Human-Drone Interaction

Zdravko Marinov, Stanka Vasileva, Qing Wang

Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology, Germany

{ufijd, uredo, ukwet}@student.kit.edu

Abstract

Drones have become a common tool, which is utilized in many tasks such as aerial photography, surveillance, and delivery. However, operating a drone requires more and more interaction with the user. A natural and safe method for Human-Drone Interaction (HDI) is using gestures. This paper describes an HDI framework, which is based on the Robot Operating System (ROS) middleware. Our framework provides the functionality to control the movement of the drone with simple arm gestures and to follow the user while keeping a safe distance. We also propose a monocular distance estimation method, which is entirely based on image features.

1. Introduction

Drones, or Unmanned Ariel Vehicles (UAVs), are becoming more and more common in everyday activities. Although drones are often used for photography and filming, they are also utilized for navigating visually impaired persons[4]. Drones are often operated via a remote control[6] or follow a pre-defined route[5], which limits their capabilities of interaction with humans.

To tackle this problem, we developed a framework, which enables the drone to recognize gestures and react to them in real-time. The drone's movement can be controlled by a pre-defined set of arm gestures, making it possible to maneuver the UAV intuitively by any user without the need for any prior technical knowledge. We also provide a face-following mode, in which the drone follows the user's face and keeps a safe distance away from him.

1.1. Preliminaries

We used the DJI Tello Drone[1] to implement our framework. The Tello drone is a quadrotor, which has a 5MP 720p RGB frontal camera and a distance sensor on its bottom side to stabilize its flight. The lack of a frontal depth sensor and a Bluetooth interface encouraged us to implement a monocular image-based distance estimation to the user. To control the movement of the drone and receive information about its state, we utilized the Tello SDK[3], which allows connecting to the drone via a WiFi UDP connection. Through this connection, text commands for movement control can be sent as well as queries about the current state of the drone (*e.g.* the current altitude).

Figure 1: DJI Tello Drone[1]

1.2. Objectives

In this paper, we have two main objectives, which are characterized by two different modes for the drone's movement control. The first mode is to use face following to track the user's face by keeping a certain distance from him. The second mode consists of controlling the drone with arm gestures, in which the drone performs a specific command corresponding to the gesture. The goal is that in both movement control modes, the drone recognizes a concrete user and reacts only to their gestures or tracks only their face to avoid confusion in crowded scenes.

2. Modules Overview

Our framework is displayed in Figure 2. Its pipeline consists of five modules, which are implemented as ROS Nodes and communicate with each other with ROS Topics. ROS Nodes can subscribe and publish to a topic, i.e., they can receive input data and send output data through the topic. This is indicated by the gray arrows in Figure 2.

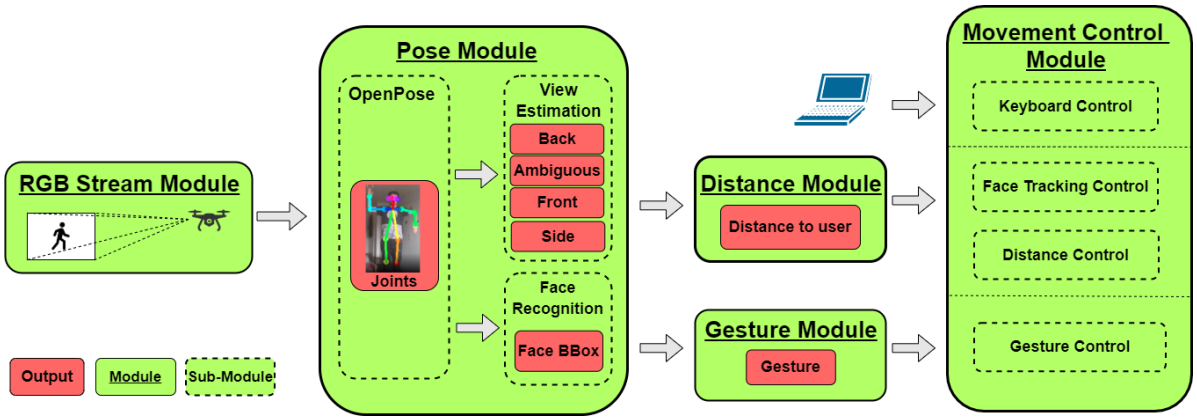

Figure 2: Our framework's pipeline. Each green box represents a ROS Node. The arrows show the ROS Topics and their corresponding subscribers and publishers. The outputs, indicated in red, are the data which is published in the ROS Topics.

2.1. RGB Stream Module

The RGB stream module serves to provide the images, captured by the drone's camera, to the pose module. The frames are retrieved via a UDP connection with the drone and are forwarded by maintaining a pre-defined frame rate. The reasoning behind this separate utility module is to make it possible to replace it with any camera device, such as an integrated webcam or another drone's camera. In addition, this is the only point, where frames are being transferred via ROS, which reduces the transmission overhead in the rest of the framework.

2.2. Pose Module

The pose module receives images from the RGB stream module and estimates three features. Firstly, the 2D joint locations of all people in the frame are estimated with CMU's OpenPose[7]. This includes 18 joints per person. These locations are used as features for the view estimation and face recognition sub-modules as indicated by the gray arrows inside the pose module in Figure 2.

2.2.1 View Estimation

The view estimation sub-module uses the 2D joint locations to classify the orientation of the person into the view classes $v_i \in \{v_{\text{Front}}, v_{\text{Side}}, v_{\text{Back}}, v_{\text{Ambiguous}}\}$. It is achieved by a purely geometrical approach by estimating the angle of the shoulder axis with the image plane. Since OpenPose's joint locations are 2D, the real 3D angle with the image plane can only be approximated. To accomplish this, we consider the ratio of the shoulder width to the nose-to-neck distance of the OpenPose joints. The motive behind this is that the nose-to-neck distance is significantly independent of the person's orientation, whereas the shoulder width becomes smaller when the person is standing sideways. We also only consider the ratio between the two distances, which makes this approach viable at any distance from the camera. Thus, we classify the view as v_{Side} if this condition holds:

$$d_s \leq 0.5 \cdot d_{ntn} \tag{1}$$

where d_s is the shoulder width and d_{ntn} is the nose-to-neck distance. If condition (1) does not hold, we check whether the left ear and shoulder are both on the left side (with respect to right ear and shoulder) and classify the view as v_{Front}. Similarly, if both are on the right side, the view is classified as v_{Back}. However, if none of these conditions hold, the view class is $v_{\text{Ambiguous}}$. We introduced this class so that the definitions of the other view classes are more strict and more consistent.

2.2.2 Face Recognition

Lastly, the face recognition ensures that only one concrete person's gestures influence the drone and only his face is followed. Other people's gestures and faces in the frame are ignored. There is no conventional face detection in this module. Instead, the bounding box of the face is directly derived from the joint locations of the face (*i.e.* ears, eyes, nose, and neck). To capture the whole head, the bounding box is extended by a fixed margin across the width and height. This method makes it possible to infer the user's face bounding box from all views, even where there is no apparent face on the frame (*e.g.* from the back view).

Afterward, the image crop from the bounding box is forwarded to the face recognition. We use a Python implementation of DLIB's[2] CNN based face recognition. The method first decides whether there is a face on the input image and then computes the distance between its embedding

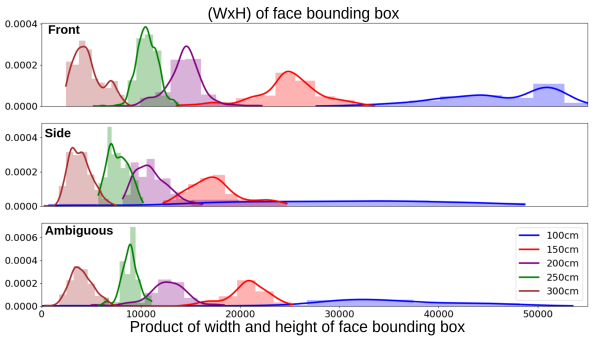

Figure 3: Distribution of the input feature width-height product for the distance estimation

in the latent space to all the face embeddings from a predefined template database. If the distance to the best match is small enough, it classifies the input image as the match's identity and only considers them further in the pipeline.

2.3. Distance Module

The distance module is responsible for estimating the distance to the user. The input for the module is a 7-dimensional vector. It consists of the width W, height H, and width-height product $W \times H$ of the face bounding box of the user. Additionally, the input also contains a one-hot-encoding[1] of the estimated view class v_i and the 2D euclidean neck-to-waist distance from the OpenPose joints. The final estimated distance is a scalar value.

The distance estimation is formulated as a classification problem with the target distance classes $c_i \in \{c_{100}, c_{150}, c_{200}, c_{250}, c_{300}\}$, where the indices represent a distance in centimeters. In Figure 3 we can see that the distributions of the width-height product in our training dataset have a distinct shape with slightly overlapping regions for all distance classes. The distinct shapes indicate that a well-trained classifier would be able to separate the classes in the latent space. However, we also see why the one-hot-encoding of the view is needed as an input feature. If the view was not included, the distributions in Figure 3 would have a significant overlap (*e.g.* 200cm in the side view and 250cm in the front view).

We apply a fully-connected neural network (FCNN) to solve this classification problem. The FCNN contains 4 hidden layers with 1000 hidden units each. A dropout layer is added to avoid overfitting and a residual connection, which proved to decrease the training time. The output is a softmax layer, which models a posterior distribution over the distance classes. The training dataset for the FCNN consists of 960 images of each distance class (4800 samples in total), where a person performs different gestures to simulate a real-use scenario. The network requires 15 epochs of

[1]We aggregate v_{Back} and $v_{\text{Ambiguous}}$ into one class.

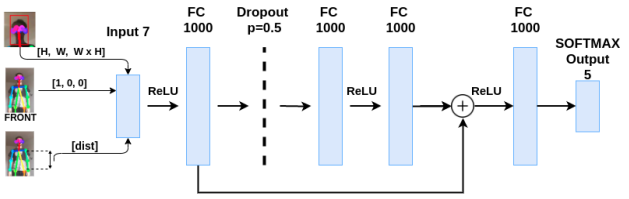

Figure 4: Distance Estimation FCNN Architecture

training with the cross-entropy loss to converge.

To achieve a continuous estimation from the FCNN, a weighted sum of the distance classes c_i with their corresponding softmax outputs s_i is computed as $\sum_i c_i s_i$. However, class labels c_i with a softmax output of s_i are weighted by 0 if $|s_{max} - s_i| \geq 0.1$, where s_{max} is the largest softmax output. The weighted sum is then computed by first normalizing the remaining non-zero softmax weights so that the output is a convex combination of the class labels. The estimated distance is then forwarded to the movement control so that the drone can keep a safe distance from the user.

2.4. Gesture Module

The goal of the gesture module is to recognize arm gestures from a user, which could be either a frontal or a lateral[2] gesture. The module receives the joint locations of the user from the OpenPose module as well as the estimated view class. The gesture module determines whether a gesture is present and if the same gesture is recognized for several frames a gesture control command is forwarded to the movement control.

The idea of the gesture recognition is based on a purely geometrical approach and requires that both of the user's arms are visible. According to the elbow angle the **angle state** of each arm is classified as *Straight*, *Perpendicular* or *None*. In practice, it is not feasible to require the elbow angle to be exactly at $\alpha_{perp} = 90°$ to be considered as perpendicular or at $\alpha_{str} = 180°$ for a straight state. Therefore, we allow a certain interval of degrees, in which the state is recognized as perpendicular $\alpha_{perp} \in (60°, 120°)$ or straight $\alpha_{str} \in (140°, 220°)$.

Additionally, we also distinguish between the **position states** *Over*, *Under* and in the *Middle*, which refer to the hand's position with respect to the shoulder. If the angle states for both arms are not *None*, the angle and position states are combined to classify a gesture. For example, in Figure 5 we see for both arms the angle state *Perpendicular* and the position state *Over*, which constitute the "Up" gesture. A gesture is only forwarded to the motion control when it has been recognized for at least five consecutive frames. If the previously sent gesture is the same gesture, a

[2]Gestures which are seen from the side view.

certain time period must first pass to avoid sending the same gesture too often *(see Figure 5)*.

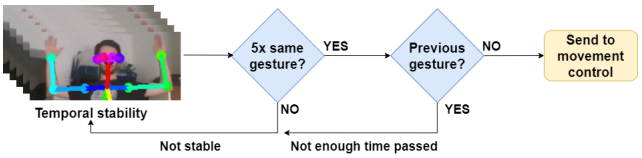

Figure 5: Temporal stability of the recognized gestures

2.5. Movement Control Module

The motion control module includes three modes: keyboard, face tracking, and gesture control mode. The keyboard mode is used to switch between modes and execute other auxiliary actions. The inputs for the face tracking mode are the distance to user d and his face bounding box center coordinates (x, y). The face tracking mode sends speed commands to the drone so that the user's face is kept in the center of the frame and a certain distance is maintained away from him. When the gesture control mode receives a gesture command G, it enables the drone to execute the demanded action.

To control the drone, PID controllers have to be created for each of these movements. The PID controller is a control loop feedback mechanism that computes the deviation between a given value (measured process value) and the desired value (setpoint) and corrects it based on the proportional, derivative, and integral terms. Equation 2 shows how the PID controller functions.

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \qquad (2)$$

where $e(t)$ is the difference between $r(t)$ and $y(t)$. The $r(t)$ value represents the setpoint and the $y(t)$ value the measured process value. The $u(t)$ value is the control signal and is described by the sum of the three terms: the P-term (proportional to the error), the I-term (proportional to the integral of the error), and the D-term (proportional to the derivative of the error).

Each of the PID controllers has to be tuned independently to find the correct parameters that will lead to a smooth movement. With this simple method, we have to set both K_i and K_d parameters to zero and increase K_p until the system reaches an oscillating behaviour. Then, we adjust the K_i parameter to stop the oscillation and finally adjust K_d for a faster response.

Additionally, in order to complete the gesture control, we need to ensure that the drone can automatically fly to the front of the user. We designed the 'front to side' control, which consists of two actions: fly from the side to the front of the user and rotate 90° to face the user.

3. Evaluation

To evaluate our modules we created a custom test dataset, which consists of a person performing all the gestures while rotating to simulate different views and altering the lighting. Over the frames, the gestures are performed with slight modifications concerning the angle and position of the arm. 600 frames are recorded for each distance class, which leads to 3000 test samples in total.

Gesture evaluation. The gesture accuracy depends entirely on the joints estimations from OpenPose and the correctness of the gesture execution from the user. The overall recognition accuracy of the gestures is 93.5%. The only gesture, which is not so easily recognized is the "Down" gesture as it is physically more difficult to perform.

Distance evaluation. The distance estimation is evaluated via the Mean Absolute Error (MAE) metric. The MAE for all the distance classes is estimated at 17.75cm with a mean standard deviation of 19.6cm. However, the farthest distance classes contribute to a much larger MAE (*e.g.* for c_{300}, the MAE is 30.94cm).

4. Conclusion

The results from the evaluation show that despite the hardware limitation of the DJI Tello drone, it is possible to achieve a robust distance estimation as well as reliable gesture recognition. The framework can control the drone in real-time, enabling a natural and simple interaction. Due to its modularity and extensibility, our framework could be integrated and further developed in various others other applications.

References

[1] Tello: Ryze robotics. `https://www.ryzerobotics.com/tello`.

[2] The world's simplest facial recognition api for python and the command line. `https://github.com/ageitgey/face_recognition`.

[3] Tello SDK 2.0 User Guide. `https://bit.ly/3qpnsI3`, 2018.

[4] Mauro Avila, Markus Funk, and Niels Henze. Dronenavigator: Using drones for navigating visually impaired persons. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*, pages 327–328, 2015.

[5] Juan Besada, Ivan Campaña, Luca Bergesio, Ana Bernardos, and Gonzalo de Miguel. Drone flight planning for safe urban operations. *Personal and Ubiquitous Computing*, pages 1–20, 2020.

[6] Francois Callou and Gilles Foinet. Method for the intuitive piloting of a drone by means of a remote control, Nov. 26 2013. US Patent 8,594,862.

[7] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.