Gesture Control of Drone Using a Motion Controller

Ayanava Sarkar
Department of Computer Science
BITS Pilani, Dubai Campus
P.O Box 345055, Dubai, UAE
f2013206@dubai.bits-pilani.ac.in

Ganesh Ram R.K.
Department of Mechanical Engineering
BITS Pilani, Dubai Campus
P.O Box 345055, Dubai, UAE
ganeshrk1995@gmail.com

Ketul Arvindbhai Patel
Department of Mechanical Engineering
BITS Pilani, Dubai Campus
P.O Box 345055, Dubai, UAE
ketulpatel1812@gmail.com

Geet Krishna Capoor
Department of Electrical and Electronics Engg
BITS Pilani, Dubai Campus
P.O Box 345055, Dubai, UAE
capoor.geet@gmail.com

***Abstract*— In this study, we present our implementation of using a motion controller to control the motion of a drone via simple human gestures. We have used the Leap as the motion controller and the Parrot AR DRONE 2.0 for this implementation. The Parrot AR DRONE is an off the shelf quad rotor having an on board Wi-Fi system. The AR DRONE is connected to the ground station via Wi-Fi and the Leap is connected to the ground station via USB port. The LEAP Motion Controller recognizes the hand gestures and relays it on to the ground station. The ground station runs ROS (Robot Operating System) in Linux which is used as the platform for this implementation. Python is the programming language used for interaction with the AR DRONE in order to convey the simple hand gestures. In our implementation, we have written python codes to interpret the hand gestures captured by the LEAP, and transmit them in order to control the motion of the AR DRONE via these gestures.**

Keywords*— *LEAP SDK; ARM CORTEX A8; ROS; gesture recognition

I. INTRODUCTION

Drones nowadays are widely used around the world for a variety of purposes including aerial videography, photography, surveillance etc. In many cases, there is a requirement of a skilled pilot to perform these tasks using the drone which proves to be exorbitant. A simple gesture controller can make the task of piloting much easier.

The topic we are concerned with here is the gesture control of an aerial drone. Gesture refers to any bodily motion or states particularly any hand motion or face motion [1]. In our implementation, the Leap Motion Controller is used for recognition of gestures, which are motion of the hand, and as a result, we can control the motion of the drone by simple gestures from the human hand [2, 3].

There are previous implementations of gesture control of an AR Drone on platforms like cyclone.js and nodecopter using the Leap. However, in our implementation, we are using ROS (Robot Operating System) as the platform due to its ability of hardware abstraction. Florian Lier had been able to get the AR Drone respond to gestures for roll, pitch and yaw movements of the drone using Robot Operating System. Here, we are extending his implementation by not only making the drone respond to gestures for roll, pitch and yaw, but also to do flips according to the corresponding flip gesture input in the python scripts. Thus, the Leap recognizes a particular hand motion and coveys it to the ground station and according to that particular hand movement, the python code translates the necessary motion which the drone performs corresponding to such hand gesture.

Here, we have been able to successfully exploit the flip functionality of the drone, that is, the AR Drone is capable of doing acrobatic flips. Hence, according to the customized code, the LEAP identifies hand gestures for flip motion and conveys it to the AR Drone.

II. OVERVIEW OF LEAP MOTION CONTROLLER

The Leap Motion Controller as shown in Fig.1 is a gesture recognition device which uses advanced algorithms for such operations. From a hardware perspective, the Leap Motion Controller is an eight by three centimeter unit, which comprises two stereo cameras and three infrared LEDs [4]. The two stereo cameras as well as the three infrared LEDs perform the function of tracking the infrared light having a wavelength of about 850 nanometers, which is outside the visible light spectrum.

Fig 1. Leap Motion Controller

978-1-4673-8743-9/16/$31.00 ©2016 IEEE

Stereoscopy using the two stereo cameras, allows the Leap to minimize the errors and provide up to point precision while tracking hand and finger gestures.

The field of view or the FOV of the Leap Motion controller is 150 degrees vertically and 120 degrees horizontally or sideways [4, 5]. The wide angle lenses contribute to its large field of view. The frame-rate of the Leap Motion Controller is less than 200 frames per second and it has a precision of about 1/100 mm per finger [6].

The Leap Motion Controller has a range to about 3 feet directly above it as shown in Fig.1. This range is limited by LED light propagation through space, since it becomes harder to recognize the position of the moving hand in 3-diminsion beyond a certain distance. LED light intensity is ultimately limited by the maximum current which can be drawn from the USB via which it is connected to the ground station.

The Leap Motion Controller has the LEAP SDK or the LEAP Software Development Kit which is the software running on the ground station. Hence, due to this the LEAP is able to distinguish and differentiate the movement of human hands in front of it.

III. Working

The Leap Motion Controller uses its two monochromatic infrared (IR) cameras and three infrared LEDs to track any hand motion up to a distance of about 1 meter or about 3 feet directly above it. Thereby, it forms a hemispherical area above itself whose radius is about 1 meter and recognizes any hand motion occurring in that plot of volume.

Hence, because of this, the interaction space of the device is roughly about eight cubic feet which is like the shape of an inverted pyramid as shown in Fig.1, due to the intersection of the binocular cameras' fields of view.

Now, the working of the LEAP can be explained by the fact that since it tracks near infrared light, so the images appears in gray scale. Intense sources of reflectors of infrared light can make hands and fingers harder to differentiate and track.

However, there is a popular misconception that the Leap Motion Sensor like the Microsoft Kinect, generates a depth map – instead it applies advanced Computer Vision algorithms to extract raw sensor data. The LEAP SDK is the software that runs on the ground station which uses advanced algorithms to relay the hand motion perceived by its sensors to the AR DRONE 2.0.

As shown in Fig.2 (b) when the palm portion is completely parallel to the LEAP motion controller, it will be unable to detect it. This happens because the palm being parallel to the controller, it will recognize it as a single finger [7].

IV. ROS with Leap Motion Controller and AR Drone

ROS or famously known as Robot Operating System is the platform used for implementation of the leap motion of the drone [8, 9, 10]. ROS, is an open-source collection of software frameworks which is used for the development of robot softwares. The Arm Cortex A8 of the AR Drone runs BusyBox v1.14.0 Linux. ROS is run on the Arm Cortex A8 Processor of the AR Drone as well as the ground station. Thus, ROS provides functionality like that of an operating system on a heterogeneous computer cluster. ROS provides the standard platform for the platform for hardware level abstraction, package management, interaction and message passing between sensors and message-passing between processes. ROS although is not a Real-time OS, it is possible to integrate ROS with real-time code. Softwares in the ROS ecosystem can be classified as: -

- Platform and language independent tools for building ROS-based softwares.
- ROS client library applications.
- ROS packages which contain application related scripts which uses ROS client libraries.

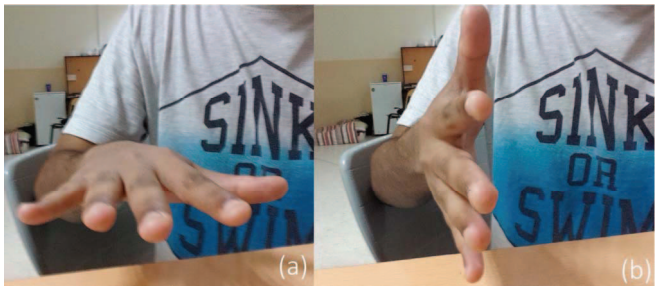

Fig 2. (a) Hand gesture recognized by LEAP (b) Gesture not recognized by LEAP

In our implementation, ROS runs on Ubuntu, a Unix-based platform. Hence, ROS forms the basic platform on which the leap motion control of the drone is implemented

Requirements of the ROS LEAP MOTION: -

- ROS HYDRO
- ROS packages like rospy and geometry_msg
- Leap Motion SDK for Linux

A. ROS Python Client-'rospy'

In ROS, 'rospy' is a package, which is a python client library specially used for ROS. Here, 'rospy' serves as a client application program interface (API), thereby, enabling us to interface Python with ROS Parameters, Topics and Services. It should be noted that 'rospy' package in ROS is designed in such a way so as to favour the implementation speed, i.e., the developer time, over the runtime performance, thus, enabling the quick prototyping of algorithms using Python and further testing them on ROS. Hence, 'rospy' forms the basis of subscribing and publishing messages between the various nodes operating between the ground station and the AR DRONE.

B. ROS Geometric Primitives-'geometry_msgs'

In ROS, 'geometry_msgs' is a sub-package of the ROS package 'common_msgs'. The package 'common_msgs' contains all the commonly used messages used for subscribing and publishing messages. In our implementation, 'geometry_msgs' is particularly used for providing messages for common generic primitives like poses, vectors and points. These primitives are primarily designed in ROS for providing a common data type and thereby, enabling the interoperability throughout the system.

Now, the ROS path must be set up suitably to accommodate the LEAP SDK. The location of the LEAP SDK is appended to the PYTHONPATH via the command: -

export PYTHONPATH=$PYTHONPATH:/path/to/SDK/lib:/path/to/SDK/lib/x64

The Parrot AR DRONE is set up to be controlled directly from the ground station using the 'tum_ardrone' packages from GitHub which is an open source project, thereby, enabling control of the drone via ground station. For running the Leap software, the command 'leapd' is entered in a terminal [11, 12, 13, 14]. Fig. 3 shows the Leap Motion Visualizer which is brought up by executing the 'leapd' command in the terminal. The Leap Motion Visualizer reflects the motion of the hand in the Field of View of the Leap.

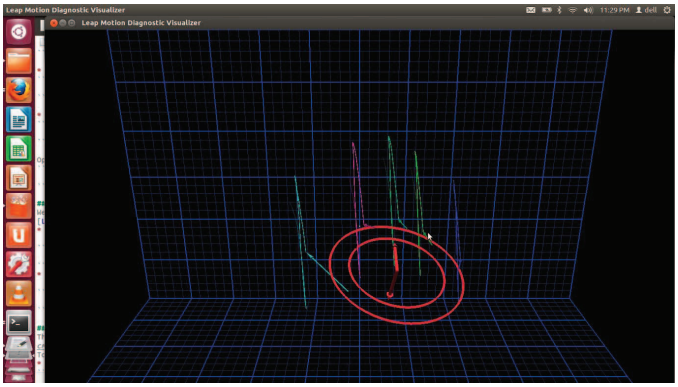

Fig 3. 'leapd' command brings up the Leap Motion Visualizer.

V. ESTABLISHING LISTENER AND TALKER METHODS

The means of communication in ROS is through the subscriber and publisher methods, which are known as the "listener" and "talker" respectively. Nodes in the ROS framework are executables or processes. The nodes running on different ROS platforms communicate through these messages. Communication in ROS can either be synchronous, like use of services or asynchronous, like use of topics in ROS.

In our implementation, there are multiple nodes running both on the ground station and the AR Drone. Hence, to run multiple nodes and make them communicate simultaneously, we have used asynchronous mode of communication. This is because, synchronous mode of communication allows only one pair of nodes to communicate at a particular time. This communication between the pair of nodes is uni-directional at a particular period of time. Synchronous mode is more like calling a function in the ROS environment.

Here, we have implemented the talker and listener methods for the purpose of the drone to respond to hand gestures. The basic design includes publishers publishing messages to ROS topics and the subscriber is subscribed to these topics, thereby, listening to the published ROS messages. The subscriber subscribes to messages through ROS topics via the established Wi-Fi connection to the Parrot AR DRONE 2.0 from the ground station. These messages get published to the AR DRONE's ARM CORTEX A8 processor, thereby enabling the quad rotor to respond to the gesture inputs made via the Leap Motion Controller. The drone responds to the hand gestures relayed through the messages by the Twist function from the 'geometry_msgs' library. The listener and talker methods are a part of the 'subscriber.py', script file.

The two main python script files aiding the gesture control include -
subscriber.py and leap_interface.py.

The file leap_interface.py is the interface that provides access to the LEAP MOTION hardware. For this, the official LEAP MOTION SDK is installed in order to load the shared provided with the SDK. It contains the pre-defined gestures in LEAP SDK and imports them from the predefined library.

VI. SIMULATION AND RESULTS

The hand gestures are captured by the Leap Motion Controller, which is then relayed onto the ground station. The Leap SDK on the ground station translates the gesture and then the talker or publisher method takes over. The publisher method publishes the translated hand gesture as a message on the AR Drone as ROS topic for communication. The subscriber method on the ROS platform of the drone receives the message or rather listens to the talker and then relays it onto the Twist method. Now, the Twist method further interprets the relayed message and then commands the drone to perform an appropriate action based on the hand gesture initially.

The 'subscriber.py' script file contains the listener and the talker methods in python for publishing the required messages to the AR DRONE. The hand gestures relayed are converted to linear and angular displacements and stored in an array. It is like an implementation of a queue, operating on the concept of "First IN First OUT". As the queue is operated, a displacement is taken and converted to ROS message to be published in order to bring about the desired movement of the drone in terms of pitch yaw and roll.

Algorithm 1 Estimating Linear and Angular Displacements based on Hand Positions and Directions

Input: Palm position, hand's normal vector and direction
Output: Angular and Linear Displacements

$init\_x \leftarrow 0$

```
        init_y ← 0
        msg.linear.x ← 0
        msg.linear.y ← 0
 1: hand_palm_pos ← li.get_hand_palmpos()
 2: hand_n ← li.get_hand_normal()
 3: hand_v ← li.get_hand_direction()
 4: if abs(hand_n)>25  AND abs(hand_v)>25 then
 5:       init_x ← hand_palm_pos[2]
 6:       init_y ← hand_palm_pos[0]
 7: endif
 8: msg.linear.x ← -(hand_palm_pos[2]-init_x)/3000
 9: msg.linear.y ← -(hand_palm_pos[0]-init_y)/3000
10: msg.linear.z ← hand_n[1]
11: msg.angular.x ← init_x
12: msg.angular.y ← init_y
```

In our implementation, the LEAP recognises "unfolding of the hand" as the takeoff command for the AR DRONE, "folding of hands" as the landing command and for doing flips the Leap recognises a "curve motion" done by the hand. All these gestures for recognition are defined in the file leap_interface.py.

Algorithm 2 Defining Hand Gestures

Input: Hand Gesture performed
Output: Action corresponding to the hand gesture

```
 1: if(unfolding of the hand) then
 2:       pub_takeoff.publish(empt)
 3: endif
 4: if(folding of hands) then
 5:       pub_land.publish(empt)
 6: endif
 7: if(curve motion)
 8:       try:
 9:            flip ←
    rospy.ServiceProxy
    ('ardrone/setflightanimation', FlightAnim)
10:            resp ← flip(19, 0)
11:       except rospy.ServiceException, e:
12:          print "Service call failed: %s"%e
13:   endif
```

In the leap_interface.py file, it imports the four gesture methods from the pre-defined leap library. This file is responsible for initialization and connection to the LEAP. It defines variables for estimating the pitch, roll and yaw values that are send to the AR DRONE according the hand motions recognised by the Leap.

The leap_interface.py file updates the position of the drone after every element of the queue is executed as pitch, roll and yaw movements for the drone.

Algorithm 3 Calculating values of yaw, pitch and roll

Input: Hand gestures
Output: Corresponding yaw,pitch and roll values

```
1: self.hand ← [0,0,0]
2: self.hand_palm_pos ← [0,0,0]
3: pos ← self.hand.palm_position
4: self.hand_palm_pos[0] ←  pos.x
5: self.hand_palm_pos[1] ← pos.y
6: self.hand_palm_pos[2] ← pos.z
7: self.hand_palm_pos[0] ← pos.x
8: self.hand_palm_pos[1] ← pos.y
9: self.hand_palm_pos[2] ← pos.z
```

The angle for pitch, yaw and roll are converted to degrees from radians and sent to the aerial drone to orient itself according to the right alignment. These angles are basically angular displacements in the x, y and z-axis. Fig. 4 represents the block diagram of the architecture.

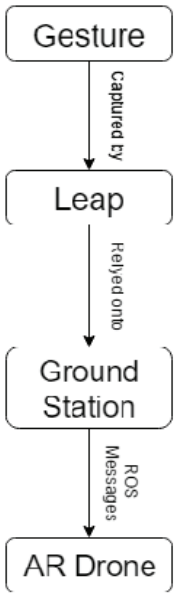

Fig 3. Block diagram of the architecture

Algorithm 4 Converting yaw, pitch and roll from degrees from radians.

Input: yaw, pitch and roll in degrees.
Output: yaw, pitch and roll in radians.

```
1: self.hand_pitch ← direction.pitch * Leap.RAD_TO_DEG
2: self.hand_yaw ← normal.yaw * Leap.RAD_TO_DEG
3: self.hand_roll ← direction.roll * Leap.RAD_TO_DEG
```

VII. Conclusion

With the help of the LEAP Motion Controller, we have been able to move the Parrot AR DRONE by using hand motion. The drone responds to any hand gesture and moves accordingly (Fig 4). We have been able to make the drone flip according to certain hand gesture which is again recognized by the 2 stereo cameras and 3 IR LEDs. Hence, it can be concluded that with the help of the Leap Motion Controller, we can use the AR DRONE to perform various tasks such as aerial videography, performing acrobatic tasks, to name a few.

The Leap can be taught to recognize more hand gestures and movements by altering the python scripts and adding more functionality to it.

Fig 4. AR Drone controlled using hand gestures using LEAP Motion

REFERENCES

[1] Alsheakhali, Mohamed, Ahmed Skaik, Mohammed Aldahdouh, and Mahmoud Alhelou. "Hand Gesture Recognition System." Information & Communication Systems 132 (2011).

[2] Bhuiyan, Moniruzzaman, and Rich Picking. "Gesture-controlled user interfaces, what have we done and what's next." Proceedings of the Fifth Collaborative Research Symposium on Security, E-Learning, Internet and Networking (SEIN 2009), Darmstadt, Germany. 2009.

[3] Singha, Joyeeta, and Karen Das. "Hand gesture recognition based on Karhunen-Loeve transform." arXiv preprint arXiv:1306.2599 (2013).

[4] Silva[1], Eduardo S., et al. "A preliminary evaluation of the leap motion sensor as controller of new digital musical instruments." (2013).

[5] Kainz, Ondrej, and František Jakab. "Approach to Hand Tracking and Gesture Recognition Based on Depth-Sensing Cameras and EMG Monitoring." Acta Informatica Pragensia 3.1 (2014): 104-112.

[6] Han, Jihyun, and Nicolas Gold. "Lessons Learned in Exploring the Leap Motion TM Sensor for Gesture-based Instrument Design." Proceedings of the International Conference on New Interfaces for Musical Expression. 2014.

[7] Potter, L. E., Araullo, J., & Carter, L. (2013, November). The leap motion controller: a view on sign language. In Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration (pp. 175-178). ACM.

[8] Leap Motion: https://www.leapmotion.com,Accessed, 9.3.2015.

[9] ROS Leap Motion: https://github.com/warp1337/rosleapmotion, Accessed, 5.5.2015

[10] leap_motion: http://wiki.ros.org/leap_motion, Accessed, 5.5.2015

[11] tum_ardrone: http://wiki.ros.org/tum_ardrone, Accessed, 5.5.2015

[12] tum_simulator: http://wiki.ros.org/tum_simulator, Accessed, 5.5.2015

[13] AR Drone Commands: https://robohub.org/up-and-flying-with-the-ar-drone-and-ros-getting-started/, Accessed, 5.5.2015

[14] Autonomy Lab SFU. AR Drone Autonomy: https://github.com/autonomylab/ardrone_autonomy,Accessed, 17.3.2015