*Dogee*

Darya Korobkina   
Department of Computer Science  
Hof University of Applied ScienceHof, Germany  
dkorobkina@hof-university.deAmina Kenges  
Department of Computer Science  
Hof University of Applied Science  
Hof, Germany  
akenges@hof-university.de

*Abstract*—This project focuses on developing a voice-controlled interface for the Unitree Go1 robot. Initially, the objective was to utilize the robot’s onboard cameras for video recognition. However, due to hardware limitations, the focus shifted to implementing a voice recognition system. The system leverages technologies such as Django, Google Speech-to-Text API, the go1-js library, and WebSocket to enable the robot to execute commands based on voice inputs transmitted through a custom-designed web interface. This paper contributes to the robotics community by documenting the development process and addressing limitations in the existing documentation for the Unitree Go1 robot.

Keywords—Voice Recognition, Robotics, Unitree Go1, MQTT, Django, Google Speech Recognition API, go1-js Library.

# Introduction

Robotics education plays a pivotal role in advancing automation and artificial intelligence. This paper documents the development of a voice-controlled interface for the Unitree Go1 robot dog as part of a crash course in applied robotics. The primary goal was to implement a functional feature that enables the robot to interpret and execute spoken commands.

The Unitree Go1 robot presents significant potential in robotics education, but it is hindered by limited and poorly organized documentation. Initially, the project aimed to utilize the robot’s cameras for video recognition. However, non-functional cameras led to a pivot toward voice recognition systems, thereby addressing the robot’s lack of support for voice-based control. This project not only provides a solution for voice-based interactivity but also contributes to improving the documentation and understanding of the robot’s capabilities.

# Approach

## Robot Architecture

The Unitree Go1 robot’s internal hardware architecture comprises a Raspberry Pi, two NVIDIA Jetson Nano modules, and an NVIDIA Jetson Xavier NX. The Raspberry Pi acts as the central interface for developers, managing network connectivity, ultrasonic sensors, and peripheral processes. Jetson Nanos handles camera modules and ultrasonic sensors, while the Jetson Xavier NX, the most powerful computing unit, is responsible for machine learning tasks and advanced video analysis. These components are interconnected through an internal Ethernet switch, with additional connectivity via Wi-Fi and Bluetooth. This architecture supports modular development and seamless communication across the robot's subsystems.

The distribution of IP addresses for these components is preconfigured as follows:

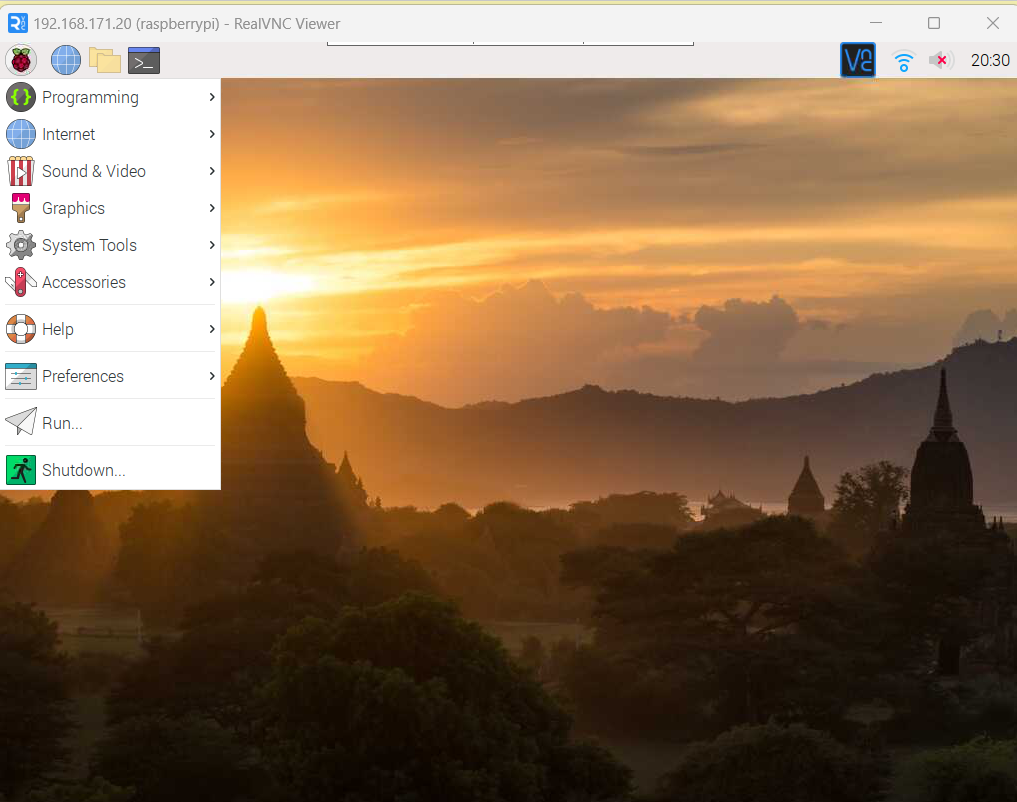
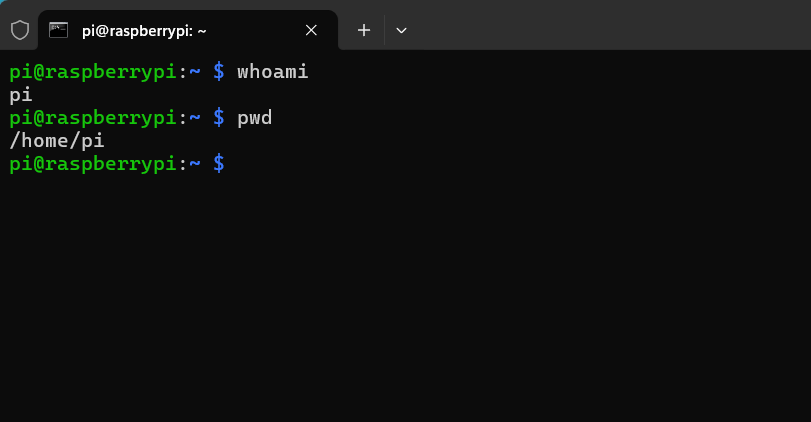
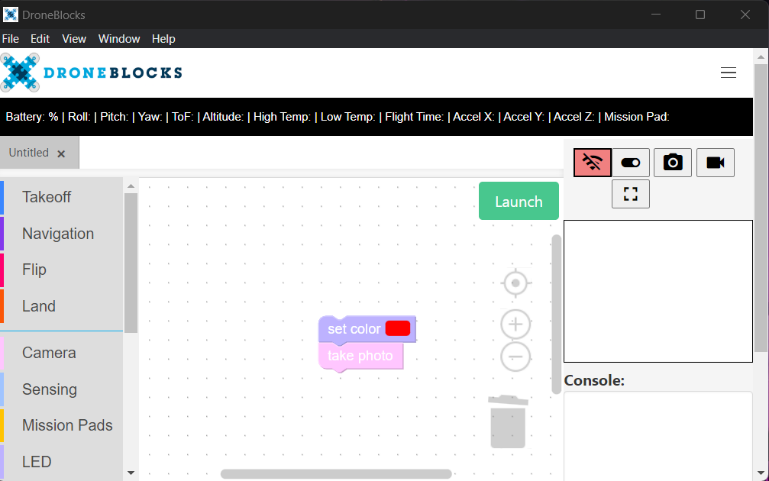
* MCU: 192.168.123.10.
* Raspberry Pi: 192.168.123.161.
* NVIDIA Jetson Nanos: 1st header – 192.168.123.13; 2nd pages – 192.168.123.14.
* NVIDIA Jetson Xavier NX: 192.168.123.15.

This architecture supports modular development and seamless communication across the robot's subsystems [1][2].

## Connection to robot

Initially, we accessed the robot’s graphical interface via HDMI by connecting to the Raspberry Pi interface. Subsequently, we discovered that the robot creates its own Wi-Fi network, allowing SSH access to the robot through the IP address 192.168.123.161. The onboard web interface, which provides access to the robot’s system settings and camera views, was also explored. Despite the cameras being inoperative, this interface clarified the robot’s existing capabilities. Additionally, via external monitor and mouse connected, we enabled the VNC functionality on the Raspberry Pi, allowing us to access the Raspberry Pi interface via RealVNC Viewer by connecting to the robot’s Wi-Fi network.

The robot was connected to the local network HAW.IoT using the Raspberry Pi graphical interface. The network’s firewall settings enabled simultaneous communication with the robot over SSH and VNC, while also providing access to the internet. Within this network, the robot was assigned the IP address 192.168.171.20, which was used for all communication and control tasks.

To enable VNC on the Raspberry Pi desktop, navigate to the start menu, then select Preferences > Raspberry Pi Configuration. In the configuration menu, choose the Interfaces tab and enable the VNC option. Click OK to save changes.  Alternatively, using the keyboard, navigate to Interfacing Options > VNC, select Yes, and confirm with OK.

## Methods and libraries

### **Unitree Camera SDK**: The Unitree Camera SDK is designed to provide developers with access to the robot’s camera modules for video processing and data capture. It includes functionality for streaming real-time video, capturing images, and integrating camera feeds into robotics applications. The SDK is implemented in C++ and supports interfacing with both the front and bottom cameras of the Unitree Go1 robot. It provides essential methods for configuring camera settings such as resolution, frame rate, and focus, as well as for synchronizing the camera feed with other sensors. However, due to hardware limitations, including non-functional cameras during this project, the library could not be fully utilized. Future work may explore the SDK’s capabilities more extensively once hardware issues are resolved.

### **Unitree Legged SDK**: The Unitree Legged SDK provides developers with tools for controlling and interfacing with the robot’s motor systems. It includes functions for implementing basic movement patterns, such as walking, turning, and balancing, and allows fine-tuned control over joint positions and velocities. The SDK supports multiple programming languages, including C++ and Python, enabling integration with various robotics systems. During this project,

### the SDK was tested for executing example scripts provided in the documentation. However, limited success was achieved due to incomplete and unclear documentation, which presented challenges in understanding the API structure and command execution. Despite these limitations, the SDK demonstrated potential for enabling low-level control of the robot’s actuators and sensors. Future efforts should focus on improving the documentation and exploring advanced control strategies using this SDK.

### **Droneblocks Application**: The DroneBlocks application provides a user-friendly platform for creating pipelines of robot commands. It offers functionalities such as changing the color of LEDs, capturing photos, moving the robot in all directions, switching modes, and executing specific actions like dancing or standing up. However, during testing, LED controls were ineffective due to hardware issues with the robot's head, and mode-switching commands failed to execute reliably. Only the functionality to capture images was successful, but it was limited to the robot's downward-facing cameras, which were not particularly useful in the context of this project.

### **Droneblocks Javascript Library**: The DroneBlocks JavaScript Library, built upon the MQTT protocol, offers developers a robust framework for programming the Unitree Go1 quadruped robot using JavaScript. This library facilitates control over the robot's movements and interactions with its sensors, enabling the creation of custom programs and scripts. The library is built upon the go1-js NPM package, which utilizes the MQTT interface for communication with the robot. Developers can implement various functionalities, including movement commands, sensor data retrieval, and execution of predefined behaviors. However, during testing, certain limitations were observed, such as the robot's inability to adjust its head position while in motion, which may be attributed to the current implementation of the MQTT interface. Despite these challenges, the DroneBlocks JavaScript Library remains a valuable tool for developing advanced applications with the Unitree Go1 robot.

*To utilize the DroneBlocks JavaScript library for controlling the Unitree Go1 robot, developers must first import the library and instantiate the Go1 class with the appropriate parameters. Specifically, the IP address of the robot within the HAW.IoT network should be provided. For instance, if the robot's IP address is 192.168.171.20, the instantiation is performed as follows.*

const MyRobotDog = new Go1({ host: "192.168.171.20" });

*After creating the instance, it is essential to initialize the robot to establish communication.*

await MyRobotDog.init();

*The DroneBlocks library offers the capability to change the robot's modes, including options such as 'stand', 'walk', and 'dance'. Additionally, it provides commands for movement, such as goForward() and goBackward(), enabling comprehensive control over the Unitree Go1's operations.*

### **MQTT**: The MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe network protocol that facilitates efficient communication between devices in constrained environments. In the context of the Unitree Go1 robot and the DroneBlocks JavaScript library, the MQTT interface serves as the primary communication channel, enabling developers to send commands and receive data from the robot in real-time.

### **Web Interface for Audio Recording Using Django Framework**: To create a web interface for recording audio and sending it to backend scripts for processing, we utilized the Django framework. The interface facilitates audio input, processing, and execution of commands derived from the input.

#### Setting Up the Django Environment: The following steps were performed to set up the Django project and application:

Create and activate virtual environment:

python -m venv .venv

.venv\Scripts\activate

Install django:

pip install django

Create Django Project and Application:

django-admin startproject audiorecorder

cd audiorecorder

django-admin startapp recorder

Update INSTALLED\_APPS in settings.py.

#### Creating Views, URLs, and Templates: To define a page for audio recording, create a view in recorder/views.py. Define URL Patterns by appending recorder/urls.py. Update the main urls.py to include application routes.

#### Creating the Audio Recording Template: An HTML template was created to enable audio recording. The template uses Bootstrap for styling and JavaScript to handle audio recording and playback and converts audio into PCM WAV format so it can then be processed by speech recognition library.

## Core Functionality: Voice Recognition

The project uses Google Speech-to-Text technology to convert audio input into text. Audio files are processed with the SpeechRecognition python library, which handles the recording and recognition process. This ensures accurate transcription of user commands while supporting the English language (en-US). The integration leverages Google’s robust cloud-based speech recognition service to handle complex natural language inputs. In the project we used only English language recognition, but in a wish, we can choose over 120 languages, it isn't hard to set up for example German language.

Recognized text is mapped to predefined commands using a dictionary called COMMAND\_MAPPING, which serves as a lookup table. Each key represents a possible voice input, while the corresponding value specifies the robot's action. For example, the voice command "go forward" maps to the action "goForward". All commands written on the table below.

| ***Voice command*** | Command |
| --- | --- |
| go forward; forward; go, walk; | goForward |
| go back; back | goBackward |
| turn left; left | turnLeft |
| turn right; right | turnRight |
| dance one | dance1 |
| dance two | dance2 |
| helicopter | helicopter |

## Web Interface for Audio Recording Using Django Framework

To create a web interface for recording audio and sending it to backend scripts for processing, we utilized the Django framework. The interface facilitates audio input, processing, and execution of commands derived from the input.

### **Setting Up the Django Environment**:

*The following steps were performed to set up the Django project and application. Create and activate virtual environment:*

python -m venv .venv

.venv\Scripts\activate

*Install django:*

pip install django

*Create Django Project and Application:*

django-admin startproject audiorecorder

cd audiorecorder

django-admin startapp recorder

*Update* INSTALLED\_APPS in settings.py*.*

### **Creating Views, URLs, and Templates:**

*To define a page for audio recording, create a view in recorder/views.py.*

*Define URL Patterns by appending recorder/urls.py.*

*Update the main urls.py to include application routes.*

### **Creating the Audio Recording Template:**

*An HTML template was created to enable audio recording. The template uses Bootstrap for styling and JavaScript to handle audio recording and playback and converts audio into PCM WAV format so it can then be processed by speech recognition library.*

## Integration of Audio Processing and Command Execution in Django Framework.

The project uses the *@droneblocks/go1-js* library to control the Go1 robot dog, commandsExecutor.js. A set of predefined actions is stored in the COMMAND\_ACTIONS object, where each action corresponds to a specific robot behavior, such as walking, turning, or performing gestures. For instance, the command "goForward" makes the robot move forward at 0.25 m/s for 4 seconds, while "dance1" initiates a predefined dance move.

The robot's modes, like walking (Go1Mode.walk) or standing (Go1Mode.stand), are controlled using asynchronous functions. This structure ensures real-time interaction and smooth execution of commands.

A WebSocketServer runs on port 8765, enabling communication between external clients (e.g., a Python script) and the robot controller. Incoming messages, representing commands, are received and checked against the COMMAND\_ACTIONS dictionary. If a valid command is found, the corresponding action is executed asynchronously.

This WebSocket-based architecture provides efficient and low-latency communication, ensuring seamless integration between the control system and the robot.

To integrate the audio recording interface with the robot control system, we appended a view in the Django project to process audio and send commands to the Unitree Go1 robot via a WebSocket connection. The high-level workflow and implementation details are described below.

WEBSOCKET\_URL Specifies the WebSocket server address for sending commands to the robot.

Django Viewrecord\_audio renders the HTML interface (recorder/record.html) where users can record or upload audio. send\_command\_to\_jsan asynchronous function responsible for communicating with the WebSocket server, initializing the robot with a default command ("walk"), and sending the processed Audio files are received fromrequest.FILES in the process\_audio view. The speech\_recognition library is used to process the audio file through its AudioFile module.

The speech\_recognition.Recognizer class is used to transcribe audio into text using the Google Speech Recognition API. Successfully recognized speech is logged for debugging and further processing. The recognized text is passed to a simplify\_command function, which maps it to robot-compatible commands using the COMMAND\_MAPPING dictionary. Valid commands are logged for execution, while unrecognized commands return an error response.

Valid commands are sent to the robot using the send\_command\_to\_js asynchronous function. This function connects to the WebSocket server, sends an initialization command ("walk"), and subsequently sends the actual command derived from the processed audio. The process ensures reliable initialization and execution of robot commands.

Speech recognition errors are addressed by returning an error response when speech is not understood. Connectivity issues with the speech recognition API are logged and handled gracefully, ensuring robust operation.

*manage.py* is is the main entry point for a Django project, which serves command "runserver". We appended this file, so it simultaneously starts a Node.js server for handling commands.

# Experiments

## Unitree Legged SDK

We initially attempted to run examples from Unitree Legged SDK to explore its capabilities. Specifically, we aimed to execute predefined movement sequences provided in the SDK. Despite thorough testing, the examples failed to execute as expected, highlighting issues with compatibility or setup that required further troubleshooting.

## Unitree Camera SDK

To integrate camera functionality, we tried to connect to the robot's camera module using the Unitree Camera SDK. Our approach included connecting to the robot via SSH and attempting to initiate operations through the library. Unfortunately, it became evident that the robot's cameras were non-functional. Despite various efforts, including recording video attempts, this feature could not be utilized due to hardware limitations.

## Droneblocks Application Testing

To test the Droneblocks application, we experimented with controlling the LED on the robot's head and changing its mode. These tests revealed that the LED functionality was inoperative, likely due to the robot's head hardware being disabled. Additionally, changing the operational mode through Droneblocks commands was unsuccessful, further emphasizing the hardware limitations.

## JavaScript Library Testing

Using the JavaScript library, we tested fundamental commands such as changing the robot's mode and making it move forward. These tests were successful, confirming the library's reliability for implementing basic command execution. As a result, this method was chosen as the foundation for the voice command system.

## Results

The final system successfully executed commands such as walking, turning, and performing pre-defined dances. Key performance metrics include:

* **Response Time:** 2–4 seconds for command execution.
* **Accuracy:** 95% of commands were correctly recognized and executed. Approximately 5% of commands were either unrecognized or incorrectly interpreted.

Despite these limitations, the project achieved its primary objective of implementing a functional voice-command system.

## Discussion

The project demonstrated the feasibility of adding voice control to the Unitree Go 1 robot, providing valuable insights into robotics development. The challenges faced, particularly with hardware limitations, highlighted the importance of adaptability in project development. The educational value of the project was significant, offering hands-on experience with robotics hardware, software integration, and troubleshooting.

# Conclusion and Future Work

This paper presents a working prototype of a voice-command system for the Unitree Go 1 robot dog. While the system achieved its primary objectives, several areas for improvement remain. Future work could focus on:

* **Enhancing Recognition Accuracy:** Incorporating advanced natural language processing techniques.
* **Expanding Functionality:** Adding support for more complex commands and behaviors.
* **Object Detection Integration:** Revisiting the integration of object detection capabilities when hardware issues are resolved.

By addressing these aspects, the system could evolve into a robust and versatile interface for robotics control.

##### References

1. <https://github.com/dillisium/Applied_Robotics/blob/docs/2023-09-19_Masterarbeit%20NL%20-%20eingereicht.pdf>
2. <https://github.com/dillisium/Applied_Robotics/blob/docs/2023-09-19_Masterarbeit%20NL%20-%20eingereicht%20en.pdf>
3. <https://github.com/MAVProxyUser/YushuTechUnitreeGo1?tab=readme-ov-file#robot-internal-architecture>
4. <https://github.com/geyang/unitree-go1-setup-guide>
5. <https://www.docs.quadruped.de/projects/go1/html/operation.html>
6. <https://github.com/unitreerobotics/unitree_legged_sdk>
7. <https://www.youtube.com/watch?v=-Ag95gOYTCU>
8. <https://github.com/maggusscheppi/Go1?tab=readme-ov-file#run-your-own-scripts-at-boot>
9. <https://www.youtube.com/results?search_query=unitree+go+1+how+to+upload+code+python>
10. <https://www.youtube.com/watch?v=XqZax8b3YdY>
11. <https://www.youtube.com/watch?v=Wnd5IUNbXnI>
12. <https://www.youtube.com/watch?v=o6JtfTM3ArY>
13. <https://www.youtube.com/watch?v=nafv21HeeEM>
14. <https://www.youtube.com/watch?v=tTCbdul7xsc>
15. <https://github.com/Marnonel6/Guide_dog_Unitree_Go1?tab=readme-ov-file>
16. <https://github.com/unitreerobotics/UnitreecameraSDK>
17. <https://www.youtube.com/watch?v=UXb2exh6yqE&list=PLxQx8K02En6sgTLxZVxIUQAUF_95n6Xev&index=3>