DARC Workshop

Drones & Aerial Robotics Conference

October 11–13, 2013 | NYU/ITP | 721 Broadway, New York, NY 10003

We will be presenting the ODC framework at the Hackathon hosted by Nodecopter on **Sunday, October 13th.** Check out the conference schedule (https://droneconference.org/schedule/) and hackathon schedule (http://nodecopter.com/2013/new-york/oct-13). The hackathon is open for free registration here: http://www.eventbrite.com/event/8293334587 (http://www.eventbrite.com/event/8293334587).

Follow the conference and ODC at @droneconference (https://twitter.com/droneconference), @opendronectrl (https://twitter.com/opendronectrl).

Workshop Schedule

- Introductions (transLAB and the Media Arts and Technology program at UCSB)
- What is ODC?
- Example Code
- Lets make stuff!!

What is ODC?

OpenDroneControl [ODC] is an open source software platform for developing interactive artworks and research projects with aerial robotics. ODC was developed to be a community-supported framework for connecting commercially available quadcopter platforms to a common programming interface. The framework provides access to platform specific sensors and optionally allows for additional functionality such as navigation and tracking.

ODC is compatible with creative coding software such as Processing, Max, and open Frameworks (oF). Additionally, ODC is designed for expansion through code modules, third party peripherals, integration with emerging aerial robotic platforms, and user developed applications. Application developers can utilize this common interface to easily target multiple drone platforms without redesigning their code which allows for rapid project development.

The development of this project was initiated by the Transvergent Research Group, including Media Art and Technology graduate student researchers Tim Wood, Sterling Crispin, and RJ Duran. The technology was developed under the direction of Professor Marcos Novak, at the transLAB, housed within the California

NanoSystems Institute (CNSI) at the University of California Santa Barbara. ODC originated from experiments within the transLAB's motion tracking environment using a Parrot AR.Drone at the beginning of 2012. Initially the focus was to create an external Max object which could algorithmically control the drone using available spatial information provided by the OptiTrack system.

Preparation

Hardware and software requirements of ODC

- Mac OSX 10.6.8 – 10.8 or Windows 7 with WiFi Card
- Java 6 or greater
- Parrot AR.Drone 1.0 or 2.0
- Project template for desired coding environment

Processing Users

- Processing (http://www.processing.org)
- Download ODC Processing template project (http://opendronecontrol.org/downloads/ODC_ProcessingTemplate.zip)
- oscP5 Library (http://www.sojamo.de/libraries/oscP5/) (Optional)

Max Users

- Max 6 (http://www.cycling74.com/)
- Download ODC Max template project (OSX) (http://opendronecontrol.org/downloads/ODC_MaxTemplate_osx.zip)
- Download ODC Max template project (other platforms) (http://opendronecontrol.org/downloads/ODC_MaxTemplate_no_xuggle.zip)

openFrameworks Users

- DroneOSC Scala Example via github source
- openFrameworks v0073 or newer (http://www.openframeworks.cc/)
- XCode (https://developer.apple.com/xcode/)

Optional (Advanced)

- Download source from github (https://github.com/opendronecontrol/odc)
- If you need a text editor, try Sublime Text 2 (http://www.sublimetext.com/2)
- Any OSC enabled device or application
- Experience using Java and Scala and your Terminal

Documentation

The java/scala docs can be found at http://opendronecontrol.org/docs (http://opendronecontrol.org/docs). To generate documentation from the code, open a terminal and navigate to your odc directory then execute the following command:

```
./sbt doc
```

The generated documentation can be found at: odc/odc/target/scala-2.10/api/index.html and odc/platforms/ardrone/target/scala-2.10/api/index.html

Examples

The examples can be found in the odc/examples directory with a .scala extension. Examples for other software such as Processing can be found in their own directories. For all the examples you will need to be connected over WiFi to an AR.Drone quadcopter. The device typically shows up as “ardrone_288520″ or something similar.

Processing Examples

Drone video:odc/examples/ODC_DroneVideo/ODC_DroneVideo.pde

Displays live video feed, draws a graphic based on sensor data and pilots the drone with the position of the users mouse.

3D physics controlling drone flight: odc/examples/ODC_BouncingDrone/ODC_BouncingDrone.pde

A sphere pulled downward by gravity bounces upon impact and the drone attempts to mimic its behavior

Communicate with OSC: odc/examples/ODC_DroneOscP5/ODC_DroneOscP5.pde

Uses the Open Sound Control library (oscP5) to send and receive messages over OSC to an ODC client that is listening for messages such as the DroneOSC example scala application or another Processing sketch that is directly connected to the drone. This lets you use multiple devices to receive navdata from and send control messages to a single drone. If you don’t have oscP5 installed, open Processing, create a new sketch, then navigate to Sketch > Import Library > Add Library > oscP5.

Scala Examples

- DroneOSC.scala — Creates an AR.Drone client and start listening for OSC messages.
- GetSensorData.scala — Read sensor data from the hardware.
- TakeoffLand.scala — Tell the device to take off, hover for 10 seconds, then land.
- TakeoffMoveLand.scala — Tell the device to take off, oscillate the drone left/right, then land.

The standalone examples are designed to be run from inside the root directory (opendronecontrol/odc) using Scala and sbt. To run, make sure you are connected to the AR.Drone, then execute the following command in terminal:

```
./sbt "project examples" run
```

This executes a script that downloads a local copy of sbt for running the examples, changes the current project, then compiles and runs available classes in the examples directory. You will be prompted to select the example to run from a list of options.

If you are using ARDrone 1.0 or 2.0 you have a default IP address of 192.168.1.1. ODC uses this as the default address but you can edit it when you make a new ARDrone object as shown below. For this example you can find it in the source code of examples/DroneOSC.scala.

```
var drone = new ARDrone("192.168.1.1")
```

DroneOSC

Navigate to the odc root directory and run examples using:

```
./sbt "project examples" run
```

Select the correct number for org.opendronecontrol.examples.DroneOSC. This will connect your computer (the client) to the ARDrone device and enable the OSC interface. Essentially, this interface allows you to send OSC messages to the drone and receiving data from the drone.

```
Multiple main classes detected, select one to run:
[1] org.opendronecontrol.examples.TakeoffMoveLand
[2] org.opendronecontrol.examples.GetSensorData
[3] org.opendronecontrol.examples.DroneOSC
[4] org.opendronecontrol.examples.TakeoffLand
Enter number: 3

[info] Running org.opendronecontrol.examples.DroneOSC
Listening for drone commands on port 8000

Example Commands:
  /connect
  /sendSensors 192.168.1.255 8001
  /takeOff
  /move 0.1 0.0 0.0 0.0
  /land
  /disconnect
```

You can send command messages from any OSC enabled device, for example an iPhone, iPad, or Android device running an OSC application like TouchOSC. You could also send OSC messages using a software based application such as Processing (as seen in Example 3).

At this point your client (computer connected to the ARDrone) is able to pass OSC messages to the drone. Further OSC messages can be found in the docs: http://opendronecontrol.org/docs/#org.opendronecontrol.net.OSCInterface (http://opendronecontrol.org/docs/#org.opendronecontrol.net.OSCInterface).

```
/connect
/sendSensors "192.168.1.255" 8001
/takeOff
/move 0.0 0.0 0.0 0.5
/config "maxEulerAngle" 0.2
```

```
/config "maxVerticalSpeed" 1.0
/led [pattern:Int] [frequency:Float] [duration(seconds):Int]
/animation [pattern:Int] [duration(seconds):Int]
/land
/disconnect
```

GetSensorData

Navigate to the odc root directory and run examples using:

```
./sbt "project examples" run
```

Select the org.opendronecontrol.examples.GetSensorData option.

This example first connects to the drone then proceeds to read the available sensor data. There are two options to reading the data:

Option 1: Set a callback function to be called whenever a sensor is updated

```
drone.sensors.bind( (s) => { println( s.name + ": " + s.value )})
```

Option 2: Get a sensor value on demand

```
var t = 0
var dt = 30
// loop for 20 seconds retrieving sensor values every 30 ms
while( t > 20000){
  if( drone.hasSensors() ) {
    println( "velocity: " + drone.sensors("velocity").getVec3 )
    println( "gyroscope: " + drone.sensors("gyroscope").getVec3 )
    println( "altimeter: " + drone.sensors("altimeter").getFloat )
    println( "battery: " + drone.sensors("battery").getInt )
  }
  t += dt
  Thread.sleep(dt)
}
```

The application then disconnects with the drone.disconnect() command.

TakeoffLand

Navigate to the odc root directory and run examples using:

```
./sbt "project examples" run
```

Select the org.opendronecontrol.examples.TakeoffLand option.

This example first connects to the drone, hovers in place for 10 seconds, then lands. The application then disconnects with the drone.disconnect() command.

Another approach to taking off and landing would be to check if the drone is flying or not. While not flying, take off.

```
while ( !drone.sensors("flying").getBoolean ) {
```

```
  drone.takeOff()
}
```

TakeoffMoveLand

This example does the same thing as example 2.3 but proceeds to oscillate the drone back and forth before disconnecting. It also prints out the sensor data to the Terminal. See the comments in code for further details.

OSC Examples

Running DroneOSC with Processing

Start by running the DroneOSC example to setup the communication with the drone and start listening for OSC messages.

Once this is done, navigate to examples/processing/DroneOscP5 and open up the sketch in Processing 2.0 or greater and run it. You should start seeing gyroscope data streaming from the device into the console.

The example sets up the ports and sends a /connect message followed by a /broadcastSensors 8001 message. The oscEvent(OscMessage theOscMessage) method reads the gyroscope data sent over OSC and prints it out in the console.

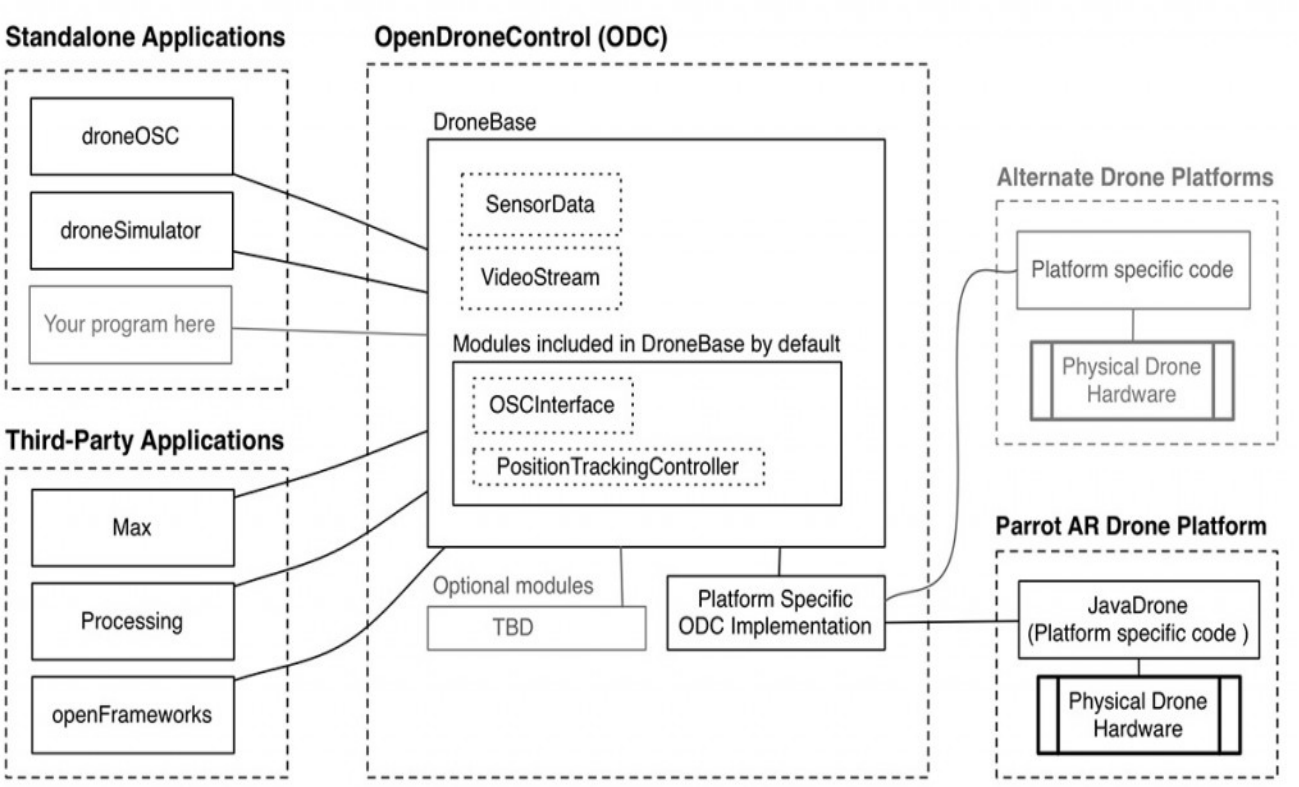
Running DroneOSC with openFrameworks

Start by running the DroneOSC example to setup the communication with the drone and start listening for OSC messages.

Once this is done, move the oF project odc/examples/openframeworks/ODC_SendRecvOsc to the openFrameworks apps/myApps/ directory. Refer to this if you need help with anything related to oF file conventions, http://www.openframeworks.cc/tutorials/introduction/001_chapter1.html (http://www.openframeworks.cc/tutorials/introduction/001_chapter1.html).

The example sets up a simple channel between oF and DroneOSC for sending/receiving OSC commands. Port 8000 is used to send from oF on the localhost and port 8001 receives messages from the drone via DroneOSC.

Framework Overview



(http://opendronecontrol.org/media/ODC_framework.jpg)

Future Work

- Object Tracking
- Computer vision
- GPS
- Hardware integration
- Sensing
- Environmental monitoring
- Lifting, pushing, and pulling of objects
- Lighting, speaker, microphone
- Augmented bodies
- Your ideas!

Modified: December 16, 2014 at 4:19 am

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

Comment

Submit Comment

Website built using the Roots Framework (http://roots.io/) and Wordpress (http://wordpress.org).