UCLA Samueli School of Engineering

Exploring Real-Time WebRTC Interface on MCUs & MPUs for Unitree GO2 Robots

Akshara Kuduvalli

Motivation and Objectives

Motivation:

- There is one main Unitree GO2 Web-based Communication Interface using WebRTC, but is designed to run on a full computer
  - No microcontroller-based solution for any type of Unitree Go2 Robot control

Objective:

- Explore building a communication platform between a microcontroller and the Go2 Unitree Robots
- Ultimate goal of the project is to establish control from a microcontroller to the Robot, which enables a MCU - based platform for interacting with Go2 robots, opening up many avenues for future work

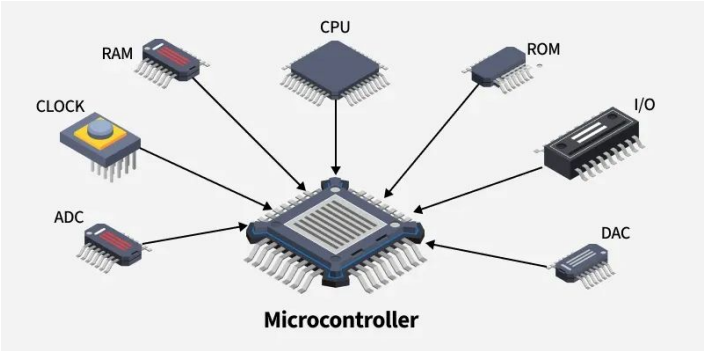
Technical Approach and Novelty

All past Go2 Communication Interfaces are over WebRTC hosted on a complete OS (ie. Ubuntu), such that the communication platforms for Go2 Robots only currently exists on MPUs / full computers.

- Robot controlled through full WebRTC stacks (Python/ROS2, Unitree desktop software).
- Not suitable for microcontrollers due to resource limits & general lack of existing WebRTC stack implementations.

Novel Approach:

- Initially, explored the potential of implementing the WebRTC stack directly on an ESP32, however, due to the little support with the full WebRTC stack (required for a Go2 robot connection) necessary on MCUs (needs full Crypto stacks, large, dynamic memory allocations, ICE + DTLS + SCTP over DTLS, **it was far too complex to implement on an MCU for now**.
- **Instead, explored a project to develop a Go2 Control platform on an MCU**, but offload the actual WebRTC stack to an intermediate Raspberry Pi.
- **Why is this still novel/cool?** This would be the first Go2 control platform intended for MCU applications, and can still be a basis for Go2 robot communication, as all other processing is offloaded to the microcontroller.



Previous Works

There exist a couple of WebRTC based Go2 Interfaces: all direct full computer - based robust solutions, leveraged these for the basis of the Raspberry Pi WebRTC stack and for gaining a deep understanding of the Go2 expected messaging protocol, which would be offloaded to the microcontroller

Unitree Go2 and G1 WebRTC Driver

This repository contains a Python implementation of the WebRTC driver to connect to the Unitree Go2 and G1 Robots. WebRTC is used by the Unitree Go/Unitree Explore APP and provides high-level control through it. Therefore, no jailbreak or firmware manipulation is required. It works out of the box for Go2 AIR/PRO/EDU models and G1 AIR/EDU.

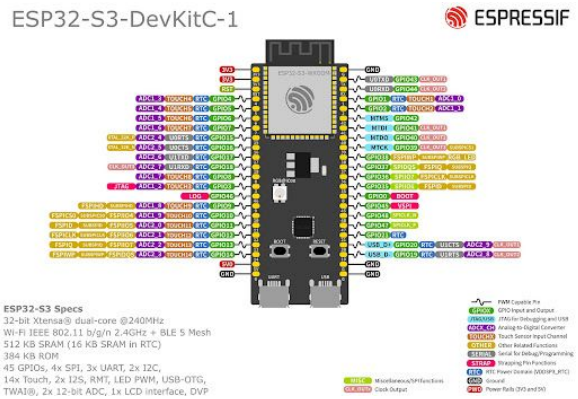
```
🕒 WebRTC connection        : 🟡 started       (12:06:17)
🕒 Signaling State          : 🟡 have-local-offer (12:06:17)
🕒 ICE Gathering State      : 🟡 gathering     (12:06:17)
🕒 ICE Gathering State      : 🟢 complete      (12:06:17)
🕒 ICE Connection State     : 🔵 checking      (12:06:18)
🕒 Peer Connection State    : 🔵 connecting    (12:06:18)
🕒 Signaling State          : 🟢 stable        (12:06:18)
🕒 ICE Connection State     : 🟢 completed     (12:06:18)
🕒 Peer Connection State    : 🟢 connected     (12:06:18)
🕒 Data Channel Verification: ✅ OK            (12:06:18)

🚨 Error Received from Go2:
🕒 Time:          2024-09-08 10:46:11
🔢 Error Source:  Communication firmware malfunction
❗ Error Code:    Battery communication error
```

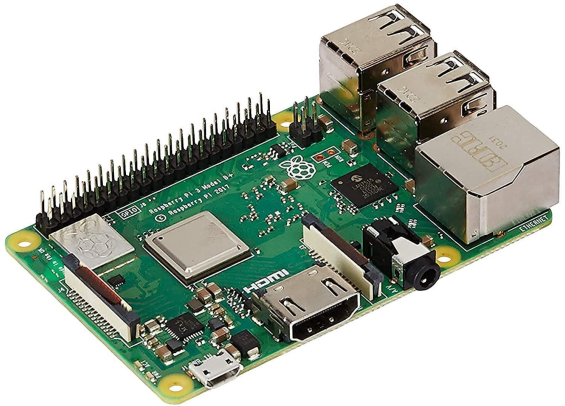
go2-webrtc - WebRTC API or Unitree GO2 Robots

Methods

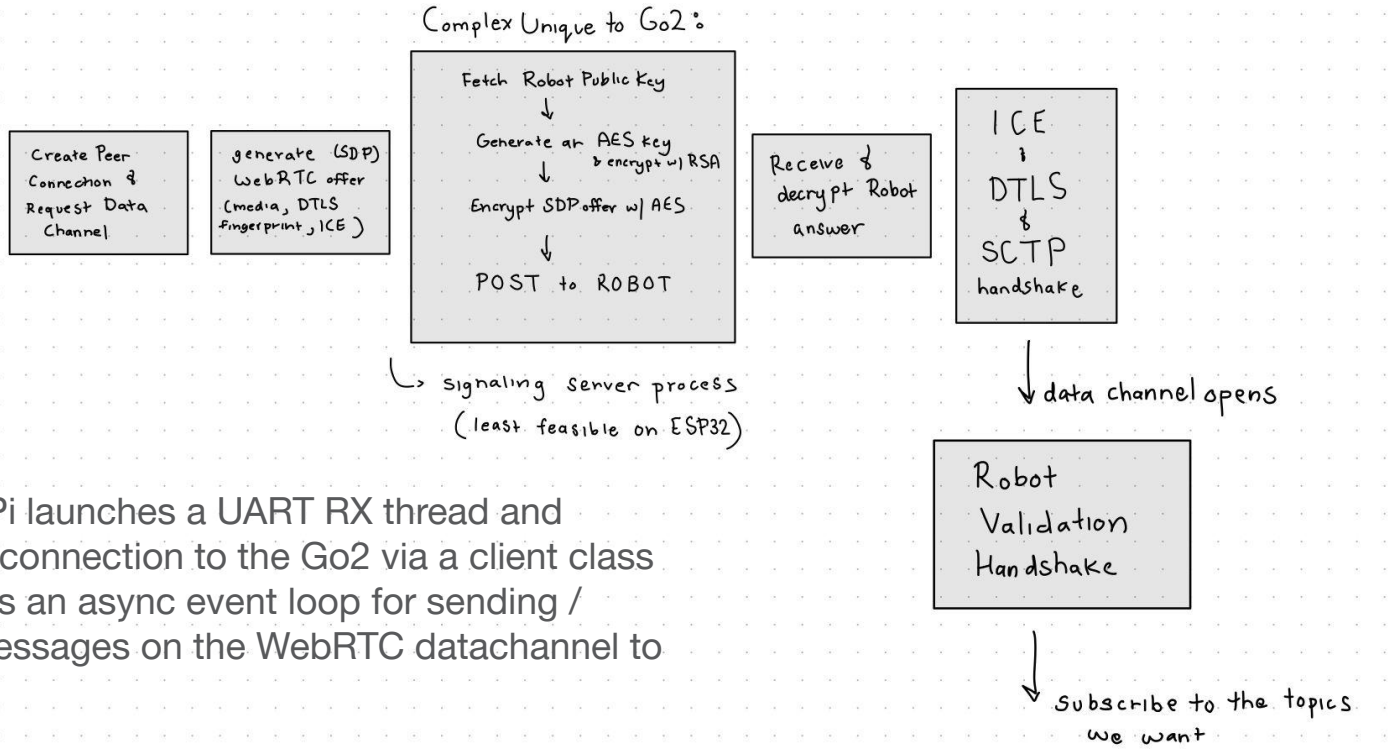
- **ESP32-S3-N16R8 board**
  - built-in Wi-Fi with additional Antenna, 8MB PSRAM, 16MB Flash, 2 existing UART ports
- **Built on the Zephyr RTOS platform**
  - Robust scheduling, existing ESP32s3 driver support, JSON parsing, multithreaded, WiFi mgmt, DTLS, WebSockets
  - Also very modular — should be easily able to port from ESP32s3 to other MCUs
- **Raspberry Pi**
  - Leverage existing Go2 WebRTC implementations to establish Go2 Robot interface

Zephyr®





WebRTC Stack on Raspberry Pi



Raspberry Pi launches a UART RX thread and establishes connection to the Go2 via a client class and contains an async event loop for sending / receiving messages on the WebRTC datachannel to the Go2

Software Design: High Level

Raspberry Pi

Establishes WebRTC connection and open data channel, forwards messages over UART to ESP32

ESP32

UART0

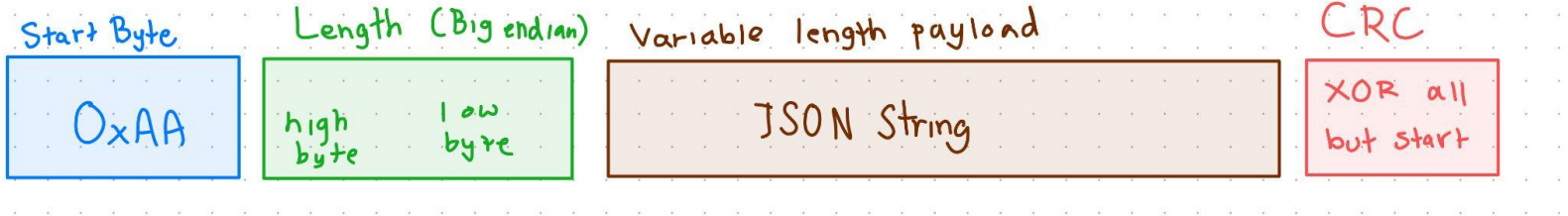
UART1

ESP32 - Raspberry Pi Communication

Why UART?

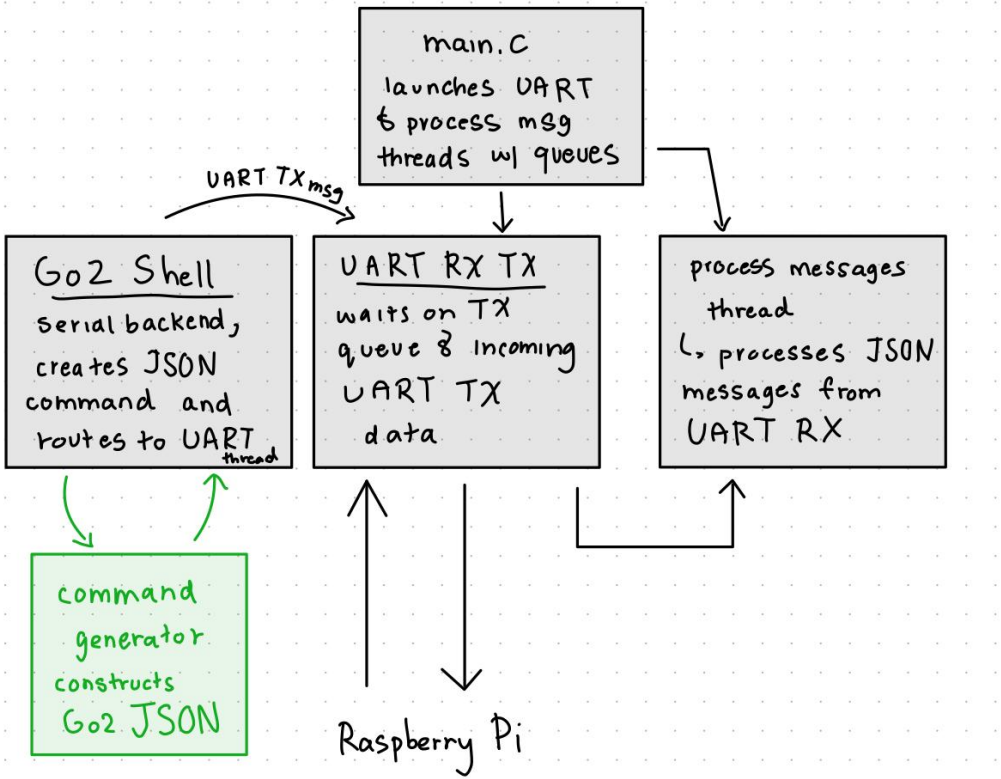
- Simple, separate TX/RX lines (as opposed to I2C), easy to design a packet protocol for both directions to accommodate larger packets
- Also looked into SPI, but ESP32 SPI device does not have peripheral mode support (in hardware / in Zephyr driver), and neither did Raspberry Pi, so it wasn't feasible (need controller-peripheral)

Designed simple UART Packet Structure between the ESP32 and Raspberry Pi



Similar Processing and Packet Construction in both Zephyr (ESP32) and in Python (Pi)

Software Design: Zephyr (ESP32s3)



Custom Zephyr Shell - Interactive Go2 Platform

Construct direct JSON messaging in Go2 expected messaging protocol in a Zephyr Thread

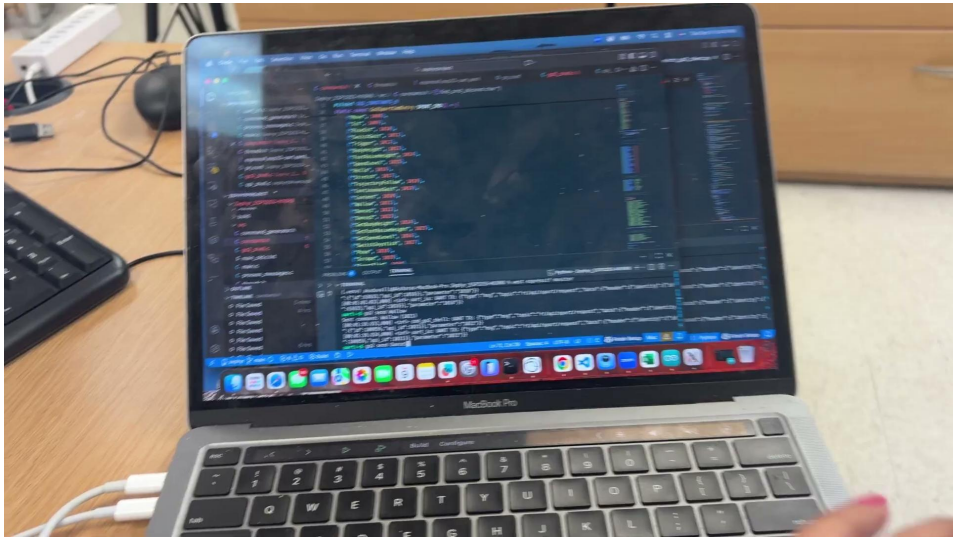
Created custom Zephyr Shell Commands (UART backend) to construct the command structure based on a struct of all command mappings

Shell places message in the TX queue, which will notify the UART TX/RX thread to send a message over UART to the Raspberry Pi

Results: Verifying ESP32 -> Go2 Robot control

```
uart:~$ go2 send Dance1
Sent command: Dance1 (1022)
[00:37:39.708,000] <inf> cmd_go2_shell: UART TX: {"type":"msg","topic":"rt/api/sport/request","data":{"header":{"id
entity":{"id":2259856,"api_id":1022}},"parameter":"1022"}}
[00:37:39.709,000] <inf> uart_io: UART TX: {"type":"msg","topic":"rt/api/sport/request","data":{"header":{"identity
":{"id":2259856,"api_id":1022}},"parameter":"1022"}}
uart:~$
```

Verifying Go2 Shell Command Construction -> Raspberry Pi over UART -> Go2 Robot as a direct send data channel message



Results - Stress Testing & Usage

```
[9/9] Linking C executable zephyr/zephyr.elf
Memory region         Used Size  Region Size  %age Used
           FLASH:      576392 B   16776960 B      3.44%
      iram0_0_seg:       61280 B     343552 B     17.84%
      dram0_0_seg:      220776 B     327168 B     67.48%
      irom0_0_seg:      377606 B         32 MB      1.13%
      drom0_0_seg:      445320 B         32 MB      1.33%
     ext_dram_seg:        1472 KB          8 MB     17.97%
     ext_iram_seg:           0 GB          8 MB      0.00%
     rtc_iram_seg:           0 GB          8 KB      0.00%
     rtc_slow_seg:           0 GB          8 KB      0.00%
         IDT_LIST:           0 GB          8 KB      0.00%
Generating files from /Users/akuduvalli/zephyrproject/Zephyr
```

```
uart:~$ clear
[00:03:34.785,000] <inf> uart_io: Received message: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:34.785,000] <inf> json_worker: Processing JSON: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:37.785,000] <inf> uart_io: Received message: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:37.785,000] <inf> json_worker: Processing JSON: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:40.786,000] <inf> uart_io: Received message: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:40.786,000] <inf> json_worker: Processing JSON: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:43.785,000] <inf> uart_io: Received message: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:43.785,000] <inf> json_worker: Processing JSON: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:46.786,000] <inf> uart_io: Received message: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:46.786,000] <inf> json_worker: Processing JSON: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:49.786,000] <inf> uart_io: Received message: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:49.786,000] <inf> json_worker: Processing JSON: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:52.787,000] <inf> uart_io: Received message: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"stamp": {"sec": 1765056882, "nanosec": 800237417}, "frame_id": "odom"}, "pose": {"position": {"x": -0.029953, "y": 0.005216, "z": 0.238949}, "orientation": {"x": 0.000776, "y": 0.014025, "z": -0.029462, "w": -0.999467}}}}
[00:03:52.787,000] <inf> json_worker: Processing JSON: {"type": "msg", "topic": "rt/utlidar/robot_pose", "data": {"header": {"st
```

Challenges

- **Understanding how WebRTC works**
- **Establishing WebRTC datachannel from the Go2 Robot to the Raspberry Pi**
  - Mismatches in the local offer made to the robot required extensive debugging
- **Designing and choosing a communication interface between the Raspberry Pi and the ESP32**
  - Processing data from the Go2 robot in Zephyr is still in an early state, requires more custom processing
  - Currently limited in message size based on UART processing TX/RX, future work could also be looking into other communication protocols

Future Work

- **Establishing more robust sensor data processing in Zephyr**
  - Currently does not do much post processing in the process_messages thread in Zephyr, goal was originally to establish the basis of the platform
  - Especially Lidar data processing on the ESP32
- **Creating complete applications using the communication platform for simultaneous robot control based on sensor input**
- **Also porting to other MCUs other than the ESP32, perhaps with more compute power and memory.**

- **Also, would continue researching the potential of establishing a direct MCU to Go2 robot control over WebRTC.**

Thank You!

Thanks Julian for the mentorship!

Previous Work & References:

https://github.com/tfoldi/go2-webrtc

https://github.com/legion1581/unitree_webrtc_connect

https://github.com/abizovnuralem/go2_ros2_sdk/tree/master/go2_interfaces