Debris Drone - Robotic Mapping for Dangerous Areas

Kahf Hussain, Luka Bagashvili, Ichhit Joshi, Niranjan Reji

DENISON

Problem

After natural disasters, the debris left behind can make search and rescue very difficult and dangerous. Maps of these areas can help rescuers by showing safe paths and highlighting where people might be trapped. Our goal is to create a robot that can safely move through rubble, identifying hazards as it goes, and be capable of creating 3D maps that show the shapes and sizes of debris piles, making rescue missions faster and safer. This project prototypes the use of robotics in disaster management, aiming to assess damaged areas quickly and safely to support rescue efforts.

Possible Solutions

Some of our initial ideas to for the above robot were:

Drone: Drones quickly cover large areas without navigating rough terrain but are limited by weather conditions and high power requirements.

Ground Walker or Ground Car: Ground robots operate under almost all weather conditions and provide detailed terrain maps, though they are slower and can get trapped.

Train Track: This robot is fast and reliable on predefined paths but cannot operate beyond existing tracks.

Main Challenges and Learnings

1: Calibrate and code the drone's sensors and motors for stable autonomous flight. We used a MAVLink to code the drone from our computer.

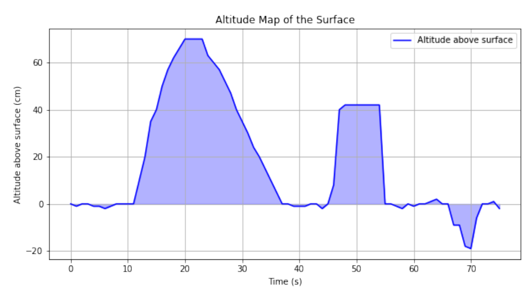
2: Transmit data accurately and efficiently from the Pi Pico to our computer for live plotting. We transmitted data from the Pi to the computer over wifi as a solution to this.

3: Adjust radio signals for reliable communication between the drone and our computer. We used a heuristic method of finding the best method to communicate with the drone along with changing some security parameters for us to directly code it.

4: Testing the drone under various circumstances. Trying it indoor/outdoors, in different weather and different space. Different speed and power to the drone.

5: Creating a ROS network for proper output. Dr. Law and the ROS documentation helped us create this network.

Test Plot



References

[1] Holybro. *PX4 Development Kit - X500 v2*. Parts acquired from this source. Available at: `https://holybro.com/products/px4-development-kit-x500-v2`

[2] *Make Your Own Pixhawk Raspberry Pi Drone in 36 Minutes (2020) | The Ultimate Project Drone*, YouTube, uploaded by The Drone Dojo, 2020. This video was used as a guide for the physical assembly of the drone. Available at: `https://www.youtube.com/watch?v=kB9YyG2V-nA`

[3] Raspberry Pi Foundation. *Raspberry Pi Pico Documentation*. Available at: `https://www.raspberrypi.org/documentation/microcontrollers/raspberry-pi-pico.html`

[4] Pixhawk. *Pixhawk Open Standards for Drone Hardware*. Used for our flight controller's documentation. Available at: `https://pixhawk.org/`

[5] ROS Wiki. *Nodes*. Used for understanding how ROS nodes work. Available at: `http://wiki.ros.org/Nodes`

[6] *QGroundControl*. Used for controlling the drone during initial stages. Available at: `http://qgroundcontrol.com/`

[7] *LiPo Guide for RC Beginners!*, YouTube, uploaded by beyondRC, 2020. This video was referenced for understanding LiPo battery usage for the drone. Available at: `https://www.youtube.com/watch?v=Lk7wzVYmXSA`

[8] *Benewake TF-Luna LiDAR Distance Sensor Manual*, World Drone Market, 2022. This manual was used for proper setup of LIDAR. Available at: `https://www.worlddronemarket.com/benewake-tf-luna-lidar-distance-sensor-manual/`

[9] Dr. Law: *Networking with the Pico.pdf, wifi.py* Used for networking documentation between the computer and pico.

[10] Dr. Law: *ros2_paper.pdf* Used for ROS documentation.

Final Design Concept

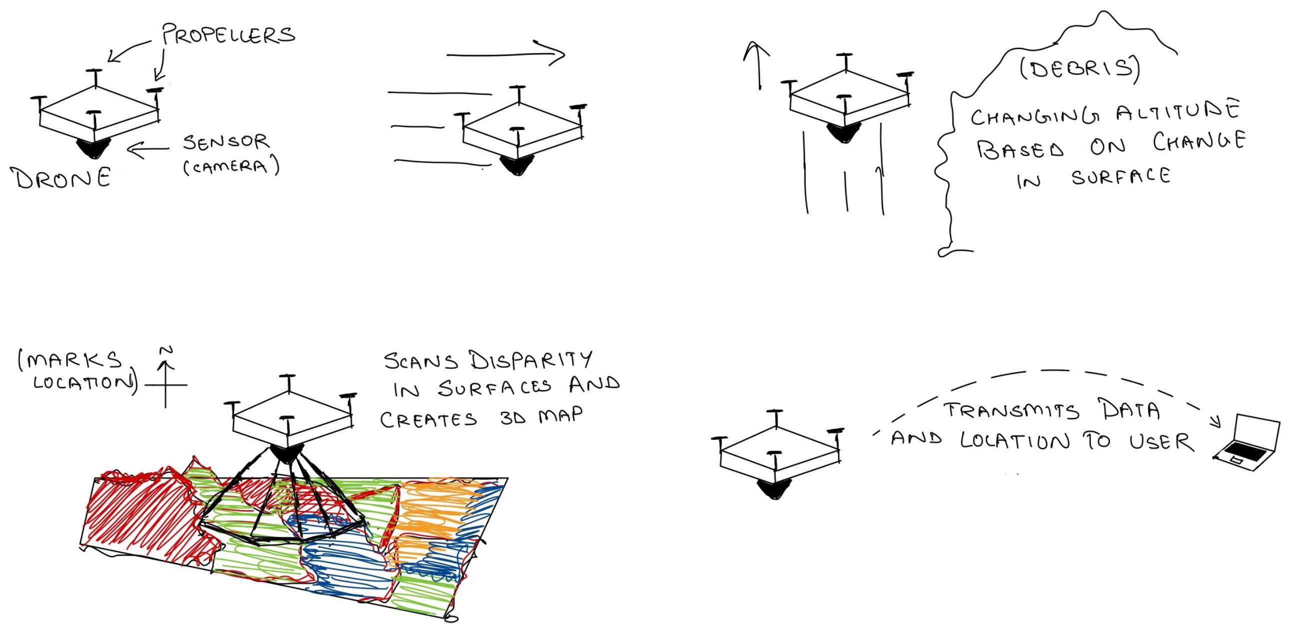
After evaluating the pros and cons, we decided to use the **Debris Drone(DD)** as our final solution. We established the following requirements for the drone to ensure it operates both accurately and efficiently.

Functional requirements for DD:

- Fly autonomously without any collisions at a steady height.
- Scan the surface below it accurately.
- Communicate the data it collects back to our computer to create a detailed map of the debris-covered area.

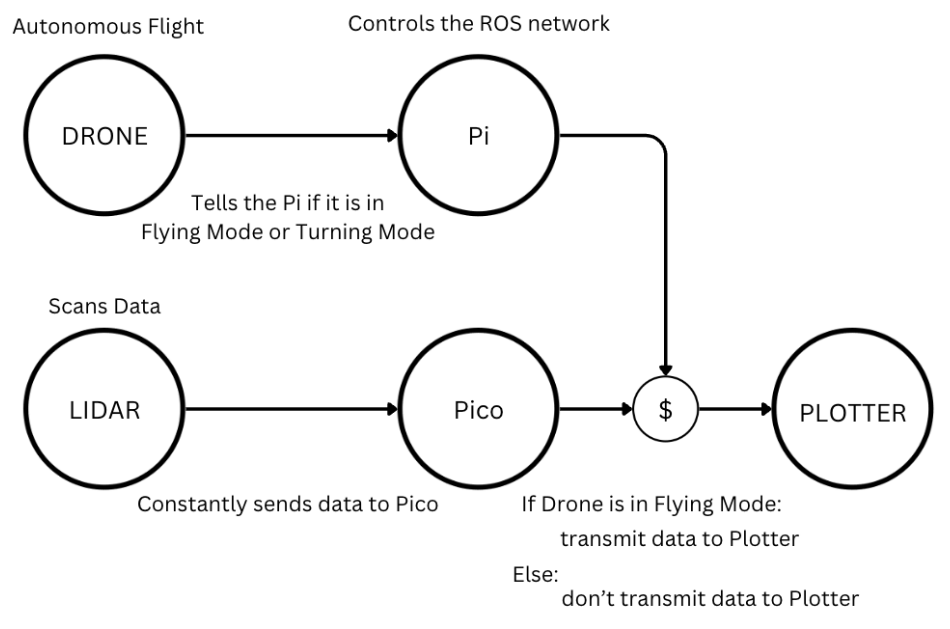
Contextual/Operational Requirements for DD:

- It must navigate through small, enclosed, and cluttered areas safely and accurately.
- It should operate without posing any risks to humans, property, or itself.
- It must be compact and easy to transport, enabling quick setup and deployment in under 5 minutes.
- It should efficiently process and store data with an accuracy of at least 90%.
- It requires sufficient battery life to operate continuously for at least 30 minutes in disaster recovery scenarios.


PROPELLERS
SENSOR (CAMERA)
DRONE
(DEBRIS)
CHANGING ALTITUDE BASED ON CHANGE IN SURFACE
(MARKS LOCATION)
N
SCANS DISPARITY IN SURFACES AND CREATES 3D MAP
TRANSMITS DATA AND LOCATION TO USER

Implementation: System Level

At a systems level, our drone utilizes the LIDAR sensor to continuously scan the environment, relaying data to the Pico for real-time processing. The Raspberry Pi issues commands based on the drone's status—either flying or turning. When in flight mode, the Pi allows the Pico to transmit data to a plotter, which visualizes the flight path live. And if in turning mode, the Pi just drops the data to ensure proper scaling between the actual zone and our created map. This smart interplay ensures the drone operates autonomously, safely navigating its surroundings while providing instant and accurate visual feedback of the rubble zone.


Autonomous Flight
Controls the ROS network
DRONE
Pi
Tells the Pi if it is in Flying Mode or Turning Mode
Scans Data
LIDAR
Pico
$
PLOTTER
Constantly sends data to Pico
If Drone is in Flying Mode:
transmit data to Plotter
Else:
don't transmit data to Plotter