# Human Follower Unmanned Aerial Vehicle

Pradip Kathiriya
*University of Maryland*
College Park, MD
pradip@umd.edu

Jeffin Kachappilly
*University of Maryland*
College Park, MD
jeffinjk@umd.edu

## 1 Problem Statement

The problem addressed in this project pertains to the development of an effective human follower capability for unmanned aerial vehicles (UAVs). The objective is to enable UAVs to autonomously track and follow a designated human subject in real-world scenarios. While there have been advancements in UAV technology and autonomous control algorithms, the reliable and precise tracking of a moving person presents several challenges that need to be addressed.

## 2 Motivation

The motivation behind this project lies in the potential applications and benefits of UAVs with human follower capabilities. Such UAV systems can find utility in various domains, including search and rescue operations, surveillance, monitoring, and even entertainment purposes. The ability to autonomously track and follow a human subject offers improved situational awareness, enhanced operational efficiency, and increased flexibility in diverse environments. By developing reliable algorithms and methodologies for human follower tasks, the effectiveness and applicability of UAVs can be significantly expanded, leading to advancements in several fields and potentially saving human lives in critical situations.

## 3 Experimental set-up

### 3.1 Hardware

The hardware equipment used for this project is as follows. The vehicle setup can be viewed in 1 and 8.

| **Hardware List** | | |
|---|---|---|
| **Type** | **Model** | **Features** |
| vehicle | VOXL m500 | Snapdragon VOXL Flight deck, PX4 flight controller |
| RC Transmitter | DX8 Transmitter | 8 channels, DSMX technology, Backlit screen |
| Camera | VOXL 4k Hi-res sensor | AI enabled video processing, 85 degree FOV |
| Lidar | GARMIN Lite v3 | Resolution - 1cm, Range upto 40m, I2C protocol |

### 3.2 Software

- The main software modules installed for this project are as follows :
  1. **QGroundControl** for Ground Control Station with **ekf2_indoor_vio** and **m500** parameters loaded in local machine.
  2. The following docker container inside vehicle - **ROS melodic** with opencv1.2.
  3. **mavros** and **voxl_mpa_to_ros** packages to interface **mpa** and vehicle using **ROS**.


Figure 1: VOXL m500 with labelled parts

4. Enabled **voxl-tflite-server** for object detection using **YoloV5** model and set it at 10Hz.

- The following modifications were made to integrate Lidar into **mavros** :
  1. Installed additional **mavros_extras** package in docker container.
  2. Made changes to parameters present in **px4_config.yaml** and **px4_pluginlists.yaml**.

## 3.3 Architecture

Figure 2 illustrates the comprehensive system architecture. As depicted in Fig 2. The architecture encompasses a ground station and an onboard computer with an integrated flight controller. The ground control system comprises a local computer that establishes communication with the onboard computer via a wireless network. Mounted on the vehicle, the onboard computer and flight controller interact using the MAVLINK protocol. Additionally, the vehicle is equipped with a camera, Lidar sensor, and motor, each communicating with the onboard computer and flight controller through their respective protocols.

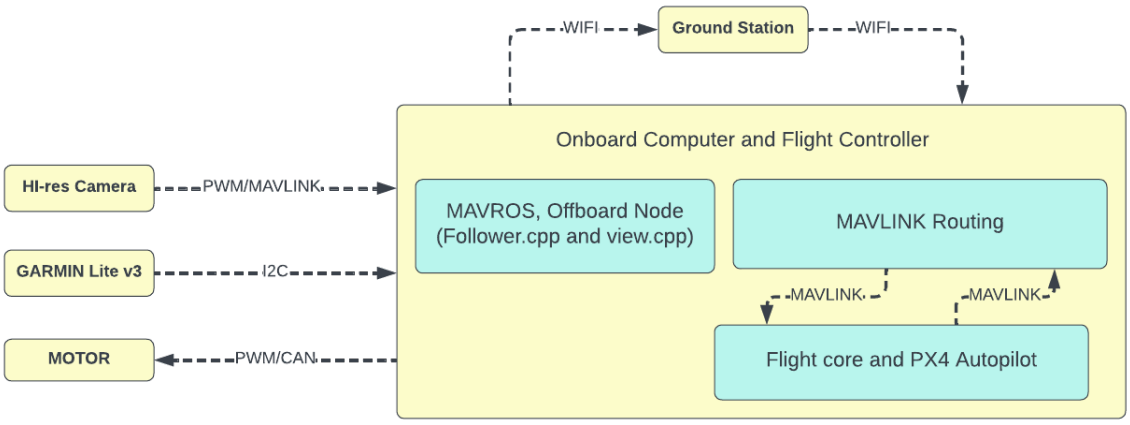



Figure 2: Architecture of the system

# 4 Approach

This section discusses the overall program logic, how different **ROS** nodes interact with each other, and other approaches that did not perform well.

## 4.1 ROS Graph

The graph in 3 visualizes how different **ROS** nodes communicate with each other in off-board mode.

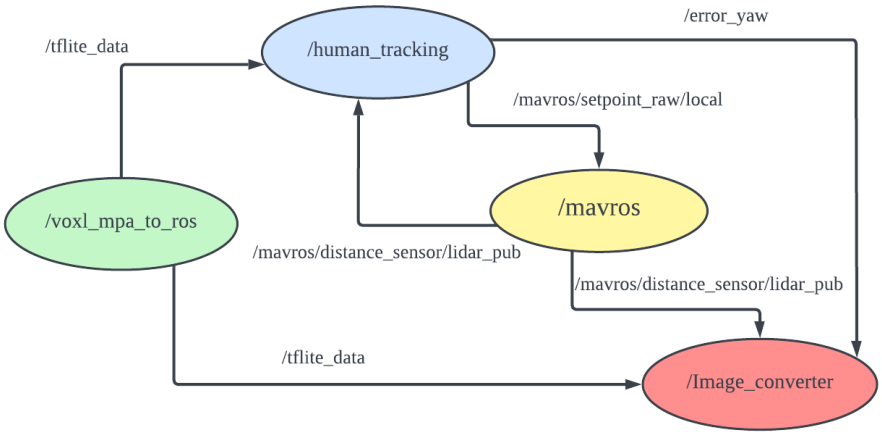



Figure 3: RQT Graph

- ROS topics :
  1. **/tflite_data** - Contains object detection information such as bounding box coordinates and classes detected.
  2. **/mavros/setpoint_raw/local** - Controls vehicle movement by taking position target values.
  3. **/mavros/distance_sensor/lidar_pub** - Contains distance values published by lidar.
  4. **/error_yaw** - Outputs the difference between actual and required yaw for heading control.
- ROS nodes :
  1. **/voxl_map_to_ros** - Converts YoloV5 output from Voxl MPA (Modular Perception and Autonomy) to ROS messages.
  2. **/human_tracking** - Takes object detection data and lidar values as inputs and sends position commands to **/mavros** in off-board mode.
  3. **/mavros** - Publishes lidar data and subcribes to /human_tracking node for position commands.
  4. **/Image_converter** - This node publishes a vehicle camera feed with a bounding box around the **person** class and displays text showcasing vehicle movement.

## 4.2 Flowchart

The chart visualizes logic flow for **/human_tracking** node.

- Position command values are sent in inertial frame.
- Bounding box information used for Yaw PD controller.
- Lidar distance used for Depth PD controller.

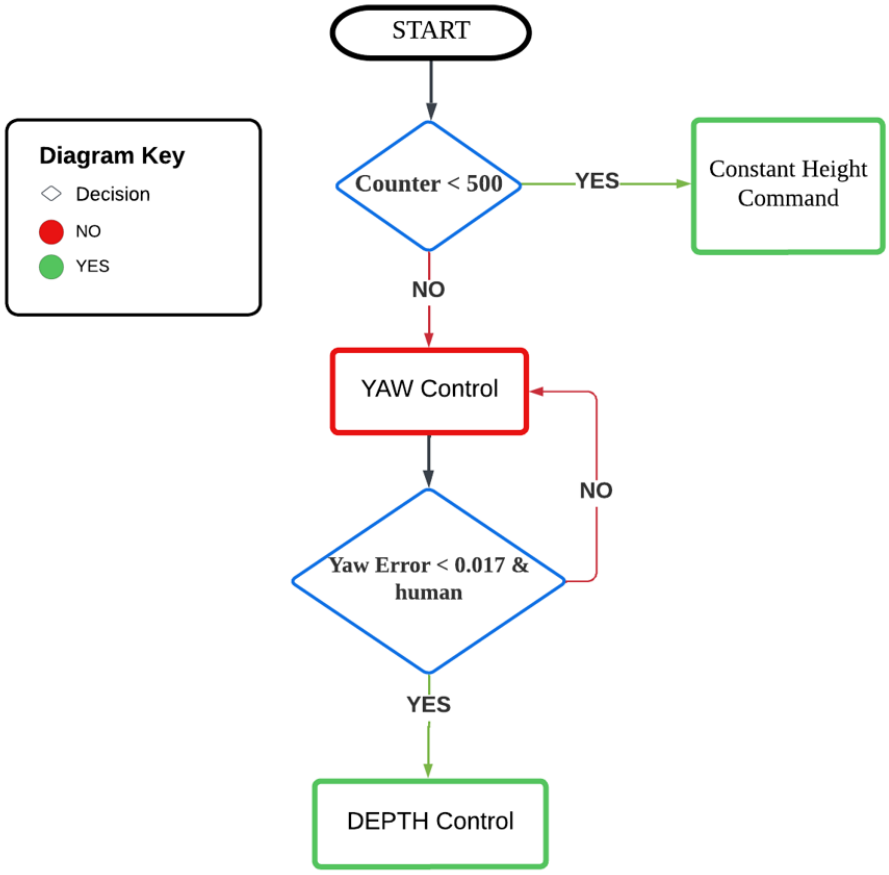



Figure 4: Controller node

Flowchart explanation:

1. When the node starts, constant height is sent to the vehicle so that the initial error does not build up for the Yaw PD controller.
2. After counter condition, Yaw control is executed only if the person is detected.
3. Only if the yaw error is less than the specified value, the PD depth control loop will be executed.
4. The program can only be terminated when the ROS node is manually shut down.

## 4.3 Yaw Controller

The yaw angle is controlled based on the bounding box information obtained from the YoloV5 service, which operates on images of size 640x480 pixels. In Figure 5, the camera center is depicted, representing the location at 320 pixels in the horizontal direction. As illustrated in Figure 6, the horizontal location of the human center is computed. The objective of the yaw controller is to align the image center and the bounding box center, making them coincide.

- Yaw control logic:
    1. The center pixel value of bounding box [BB] is converted into the required yaw angle empirically. BB center at image center corresponds to 0 radian. If BB center is left to image center, then computed yaw be +ve and vice-versa.

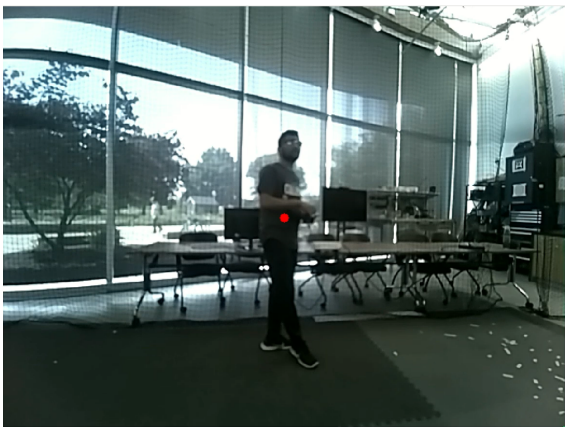


Figure 5: Red circle - Camera center

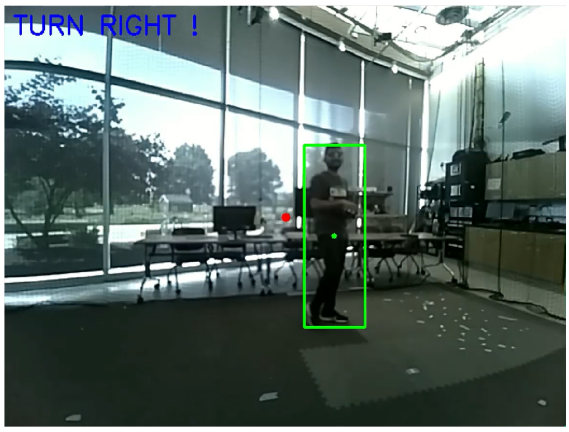



Figure 6: Green Circle - Human Bounding box center

2. PD controller is programmed to reduce the yaw error.

$$\textbf{Error} = \textbf{Current} - \textbf{Required} \tag{1}$$

$$\textbf{computed\_yaw} = \mathcal{K}_p * \textbf{Error} + \mathcal{K}_d * (\textbf{Error} - \textbf{Prev\_error}) \tag{2}$$

3. If person is present, then **computed_yaw** gets added to **prev_yaw** and current **Error** is stored.
4. The final yaw value is published to **/mavros/setpoint_raw/local** at constant height.

### 4.4 Depth Controller

The vehicle's forward and backward motion for **complete mobility** within the entire 2D plane is accomplished by utilizing distance values obtained from the 1D lidar output. In order to ensure safety, the vehicle is consistently instructed to maintain a distance of 2.5 meters from any detected person. The Lidar is mounted in same direction as camera center 8.

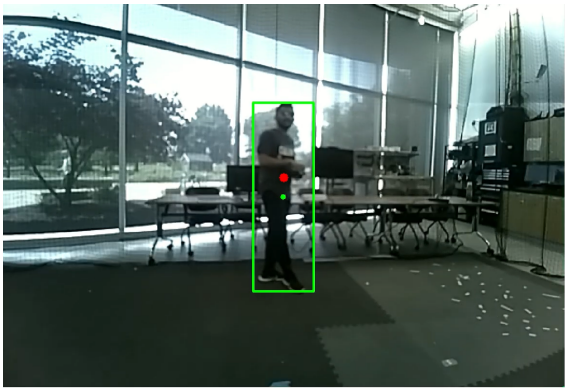


Figure 7: Depth control starting condition

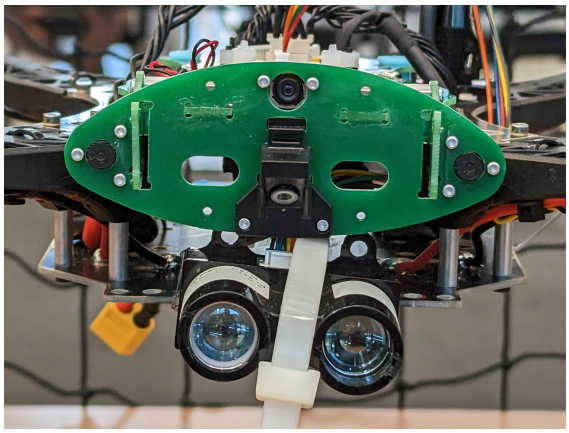


Figure 8: Hi-res camera and lidar arrangement

- Depth control logic:
  1. Depth control only starts when **yaw_error** is close to 0 as shown in figure 7.
  2. PD controller is programmed similarly to yaw control to maintain a distance of 2.5m.

3. If person is present, then **computed_depth** gets added to **prev_depth** to obtain **dist_to_send** and current **Error** is stored.
4. The **x** and **y** values for position control to be sent in the inertial frame are computed as follows.

$$\mathbf{x} = \mathbf{x\_previous} + \mathbf{dist\_to\_send} * \mathbf{cos(yaw)} \tag{3}$$

$$\mathbf{y} = \mathbf{y\_previous} + \mathbf{dist\_to\_send} * \mathbf{sin(yaw)} \tag{4}$$

### 4.5 Custom View node

This subsection explains the working of **Image_converter** node from 3. The purpose of this node is to present a **real-time feed** from the vehicle's camera, which includes bounding boxes around **detected humans**. Additionally, it provides **textual indications** regarding the vehicle's movement, such as whether it is moving forward, backward, turning left, or turning right, as in 6.

The YoloV5 messages are in ROS image format, so **cv_bridge** package was utilized to convert them into OpenCV images. Then text commands were printed based on error conditions from **human_tracking** node. After the modifications, they were reverted back into ROS format.

### 4.6 Different Attempted Approaches

Following are different approaches that were carried and reasons for not making it into final implementation:

1. **Bounding Box height for depth control**: A fixed box height was set for depth control, and the vehicle adjusts the motion (forward or backward) based on changes in the bounding box height. However, the bounding box remained unstable even when the person was stationary resulting in unstable control of the vehicle.
2. **Velocity control for tracking**: Tested both velocity and position control, and the vehicle seemed more stable and controllable in position control.

## 5 Results

The algorithm is assessed through a sequence of trials wherein the vehicle tracks the movement of a person. The evaluation metrics comprise the standard deviation of two vehicle control parameters: Yaw and Depth. The ensuing table 1 illustrates the standard deviation of these parameters at the steady-state position of the vehicle during one of the experiments. The algorithm's precision and safety in tracking a person are determined by evaluating its standard deviation.

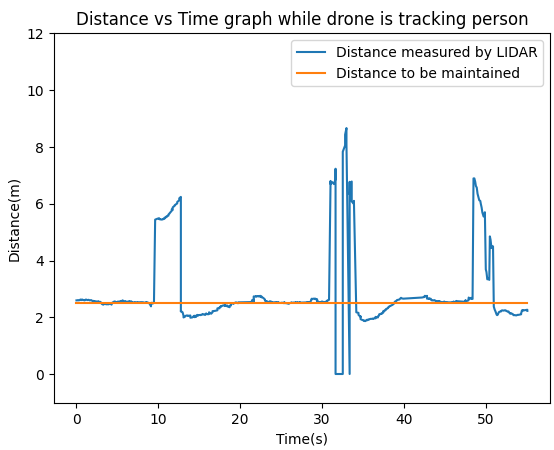
For illustration, Figure 6 shows the Hi-res camera feed as the vehicle tracks the person. The green dot is the person's center and the red dot is the image center.

Table 1: Steady State Standard Deviation of Distance and Yaw angle

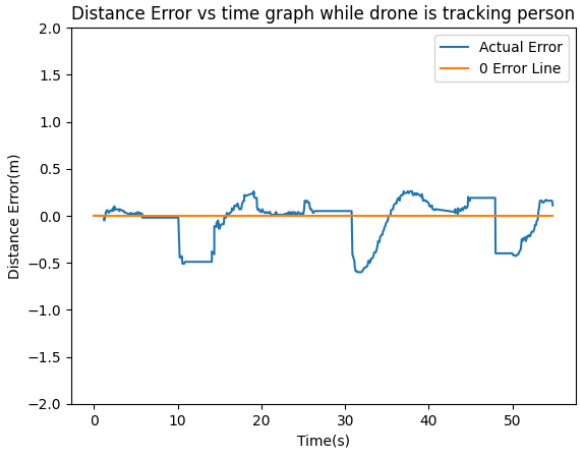
| **Parameters** | **Standard Deviation** |
|---|---|
| Distance | 0.088037 |
| Yaw Angle | 0.00413 |

The accompanying graphs illustrate the dynamic changes of this parameter and its associated error as the vehicle follows the person.

Figure 9 illustrates the dynamics of distance (depth) and distance error as the vehicle tracks the targeted person. The spikes observed in the graph 9a and 9b denote instances where the person is in motion, causing the vehicle to enter a transient state as it adjusts to follow the person's movements. Conversely, the flat line segments indicate periods when the person remains stationary, allowing the vehicle to achieve a steady state position. Notably, during these steady-state intervals, the distance error converges towards zero, signifying precise tracking, while the steady-state distance aligns closely with the desired distance to be maintained.

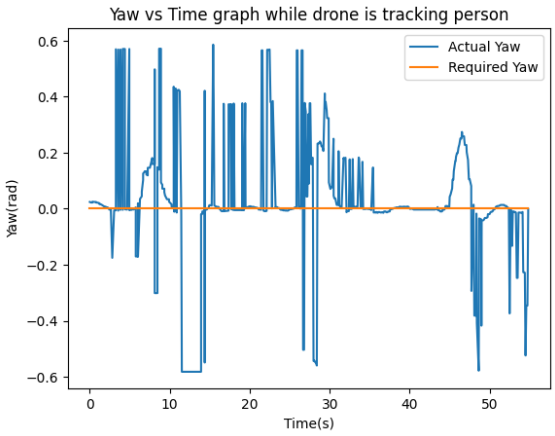



(a) Distance vs Time

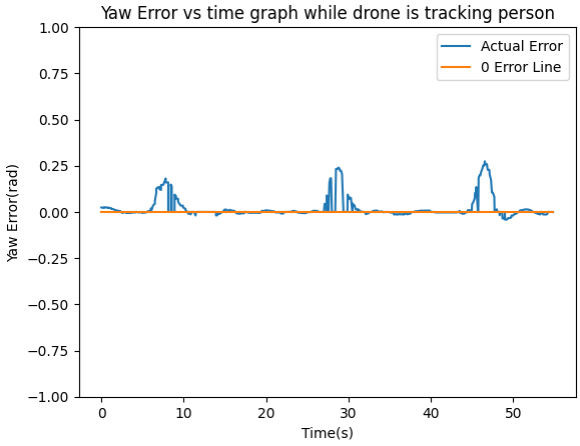



(b) Distance Error vs Time

Figure 9: Dynamics of Distance and Distance Error during tracking




(a) Yaw vs Time




(b) Yaw Error vs Time

Figure 10: Dynamics of Yaw and Yaw Error during tracking

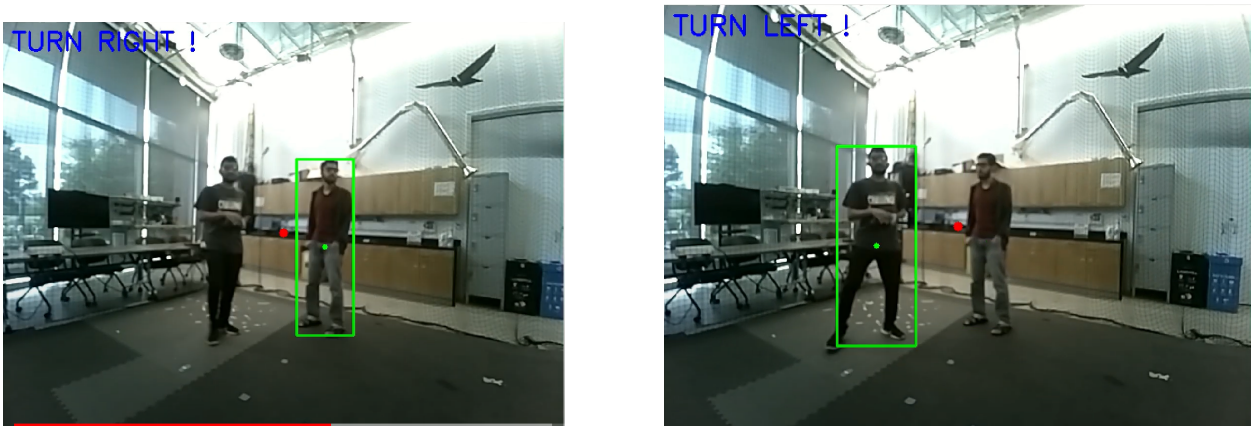
Figure 10 depicts the dynamics of the yaw and yaw error as the vehicle tracks the person. Figure 10b illustrates the corresponding dynamics of the yaw error. The spikes observed in the graph indicate instances when the person is in motion, resulting in transient states in the vehicle's yaw response. Conversely, the flat line segments signify periods when the person remains stationary, allowing the vehicle to attain a steady-state position. In contrast to the consistent behavior observed in distance dynamics, the yaw dynamics exhibit inconsistencies. The yaw error showcases a distinct pattern of spikes and flat lines, indicative of discernible behavior. However, the yaw readings themselves demonstrate significant noise. This is a consistent observation across multiple experiments. One plausible explanation is that: the *Yolo* detection algorithm is inconsistent and, occasionally fails to detect the person in certain frames. Consequently, the algorithm compensates for the missed detection by assigning random and high values. Notably, the yaw control mechanism exhibits a high degree of consistency and swift response while observing the actual vehicle following the person. Hence, a detailed investigation is necessary to understand this yaw behavior.

# 6 Edge cases and Future work

As described in the previous section, the developed algorithms demonstrated satisfactory performance in the laboratory setting, implying their potential effectiveness in real-world scenarios. However, certain situations arise where the algorithm does not exhibit the desired behavior. Therefore, further work is required to enhance the algorithm's reliability and robustness.

### Case 1: Multiple persons are in FOV of camera

As shown in fig 11, When there are multiple people in the field of view of the camera, the algorithm exhibits undefined behavior. As shown in fig 11a & 11b, the vehicle instantaneously changes the tracking from the person on the left side to the person on the right side, resulting in undefined behavior. This behavior stems from the fundamental limitation of the *yolo* to distinguish similar objects e.g. distinguishing one person from another. Yolo classified both the person as 'person' rather than 'person 1' and 'person 2'. To make the algorithm full-proof against this edge case, it is necessary to develop an algorithm on the top of *yolo* to consistently track a single person when there are multiple people in FOV. Alternatively, the use of 'April tag' or 'QR-code' can be a potential solution for this edge case. In both cases, future work is necessary to make this algorithm more generalized.




(a) Vehicle tracking person on right side

(b) Vehicle tracking person on left side

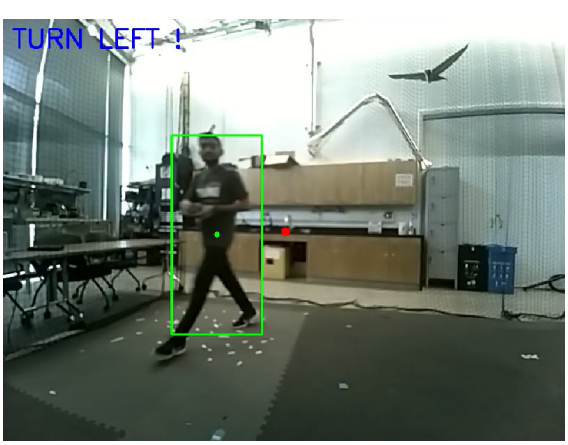
Figure 11: Edge case 1: There are multiple people in FOV of camera

## 6.1 Case 2: When the person is moving at a very high pace

As shown in fig 12, when the person to be tracked moved at a very high pace, the vehicle lost track of the person resulting in undefined behavior. It is empirically observed that this happens when a person moves at $\gtrapprox 1.5m/s$. This upper bound in the speed is governed by the feed speed of the Hi-res camera. The feed speed of the camera is set to $10Hz$ (the actual feed is about $7-8Hz$) in order to get consistency in bounding box prediction. It is observed that when the camera feed speed is higher than $10Hz$, the predicted bounding box is very noisy resulting in unpredictable Yaw control. A couple of the potential solutions for this problem is to develop an algorithm to smooth out the noise in the predicted bounding box, and use of an upgraded version of *yolo* (v8 has pixel-level segmentation and predicts an accurate bounding box). In any case, future is work in necessary to make this algorithm robust against this edge case.

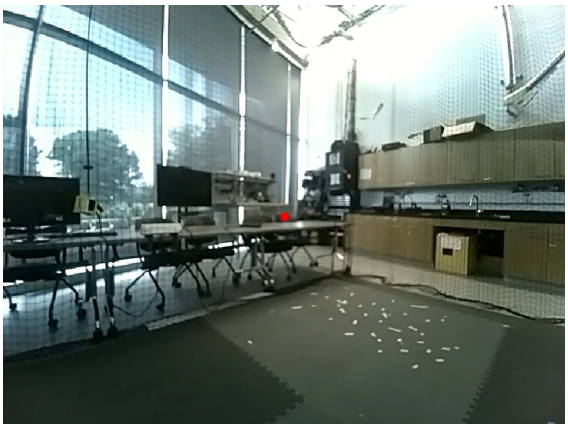
# 7 Conclusion

This project aims to develop an algorithm capable of enabling unmanned aerial vehicles (UAVs) to track human subjects. The experimental setup involved utilizing ModalAI's VOXL m500 quadrotor equipped with




(a) Vehicle tracking person moving at high pace

(b) Vehicle lost the track of person



(c) person is out of field view of camera

Figure 12: Edge case 2: Person moving at high pace

a PX4 flight controller. The key sensors employed by the algorithm include the VOXL 4k HI-res sensor and the GARMIN Lite v3 Lidar.

The algorithm itself comprises two primary controllers: the Yaw controller and the Distance controller. The Yaw controller utilizes data from the HI-res camera and employs the YOLO algorithm to detect the person, subsequently calculating the Yaw error. The controller then adjusts the yaw angle of the vehicle to minimize the yaw error. On the other hand, the Distance controller employs the Lidar sensor to calculate the distance error and modifies the X and Y position of the vehicle to minimize the distance error.

To validate the algorithm's capabilities, a series of laboratory experiments were conducted wherein the vehicle successfully tracked the person. The algorithm achieved a steady-state standard deviation of 0.08803 & 0.00413 for Yaw & Distance, respectively. Additionally, this report presents the limitations encountered during the experimentation and proposes potential mitigation strategies.

## References

1. https://docs.modalai.com/voxl-tflite-server/
2. http://wiki.ros.org/cv$_b$*ridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages*
3. https://youtu.be/unwS78C-N$_c$
4. https://github.com/mavlink/mavros