ELEC5308 Project 2: Drone-Based Search and Rescue System for Drowning Detection

Pulin Zhang, Tina Lo, Ellen Lee, Hee Chan Kim, Erick Gomez

School of Electrical and Information Engineering

The University of Sydney, NSW, Australia

I. Introduction

Bathing associated drownings are actually a leading cause of accidental or unintentional death noted globally especially along the shoreline The chances of survival are often hindered by factors such as complex topography of the surrounding environment and inhospitable weather, distance or time taken for a human to reach the scene of the accident. The proposed plan therefore seeks to establish an **Automated Drone-Based Search and Rescue System for Drowning Detection** that would respond to these challenges by servicing within shorter time. For water-based rescue missions, by incorporating autonomous navigation, real-time object identification, digital twin simulation, and voice connection, our system gives a rich solution.

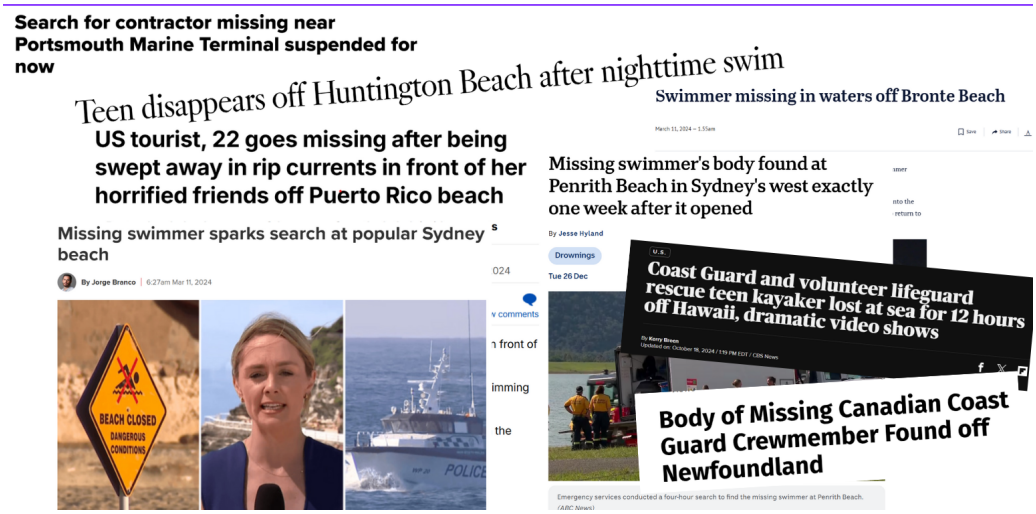

Fig. 1: news about drowning

The project was executed in two phases: The first one is about the simulation phase and the second one is about the actual application of the plan phase. The simulation phase of this study utilised **Webots**, **MAVROS** and **Ardupilot** to affirm the system's capability in a simulation environment; whereas the reality phase involved utilising a real drone with an integral depth camera, a companion computer and a lifebuoy releasing mechanism. This paper

will discuss the details of the implemented system, the technical aspect of the system, difficulties encountered during the project, and the prospects of this endeavor.

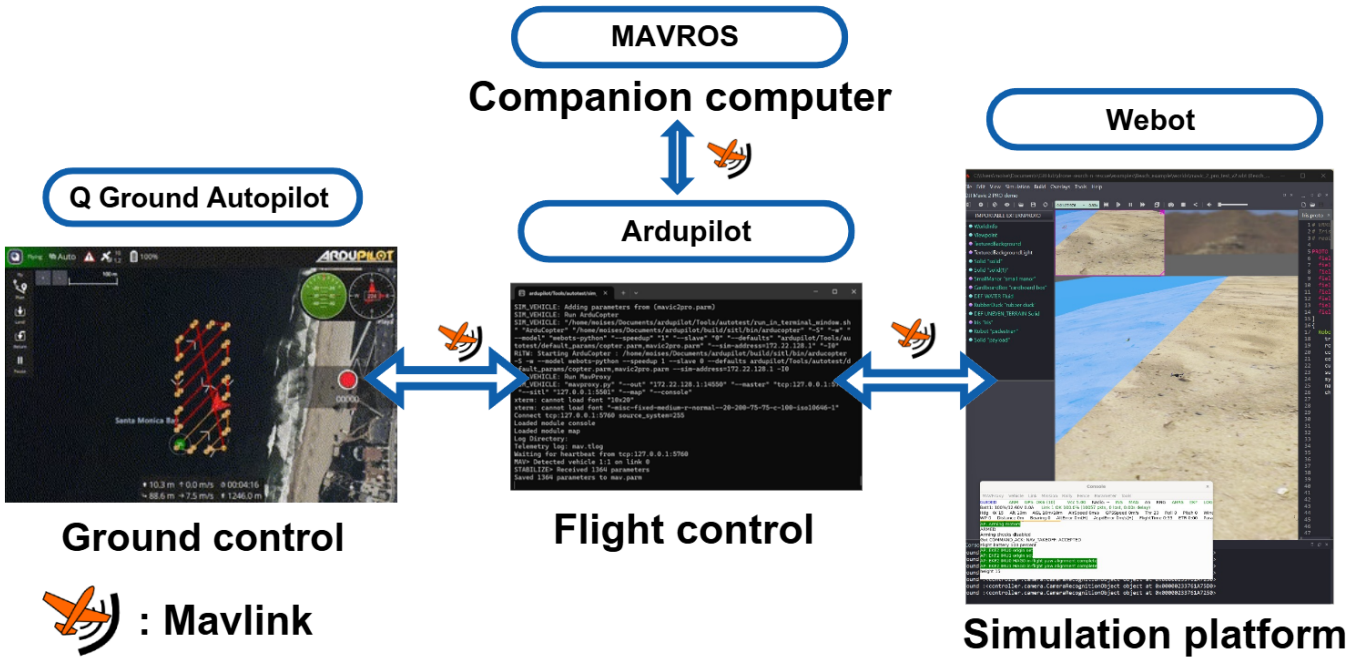

Fig. 2: Phase 1:Simulation

II. System Functionality

The **Drone-Based Search and Rescue System** autonomously detects, navigates, and assists drowning individuals. This section provides a detailed description of the system's functionality in both the simulation phase and real-life applications.

A. Simulation Phase

The simulation phase was conducted using **Webots**, an open-source robotics simulator. In the virtual environment, we created a coastline map and simulated the physical elements involved in a rescue mission. The simulated environment allowed us to test the drone's behavior and refine key functionalities, including autonomous navigation, object detection, and lifebuoy deployment. The integration of **MAVROS** and **Ardupilot** providing control of the vehicle in both modes autonomous or remote control mode, allowing it to receive instructions through commands in the **Mavproxy** such as takeoff, waypoint navigation, and PID value adjustments during flight. Additionally, the system can transmite important information to the user, such as sensor measurements, GPS position, battery level, and point of view of the camera. All commands and data are transmitted via **MavLink**, a lightweight, open-source communication protocol specifically designed for drones. This protocol sends messages defined in XML files, providing an efficient method to exchange data. all data is received and transmitted through **Mavproxy**.

Commands can be transmitted through a console, using written commands or via ground control stations, which provide a more interactive and user-friendly interface. It is necessary in our application to have all the necessary information on hand and is mandatory to be issue instructions to the drone quickly. For this project, we used **QGroundControl**, This ground control display the GPS position, gyroscope measurements, and other relevant drone data. It also allows the creation of search paths within a specific area and update it to the onboard computer.

The outcome of the simulation phase was a successful validation of the core functionality, which helped in the subsequent real-life implementation.

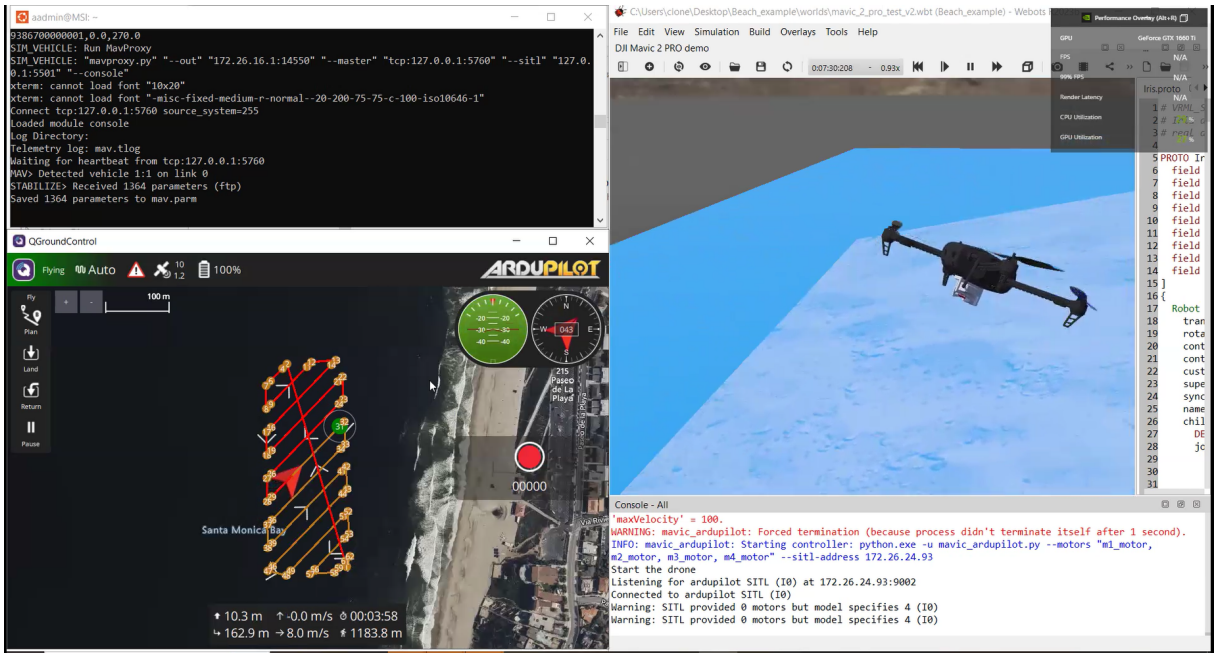

Fig. 3: Simulation environment

Additionally, during the simulation, we implemented a script to trigger our mechanism and utilize computer vision, enabling dynamic adjustment of the drone's instructions.

B. Real-Life Applications

Some important progress toward real-world application of the system was achieved after the validation of the system during the simulation phase. The drone framework was fully installed in the companion computer, **Tinkerboard**. There was also included an advanced depth camera for enhanced environmental perception.

In this phase, the development team was able to deploy the **YOLOv7** object detection model for detecting persons in distress. Moreover, a component for speech recognition was designed for such interaction scenarios as 'yes' and 'no'. The team also used *NERV* to create a model and a mirror image of the system to strengthen the capacities of the general framework by simulating and integrating it with real-world conditions. An additional gripper was developed and integrated with the drone to enrich drones' capability in terms of possible assistive actions.

A substantial amount of experimentation was performed to assess the capabilities of these individual modules in different scenarios that were emulated in different controlled settings. Specifically, this testing phase was carried out to confirm that each module performed optimally from design perspectives, under conditions resembling the real world. While full system integration testing was not done, enough progress was made to show that the project was productive for gathering ideas for future development and doing thorough systems testing.

III. Simulation Details

The simulation was carried out in **Webots**, which enabled us to create and manipulate a 3D model of the drone. The virtual model included sensors, actuators, and simulated environmental elements, allowing us to test and validate the drone's behavior in a controlled environment. **Ardupilot** served as the software that connected **Webots** and QGroundControl, providing seamless command execution and sensor data processing. The primary controller it is managed through **Qgoundcontrol**, one critical feature is it is setup the sensors calaibtatration in ardupilot, which enables more accurant and consistent snensor measurements, ending up in a good and smooth control of the drone. As shown in table 4, the initial telemetry indicates erratic altitude reading for the drone. This problem can be fixed, by calibrating the sensors using the calibration tool available in Qgroundcontrol, as demonstrated in 5. The Telemetry graph shown in figure 6 demonstrate more stability in the drone at the hight of 10 meters after the calibration compared to the initial -5 meters as is visible in figure 4.

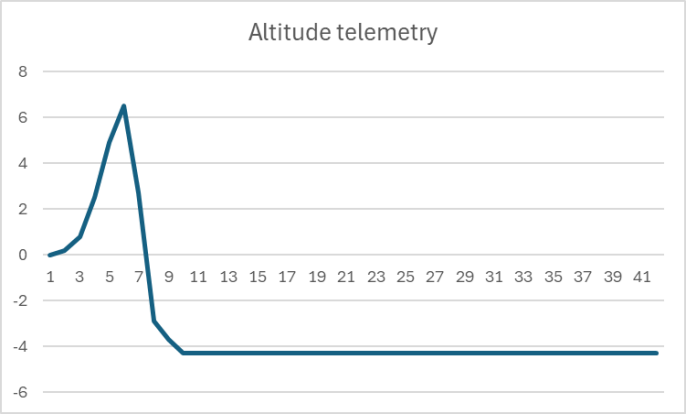

Fig. 4: Telemetry without calibration

In the figure 7 is visible the path created to search individuals along the coast. The path is generated using Qgroundcontrol, which establishes different waypoints in specific order to allow the drone to cover an entire designated area, Additionally can command the drone to interrupt these actions to perform a different one, such as return to home location or to define a new area to cover as is needed by the user and the situation.

Each software had a specific role for the whole simulation. First in Webots we were able to integrate a 3d model of a drone and test different controllers within a real environment due to the physics engine of the software ([**?**]).

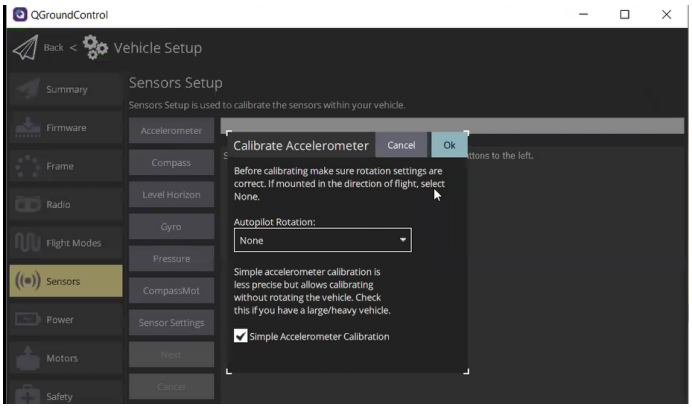

Fig. 5: QGroundcontrol calibration of sensors

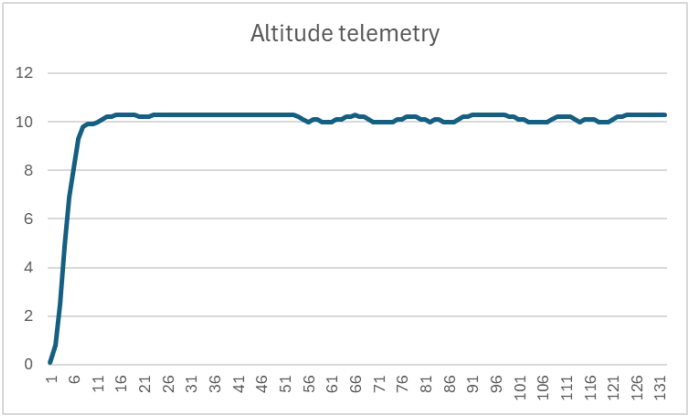

Fig. 6: Telemetry with calibration

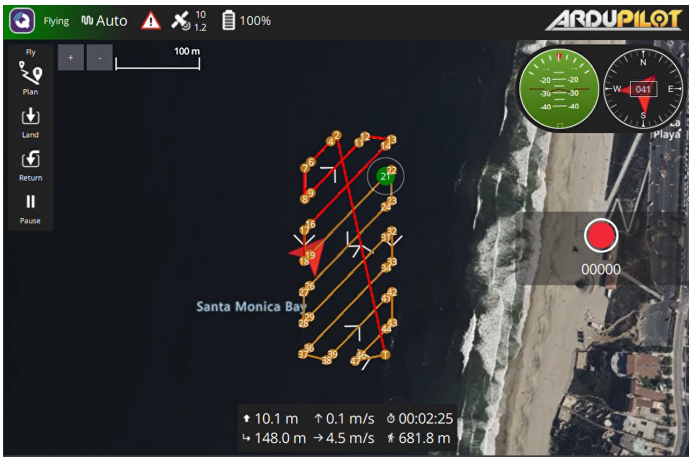

Fig. 7: Path

Two different drones were tested in Webots for experimenting with different drones and controllers. In the software, each drone is called a *proto* ([**?**]). Additionally, to each proto a camera, a gripper, an IMU, a GPS and a gyroscope were added to have fully devices for the application of search and rescue.

It's worth mentioning that each component added to the drone was already built in the software with the exception of the gripper. The gripper was design, imported to the software and tested to validate its functionality making

Webots a powerful software for future custom simulations due to the capability of adding any 3d model to the software. The gripper added was designed in Fusion 360 and uses a crank bar mechanism for holding and releasing the load. The final design is shown in Figure 8.

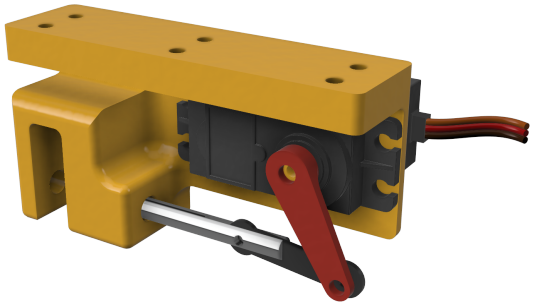

Fig. 8: 3D model of the drop mechanism

For controlling the drone two different controllers were implemented. The first one was Ardupilot and QGround-Control is the user friendly interface. This controller requires some data packed in a flight dynamics model, FDM, structure to properly compute the thrust, lift and drag of the drone ([**?**]). The information in the FDM structure includes time stamp, gravity, acceleration that is equal to zero due to the drone missing an accelerometer, IMU, GPS velocity and GPS position. The information was packed with the following lines of code:

```
    def _get_fdm_struct(self) -> bytes:
    """
    Form the Flight Dynamics Model struct (aka sensor data)
    to send to the SITL

    Returns:
        bytes: bytes representing the struct to send to SITL
    """
    # get data from Webots
    i = self.imu.getRollPitchYaw()
    g = self.gyro.getValues()
    a = [0,0,0]
    gps_pos = self.gps.getValues()
    gps_vel = self.gps.getSpeedVector()

    # pack the struct, converting ENU to NED (ish)
    # https://discuss.ardupilot.org/t/copter-x-y-z-which-is-which/6823/3
    # struct fdm_packet {
    #     double timestamp;
```

```
    #     double imu_angular_velocity_rpy[3];
    #     double imu_linear_acceleration_xyz[3];
    #     double imu_orientation_rpy[3];
    #     double velocity_xyz[3];
    #     double position_xyz[3];
    # };
    return struct.pack(self.fdm_struct_format,
                       self.robot.getTime(),
                       g[0], -g[1], -g[2],
                       a[0], -a[1], -a[2],
                       i[0], -i[1], -i[2],
                       gps_vel[0], -gps_vel[1], -gps_vel[2],
                       gps_pos[0], -gps_pos[1], -gps_pos[2])
```

The second controller written in Python was directly loaded to the drone and logic of when to switch to the second controller relies on the camera on board detected a person. So, before the camera detects a person the drone is controlled by Ardupilot and flying according to the flight planned by QGroundControl. Whenever the camera detected a person the second controller came into action overwriting the values sent by Ardupilot to the drone with the values computed using a PID as an orientation controller and an Exponential decay controller for the position.

First, when the person is recognize by the camera the controller uses the centroid which comes from the bounding box for the detected person in every single frame. Since the position of the person is relative to the drone, the drone defines the person's reference frame relative to the information capture by the camera plus the frame of the camera relative to the local reference frame of the drone as shown in Figure 19.

The controller to approach the recognized person is explained next.

The *Orientation Error Calculation* for the x and y directions is calculated by comparing the centroid of the detected person in the image to the center of the image. Some information from the camera is required for computing these errors, the field of view (FOV), the width in pixels, and the height in pixels. For this particular simulation, the FOV was 0.785 for x and y, and the size of the image was 400x240 pixels. The equations that describes the errors are:

$$e_{Ox} = \frac{u - W/2}{W} * \theta_x \tag{1}$$

$$e_{Oy} = \frac{u - H/2}{H} * \theta_y \tag{2}$$

where:

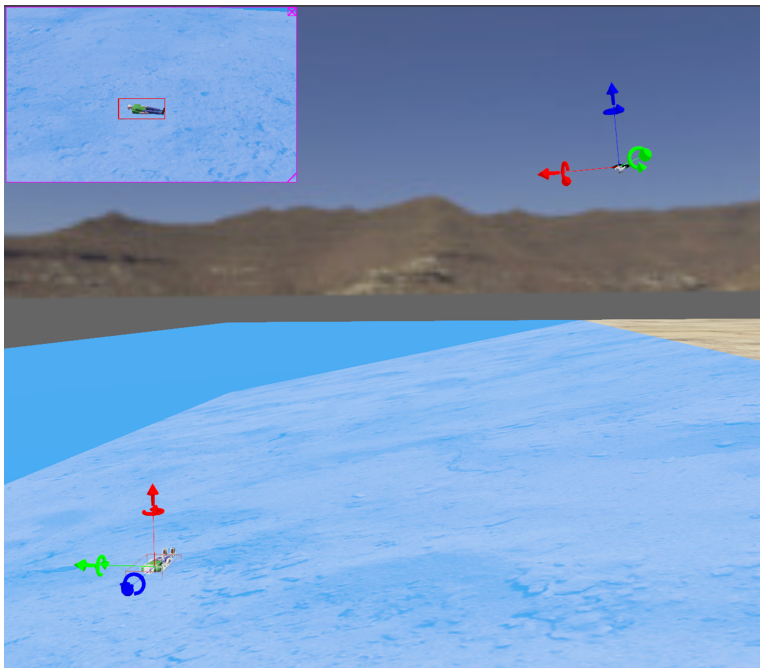

Fig. 9: Person reference frame relative to local drone reference frame

- u, v are the coordinates of the object's position in the image.
- W, H are the width (400 pixels) and height (240 pixels) of the image.
- θ_x, θ_y are the FOVs in radians (0.785).

Additionally, the focal length is required to estimate the distance of the drone to the person ([**?**]), thus the following equation was used to determine the focal length in the y axis.

$$f_y = \frac{H}{2tan(\frac{\theta_y}{2}} \tag{3}$$

The estimated person position only consider the y-axis position and the error in the x direction was not included. This omission is due to the orientation controller being responsible for aligning the camera with the person, and the camera's yaw motion being locked. Thus, if the camera is aligned with the person, the drone will also be aligned. The person was also treated as a point to simplify the equation and because each person has a different height. Therefore the estimated person position's equation is:

$$obj_y = \frac{1.2 * f_y}{v} \tag{4}$$

The error for the y-axis positions is calculated as follow:

$$e_y = \frac{obj_y}{tan(e_{Oy}} \tag{5}$$

and because the x-axis error was omitted, the total position error (e_p) is:

$$e_p = e_y \tag{6}$$

Finally the *linear speed controller* uses an exponential decay function to calculate the disturbance for the drone. The reason for using an exponential decay function is that it has a faster response than a PID. The equation for the input for the drone is:

$$pitch\_disturbance = -1 * v_o * (1 - e^{-\alpha * e_p^2}) \tag{7}$$

where v_o is the base velocity constant for the drone and α is a tuning parameter to tell how fast or slow the controller should response.

The *angular speed controller* adds inputs to the yaw velocity of the drone and the pitch velocity of the camera. These controllers are based on a PID which is a proportional, integral and derivative controller ([**?**]). The controllers used are shown next:

$$yaw\_disturbance = -(k_{px} * e_{Ox} + k_{ix}\frac{E_{Ox}T}{1000} + k_{dx}\frac{eO_{Dx}1000}{T}) \tag{8}$$

$$pitch\_camera\_disturbance = k_{py} * e_{Oy} + k_{iy}\frac{E_{Oy}T}{1000} + k_{dy}\frac{eO_{Dy}1000}{T} \tag{9}$$

where:

- $k_{px}, k_{ix}, k_ddx, k_{py}, k_{iy}, k_{dy}$ are the PID gains for drone's yaw and camera pitch disturbance.
- $E_{Ox} and E_{Oy}$ are the cumulative erros for itnegral components.
- $eO_{Dx} and eO_{Dy}$ are the derivative errors.
- T is the time step in milliseconds

The cumulative error and previous error terms are updated as:

$$E_{Ox} = E_{Ox} + e_{Ox}$$

$$E_{Oy} = E_{Oy} + e_{Oy}$$

$$eO_{Dx} = e_{Ox} + eO_{1x}$$

$$eO_{Dy} = e_{Oy} + eO_{1y}$$

$$eO_{1x} = e_{Ox}$$

$$eO_{1y} = e_{Oy}$$

The implementation of the second controller is the following.

```
    # Orientation error
            eOx = (object.getPositionOnImage()[0]-400.0/2.0)/(400.0)*0.785
            eOy = (object.getPositionOnImage()[1]-240.0/2.0)/(240.0)*0.785

            # Exponential controller with PID
            # y focal length
            fy = 240/(2*math.atan(0.785/2)) #fov = 0.785, height = 240 pixels

            obj_y = 1.2*fy*object.getPositionOnImage()[1];
            e = [0, obj_y/math.tan(eOy)]

            # Position error
            eP = math.sqrt(pow(e[0],2)+pow(e[1],2))


            # Angular speed controller
            eO_Dx = eOx - self.eO_1x
            eO_Dy = eOy - self.eO_1y

            self.EOx = self.EOx + eOx
            self.EOy = self.EOy + eOy

            # Linear speed controller
            kP = self.v0*(1-math.exp(-self.alfa*pow(eP,2)))/eP

            pitch_disturbance = -1.0*kP*eP

            yaw_disturbance = clamp(-1.0*(self.kpOx*eOx + self.kiOx*self.EOx*self._timestep
    /1000.0 + self.kdOx*eO_Dx*1000.0/self._timestep),-1.3,1.3)
            camera_yaw_disturbance = clamp(1.0*(self.kpOy*eOy + self.kiOy*self.EOy*self.
    _timestep/1000.0 + self.kdOy*eO_Dy*1000.0/self._timestep),-1.5,1.5)

            self.eO_1x = eOx
            self.eO_1y = eOy
```

IV. Real-life applications

A. Hardware Components

The real-life implementation of the system involved several hardware components that worked together to accomplish the rescue mission.

1) Depth Camera: The **Depth Camera** captures three-dimensional spatial information about the environment, which is crucial for digital twin modeling and obstacle avoidance. Accurate calibration of the depth camera was essential to ensure that distance measurements were correct during flight.

Fig. 10: Depth Camera

2) Companion Computer: The **Tinkerboard** companion computer with Debian 11 is responsible for handling the computational tasks of processing depth camera data, running the model for object detection, and controlling the lifebuoy release mechanism. It acts as the central hub for decision-making, integrating data from various sensors to autonomously control the drone's behavior.

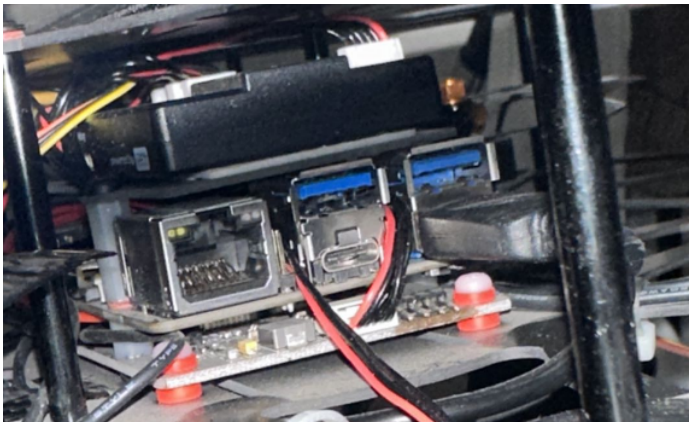

Fig. 11: Tinkerboard

3) Lifebuoy Release Mechanism: The **Lifebuoy Release Mechanism** is an electromechanical system comprising a servo motor that releases the buoy when a drowning person is detected. The deployment mechanism was tested extensively to achieve precise and consistent buoy releases.

Fig. 12: Lifebuoy Release Mechansim

B. Software Components

The software components are responsible for autonomous control, real-time detection, and situational awareness, all of which are crucial to the success of the rescue mission.

1) MAVROS and Ardupilot: The **MAVROS** and **Ardupilot** combination is used to control the drone autonomously. **MAVROS** provides a bridge between the Robot Operating System (**ROS**). Commands such as takeoff, landing, and waypoint navigation are issued by the ROS environment and executed by **Ardupilot**. **Ardupilot**'s sophisticated autopilot algorithms are employed to maintain the stability of the drone throughout the mission.

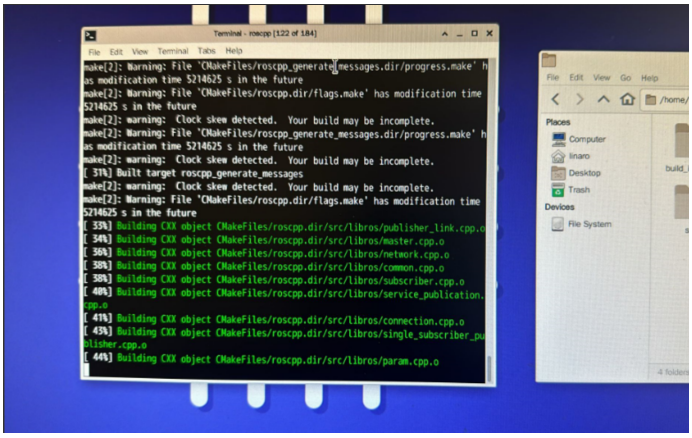

Fig. 13: Ros installation

2) Computer Vision: The **YOLOv7** model was trained using a dataset comprising images representing different drowning scenarios. During the training process, data augmentation techniques were used to improve model

generalization across various conditions. The model is specifically designed to be used in real time inference where the video signal is fed to a connected computer. Such a scenario makes processing and analysis possible while the model identifies drowning cases, from video archives that are fed to the system in real-time [12].

In collecting the dataset, a lot of caution has been taken to ensure that it included images of different subject engaging in either drowning or swimming in water. This dataset was further divided into training, validation, and test datasets in the ratio of 70/20/10, and contains the following images: 5070 images for training set, 1478 images for validation set and 747 images for the test set [2]. For the YOLOv7, the model training was performed

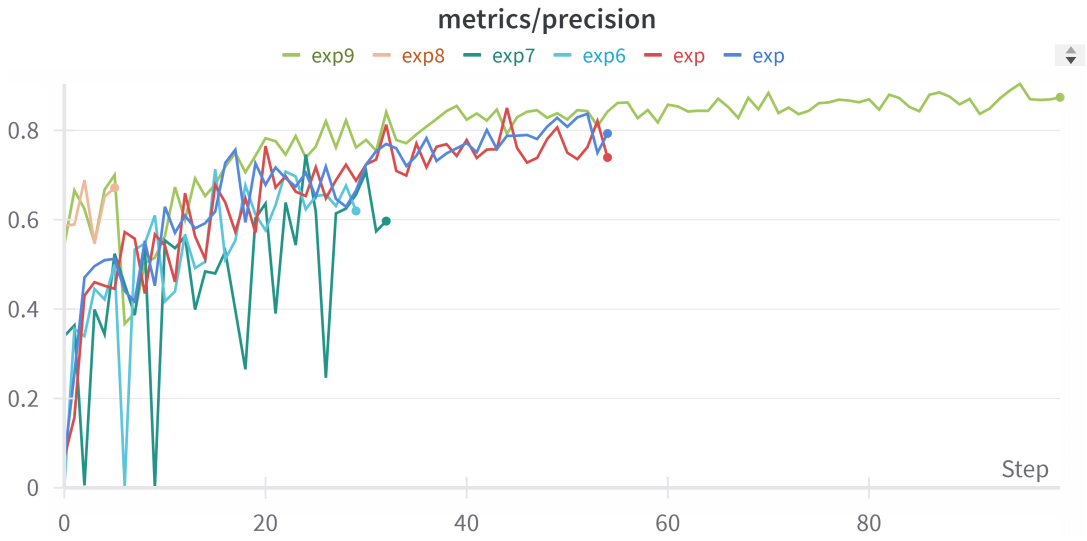

Fig. 14: Precision metrics over training steps for different models

on Google Colab, an A100 GPU as well with the configuration aimed at maximizing both precision and recall. Hyperparameters included learning rate which were set at 0.01 while weight decay was set at 0.0005 and the initial confidence threshold of the last layers was also initially set at 0.65. Especially, the model was trained at 100 epochs and the image size is 640 pixels to accommodate the model performance and computational complexity simultaneously. Batch size of 16 was determined to meet the model's computational need within allowable memory space while achieving reliable gradient estimates [3].

These parameters together with real-time augmentations were helpful to generalize the model across different lighting and environment conditions necessary for real time rescue operations. The arrangement allowed actual video signals of the rescue site to reach the connected computer where detection takes place. This configuration makes it efficient and also reliable especially for the life-saving deployment in an unparable water environment.

After many training and tuning, the model succesfully managed to achieve the validation accuracy of about 90%, which means that model accurately identifies drowning in the form of RWD without false positives. Also, precision with of 0.84 was also obtained and increased gradually from epoch to epoch, demonstrating precise detection by

the model [12].

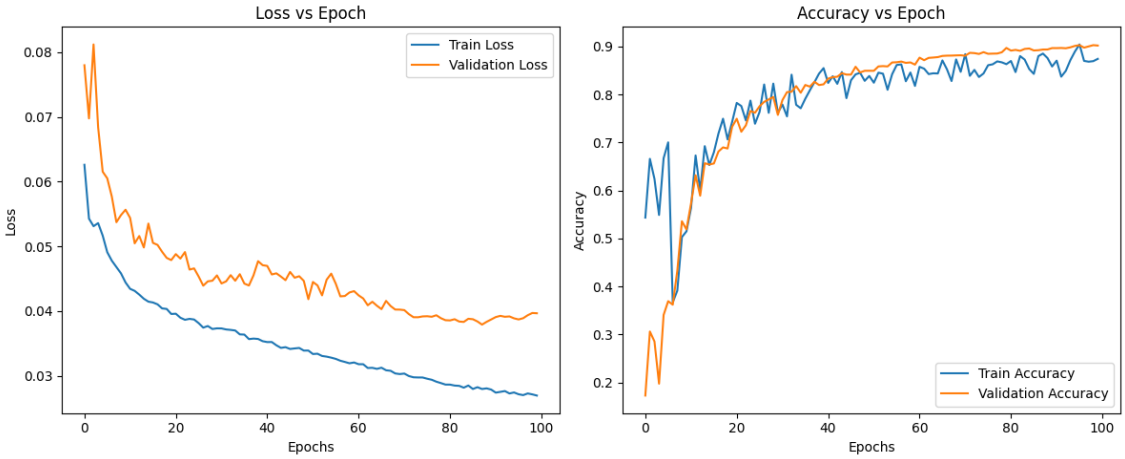

Fig. 15: Loss and accuracy vs. epochs.

Upon applying the model, footage trials which can aid in the real-time object detection tests were carried out using video stream input. In the live tests the model achieved significant accuracy and available detection confidence scores for drowned cases were above 0.8, as shown in figure X. Also, for better visualization of detection on output, we set the confidence threshold to 0.65 and it was more stable to keep the boxes and the detection continuing smoothly in the video. This real time targeting is critical in speedy response because the system allows for real time location of the distressed persons [2].

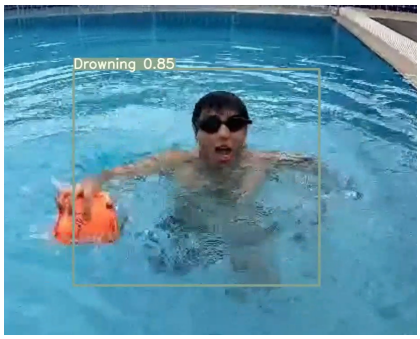

Fig. 16: Drowning detection with 0.85 confidence.

Fig. 17: Sample frames from the validation set.

Conclusively, the application of YOLOv7 for the computer vision subproblems in the Drone-Based Search and Rescue System indicates a great leap in automated detection systems. Using this model, the system offers a faster and more efficient method of locating endangered individuals and thus improves the generality of rescue operations [12].

3) Digital Twin: The **Digital Twin** model primarily focuses on modeling the environment and targets by creating a real-time virtual representation using data from the depth camera. By utilizing the instant NGP (Neural Graphics Primitives) binary, the digital twin provides high-resolution visual information to assist rescue personnel. This

enhances situational awareness, allowing the drone to make informed decisions and dynamically adjust its flight path during rescue missions, effectively supporting the search and rescue process.

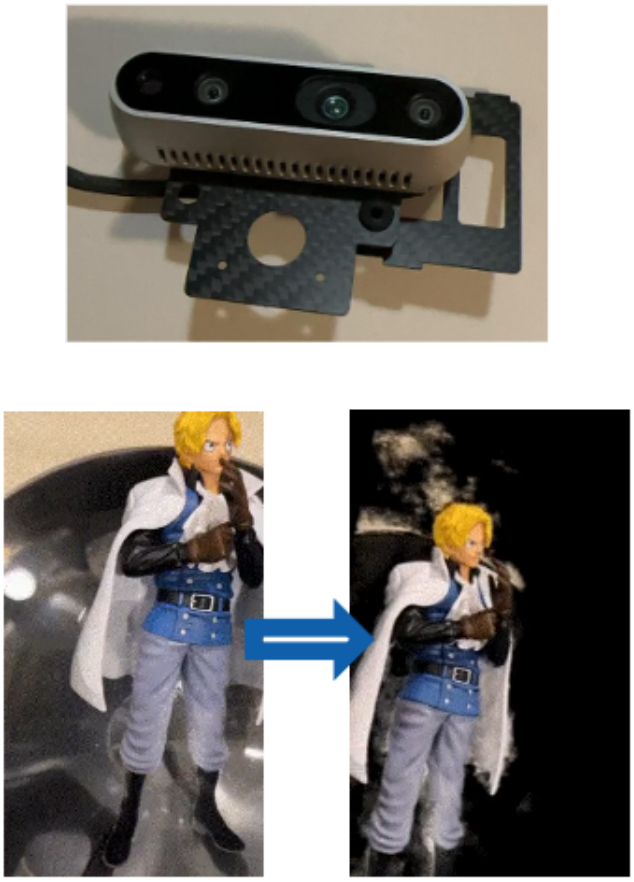

Fig. 18: Digital twin trough nerfstudio

4) Voice Communication System: The **LMNT Text-to-Speech API** was integrated into the system to provide comforting messages to individuals in distress. The voice communication system aims to provide psychological support, reducing panic and contributing to the effectiveness of the rescue mission [4].

```
[{'id': 'cd10855c-e2c3-4163-8efd-65fd1d249c70',
  'name': 'ellen',
  'owner': 'me',
  'starred': False,
  'state': 'ready',
  'type': 'professional'},
```

Fig. 19: LMNT voice ID configuration.

Connection to the LMNT API was made with an API key that was provided to enable the application to query

a few carefully chosen voices from the LMNT platform. We customized the voice model with our own voice thus it can produce a calm and clear tone. Once an appropriate voice ID is selected, the system produces audio files for individual rescue prompts. Such prompts are the first messages to comfort the client, the questions to evaluate the state of the client, and the following answers containing the detected keywords.

The TTS output produced again by the LMNT API using the synthesize function is then stored as an audio file as for instance output_tts.wav before being played on the speaker system of the drone. This kind of experience creates a convenient and continuous conveyance process which directly meets the psychological state of those in distress and makes the process of rescue more effective.

Then, Yes_no_classifier.joblib was used to classify the user response as either ”yes” or ”no”. This is to incorporates the use of an Auto response system through voice recognition together with text to speech (TTS) during emergencies The classifier uses Mel-Frequency Cepstral Coefficients (MFCC) obtained through librosa and the spoken response [5], and was vital for decision making in real time depending on user input provided. The “yes” responses identified leads the system to ask more questions to retrieve relevant needs from individuals in distress.

The TTS features which use the LMNT API, provide concise audio output such as the voice message, ‘Are you in any form of danger?’ and ”Any items in need?” to assist the users in going through certain response patterns. Audio is recorded using a script integrated into the browser using JavaScript [5], converted into.wav format using FFmpeg and analyzed [5]. This arrangement enables the system operates optimally within even the most confined environment, such as browser-based frameworks.

5) Voice recognition: In this section, to recognize the answer is yes or no, we use a random forest classifier to train the model. Moreover, combine with speech-to-text API to record the item that the victim needs.

Random forest classifier is a machine learning algorithm to classify the data to different classes, and it consists of many small decision trees that vote together to determine the final result [6]. This design allows for more accurate and stable results with fewer errors. Fig. 20 demonstrates the process of how random forest classifier works. First, the data set X is randomly sampled to generate different subsets of data, each of which will be used to train a decision tree. This is called bagging [7]. Each decision tree builds a tree structure based on a subset of the samples it receives and randomly selected features to learn patterns in the data through multiple dichotomies (e.g., yes/no judgements). Then, the final result was determined by a majority vote of all trees.

Since random forest cannot directly be trained and dealt with audio file such as wav, it is necessary to extract the audio files to different features. The goal for this machine training, is to let the machine recognize ”yes” or ”no” audio, which is a short sentence. Therefore, we do not need a really complex algorithm. In this project, we simply use MFCC (Mel Frequency Cepstral Coefficents), chroma feature and spectral contrast to get the features and feed into random forest.

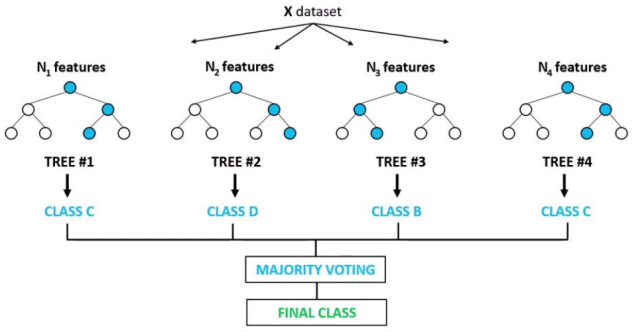

Fig. 20: Random Forest Classifier [6]

- **MFCC:** A powerful tool for audio classification, that uses MEL scale to divide the frequency and extract the coefficient with discrete cosine transform (DFT) [8]. It can mimic the frequency perception of the human ear, making it effectively captures the key messages in speech. As assessed at different times, each coefficient shows the proportion of a speaker's voice output that falls within a specific frequency range [9].
- **Chroma feature:** One of the common feature extraction [10]. Since different individual has different tune of voice, and different words may have different frequency distributions. The chroma feature can help the model capture these differences in frequency distributions.
- **Spectral contrast:** This is a measure of audio energy between the highest and lowest value [11]. Enhances the ability to recognize subtle sound changes and helps to distinguish speech characteristics in different tones or background noise.

All these feature can be called in the librosa library. Before starting to extract the file, we first detect if there is any file that is too silent and remove it, to increase the accuracy.

```
import os
import librosa
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
import joblib

# Paths to folders containing "yes" and "no" audio samples
yes_path = 'Yes'
no_path = 'No'

def is_silent(audio, threshold=0.02):
    """Check if the audio is mostly silent based on a threshold."""
    return np.max(np.abs(audio)) < threshold

```

```

# Function to extract MFCC features from an audio file
def extract_features(file_path):
    audio, sr = librosa.load(file_path, sr=16000)

    # Check if the audio is silent to avoid errors in chroma feature extraction
    if is_silent(audio):
        print(f"Warning: {file_path} is silent or too quiet. Skipping chroma extraction.")
        chroma_mean = np.zeros(12)  # Filler for chroma features if audio is silent
    else:
        # Extract Chroma features only if audio is not silent
        chroma = librosa.feature.chroma_stft(y=audio, sr=sr)
        chroma_mean = np.mean(chroma.T, axis=0)

    # Extract MFCC and Spectral Contrast features
    mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=5)
    mfccs_mean = np.mean(mfccs.T, axis=0)

    spectral_contrast = librosa.feature.spectral_contrast(y=audio, sr=sr)
    spectral_contrast_mean = np.mean(spectral_contrast.T, axis=0)

    # Concatenate all features
    features = np.concatenate([mfccs_mean, spectral_contrast_mean, chroma_mean])
    return features
```

After all the audio features are combined, we then split the data into two section, training and validation with the percentage of 80% and 20% respectively. After the preparation, we started our training.

```
# Train a Random Forest Classifier with additional regularization
clf = RandomForestClassifier(
    n_estimators=10,
    max_depth=2,
    min_samples_split=10,
    min_samples_leaf=5,
    random_state=42,
    class_weight='balanced'
)
clf.fit(X_train, y_train)
```

In order to avoid overfitting, we have to choose the best number of trees, the estimators, to get the best training outcome. Therefore, we range and plot the accuracy of different number of trees with the sklearn package. According to Fig. 21, the result will be the best with around 10 trees, since there is no significant improvement after. Therefore,

we decided to choose 10 trees as our training model. From sklearn package, we can generate the accuracy, the classification report as well as the confusion matrix. In Fig. 22, we can see that the accuracy is around 94%, since we split the dataset to 80-20, that means it should have 80 out of the total of 400 data be tested. Moreover, we got the result of f1-score with around 94%, and over 92% of precision in of the performance, and thus, the result is relatively good.

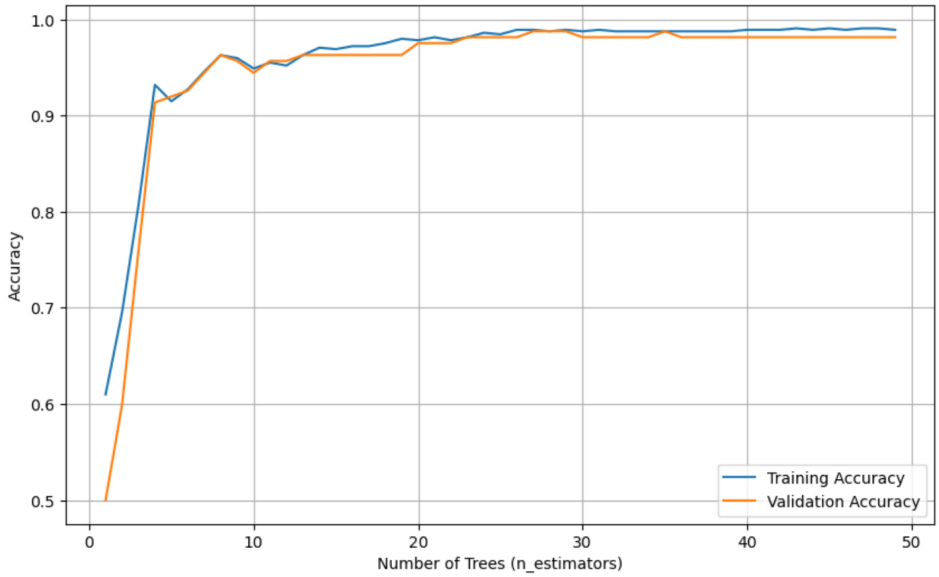

Fig. 21: Accuracy result

```
Validation Accuracy: 94.44%
Classification Report:
              precision    recall  f1-score   support

          No       0.97      0.91      0.94        77
         Yes       0.92      0.98      0.95        85

    accuracy                           0.94       162
   macro avg       0.95      0.94      0.94       162
weighted avg       0.95      0.94      0.94       162

Confusion Matrix:
 [[70  7]
 [ 2 83]]
```

Fig. 22: Classification report

After the training part, we next going to have a conditional statement, that if the person needs help, we then ask them what they need. Otherwise, no further action is needed, and the result will be sent back to the rescue team for them to prepare the things they mentioned. Below is the flow diagram of how the system work. We use speech-to-text API to record the thing they need and record it since the content of this is completely random, and it will be really complicated to train our own model. Therefore, we chose this pretrained model to make the result

more accurate.

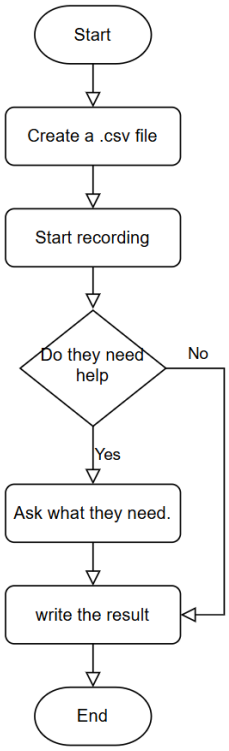

Fig. 23: System flow chart

V. Team Collaboration and Challenges

Each team member has contributed as follows:

- **[Hee Chan Kim]**: Build the controller for webots, design the gripper in Fusion360, make the proto or import the gripper to the simulation, connect ardupilot to our specific world, tune the parameters of the drone used by ardupilot to compute the controller for the drone in the simulation.
- **[Tina Lo]**: Built the world of the simulation in Webots, and dealt with voice recognition with machine learning on random forest classifier and features extraction.
- **[Ellen Lee]**: Built a YOLOv7-based detection model for real-time emergency identification, integrated TTS for immediate guidance, and combined it with voice recognition for responsive interaction in rescue operations.
- **[Pulin Zhang]**: Built a 3D model BUilding system by Instant-NGP-binary to create 3 D model for target for rescue team to diagnosi;worked on ROS and debian 11 operation system installation.COllated the drone by mission planner and Q ground Control,
- **[Erick Gomez]**: build the simulation environment and set up the environment , set up the parameters for the initial flight such PDI and calibrating sensors to have a stable drone establish the communication of the ground control station and stablish paths of exploring.

During the development process, the team encountered several challenges that required creative problem-solving. One major challenge was dealing with environmental factors such as wind and waves, which significantly impacted drone stability and detection accuracy. To address this issue, the **PID controller** in the **Pixhawk** flight controller was fine-tuned and using the calibration and autotune of the ground control **QGroundcontrol**, this application was crucial to establish a stable PID and have a good measurements of the sensors resulting in a stable flight of the system and good control of this, and additional training data was used to improve the robustness of the **YOLOv7** model under varying conditions. Communication latency between the companion computer and the flight controller was another challenge that affected real-time decision-making. This was mitigated by optimizing **MAVROS** configurations and implementing redundant communication channels to ensure reliable data transmission. In the voice recognition part, more data is needed to increase the accuracy of the model training.

VI. Innovative Aspects and Future Work

With several innovative aspects, the **Drone-Based Search and Rescue System** is different from other solutions.

Also, the system is **human-centric** Through the integration of a voice communication system based on the **LMNT Text to Speech API**, the rescue mission acquires a human development, transmits comforting messages to those in trouble and deals with their psychological requirements. This is particularly important since it calms victims, lowering the risk of panic and in turn bettering the odds of saving your victim.

Digital Twin technology is used to increase the drone's real time decision making capabilities. Using the depth camera, the system updates the digital twin with data, and therefore creates a dynamic model of the environment. The proposed model helps in the optimization of rescue path planning and obstacle avoidance to enable the drone to quickly find the target in a complex environment. Furthermore, the digital twin represents the process in a visual form, which is beneficial for the operator in terms of monitoring and decision making during rescue missions.

The **YOLOv7** model for real time object detection is another innovative aspect of the system as it allows the system to quickly and accurately identify persons in distress using the deployment. This all lends to the model's efficiency, as the drone needs to be able to respond quickly in emergency situations, where every second matters. This combination of autonomous flight control and computer vision allows the drone to auto correct its trajectory based on detected targets, and hence ends up as an effective and reliable rescue operation.

The current system has already demonstrated promising results but there are several things it needs to work on in the future. Another planned enhancement is the integration of **thermal imaging cameras** to enhance detection when it is nighttime, or in low visibility conditions. However, an enhancement of the chances of successful detection based on thermal cues will be provided by the thermal imaging, as visual cues may be limited.

The deployment of **swarm drone coordination** is another area of future work. The search and rescue coverage area is greatly expanded using multiple drones, giving us better ability to locate people in distress quickly. We

discuss how swarm intelligence algorithms will allow drones to communicate and collaborate, and how they will autonomously divide tasks and guarantee optimal area coverage.

Future development is also aimed at integration with existing emergency response services. Communication modules could be added to the system which could relay the system's real time data, e.g. real time GPS coordinates and live feeds of video onto emergency response teams to aid faster, more coordinated rescue missions. Ultimately, our **Drone Based Search and Rescue System** should become an indispensable resource to emergency responders, efficiently and timely supporting life threatening situations.

VII. Conclusion

This comprehensive as well as innovative solution is called **Drone Based Search and Rescue System for Drowning Detection**, which provides a solution to the serious problem of drowning incidents. The system combines autonomous navigation, real time object detection, digital twin modeling and human centered communication, significantly improving the effectiveness and efficiency of water based rescue operations. Results show that drones can be used for search and rescue missions and that such use is a cost effective and scalable approach for reducing response time and increasing likelihood of saving lives.

A lot of cutting edge technologies were used for this project and as a result the project was a success because of effective teamwork and collaboration. This simulation phase allowed us to polish out our system, before real life testing, and the real world validation of the core functionalities of the system. The project encountered environmental challenges and challenges in real time communication, however, with careful design and optimization coupled with teamwork these were overcome. Future work will increase system capabilities such as thermal imaging, swarm coordination, and integration with emergency response services to further enhance the reliability of the system in real world environments.

References

[1] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," arXiv, Jul. 2022. http://arxiv.org/abs/2207.02696

[2] C. Dong, Y. Tang, and L. Zhang, "Higher efficient YOLOv7: a one-stage method for non-salient object detection," Multimedia Tools and Applications, vol. 83, no. 14, pp. 42257–42283, Oct. 2023. https://link.springer.com/10.1007/s11042-023-17185-w

[3] K. Li, Y. Wang, and Z. Hu, "Improved YOLOv7 for Small Object Detection Algorithm Based on Attention and Dynamic Convolution," Applied Sciences, vol. 13, no. 16, p. 9316, Aug. 2023. https://www.mdpi.com/2076-3417/13/16/9316

[4] "LMNT," LMNT. [Online]. Available: https://www.lmnt.com/?ref=theresanaiforthat

[5] C. T. Jin, "Speech-and-AudioFeatures.ipynb," Canvas, University of Sydney. Available: https://canvas.sydney.edu.au/courses/59739/external_tools/113580?display=borderless

[6] "Random Forest Classifier Tutorial," kaggle.com. https://www.kaggle.com/code/prashant111/random-forest-classifier-tutorial

[7] GeeksForGeeks, “Random Forest Classifier Using Scikit-learn,” GeeksforGeeks, Sep. 04, 2020. https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/

[8] “MFCC implementation and tutorial,” kaggle.com. https://www.kaggle.com/code/ilyamich/mfcc-implementation-and-tutorial

[9] T. S, “Using MFCCs and Machine Learning for Audio Classification in Python,” Medium, Mar. 14, 2023. https://medium.com/@tracystrickel/using-mfccs-and-machine-learning-for-audio-classification-in-python-6c011c9c1994

[10] H. Kalam, “Different Ways to Build Audio Classifiers— Stackademic,” Medium, Oct. 12, 2024. https://blog.stackademic.com/different-ways-to-build-audio-classifiers-bb0c700931c3(accessedNov.03,2024).

[11] Y. Verma, “A Tutorial on Spectral Feature Extraction for Audio Analytics,” AIM, Sep. 19, 2021. https://analyticsindiamag.com/ai-mysteries/a-tutorial-on-spectral-feature-extraction-for-audio-analytics/

[12] QGroundControl. (2024). QGroundControl Documentation. Retrieved November 3, 2024, from https://docs.qgroundcontrol.com/

APPENDIX

Source file:https://github.com/moiseskim98/drone-search-n-rescue

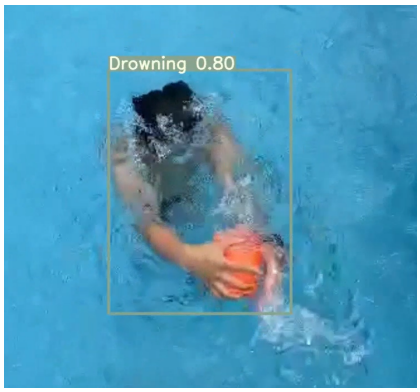

Fig. 24: Drowning detection with 0.80 confidence

Fig. 25: Sample frames from the training set.