Pose2Drone: A Skeleton-Pose-based Framework for Human-Drone Interaction

Zdravko Marinov*, Stanka Vasileva*, Qing Wang*, Constantin Seibold†, Jiaming Zhang† and Rainer Stiefelhagen†

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

*{firstname.lastname}@student.kit.edu, †{firstname.lastname}@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. In this paper, we introduce an HDI framework building upon skeleton-based pose estimation. 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 and does not require any additional depth sensors. To perform comprehensive experiments and quantitative analysis, we create a customized testing dataset. The experiments indicate that our HDI framework can achieve an average of 93.5% accuracy in the recognition of 11 common gestures. The code will be made publicly available to foster future research.

Index Terms—Human-Drone Interaction, Face and Gesture recognition, UAV, Monocular Distance Estimation

I. Introduction

As drones, or Unmanned Ariel Vehicles (UAVs), have become increasingly more accessible, the interest in integrating them into different aspects of our lives is continuously growing. As such, drones are not only used for photography and filming [1], [2], or shipping and delivery [4], but they are also utilized for navigating visually impaired persons [3]. For this task, assisting drones are often operated via a remote control [8]. This hinders the dynamic interaction as one requires either excessive human input [11], outlaying a path via markers [7] or an integrated GPS-module [18]. This issue becomes especially apparent when one wants to translate interactive drones into unseen environments. To tackle this problem, one needs a method that is robust to domain changes such as switching between indoor and outdoor environments and requires a minimally complex user input. Thus, we propose a framework, which is based on a skeleton pose prediction of the human pose. This enables the drone to recognize gestures and react to them in real-time while being robust to domain changes. The drone's movement can be controlled by a predefined set of arm gestures, making it possible to maneuver the UAV intuitively by any user without the need for any prior technical
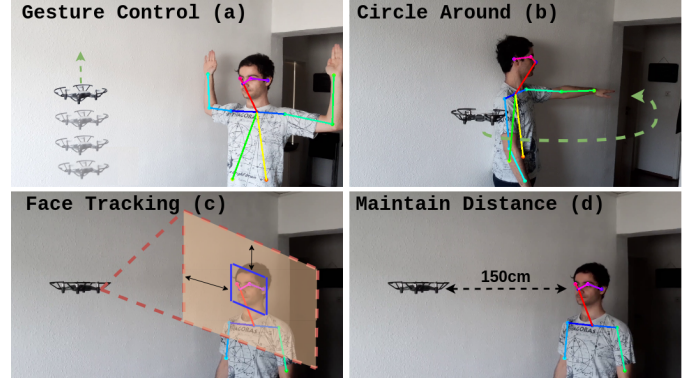
Fig. 1. Examples of our framework's functionality: (a) A frontal gesture command for an upward movement, (b) A gesture which commands the drone to come in front of the user, (c) Face tracking keeps the user's face in the center of the frame, and (d) The drone maintains a distance to the user.

knowledge. We also provide a face-following mode, in which the drone follows the user's face and maintains a safe distance.

Contributions. In this paper, we propose the first, to our knowledge, existing skeleton-pose-based drone control framework. After a specific user registers as main applicant for the drone via facial recognition features, the user's control is characterized by two different modes for the drone's movement. The first mode, Face Following (FF), allows the drone to track the user's face while keeping a certain distance from him as seen in Fig. 1(c) and Fig. 1(d). The second mode, Gesture Control (GC), allows commanding the drone with arm gestures, in which the drone performs a specific action according to a predefined gesture-action pair. The functionality of our framework can be seen in Fig. 1.

II. Related Work

A common method to interact with drones is *e.g.* via a remote control [8]. This way, drone operators can maneuver drones with precision and create highly synchronized drone swarms [11]. However, controlling drones in this manner demands professional prior training, which complicates the interaction for regular users and this type of HDI often requires setting a fixed route or executing a series of predefined movements, which limits the natural interaction with the drone [11]. In contrast, while our HDI framework can integrate remote control via keyboard inputs, it enables any face-registered user
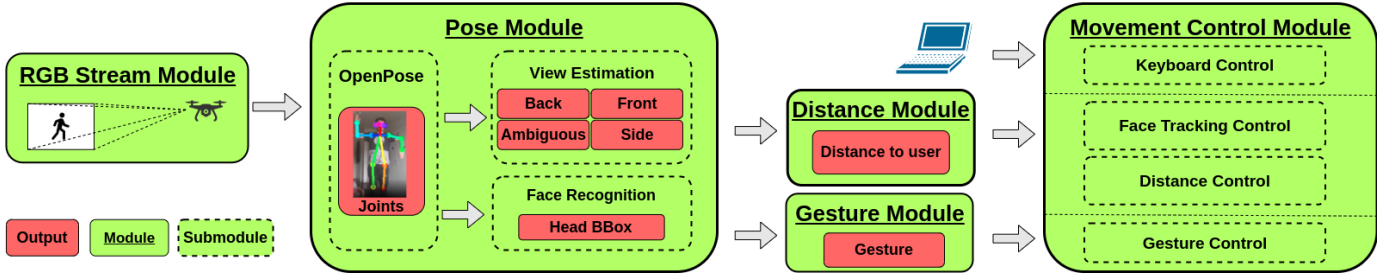
Fig. 2. Our framework's pipeline. Each green box represents a ROS Node. The arrows show the ROS Topics and their corresponding subscribers and publishers. The outputs, indicated in red, are the data which is published in the ROS Topics. All modules are run on a Linux machine.

to safely and intuitively manipulate the drone, utilizing the FF and GC modes.

Gestures are the most commonly used alternative modality for HDI [11] and are well-perceived [18]. Some previous approaches use depth cameras in addition to an RGB stream to determine the visibility, position, and shape of the arms and hands and recognize a gesture [9], [10]. Other methods have been proposed based only on an onboard RGB camera [12], [13]. Such methods extract visual features from the image to recognize the gesture without requiring additional depth sensors. Cauchard et al. [9] use hand gestures for navigating a projected user interface menu and selecting actions for the drone to perform. Another usage of hand gestures is to issue movement commands directly with the gesture, *e.g.*, by pointing at a direction [13]. However, traditional image-based gesture recognition approaches suffer from sensitivity to occlusion and illumination variation [19]. Thus, we propose a skeleton-pose-based framework for gesture recognition, which is shown to be significantly robust to both issues.

III. PROPOSED FRAMEWORK

Our framework is displayed in Fig. 2. Its pipeline consists of five modules, which are implemented as Robotic Operating System (ROS) [20] Nodes and communicate with each other via ROS Topics. ROS Nodes can subscribe and publish to a topic, i.e., they can receive input data and send output data via the topic. This is indicated by the gray arrows in Fig. 2.

A. RGB Stream Module

The RGB stream module serves to provide 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 predefined frame rate. 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.

B. Pose Module

The pose module receives images from the RGB stream modules, which are used to estimate the joint locations $\mathbf{j}_i = (j_i^x, j_i^y) \in \mathbb{R}^2, i \in \{0,..,17\}$ of the user in the image with OpenPose [5]. For our framework, we only consider a subset of all joints. These are labeled with their indices i in Fig. 3 in the "Cheese" gesture. These locations are used as features for the

view estimation and face recognition submodules as indicated by the gray arrows inside the pose module in Fig. 2. We further define the distance $d(i,k)$ between two joint locations $\mathbf{j}_i$ and $\mathbf{j}_k$ as their 2D Euclidean distance:

$$d(i,k) = ||\mathbf{j}_i - \mathbf{j}_k|| \tag{1}$$

1) View Estimation: The view estimation submodule uses the joint locations $\mathbf{j}_i$ 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. We accomplish this by comparing the γ-weighted nose-to-neck distance $d(0,1)$ with the shoulder width $d(2,5)$, where γ is a hyperparameter. We also differentiate between the frontal and back view with the help of the relative positions of the ears to each other.

$$v_i = \begin{cases} v_{\text{Side}}, & \text{if } d(2,5) \leq \gamma \cdot d(0,1) \tag{2a} \\ v_{\text{Front}}, & \text{if } d(2,5) > \gamma \cdot d(0,1) \text{ and } j_{16}^x \leq j_{17}^x \tag{2b} \\ v_{\text{Back}}, & \text{if } d(2,5) > \gamma \cdot d(0,1) \text{ and } j_{16}^x > j_{17}^x \tag{2c} \\ v_{\text{Ambiguous}}, & \text{else} \tag{2d} \end{cases}$$

The motive behind this method 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 compute the ratio between the two distances, which makes this approach viable at any distance from the camera.

If (2a) does not hold, we check whether the left ear is on the left side with respect to right ear and classify the view as v_{Front}. Similarly, if it is on the right side, the view is classified as v_{Back}. However, if all (2a), (2b), and (2c) do not 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. The modularity of our framework also allows to use a deep learning method, but our evaluations show that our geometric method is sufficient.

2) Face Recognition: 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, we use the bounding box of the convex hull of the head joint locations (*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 head 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 submodule, where we use a standard face recognition framework (see Subsection IV-A). This method requires a predefined template database images of faces and searches for the best match to the input bounding box. If the user's face is not matched in the current frame, the closest head bounding box to the last matched face is taken and tracked.

C. Gesture Module

The goal of the gesture module is to recognize arm gestures from a user, which could be either frontal or side gestures. The module receives the joint locations $\mathbf{j}_i$ of the user from the OpenPose module as well as the estimated view class v_i. The gesture module determines whether a gesture is present and if the same gesture is recognized within N frames, then a corresponding gesture control command is sent to the movement control, where N is a hyperparameter.

The idea of the gesture recognition is, similarly to the view estimation, based on a purely geometrical approach and requires that both of the user's arms are visible. According to the elbow angle α, we classify for each arm the angle state $s_\alpha \in \{$Straight, Perpendicular, None$\}$. In practice, it is not feasible to require the elbow angle to be exactly at $\alpha = 90°$ for a perpendicular state or at $\alpha = 180°$ for a straight state. Therefore, we allow a certain interval of degrees, in which the state is recognized as perpendicular or as straight.

$$s_\alpha = \begin{cases} \text{Perpendicular,} & \text{if } \alpha \in (60°, 120°) & \text{(3a)} \\ \text{Straight,} & \text{if } \alpha \in (140°, 220°) & \text{(3b)} \\ \text{None,} & \text{else} & \text{(3c)} \end{cases}$$

Additionally, we also distinguish the position states $s_y \in \{$Over, Under, Middle$\}$, which refer to the hand's position with respect to the shoulder. This gives us the tuples (s_α^l, s_y^l) and (s_α^r, s_y^r) for the left and right arm. If s_α^l and s_α^r are both not *None*, the tuples are used to classify a gesture. For example, in Fig. 4 we see for both s_α^l and s_α^r are Perpendicular and both s_y^l and s_y^r are Over, which constitute the "Up" gesture. As seen in Fig. 3 not all possible combinations of angle states
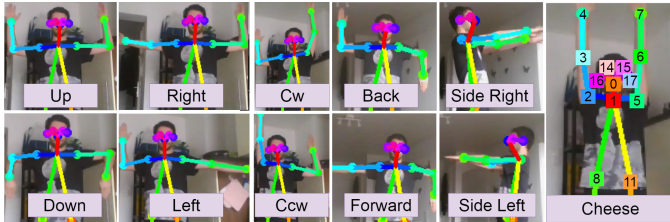

Fig. 3. Examples for all the implemented gestures in our framework. The joint labels can be seen in the "Cheese" gesture.
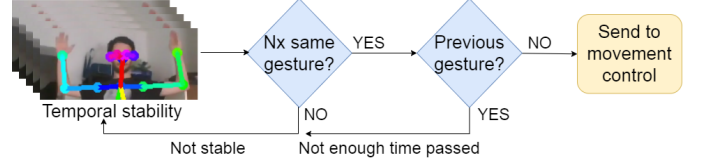

Fig. 4. Temporal stability of the recognized gestures

and position states are implemented, but the gesture set could easily be extended.

The side gestures are special gestures, which require the arm to be perpendicular to the body and the view class to be v_{Side}. We included these two gestures so that the user can issue the "Circle Around" (see Fig. 1) command, where the drone flies in front of the user, following a quarter-circle trajectory. We distinguish between "Side Left" and "Side Right" so the drone correctly flies in front of the user, instead of behind him.

A gesture is only forwarded to the movement control when it has been recognized for at least N consecutive frames. If the previously sent gesture is the same gesture, a certain time period t must first pass to avoid sending the same gesture too often as illustrated in Fig. 4. A summary of all implemented gestures can be seen in Fig. 3.

D. Distance Module

The distance module is responsible for estimating the distance to the user. We use the dimensions of the face bounding box $W, H \in \mathbb{R}$ as well as their product $W \times H \in \mathbb{R}$ as input features as their values are distance dependent. We also include the neck-to-waist distance $d(1, 11) \in \mathbb{R}$ for the same reason. Lastly, a one-hot-encoding $\hat{v}_i$ of the estimated view class $v_i \in \mathbb{R}^3$ is used, where v_{Back} and $v_{\text{Ambiguous}}$ are combined into one common class. Then $W, H, W \times H, d(1, 11)$ and $\hat{v}_i$ are concatenated into a 7-dimensional input vector for the distance estimation. The area $W \times H \in \mathbb{R}$ is a vital feature for the distance estimation, thus we represent it explicitly as input instead of learning it from H and W implicitly.

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. We apply a fully-connected neural network (FCNN) to solve this classification problem as seen in Fig. 5. 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 of the distance classes. The training and evaluation of this model are discussed in Section IV.

To achieve a continuous estimation from the FCNN, for each distance class c_i and its corresponding softmax output s_i, a weight w_i is computed by

$$w_i = \begin{cases} s_i, & \text{if } |s_{\max} - s_i| < 0.1 & \text{(4a)} \\ 0, & \text{else} & \text{(4b)} \end{cases}$$

where $s_{\max}$ is the largest softmax output. The resulting weight vector $\mathbf{w}$ is then normalized to $\mathbf{w}'$ so that the sum of the
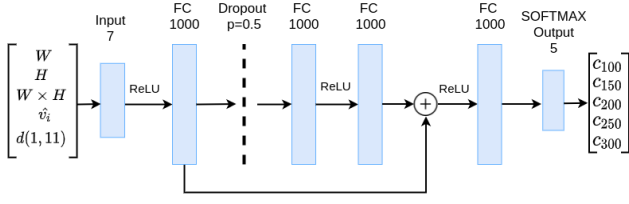
Fig. 5. Distance Estimation FCNN Architecture

weights is equal to 1. Finally, the weighted sum $\hat{d} = \sum_i c_i w_i'$ is computed, which is a convex combination of the class labels. After the post-processing, the continuous estimated distance $\hat{d}$ is forwarded to the movement control so that the drone can keep a safe distance from the user.

E. Movement Control Module

To ensure the experimental process and provide a variety of control methods, we divide the movement control module into three modes: keyboard, face tracking, and gesture control mode. The distance control is used for the face tracking mode. The keyboard mode is used to switch between modes and to ensure the safe flight of the drone. It can also execute an emergency shut down to avoid collisions. The inputs for the face tracking mode are the distance to user $\hat{d}$ and the center coordinates of the face bounding box (x, y). The face tracking mode sends movement commands with speed and direction information 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 the user. When the gesture control mode receives a gestures command, (*e.g.*, "Up"), from the gesture module, it enables the drone to execute the demanded action.

To control the drone, PID controllers [15] 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 (5) illustrates this.

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \qquad (5)$$

where $e(t)$ is the difference between $r(t)$ and $y(t)$. $r(t)$ represents the setpoint, $y(t)$ the measured process value, and $u(t)$ the control signal. Each PID controller has to be tuned independently to find the correct parameters that will lead to a smooth movement. With Trial and Error method [16], we start with setting both K_i and K_d to zero and increase K_p until the system reaches an oscillating behaviour. Afterwards, we adjust K_i 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 "Circle Around" control as seen in Fig 1(b), which consists of two actions: fly from the side to the front of the user and rotate 90° to face the user.

IV. EVALUATION

A. Implementation Details

1) Preliminaries: We used the DJI Tello Drone [14] 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 [14], 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. We use a Python implementation of DLIB [6] as CNN-based face recognition. The distance module FCNN is implemented with PyTorch and we use a Tensorflow implementation of OpenPose. We utilize an Nvidia GeForce RTX 2070 8 GB and 16 GB of RAM to run our framework with 8-9 FPS. We set $\gamma = 0.5$, $N = 3$ and $t = 625$ms for our hyperparameters.

2) Datasets and training: To perform the supervised training of the FCNN distance estimator, we create a training dataset consisting of 960 images of each distance class c_i (4800 samples in total), where a person performs various gestures to simulate a real-use scenario. We train the network for 15 epochs with softmax-cross-entropy. Additionally, 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. In all datasets, the gestures are performed with slight modifications concerning the angle and position of the arms. 600 frames are recorded for each distance class, which leads to 3000 test samples in total.

B. Experiments

1) Distance evaluation: The distance estimation is evaluated via the Mean Absolute Error (MAE) and Mean Standard Deviation (MSD) metric in centimeters. The MAE for all the distance classes is estimated at 17.75cm with an MSD of 19.6cm. The farthest distance classes contribute to a larger MAE, *e.g.*, for c_{300}, the MAE is 30.94cm as seen in Table I.

In Fig. 6 we can see that the distributions of the width-height product $W \times H$ in our training dataset have a distinct shape with slightly overlapping regions for all distance classes. The distinct shapes of the distributions are a motivation for the FCNN classifier, which is able to separate the distance classes in the latent space. We also see why the one-hot-encoding of

TABLE I
RESULTS OF THE DISTANCE ESTIMATION EVALUATION

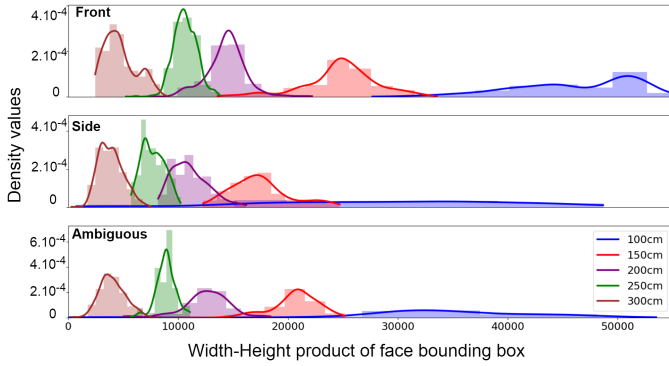| Class Metric	c_{100}	c_{150}	c_{200}	c_{250}	c_{300}	**Average**
MAE (cm)	2.47	15.15	13.27	12.99	30.94	17.75
MSD (cm)	5.22	12.73	15.98	11.82	24.01	19.6

Fig. 6. Distribution of the width-height product in the training dataset.

the view is needed as an input feature. If the view was not included, the distributions in Fig. 6 would have a significant overlap, *e.g.*, 200cm in the side and 250cm in the front view.

2) Gesture evaluation: As the gesture recognition is entirely based on a geometrical approach, it does not require any training. The only hyperparameter is N which is responsible for the temporal stability of the gestures. It can be set higher if a preciser control is required or lower for a smaller latency. We use the evaluation dataset described in Subsection IV-A. In our experiment we consider each frame as stable, *i.e., $N = 1$.* The accuracy of a gesture is equal to the proportion of its correctly recognized frames from the dataset. The proportion of the recognized gestures to all frames is seen in Table II. The average recognition accuracy of the gestures is 93.5%.

The gesture accuracy depends entirely on the joints estimations from OpenPose and the correctness of the gesture execution from the user. The gestures are designed in a natural way as each gesture represents the demanded movement, *i.e.,* pointing left for the drone to fly left. As many of the gestures have an opposite gesture, they are mapped to mirrored arm positions as in Fig. 3. (*e.g.,* "Up" and "Down" gestures). The only gesture, which is not easily recognized is the "Down" gesture as it is physically difficult to perform.

TABLE II
RESULTS FROM THE GESTURE RECOGNITION EVALUATION

Gesture			Accuracy
Type	*Action*	*Name*	
Frontal	Translational movement	Up	98.8%
		Down	66.6%
		Left	98.6%
		Right	97.1%
		Forward	97.4%
		Backward	94.1%
	Rotational movement	Clockwise (Cw)	97.4%
		Counter-clockwise (Ccw)	97.2%
	Take a photo	Cheese	96.6%
Side	Circle around	Side-left	89.2%
		Side-right	95.6%
Average accuracy			**93.5%**

V. CONCLUSIONS

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 a reliable gesture recognition. This is made possible by leveraging a skeleton-based pose estimation, which we showed provides sufficient features for both tasks. The high average gesture recognition accuracy (93.5%) enables a smooth and natural HDI and the low MAE/MSD for the distance estimation ensures the user's safety. 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 other applications.

REFERENCES

[1] Patrona, Fotini, et al. "Visual Object Detection For Autonomous UAV Cinematography." Proceedings of the Northern Lights Deep Learning Workshop. Vol. 1. 2020.

[2] Patsiouras, Emmanouil, Anastasios Tefas, and Ioannis Pitas. "Few-shot Image Recognition for UAV Sports Cinematography." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. 2020.

[3] Avila, Mauro, Markus Funk, and Niels Henze. "Dronenavigator: Using drones for navigating visually impaired persons." Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility. 2015.

[4] McFarland, Matt. "Google drones will deliver chipotle burritos at Virginia Tech." CNN Money, September (2016).

[5] Cao, Zhe, et al. "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields." IEEE transactions on pattern analysis and machine intelligence 43.1 (2019): 172-186.

[6] Geitgey, A. "The world's simplest facial recognition api for Python and the command line." URL: https://github. com/ageitgey/face recognition (visited on 08/08/2019) (2020).

[7] Besada, Juan, et al. "Drone flight planning for safe urban operations." Personal and Ubiquitous Computing (2020): 1-20.

[8] Callou, Francois, and Gilles Foinet. "Method for the intuitive piloting of a drone by means of a remote control." U.S. Patent No. 8,594,862. 26 Nov. 2013.

[9] Cauchard, Jessica R., et al. "drone. io: a gestural and visual interface for human-drone interaction." 2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI). IEEE, 2019.

[10] Sanna, Andrea, et al. "A Kinect-based natural interface for quadrotor control." Entertainment Computing 4.3 (2013): 179-186.

[11] Tezza, Dante, and Marvin Andujar. "The state-of-the-art of human–drone interaction: A survey." IEEE Access 7 (2019): 167438-167454.

[12] Sun, Ting, et al. "Gesture-based piloting of an aerial robot using monocular vision." 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2017.

[13] Nagi, Jawad, et al. "Human control of UAVs using face pose estimates and hand gestures." 2014 9th ACM/IEEE International Conference on Human-Robot Interaction (HRI). IEEE, 2014.

[14] Tello: Ryze Robotics. Available in https://www.ryzerobotics.com/tello.

[15] Knospe, Carl. "PID control." IEEE Control Systems Magazine 26.1 (2006): 30-31.

[16] Kuyvenhoven, Neil. "PID tuning methods an automatic PID tuning study with MathCad." Calvin college ENGR 315 (2002).

[17] Fernandez, Ramon A. Suarez, et al. "Natural user interfaces for human-drone multi-modal interaction." 2016 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, 2016.

[18] Cauchard, Jessica R., et al. "Drone & me: an exploration into natural human-drone interaction." Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing. 2015.

[19] Chakraborty, Biplab Ketan, et al. "Review of constraints on vision-based gesture recognition for human–computer interaction." IET Computer Vision 12.1 (2017): 3-15.

[20] Stanford Artificial Intelligence Laboratory et al. "Robotic Operating System", 2018