Gesture Based Drone Control Abstract

The objective of this project was to develop and create a control system for a 3DR Solo drone that utilizes the user's hand movements to influence the flight of the drone. A key aspect of the project was flexibility - the control system had to be easy to modify and readily accessible. The project was accomplished through the use of a Leap Motion device, a 3DR Solo drone, and a laptop computer. The control system was developed such that the user first connects to the drone via wireless (WiFi) and then runs the designated program (GestureControl_finalRelease.py). Execution of the control strategy was accomplished through a constant loop that first collects 15 frames of hand data from the Leap Motion which is processed through a conversion and is sent to the vehicle over 0.25 seconds. The program waits 1 second to avoid command overlap, then re-loops. Multiple users without any/limited experience were able to use the system without difficulty during testing. All of the code is publicly available with several adjustable parameters, such as initial takeoff height & simulation capability, thus also satisfying the secondary objectives of flexibility.

Gesture Based Drone Control Research Paper

Introduction

On October 7th, 2001, the CIA deployed Predator Drone 3031 in US government's first use of a drone during a military operation[1]. Almost 15 years later, both military and civilian drone use has skyrocketed. It has become quite common to see these vehicles at sporting events, news gatherings, or even in your own neighborhood. In the year of 2016 alone, the aerospace research company Teal Group predicts that two million consumer drones will be sold[2]. This increase in usage is expected to continue with the sales of military and civilian drones to total over $89 billion by 2023[3]. Regarding civilian drones, they are primarily used for photography and in some places, environmental research[4]. These drone are either flown using a handheld controller or simply programmed to fly in set course (often a in a loop pattern).

FIGURE 1: Military Predator Drone in Flight[8]

The first publicized effort to create a gesture based control system for a drone was in 2013. A pair of German software developers used a Creative Labs 3-D camera and Intel Perceptual Computing SDK to map a virtual airplane yoke. This project, called ‘The Parroteer Virtual Flight Controller” was able to effectively control a Parrot AR 2.0 drone[5].

However, this project had two main drawbacks. The first was that the code was written in Java, instead of the usual C or C++. Because of this, the user is not able to easily adjust the code to their own individual needs. The other drawback was that the control scheme was not ideal for the ordinary user. Most of the civilians that choose to purchase a drone, do so for their availability and ease of control. As such, many of them are unfamiliar with conventional flight systems, hindering their ability to use a virtual airplane yoke. The other publicized effort in this area was a Taiwanese group called PVD+ which completed a wrist based system, called Dong Software, in 2014. They chose to have the wrist system and drone to be powered by Arduino boards which communicate through auto-paired HC-06 and HC-05 Bluetooth breakout boards to send sensor readings[6]. However, that project also suffers from the lack of adaptability. Everything is coded in Arduino, allowing for a user with training to make modifications, but they would require some rudimentary knowledge of the code. The other drawback is the hardware. The Dong Software requires additional components beyond a Parrot AR Drone 3.0, which the user will have to either buy from PVD+ or build themselves[7]. As such, this severely limits the accessibility of the Dong Software system.

FIGURE 2: Civilian Parrot AR Drone 2.0[9]

The purpose of this project was to create a control system for a 3DR solo drone that utilizes the user's hand movements to influence the flight of the drone, all while making the project easy to modify and readily accessible. This was accomplished over nine weeks, through the usage of a Leap Motion device, which was linked to a computer via a WiFi connection, in order to broadcast to the drone. Beyond the gimmicky nature that appeals to the consumer market, this gesture control approach has benefits to users who cannot use conventional control systems. Examples of this include users with arthritis or simply just younger users that are not accustomed to the conventional drone controller. Beyond this, certain parameters within the program can be changed, allowing it to be easily configured and adapted to the end user's needs. Hence the elements of this project overcame the limitations of the previous two attempts at gesture based control cited above.

Materials & Method

The following materials were used in the project:

- 1 3DR Solo drone & controller (for safety)
- 1 Computer capable of wireless connection & running Python 2.7.x
- 1 Leap Motion device (with connector cable)

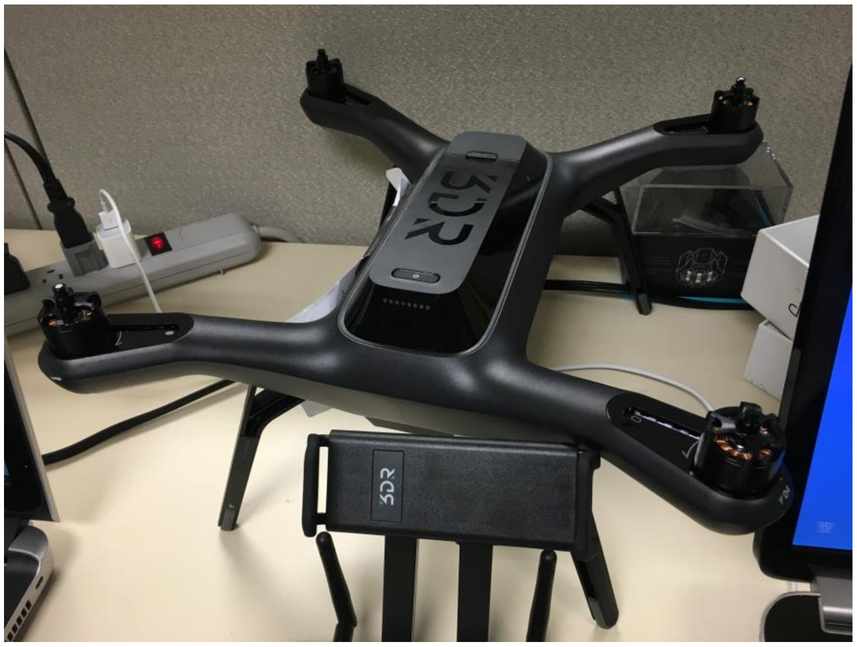

FIGURE 3: 3DR Solo Drone (propellers were not attached during indoor testing for safety reasons) with Accompanying Controller

FIGURE 4: Leap Motion Device[10]

For an average user with a 3DR Solo drone, the cost of this project would be between $50 and $80 since the only additional item required is the Leap Motion device. All of the programs, as well as an explanation of each component, are available in a public Github repository (cannot be released in this paper due to it containing identifying information such as the developer's user ID). Special thanks to the Wireless Networking Information Laboratory for purchasing both the Leap Motion device and the 3DR Solo drone.

The entire project was carried out independently in sections, then pieced together to create the final product. One student worked on constructing the drone program that would carry out the flight commands, another student worked on retrieving data from the Leap Motion device, while third student worked on linking the two pieces together and then feeding the information over wireless to the drone.

Leap Motion Data Retrieval

The Leap Motion retrieves data using infrared cameras that detect hand position within an eight cubic foot area. If a part of a hand is not detected, the Leap Motion will use the data it is given to simulate the location of the hand and will superimpose it onto the data it has collected. It collects the data as a series of points that are measured on an x, y, and z coordinate field. These points are recorded per frame and are replaced by new results every 0.25 seconds unless the program calls for the previous frame using the command *previousFrame*.

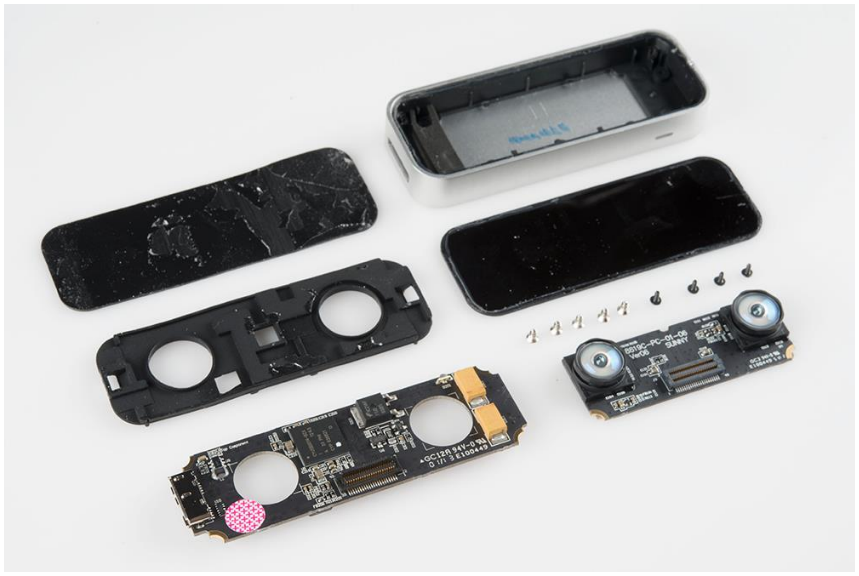

FIGURE 5: Completely Disassembled Leap Motion Sensor[11]

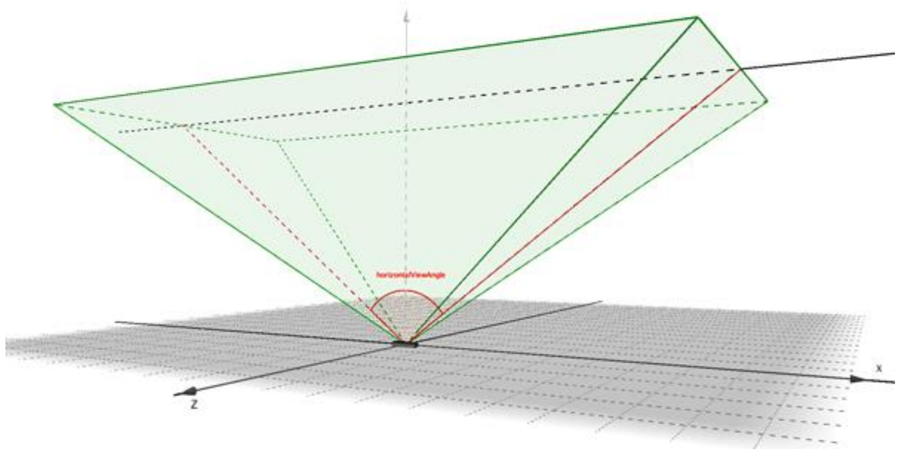

FIGURE 6: Leap Motion Sensor Range of Detection[12]

Data Smoothing

Since the Leap Motion device records data at 60 frames per second over the course of 0.25 seconds, totalling 15 frames, there are 45 data points (15 for each direction). In each frame, the *LocationGlobalRelativeLeap* data for the X, Y and Z is appended to a respective array (*xData*, *yData*, *zData*). Of note, the Leap Motion device's coordinate system is configured for a screen, rather than three dimensional motion. As such, the Leap y-coordinate data is assigned to the *zData* array and the Leap z-coordinate is assigned to the *yData* array. For each array of location data, the median is determined. This median is then compared to the median of the other two arrays. The greatest median is then assigned to a directional variable (*x, y, z*), while the other two directional variables are assigned a value of zero. Essentially, this process keeps the greatest directional value and sets the rest to zero. This approach is taken so that small hand movements are ignored and only the overall gross motion of the hand is kept.

Drone Flight Commands

Overview

All of the flight commands are conveyed through the “dronekit” library which is publically available and imported in line 1 of the program. Upon starting the program, the device connects to the drone using the line “*vehicle = connect(target, wait_ready=True)*”, where target is the connection string of the drone. The program will not run if it cannot find the vehicle specified. The function *send_ned_velocity()* accepts the parameters of velocity in the X, Y, and Z direction, in addition to the duration of movement. These parameters are compiled into a message, which are then sent to the drone every 0.05 seconds over the default duration of 0.25 seconds. At this point, the function exits and the program again requests the Leap Motion data after one second.

Updating the Directions

Since the Leap Motion device records hand data in millimeters, a small motion produces a value typically greater than 20. If directly sent to the drone, it would attempt to fly at a velocity of 20 meters per second, which could result in injury to the user or damage to the drone. As such, the data stored in the variables *x, y, z* need to be assigned a proper value. The function *updateDirections()* takes the sign of the previously recorded hand data in x/y/z and pastes the sign of it onto the value 0.3 (corresponding to 0.3 meters per second) effectively scaling or normalizing the velocity. This value is then assigned to *xDirection*, *yDirection*, or *zDirection* respectively. Of note, the 3DR Solo drone interprets movement upwards along the z-axis as negative, while the Leap Motion defines motion downwards in the z-axis as negative; this is accounted for when updating the *zDirection*.

Sending the Flight Commands

The function *send_ned_velocity()* is used to transmit the flight commands from the device to the drone. This function accepts the parameters of *velocity_x*, *velocity_y* and *velocity_z*, in that order. These parameters are then compiled into a message, which is then sent to the vehicle every 0.05 seconds for a duration of 0.25 seconds. However, this is typically accomplished with a *for* loop, which cannot be run with a float value. To get around this constraint a pseudo-timer was constructed. The variable *dur* is created and initialized to a value of 0. After each time the message is sent, the *time.sleep(0.05)* command is used to delay the next time by 0.05 seconds. Then, *dur* is incremented by 0.05 and the loop exits once *dur* is greater than or equal to 0.25 seconds. After the flight commands have been sent for that loop's Leap data, the program waits one second before beginning another loop to avoid overlapping commands.

Linking Components

Each component was developed separately, then compiled into one program towards the end of the project resulting in each section of the program having an assigned function. These functions are called within the main section of the program and feed each other the necessary information via variables/parameters.

On the hardware side, the Leap Motion device is connected to the computer via the provided USB cable (comes with the Leap). The computer connects to the drone controller via its own generated WiFi network. This same network allows the controller to communicate with the drone.

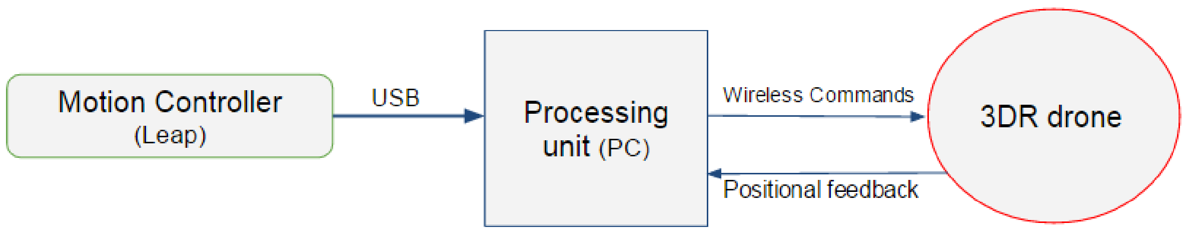

FIGURE 7: Simplified Diagram of Physical Components and Their Interactions

Configuration Notes and Additional Features

Within the program, there are numerous items that can be configured by users to suit their individual needs. The first of these items is the connection string of the vehicle. Each drone has its own unique connection string, often varying by type of connection and company/model. This information can be changed on line 27, by modifying the variable *target* to the connection string in question. Also configurable is the duration that the Leap Motion device collects data. This is set to a default of 0.25 seconds on lines 113-114 via *time.sleep(0.25); controller.remove_listener(listener)*. Of note, the variable *dur* (line 126) must be changed to match this which prevents a mismatch between the Leap Motion data and the flight data commands sent to the drone. It is possible to not have them match, though that would require additional configuration by users and should not be attempted unless they are certain it will not interfere with other components. The initial takeoff height can also be easily changed via the parameter of the function *arm_and_takeoff()* on line 249. The default height is currently seven meters and should not be changed to less than four meters. Also available to be changed is the key(s) to exit the program on line 264. The default key is “q”, but can easily be changed in line of code reading *if (msvcrt.getch() == "q"):* .

This project also offers several additional features within the program, the first of which is the option of running the entire program on a simulation. To do this, all the user has to do is uncomment (remove the “”” from the start and end) lines 12-26 and comment out (add “”” to the start and end) lines 27 and 30. Doing this will cause the program to run on the SITL (Software In The Loop) simulator. Alternatively, the user could simply opt to run the program "3DR_handControlSIMULATOR.py" under droningOn>DroneScripts. The second additional feature is the option of performing a flight check which is accomplished through the function *flightCheck()*. This function brings up information about the vehicle such as connection string, GPS location, battery percent, status, if it is armable, and much more. In order to run this, the user simply has to type “flightCheck()” onto line 248. The third and final additional feature is the option of having the vehicle takeoff and fly to a specified altitude, then come back and land. This is accomplished through the function *upAndDown()* and just like the flight check function, it can be added via the text “upAndDown()” on line 248.

Testing

To run the program (GestureControl_finalRelease.py):

1. Follow the necessary steps to connect the vehicle. If the connection string for the vehicle is not udpin:0.0.0.0:14550, be sure to change it on line 27 of the program.

   3DR Drone specific steps

   a. Turn both the controller and drone on.
   b. Wait for the controller to acquire a GPS fix.
   c. Connect to the controller on a computer just as one would for any other wifi network. The default password is "sololink".

   d. To connect both the vehicle and computer to the internet, type: "solo wifi --name=<n> --password=<p>". Then reconnect to the controller once it has finished.

   e. Run the command "solo info" to insure that the connection has been established. If there is no connection, repeat steps a-c (d if applicable).

   Further help can be found at: https://dev.3dr.com/starting-utils.html

2. Open command prompt or equivalent application on the computer.
3. Navigate to the directory containing "GestureControl_finalRelease.py" .
4. Modify any items in the program (as needed) using any code editor. Be sure to save it when any necessary changes have been made.
5. Run the program by typing "python GestureControl_finalRelease.py"
6. After the vehicle reaches the specified target altitude, the program will output the text "Reached target altitude". At this point, a hand must be placed over the Leap Motion device within 2 seconds.
7. The program will display "Hand control enabled" once a hand is detected and the user can now begin controlling the drone.

   IMPORTANT: The Leap Motion by default samples a space of 0.25 seconds at intervals 1 second. Any motion not within the sample space will not be taken into account. To modify this, see "Configuration Notes".

To exit the program:

1. Press "q" or the esc key on the keyboard.
2. The program will display "Activating 'Landing' Mode... " then "Landing..." .
3. The vehicle will return to its initial location, land and then turn off.

All instructions and options can be found in the “readMe.txt” file, in addition to links to help with troubleshooting.

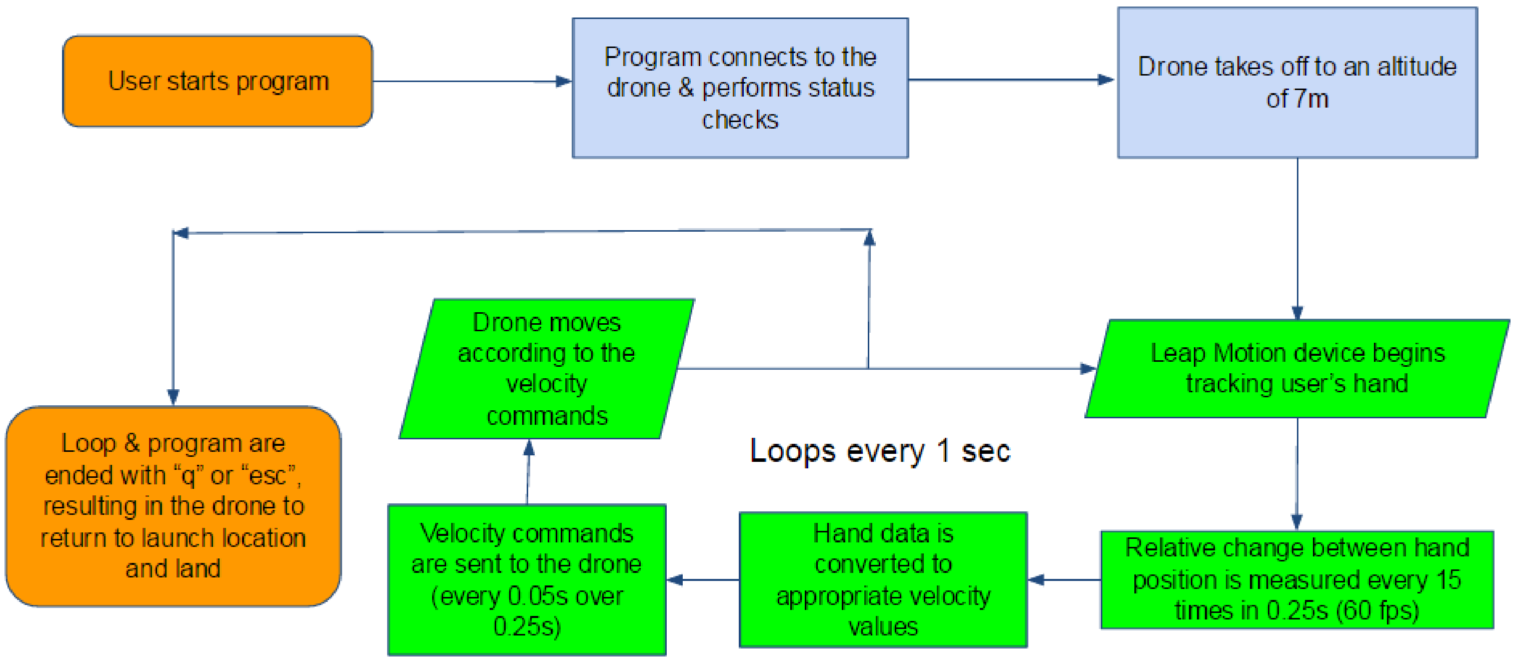
User starts program
Program connects to the drone & performs status checks
Drone takes off to an altitude of 7m
Leap Motion device begins tracking user’s hand
Relative change between hand position is measured every 15 times in 0.25s (60 fps)
Hand data is converted to appropriate velocity values
Velocity commands are sent to the drone (every 0.05s over 0.25s)
Drone moves according to the velocity commands
Loops every 1 sec
Loop & program are ended with “q” or “esc”, resulting in the drone to return to launch location and land

FIGURE 8: High-level Diagram of Program Flow

Results & Discussion

The program was initially tested on the SITL simulator, in order to prevent damage to the drone in case of any major bugs. No major flaws were found and a few of the smaller bugs (ex. inadvertent incorrect variable assignment) were repaired. Three test flights were done on the physical drone with no issues. The program performed as expected. Hand motion was correctly corresponded to flight movement with the intended delay. The only things of note were the drone’s restabilization protocols due to wind. When air currents disrupt the flight pattern of the drone, it automatically attempts to fly back to the original location before the disruption. As such, it is typically best to wait for the disturbance to subside before issuing more commands.

Conclusions and Future Work

The designed control system performed admirably (responded to hand motion with the correct trajectory) and shows promise beyond research testing. As stated in the introduction of this paper, this gesture control approach has benefits to users who cannot use conventional control systems examples of which include users with arthritis or simply just younger users that are not accustomed to the conventional controller. Multiple users without any, or very limited experience, were able to use the system without difficulty when testing. As such, the primary objective of creating a user friendly gesture control system was met. The additional objectives of making the system easy to modify and readily accessible were also accomplished. All of the code, as well as the entire process, is publically available on GitHub (though the repository cannot be listed in this paper due to it containing identifying information), where anyone can download and use it. The program supports numerous configuration options for the user with both labels and explanations.

With regards to future work, the main focus would be on improving the data collection aspect of the system. In its current state, the Leap Motion device records 15 frames of hand data in 0.25 seconds. However, it takes 0.25 seconds to send the equivalent commands to the drone. On top of this, there is a one second delay between data samples to prevent the overlap of flight commands. In future releases of the system, this delay would be minimized even more so, thus allowing for a smoother & more intuitive user end control.

References

1 Woods, C. (2015, May 30). The Story of America's Very First Drone Strike. Retrieved September 10, 2016, from http://www.theatlantic.com/international/archive/2015/05/america-first-drone-strike-afghanistan/394463/

2 Wingfield, N. (2016, August 29). A Field Guide to Civilian Drones. Retrieved September 10, 2016, from http://www.nytimes.com/interactive/2015/technology/guide-to-civilian-drones.html

3 Paganini, P. (2015, March 31). Why Civilian Drone Use Is A Risky Business. Retrieved September 10, 2016, from http://www.foxnews.com/tech/2015/03/31/why-civilian-drone-use-is-risky-business.html

4 Hollenhorst, J. (2012, May 3). Eyes in the sky: Growing prevalence of drones a plus for some, privacy concern for others | KSL.com. Retrieved September 20, 2016, from http://www.ksl.com/?sid=20257618

5 Drones Fly 'Hands Free' with Gestural Technology. (2013, September 17). Retrieved September 10, 2016, from http://www.intelfreepress.com/news/drones-fly-hands-free-with-gestural-technology/6877/

6 Using The Force? No, it's an Apple Watch flying this drone. (2015, December 31). Retrieved September 10, 2016, from http://www.reuters.com/article/us-apple-watch-drone-idUSKBN0UE14Q20160101

7 Jimeno, J. M. (2014, October 16). Wearable: Gesture Controlled Drone Component List. Retrieved September 10, 2016, from https://hackaday.io/project/3180-wearable-gesture-controlled-drone

8 Weinberger, S. (2014, May 17). The ultra-lethal drones of the future [Predator Drone]. Retrieved September 19, 2016, from http://nypost.com/2014/05/17/evolution-of-the-drone/

9 Parrot AR.DRONE 2.0 Elite Editio [Picture of the Parrot AR Drone 2.0]. (n.d.). Retrieved September 19, 2016, from https://www.parrot.com/fr/#disco

10 Factory Leap Motion [Image of the Leap Motion device]. (n.d.). Retrieved September 15, 2016, from http://www.3ders.org/images/hot-pop-factory-leap-motion-1.png

11 [Disassembled Leap Motion device]. (n.d.). Retrieved September 18, 2016, from https://cdn.sparkfun.com/assets/f/7/6/3/6/51c471f3ce395f7f72000000.jpg

12 Leap_horizontalViewAngle [View of Leap Motion interaction space]. (n.d.). Retrieved September 18, 2016, from https://cms-assets.tutsplus.com/uploads/users/289/posts/20455/image/Leap_horizontalViewAngle.png