GLOBAL RESEARCH IMMERSION PROGRAM FOR YOUNG SCIENTISTS

3-FINGER EXOSKELETON TO DRONE

BACKGROUND

There are five significant open-source software frameworks: CHAI3D, ROS Noetic, PX4, Gazebo 11, and QGroundControl. CHAI3D creates haptic, visual, and auditory simulations for interactive touch-based experiences, essential for medical simulation, robotics, and virtual reality. ROS Noetic Ninjemys, the latest LTS release for Ubuntu 20.04, offers comprehensive tools for developing robust robotic applications with enhanced Python 3 support and improved packages. PX4, an advanced flight control software for drones and unmanned vehicles, provides a scalable platform for flight control and mission planning, often used with Gazebo 11, a high-fidelity robotics simulation software. QGroundControl, a ground control station software, facilitates UAV management with features like mission planning and real-time telemetry, offering a versatile tool for both professional and hobbyist drone operators. The research objective is to construct and operate a physical drone, simulate its environment using PX4, Gazebo 11, and QGround Control, and control the simulated drone with exoskeleton hand signals.

ROS, PX4, GAZEBO 11 & SIMULATIONS

To start, we downloaded ROS1 from the ROS Wiki and familiarized ourselves with its fundamentals through a series of twelve YouTube tutorials provided by Robotics-Back End. Initially, the turtle simulation enables the turtle to travel in straight lines, creating squares, according to the arrow keys but we learned to modify the turtle's movement to circles as well as adjusted its ROS services to change the color of the line it draws from white to red. Following this, we downloaded QGroundControl, PX4, and Gazebo 11 from Professor Li Shuo's Post Graduate student's GitHub repository. This process involved navigating through some learning curves, including two minor errors related to directory paths and ROS configurations. After successfully downloading the applications, we studied and modified MAVROS drone code to learn its functionalities. As a result, we developed a Python script called `tt.py` that maps WASD and [] to control the drone's movements forward, left, backward, right, up and down respectively. We used this file as a sample in developing `drone_move_new.py`, the final python listener in the exoskeleton-hand-to-simulated-drone process, in addition to completing the drone portion of the bridge Python script, `states_new.py`, designed to connect the drone's Python-based control system with the exoskeleton's C++ signal processing.

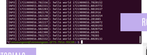
VIDEOS

HTTPS://PAN.BAIDU.COM/S/1AGH55754RAQCEDOELBGWNW

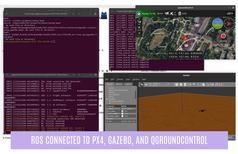
PHOTOS


ROS TURTLE SIMULATION


LAUNCHED PX4 AND GAZEBO


ROS SUBSCRIBER


COMPLETED ROS TUTORIALS


ROS CONNECTED TO PX4, GAZEBO, AND QGROUNDCONTROL

CHAI3D

Xu Peisen and Bao Yang Bin assisted with modifying Su Yuan's exoskeleton code to focus on the positional data, namely the XYZ coordinates, rather than the angles of the three fingers, creating the `position_publisher.cpp` file. We then completed the exoskeleton portion of the `states_new.py` script that listens to the C++ file and subscribes the True or False of each finger's X, Y, Z flags. These flags represented the greatest change in a certain coordinate axis; the thumb used flag Y, while both the index and middle finger utilized flag X. True`indicated movement, while False`meant no movement. However, it involved extensive debugging because our threshold and range values were off. We experimented with the exoskeleton and gathered the following data:

	Thumb	Index	Middle
Extended - 2	0.707	0.648	0.500
Threshold 2	0.370	1.330	1.400
Initial - 1	0.248	1.167	1.280
Threshold 1	0.120	0.850	0.850
Closed - 0	0.040	1.503	1.650

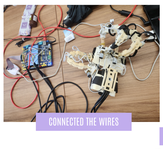
The finger can be in one of three states: extended or stretched, initial or resting, and closed. The thresholds were set up in the following order: "0" is less than Threshold 1, which is less than "1", which is less than Threshold 2, which is less than "2". Additionally, a True/False command was integrated at the beginning and within the if-else statements of `state_new.py` to link with the `drone_move_new.py` script, facilitating drone movement in all six directions.

VIDEOS

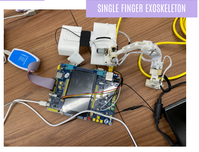
HTTPS://PAN.BAIDU.COM/S/1AGH55754RAQCEDOELBGWNW

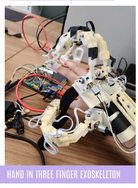
PHOTOS


STM32 FOR THE EXOSKELETON


CONNECTED THE WIRES


THREE FINGER EXOSKELETON


SINGLE FINGER EXOSKELETON


HAND IN THREE FINGER EXOSKELETON

QUADCOPTER BUILDING

The assembly process of the drone began with securing the flight controller unit (FCU) and the electronic speed controller (ESC) to one end of the carbon fiber drone body. Next, the motor, battery, and receiver wires were meticulously soldered to the ESC board, ensuring robust electrical connections. Two sets of black and red wires were then soldered to connect the power distribution board (PDB) and the voltage reduction module to the FCU. Each wire was then carefully attached to its respective component to ensure proper functionality. The drone system was set up on Betaflight, and initial tests were conducted to verify the operation of the motors. 3D printed vertical mounds were constructed to hold three motion capture reflective spherical markers. On the opposite end of the drone's body, the PDB, an additional carbon fiber base, and the battery adhered with a motion capture reflective spherical marker were securely nailed in place like a hamburger. Propellers were then added and fastened, and to maintain organization, the wires were taped neatly. Following this, the battery and receiver were connected, and a thorough motor direction test was conducted to confirm that opposite corners of the drone rotated in matching directions and that same sides rotated in opposite directions. The quadcopter was then linked to a physical controller for flight.

VIDEOS

HTTPS://PAN.BAIDU.COM/S/1AGH55754RAQCEDOELBGWNW

PHOTOS


FCU


ESC


CONNECTED THE WIRES


ROS CODE TO INITIATE THE PYTHON FILE TO FLY


SCREWING IN THE MOTORS


HAMBURGER ALIGNMENT


COMPLETED QUADCOPTER


MOTOR TESTS VIA BETAFLIGHT


NEATLY TAPED UP

CONCLUSION

After launching ROS, opening QGroundControl, initiating PX4, accessing `position_publisher_new.cpp`, opening `states_new.py` and `drones_move_new.py`, and monitoring the rostopic `/mavros/local_position/pose`, the exoskeleton and drone simulation is finally connected. Thus, we successfully integrated hand signal inputs from the thumb, index finger, and middle finger of the three-fingered exoskeleton with the Robot Operating System, enabling data transmission to PX4 and QGroundControl. This setup facilitated the precise control of a simulated drone within the Gazebo environment. Additionally, we constructed a functional drone capable of manual flight.

VIDEOS

HTTPS://PAN.BAIDU.COM/S/1AGH55754RAQCEDOELBGWNW

AUTHOR INFORMATION

Contact & Affiliation:

1. Stephanie Loo – UC Berkeley - yanceeloo@berkeley.edu
2. Seungeon Lee – NTU Singapore - seungeon001@e.ntu.edu.sg

Host University: Zhejiang University

KEY REFERENCES

1. Robotics Back-End. "Intro: Install and Setup ROS Noetic - ROS Tutorial 1 (ROS1)." YouTube, 17 Jan. 2022, www.youtube.com/watch?v=Qk4vLFhvfbI&list=PLLSegLrePWgIbIrA4iehUQ-impvlXdd9Q&index=1.
2. "ROS/Tutorials - ROS Wiki." Wiki.ros.org, wiki.ros.org/ROS/Tutorials.
3. "Drone_build_tutorial/2_1_Simulation_ROS_PX4.Md at Main · EEEManchester/Drone_build_tutorial." GitHub, github.com/EEEManchester/drone_build_tutorial/blob/main/2_1_Simulation_ROS_PX4.md.
4. "MAVROS_sample_code/Mavros_circle.py at Main · Maponarooo/MAVROS_sample_code." GitHub, github.com/maponarooo/MAVROS_sample_code/blob/main/mavros_circle.py.
5. "MAVROS_sample_code/Mavros_square.py at Main · Maponarooo/MAVROS_sample_code." GitHub, github.com/maponarooo/MAVROS_sample_code/blob/main/mavros_square.py.
6. Zhou Jin's Drone Code
7. Su Yuan's Exoskeleton Code