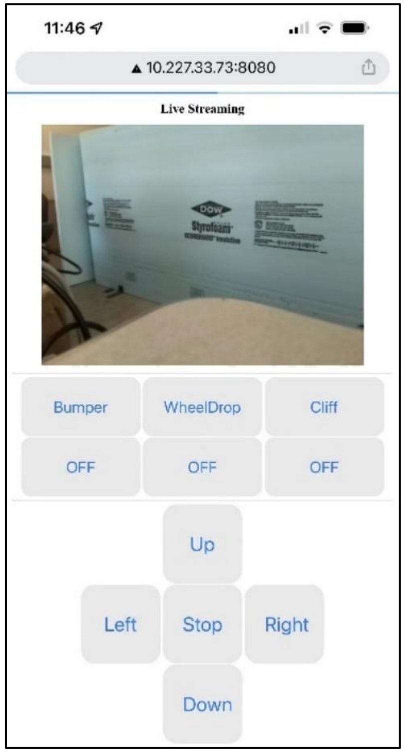
Final Project Objectives

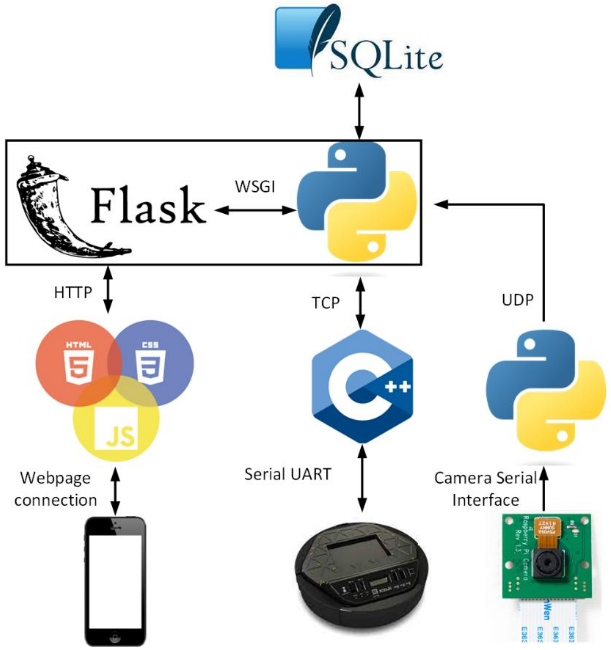
- Learn how to use HTML, CSS and JavaScript to build a simple webpage
- Learn how to use Flask web framework to create a web application
- Learn how to control the robot movement through a webpage
- Learn how to display live stream image obtained from a Raspberry Pi camera and sensor status

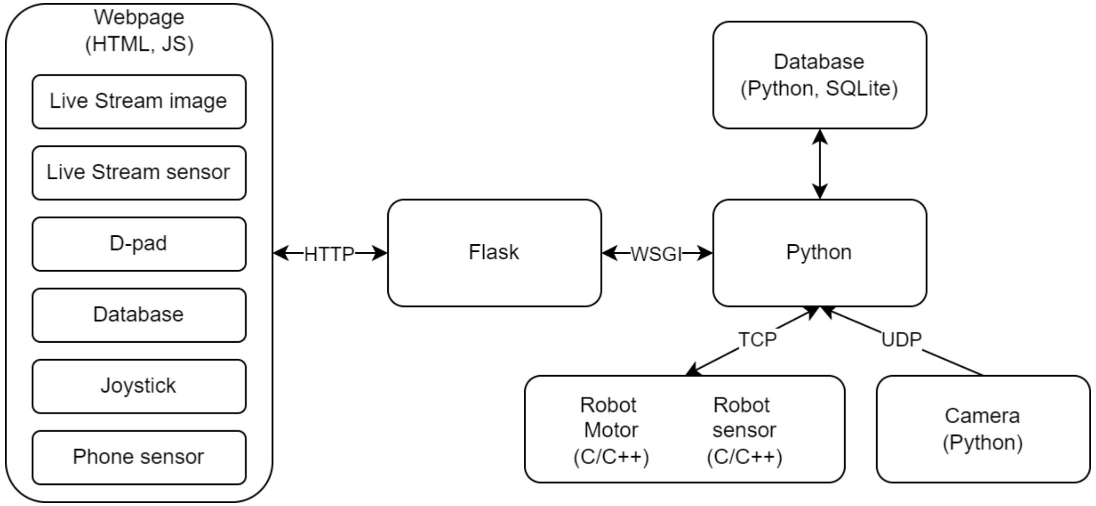
Final Project System Overview

The diagram outlines the architecture of the IoT-based robotic system. It integrates the following elements:

- **Frontend (Web Interface):** HTML/CSS/JS code running on a mobile device enables real-time interaction with the robot via a webpage.
- **Backend (Flask Server):** A Flask-based Python web server processes HTTP requests from the client and communicates with the hardware.
- **Robot Control:** Motor and sensor data exchange happens over Serial UART (C/C++), while camera streaming uses a Python UDP pipeline.
- **Database Management:** All logged data (e.g., sensor readings) are stored using SQLite, accessible by the Python backend.
- This setup allows bidirectional interaction – users can send commands to the robot and receive sensor/video feedback in real time.







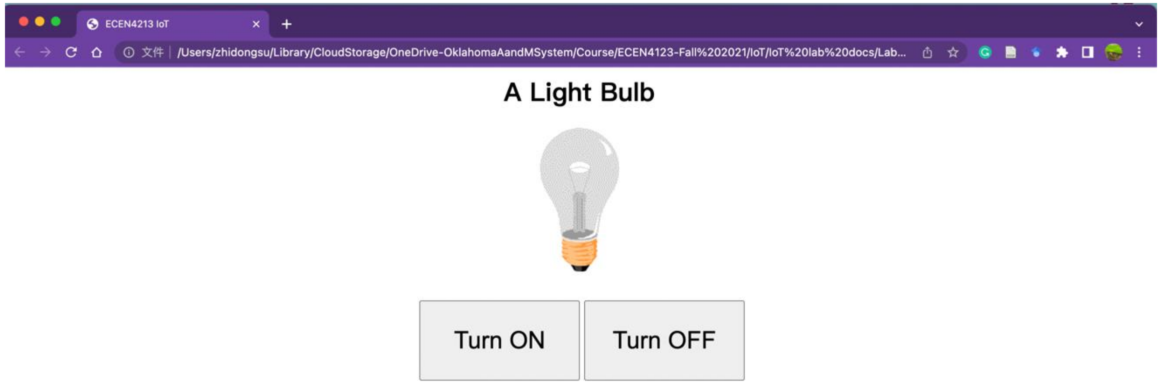
Web Programming (What you need to know)

- **HTML:** For webpage structure (buttons, images, text)
- **CSS:** For styling (color, size, layout)
- **JavaScript:** For making buttons and sensors work (interactivity)
- **Flask (Python):** Connects the webpage to the robot
- **C++:** Runs on the robot to move and sense
- **JSON:** Format used to send data (like { "x": 1, "y": 2})

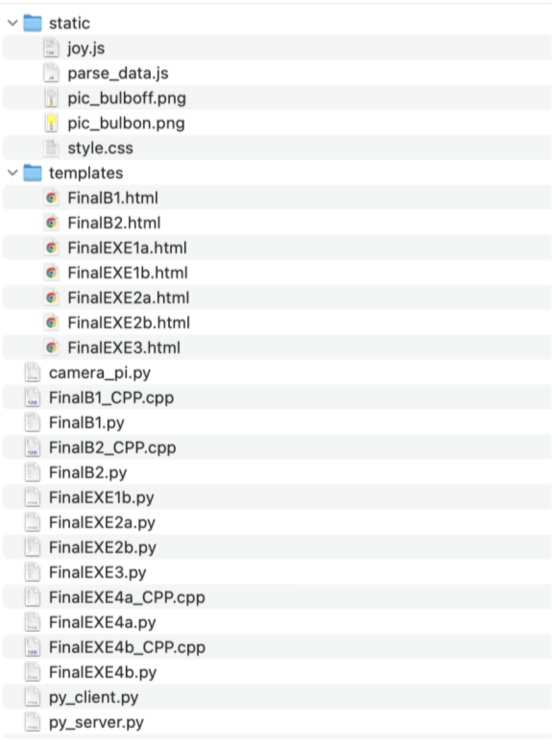
Exercise 1a – Basic Web Interface Setup

Objective: Build a simple webpage to simulate an IoT interaction.

- The webpage should display a title (“ECEN4213 IoT”), a description sentence (“A Light Bulb”), an image, and two buttons (Turn ON/Turn OFF) in the center of the webpage.



- File management is organized under */static* and */templates*, showing separation of styling, media, and logic.



Exercise 1b – Flask Integration and Remote Control

Objective: Make the light bulb image turn ON or OFF from a webpage you can access with your phone.

1. **Add JavaScript to make the buttons work**

   Open the file *FinalEXE1b.html* and complete the JavaScript so that:

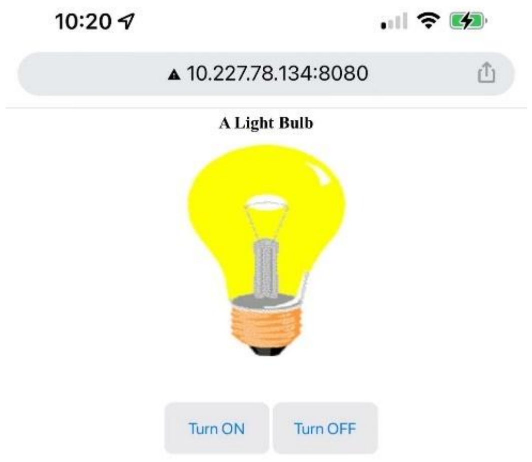
   - When you click the "Turn ON" button, the light bulb turns on by displaying the image *pic_bulbon.png*.
   - When you click the "Turn OFF" button, the light bulb turns off by showing the image *pic_bulboff.png*.

2. **Host your webpage using Flask**

   Once your HTML and JavaScript are working:

   - Use Flask, a Python web framework, to turn your webpage into a real web app.
   - This will allow you to open the webpage from your phone browser.

✅ **Important:** Your RPi and your phone must be connected to the same Wi-Fi network.



Notes:

1. Run the Python file using this command in the terminal: *python3 FinalEXE1b.py*
2. If your RPi IP address is *10.227.1.1*, open this link on your phone's browser: *http://10.227.1.1:8080*

Exercise 2a – Building a Web Dashboard to Control the Robot

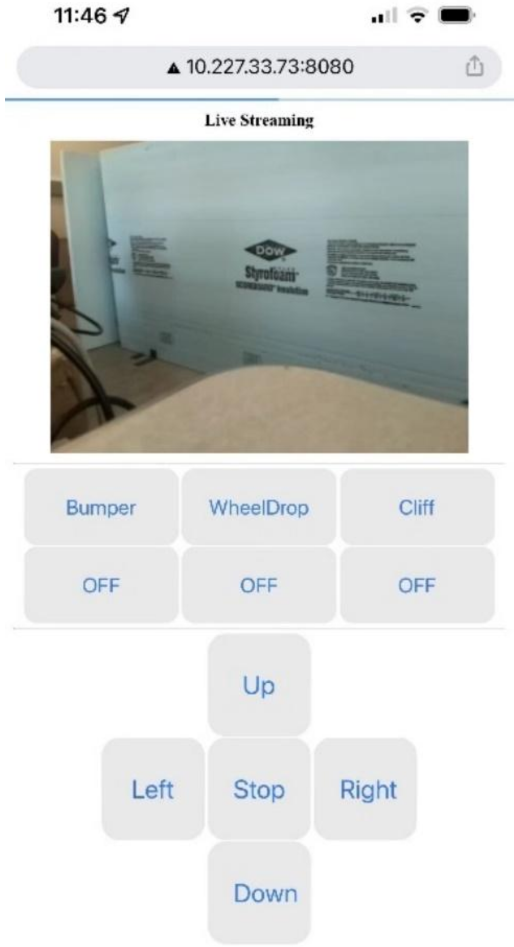
Objective: Design a simple control dashboard for the robot that you can open from your phone.

The webpage will include:

1. Live camera stream
2. Sensor status (e.g., bumper, wheel drop, cliff)
3. D-pad control buttons (Up, Down, Left, Right, Stop)

What to do:

1. Open the file *FinalEXE2a.html*
2. Complete the HTML so it includes:
    - An area for the live video feed
    - Buttons that show sensor states
    - A D-pad layout for controlling the robot
3. Run the Flask server with the Python file: *python3 FinalEXE2a.py*
4. Open the webpage on your phone using the IP and port shown in the terminal (e.g., *http://10.227.33.73:8080*)



✅ **Tip:** Place the elements neatly in your HTML using *<div>* containers and proper styling (CSS).

Exercise 2b – Dashboard Interactive with JavaScript and Flask

Objective: Make the control buttons and sensor status updates work in real time.

<u>Part 1</u>: Control the Robot with the D-Pad

1. In *FinalEXE2b.html*, use JavaScript to detect when a D-pad button is clicked (like "Up", "Left", etc.)
2. When a button is pressed, send a command to Flask using this structure:

```
<!-- callback function for the upbutton pushing event -->
<script type=text/javascript> $(function() { $("#upbutton").click(function (event) { $.getJSON('/UpFunction', { },
function(data) { }); return false; }); }); </script>
```

3. In the Python file *FinalEXE2b.py*, define the matching Flask route:

```
@app.route('/UpFunction')
def UpFunction():
    print('In UpFunction')
    return "Nothing"
```

Do the same for Left, Right, Down, and Stop buttons.

```
# define four funtions to handle the left, right, down and stop buttons
@app.route('/function_name')
def function_name():
    print('In XXFunction')
    return "Nothing"
```

Part 2: Show Live Sensor Updates

1. Open *parse_data.js* in the *static* folder
2. Complete the JavaScript code to update sensor buttons using data received from the server

```
if (!!window.EventSource) {
    var source = new EventSource('/');
    source.onmessage = function(e) {
      var bumper = e.data[1]
      var cliff = e.data[3];
      var drop = e.data[5];


      // finish the code to handle the bumper status
        if (bumper=="0")
          {
            document.getElementById("but1").value = "OFF";
          }
        if (bumper=="1")
        {
          document.getElementById("but1").value = "Right";
        }



        // finish the code to handle the wheel drop status
      if (drop=="0")
        {
          document.getElementById("the id of button where you need to display the sensor status").value = "OFF";

        }

```

3. Do the same for Wheel Drop and Cliff sensors

Recap:

- Use JavaScript to handle button clicks and send them to Flask
- Use Flask to receive commands and send back sensor data
- Your phone will display a live, interactive dashboard that controls and monitors the robot in real time

Exercise 3 – Live Stream Camera to Web Browser

Objective: Display a real-time video from the RPi camera in a browser using UDP communication.

1. Understand UDP Communication

- Review the sample codes: *py_server.py* (server) and *py_client.py* (client), which show a Python version of UDP client and server communication
- Run them in two different terminals to understand how the RPi sends and receives data using UDP

2. Server side: Receive image in Flask

- Open *FinalEXE3.py*
- In the Flask server, finish the gen() function to receive the images sent by the client:

```
def gen(camera):
    max_len = 65507
    frame = ''
    while True:
        # receive image to the client: frame = .....

        yield (b'--frame\r\n'
            b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

3. Client side: Send image from camera

- Open *camera_pi.py*
- Complete the *get_f()* function to send the image to the server

```
def get_f():
    global camera,connection
    image = camera.get_frame()
    print(len(image))
    try:
        # send image to the server
        pass
    except:
        print("something happened in client.sendto(image,server_address)")
```

🎛 Test: Run both scripts in separate terminals. Open your Flask server’s browser link – you should see a live video stream from the Pi camera!

Exercise 4a – Control Robot Movement via D-Pad

Objective: Send D-pad button actions from the webpage to a C++ program using sockets.

1. **Flask (Python) side: Handle button commands**
   - Open *FinalEXE4a.py*
   - Each D-pad button triggers a function and sends a command via socket to the C++ code

```
@app.route('/UpFunction')
def UpFunction():
    print('In UpFunction')
    cmd = 'u'
    connection.send(cmd.encode('utf-8'))
    return "None"
```

🔁 Repeat this logic for Left, Right, Down, and Stop buttons using unique characters.

2. C++ side: Receive and act on commands

- Open *FinalEXE4a_CPP.cpp*
- Complete the *read_socket()* function:

```
void read_socket(){
    char buffer[100];
    while(1){
        read(sock , buffer, 50);
        /*Print the data to the terminal*/
        cmd = buffer[0];
        printf("received: %c\n",cmd);

        // use cmd to control the robot movement


        //clean the buffer

    }

}
```

Run *FinalEXE4a.py*, *FinalEXE4a_CPP.cpp*, and *camera_pi.py* in separate terminals. Use your browser to control robot movement in real time

Exercise 4b – Display Real-Time Sensor Data

Objective: Send sensor data from the robot (C++) to the browser and show live updates.

1. Flask (Python): Stream sensor data to webpage

- Open *FinalEXE4b.py*
- Replace the fake sensor string *'b0c0d0'* with real variable values so that the sensor data can be sent to the webpage:

```python
@app.route('/')
def index():
    if request.headers.get('accept') == 'text/event-stream':
        def events():
            for i, c in enumerate(itertools.cycle('\|/-')):
                yield "data: %s\n\n" % ('b0c0d0')
        return Response(events(), content_type='text/event-stream')
    return render_template('Lab6EXE3.html')
```

2. C++ side: Send sensor data

- Open *FinalEXE4b_CPP.cpp*
- In the *main()* function, complete the logic to send real-time sensor data:

```
int main(){
    setenv("WIRINGPI_GPIOMEM", "1", 1);
    wiringPiSetup();
    kobuki = serialOpen("/dev/kobuki", 115200);
    createSocket();
    char buffer[10];
    std::thread  t(read_socket);

    while(serialDataAvail(kobuki) != -1)
    {
        // Read the sensor data.


        // Construct an string data like 'b0c0d0', you can also define your own data protocal.


        // Send the sensor data through the socket


        // Refer to the code in previous labs.
    }
    serialClose(kobuki);

    return(0);
}
```

Run *FinalEXE4b.py*, *FinalEXE4b_CPP.cpp*, and *camera_pi.py* in separate terminals. Then, visit the webpage. You'll be able to:

- 🔴 See live video
- 🕹️ Control the robot
- 📟 Monitor real-time sensor status

Bonus 1: Using a virtual Joystick to control the robot

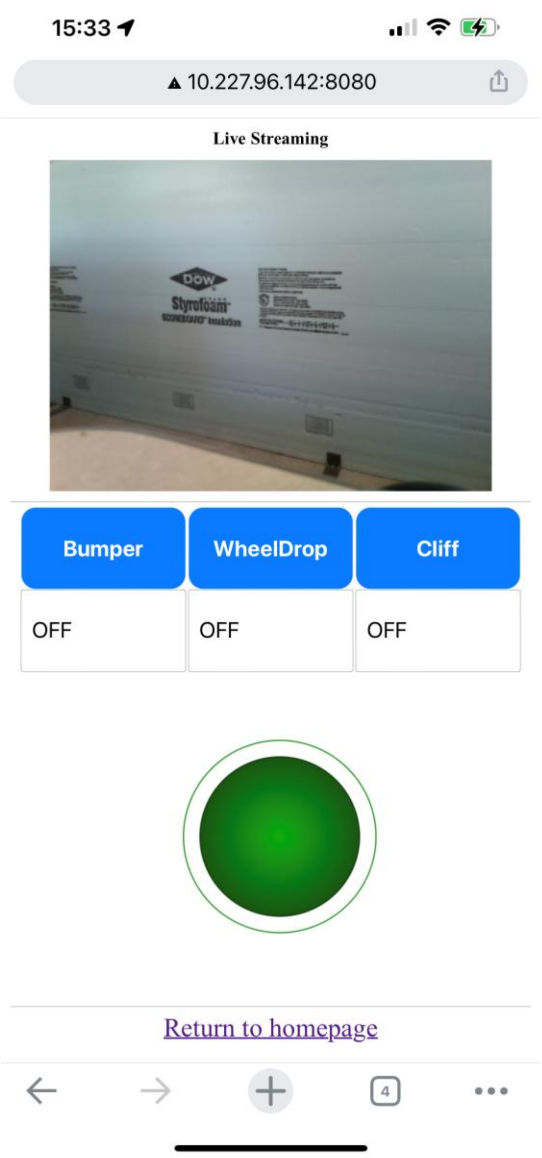
Objective: Control the movement of the Kobuki robot using a virtual joystick on a webpage.

- **Step 1: JavaScript – Send joystick data**

  File: *FinalB1.html*

  In the JavaScript section, complete the *sendJoystick()* function to collect joystick values (‘X’ and ‘Y’ positional data) and send them as JSON:

```
<script type="text/javascript" >
    var joy = new JoyStick('joy1');//,joy1Param);


    function sendJoystick(){
        // get the x axis position xpos
        var xpos = joy.GetX();

        // todo: get the y axis position ypos

        const xhttp = new XMLHttpRequest();

        xhttp.open('POST', "/joydata",false);
        xhttp.setRequestHeader("Content-Type", "application/json");

        // todo: format xpos and ypos into a JSON message with the format: {"x": xpos, "y": ypos}
        const json = {     };

        xhttp.send(JSON.stringify(json));
    }
    setInterval(sendJoystick ,500);
</script>
```



- **<u>Step 2</u>: Python Flask server**

  File: *FinalB1.py*

  This code already works – it receives the joystick JSON from the HTML webpage and forwards it to the C++ socket:

```
@app.route('/joydata',methods = ['POST', 'GET'])
def JoystickFunction():
    content_type = request.headers.get('Content-Type')
    if (content_type == 'application/json'):
        json = request.get_json()
        connection.send(str(json).encode('utf-8'))
        return "Content supported\n"
    else:
        return "Content not supported\n"
```

✅ No coding needed for this part

- **<u>Step 3</u>: C++ side – Control the robot**

  File: *FinalB1_CPP.cpp*

  Complete *read_socket()* to extract joystick values from the received buffer and use them to control the robot:

```
void read_socket(){
    char buffer[100];
    while(1){
        read(sock , buffer, 50);
        /*Print the data to the terminal*/
        cmd = buffer[0];
        printf("received: %c\n",cmd);

        // parse xpos and ypos from the buffer


        // use xpos and ypos to control the robot movement


        //clean the buffer

    }

}
```

Run *FinalB1.py*, *FinalB1_CPP.cpp*, and *camera_pi.py* in separate terminals

Bonus 2: Using phone orientation to drive the robot

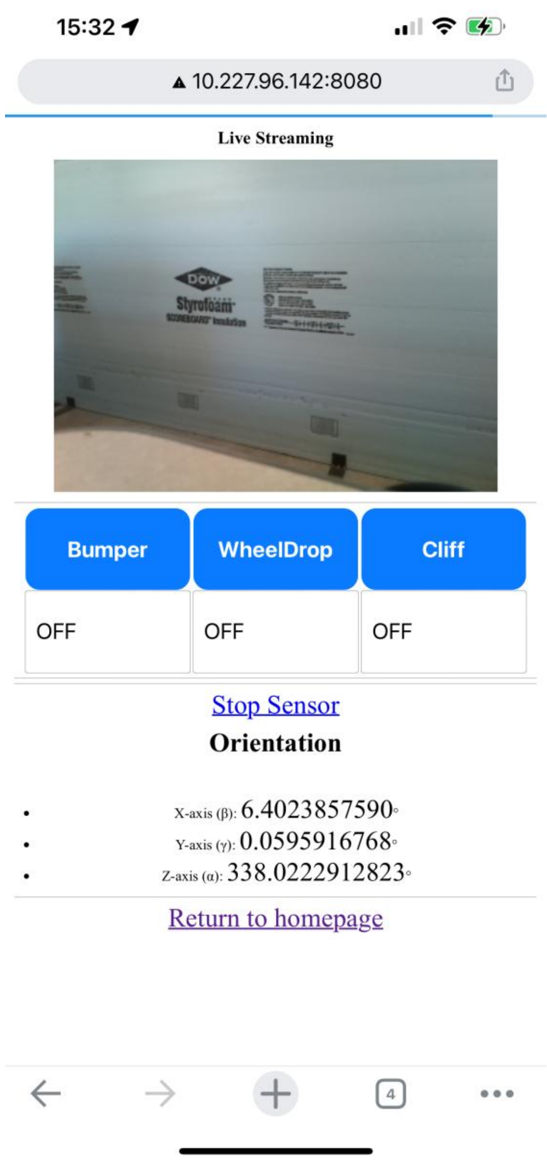
Objective: Control the movement of the Kobuki robot using your phone's tilt/rotation (gyroscope).

- <u>Step 1</u>: **JavaScript – Get phone sensor data**

  File: *FinalB2.html*

  Refer to the POST method used in the HTML code in *FinalB1.html* to send orientation as JSON data to the Flask server:

```
<script>
  function handleOrientation(event) {
    updateFieldIfNotNull('Orientation_a', event.alpha);
    updateFieldIfNotNull('Orientation_b', event.beta);
    updateFieldIfNotNull('Orientation_g', event.gamma);
    incrementEventCount();
  }
  function sendOrientation(){
    const xhttp2 = new XMLHttpRequest();
    var alpha = parseInt(document.getElementById('Orientation_a').innerHTML);
    var beta  = parseInt(document.getElementById('Orientation_b').innerHTML);
    var gamma = parseInt(document.getElementById('Orientation_g').innerHTML);
    const pjson = {"d": String('p'), "x": String(beta), "y": String(gamma), "z": String(alpha)};

    // refer to the POST method used in the HTML code in Lab6EXE6.html to
    // send the JSON data pjson to the Flask server.
    // todo: set up the POST method
    if (is_running){

    // todo: send pjson;

  }

}
```



- **Step 2: Python Flask server**

  File: *FinalB2.py*

  This code already works – it receives the JSON data from the HTML webpage and forwards it to the C++ code:

```
@app.route('/phonedata',methods = ['POST', 'GET'])
def PhoneFunction():
    content_type = request.headers.get('Content-Type')
    if (content_type == 'application/json'):
        json = request.get_json()
        print("Data ", json)
        connection.send(str(json).encode('utf-8'))
        return "Content supported\n"
    else:
        return "Content not supported\n"
```

  ✅ No coding needed for this part

- **Step 3: C++ side – Control based on phone sensor**

  File: *FinalB2_CPP.cpp*

  Complete *read_socket()* to read and process phone orientation data:

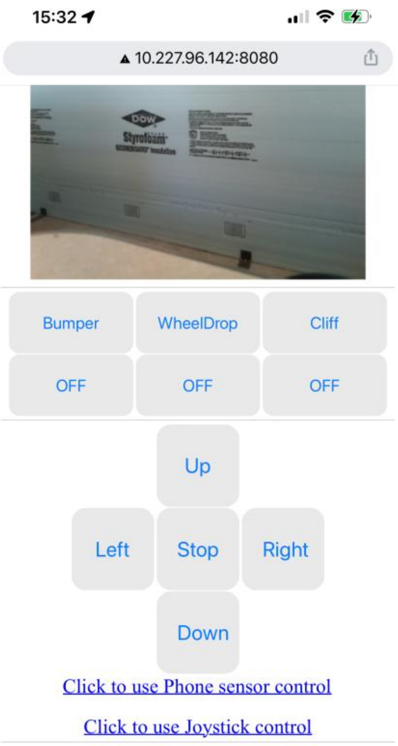
```
void read_socket(){
    char buffer[100];
    while(1){
        read(sock , buffer, 50);
        /*Print the data to the terminal*/
        cmd = buffer[0];
        printf("received: %c\n",cmd);
        // parse sensor data from the buffer


        // use the sensor data to control the robot movement


        //clean the buffer

    }

}
```

  Run *FinalB2.py*, *FinalB2_CPP.cpp*, and *camera_pi.py* in separate terminals

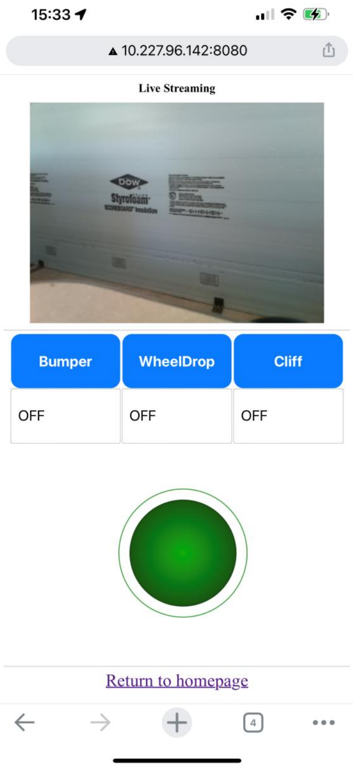
Bonus 3: Using phone orientation to drive the robot

Objective: Combine everything into one webpage with the three control options:

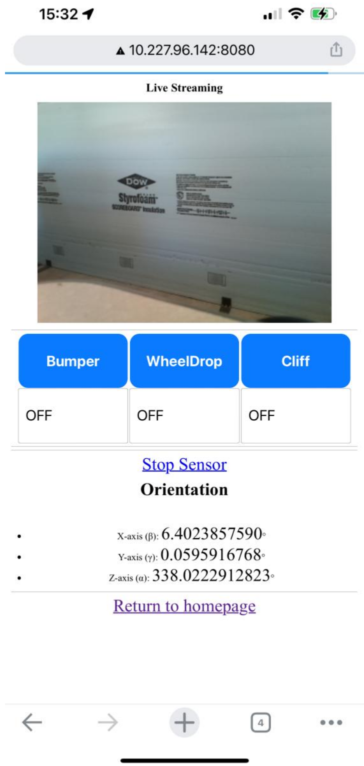
1. **D-Pad**
2. **Virtual joystick**
3. **Phone sensor**



Main page



Joystick control page



Phone sensor control page

Interface features:

- **Main page:** Contains buttons for sensor view and navigation to control modes
- **Joystick page:** Real-time robot control via virtual joystick
- **Sensor page:** Tilt-based control using phone's motion sensors

Navigation:

- Each subpage includes a "Return to homepage" link
- All functionalities (camera stream, sensor display, robot control) are integrated
- Subpages keep the same layout/functionality from earlier exercises

Outcome: By completing Bonus 3, you now have a fully functional web-based interface to:

- View the robot's camera
- See real-time sensor values
- Control the robot using three different input methods

Debugging Checklist

🛠 Common Mistakes & Fixes

- Is Flask running?
- Are your RPi and phone on the same Wi-Fi?
- Are you using the correct IP address?
- Did you restart the server after changing the code?
- Is the camera connected and accessible?

I. Lab 4 Due Date and Submission

II. Final Project Introduction

III. **Final Project Due Date and Submission**

Due date (Four weeks)

- Lab demonstration:
  - ✓ no later than 7: 20 pm, December 2, 2025 (Tuesday Session)
  - ✓ no later than 7: 20 pm, December 3, 2025 (Wednesday Session)
  - ✓ no later than 5: 20 pm, December 5, 2025 (Friday Session)

- Lab report:
  - ✓ no later than 11: 59 pm, December 2, 2025 (Tuesday Session)
  - ✓ no later than 11: 59 pm, December 3, 2025 (Wednesday Session)
  - ✓ no later than 11: 59 pm, December 5, 2025 (Friday Session)

What to submit?

A ZIP file that includes:

- Lab report (Word or PDF file)
  - Supplemental questions
  - Screenshots of your results
  - Pictures of the circuits
- Your code

Note: One group, one lab report.

Grading Criteria

The grading criteria is same as listed on the handout; however, if you don't demonstrate your code to TA, then 50% of maximum points are reduced directly.

Office hours

- Tuesday : 4:30 pm – 5:30 pm, Endeavor 350
- Wednesday: 4:30 pm – 5:30 pm, Endeavor 350
- Friday: 3:30 pm – 4:30 pm, Endeavor 350