SCADrone - PPC Project 2016

Carlos Agon - (agonc@ircam.fr)
Philippe Esling - (esling@ircam.fr)

1 Introduction

Before beginning the final invasion, SkyNet needs to know what is going on over all the planet. Hence, we are going to transform a classical human-piloted drone into an autonomous unit. The goal of this brainless drone will be to perform a *topographical map* of the surrounding environment. This will allow to obtain the information over a specific location and create a 3-dimensional map of a region, to lead the way for the Terminator armies.

2 Sample code

In order to facilitate the coding surrounding the drone, we pre-coded the entire interface in C POSIX for the Drone, with a direct access to all its commands (directions, flips, callbacks and sensors). You can also take the liberty to go through the Parrot official documentation or inside the SDK code to find more advanced command (like the GPS and Go Home options that allow to put the drone in auto-pilot mode back to its point of origin).

2.1 Pre-requisites

2.1.1 Purpose of STLDrone

The sample project *STLDrone* shows you how to receive video stream from a *Bebop Drone*, decode it, display the decoded stream with *mplayer* and get inputs from users to control the Bebop drone. You can use this sample project as a basis for your project. Remember that you can also start from the sample project on the Parrot website to make your project on the *iOS* or *Android* platforms.

2.1.2 Dependencies of STLDrone

First, you need to download and compile the SDK libraries that can be found on the Parrot website

```
http://developer.parrot.com/docs/bebop/
```

We already provide in the package the pre-compiled libraries for MacOSX (which is a bit hard to obtain on your own), while the Unix installation should be straightforward by relying on the command

```
./SDK3Build.py -t Unix
```

For compiling and using this code, you will also need the **mplayer** library to show the video stream, **ffmpeg** library to get the video decoded and **ncurse** library to get inputs from console

On Linux you can get the libraries through apt-get:
`apt-get install ncurses-dev`
On Mac you can rely on Homebrew to retrieve the libraries:
`sudo brew install ffmpeg`

2.1.3 Compiling STLDrone

On Linux and MacOS X platform :
`make clean all`
To run the STLDrone :
`make run`

To clean the compilation :
```
make clean
```

When you run this sample, be sure to be in the console to catch your keyboard event. As MPlayer will open a new window, you could have to click again in the console. The IHM inputs are implemented on an azerty keyboard. Feel free to adapt it just by changing the key comparison in the function IHM_InputProcessing.

2.1.4 Discussion

This project is separated into 6 main parts

DecoderManager

This is an utility class that will perform h264 frame decoding thanks to ffmpeg.

IHM

This is an utility class that will catch inputs from console and send these events to STLDrone. It will also display some pieces of information about the drone, like its battery level, flying state and almost every sensor available.

STLDrone

This is the main class. It will operate the connexion to the drone and the setup of the network and video part. It is also responsible for linking different parts of the project together.

STLDrone_ Callbacks

This code will register for commands callback. If you need to add callbacks, add it in registerARCommandsCallbacks. The Callbacks are used to know either the various *states* of the drone but also to retrieve the values of different sensors (ultrasound, GPS, altitude, position, orientation).

STLDrone_ Commands

This class allows to send control commands (flight, directions and flips) directly to the drone

STLDrone_ Video

In this class, the callback for a new frame received will be called. When it does, it will take a free empty frame from a pool (to avoid an allocation each time a frame is received), put the received data in this free frame and add the free frame to a frame buffer. Another thread will loop, and try to take a frame and pass it to the decoder. Once the frame has been decoded, it will write the decoded frame in a pipe file. MPlayer will read this pipe file and display the frames.

3 The project

3.1 Compiling and connecting

The first part of the project simply consists in being able to communicate with the drone manually and to add a small command allowing for the drone to automatically perform a few succession of commands. To do so

1. Compile the STLDrone code (`make clean all`)
2. Connect your laptop to the WIFI network of the drone (**Bebop_******)
3. Run the STLDrone code (`make run`)
4. Check the commands manually (don't forget to click on the Terminal).

Your work will consist in updating the code (create a function called *simpleDroneBehavior*) to allow the drone to perform a succession of simple commands

1. Takeoff and take some altitude
2. Perform a "square" movement (Forward, Left, Backwards, Right)
3. Ensure that you are back on the original takeoff position and land

3.2 Retrieving ultrasound values

We have simplified your work by already coding the appropriate callbacks to obtain the altitude values, that can be found in the `altitudeChangedCallback`. However, we also need a proper way to retrieve and collect these values. Indeed, even if the drone is in *idle* state, it might still send values anyways.

1. Find a way to collect ultrasound values only if in required state
2. Also only collect values that have changed (not only regarding altitude but also position of the drone)
3. Check that the corresponding values are correct.

3.3 Creating a topographic map

In order to ensure that the drone will create a true topographic map, we must now turn our attention to the current position of the drone. **However,** remember that the altitude also depends on the current *attitude* of the drone (ie. roll, pitch and yaw). We have to rely on two different callbacks (that we have also already coded for your convenience), that are `attitudeChangedCallback` and `speedChangedCallback` (do not bother with the `positionChangedCallback` which only works when the GPS is enabled).

1. Find a way to collect both the position and altitude altogether
2. Even though we provided a position through speed change, it seems buggy, check it.
3. Ensure that the values are correct and create a 3D map of collected values (use MATLAB, OpenGL, d3.js or any program allowing to produce a 3-dimensional scatter plot)
4. Also add the successive positions of the drone as a separate 3D line.

3.4 SCADE Modeling

3.4.1 Handling alert states

The first **mandatory** modeling is to follow the third law of robotics from Isaac Asimov; "*A robot must protect its own existence*". The problem with our existing piece of software is that we completely override the on-board protection mechanisms of the drone. Hence, the flight states of the drone must be handled correctly.

1. Find all possible "states" of the drone in the documentation
2. Model these through a SCADE state machine and design the appropriate responses

Do not forget that also other types of signal could give an idea on the state of the drone and you might want to model these as well, over the simple "flight states" (think about the battery level and motor states for instance).

3.4.2 Cruise, altitude and speed controls

Based on the same operators that we created together along the first two parts of this course, we want to be able to control the overall status of the drone regarding the *cruise*, *altitude* and *speed*.

1. Devise the cruise, altitude and speed controls as three different SCADE operators
2. Think about the potential need for external signals and boundaries
3. Change your controls so that the boundaries can be adapted in real time

3.4.3 Flight modes and manual override

The drone needs to be set to four different modes, *idle, topograph, override* and *flight control.* Basically

idle	the drone is currently waiting (takeoff, landing and hovering)
topograph	the topographic mode imply that the drone currently performs value recording
override	the drone is now manually controled
control	the drone is adjusting its position or attitude

Based on these states, find a way to produce a global state machine for all states of the drone.

1. Topographic and control mode are linked and allow the drone to either adjust itself or record the map of the region
2. The override mode can be set on or off and must allow the user to control the drone manually

3.4.4 Topographical logic

The topographic logic followed by the drone is that it should densely sample points of the space to have the most possible altitude values. You must devise at least two different strategies to perform the topographic map

basic	Based on the "square" movement, you must at first define "hard" boundaries on the drone (maximum authorized distance to the left, right, front and back of the original position of the drone) and it will start performing a "snake" pattern. The idea is to oscillate between the *control* mode (where it makes a slight movement in either direction) and then the *topograph* mode (where it records the sensor values).
statistic	This mode is more free on the kind of movement pattern that the drone should implement. However, the hard constraint is that the drone should have recorded at least a certain number of topographic values in each cell of a pre-defined grid.

3.4.5 Boundaries finder and wifi strength tracking

The last part of the project will allow to obtain a truly automatic drone.

1. *Boundaries finder.* Think about a way for the drone to find and adapt by itself the boundaries of the region to explore

2. *Wifi strength tracking.* A specific type of callback called `wifiSignalChangedCallback` gives you the current strength of the WIFI connection. If you think about it, this can be hacked to give you the current location of the computer controling the drone. Implement a *kamikaze* mode in your state machine where the drone turns against the person that is supposed to control it.

3.4.6 (Optional) Real brain

You might have noticed the *brain* folder inside the original starter code that we provided. This folder contains a code being able to perform image recognition with a Deep Convolutional Neural Network (DCNN) from a complete image in MATLAB. For the most apocalyptic effect, try to connect this code to the drone, so that it can even be able to perform person recognition in live through the use of its video feed.

3.5 Tying everything together

Your final project should be a global program handling all parts of the drone together (you can implement as many sub-operators as you want). Compile the overall project along with your SCADE models that are exported to C code prior to this. Summarize everything into a convenient Makefile and test your production.

4 Deadline

The source code, SCADE models, compiled files and a 2-page report on your solution choices must be sent before February 25th to **esling@ircam.fr** and **agonc@ircam.fr**