

# Coax – software

## Contents

### 1 Introduction

- 1.1 Typical program 1: basic monitoring
- 1.2 Typical program 2: basic control
- 1.3 Typical program 3: advanced monitoring
- 1.4 Typical program 4: advanced remote control

### 2 General Remarks

- 2.1 Navigation mode
- 2.2 struct SBControlContext
- 2.3 struct SBApiSimpleContext
- 2.4 Return values

### 3 The simplified API

- 3.1 sbSimpleDefaultContext
- 3.2 sbSimpleInitialise
- 3.3 sbSimpleReachNavState
- 3.4 sbSimpleTerminate
- 3.5 sbSimpleWaitState
- 3.6 sbSimpleControl
- 3.7 sbSimpleRawControl

### 4 Further informations

- 4.1 Complete API
- 4.2 Code sample

## Introduction

This document proposes an API for an easy to use CoaX helicopter. The application context here is to have a host computer or the embedded gumstix communicating with the micro-helicopter to send it high level commands. The communication channels are:

- Serial line between embedded gumstix and micro-controller (reliable, low latency)
- Bluetooth serial line between host PC and micro-controller (reliable up to 2 metres, variable latency)
- UDP over WIFI, between host PC and embedded gumstix, the latter forwarding the commands to the micro-controller on the direct serial line (long-range, variable latency)

Most importantly, this API has been written from a user point of view: it shows what I think a user would expect. The 4 points below shows the typical use-cases considered when developing this API.

### ***Typical program 1: basic monitoring***

```
Connect
Do forever {
    Request helicopter state
    Display helicopter state
}
```

### ***Typical program 2: basic control***

```
Connect
Do forever {
    Request helicopter state
    Compute control using state
    Send control commands
}
```

### ***Typical program 3: advanced monitoring***

```
Connect
Configure communication channels for continous streaming
Do forever {
    Receive helicopter state
}
```

### ***Typical program 4: advanced remote control***

```
Connect
Configure communication channels for continuous streaming
Do forever {
    Receive helicopter state
    Compute control
    Send control commands
}
```

## General Remarks

The API provides 3 layers of increasing complexity:

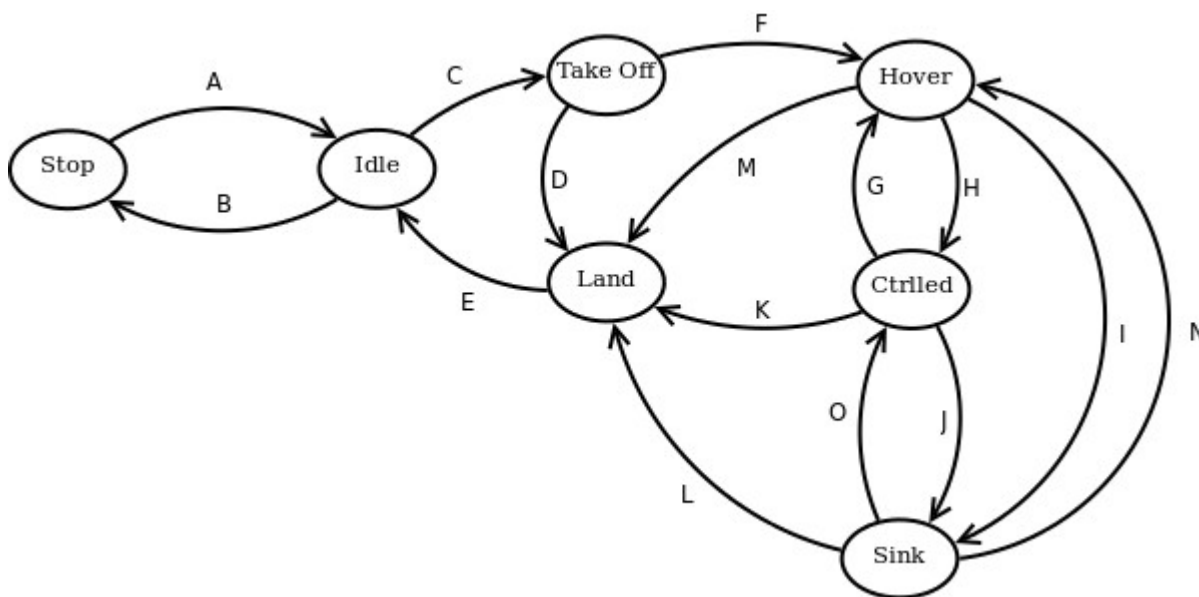
1. Simplified, defined in sbsimple.h provides a default initialisation function, and only the basic function for doing sensor-based servoing.
2. Complete, defined in sbapi.h provides access to all the helicopter configuration functions and controls, including raw commands.
3. Raw communication channels, defined in sbchannel.h, used to send message directly to the helicopter.

## Navigation mode

The helicopter has 8 navigation modes:

- SB NAV STOP: the helicopter is stopped, on the ground, rotors stopped.
- SB NAV IDLE: the helicopter is on the ground, rotors rotating but not enough for taking-off.
- SB NAV TAKEOFF: the helicopter takes off, and transition to SB NAV HOVER when take-off succeeded.
- SB NAV HOVER: the helicopter stays at constant height and yaw angle, with no pitch or roll. No horizontal stabilisation is active. If a watchdog is active, and no sbSetNavMode request is received after this timeout, the helicopter transition to SB NAV SINK.
- SB NAV LAND: the helicopter comes to the ground, and transition to SB NAV IDLE automatically after touch down.
- SB NAV CTRLLED: the helicopter is ready to receive user commands using sbSetControl or sbSimpleReachState. If a control timeout is active, and no control is received after this timeout, then the helicopter goes back to SB NAV HOVER.
- SB\_NAV\_SINK: the helicopter slowly goes down until shutdown. This is an error mode, and can only be reached when a timeout is triggered while flying.
- SB\_NAV\_RAW: the helicopter accept raw commands (servo and motor).

The transition between modes follows the following diagram:



## ***struct SBControlContext***

This C structure contains all the information relative to the communication channel and the reception of data. It should be used as an abstract type, without relying on a specific implementation of its content.

All functions in complete API take as arguments a pointer on a SBControlContext struct. Initialisation functions fill in the required contents.

## ***struct SBApiSimpleContext***

This C structure contains an instance of SBControlContext, plus numerous useful variables to control the execution of a basic control loop. In particular, it stores information about the data reception thread used to listen for helicopter messages.

An important object in the SBApiSimpleContext structure is 'state'. This is updated each time a new data packet is received from the helicopter.

All functions in the simplified API take as arguments a pointer on a SBApiSimpleContext struct. The initialisation functions fill in the required contents.

## ***Return values***

All the functions of this API return 0 on success and anything else on failure. Error codes are not fully documented yet, but should follow the unix errno semantic whenever possible.

## **The simplified API**

### ***sbSimpleDefaultContext***

```
int sbSimpleDefaultContext(SBApiSimpleContext *context);
```

Initialise the context with default values. Should be called before any use of the context variable

### ***sbSimpleInitialise***

```
int sbSimpleInitialise(SBApiSimpleContext *context);
```

Initialise the communication, the helicopter and possibly the state reception thread, based on the information in context. All configuration options are only read during this stage.

### ***sbSimpleReachNavState***

```
int sbSimpleReachNavState(SBApiSimpleContext *context, int desired,  
double timeout_sec);
```

Reach the desired navigation state within timeout sec. If several intermediary state are required, they will be reached one by one.

### ***sbSimpleTerminate***

```
int sbSimpleTerminate(SBApiSimpleContext *context);
```

Terminate the connection to the helicopter and bring it to a safe configuration (landed, motor stopped)

### ***sbSimpleWaitState***

```
int sbSimpleWaitState(SBApiSimpleContext *context, SBHeliState *state,  
double timeout_sec);
```

Wait for a new state to be received. Only make sense in continuous communication mode. If state is not null, the received state is copied inside. If it is null, it is only copied in the internal storage, and in context->stateP pointer if not null

### ***sbSimpleControl***

```
int sbSimpleControl(SBApiSimpleContext *context, double roll, double  
pitch, double yaw, double altitude);
```

Send a control command to the helicopter. The meaning of the command depends on the control mode configuration at initialisation, if not changed with sbConfigureControl

### ***sbSimpleRawControl***

```
int sbSimpleRawControl(SBApiSimpleContext *context, double motor1,  
double motor2, double servol, double servo2);
```

Send a raw control command to the helicopter. Only work in RAW COMMAND control mode.