Gesture-Based Control of the ArDrone

Victor Amaral
Department of Electrical &
Computer Engineering
Johns Hopkins University
Baltimore, MD 21218
vamaral1@jhu.edu

Abstract

I have implemented a closed loop system to control a simple quadcopter using gestures. I used the commercially available Parrot ArDrone 2 and open-source Robot Operating System (ROS) as the main software development platform. Using an Asus Xtion Pro Live depth sensor, I was able to track certain arm movements and map them to specific controls that were sent to the quadcopter over Wifi.

1 Introduction

Humans naturally communicate and express themselves with gestures. A gesture is a non-verbal communication made with any part of the body [1]. Many current technologies require hardware for interfacing such as a joystick and remote control. This requires the user to learn how to use the hardware and can often be difficult for inexperienced users or elderly people. Gestures, on the other hand, are much more intuitive.

Research has been growing in gestures recognition especially in applications for human-computer interaction. It is simple and effective because it does not require much hardware. Most of the technologies that have been developed involve mainly gesture recognition of the hands and arms. Earlier developments of gesture recognition for the hands include gloves which would contain many wires, sensors, and a microprocessor. Recent advances are using image processing and computer vision to create 3D visualizations[1].

In robotics, we have seen the migration from solely industrial settings, to robots being part of our daily lives. A couple decades ago robots were seldom seen in our household, they were mainly used in large manufacturing plants for assembly or building of parts for various applications. With robots becoming closer and closer to humans, the question rises of how we should interact with them. If they are to help us in our daily lives, an important thing to learn is to recognize gestures to control the robot for moving or even picking up an object. This could have applications for the handicapped who may need assistance around the house.

This paper presents a real-time system developed to control to Parrot ArDrone 2 using an Asus Xtion Pro Live depth sensor and the Robot Operating System (ROS) with C++.

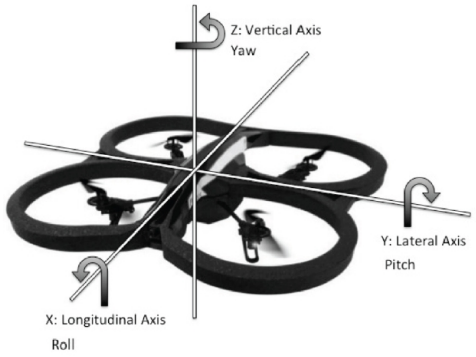

Figure 1: Parrot ArDrone http://www.yorku.ca/mack/etra2014-f1.jpg

2 Methods and Algorithm

2.1 Hardware Specification

The Parrot ArDrone 2.0 is a commercially available quadcopter costing about $300 designed to be controlled via a smart phone application. However, ROS packages have been developed to facilitate its use in research. It contains a 1GHz 32 bit ARM Cortex A8 processor and 1GB DDR2 RAM. For communication, it offers Wi-Fi g,b,n which I connected to a 2.4GHz network. For navigation, it contains a 3 axis gyroscope, a 3 axis accelerometer, and a 3 axis magnetometer. The ArDrone is controlled by 4 brushless inrunner motors with 1 8 MIPS AVR CPU per fully re-programmable motor controller. It also contains 2 cameras, an ultrasound sensor, and a USB connection for external media storage.
[1] The Asus Xtion Pro Live is an RGB and depth sensor containing two microphones available for about $180. Its resolution is 1280x1024 and allows for a depth image size of 640x480 at 30fps and 320x240 at 60fps. It can interface with a computer using a USB 2.0 port and contains the OpenNI SDK supporting Windows and Linux with C++, Java, or C# (Windows).

2.2 Installation and Architecture

The Robot Operating System (ROS) is a framework for writing robot software. It contains tools and open-source libraries to facilitate development. There are many versions of ROS available but I used Hydro to complete this project. The first step is to install ROS Hydro along with the OpenNI package available for development with 3D sensors such as the Asus Xtion Pro Live and the Microsoft XBOX Kinect. Additionally, I used the ardrone_autonomy package developed by the Autonomy Lab in Simon Fraser University, for controlling the ArDrone.

ROS allows communication of information between processes using nodes, messages, topics, and services. Nodes are processes that perform computation. They communicate through messages which are data structures with typed fields. A node sends out a message by publishing it to a given topic. Another node that needs to use that data will subscribe to that topic. There can be many nodes publishing and subscribing to a given topic. Since topics are one way communication, it's not suitable for request/reply applications. For this we use services. A client uses a service by sending a message and awaiting a reply. This process is illustrated in Figure 1.

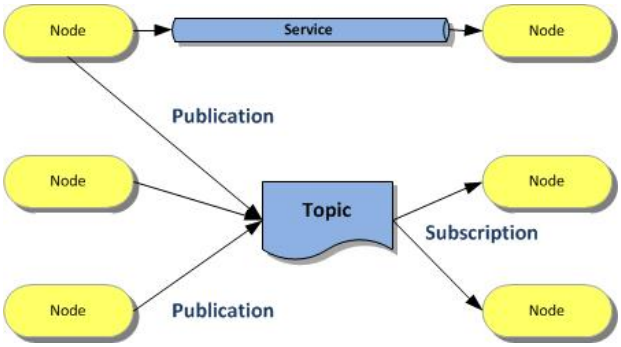

Figure 2: Illustration of the ROS communication framework http://www.generationrobots.com/en/content/55-ros-robot-operating-system

Figure 3: This demonstrates the frames of each body part being tracked by running openni_launch, openni_tracker, and rviz

[1] Specification details for hardware found at http://ardrone2.parrot.com and http://www.asus.com/Multimedia/xtion_pro_live/specifications

2.3 Algorithm

In my implementation I wrote a node that subscribes to the tf topic and publishes messages to the ArDrone topic. The tf topic keeps track of multiple coordinate frames and allows for transformation of points or vectors between any two frames. The openni_tracker package creates the skeletal tracking using a frame for each of the different parts of the body and broadcasts to the tf topic.

First I initialize ROS and some variables. I publish an empty message to the ardrone/takeoff topic which causes the ArDrone to take off and I let it hover for 10 seconds so that the user has time to make sure that the skeleton tracking is working. Using rviz, the user can visualize the openni depth frame which shows the frames of each body joint, as shown in figure 3. After 10 seconds, the user is then in control of the ArDrone for 60 seconds (this time can be adjusted). I calculate the transformation from my neck frame to my right hand frame and operate on the x axis to control left/right movement of the ArDrone (roll). I then calculate the transformation from my neck frame to my right hand frame and operate on the y axis to control the forward/backward movements of the ArDrone (pitch).

$$A_W^H = A_N^H A_W^N \tag{1}$$

$$A = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

As shown by the transformation matrix A, the transformation from the hand to the world frame - the frame the robot is operating on - is the transformation of the hand to the neck frame multiplied by the transformation of the neck to the world frame. Each transformation is represented by a matrix A having some rotation matrix R and a translation vector t.

The ArDrone is controlled by publishing a twist message - expresses velocity in 3-space broken into linear and angular parts - to its velocity topic. For each transform, I get its translation vector and use that to set the velocity of the ArDrone. After 60 seconds, if the quadcopter does not crash, it will land.

Currently I only implemented control of the ArDrone on the x and y axis, thus it is not possible to make it rotate about the z axis or increase or decrease in altitude (yaw) because I found that it was too difficult using only two hands. Additionally, I'm only sending linear velocity commands for similar reasons. The control was much more stable if each hand is assigned to operate on only one axis. I was constrained by the space I had to do the testing which was very small so the ArDrone would crash often. It would also create small wind tunnels in areas making it difficult to navigate the ArDrone near corners. Everytime the ArDrone crashed, it would need to be reset by removing the battery from it and plugging it back in. To make resetting quicker, I wrote a node to reset the ArDrone after a crash so that the user doesn't have to manually reset it.

3 Experimental Results

When the quadcopter takes off, the user has 10 seconds to calibrate the skeletal tracking in front of the depth sensor which was plenty of time for me. Nevertheless, this parameter is also adjustable. The default starting configuration is with the right hand in front of the neck with the palm towards the camera and the left hand extended horizontally as shown in Figure. The left hand controls the pitch. Moving the left hand up will cause the quadcopter to go backwards, as shown in Figure 5, and moving the left hand down will cause the ArDrone to fly forwards. To stabilize it to a hover, it should be extended horizontally. The right hand controls the roll. From the starting position, moving it across the chest towards the left shoulder will cause the ArDrone to fly to the left, as shown in Figure 4, and moving it to the right will cause the ArDrone to fly to the right. The plots shown are representing time in seconds on the x-axis and velocity in meters per second on the y-axis. Whenever roll or pitch is controlled, the other is kept relatively constant. Positive values corresponds to movements in the "right" or "up" directions and negative values corresponds to movements in the "left" of "down" directions.

The control is real-time and is communicating over a 2.4GHz Wifi connection. Depending on the Wifi connection and how many users are connected to it, there may be a slight delay.

4 Conclusion and Future Work

I have not utilized all the degrees of freedom of the ArDrone. Future work can be done to implement angular velocity as well as linear velocity along the

z axis (yaw). In addition, the system can be made more user friendly by having a gesture to takeoff and another gesture to land the ArDrone instead of having everything set by a certain amount of time. This was difficult for me to think of how to do because I assigned the gestures of each of my arms to controlling the ArDrone but perhaps a head tilt could control takeoff/landing. Furthermore, a GUI to start all the processes integrated with rviz would avoid having many terminal windows open and would facilitate visualization. As mentioned in the introduction, a system like this could be used for human assistance so adding a manipulator to the quadcopter, giving it the ability to grasp objects, would also be useful.

5 Acknowledgments

I would like to thank Dr. Rene Vidal for mentoring me throughout this project. Additional acknowledgments go to Dr. Marin Kobilarov for allowing me to use his laboratory and equipment as well as his PhD student Subhransu Mishra for helping me troubleshoot my code.

References

[1] Moniruzzaman Bhuiyan, Rich Picking 2009. *Gesture-controlled user interfaces, what have we done and whats next?*

[2]Van den Bergh, M. and Carton, D. and de Nijs, R. and Mitsou, N. and Landsiedel, C. and Kuehnlenz, K. and Wollherr, D. and Van Gool, L. and Buss, M. July, 2011. *Real-time 3D hand gesture interaction with a robot for understanding directions from humans* RO-MAN, 2011 IEEE pg 357-362

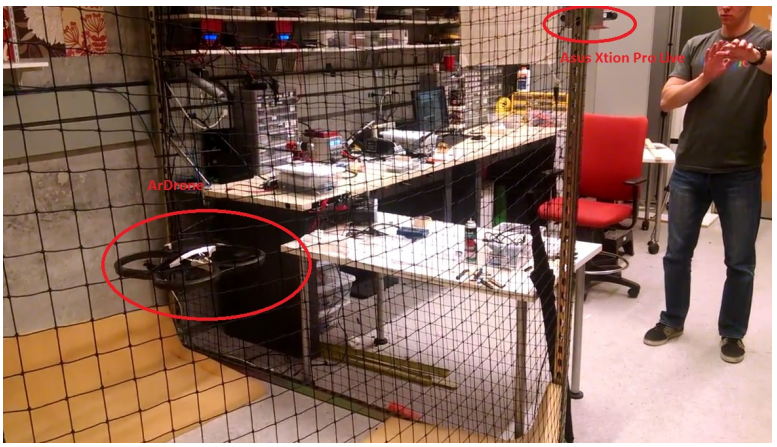

Figure 4: Starting position as the ArDrone hovers for 10 seconds to wait for the user

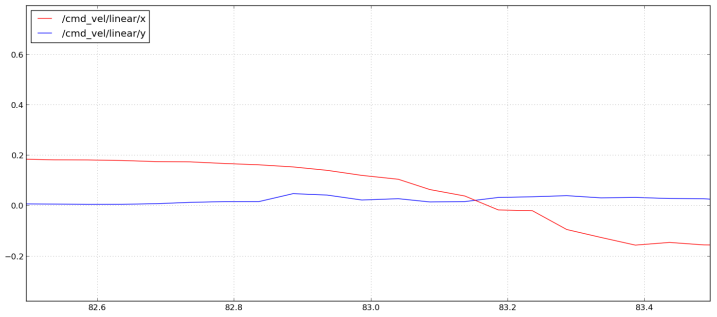

Figure 5: Linear velocity of the quadcopter as the roll is controlled

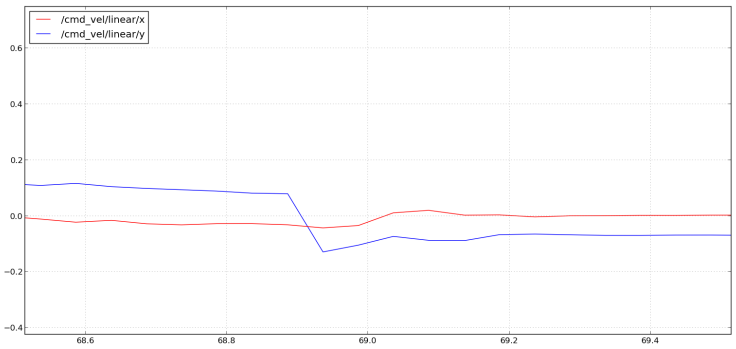

Figure 6: Linear velocity of the quadcopter as the pitch is controlled