



# A Novel Implementation of an Autonomous Human Following Drone using Local Context

Jayson Piquero<sup>1,\*</sup>, Edwin Sybingco<sup>1</sup>, Alvin Chua<sup>2</sup>, Marc Say<sup>1</sup>, Clarisse Crespo<sup>1</sup>, Reginald Rivera<sup>1</sup>, Ma. Antonette Roque<sup>1</sup>, Leonard Ambata<sup>1</sup>

<sup>1</sup>De La Salle University. Department of Electronics and Communications Engineering

<sup>2</sup>De La Salle University. Department of Mechanical Engineering

\*Corresponding author: [jayson\\_piquero@dlsu.edu.ph](mailto:jayson_piquero@dlsu.edu.ph)

Received: 1<sup>st</sup> April 2019

Accepted: 6<sup>th</sup> June 2019

OPEN ACCESS 

**Abstract:** This paper presents a novel implementation of an autonomous human following drone using local context algorithm. The vision system consists of a drone equipped with a companion computer and camera. Through the vision system, drones can behave according to its desired application. In this study, the Pixhawk 2 Cube flight controller of the drone is connected to Odroid XU4, the companion computer. The OpenMV camera is connected to the Odroid XU4 and streams image bytes whereas the Odroid XU4 performs image processing to detect the human through local context. The Odroid XU4 controls the drone by sending commands to the flight controller based on detected objects. Drone following is implemented by detecting humans from the image stream provided by OpenMV and maintaining the detected human on the center of the image and within a specified distance through drone movements. The results show the single movements right, left, forward, and backward yielded low steady-state errors with settling time of about 3 to 4 seconds.

**Keywords:** quadrotor, vision, human tracking, local context

## Introduction

Drone technologies have been popularized with their many applications such as media, surveying, agriculture, weather forecasting, and many more. Because of this, research on the extension and improvement of applications of drones have been conducted as mentioned in [1], [2], and [3]. There have been many researches on how to incorporate new methods of navigation for drones other than GPS because of the limitations of GPS on indoor locations.

The study done in [1] is an object following drone using an algorithm called tracking-learning-detection (TLD) which was used to detect the desired object from the image stream. However due to the limited computing capabilities of the drone, an external personal computer was used for object detection. The response of the system presented in [1] is slow because the tracking algorithm

waits about a second to stabilize after a command. This is a disadvantage for tracking objects that move fast. The presented idea was in the software called FollowMe.

The object detection implemented in [2] also uses TLD for object detection. Using OpenTLD allows detection of a multiple different target, however it also uses an external personal computer which is connected to the drone through WiFi. Similar to [1], the video stream from the drone to the computer and the commands from the computer to the drone are sent through the WiFi. This limits the speed and the range of the system. Because the detection relies on the wireless communication, the response of the drone is heavily affected by distance and interferences. The limitation in distance will also require the external computer to move closer to the drone in order to continuously detect the desired object and send commands. This is a disadvantage for users especially for long duration flights. The maximum duration of the flight test presented was only up to 45 seconds.

The object following drone introduced in [3] is a robust drone vision system that makes use of deep learning as its detection framework. Hand gesture detection is implemented in [3]. Drone movements are controlled by the appearance of the hand. Since hand gestures are relatively small in a frame, the resolution of the camera was increased to 640x480 pixels which decreases computation speed. This study displayed a robust drone response, however this was only implemented on an HITL simulation. It does not include actual drone testing.

In [4], the study uses Pixhawk 2 Cube Flight Controller and NVIDIA TK1, a companion computer. The connection of the flight controller and the companion computer through Dronekit-Python and the extended capability of image processing were also introduced; however, this was not implemented and tested in an actual drone.

In addition, [5] also implemented a connection between the Pixhawk Flight Controller, Odroid XU4, and a GoPro camera. Similar to this study and [4], Dronekit-Python was introduced as an extension to the capabilities of drone systems such as commanding waypoints, distance computation between locations, and image processing; however, this also was not implemented and tested in an actual drone.

The disadvantages that are present in [1] and [2] are the distance constraints and response speed due to the limitations of WiFi. A solution that can address these problems is the use of a companion computer such as Raspberry Pi, Odroid XU4, NVIDIA TK1, or others. Because of the direct connection of the flight controller and the companion computer, the response speed of the tracking is faster and more ensured. This also removes the requirement of the external computer to be near the drone to maintain tracking.

The work in [3] is robust and accurate due to the trained deep learning framework, however the system was not implemented in an actual drone. Similarly, [4] and [5] introduced extended capabilities of drones using companion computers. These studies, however, did not present any data or testing to verify the functionality. An implementation to actual drones and flight tests can verify the functionality of these extended capabilities.

In [6], a drone was used to take aerial images from rivers. These images were used to train a classifier cascaded with the Viola Jones Algorithm, however compared to this paper which detects humans, the study in [6] detects and identifies hydromorphological features in the vicinity of a river such roofs, roads, shore, and trees.

The work in [7] made use of the local context detector to find humans in order to analyze their facial expressions. The local context detector was used as the

robust detector to find the upper body of a human in each frame of the video. This highlights the effectiveness and robustness of local context in finding humans. The study differs as local context is used in this paper to navigate drones.

This paper introduces a novel implementation of an autonomous human-following drone using local context algorithm. The system uses an Unmanned Aerial Vehicles (UAVs) with Pixhawk 2 Cube and Odroid XU4. The companion computer is on board the drone and is connected to the flight controller through serial telemetry and a python library called Dronekit. Dronekit-Python is a set of python libraries that are developed for companion computers and the flight controller. This establishes a fast communication between the companion computer and the flight controller. Through this fast communication and robust human detection, a human-following drone is achieved.

## Local Context and PD Controller

### *Human Detection via Local Context*

The fast and robust human detection via local context is a detection framework that detects features surrounding the face instead of features within the face. Because of this, human detection via local context can easily find a human within the boundary. This algorithm that was introduced in [8] is a variation to the object detection introduced by Viola et al. [9], however this algorithm uses additional rotated features described by Lienhart et al. in [10] to accurately detect the presence of a person's head, neck, and upper body. As seen on Figure 1, the bounding box that detects the human covers the entire upper body: shoulders, neck, and head.

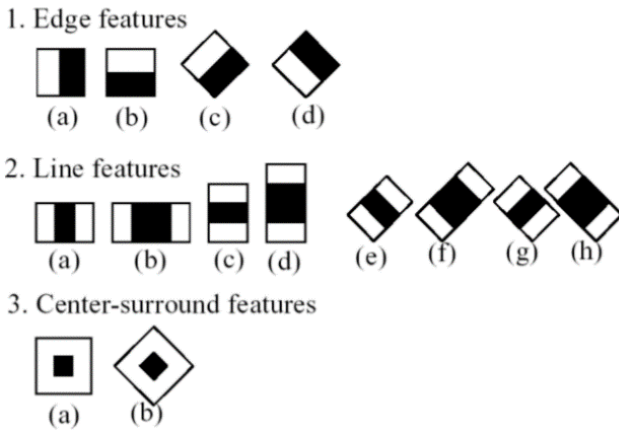


**Figure 1.** Human Detection using Local Context

Because human detection using local context made use of different features of human such as shoulders and heads, it is more reliable to detect humans despite longer distance and rotation of poses. This changes the detection from facial detection to upper body detection. Human

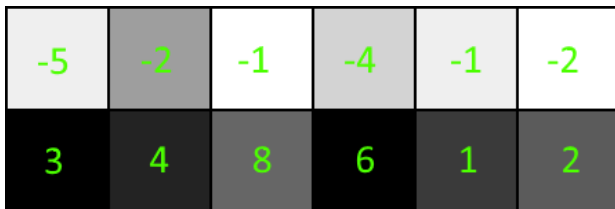
detection via local context is desirable because it is small enough to detect near the camera and large enough to be detected approximately 4 meters from the camera as seen on Figure 1.

Listed in Figure 2 are the features used for enhanced learning and detection of the local context system.



**Figure 2.** Extended Set of Haar-like Features according to [10]

These sets of features are employed in the local context detector framework including the rotated features where the white and black pixels are assigned negative and positive values respectively. Figure 3 shows a sample feature where dark and light rectangles have pixel values. This feature is similar to the edge feature 1b as shown on Figure 2.



**Figure 3.** Sample Edge Feature with Pixel Values

According to Linehart et al., feature calculation is done by summing all the pixel values in the rectangles. It can be seen on Equation 1 that feature calculation is accomplished by adding all the pixels inside each rectangle of the feature. Each value obtained in this calculation determines whether a certain number of pixels is a feature.

$$feature_i = \sum_{i \in I = \{1, \dots, N\}} \omega_i * RecSum(r_i) \quad (1)$$

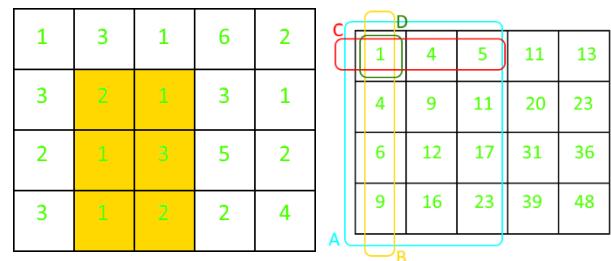
Where:  $\omega_i$  – weights  
 $r_i$  – rectangles

Since this calculation process takes long for larger images, a method called Integral Image is used where an image of the same size as the original image is created but each pixel on the Integral Image is the sum of all the pixels from the upper and left sides of that pixel in the original image. Figure 4 shows a sample original image with pixel values (upper) and the integral image (lower).

1	3	1	6	2	1	4	5	11	13
3	2	1	3	1	4	9	11	20	23
2	1	3	5	2	6	12	17	31	36
3	1	2	2	4	9	16	23	39	48

**Figure 4.** Sample Original Image (Upper) and Integral Image (Lower)

The upper image shown on Figure 4 shows a sample original image, and the lower image shown is the Integral Image. Each value of every pixel on the Integral Image is the sum of all the pixel values above and to the left of it. For example, the pixel on the 2<sup>nd</sup> row and 3<sup>rd</sup> column of the Integral Image is equal to the sum of all the pixels above and to the left of the pixel on the original image. This means that this value is the sum of 1,3,1,3,2, and 1 which is 11. That is how the Integral Image is calculated.



**Figure 5.** Original Image with sample boundary (Upper) and Integral Image with boundaries (Lower)

The upper image shown on Figure 5 is a sample original image. For instance, the highlighted portion is to be summed up to obtain its total pixel value for feature calculation. It would take long especially with larger images and regions of interest. The lower image shown on Figure 5 shows the integral image. Using the lower right pixel value of boundaries A, B, C, and D on the Integral Image, the total pixel value of the highlighted rectangle on the original image can be obtained. In order to calculate the total pixel value of the highlighted region on the original image using the Integral Image, boundary A must be subtracted with boundaries B and C and added with boundary D since the first pixel (with value of 1 from

Figure 5) was deducted twice.

$$\begin{aligned} \text{Pixel Value of Window} &= A - B - C + D \\ &= 23 - 9 - 5 + 1 = 10 \end{aligned}$$

Because of this, the computation for the pixel values of each rectangle in all images can be computed faster. By manually adding all the pixels in the on the highlighted region on the upper image of Figure 5, the same results may be obtained as shown:

$$\text{Pixel Value of Window} = 2 + 1 + 1 + 3 + 1 + 2 = 10$$

As seen on the previous computations, all large additions that are required by large regions may be reduced to adding and subtracting only 4 boundaries. This is very convenient especially for a larger number of features and larger images. The pixel value obtained in each region is used to determine whether that region is a feature. For instance, one feature contains the forehead and the hair. A typical person would have lighter forehead and darker hair.

With a very large number of features used in this detector, a method called Adaboost is implemented where relevant and irrelevant features are weighted and checked. The relevant features are the features that yield a minimum error rate and accurately classify humans and non-humans. Features are also considered to be weak classifiers. Adaboost groups classifiers, a set of features, to create strong classifiers. Each weak classifier checks if a feature is existing in a window and outputs a true or false binary value. The groups of classifiers are weighted and contribute to become a strong classifier.

Cascading is also used where a number of classifiers are cascaded together for the algorithm. Each stage of the classifier contains a strong classifier which determines whether a window could possibly contain a human. Once the window passes a stage, it moves on to the next. If a window does not pass a stage, it is discarded and never processed again. This eliminates all non-human features that are detected.

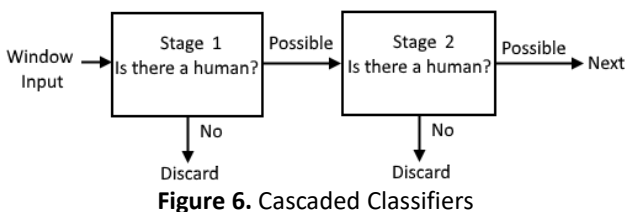


Figure 6. Cascaded Classifiers

Figure 6 shows a cascading of classifiers. This filters windows to detect humans. If a certain window with a certain size passes all the stages of the classifiers, this window could possibly contain a human. A window that

does not pass even on stage is discarded. The detection rate and false positive rate of the cascade, according to [8], is given on Equation 2. The system detection rate and false positive rate are obtained by multiplying the detection rate and the false positive rate of each stage in the cascade.

$$F = \prod_{i=1}^K f_i \quad D = \prod_{i=1}^K d_i \quad (2)$$

### PD Controller

A PD Controller was chosen for the response controller of the drone based on the value of the commands obtained from image processing. This type of controller was used due to its ease of its implementation and tuning. Compared to [11] which uses Sliding Mode Controller to move the drone, this paper implements a simpler controller. Since Sliding Mode Controller is harder to tune and involves more computation, a PD Controller is preferred to obtain a faster computation speed.

Similarly, a Fuzzy GS-PID Controller was implemented in [12] for the response of a payload dropping drone. This controller lessens the drone overshoots during a payload drop. This type of controller is not necessarily needed in this paper as the human following drone does not change in mass during flight. Because of the simpler and faster performance of PD Controllers, this was used over Sliding Mode Controller and Fuzzy GS-PID Controller.

The input to the PD Controller in this system is obtained from the image processing. Once the desired human is detected, the x coordinate, y coordinate, width, and height of the bounding box are used to control the movement of the drone by maintaining the detected human on the center of the frame as seen on Figure 7. The coordinate system used in each frame is similar to the Cartesian Plane where the center is (0,0). The frame is 240 x 240 pixels in dimension.

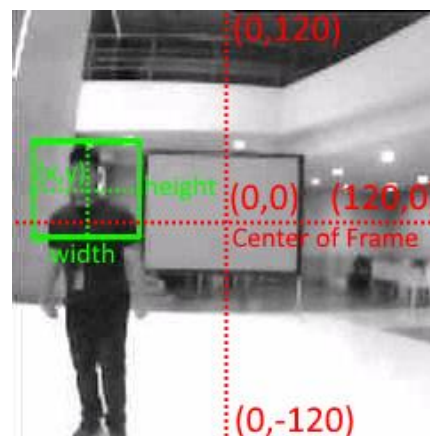
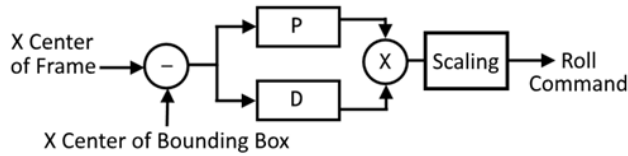


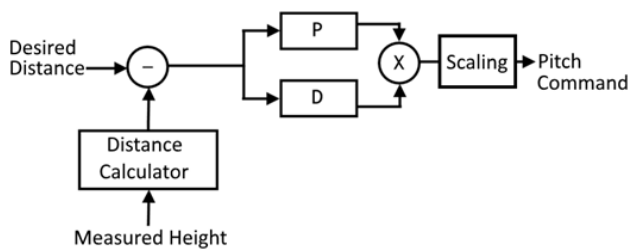
Figure 7. Coordinate System of Bounding Box

In order to maintain the x coordinate of the bounding box on the horizontal center of the frame, the drone must be controlled using a roll (tilt) angular movement. A PD Controller similar to [1] and [13] is used to center the bounding box.

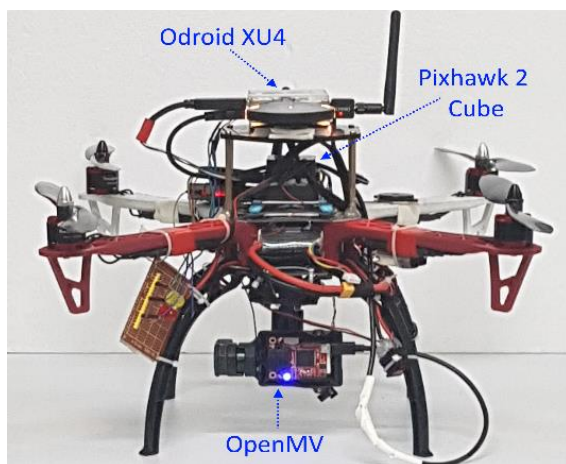


**Figure 8.** PD Controller for Roll

As seen on Figure 8, the error inputted to the PD Controller is the difference between the x coordinate of the center of the frame which is at (0,0) and the x coordinate of the center of the bounding box. This denotes that the desired x coordinate of the bounding box is at zero. By using a PD Controller to command a roll movement, the drone is able to achieve left and right movement to center the bounding box.



**Figure 9.** PD Control for Pitch



**Figure 10.** Parts of the Vision Quadrotor

Similarly, the drone is also required to maintain a certain distance from the detected human. To achieve this, the height of the bounding box is translated to distance as described in [14]. As seen on Figure 9, the measured

height is used to approximate the actual distance of the detected human from the camera. A PD Controller is also used for pitch commands to control the forward and backward movements of the drone.

## Methodology

### Flowchart of the System

The drone system is composed of the camera, the companion computer, and the flight controller. The OpenMV camera is a popular machine-vision camera capable of doing image processing just by itself. This low-powered camera is able to run real-time MicroPython, however because of its unreliability and short-ranged on its image processing, this feature of the camera was not used. The OpenMV camera was only used to capture a live video stream and send image bytes of the snapshots to the companion computer. The Odroid XU4 is a small and light weight companion computer which offers more processing speed and energy-efficiency than other companion computers like Raspberry Pi. Because of its high processing speed, the Odroid XU4 is a good choice for the companion computer as this device performs image processing and transmits commands to the Pixhawk 2 Cube Flight Controller.

Figure 11 shows a more detailed flowchart of the vision system of the drone. The image bytes are sent from the OpenMV camera to the Odroid XU4 companion computer. Image processing is obtained by using python Open Computer Vision (OpenCV) libraries and through local context for face detection as described in [8]. As previously stated, face detection through local context is fast and robust due to its ability to detect humans even with changes in facial poses. Upper Body detection through local context is also capable of detecting humans farther in distance compared to the rapid object detection introduced by Viola et al., thus human following through drones is exceptionally safer with local context because of its capability to detect and operate in farther distance.

### Distance Approximation

According to [14], the distance of a detected object can be approximated using the height of the bounding box. This is done by determining the focal length of the camera. The focal length is determined by conducting multiple experiments. As seen on Figure 12, the relationship of the focal length and the actual distance can be computed as described in Equation 3.



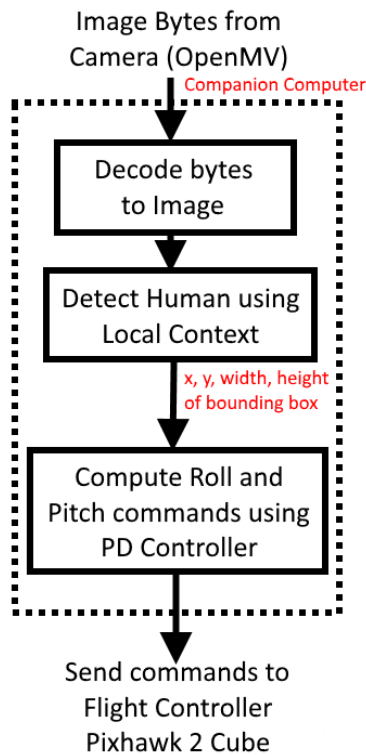


Figure 11. Block Diagram of Vision System

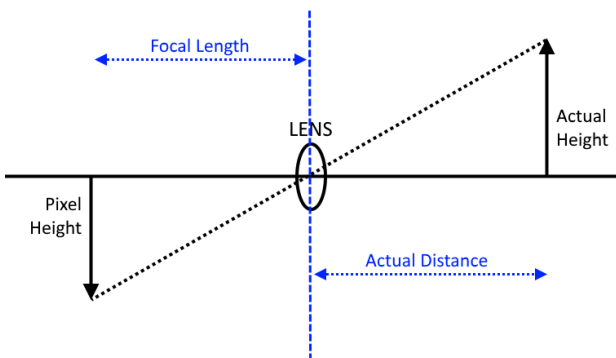


Figure 12. Lens Diagram of Camera

$$Actual\_Distance = Actual\_Height * \frac{Focal\_Length}{Pixel\_Height} \quad (3)$$

## Experimental Setup

### Hardware

The Pixhawk 2 Cube flight controller is a low-cost autopilot capable of controlling different types of drones such as multicopter, fixed-wing plane, and rovers. This autopilot is able to run Ardupilot, an open-source firmware for autopilots. This autopilot is equipped with triple gyroscope, accelerometer, and compass sensors for redundancy. Despite being equipped with multiple IMUs, this device is not capable of performing image processing

[4]. This is where the companion computer plays its part. A connection was established, through one of the two telemetry ports of Pixhawk, between the companion computer and the flight controller in order to properly receive commands. This device is controlled by the pilot through the Radio Controller (RC). Flight Modes and different movements can be commanded by the pilot using the RC.

The Odroid XU4 is a fast and efficient companion computer capable of running Linux operating systems such as Ubuntu and Android. This computer consists of octa core CPUs which enables it to perform image processing. This device also has a small voltage requirement which allows it to be supplied by the Lithium Polymer Battery that also supplies power to the flight controller.

The OpenMV camera is a small and low-cost camera, capable of running programs in Python. This also has built-in libraries for image processing. This is a very powerful camera, yet still low-powered. Since this camera is able to run image processing, it can replace the job of the Odroid XU4, however this camera currently has face detection. Human detection is less effective in this application because of its limitation in distance and pose changes of humans [8]. Because of this, the Odroid XU4 performs the image processing instead.

The python script that computes Roll and Pitch commands as shown on Figure 8 and Figure 9 are not sent to the Pixhawk 2 Cube flight controller unless the flight mode is on GUIDED, GUIDED\_NOGPS, or AUTO.

During the flight testings, the pilot uses other flight modes of the Pixhawk 2 Cube such as STABILIZE, ALTHOLD, or LOITER, however because the companion computer cannot command the flight controller while in the mentioned flight modes, the movement of the drone is still under the control of the pilot using the Radio Controller (RC).

While in GUIDED, GUIDED\_NOGPS, or AUTO flight modes, the Roll and Pitch commands computed from the PD Controller are sent from the companion computer to the flight controller using MAVLink messages that are encoded using the Dronekit-Python libraries.

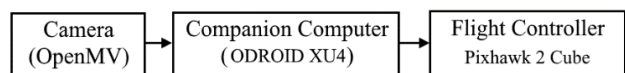


Figure 13. Basic Block Diagram of System

Figure 13 shows the flow of the algorithm. Upon obtaining the image stream from the OpenMV camera, it is sent to the Odroid XU4 companion computer. The companion computer detects humans in the image stream using local context. Drone movements are then

sent from the companion computer to the Pixhawk 2 Cube flight controller in order to center the detected human to the frame.

### Software

The software setup describes the initial configurations of the system. This includes setting up the connection between the odroid and a local computer.

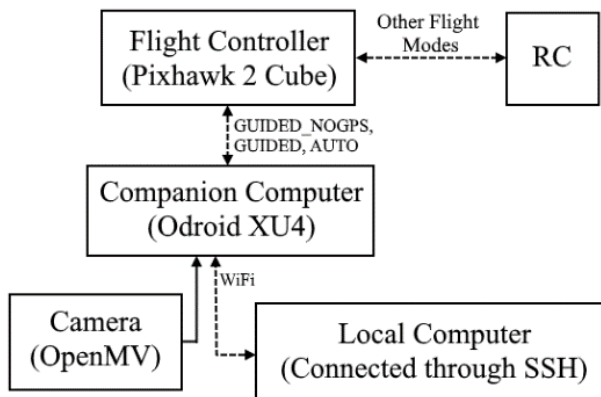


Figure 14. Detailed Block Diagram of System

Figure 14 shows a more detailed flow of the block diagram on Figure 13. Since the system python script is not run immediately after booting the companion computer, a local computer must command the companion computer to run the python script. The python script is saved on the Odroid XU4 and is run by the user through a local computer. The local computer can access the odroid XU4 through Secure Shell (SSH) protocol. The local computer and the companion computer must be connected to the same network. In this work, they are connected to the hotspot of a mobile phone. By running the script, the GUI will appear as shown on Figure 15. After choosing a desired human on the GUI, the GUI will close and the drone following will begin. This means that the system is no longer dependent on wireless connection once the desired human is chosen.

The python script that is run through the companion computer continuously sends movement commands to the drone, however the drone does not follow these commands unless it is on GUIDED, GUIDED\_NOGPS, or AUTO flight modes where the pilot has no control over the drone using the Radio Controller (RC). To enable human-following, the pilot must change the autopilot flight modes to the mentioned flight modes.

The system is written in python script. This python program is saved in the Odroid XU4 and is run by the local user from another computer through Secure Shell (SSH) protocol. The program creates a GUI that displays the video stream from the camera as seen on Figure 15.

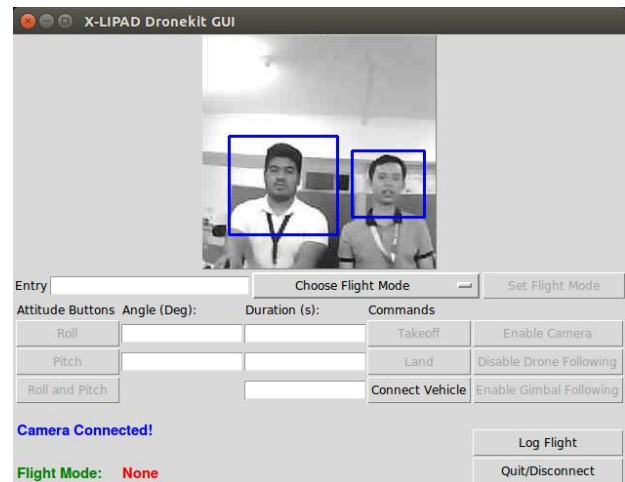


Figure 15. Detected Human in GUI

As seen on Figure 15, all detected humans were detected with bounding boxes. In order to choose a desired detected human, the user must click the bounding box of the desired human. Once clicked, the desired human is followed as seen on Figure 16.

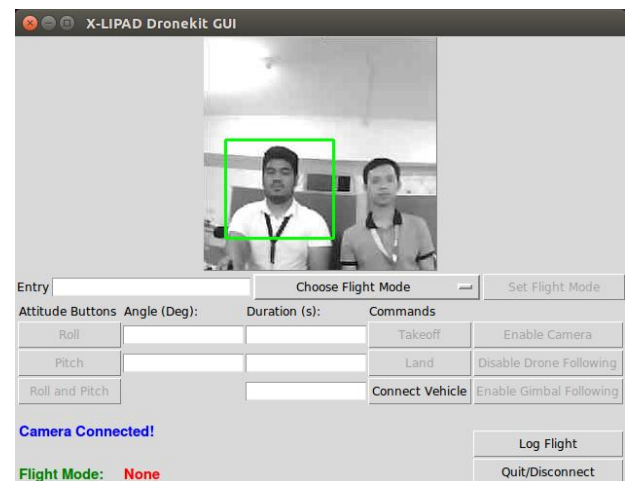


Figure 16. Chosen Human in GUI

## Data and Results

### Focal Length Calculation

Table 1 shows experimental values used to approximate the focal length. Since the focal length is constant, it can be used to approximate the actual distance of the drone from the detected human. In the trial values seen in Table 1, the focal length is averaged. Since the actual height of the upper body as seen in Figure 7 is 600mm, the Actual Height column of Table 1 are all set to 0.6m. The Actual Distance column is set by having a sample detected human standing a distance from the camera. These are the values chosen as these are the actual distances that the drone will be tested in. The Pixel Height column is the value of the height of the bounding

box obtained from the image processing. With this, the focal length can be computed.

**Table 1.** Focal Length Experimental Values

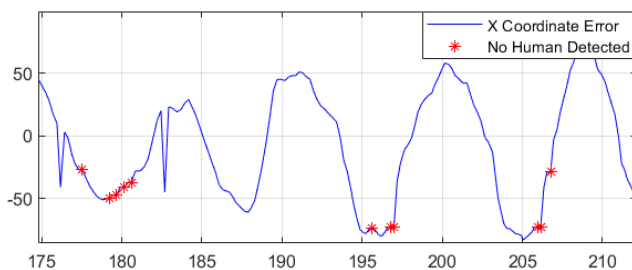
Actual Distance (m)	Pixel Height (pixels)	Actual Height (m)	Focal Length (pixels)
1.0	138.6	0.6	231
1.2	111.5	0.6	223
1.5	89.6	0.6	224
1.8	78.3	0.6	235
2.0	65.1	0.6	217
2.3	59.0	0.6	226
2.5	54.96	0.6	229
2.8	48	0.6	224
3.0	45.2	0.6	226

Basing from the values obtained on Table 1, the average focal length is approximately 226. This will be the focal length constantly used to determine the actual distance of the drone during the flight tests.

*PD Tuning*

The aim of this study is to provide another method of navigation for indoor flight tests of drones. To confirm the functionality, the basic movements of drones (Left Roll, Right Roll, Forward Pitch, and Backward Pitch) are tested along with the image processing. This study does not include or use yaw control or movements. In tuning the PD Controller, the Ziegler-Nichols method introduced by Nichols et al. in [15] were used.

To tune a PID Controller, the integral and derivative contributions to the controller must first be set to zero. The Proportional controller, Ku, must then be set to a value increasing in value until a certain oscillation, termed as Tu, is obtained from the system. Ku and Tu will then be used to determine the correct values of the PID Controller [15].

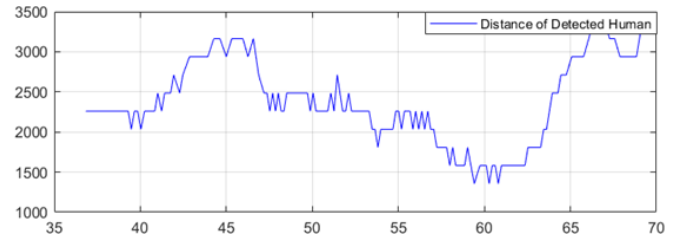


**Figure 17.** X Coordinate Oscillation

Figure 17 shows the oscillations of the x coordinate of the center of the bounding box for tuning the roll movement. This is obtained when using a Ku value of 1.5.

The average period of oscillation, Tu, is computed to be 8.4 seconds. According to [15], the Kp and Td value for the PD Controller can be obtained using these testing data. The values for Kp and Td are obtained in the computation shown below:

$$Kp = 0.8 * Ku = 1.2 \quad Td = Tu / 8 = 1.05$$



**Figure 18.** Distance Oscillation

Figure 18 shows the oscillations of the approximated distance of the bounding box for tuning the pitch movement. This is obtained when using a Ku value of 0.5. The average period of oscillation, Tu, is computed to be 17.19 seconds. According to [15], the Kp and Td value for the PD Controller can be obtained using these testing data. The values for Kp and Td are obtained in the computation shown below:

$$Kp = 0.8 * Ku = 0.4 \quad Td = Tu / 8 = 2.15$$

*Single Movement Flight Tests*

Each movement of the drone was tested: left, right, forward, and backward. The roll movement, which centers the bounding box to the center of the frame, is tested to determine whether the response lessens the error.



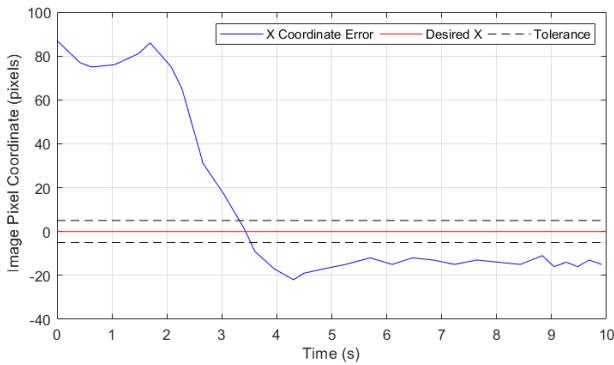
**Figure 19.** Right Movement Testing Setup

Figure 19 shows a testing setup wherein the detected human is in the front-right of the drone. To follow the detected human, it must move right. This is to test the right roll movement of the drone. The drone is



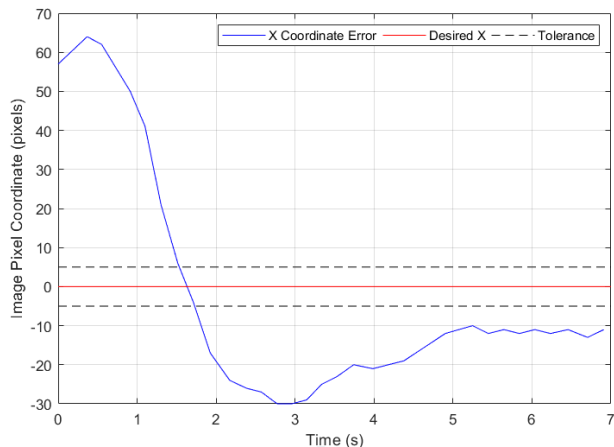
positioned 1 meter to the right of the detected human and 2.25 meter in front of the drone, however in this setup, the drone is only tested for its roll movement (left and right). The pitch movement is not tested in this setup (forward and backward).

Figure 20 and Figure 21 shows the responses of the drone to the right movement as seen on Figure 19. The desired X Coordinate Error is zero. The settling time and the steady-state error are obtained.



**Figure 20.** Right Movement Response 1: Time vs. Image Pixel Coordinate

The response of the drone moving to the right to center the detected human is shown on Figure 20. The response is approaching zero, its desired value as time progresses. The settling time obtained from this response is 4.3 seconds. The minimum and maximum steady-state errors are -11 pixels and -16 pixels respectively.



**Figure 21.** Right Movement Response 2: Time vs. Image Pixel Coordinate

Another response of the drone moving to the right to center the detected human is shown on Figure 21. The response is approaching zero, its desired value as time progresses. The settling time obtained from this response is 4.9 seconds. The minimum and maximum steady-state errors are -10 pixels and -13 pixels respectively.

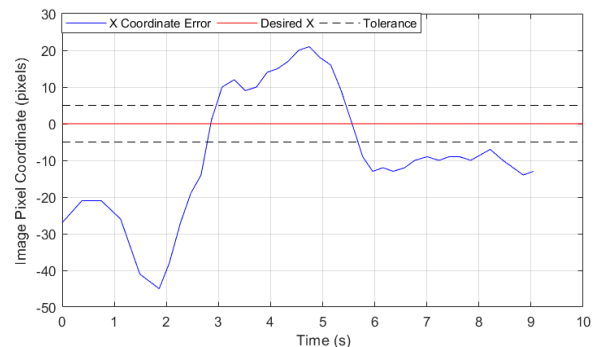
Figure 22 shows a testing setup wherein the detected human is in the front-left of the drone. To follow

the detected human, it must move left. This is to test the left roll movement of the drone. The drone is positioned 1 meter to the left of the detected human and 2.25 meter in front of the drone, however in this setup, the drone is only tested for its roll movement (left and right). The pitch movement is not tested in this setup (forward and backward).



**Figure 22.** Left Movement Testing Setup

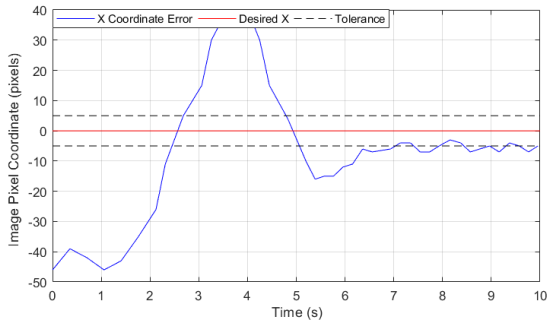
Figure 23 and Figure 24 shows the responses of the drone to the right movement as seen on Figure 22. The desired X Coordinate Error is zero. The settling time and the steady-state error are obtained.



**Figure 23.** Left Movement Response 1: Time vs. Image Pixel Coordinate

The response of the drone moving to the left to center the detected human is shown on Figure 23. The response is approaching zero, its desired value as time progresses. The settling time obtained from this response is 6 seconds. The minimum and maximum steady-state errors are -7 pixels and -12 respectively.

Another response of the drone moving to the left to center the detected human is shown on Figure 24. The settling time obtained from this response is approximately 6 seconds. The minimum and maximum steady-state error are -3 and -7 pixels respectively.



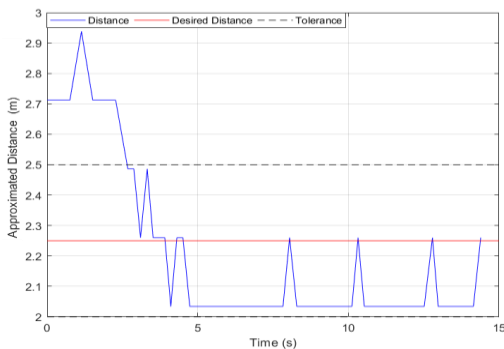
**Figure 24.** Left Movement Response 2: Time vs. Image Pixel Coordinate

Figure 25 shows a testing setup wherein the detected human is 3 meters in front of the drone. To follow the detected human, it must move forward and maintain a distance of 2.25 meters with a tolerance of  $\pm 0.25$  meter. This is to test the pitch movement of the drone. This testing is used to verify the functionality of the pitch movement (forward and backward). Roll movement (left and right) is not tested in this setup.



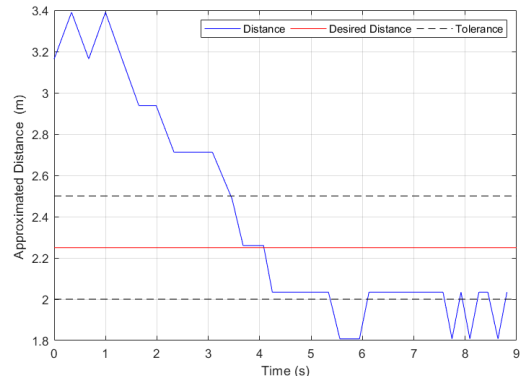
**Figure 25.** Forward Movement Testing Setup

Figure 26 to Figure 27 shows the response of the drone to the forward movement as seen on Figure 27. The desired distance is 2.25 m. The settling time and the steady-state error are obtained.



**Figure 26.** Forward Movement Response 1: Time vs. Approximated Distance

The response of the drone moving forward to the desired distance from the detected human is seen on Figure 26. The response is approaching 2.25 meters, its desired value, with a tolerance of  $\pm 0.25$  meter as time progresses. The settling time obtained from this response is approximately 5 seconds. The minimum and maximum steady-state errors are 0.01m and -0.216m respectively.



**Figure 27.** Forward Movement Response 2: Time vs. Approximated Distance

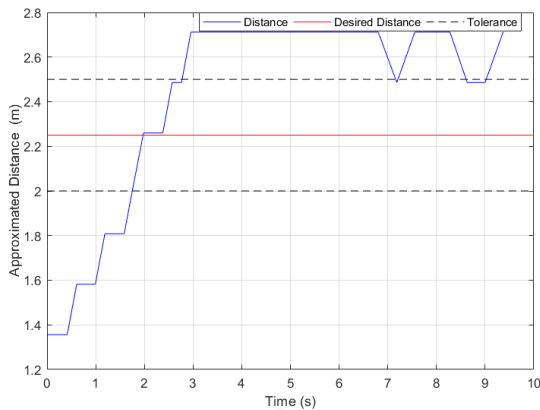
Another response of the drone moving forward to the desired distance from the detected human is seen on Figure 27. The settling time obtained from this response is approximately 4.2 seconds. The minimum and maximum steady-state errors are -0.216m and -0.442m respectively.

Figure 28 shows a testing setup wherein the detected human is 1.5 meters in front of the drone. To follow the detected human, it must move backward and maintain a distance of 2.25 meters with a tolerance of  $\pm 0.25$  meter. This is to test the pitch movement of the drone. This testing is used to verify the functionality of the pitch movement (forward and backward). Roll movement is not tested in this setup.



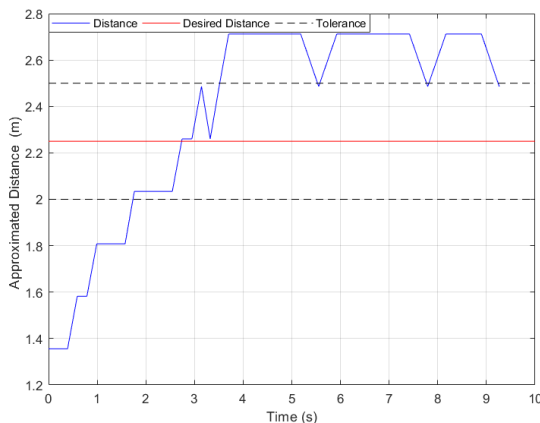
**Figure 28.** Backward Movement Testing Setup

Figure 29 to Figure 30 shows the response of the drone to the backward movement as seen on Figure 28. The desired distance is 2.25 m. The settling time and the steady-state error are obtained.



**Figure 29.** Backward Movement Response 1: Time vs. Approximated Distance

The response of the drone moving backward to the desired distance from the detected human is seen on Figure 29. The response is approaching 2.25 meters, its desired value, with a tolerance of  $\pm 0.25$  meter as time progresses. The settling time obtained from this response is approximately 3 seconds. The minimum and maximum steady-state errors are 0.236m and 0.462m respectively.



**Figure 30.** Backward Movement Response 2: Time vs. Approximated Distance

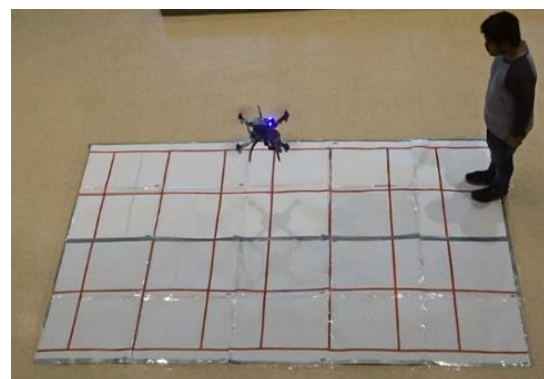
Another response of the drone moving to the backward to the desired distance from the detected human is seen on Figure 30. The settling time obtained from this response is approximately 3.7 seconds. The minimum and maximum steady-state errors are 0.236m and 0.462m respectively.

### Overall Flight Tests

The tuned human following drone was tested in

indoor flights. To further demonstrate the ability of the drone to follow the basic movements of the detected human, a grid flooring was implemented to verify the proper positioning of the drone. This flight setup is shown on Figure 31. Each side of square on the grid is 0.5 meter in length. In this setup, the detected human walks around the grid, and the drone is tested to follow all the movements of the human.

The desired X coordinate error of the bounding box is zero since the aim of this drone is to center the detected human to the frame. An error allowance of  $\pm 5$  pixels from the center of the frame. The desired distance used is 2.25 meters from the detected human. An error allowance of  $\pm 0.25$  meter is also used.



**Figure 31.** Grid Flooring Test

Table 2 shows a sample testing on the movements of the human that must be followed by the drone. Figure 32 shows the error response of the drone due to each movement shown on Table 2.

**Table 2.** Overall Movement Testing 1

Movement	Distance (m)
Left	0.5
Left	0.5
Right	0.5
Forward	0.5
Forward	0.5

Figure 32 shows the response of the drone to the 5 movements described in Table 2. The green lines on Figure 32 depicts the timing of each movement indicated in Table 2. This means that the response of the drone to the first movement, which is a 0.5 meter step to the left, is shown on 0 to 12 seconds of the graph on Figure 32.

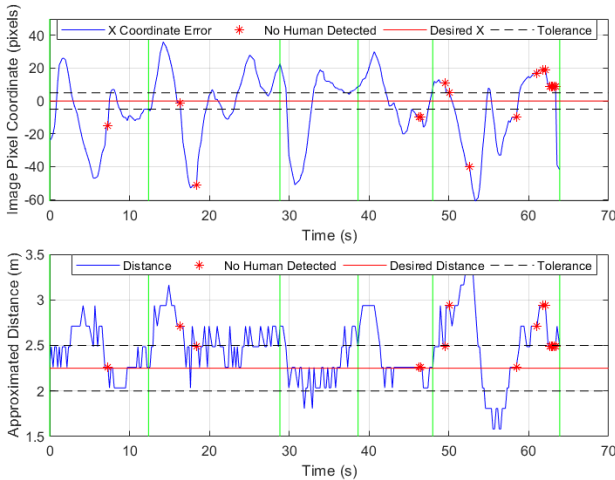


Figure 32. Overall Movement Testing 1

As seen on Figure 32, the drone is able to achieve following the movement of the target. For each movement, the X coordinate Error approaches zero. The distance computed is also approaching its desired value which is 2.25m with an error allowance of  $\pm 0.25$ m. This shows a proper response of the drone from the movement testings indicated in Table 2. The track duration of this flight is approximately 1 minute, and the tracking is lost only 6.391% of the duration.

Another movement testing is performed as seen on Table 3. Similar to Table 2, the response of the drone is shown on Figure 33. The movements described in the first column is continued by the movements on the third column. Similarly, the distances described in the second column is continued in the fourth column.

Table 3. Overall Movement Testing 2

Movement	Distance (m)	Movement	Distance (m)
Right	0.5	Forward	0.5
Right	0.5	Forward	0.5
Left	0.5	Backward	0.5
Left	0.5	Backward	0.5
Forward	0.5	Forward	0.5

Figure 33 shows the response of the drone to the human movements indicated in Table 3. This shows a proper response of the drone from the movement testings indicated in Table 3. The track duration of this flight is approximately 1 minute and 45 seconds, and the tracking is lost only 4.88% of the duration.

Another movement testing is performed as seen on Table 4, and its response is shown on Figure 34. The movements described in the first column is continued by the movements on the third column. Similarly, the

distances described in the second column is continued in the fourth column.

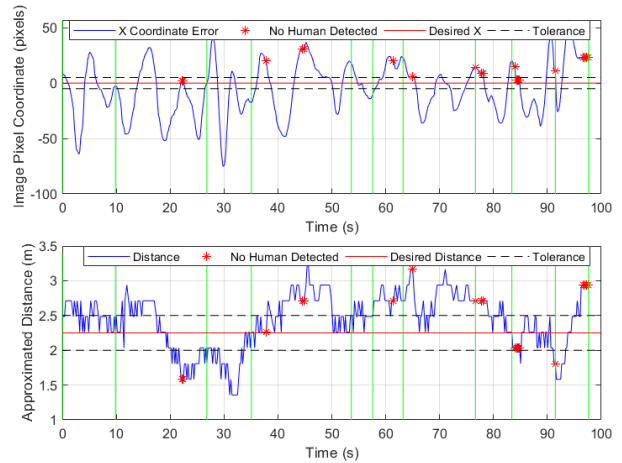


Figure 33. Overall Movement Testing 2

Table 4. Overall Movement Testing 3

Movement	Distance (m)	Movement	Distance (m)
Right	0.5	Backward	0.5
Forward	0.5	Left	0.5
Backward	0.5	Right	0.5
Right	0.5	Left	0.5
Right	0.5	Right	0.5
Left	0.5	Forward	1
Right	0.5	Backward	1
Backward	1	Left	0.5
Forward	0.5	Left	0.5
Right	0.5	Right	0.5
Forward	1	Right	0.5
Backward	0.5	Forward	1
Backward	0.5	Backward	1

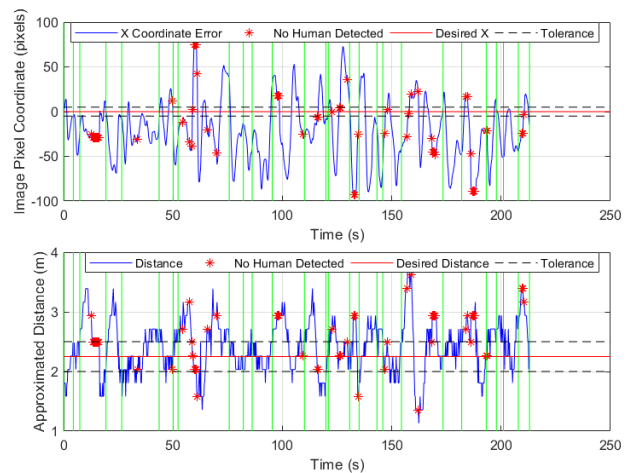


Figure 34. Overall Movement Testing 3

Figure 34 shows the response of the drone to the human movements indicated in Table 4. This shows a

proper response of the drone from the movement testings indicated in Table 4. The track duration of this flight is approximately 3 minutes and 45 seconds, and the tracking is lost only 7.7426% of the duration.

**Table 5.** Summary of Single Movement Testing

Movement	Steady-State Error		Settling Time (seconds)
	Minimum	Maximum	
Right Test 1	-11 pixels	-16 pixels	4.3
Right Test 2	-10 pixels	-13 pixels	4.9
Left Test 1	-7 pixels	-12 pixels	6
Left Test 2	-3 pixels	-7 pixels	6
Forward Test 1	0.01 m	-0.216 m	5
Forward Test 2	-0.216 m	-0.442 m	4.2
Backward Test 1	0.236 m	0.462 m	3
Backward Test 2	0.236 m	0.462 m	3.8

A summary of the single movement testing is shown on Table 5. This shows the maximum and minimum steady-state error of each movement with the percentage of tracking lost.

**Table 6.** Summary of Overall Movement Testing

Testing	Flight Duration	Tracking Lost
Test 1	1 min	6.391%
Test 2	1 min 45 sec	4.88%
Test 3	3 min 45 sec	7.7426%

A summary of the overall movement testing is shown on Table 6. This shows the flight duration and the percentage of tracking lost.

## Conclusion

The human-following drone was successful in detecting and following a chosen human. The human detection via local context is a useful algorithm able to detect humans despite longer distances and pose changes. As seen on Figure 20 to Figure 30, the human-following drone is able to accomplish following the single movements performed by the detected human as the responses approaches their desired values. The maximum steady-state error for the right, left, forward, and backward movements are -16 pixels, -12 pixels, -0.442 m,

and 0.462m respectively with average settling times of 4.6, 6, 4.6, and 3.4 seconds respectively. Since these responses are approaching zero with low steady-state errors, the human-following drone is capable of following its desired detected human.

The overall movement testing of the human-following drone yielded a maximum of 7.7426% tracking lost for the duration with a longest tracking duration of approximately 3 minutes and 45 seconds. Because the human-following drone was able to respond correctly with all the human movements and maintain its track in all its overall flight tests, a new method of navigation is achieved. For future studies, a different human detection algorithm may be employed. The human features used for the algorithm determines the operating distance of the drone is able to detect the human. Another future study can also implement yaw movements. Because of the difficulty of the implementation of yaw controls, this study only focused on following by Roll and Pitch movements. This study may also be applied to outdoor situations and to different UAV frames such as hexacopters, helicopter, fixed-wing, and other applications.

## Acknowledgements

The authors would like to acknowledge the Department of Science and Technology – PCIEERD with project number 04254 and the University Research Coordination Office in De La Salle University for funding and supporting this research.

## References


- [1] R. Bartak and A. Vyskovsk, "Any Object Tracking and Following by a Flying Drone", 2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI), 2015.  
<https://doi.org/10.1109/MICAI.2015.12>
- [2] J. Pestana, J. Sanchez-Lopez, S. Saripalli, and P. Campoy, "Computer Vision Based General Object Following for GPS-denied Multirotor Unmanned Vehicles", 2014 American Control Conference, 2014.  
<https://doi.org/10.1109/ACC.2014.6858831>
- [3] C. Wang, R. Zhao, X. Yang, and Q. Wu, "Research of UAV Target Detection and Flight Control Based on Deep Learning", 2018 International Conference on Artificial Intelligence and Big Data (ICAIBD), 2018.  
<https://doi.org/10.1109/ICAIBD.2018.8396188>
- [4] R. Dan, U. Shah, and W. Hussain, "Development



- Process of a Smart UAV for Autonomous Target Detection", 2018.  
<https://doi.org/10.18687/LACCEI2018.1.1.480>
- [5] H. David Mathias, "An Autonomous Drone Platform for Student Research Projects", *Journal of Computing Science in Colleges*, vol. 31, issue 5, pp. 12-20, 2016.
- [6] J. Cuevas, A. Chua, E. Sybingco, and E. Bakar, "Identification of River Hydromorphological Features Using Histograms of Oriented Gradients Cascaded to the Viola-Jones Algorithm", *International Journal of Mechanical Engineering and Robotics Research* Vol. 8, No. 2, March 2019  
<https://doi.org/10.18178/ijmerr>
- [7] A. Rabie, C. Lang, M. Hanheide, M. Castrillon-Santana, and G. Sagerer, "Automatic Initialization for Facial Analysis in Interactive Robotics", *Proceedings of the 6th international conference on Computer vision systems*, 2018.  
[https://doi.org/10.1007/978-3-540-79547-6\\_50](https://doi.org/10.1007/978-3-540-79547-6_50)
- [8] M. Santana, H. Kruppa, and B. Schiele, "Fast and Robust Face Finding via Local Context", 2003.  
[https://doi.org/10.1007/978-0-387-88777-7\\_3](https://doi.org/10.1007/978-0-387-88777-7_3)
- [9] P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features", *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001.  
<https://doi.org/10.1109/CVPR.2001.990517>
- [10] R. Lienhart and J. Maydt, "An Extended Set of Haar-like Features for Rapid Object Detection", *Proceedings. International Conference on Image Processing*, 2002.  
<https://doi.org/10.1109/ICIP.2002.1038171>
- [11] V. Delica, A. Orquia, J. Piquero, E. Reynaldo, J. Ilaio, E. Sybingco, M. Roque, A. Chua, J. Katupitya, and H. Jayakody, "A New Sliding Mode Controller Implementation on an Autonomous Quadcopter System", *International Journal of Automation and Smart Technology*, vol. 9, no. 2, 2019.  
<https://doi.org/10.5875/ausmt.v9i2.1876>
- [12] I. Gue and A. Chua, "Development of a Fuzzy GS-PID Controlled Quadrotor for Payload Drop Missions", *Journal of Telecommunication, Electronic and Computer Engineering*, vol. 10, Issue 1-5, pp. 55-58, 2018.
- [13] F. Vasconcelos and N. Vasconcelos, "Person-following UAVs", 2016 IEEE Winter Conference on Applications of Computer Vision (WACV), 2016  
<https://doi.org/10.1109/WACV.2016.7477660>
- [14] A. Rosebrock, "Find distance from camera to object/marker using Python and OpenCV", <https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/>. Accessed 26 February 2019.
- [15] J. Ziegler and N. Nichols, "Optimum Settings for Automatic Controllers", 1942.

---

**Publisher:** Chinese Institute of Automation Engineers (CIAE)  
**ISSN:** 2223-9766 (Online)

 **Copyright:** The Author(s). This is an open access article distributed under the terms of the [Creative Commons Attribution License \(CC BY 4.0\)](https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are cited.