Jedi Training V.3

- Calandriello Daniele
- Martinoia Diego

Abstract:

Jedi Training is an Augmented Reality Robo-Game.
It recreates the Jedi training of Luke Skywalker aboard the Millenium Falcon during the XXX movie of the “Star Wars” saga.
In the 3rd version of the project, a real flying drone (Parrot ArDrone) has been included in the game, in order to enhance the game experience.

The Player, wearing a specific game uniform, is armed with its light-saber. The drone flies around him in circles and tries to hit him with its laser blast. The Player has to parry those shots with his light-saber. The game is highly focused on giving a 3D-depth experience, and so the Player can move around while the drone follows him/her.

The drone behavior is completely autonomous once the game is started (i.e. there is no human control of the drone). This is achieved via processing the video stream of the frontal camera in order to understand the Player's position and his/hers current parry stance. OpenCV V2.1 libraries have been used for this processing, in their C version.

All the game runs under Linux / C environment.

Index:

User Manual:

1) Os Overview:

Jedi Training V3 is completely written in C language, and runs under Linux environment. The product consists of 3 main parts:

a) The ArDrone Official SDK
b) The OpenCV v2.1 Libraries
c) The game itself.

The ArDrone Official SDK was mainly used in order to achieve a much easier decompression of the video stream, which otherwise would have been a non-trivial task.
Unfortunately, the usage of the SDK imposes not few restrictions on the nature of the software, which has strictly to be multi-threading (See apposite section for details).
Also, the SDK has to be compiled using “make”, and not gcc directly, thus possibly causing some problems to a non-expert programmer.

The OpenCV libraries are used to process the decompressed video stream to understand the Player's position and his/hers parry stance. Using these data, the artificial intelligence behind the game makes some probabilistic decisions about what to do next (this question is raised each frame, 15 times a second), and this “fuzzy” decision is then translated in specific power indications to the drone's engines.

In addition, GTK libraries have been used to implement the GUI and libconfig to load several parameters, such as default distance and the coefficients of the controllers, at runtime.

In sum, the libraries required by the software are:

ArDrone SDK1.5
OpenCV2.1
LibGTK2.0
LibSDL1.2
Libconfig1.3.2
Joystick Package for Linux

Please note that all libraries must be installed since they are not statically linked.

There is also a Shell script (ardrone.sh) that is needed before the launch of the game, in order to achieve a connection with the drone.

2) User Interface

3) Configuration File

In the configuration file, all the parameters of the software are stored. This is the meaning of each one of them:

MINHROBE = 141;

MAXHROBE = 180;
MINSROBE = 145;
MAXSROBE = 195;
MINVROBE = 53;
MAXVROBE = 113;
MINHSWORDH = 224;
MAXHSWORDH = 255;
MINHSWORDL = 0;
MAXHSWORDL = 19;
MINSSWORD = 184;
MAXSSWORD = 255;
MINVSWORD = 135;
MAXVSWORD = 215;

Hue,saturation and values of both the robe and the sword, should always be produced with the automatic regulator, it is reported here because the regulator needs a clean videoconfig.txt file to update the values.

ERRORFROMCENTER = 5;
Threshold in pixels over which the drone take measures to bring the player back to the center of the image, influences the movement on the z-axis of the drone (yaw)

ERRORDISTANCE = 25;
Threshold in centimeters over which the drone take measures to adjust the distance from the player , influences the movement on the y-axis of the drone (roll)

DEFAULTHEIGHT = 850;
Default height at which the drone should hover.

ERRORHEIGHT = 100;
Threshold in centimeters over which the drone take measures to bring it's altitude back around DEFAULTHEIGHT, influences the movement on the z-axis of the drone

YAWK = 0.007;
Rotation's controller coefficient.

GAZK = 0.001;
Vertical movement's controller coefficient.

PASTWEIGHT_PITCH =0.9;
Weight of the past coefficient in the weighted mean used to smoothen the changes of direction.

DEFAULTDISTANCEINCM=180;
Default distance in centimeters from the player at which the drone should hover.

SHOTEVALUATIONREQUESTVALUE=20;
Number of frames between a shot and it's evaluation, look under for the meaning of this.

buffWidth = 320;
buffHeight = 240;
Width and height of the video stream in pixels.

itemHeightInCm = 115;
Height in centimeters of the robe, used to determine distance

minAllowedPixelHeight = 30;
Inferior limit in pixel under which objects are not considered as robes.

pointOfFocus = 210;
210 is a magic number that comes out from the point of focus of the camera!

PASTWEIGHT_DIST = 0.5;
Coefficient for the weighted mean used to avoid spikes in distance relevation.

maxChancheOfShoot = 0.003;
Coefficient used in the formula actualChancheOfShoot = maxChancheOfShoot * chancheOfShootParameter to calculate the actualChancheOfShoot starting from the chancheOfShootParameter obtained from the video stream.

DISTLOWKPLUS = 0.002;
DISTHIGHKPLUS = 0.008;
DISTZEROLOWPOINTPLUS = 20.0;
DISTLOWHIGHPOINTPLUS = 100.0;
DISTLOWKMINUS = 0.00075;
DISTHIGHKMINUS = 0.0025;
DISTZEROLOWPOINTMINUS = 20.0;
DISTLOWHIGHPOINTMINUS = 100.0;
Parameters of the fuzzy controller. See picture for more info.

4) Batteries

At the time of writing, there are 2 batteries in the AirLab. Each one of them lasts for about 15 minutes of flight, and it takes something less than 2 hours to recharge (the drone is very power-expensive, each motor consuming up to 15W). The drone has no on/off switch: plugging the battery in is the on signal, disconnecting it is the off signal. It is VERY important to plug the battery in while the drone is laying on a horizontal plane, since that is when it calibrates its gyroscopes. In case you notice that the drone configured its planar position wrong, restart it laying horizontal. If this persists, telnet to the drone, change directory to /data and copy the files back.trims.bin and back.fact_trims.bin over trims.bin and fact_trims.bin respectively, If the back. Files are missing just erase trims.bin and fact_trims.bin and restart the drone on a flat surface, then make copies of trims.bin and fact_trims.bin as back. trims.bin and back.fact_trims.bin.

After you plug the battery in, the motors' leds should turn red. After some seconds, when they turn green, the drone is ready to fly. For more information, consult the drone guide.

Hardware:

1) Drone Specifications:

AR.Drone Specifications

The AR.Drone has been designed using very light and robust materials. The main structure is made of carbon-fibre tubes and fibre-reinforced PA66 plastic parts. The hulls are injected with EPP.

SPECIFICATIONS

Safety system(automatic locking of propellers in the event of contact)
Running speed: 5m/s; 18 km/h
Weight:

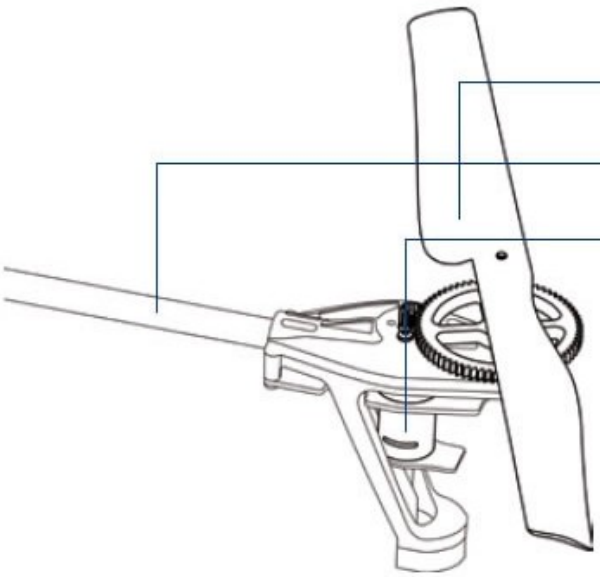
- 360 g without hull
- 400 g with the hull

Running time 15 minutes

DIMENSIONS

With hull: 52,5 x 51,5 =cm
Without hull: 45x29cm

- Embedded computer system
  Linux and inertial guidance system with MEMS

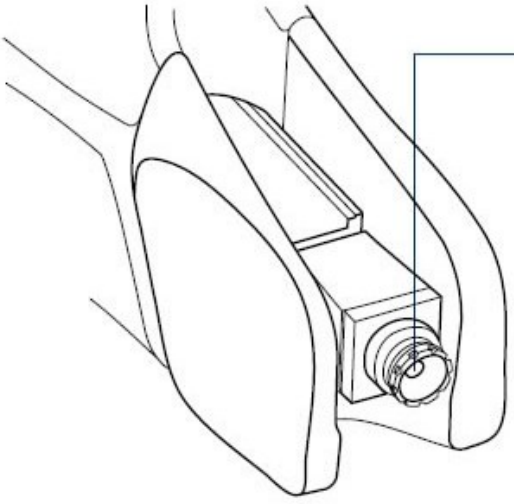


AERONAUTIC AND STRUCTURE

- **High-efficiency propellers** (specially designed for the Parrot AR.Drone)
- **Carbon tube** structure

MOTORS AND ENERGY

- 4 brushed or brushless motors, depending on the version (3,500 rpm, power: 15W)
- Lithium polymer battery (3 cells, 11,1V, 1000 mAh)
- Discharge capacity: 15C
- Battery charging time: 90 minutes

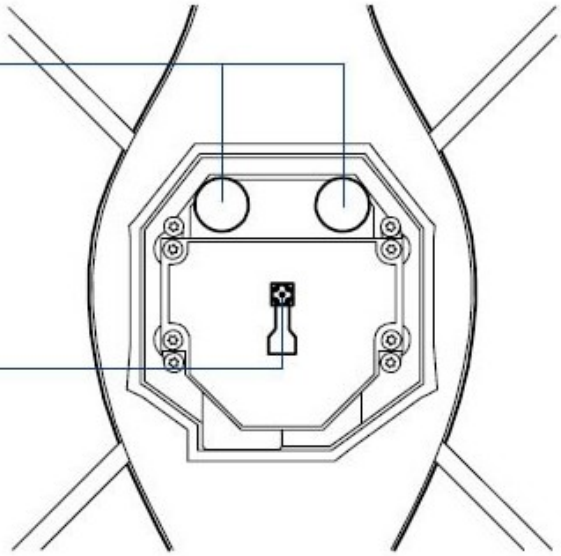


FRONT CAMERA:

- **93° wide-angle diagonal lens camera, CMOS sensor**
- Encoding and streaming of images on iPhone®
- Video frequency: 15 fps
- Camera resolution 640x480 pixels (VGA)
- 1D tag detection
  - Validation of shots fired at enemy drones
  - Estimate of distance
  - Detection distance: 5 metres
- 2D tag detection
  - Positioning of virtual objects
  - Calculation of markers of virtual objects
  - Detection distance: 5 metres
- Video feedback on the iPhone® screen

ULTRASOUND ALTIMETER

- **Emission frequency:**
  - 40kHz
- **Range 6 metres**
- Vertical stabilization



VERTICAL CAMERA: HIGH SPEED CAMERA

- 64° diagonal lens, CMOS sensor
- Video frequency: 60 fps
- Allows stabilization even with a light wind
- Resolution: 176x144 pixels (QCIF)

EMBEDDED COMPUTER SYSTEM

- Processor ARM9 RISC 32 BITS - 468MHZ
- DDR RAM 128 M – 200 Mhz
- Wifi b/g
- USB high speed for future expansions
- Linux OS

INERTIAL GUIDANCE SYSTEM WITH MEMS

- Microelectromechanical systems (MEMS)
- 3 axis accelerometer
- 2 axis gyroscope
- 1 axis yaw precision gyroscope

Removable shaped hull

2) Game Uniform

The game uniform consists of a blue robe, with a hood. When wore, it is long about 115 cm without the hood, 145 including the hood. The length is very important, since this is the main parameter the drone uses to calculate the distance from the player.
It is highly recommended that Player wears the hood during the game, both for increased experience and easier recognition from the drone.

3) Light-saber

The game sword consists of a red plastic tube of those used in swimming-pools as floating helps. Unfortunately, its color was too light-sensitive (i.e. it changed much according to the environment light) and also its structure was making too much friction with the air during the swings, risking to break it.
It has then been reinforced with an aluminum L support (thus making it much more dangerous when swing) and covered in red fabric. An XXX wielding has been added to avoid hand suffering from the L support.

Software:

1) SDK structure and File position

ARDrone_SDK_1_5_Version_20101004 root folder of the sdk

- ArdroneLib libraries for the sdk
- Examples examples programs, the Jedi Training is here
  - Linux this folder contains the root makefile from which you will be compiling the whole application
    - Navigation contains the source code of the ardrone_navigation program, good for inspiration
    - Build/Release contains the compiled programs, JediTraining is the one named linux_sdk_demo
    - sdk_demo/Sources contains the source code of JediTraining

Important files are also:

ARDrone_SDK_1_5_Version_20101004/Examples/Linux/sdk_demo/Build/Makefile:
contains the important lines
GENERIC_INCLUDES+=$(shell pkg-config --cflags opencv)
GENERIC_INCLUDES+=$(shell pkg-config --cflags libconfig)
GENERIC_LIBS+=$(shell pkg-config --libs opencv)
GENERIC_LIBS+=$(shell pkg-config --libs libconfig)
these are used to include gtk and libconfig libraries, and any other library should be added here

ARDrone_SDK_1_5_Version_20101004/ARDroneLib/Soft/Build/config.makefile:
gtk libs are included here

ARDrone_SDK_1_5_Version_20101004/ARDroneLib/VP_SDK/Build/generic.makefile:

the line
$(INTERNAL_STRIP) $@
has been removed from here to avoid stripping and allowing debugging
ARDrone_SDK_1_5_Version_20101004/ARDroneLib/VP_SDK/Build/Makefile:
after the line
GENERIC_CFLAGS:=$(filter-out -DNDEBUG,$(GENERIC_CFLAGS)) -DNDEBUG
added -g to add debug, very dirty solution.

From this point onward everything has to be considered as rooted in
ARDrone_SDK_1_5_Version_20101004/Examples/Linux/sdk_demo/Sources

global_variables.h:
contains all the global variables used in the application, as well as most of the initialization.

ardrone_testing_tool.c:
contains the main code of the application.
ardrone_testing_tool.h:
mainly contains the prototypes of the threads, consult the source code and manual for more information.
fuzzy_controller.h:
contains the code of the fuzzy controller for the distance.
gtk_thread.h:
contains the main loop of the UI.
gtk_callbacks.h:
contains the callbacks used by the UI

2) Communication Principles:

The drone sends the video stream to the application, which decompresses it using the official Parrot SDK. Once the frame is stored in the memory in a RGB 256 depth level format, it gets processed using OpenCV's. From the frame and the navigation data, the application extracts 2 main parameters:

a) The Player's position
b) The “goodness” of the Player's parry stance

In a “normal” situation (Player in sight, parry stance), a decision for the Drone's behavior is taken according to these parameters and translated in terms of
Yaw (Z Axis rotation),
Pitch(X Axis rotation, i.e. side speed),
Roll (Y Axis rotation, i.e. forward/backward speed),
Gaz (Z Axis propulsion, i.e. height speed) ,
PLEASE NOTE: the SDK documentation states an opposite configuration for Pitch and Roll. The one reported here is the one that the drone listens to when sent commands.
These values are then converted in the standard floating-point number configuration (see SDK documentation for details) and sent to the Drone for execution.

If the Player is not in sight, the “normal” execution is suspended and the Drone stops on place and starts spinning on the Z Axis, “looking” for the Player. The Player is considered “in sight” if one of the areas detected as of the color of the Player's robe is big enough (this values is set in the configuration file).

If the Player's parry stance is considered weak enough by the probabilistic algorithm, the Drone will shoot at the Player. This phase consists of an animation in the Drone flight that simulates the firing of lasers, and a certain number of frames after this (this number is stored in the configuration file as SHOTEVALUATIONREQUESTVALUE in frames), the application will check whether the shot was parried or not, by considering the percentage of pixel of the “heart” (a sub-area of the detected robe as of now 1/8, this is hardcoded in the application, in video_stage.c, and not supposed to be changed by the player) of the Player are covered by the sword. This delay is given in order to allow the Player some reaction time.

3) Threads Meaning

The application consists of 7 parallel running threads:

System Threads

1. ardrone_control: sends the AT*CMD commands (movement) to the drone
2. navdata_update: updates the navdata (duh)
3. video_stage: the prototype is defined in Video/video_stage.c that in turn calls the function output_gtk_stage_transform() with a pointer in to the last decoded frame

JediTraining Threads

4. userinterface: the prototype is defined in gtk_thread.h with the main loop and the callbacks in gtk_callbacks.h
5. chasing: this thread takes care of the chasing of the player, it sets yaw and roll to follow the movements and gaz to keep altitude
6. intelligence: the core of the decision. Every 15 ms it picks a random number to choose, based on the percentage of the shoot chance, whether to shoot or not. It also picks a random number to choose how long will it go in a certain lateral direction and keeps going until a countswitch is decremented to 0
7. atmanager: every millisecond sends the movement commands to the drone. If both intel and chasing are off it hover on the fixed position. If both roll and pitch are 0 it sets enable to 0 to make the drone hover on top of a fixed position.

4)High-Level Logic

There are several AI-like choices performed by the application.

At first, the application receives a video frame by the drone central camera (320x240), and starts processing it using openCV libraries. This is more or less the processing algorithm:

- The software converts the color space from RGB / BGR to HSV, at 8 bit depth. From this HSV image, two binary images are created, one corresponding to the color of the Robe, the other to the color of the Sword.
- A morphology transformation is applied to both images. What this do is expanding the 1 areas and then contracting them. This is done in order to merge those areas that are separated by small noise areas (or, in case of the Robe, separated by the Sword itself).
- After this is done, a contour-search algorithm is applied to both images, and the biggest “box” of color in the Robe image is considered the target. This is not considered for the Sword because of its possible inclinations.
- The grade of strength of the current stance is considered as the ratio between the Sword area and the Robe area. This % is one of the parameters used in the movement decision.
- Knowing the size of the Robe in real life, according to its pixel size and the camera point of focus (determined by experiment), the distance from the target is established. This measure is not influenced by the relative position (i.e. above, below) but from the inclination. This distance is one of the parameters used in the movement decision.
- According to the target position on the screen, the error from the center is established. This

parameter is used in the movement decision.

Using all the data from the image, the application calculates Roll (front/back movement, f(distance)) and Yaw(z-axis rotation, f(angle)) according to the fuzzy controller's gain.

To choose whether to shoot a random number is extracted between 0 and 1 and the decision is made according to some threshold that are f(% of cover). If shoot is decided, the drone plays its animation and after a predetermined number of frame the shot outcome is decided according to how much area of the “heart” (a square located in a special area of the Robe) is covered by Sword – colored pixels, and score is assigned according to it.

The drone lateral movement is also random. It keeps moving in a certain direction for a random number of seconds, then switches to the other for a random number of seconds again.

The resulting commands are then encoded in the floating-point notation and sent to the drone.

Issues may occur in very mono-chromatic settings (or when the player is too close / using the sword too close to the camera) because the drone has an uncontrollable color auto-adjustment that really messes up with the logic of the application, which color parameters are static.

If no big enough robe-color blob is detected, the program understands that the player is out of sight, and stops the drone from moving and makes it spin clockwise until the player is found again.

4) OpenCV remarks

There are a couple of things that should be noted using openCV. The first one is that 8-bit depth means 256 levels, while the H coordinate of an HSV image is theoretically 360 levels. To achieve this result, the raw H channel in a HSV image actually contains the half of the real value. In the application, all H values are normalized on 256 levels.

Another things is about the pointer usage. In the application, the image data comes from the drone, so we create the openCV header and assign the data field to the drone buffer. This in openCV makes so that once we de-allocate the image, the data are preserved because are seen as “external”. Thus, a manual de-allocation is required for the data of the image.

Game Evaluation:

The final product appears definitively playable, also enjoyable and funny, although two peoples are required to operate the software (the player and someone pushing the buttons).
The game is extremely light-sensitive though, performing properly only in stable, non-direct light environments.
Also, it has been noticed that there are problems with female players: although it may sound ridiculous, the breasts cast shadows on the lower part of the Robe, changing its color and creating problems with the recognition of the size of the player.

Next Generation Suggestions:

We did not have time to finely tune the coefficients of the controllers, and the game speed can surely be improved in that sense. The user interface can definitively be improved too. For example, it would be good to study some way to remove the necessity of two players, maybe adding a remote control.
Another direction of improving might be looking for a more robust algorithm to make the game less light-sensitive.