Flexy^2 Air v.2.0 User Manual

Document version 1.0.0

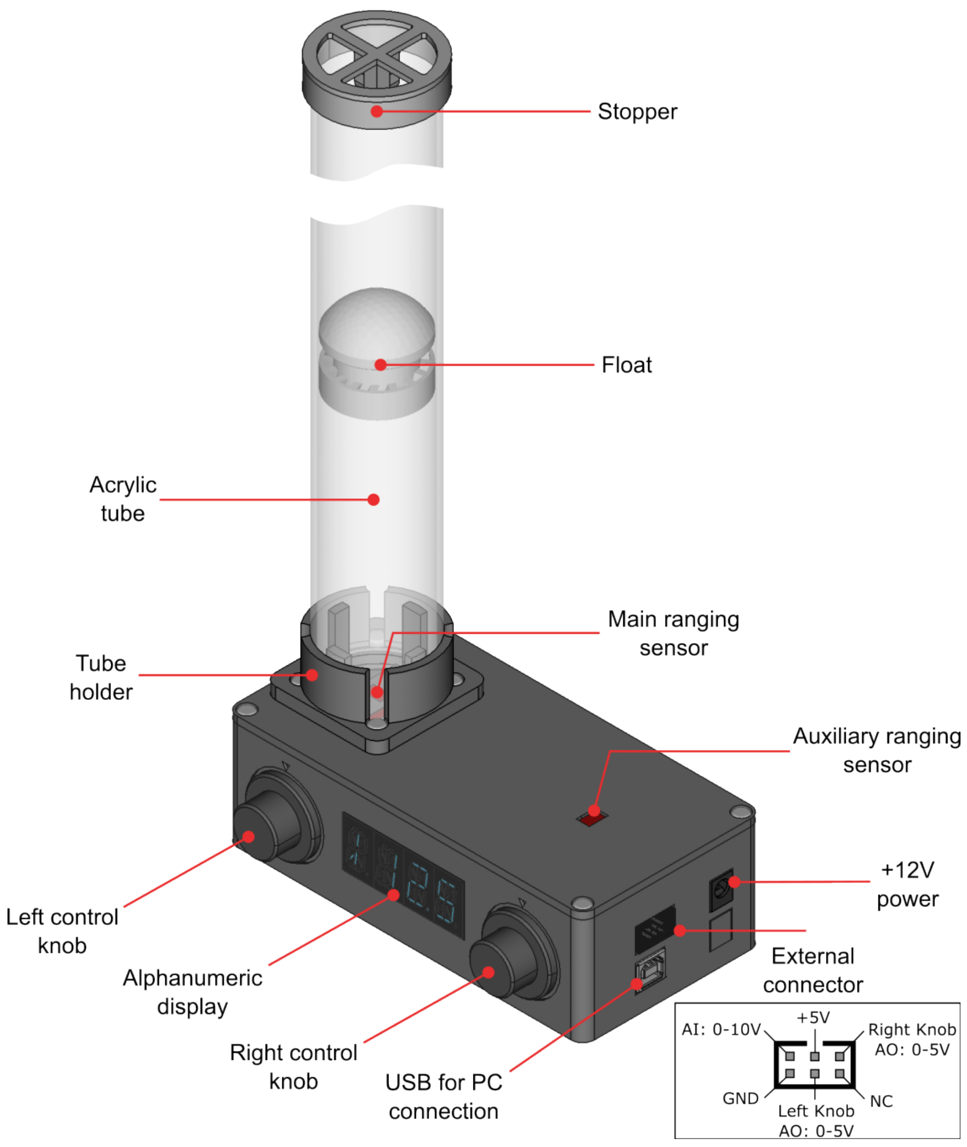
Optimal Control Labs

January 20, 2022

Contents

1 Device Description and Features

1.1 Device Overview



1.2 Dynamical Properties

Flexy^2 Air levitates a float in a tube, while a lift distance can be measured and controlled. Control is provided by a fan, located at the base of the device. The physical principle is quite simple. The fan creates a controlled flow of air that enters the tube. When air hits the float, it creates a high pressure area, which induces a force at the bottom flat area of the float. When this aerodynamic pressure force overcomes the gravity, the float is lifted. Since the tube has the same diameter along its whole length (unlike rotameters that are conical), the pressure drop is caused only by a friction between air and tube surface. This pressure drop slightly increases with the distance between float and fan, which makes the Flexy^2 Air a dynamical process with near-integrator behavior.

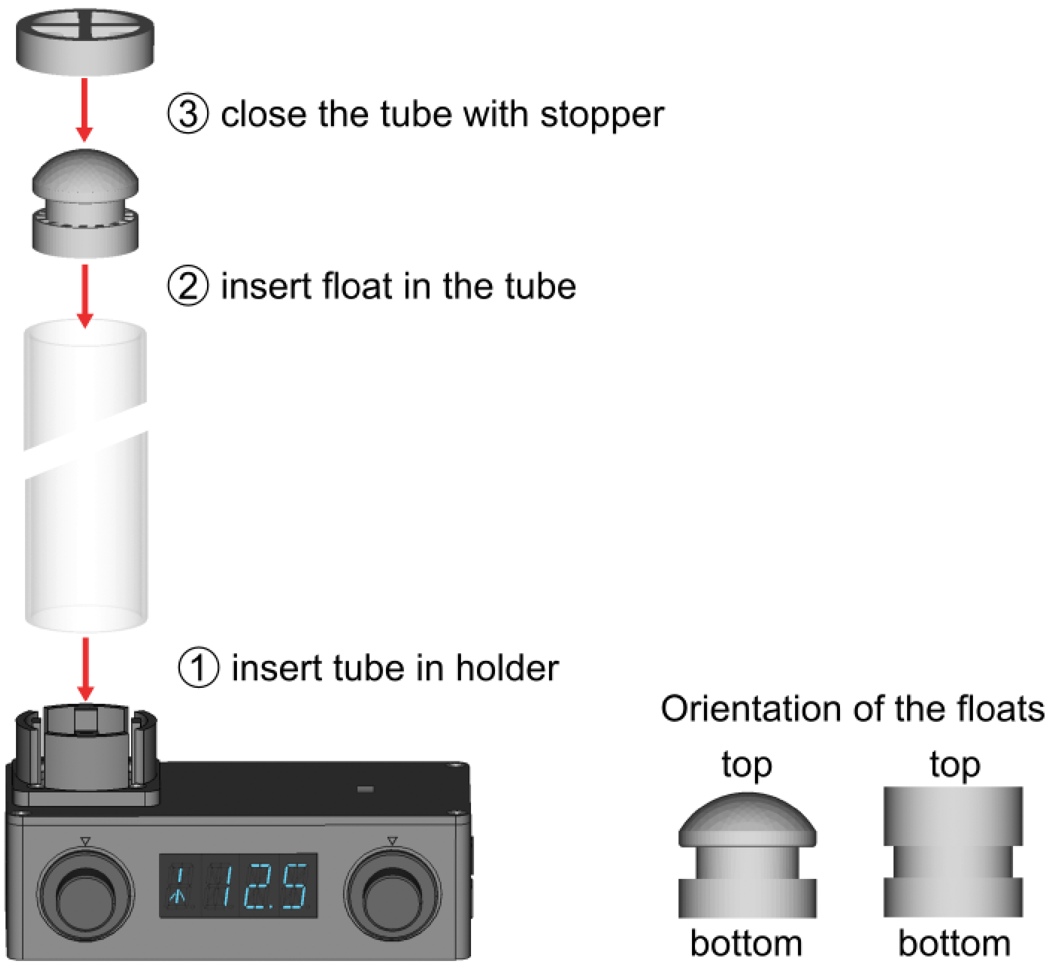
The two knobs on the front-facing side of the device are general purpose inputs and can be used in user-defined program for any task (e.g. controlling the fan, setting control reference, tuning parameters, etc.).

2 Specifications Summary

Interface	UART over USB (Serial port 115200/8-N-1)
Electronics Power Supply	+5VDC via USB (Type A to Type B)
Fan Power Supply	+12VDC, 1A via adapter, 2.1/5.5mm connector
Display	4-digit 14-segment alpha-numeric (I^2C)
Ranging Sensors	2x VL6180X ToF sensor (I^2C)
Control Knobs	2x 270° 10kΩ potentiometer
Fan Specification	+12VDC, 4W, PWN Control
Device Dimensions	15 x 9 x 7.5 cm
Tube Dimensions	length: 41 cm, diameter: 36/40 mm

3 Getting Started

3.1 Assembly


③ close the tube with stopper
② insert float in the tube
① insert tube in holder
125
Orientation of the floats
top
top
bottom
bottom

***Notice**: The interior of the tube is coated by a silicon coating to reduce the friction between the tube and float. This coating wears out after some time. If the float visibly rubs against the tube's wall, it is necessary to reintroduce the coating. This is done by applying a silicon/teflon spray on a piece of cloth or paper tissue and swabbing the tube from inside.*

3.2 Power Delivery

Flexy^2 Air uses two power sources to operate. Internal electronic is is powered via USB (+5V). Integrated fan uses separate voltage supply provided by 12V power adapter (included).
Notice: *When using a different adapter or power supply that the one included in the box, make sure that it meets required specification: +12V DC, 1A, 2.1/5.5mm barrel connector (**center positive!**).*
Notice: *Using a power adapter other than specified may damage the device.*

3.3 Connection to PC and Installation

The operating system of computer that device is connected to must have installed an USB-to-serial driver for the device to function properly. Use provided USB cable to connect the Flexy^2 Air to the USB Type A port on a host computer. Before using the device make sure that also the 12V power adapter is connected to a barrel connector.

3.3.1 Windows

After first connection to a Windows computer, the device should be recognized automatically and Windows Update will install the driver. The device will be listed in the *Device Manager* as *Arduino UNO R3* and will be sitting on a specific COM port. If the device is not recognized and installed automatically, the best way to install driver is to download and install Arduino IDE that comes with the driver by default.

3.3.2 Linux and Mac

Most of the Linux distributions and OSX should recognize the device as a file in `/dev` directory.

A comprehensive guide how to install USB driver can be found in this tutorial by Sparkfun.

3.4 General Communication Interface

3.4.1 Serial Communication: UART over USB

Flexy^2 Air uses a UART serial link to communicate with computer over USB. Communication encoding is ASCII. Messages sent from Flexy^2 Air device to computer are terminated by CR/LF sequence (new line characters). Commands sent from the computer to the device do not require CR/LF termination. Communication with the device can be established with any program or programming language that supports serial reading/writing to port or file (basically any high level language, such as Python, C/C++, Java, JavaScript (node.js), MALTAB, Octave, LabView, etc.).

To establish successful connection, following serial port settings must be used.

BaudRate: 115200

Data bits: 8

Stop bits: 1

Parity: No

3.4.2 Data Format

Data format used by Flexy^2 Air for serial communication in both direction is as follows.

Commands

Notice: *all values are in integer format*.

`<P:1>` or `<P:0>` - Turns on the stream of data from device (option `1`) or turns it off (option `0`). This parameter is `0` by default and in order to get data from device it must be set to `1`.

`<S:5>` to `<S:255>` - Sets a sampling rate (frequency) of measured data stream. For example by sending command `<S:100>` the device will send new data every 10 milliseconds (100 times per second).

`<F:0>` to `<F:255>` - Sets a speed command for the fan. The value 255 is maximum speed.

`<L:W>` - Sets a simple exponential smoothing filter for sensor data. `W` is a weighting factor between current and previous measurement in percent (`0` - `100`). The weighting is calculated as y(k) = (1-W)*x(k)+W*x(k-1).
Examples:

`<L:0>` (W = 0) - No smoothing (just current measurements)

- `<L:10>` (W = 10) - Minor smoothing
- `<L:50>` (W = 50) - Mediate smoothing
- `<L:90>` (W = 90) - Heavy smoothing
- `<L:100>` (W = 100) - Current measurements are lost (this setting is not useful at all)

`<N:0>` to `<N:100>` - Adds artificial random noise to the sensor signal. Value `0` preserves original signal, value `100` adds noise with maximum amplitude to the signal.

`<D:N>` - Adds artificial measurement delay of `N` samples of internal sampling frequency. For value of `N` = `0`, no delay is present. Maximum value of `N` is `100` samples.

`<V:1>` - Requests a version of firmware. Device will respond with `<V:VERSION_STRING>`.

Measurements

***Notice**: Measured data are streamed automatically by device with the frequency set by `S`-command (10Hz by default) and only if `P`-command was set to `1`.*
***Notice**: Measurements from distance sensors are in millimeters. All other measurements are in percent.*

Format of data measurement message is:

```
<D:Distance1,Distance2,UserInputL,UserInputR,TerminalInput>
```

where:

- `Distance1` - distance measurement from main ranging sensor (distance of the float from zero reference)
- `Distance2` - distance measurement from auxiliary ranging sensor
- `UserInputL` - signal from left user input (knob) in percent
- `UserInputR` - signal from right user input (knob) in percent
- `TerminalInput` - voltage level (in percent, representing 0-10V) of a signal connected to the external IDC connector (IDC pin 1)

3.5 MATLAB and Simulink

MATLAB library for Flexy^2 Air comes with the following files.

```
Flexy2_Air
├─lib
│  ├─flexy_air_lib.slx
│  ├─flexy_air_blank.slx
│  └─flexy_air_fun.m
├─models
│  ├─flexy_air_man.slx
│  ├─flexy_air_pid.slx
│  ├─flexy_air_pid_hand_tracking.slx
│  ├─flexy_air_pid_init.slx
│  └─flexy_air_pid_tune.slx
├─detectFlexyAirPort.m
├─FlexyAir.m
└─runme.m
```

The main file in the root directory is `FlexyAir.m` - a class file that can be used to directly control the device via commands (see the following section). Function `Port = detectFlexyPort()` will automatically detect a communication port on a host computer (currently this only works in Windows systems). Script `runme.m` contains examples of usage for both command line and Simulink.

3.5.1 Command-based usage (Command Window, M-files, Scripts)

Notice: *To explore all available commands run* `help FlexyAir`.

Use a following command to create an instance variable of Flexy^2 Air

```
flexy_air = FlexyAir('[PORT]')
```

The `[PORT]` is a character string that represents the name of communication port that host OS detected after connecting the device. On Windows, the port is in the form `COMX`, where `X` is a number (may vary, depending on other connected serial devices). Port can be automatically detected on

Windows either by running `Port = detectFlexyAirPort()` or by looking into *Device Manager*, under *Ports (COM & LPT)*, where device called *Arduino UNO R3 (COMX)* should be listed.

The following commands (with example values) are implemented to set-up and control the device. ***Notice***: *All commands use a dot referencing in the form* `instance.method`.

`flexy_air.setInternalSamplingFreq(25)` - sets an internal sampling rate for the device. In this case the device will sample the readings from sensors and send the data to computer with 25Hz frequency (every 40ms).

`flexy_air.setDisplayStateOnOff(0)` - sends a command to turn off the display of the device. This action is persistent even after power cycle. To turn the display on again, just send the same command with option 1.

`flexy_air.dataFlowOnOff(1)` - turns on the data stream from device to computer. To turn it off, use option 0. This command is called automatically with option 1 upon instance creation.

`flexy_air.setVerboseModeOnOff(1)` - turns on the data stream print-out to command window. This is for debugging purposes. Use option 0 to turn it off again.

`flexy_air.setArtificialNoise(25)` - introduces an artificial noise into main sensor measurement of 25% of max. Signal-to-Noise ratio (SNR = 4).

`flexy_air.setFilter(0.9)` - initializes a an exponential smoothing filter: $y(k) = (1 - W) * x(k) + W * x(k - 1)$, where the argument is W is a weighting factor between current and previous measurement. In the case of $W = 0.9$ a heavy smoothing is applied.

`flexy_air.off()` - turns off the device's fan.

`flexy_air.close()` - terminates the connection with the device.

`flexy_air.setGracefulShutdown(1)` - sets the power down of device's sensors after the connection is closed. It is recommended to use this command with OPT=1 for Simulink-based control. This command should be send before the `close` command.

`flexy_air.setFanSpeedPerc(75)` - sets the power of fan to 75%.

`flexy_air.getSensor1DistanceMm()` or `.getSensor1DistanceCm()` - returns a distance measurement in millimeters or centimeters from main sensor. Example:

```
>> distance1_mm = flexy_air.getSensor1DistanceMm()

distance1_mm =

       271

>> distance1_cm = flexy_air.getSensor1DistanceCm()

distance1_cm =

       27.1000
```

`flexy_air.getSensor2DistanceMm()` or `.getSensor2DistanceCm()` - returns a distance measurement in millimeters or centimeters from auxiliary sensor.

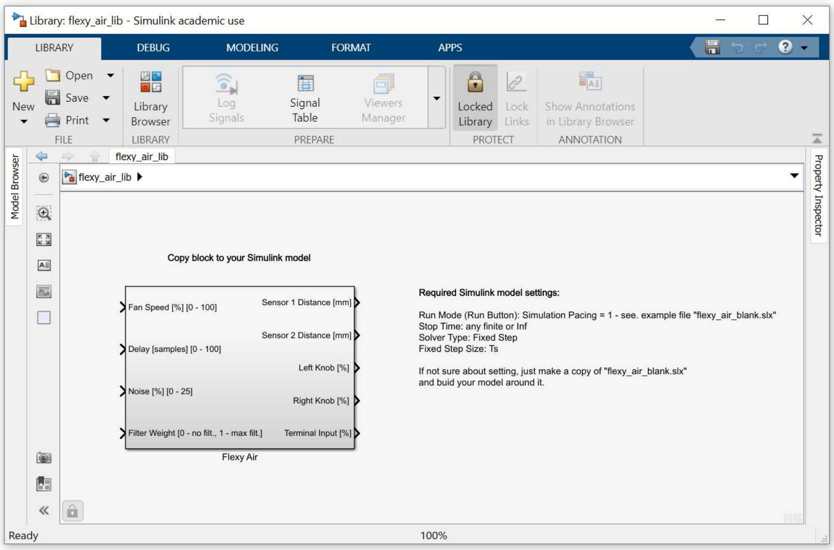
`flexy_air.getTerminalInputPerc()` - returns the voltage level (in percent of 0-10V) of a signal connected to input terminal's analog input (IDC pin 1). This pin can be used to connect additional analog sensor.

`flexy_air.getUserInputLPerc()` or `.getUserInputRPerc()` - returns a turn ratio (in percent) of left or right hand knob. Example:

```
>> knob_left = flexy_air.getUserInputLPerc() % turned half way

knob_left =

       50

>> knob_right = flexy_air.getUserInputRPerc() % fully turned

knob_right =

       100
```

3.5.2 Simulink usage

Flexy^2 Air comes with Simulink library `lib/flexy_air_lib.slx` (fig. 1) that contains interface block that represents the device. The library also contains a blank model with pre-configured options `flexy_air_blank.slx`.



Copy block to your Simulink model

Required Simulink model settings:

Run Mode (Run Button): Simulation Pacing = 1 - see. example file "flexy_air_blank.slx"
Stop Time: any finite or Inf
Solver Type: Fixed Step
Fixed Step Size: Ts

If not sure about setting, just make a copy of "flexy_air_blank.slx"
and buid your model around it.

Figure 1: Simulink Library for Flexy^2 Air

To create a custom Flexy^2 Air Simulink model, just make a copy of this file and build you model around it. Alternatively, to make a model from scratch, copy the `Flexy Air` block from library to the model and configure the following model settings.

`Run Mode:` Simulation Pacing = 1 - see. example file `flexy_air_blank.slx`

`Stop Time:` any finite or Inf (model will run in soft real time)

`Solver:` Any that fits the need (auto by default)

`Solver Type:` Fixed Step

`Fixed Step Size:` Ts (it is recommended to run the model on the same sampling as the device)

To use the `Flexy Air` Simulink block, three variables must be defined in the global workspace of MATLAB. These are `Port` - communication port where device resides, `Ts` - sampling period for both Simulink model and device, and `graceful_shutdown_opt = 1`. The best way to initiate parameters and model are by running script `runme.m`. A specific example Simulink model can be opened by uncommenting corresponding line in section `Simulink models`.

Notice*: All folders and subfolders of provided library must be listed in the MATLAB path. This can be done permanently via* `Set Path` *button located* *on `Environment` panel in `Home` tab of MATLAB. Alternatively, the path can be set via command located at the top of `runme.m` file.*

Directory `models` contain several example Simulink models:

- `flexy_air_man.slx` - provides manual control over fan using right knob with possibility to add noise via left knob.
- `flexy_air_pid.slx` - provides simple PID control scenario.
- `flexy_air_pid_init.slx` - PID control scenario with initialization (the setpoint is reached faster after start of the model).
- `flexy_air_hand_tracking.slx` - PID control scenario with control setpoint taken from auxiliary sensor (e.g. by hovering the hand over it).
- `flexy_air_pid_tune.slx` - PID control scenario that demonstrates possibility to live tune the controller parameters using knobs.