Personal Interactive Autonomous Drone

Proposal of a Supervised Implementation

Bastien Chatelain
EPFL
bastien.chatelain@epfl.ch

Immanuel Koh*
EPFL
immanuel.koh@epfl.ch

Jeffrey Huang†
EPFL
jeffrey.huang@epfl.ch

January 2018

Abstract

Today, using drones as a work tool is common business. Each generation of drones becomes cheaper, more independent, less supervised, allowing nearly anyone to use them. In this project we implement a supervised autopilot for a Parrot Bebop 2 drone in an Android application[1]. With Parrot SDK and the drone's real time single perspective view, computer vision algorithms control the drone's behavior. From color blob detection and tracking, we teach the drone to recognize predefined subjects and to react accordingly using predefined procedures. With a small interaction experiment, we discuss some limitations of our implementation and some possible future improvements. With this first step of interaction, we want to seek possibilities for drones to replace personal assistants or to become an autonomous support for everyday life.

Keywords. Android, Parrot Bebop 2, Autonomous Flight, Blob Detection, Face Detection, Face Recognition, Drone Interaction

*Supervisor
†Professor
[1] https://github.com/blchatel/DroneProject

Contents

1 Introduction

Drones, a pilotless radio-controlled aircraft, were first developped during World War I. For a long period, mainly due to their costs, drones were only available to military missions as field reconnaissance or bombing. Today, even if military usage is still in expansion, the market offers a huge variety of smaller devices aimed at the general public. As these drones cover a wide range of application such as work, business, fashion and recreational, they are more and more present in our everyday life.

Today, modern drones are already partially or fully autonomous. Some specific devices excluded, all flying objects have some sort of flying assistance. The user interface is designed to allow people such as photographers, architects, lumbermen or firemen and not only drone's pilots to efficiently use them as a work tool. Even if the operator is supposed to give periodical inputs, most drones can, on their own, take-off, fly, hover and go back to their initial position. As drones should be easy to fly, direct inputs given by the user are often remapped and supervised by softwares before being send to the drone. For example, the gaz axis which is between -1 and 1 and mapped to *climb v.s. descend* concepts, is not directly linked to the gaz control which is between 0 and 100 as we could imagine. The climb and descend concepts can be configured and restricted (i.e. max altitude, max vertical speed etc.) making the flight experience safer and easier for everybody. Also, when no inputs are given, the drone stabilizes immediately in a stationary flight.

Drones are used for fully supervised tasks such as filming, surveying, mapping and creating 3D renderings but also for less supervised tasks such as deliveries of goods [1]. A drone can be much more efficient to transport light and small objects like organs from one hospital to another[2] or for accessing rugged and isolated areas. Drones are also used by the police, the firefighters and the state's services during emergencies to assess the situation at a much lower cost than traditional surveys with a manned helicopter.

Professional devices made by DJI[3] or Parrot[4] become accessible for domestic usage. Latest models are better equipped and more robust, they often include a high definition video camera and sometimes other sensors, making drones attractive and ready to use. Video cameras are an open door for all computer vision usage with a single image perspective [2]. Drone could now be capable of tracking [3][4] or recognizing objects[5] and people[6].

This project aims to deal with the unresolved problem of computer vision and establish a first step in Drone-Human interactions. We want to open ways for drones to replace personal assistants or to become an autonomous support for everyday life. By implementing our supervised autopilot, we are looking to get the drone closer to its own autonomous interaction to pave the way towards fully autonomous drones.

2 Project Selection

2.1 Initial Project

We use drones to achieve tasks that fixed cameras and satellites can not manage. Those tasks must need multiple points of view (from different angles and altitudes) and benefit from the high range of speed and altitude a drone can reach. Our first idea was a "counting a crowd of people" task divided into two sub-parts. Firstly, assuming people were static compare to the drone, the idea was to count people in a single path at some key points in time and space. Secondly, we would adapt the algorithms to count people in a bounded two dimensional area. The final goal would be to count people or at least compute an estimation of the amount of people attending any given events and how they behave in terms of time and position.

A project of this kind implies autonomous flight [7], blob, face detection and recognition. The use of drones for research and development is subject to a number of restrictions.

[2] jonbarron.org/healthcare-hospitals/do-gooder-drones
[3] www.dji.com
[4] www.parrot.com

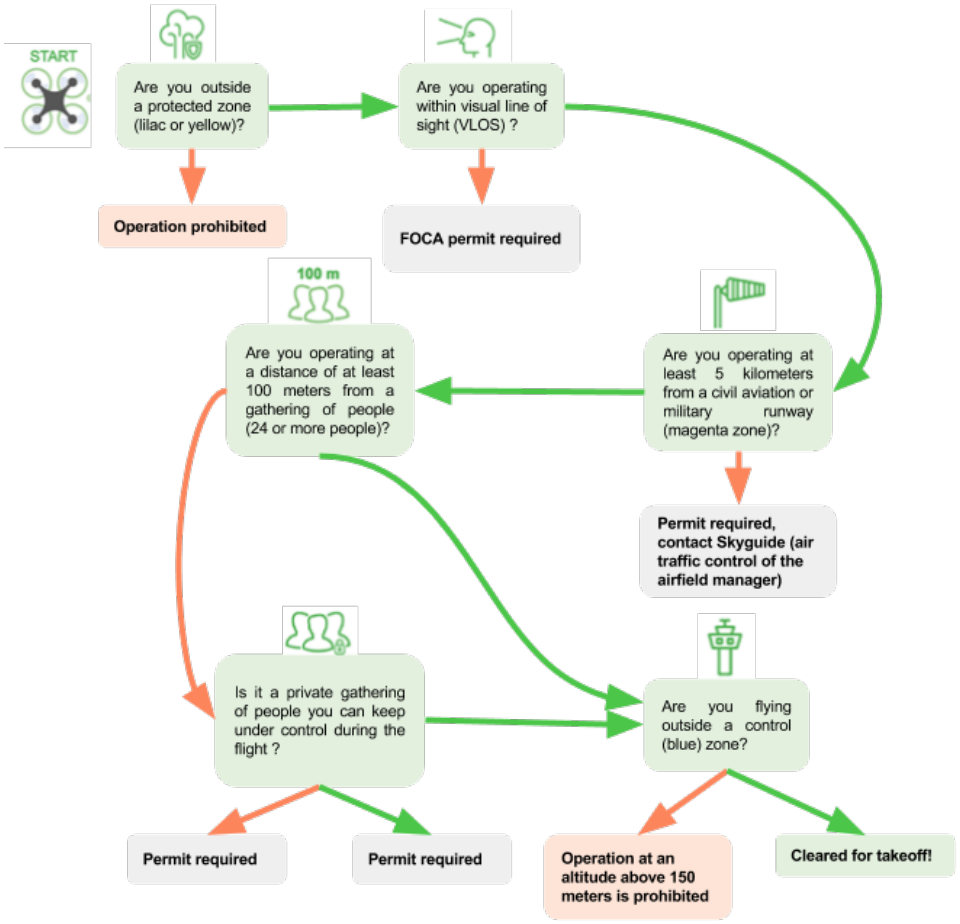


Figure 1: The Drone Guide: It clearly shows drone's pilots whether they are allowed to fly in their desired zone without restriction. In addition, it also indicates which permits may have to be obtained. *www.bazl.admin.ch/bazl*

2.2 Restrictions

Legal

The proposed project must deal with the EPFL's rules[5] and the Swiss regulation laws[8]. The pilot must permanently maintain a direct visual contact with the drone which limits the range of application for autonomous flight. At any time during the flight, the operator must be able to safely take back the complete control of its aircraft. All flying object must be at least 100 meters away from any group of people. A group's definition differs depending on the location, but in our case, the amount of people was 24. To visualize this restriction, we can imagine the drone as the center of a 100 meters radius sphere which should never contains more than 23 people. See Figure 1 for the specific decision tree. If we assume the privacy rules implied by picturing, filming or making noise are not the most annoying ones, we have to consider the last set of rules which are more constraining. Drones above a given weight of 500 grammes need special insurance and flight authorizations, especially when the planned flights are located in a 5km radius of any air field. Moreover, EPFL asks for a delay of five days before any flights on its campus.

Technical

On top of these legal restrictions, we also face some technical restrictions. As we have to stay 100 meters away from groups, we need a very good camera to distinguish faces or even blobs. At night of with minimal natural light, we would need a thermal sensor to record something useful and it becomes difficult to keep a drone in sight. We also want a drone which handle rain, wind and impacts well. Finally, to respect both the rules and the semester's project budget, our drone has to weight less than 500g and cost less than 2000.- CHF.

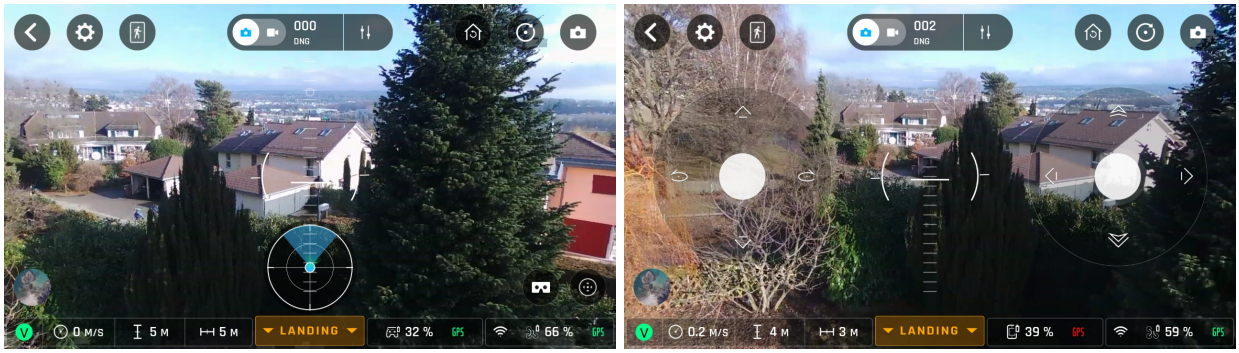
[5] securite.epfl.ch/drones-en

Figure 2: Equipment used for this project, a Parrot Bebop 2 drone, the SLAM dunk sensor we never got and a SkyController 2 connected by USB to a smartphone.

2.3 Material

Considering our goals, the regulations and the restrictions, we chose the Parrot Bebop 2 drone and its SkyController2. This drone weighs 500g and is quite efficient for its purpose. We decided to buy the Power version which comes with a second battery, doubling the average autonomy of 25min. From the drone's specifications[6], we can notice the 14 mega-pixels CMOS wide angle camera which records video in full HD (1080p). This drone is a very robust quadcopter controlled via Wi-Fi by a four axis controller with a range of up to 2km. If a smartphone replaces the controller, the range is limited to 0.2km . It has a very sable flight and a top speed of 60km/h, making it very efficient for "take place" and "go back home" operations.

The Bebop can be controlled either by the SkyController or by any smartphone (Android or iOS) using the FreeFlight application[7] and by a combination of the two solutions. In this case, the drone is controlled by the SkyController while the smartphone is used for settings, calibration, displaying flight informations or video streaming. This combined solution is recommended to take advantage of each component. The SkyController for flying operations, as its axis, buttons and triggers both more accessible and very precise, makes flying the drone in manual control very pleasant. And the smartphone for the streaming and access to flight informations only, as otherwise we would have to touch the screen to fly it. Finally, the smartphone can either be fixed on the SkyController or put in FPV glass providing really good immersion. Note that we do not use the FPV glasses in this project because it does not make sense and does not respect the regulations.



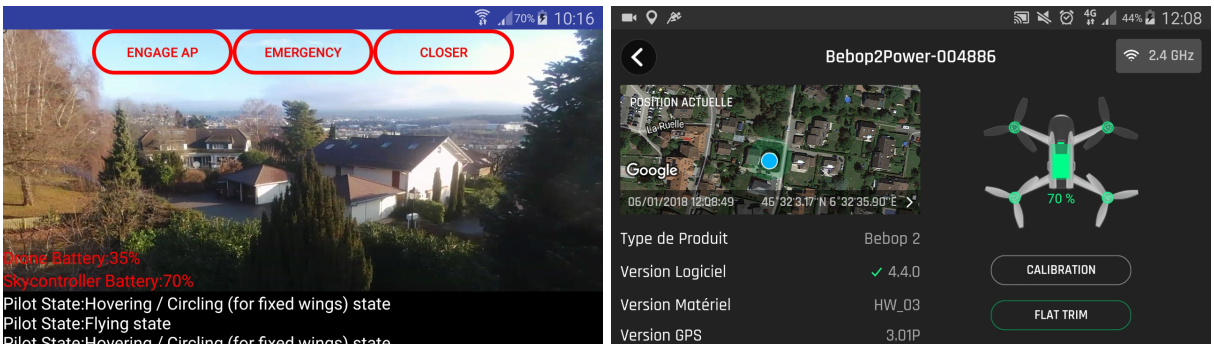
(a) (b)

Figure 3: FreeFlightPro Android application with (a) and without (b) the SkyController2

The application provided with the drone (see Figure 3) is useful but limited. This project will now consist in creating an Android application to substitute to the existing one. This application will use the phone as computational power and as an extension of the controller. Note that we keep the FreeFlight application for some specific functionalities like software updates (Bebop 2 and SkyController 2) and for calibration with visual feedback (see Figure.4).

Parrot launched one year ago a new product on the market, the SLAM dunk sensor (Simultaneous Localization And Mapping). Compatible with multiple drones, it uses its two ultrasound sensors and its high

[6] www.parrot.com/fr/drones/parrot-bebop-2-power-pack-fpv
[7] community.parrot.com/t5/Base-de-connaissances-Bebop-2/FreeFlight-Pro/ta-p/150177



(a) (b)

Figure 4: New Application layout combined with some specific FreeFlightPro features

resolution stereo vision to compute a point cloud of obstacles. Our project could benefit a lot from the SLAM but it has never been available. Although there are other 3D sensors like ZED[8] or Structure Sensor[9], there are currently not directly compatible with Parrots.

2.4 Final Project Proposal

Our initial project clearly needed new directions as counting less than 24 people is mostly useless. Brainstorming on the possible alternatives, we thought about counting people from the side, analyzing people's emotions[10] and their behavior or finding someone in particular in a crowd. We had to find something compatible with the available material, our time frame and the flight regulations. So we decided to work on the following points. What about using the drone as a scout in an everyday situation? By "scout" we visualize something in between a dog and a personal assistant. Could the drone interact with someone? How will people react and interact with a drone? In which conditions and situations will people feel the most comfortable with a drone? We want to find answers to all these questions and make the application usable by everybody.

The final project proposal takes all these points into consideration. We will implement an Android application as a SkyController2 extension to make a Bebob 2 drone interacts in a supervised autonomous flight with people. The drone should be able to track something, determines if it is a human being, and decides if it recognizes the subject or not to produce an adapted interaction.

3 Implementation

As mentioned before, we decided to develop an Android application as an autonomous flight extension to the SkyController2. The application should then be able to control the drone and analyze the received video stream to bring the drone to the next autonomous state. Let us briefly present the tools we used and the Android implementation. We will then explain some testing procedures we used to verify the robustness of the implementation.

3.1 Tools

Parrot SDK

The Parrot Software Development Kit[11] is well documented and is quite a complete interface. We can send commands to the drone and receive events periodically. It is mainly written in C, but it also provides libraries for Unix system, Android (Java) and iOS (Objective C). We can find some modules or intermediate

[8] www.stereolabs.com/zed/
[9] structure.io
[10] developer.affectiva.com
[11] developer.parrot.com/docs/

extensions to use the SDK in Python[12] or in C++[13]. The SDK also comes with a simulator and a responsive development forum.

Android

Today, almost everyone has a smartphone. Even if the computational power is limited and not really adapted to real time computation, we decide that everyone should be able to fly his drone using our application. The tools provided by Parrot (SDK) are available for Android and we can also use native library in C/C++ inside our code to accelerate it. In the following chapters, we will assume that the reader has basic knowledge regarding Android development. If some terms or concepts introduced below are not clear, please refer to Android Developers[14] for more details.

OpenCv

The drone has to fly autonomously, i.e. from its own single perspective vision. Therefore the main work of the autopilot is an image processing part, which should be achieved using OpenCv, a powerful Computer Vision tool. OpenCv supports C++, C, Python and Java interfaces for Windows, Linux, Mac OS, iOS and Android. Even if compatibility is a challenge, it is always possible to build our own version which is optimized for our application. During this project, we finally decided to use JavaCv, a OpenCv wrapper for Java.

3.2 Android

Basic App

The first step was the basic application which we built from the Parrot samples[15], which implements a `DroneDiscoverer` and corresponding Android Activity to see the drone's Wi-Fi and to succesfully connect to it. This first application also gets the video stream from the drone and includes buttons to produce a full duplicate of the SkyController. The project cannot run smoothly on emulated Android tool because it needs an hardware connection and/or its own Wi-Fi connection. We decide to run the application during development on a *Samsung GT-I9505* on *Android 5.0.1 (SDK 21)*

Debug Adapter

Because of the last point and for debugging purposes, we implemented the following. The application architecture is different if we use the phone with the SkyController or just the phone alone. The basic problem comes from the SkyController which needs to be connected by USB to the phone, taking then the only micro-USB port on the phone. The procedure of getting log console messages if the phone is not directly connected to the computer is cumbersome. Hence we created some intermediate class simulating the SkyController. This way, the phone can remain plugged into our computer during development. Note the project running with this adapter is not fully functional and only adapted for some specific test and debug situations.

`SkyControllerExtensionModule`

This module is the file which contains all the Parrot SDK calls. Basically it is a set of functions we can call elsewhere in the program. It allows the settings of the drone's configuration and sends behavior commands like take-off, land, setPitch, setYaw, etc.

`FlightPlannerModule`

The flight planner module uses the Google Maps API to load, create, modify, send and execute flight plans using the Mavlink flight protocol. This protocol is explained more in details here[16]. It allows us to give

[12] github.com/robotika/katarina
[13] github.com/lukaslaobeyer/libdrone
[14] developer.android.com
[15] github.com/Parrot-Developers/
[16] mavlink.io

Figure 5: Color blob detection example, on a yellow machine

GPS mission to the drone using fixed points (i.e. a point in space defined by longitude, latitude, altitude and orientation) and transitions between them. From the flight planner, we need a set of tools to create the Mavlink files and we can find those in the `MavlinkUtilities` class.

OpenCv Integration

To integrate correctly OpenCv features in the application, we mainly worked in three steps. Firstly, we implemented everything in Java on Ubuntu, using a webcam and a computer. Then we migrated to Android on a OpenCv-only project using the phone camera as input. Finally, we integrated the code into the final project with the drone's features and the drone's input frames. With this last step, we lose many frames per second because the stream must be sent and decoded before being used by OpenCv tools. Also, as a smartphone computational power is much lower than a desktop computer's, we cannot always compute everything for each frame. The selected solution was to use the last known frame each time the computation begins. OpenCv also provides some samples for Android application using the phone camera[17]. The samples are available as demo on the Android's *Play store*[18].

Blob Detection

The implemented algorithm is based on color blob detection assuming a reference color is selected by the user in HSV format. The user selects a color by touching the screen at his point of interest. The touched pixel becomes the reference pixel with a relative x and y coordinates in the frame and a color value.

We process the frame by smoothing the input image with a small Gaussian kernel filter and then down sampling it to reduce complexity (i.e. working OpenCv algorithm on a smaller frame is cheaper). Now, we can convert the RGB frame into HSV and apply a mask to keep only the pixels where the color is close enough to the reference one. To define the metrics, we use a constant vector with a maximum accepted distance for each component (i.e. H, S, and V). A lower and upper bound can be computed and double thresholding is applied on the HSV frame. At this step, we get all blobs with a similar color than the reference. Because the user selects, by touching the screen, the reference pixel in the image, the single desired blob is the closest from it. To determine this last information, we compute the blobs' contour and bounding box to estimate corresponding center point. The closest center is finally determined using the euclidean distance between the reference pixel and the estimated blob's center pixel. While the user does not click to change its blob, at each frame, the closest center estimated on the previous frame becomes the new reference pixel. Hence the blob is tracked frame by frame until it disappears from the screen. To prevent that, we ask the autopilot to adapt its camera vertically and its yaw orientation to keep the blob at the center of the screen.

Figure 6: The same color blob as in Figure 5 but after getting closer

Get Closer Function

The blob is now tracked, but it is probably too far from the drone. The idea is now to ask the autopilot to get closer. Because we do not have the SLAM dunk sensor, getting closer to a undefined object (i.e we only have a blob), is complicated. A naive idea would be to use the size of the blob, comparing it to the size of the whole frame and get closer until the size is under a given threshold. This solution is not only naive but also dangerous. The final goal is to select people and at the beginning we only get a blob of them. This blob can really reach different sizes if it is for example the head or the whole body. Then when should the drone stop? If we assume the head as reference size, the drone will not get close enough. If we select the body, and if we assume the body as reference, the drone will get dangerously close to the subject. Determining the depth position of objects with a single image is very difficult and impossible without a context (i.e. the object is a average human). We thought about motion parallax between multiple pictures but it provides results that are too approximative. To work around this problem, we decided to implement a supervised get closer function. In other words, the drone gets closer to a blob when the user presses a button and immediately stops when the button is released. The get closer function assumes the blob is kept at the center position of the camera. It uses then the yaw direction and the vertical camera tilt to determine the direction to follow. By definition, the drone always faces the x axis and the z axis is oriented to the ground. A look at Figure 7, shows that the $dx = c * cos(tilt)$ and $dz = c * sin(tilt)$, where c is a relatively small constant. Hence we can command the drone to move on its relative $(dx, 0, dz)$ direction.

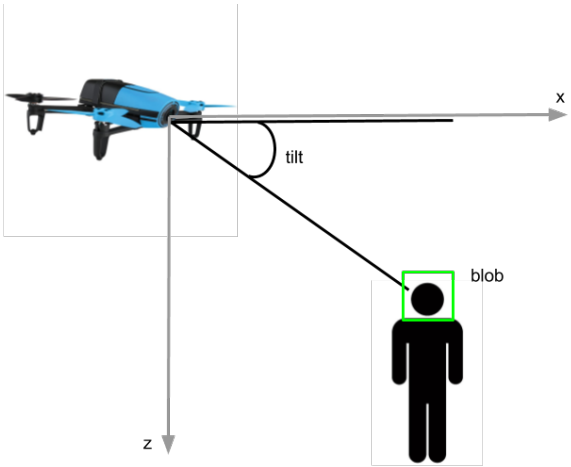


Figure 7: Direction of the drone when getting closer to the blob

[17] github.com/opencv/opencv/tree/master/samples/android
[18] play.google.com/store/apps/details?id=com.jnardari.opencv_androidsamples

Figure 8: The drone recognizes a friend. If the autopilot was engaged, it would react accordingly.

Face Detection

Once the drone is close enough to the blob, it is supposed to detect the face of the subject. We assume the noise made by the drone will tend to stimulate the subject attention, making the subject to look towards the drone's direction. The face detection is realized using OpenCv and Haar Cascades file for frontal face [9]. Native methods are already implemented and return a matrix of rectangles containing the faces. The face detector takes as parameter the minimum face size to validate a detection. In the implementation, the autopilot starts to search for a face in the frame when the blob size reaches a threshold size. When a face is detected, the drone replaces the blob's center and tracks the face's center. If multiple faces are detected, we take into account only the closest one to the blob using once more euclidean distance. At this step, we assume the drone will not miss-detect the desired face when faces are detected. In other words, the drone will never detect a set of faces not containing the desired one. Once the face tracking replaces the blob tracking, we can disable the blob's computation and move to face detection at every frames.

Face Recognition

Once a face is detected, the autopilot tries to recognize it using the Eigen face process developed by Sirovich and Kirby (1987) and used by Matthew Turk and Alex Pentland in face classification[6]. From its own database of known people, the autopilot can determine how similar the detected faces are to an average image and the database[19]. If this similitude is higher than a given threshold, we can assume the face is known. The database is composed of n images of each known person. Increasing n would increase the algorithm accuracy, but also the training and testing computation's times. The limitations of Eigen face are obviously the training part and the model size. Every time we want to add a new face in the database, we need to perform a whole training again over the mn samples (m is the number of known person). If we want to train the model in real time we will quickly be limited. The same problem appears if we pre-trained the model, because loading it is also expensive and takes a while. For this project, we have two alternatives: we can either precompute the known people's Eigen components for a small amount of ($m < 10$) people or give the drone the ability to train real time with a buffer of the ($m \leq 3$) last people met. Finally, we decide to implement the first part with a few solution for the second. In other words, the drone has a small database of known people labeled with a unique name and a type like *Admin, Friend or Enemy* (the algorithm can differentiate between two suject of the same type). Other faces are *Unknown* and at this point the drone's behavior allows us to take new pictures for the database. We decide not to add this feature because we only use a few samples per person and the conditions under which pictures are taken are very important. The whole known people are captured with the same light, same position, etc.

[19] http://www.shervinemami.info/faceRecognition.html

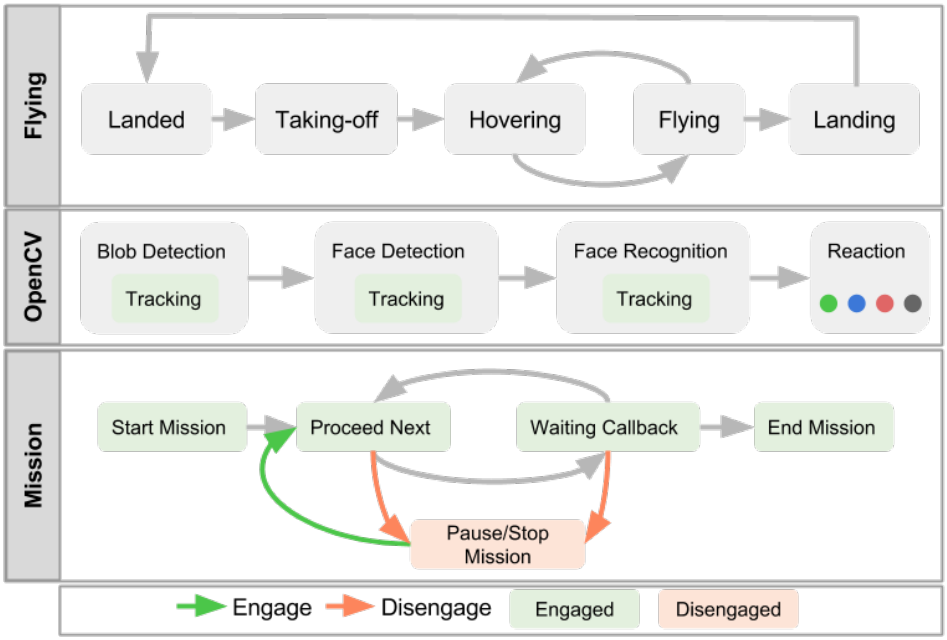


Figure 9: The three State cycle blocks of the drone. They are completely independent. Once switched on, the drone is in one of the flying states. Once the application is started the user can initiate or reset the OpenCv thread by touching a pixel on the screen. When the drone is not currently inside another mission, the user or the autopilot (via reaction state) can start the mission pipeline.

Iterative Mission

As already explained, the drone's autopilot acts more like a finite-state machine than a real artificial intelligence. It links together all the modules making the drone more autonomous. The iterative mission implements the concept of very simple mission elements (i.e. we can assimilate them to a single order sentences like *"climb 10 meters"* or *"climb at 10 meters"*), and a list of elements. The mission can evolve dynamically acording to the situation.

`AutoPilot`

The autopilot basically gives full controls to the smartphone and its computer vision algorithms (i.e the operator loses the controls). For security reasons, the autopilot needs to be fully supervised by the operator and we also include an auto disengage feature on controller axis inputs. From previous implemented modules, the drone is able to follow a Mavlink Flight Plan, which needs GPS and then to be in outdoor mode. It is able to follow an Iterative Mission to complete the Mavlink features (i.e. indoor and outdoor mode). The drone can track a selected blob and keep it in the middle of the screen, get closer until a face can be detected, tracked, and recognized. Finally it can respond accordingly to its recognition.

In summary, let us describe a basic standard state pipeline. Initially after switching it on, the drone is *landed*. From this state it can *take-off* and then *hover*. When inputs are given (manually or from the autopilot) the drone switches to *flying* state while applying the input and then *hovering* again. When it lands the drone switches to *landing* before finally getting back to the *landed* state. With our application, this pipeline differs a little bit. Firstly, the autopilot can be engaged or disengaged at any time and into any state. Secondly, at any time, the drone can switch into one or multiple sub-states: the drone can be in a mission or in a flight plan state and also be in a OpenCv thread states. See Figure 9 for more details about the states. Note that flying states are part of the Parrot architecture while Mission state and OpenCv thread states are added by the application. The OpenCv Thread starts when it is ready (i.e after loading everything needed). Missions and Flightplans can be started from everywhere if the autopilot is engaged and must be ended (or at least stopped) before starting a new one.

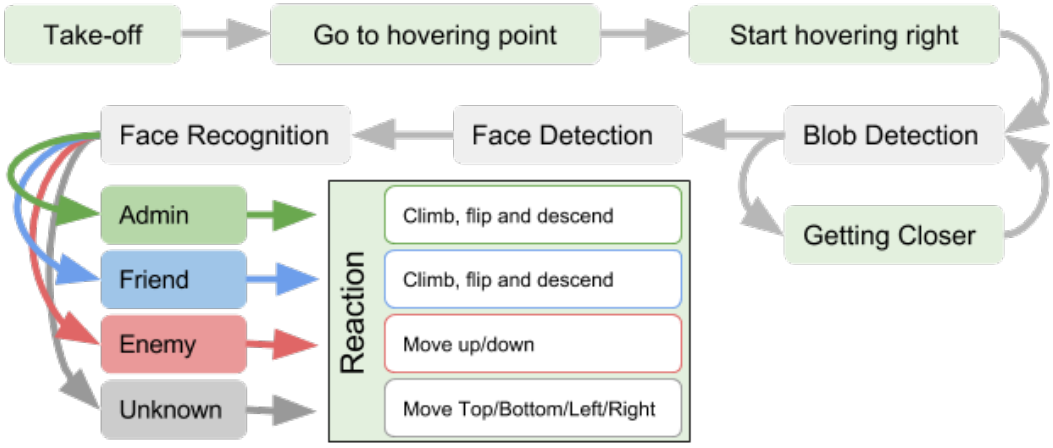


Figure 10: The proposed experiment represented as a decision tree

3.3 Tests

Each module has been tested separately in a free flight area field. The Parrot company provides a simulator called Sphinx[20] which runs only on a subset of Linux-64-bits distributions. The simulator is too heavy to run on our available machine or is simply incompatible. Even if the simulator would probably be useful, we decided against using it. Instead we decided to implement everything as separated modules which could be tested independently from one another. For the OpenCv part, we tested the correctness of the implementation using first a computer with a webcam and a smartphone with its own camera, before testing on the smartphone using the drone's camera. Finally we let the OpenCv thread controls the drone in flight.

4 Experiment

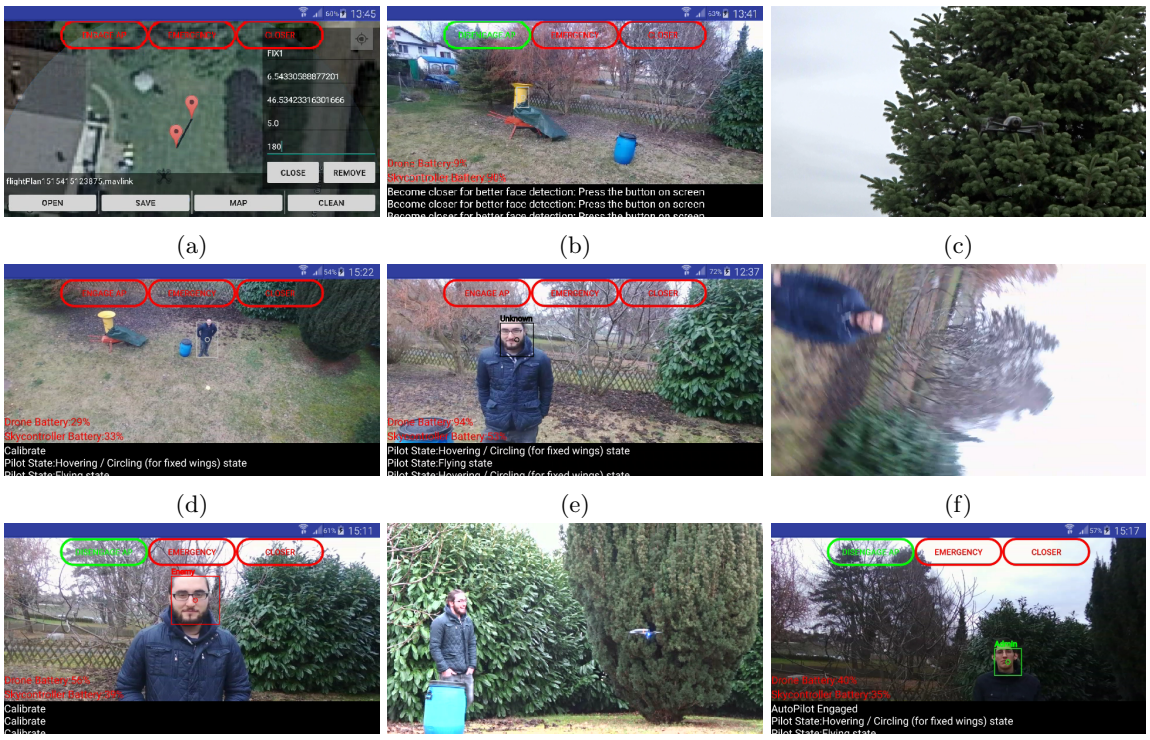
Once the application was functional, we designed a small experiment. The experiment tries to observe how people react and interact with a drone, and also how the drone can response when recognizing someone. The experiment is designed as follow:

1. The drone takes off and reaches a hovering point given Mavlink protocol.
2. It turns slowly on the spot waiting for an input from the operator.
3. Once the operator selects a pixel in the screen, the drone adapt its yaw and camera tilt to track the corresponding blob.
4. The drone gets closer to the blob following the operator's instructions (i.e he decides when the drone is close enough).
5. Computes face detection to replace the tracked blob by a face.
6. Computes face recognition, and reacts accordingly.

For this experiment, we train the drone on a database of three people (an *Admin*, a *Friend* and an *Enemy*) with 50 samples of each faces. The labels are attributed arbitrarily, and the samples are all taken under the same conditions (i.e with the same camera, under the same light field and with the same background). The capture is made using a video camera and samples are extracted from the current stream. We ask the samples to be a bit different (i.e not capturing consecutive frames, or too similar samples). The head motion during the capture is assumed equivalent for all three subjects.

The model is already about 50Mo and take approximatively 10s to load, and 15s to train. We tested with a fourth person, but the time is almost doubled (about 17s to load and 28s to train).

[20] http://developer.parrot.com/docs/sphinx/



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 11: A nine frames selection from the realized video. (a) edition of a Fix inside a flighplan. (b) blob detection on a yellow object. (c) the drone is hovering left waiting user input. (d) user clicks on a blob on ground from hovering spot view. (e) The blob has a face which is not recognized. (f) after recognizing a friend, the drone realizes a flip. (g) this time the recognized face is an enemy. (h) the drone is reacting accordingly making fast top/bottom movements. (i) the drone recognizes its Admin

During face recognition, the drone can recognize *Admin*, a *Friend* or an *enemy*. When the recognition is verified during 1s, the drone start an interaction. If the drone recognizes *Admin* or a *Friend*, it is *happy*, it climbs and makes a flip. If the drone recognizes an *Enemy*, it is *nervous* and it climbs/descends a few times trying to force the *Enemy* to retreat. Finally, if the drone recognizes no one, it will make some slow movements in the four directions (top, down, left, right) allowing its camera to add the new samples to the database. See Figure 10 for corresponding pipeline.

This experiment assumes the following points:

- The tracked blob stays inside the frame during the whole *get closer* operation.
- Once the drone is close enough, if the blob has a face, then this face is looking in the drone's direction.
- The subject stands in front of the drone during the whole experiment.

5 Results

5.1 Video

To illustrate the whole project and the explained experiment, we realized a short video[21]. In just over four minutes, you can see the whole application working under specific conditions. All features are shown and some parts of the experiment are demonstrated. See Figure 11 for a selection of frames of the video.

[21] `https://youtu.be/TUmERdd$_$JeU`

5.2 Limitation

As we know, this project has some limitations. If we exclude limitations induced by regulation and materials, we can notice this project is facing limitations regarding computer vision algorithms and some machine learning concepts (face detection and recognition). None of these problems are completely resolved. The human detection and especially the face detection are based on shape classification. From Haar Cascade files, there is one different classifier for each specific body or face sub-part. There is multiple files only for frontal face detection and others for profile face. For better predictions, we could combine those files and enforcing the model to using also left and right eyes or mouth detection. But using multiple files, we would have to compute multiple predictions on each frame, in addition to a merging results algorithm, making the computation too heavy for a traditional smartphone. From the used file, face are well detected in a frontal way (i.e if the subject is looking at the camera). The face detection from the top is a kind of bottleneck and it is in part why we compute blob detection first.

For the face recognition, we already talked about some limitations, but we also noticed the results are not as good as expected. The model tends to make wrong predictions when the testing condition are too far from the training ones (light, orientation, motion, etc.). This problem definitely comes from the model's size, we should use more samples per subject but as seen before, the model is not really scalable. The phone will be limited in memory (RAM) and the training, loading, and testing processes will overflow the computational power. Not forgetting that finding correct hyper-parameter (i.e threshold) is not easy with a small model like that. Sometimes, under some arbitrary conditions, our threshold will be too large and a *Unknown* people will be recognized and other times the same threshold will be too small ending with a *Admin* unrecognized. This last point shows how computer vision algorithms are complicated to verify and are almost never truthful.

5.3 Complementary Observation

During the previous experiment as well as during the whole application development, we observe the following kinds of people. When using the word "people" we mean "every person who has been in contact with the drone". Some people do not approve of the drone, the noise is disturbing and the camera is a lack of privacy. Some others are really interested but are also wary. A drone can be dangerous and they first keep their distances, sometimes even if the drone is switched off. Then they get used to it. The drone can fly closer and closer to them until a safety threshold (about 1m) where the subjects feel uncomfortable and step back naturally while the drone is getting closer. We observe that everybody is keeping the drone in sight, especially when it is close. We also observe than people are more tolerant to drones at very low altitudes (about 30cm) or higher ones (from 2.5m and above). The critical altitude is the human height, because people care about their faces, and feel often nervous or uncomfortable with a drone so close. The experiment exploits all those points. As expected, enemies are surprised by the amplitude of the drone during interaction and tend to step back. On the other hand, the flip also surprises Friends making them step back too. A flip is probably too sudden and does not really put subjects in trust.

6 Future Work

As we see, this project is composed of modules and of an autopilot using them. We can add as many modules as we want as long as the smartphone computational power can handle it. Implemented interaction are probably too simplistically. More advanced interaction with maybe more intermediate states can be an interesting feature to add. For now, the drone interaction is a small list of procedures. We could imagine mission of hundreds of procedure, with multiple splits inside the decision tree. With more time available, we would also like to make the application more complete and user friendly. It could be interesting to implement all FreeFlightPro's features so as to not have to keep both applications.

Then the test part can be more elaborated and systematic. It can be useful to get some warranties for all functionalities. The Computer Vision field is in full expansion. Every days, algorithms are improved and accelerated. Reaching a better accuracy in face detection and recognition can be a very good improvement for this application. Moreover, the application is working quite well under very specific conditions and environments, therefore a huge improvement could be to increase the robustness of those variations.

This project can be considered as a first step towards human and drones interaction. Even if now the results are limited and the procedures are fully supervised, we are convinced drones will become more and more

present, independent and autonomous in the future. We already see movies with android robots and artificial intelligence, drones will probably join them. Drones have many advantages, and the disadvantages tend to be reduced or removed (i.e. price, noise, friendliness). Questions to ponder are : how will drones be used in the future? Which direction will the drone development take? Will everybody get one or use one like a smartphone? What about drones in five, ten and twenty years?

7 Conclusion

As we see, Human-Drone interaction is not well defined yet. For most people, drones are only small supervised aircrafts and they are right, the operator being always needed to react in case of problems. The interaction is truncated and it is difficult for the human part of it to fully grasp the implications of an interaction with a fully autonomous flying object. Even if the use of the drone's single perspective camera as control inputs is very complicated and raises many problems which still need to be addressed, the concept is no less interesting. The results must be put into perspective considering the constraints induced by both the camera and the smartphone. In this project, we showed a subset of features which will certainly be improved and completed with the use of multiple or better video cameras and sensors. One should consider this project as a first implementation step opening a lot of possibilities towards Human-Drone interactions. We think this subject deserves to be investigated again, maybe with a different or a complementary approach.

8 Acknowledgements

I would like to thanks Professor Jeffrey Huang for giving me the opportunity and the budget to realize this project. Immanuel Koh for his support, his help during our brainstorming sessions, and his periodical feedback. I also want to thank Pierre-Yves Mottier for his proof-reading of this report, and Florine André, Frederik Künstner and Adan Häfliger for being my subjects during the experimental part.

References

[1] CBSNews. Amazon unveils futuristic plan: Delivery by drone, December 2013.

[2] Cooper Bills, Joyce Chen, and Ashutosh Saxena. Autonomous mav flight in indoor environments using single image perspective cues. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 5776–5783. IEEE, 2011.

[3] Jesus Pestana, Jose Luis Sanchez-Lopez, Pascual Campoy, and Srikanth Saripalli. Vision based gps-denied object tracking and following for unmanned aerial vehicles. In *Safety, security, and rescue robotics (ssrr), 2013 ieee international symposium on*, pages 1–6. IEEE, 2013.

[4] It Nun Thiang and Hla Myo Tun LuMaw. Vision-based object tracking algorithm with ar. drone. *Red*, 160:179, 2016.

[5] Omar Javed and Mubarak Shah. Tracking and object classification for automated surveillance. In *European Conference on Computer Vision*, pages 343–357. Springer, 2002.

[6] Matthew A Turk and Alex P Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE, 1991.

[7] Jacobo Jiménez Lugo and Andreas Zell. Framework for autonomous on-board navigation with the ar. drone. *Journal of Intelligent & Robotic Systems*, 73(1-4):401–412, 2014.

[8] FOCA. Drones and aircraft models, 2017.

[9] Michael J Jones and Paul Viola. Robust real-time object detection. In *Workshop on statistical and computational theories of vision*, volume 266, page 56, 2001.