Machine-Learning-Based Wind Detection and Avoidance Using a Crazyflie Micro Drone

Kyle VanHorn *
University of North Carolina at Charlotte, Charlotte, NC, 28223

Urban environments pose significant challenges for drone flights due to their complex wind patterns that can impact flight performance and stability. Detecting and mapping wind conditions can assist in navigating drones within such environments by providing additional information to onboard vehicle path planning and control systems. This paper investigates the feasibility of utilizing sensor data from the inertial measurement unit (IMU) of a micro-quadrotor to detect wind patterns in an indoor scaled urban environment using machine learning techniques. This work utilizes the Crazyflie 2.1 micro-quadrotor system to investigate the ability to detect wind in motion and dynamically adjust vehicle trajectories. Artificial wind was simulated by a module of four computer fans connected in series and modulated with PWM signals. A wind detection framework was developed using Python's TensorFlow library. The artificial neural network implemented was a binary classification model. It processed the triaxial gyroscope readings from the IMU as inputs and outputted a signal value between 0 and 1 representing the likelihood of wind presence. The training data for the model was collected over a series of 50 flights with the quadrotors, during which the position of the wind was recorded and cross-referenced with the quadrotor's position to determine whether the quadrotor was in the flow path of the known wind. Following training data collection and model training, a Python script was developed to autonomously fly the quadrotors. The detection model was successfully deployed during flight with the gyroscope data collected at 100 Hz and wind predictions calculated at 1 Hz. Analysis of 10 test flights found that at a wind prediction confidence threshold of 0.60, the model successfully predicted 91% of wind occurrences, while falsely predicting wind in less than 2% of non-wind occurrences. The success of this approach highlights the potential for leveraging machine-learning-based wind detection using existing onboard sensor suites and adaptation strategies to enhance the flight capabilities of drones in urban environments.

Nomenclature

UAV	=	uncrewed aerial vehicle
MAV	=	micro aerial vehicle
IMU	=	inertial measurement unit
DOF	=	degree of freedom
Bluetooth LE	=	Bluetooth Low Energy
ISM	=	Industrial, Scientific, and Medical
LPS	=	Loco Positioning System
TWR	=	two-way ranging
TDoA	=	time delay of arrival
ToF	=	time of flight
cflib	=	Crazyflie Python Library
PWM	=	pulse-width modulation
CSV	=	comma-separated values
P_x	=	x position
P_y	=	y position
P_z	=	z position

*Undergraduate Researcher, Mechanical Engineering and Engineering Science, AIAA Student Member.

P_ψ	=	yaw
ANN	=	artificial neural network
ReLU	=	Rectified Linear Unit

I. Introduction

Uncrewed aerial vehicles (UAV's) are becoming a popular aspect of daily life. UAV technologies are reaching many industries, including defense, search and rescue, healthcare, package delivery, and agriculture. UAVs are often resource constrained with limited sensor capabilities and battery lives. Certain flight environments can be more dangerous or costly to fly through. This work focuses on flying in urban environments where there can be very complex wind patterns due to building geometry. Having the ability to detect and measure wind in such environments can allow for the generation of safe and energy efficient trajectories. In this work, an indoor scaled urban environment with a micro-aerial vehicle (MAV) is examined. Due to their small size, MAVs are heavily affected by turbulence, causing them to accumulate trajectory tracking errors.

This paper explores the feasibility of using the onboard IMU sensors of a MAV to detect wind and dynamically adjust the trajectory. The presence of wind should cause a disturbance in the flight of the MAV, such that already onboard sensors in the IMU will make the presence of wind detectable to a machine learning model trained on known wind scenarios.

II. Related Works

Recently, much work involving environmental sensing with UAVs has been performed. Environmental sensing is the task of determining the presence of entities in a physical environment. These entities may include physical obstacles such as buildings, walls, and other robots. Other invisible entities are often desirable to detect such as wind, turbulence, and gases. Much of the prior work has focused on visible environmental sensing as this can be done with imaging sensors. Chadehumbe and Sjöberg developed an algorithm to autonomously fly a Crazyflie through an obstacle course using an optical flow sensor and laser-range finding for obstacle detection [1]. Engel, Sturm, and Cremers developed a complete visual navigation system for a MAV, relying on a monocular camera to perform simultaneous localization and mapping (SLAM) [2]. Other works have added specialized sensors to MAVs to perform specific measurements. Neumann et al. developed a swarm of Crazyflie drones mounted with indoor air quality sensors to generate gas distribution maps [3].

The previously mentioned works all relied upon adding additional sensor suites to the MAV. Some work has been done focusing on leveraging the data available from sensors typically available on all UAVs. An IMU consisting of an accelerometer and gyroscope is a typical sensor found on most drones. Zhao developed a framework for detecting obstacles based on air disturbances measured by the accelerometer and gyroscope while a drone flies near objects [4]. In a later work, Zhao, Hughes, and Lyons focused on detecting the wakes generated by a drone flying above using data mining techniques [5]. Gu and Lin developed machine learning models to compare the ability of the gyroscope and the accelerometer to detect wind gusts [6]. The focus of this work will be to develop a new machine learning model capable of detecting any presence of wind, not just gusts, and be able to create a windmap that can be used in trajectory planning.

III. Hardware

A. Crazyflie 2.1 Micro-Aerial Vehicle

The drone employed in this work was the Crazyflie 2.1 platform, developed and manufactured by Bitcraze. The Crazyflie is a 27g, open source flying development platform specifically designed for education, research, and swarming. The Crazyflie uses a printed circuit board (PCB) as the frame of the drone [7]. A Crazyflie can be seen in Fig. 1.

The Crazyflie is equipped with both radio and Bluetooth Low Energy (LE). It can be controlled using a mobile device over Bluetooth or with a computer and the Bitcraze Crazyradio 2.0. The Crazyradio 2.0 is a USB radio dongle compatable with the Crazyflie ecosystem. It is based on the nRF52840 from Nordic Semiconductor and communicates with the Crazyflie over 2.4 GHz Industrial, Scientific, and Medical (ISM) band radio [8].

In this work, the Crazyflies were operated using a Crazyradio connected to a Linux computer. The Crazyflies were controlled autonomously using Python and the Crazyflie Python Library [9]. Python scripts were written to send commands to the Crazyflie. All trajectory planning calculations and wind prediction calculations were performed by the computer ground station and the flight commands were uploaded to the Crazyflie.

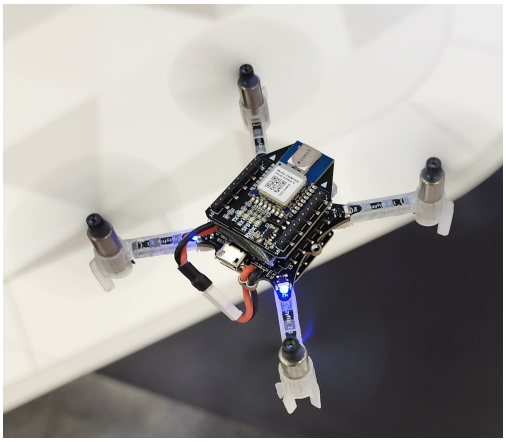

Fig. 1 Crazyflie 2.1 in Flight, Equipped with Loco Positioning Deck

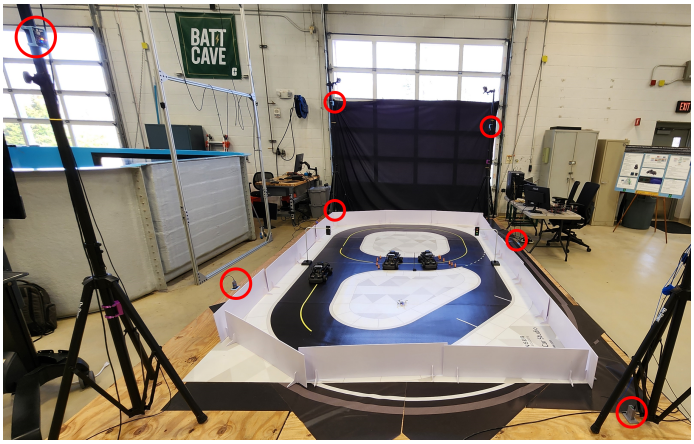

Fig. 2 Indoor Scaled-Urban Flying Environment with Loco Positioning Nodes Circled in Red

B. Sensors

The Crazyflie 2.1 comes with a 7 DOF IMU. The onboard sensors are

1) 3-axis Gyroscope (BMI088)
2) 3-axis Accelerometer (BMI088)
3) High Precision Pressure Sensor (BMP388)

C. Positioning System

The capabilities of the Crazyflie drone can be expanded through the addition of decks. Decks, manufactured by Bitcraze, are expansion boards that can be mounted above and below the main PCB frame of the Crazyflie to add capabilities. While Bitcraze offers a multitude of options, the decks selected for use in this work added positioning capabilities to the Crazyflie. Access to absolute position data was necessary for this work to plan and follow trajectories and map wind locations. The Loco Position System was selected as the primary absolute positioning system, with the Z-ranger V2 deck utilized to improve the z-position accuracy.

The Loco Positioning System (LPS) can operate in one of two ways, two-way ranging (TWR) or time delay of arrival (TDoA). TWR is a more robust form of determining location; however, it only supports a single Crazyflie. TDoA was used in this work to allow for swarming in future applications. The LPS consists of eight anchors or nodes and a loco positioning deck (tag) on each Crazyflie. The LPS relies on Ultra Wide Band radio to determine the 3D position of the Crazyflie. The anchors are positioned in the flying environment and their positions are measured to form the reference, as pictured in Fig. 2. The LPS can reach positional accuracies of ± 10 cm from the actual position of each vehicle [10].

One node was placed in each corner of the cubic flying environment. The absolute positions of the nodes could not be measured with sufficient accuracy due to the uncertainty of the mounting locations; therefore, the measured coordinates were approximate. To optimize the position estimates for each node, pairwise distances between all nodes were measured using a laser. A MATLAB script was used to optimize the originally measured positions to fit the distance matrix. This was done through constrained nonlinear optimization using MATLAB's 'fmincon' function. Fig. 3 shows how the maximum difference between the original positions and the distance matrix was reduced from 24 cm to 3 cm.

Preliminary test flights with the Crazyflie drones revealed that the LPS performed to its listed accuracy in the x-y plane; however, there were variations of much greater than ± 10 cm in the z direction. To improve the accuracy of the z-position, the Z-Ranger V2 deck was implemented. The z-ranger deck utilizes a VL53L1x ToF sensor to measure distances up to 4 meters [11]. This sensor provided much greater accuracy for the z-position estimate, assisting the Crazyflie in flying stable trajectories.

D. Artificial Wind Generation

To generate artificial wind in the indoor flying environment, a module consisting of four 120mm computer fans was built, as pictured in Fig. 4. To orient the fans in the same plane and operate them in the same direction, they were

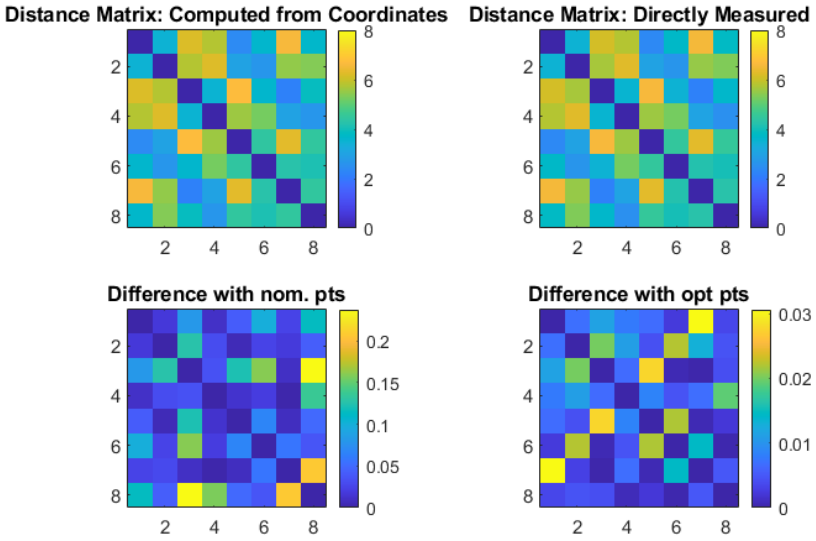

Fig. 3 Optimization of Loco Positioning System.

constrained using 3D printed interlocking mounts, allowing them to operate vertically or horizontally. The wind speed generated by the fan module was controlled via an Arduino. The computer fans were wired in series, connected to a 15V power supply, and the PWM wire connected to the Arduino. An Arduino sketch was written to listen to inputs over the USB communication cable. This allowed for a Python script to generate a time-varying wind field. The magnitude of the wind generated by the computer fan module was measured with a handheld digital vane anemometer at different fan RPMs as a function of distance from the fans. The results can be seen in Fig. 5.

Fig. 4 Computer Fan Module Used for Wind Generation Positioned in Flying Environment

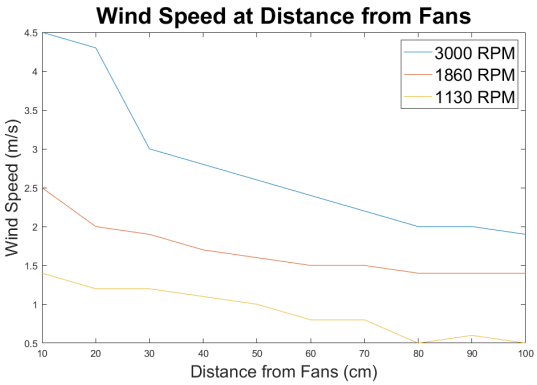

Fig. 5 Fan Generated Wind Speeds Collected with Handheld Anemometer

IV. Control Methodology

The Crazyflie Python Library contains multiple different flight commanders, including the commander, motion commander, and high level commander. The commander allows for control of the Crazyflie through a variety of command types including sending roll, pitch, yaw rate, and thrust commands; sending x, y, and z velocity commands; and sending absolute position setpoints. The motion commander allows for relative motion commands including forward, left, right, back, up, and down. The high level commander was used in this work and is explained in detail below.

A. Trajectories

The high level commander requires the calculation of four 7th order splines for each time step to define the desired trajectory of the quadrotor. Each 7th order spline defines the desired x, y, and z positions, and the desired yaw angle with respect to time, allowing for smooth trajectory generation. The 7th order polynomials for the trajectory can be represented as:

$$P_x(t) = a_{0x} + a_{1x}t + a_{2x}t^2 + a_{3x}t^3 + a_{4x}t^4 + a_{5x}t^5 + a_{6x}t^6 + a_{7x}t^7 \tag{1}$$

$$P_y(t) = a_{0y} + a_{1y}t + a_{2y}t^2 + a_{3y}t^3 + a_{4y}t^4 + a_{5y}t^5 + a_{6y}t^6 + a_{7y}t^7 \tag{2}$$

$$P_z(t) = a_{0z} + a_{1z}t + a_{2z}t^2 + a_{3z}t^3 + a_{4z}t^4 + a_{5z}t^5 + a_{6z}t^6 + a_{7z}t^7 \tag{3}$$

$$P_\psi(t) = a_{0\psi} + a_{1\psi}t + a_{2\psi}t^2 + a_{3\psi}t^3 + a_{4\psi}t^4 + a_{5\psi}t^5 + a_{6\psi}t^6 + a_{7\psi}t^7 \tag{4}$$

Each trajectory segment is defined by 32 coefficients ($a_{i,(x/y/z/\psi)} \forall \{i \in 0, 1, \ldots, 7\}$), with an extra coefficient to define the timespan over which that segment of the trajectory is executed. Calculating these 33 parameters for each segment of an entire trajectory was done using genTrajectory from the whoenig/uav_trajectories GitHub repository [12]. The genTrajectory program, written in C++ and executable in Linux, takes a series of waypoints as an input and with a given maximum velocity and acceleration, will output the 33 coefficients for a series of functions, creating a smooth and safe trajectory. Each inputted waypoint is visited in order and the arrival time for each waypoint is automatically calculated. The waypoints are given as a CSV file and the output calculated by genTrajectory is also a CSV file containing the desired coefficients.

B. High Level Commander

The high level commander is used for sending high level setpoints to the Crazyflie [13]. It includes commands such as takeoff, land, go to, and start trajectory. In this work, the high level commander was used to control the trajectory for the Crazyflie. To use the start trajectory command, a trajectory must first be calculated, defined, and uploaded to the Crazyflie. The required format is a list of lists, where each internal list defines one segment of the trajectory with the 33 floating point coefficients described above, with the first value representing the duration of that segment.

V. Machine Learning Model Methodology

The IMU sensor data is primarily used internally by the Crazyflie to maintain stable flight. When flying using a Python script, this data can be logged and used as desired. The IMU data can be used by a machine learning model to allow the Crazyflie to sense its environment. Specifically, the model can be trained to detect the presence or absence of wind. The gyroscope readings from the IMU were selected to be used in the detection framework.

A. Training Data Collection

The first step in developing this detection framework was to collect data where the presence or absence of wind was known and use this data to conduct supervised training of an artificial neural network (ANN). Training data for the ANN was collected over a series of 50 flights. During each flight, gyroscope and position data were collected and stored in CSV files. The absolute location of the wind presence was measured and recorded. The true wind presence for each flight was determined by comparing the logged position of the Crazyflie to the location of the fan. To eliminate the turbulence during the landing and takeoff portion of the flight, the data logging began 0.5 seconds after completing takeoff and stopped before the landing sequence.

Over the 50 trials, the Crazyflie flew a variety of trajectories with the artificial wind blowing from several different directions. The following configurations were performed 10 times each:

1) Fans blowing vertically, Crazyflie flies over top
2) Fans blowing horizontally from the starboard side, Crazyflie yaw in direction of travel (i.e. forward flight)
3) Fans blowing horizontally from the starboard side, Crazyflie yaw in opposite direction of travel (i.e. reverse flight)
4) Fans blowing horizontally from the port side, Crazyflie yaw in direction of travel
5) Fans blowing horizontally from the port side, Crazyflie yaw in opposite direction of travel

B. Model Architecture

Once the necessary data was collected for model training, the artificial neural network architecture was created using Python's TensorFlow library. The model consisted of an input layer, two hidden layers, and an output layer. The input layer consisted of the triaxial gyroscope data from the IMU (G_x, G_y, G_z). No pre-processing or scaling of the data was performed because this would require additional calculations during implementation. A sequential model was used with dense, or fully connected neurons. The first hidden layer contained 64 neurons and the second hidden layer contained 32. There was a single binary output representing the probability of wind presence. The hidden layers used the ReLU activation function while the output used Sigmoid. The optimizer selected was Adam with a constant learning rate of 0.001. The Adam optimizer uses "a stochastic gradient method that is based on adaptive estimation of first-order and second-order moments" [14]. The loss function used was binary cross entropy. The model was trained on 47,872 data points over 200 epochs, with a batch size of 32. An 80/20 split was used to divide the data for training and validation. Fig. 6 illustrates the model architecture.

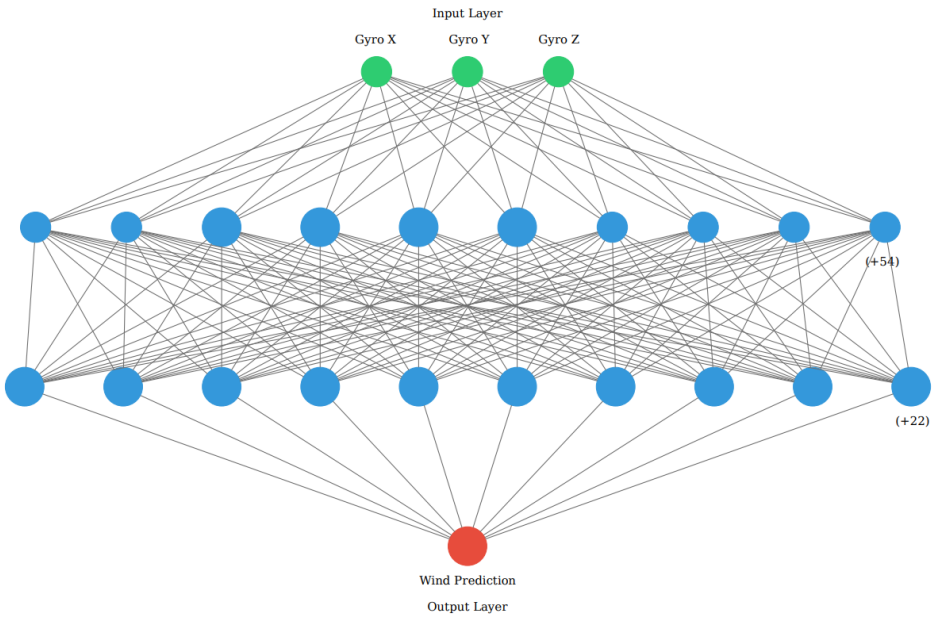

Fig. 6 Artificial Neural Network Architecture

C. Model Implementation

The model was successfully deployed in near real time during the flight of a Crazyflie. The gyroscope and position data were logged at 100 Hz. Every second, the last 100 data points from the gyroscope and position data were sent to the model as a batch. The model would predict the likelihood of wind presence at each point based on the gyroscope data. Any predictions over the selected confidence threshold (>0.60) were recorded as locations of wind presence and appended to a variable called wind map.

VI. Wind Avoidance

A wind avoidance algorithm was defined using a Python script. The algorithm controlled the trajectory of the Crazyfle, made the wind predictions, and dynamically adjusted the trajectory to avoid detected winds. The algorithm starts with the operator defining a series of initial waypoints, which are written to a CSV file and sent to genTrajectory. The trajectory CSV file containing the trajectory coefficients is then read by the Python algorithm and uploaded to the Crazyflie's memory. The Crazyflie takes off and the initial trajectory is started. During the flight, the gyroscope and positioning data is logged at a frequency of 100 Hz. This data is stored in a queue, using Python's queue library [15]. When the queue contains 100 entries, the gyroscope data is batched and sent to the ANN. The ANN then outputs a wind prediction value for each entry, providing wind presence information to the system. For values over the designated confidence threshold, the associated location is added to the wind map. The wind map is analyzed and used to update

the waypoints. This process repeats every second until the initial trajectory is completed. Once completed, a new trajectory is generated using the series of waypoints that have been updated to avoid the detected wind. Fig. 7 provides an overview of this algorithm.

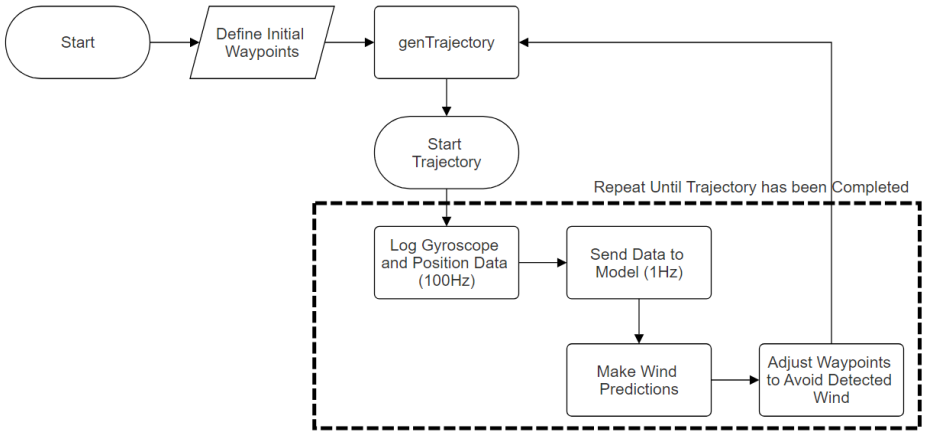

Fig. 7 Flowchart of Wind Avoidance Code

A. Updating Waypoints

Using the subprocess genTrajectory to calculate the trajectories for the high level commander is computationally expensive. Due to the computation, read/write, and upload time, the trajectory of the Crazyflie cannot be updated as soon as the wind is detected; therefore, the implementation used was to allow the Crazyflie to finish its current trajectory, hover in place while calculating and uploading an updated trajectory, and then resume flying. The updated trajectory is defined by the initial waypoints plus modifications based on the wind map data. The wind avoidance algorithm utilized in this work employs a waypoint updating mechanism to adjust the trajectory of the Crazyflie drone in response to detected wind patterns. Initially, a series of waypoints is predefined by the operator to outline the desired flight path. When wind presence is detected during flight, the algorithm dynamically updates these waypoints to navigate around the affected areas, thus minimizing wind interference. The updating process involves identifying the closest existing waypoint to the detected wind location and assessing its proximity. If the wind is found to be sufficiently far from the last adjusted waypoint or any existing waypoint, a new waypoint is added near the wind location to divert the flight path. This waypoint is added at the x and y coordinates of the detected wind location with a z coordinate shift, causing the Crazyflie to fly above the wind. Conversely, if the wind is in close proximity to an existing waypoint, the existing waypoint is z-shifted to avoid the wind. A z-shift was used because in a true urban environment, shifting in the x and y is not possible due to buildings; therefore, changing the flight altitude is the best way to avoid turbulent air and wind canyons. When deciding where to insert a new waypoint near a detected wind location, the algorithm must consider the direction of the flight path and ensure that the trajectory remains continuous and coherent. To achieve this, the algorithm evaluates whether inserting the new waypoint before or after the closest existing waypoint results in a smoother transition in the flight path while still effectively avoiding the wind. The updated waypoints are then utilized to recalculate the trajectory, ensuring smooth navigation despite wind disturbances. This iterative process of waypoint adjustment and trajectory recalculation allows the drone to autonomously adapt its flight path, effectively mitigating the impact of wind on its performance. Fig. 8 provides a flowchart of the decision making process for updating the waypoints.

VII. Results

To evaluate the ANN's ability to accurately and precisely detect the presence of wind and the wind avoidance algorithm's ability to adjust the trajectory, a series of test flights were conducted. During these flights, the drone experienced wind from a variety of different angles while flying different trajectories. A total of 10 test flights were performed. For each flight, an initial set of five waypoints were defined, with the first matching the last, creating a closed, circular-like trajectory. During each test, the Crazyflie flew three complete laps, adjusting the trajectory between each lap to avoid the detected wind. The true wind position was recorded by using the Crazyflie's positioning system to

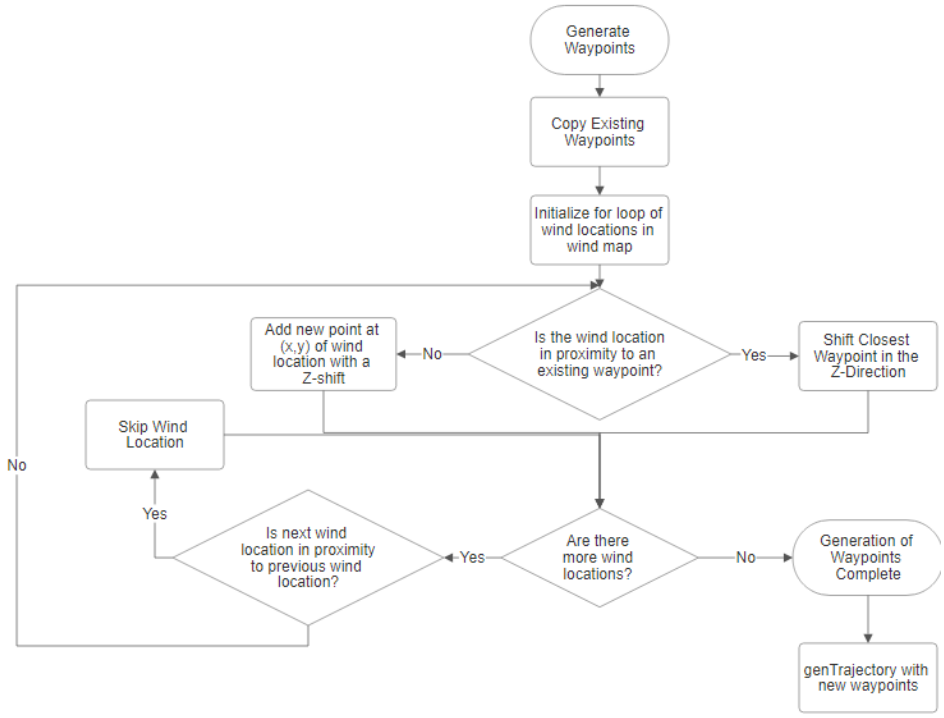

Fig. 8 Flowchart of Waypoint Generation for Wind Avoidance

take manual measurements of the fan module's location. This data was used to cross-reference the detected wind to determine the accuracy of the ANN.

Fig. 9 provides a sample of the data collected during a test flight. The top graph is the x, y, and z gyroscope data over time. The middle graph is the actual true wind presence, generated after the flight by cross-referencing the recorded position of the drone with the known position of the wind. The bottom graph represents the output generated by the ANN with the input of the triaxial gyroscope data from the top graph. It can be seen that the Predicted Wind Presence matches the true wind presence with enough accuracy for this approach to be used to detect wind. There are no spikes outside of the true wind presence zones in the predicted wind presence that would trigger an undesired change in trajectory. Every instance of true wind presence is predicted by the ANN within the necessary time frame to make the necessary changes in the trajectory.

Fig. 10 provides a sample of the flight trajectory used and how the trajectory changes on subsequent laps to avoid the previously detected wind. In this example, the Crazyflie flies the first loop and experiences the wind, represented by the red arrow. The results indicate that the ANN successfully detected this wind by the change in color of the data points along the trajectory. After completing the first lap, a new trajectory was calculated. In this case, the trajectory around the detected wind was shifted in the z-direction, causing the Crazyflie to fly above the wind stream.

To analyze the accuracy of the ANN's ability to detect the presence of wind, a true positive versus false positive analysis was conducted using the data from the 10 test flights. In this analysis, each collected data point was analyzed. A true positive was defined as a data point that met the desired confidence threshold for wind detection and occurred in the wind stream. For the wind avoidance algorithm to function as desired, it was not necessary for all data points occurring in the wind stream to return a positive wind prediction. For example, if the Crazyflie flies through the wind stream for 2 seconds and the gyroscope data is collected at 100 Hz, there would be 200 data points that should return positive wind predictions. It is unnecessary for all 200 data points to return positive wind predictions for the wind avoidance algorithm to accurately avoid the wind. Therefore, any data point that occurred in the wind stream was deemed a true positive if any of the 50 surrounding data points, representing 0.5 seconds of data, correctly predicted wind at the specified threshold value. A false positive was defined as any positive wind prediction that occurred outside of the wind stream. Fig. 11 illustrates the accuracy of the ANN as the confidence threshold value changes. With a confidence threshold value of 0.60, the ANN accurately detected 91% of wind occurrences, while falsely predicting wind in less than 2% of non-wind occurrences.

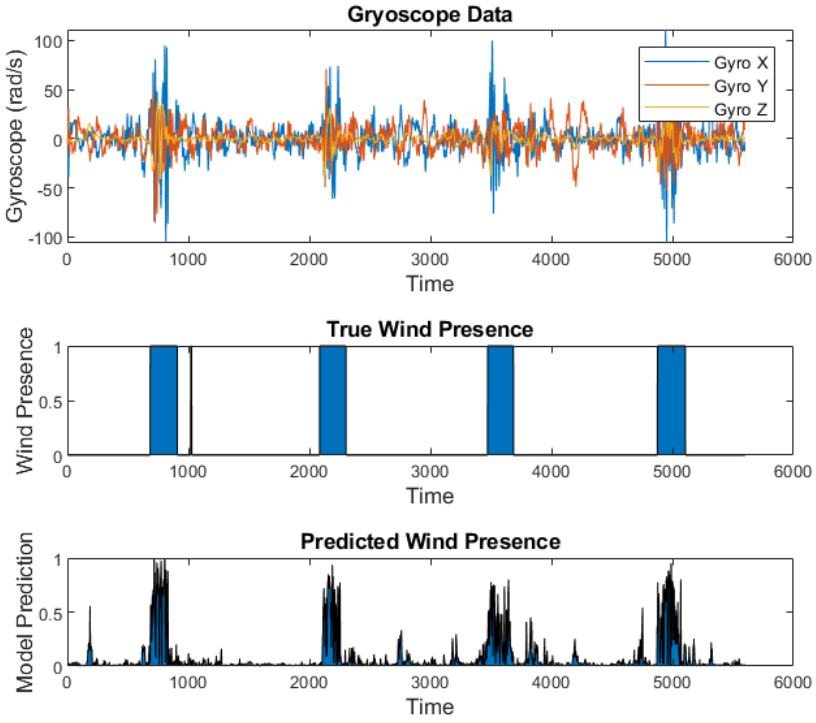

Fig. 9 Example of Collected Data

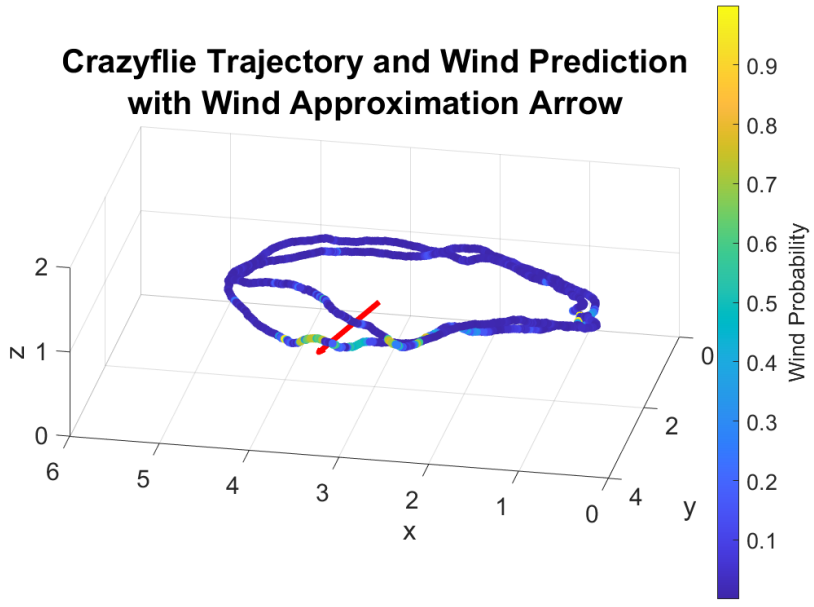

Fig. 10 Example of Wind Avoidance Trajectory

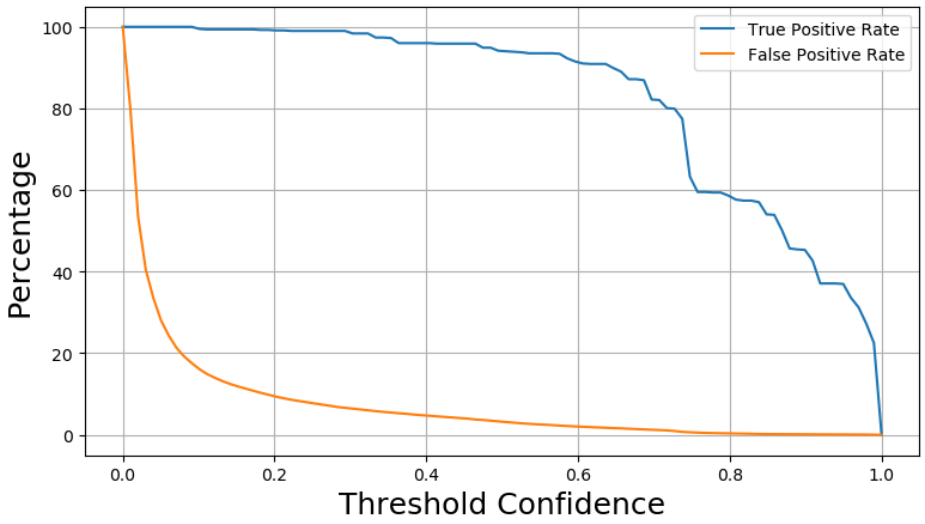

Fig. 11 True Positive and False Positive Rates versus Confidence Threshold Values

VIII. Conclusion

The goal of this work was to examine the feasibility of developing a wind detection and avoidance framework on a resource-constrained UAV. Using the Crazyflie 2.1 drone, an artificial neural network was successfully trained and deployed during flight to detect wind. The artificial neural network relied strictly on input readings from the existing gyroscope onboard in the IMU. The tests results prove that this is a viable method of detecting the presence of wind. A wind avoidance algorithm was develop utilizing wind predictions from an artificial neural network to dynamically adjust the trajectory to avoid the detected winds. These findings highlight the potential for leveraging machine-learning-based wind detection using existing onboard sensor suites and adaptation strategies to enhance the flight capabilities of drones in urban environments. Future work may include improving the trajectory planning and controls to better support swarming and developing a new artificial neural network capable of estimating the wind speed and direction, not just a binary classification of wind presence.

Funding Sources

This work was supported by the University of North Carolina at Charlotte's Office of Undergraduate Research (OUR) and by NSF grant No. 2301475.

Acknowledgments

The author thanks Dr. Artur Wolek and Nick Kakavitsas for providing guidance throughout this research and for paper review. The author also thanks Dr. Amir Ghasemi for allowing access to the smart city environment which was used throughout testing.

References

[1] Chadehumbe, C., and Sjöberg, J., "Autonomous flight of the micro drone Crazyflie 2.1 through an obstacle course," , 2020.

[2] Engel, J., Sturm, J., and Cremers, D., "Scale-aware navigation of a low-cost quadrocopter with a monocular camera," *Robotics and Autonomous Systems*, Vol. 62, No. 11, 2014, pp. 1646–1656. https://doi.org/https://doi.org/10.1016/j.robot.2014.03.012, URL https://www.sciencedirect.com/science/article/pii/S0921889014000566, special Issue on Visual Control of Mobile Robots.

[3] Neumann, P. P., Hirschberger, P., Baurzhan, Z., Tiebe, C., Hofmann, M., Hüllmann, D., and Bartholmai, M., "Indoor Air Quality Monitoring using flying Nanobots: Design and Experimental Study," *2019 IEEE International Symposium on Olfaction and Electronic Nose (ISOEN)*, 2019, pp. 1–3. https://doi.org/10.1109/ISOEN.2019.8823496.

[4] Zhao, Q., "The Development of a ROS Driver for the Crazyflie Micro-drone and Its Use to Study Drone Proximity Detection via Air Distrubance Analysis," , 2019.

[5] Zhao, Q., Hughes, J., and Lyons, D., "Drone proximity detection via air disturbance analysis," *Unmanned Systems Technology XXII*, Vol. 11425, edited by H. G. Nguyen, P. L. Muench, and C. M. Shoemaker, International Society for Optics and Photonics, SPIE, 2020, p. 114250L. https://doi.org/10.1117/12.2556385, URL https://doi.org/10.1117/12.2556385.

[6] Gu, S., and Lin, M., "Wind Gust Detection using Physical Sensors in Quadcopters," *CoRR*, Vol. abs/1906.09371, 2019. URL http://arxiv.org/abs/1906.09371.

[7] BitCrazeAB, "Crazyflie 2.1," https://www.bitcraze.io/products/crazyflie-2-1/, 2023.

[8] BitCrazeAB, "Crazyradio 2.0," https://www.bitcraze.io/products/crazyradio-2-0/, 2023.

[9] BitcrazeAB, "Crazyflie Python Library," https://github.com/bitcraze/crazyflie-lib-python, 2023.

[10] BitCrazeAB, "Loco Positioning system," https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/, 2023.

[11] BitCrazeAB, "Z-ranger deck v2," https://www.bitcraze.io/products/z-ranger-deck-v2/, 2023.

[12] Hönig, W., "uav_trajectories," https://github.com/whoenig/uav_trajectories, 2021.

[13] BitcrazeAB, "high_level_commander.py," https://github.com/bitcraze/crazyflie-lib-python/blob/master/cflib/crazyflie/high_level_commander.py, 2021.

[14] TensorFlow, "tf.keras.optimizers.Adam," https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam, 2021.

[15] "queue - A synchronized queue class," https://docs.python.org/3/library/queue.html, 2022.