User-guide: DUNE source code

Jostein Borgen Moe

June 13, 2016

Digital Appendix – Master thesis's
Department of Engineering Cybernetics
Norwegian University of Science and Technology

Table of Contents

Chapter 1

DUNE framework

Here follows examples on how to configure a recovery with explanations (vehicles names are for illustration only). Slave vehicle(s) is optional, but must be specified in the global configuration.

1.1 Main

Each vehicle in the system has a global configuration file, there exists a `uav/netrecovery/fixedwing.ini`, `uav/netrecovery/master.ini` and `uav/netrecovery/slave.ini` and should be used as follows.

1.1.1 Fixed-Wing

```
# Fixed-Wing, ntnu-x8-002.ini

[Require uav/arduplane.ini]
[Require uav/netrecovery/fixedwing.ini]

[General]
Vehicle                                 = ntnu-x8-002

[Control.UAV.Ardupilot/AP-SIL]
TCP - Port                              = 5760 # Instance 0
```

1.1.2 Multirotor (Master)

```
# Multirotor (Master), ntnu-hexa-004.ini
[Require uav/arducopter.ini]
[Require uav/netrecovery/master.ini]

[General]
Vehicle                                 = ntnu-hexa-004

[Control.UAV.Ardupilot/AP-SIL]
TCP - Port                              = 5770 # Instance 1
```

1.1.3 Multirotor (Slave)

```
#Multirotor (Slave), ntnu-hexa-003.ini
[Require uav/arducopter.ini]
[Require uav/netrecovery/slave.ini]

[General]
Vehicle                                 = ntnu-hexa-003

[Control.UAV.Ardupilot/AP-SIL]
TCP - Port                              = 5780 # Instance 2
```

How to configure the TCP ports for the Ardupilot layer will be explained in detail in Chapter 2.

1.2 Transport

In order for the vehicles to communicate the different transport layers must be specified between them. Each layer specifies one or more messages `<messages>` to send with an option to send only messages from a task with a given `<entity>`. Each transport-layer is connected to a discovery tasks where the desired intervehicle can be specified. Then this task will configure the transport-layer, such that all messages are sent inbetween the vehicles.

For each layer the following configuration must be specified:

```
# uav/netrecovery/<vehicle type>_transport.ini

[Transports.UDP/<name>]
Enabled             = Always
Entity Label        = <label>
Transports          = <messages>:<entity>
Local Port          = <local port> # which local port to listen to
Dynamic Nodes       = False
Local Messages Only = True
Announce Service    = True
Custom Service      = <custom_service>

[Transports.DiscoverVehicle/<name>]
Enabled              = Always
Entity Label         = <label>
Entity Label To Cons = <udp_task_entity>
Vehicles             = <vehicle1>,<vehicle2>,...
                     = imc+udp+<custom_service>
```

The Custom Service in the UDP task will ensure that this task dispatches a new service in order to avoid conflict with the existing framework. Then the DiscoverVehicle task can consume the IMC::Announce message from all vehicles in the vehicle list with the custom service active. This message gives the IP address and port for the desired transport layer for each destination vehicle. Furthermore, the DiscoverVehicle configures the local transport layer (on the current vehicle) by setting the Static Destination string. The local port in the UDP task is not necessary to specify, as it would search for an available port (however, by setting a port, you are aware of which port it will try binding to initially). For the setup in this Master's thesis, two transport layers were considered.

Firstly the *Common transport* layer defines all common messages dispatched between all multirotors and the fixed wing, this is the global configuration message (IMC::CoordConfig) and the states of all vehicles in the same reference-system (IMC::EstimatedLocalState). To ensure only the vehicle state, and not the local centroid message is dispatched, one must filter on the Entity Label for the task producing the vehicle state in the new common reference system. Further, a transport-layer for the formation control in between the multirotors where defined. The messages are summarized in Table 1.1

	Transports	Filtered entities
Formation	DesiredLinearState	Desired Linear Formation Path Control+ Desired Linear Net-Catch Coordinator
	DesiredHeading	Desired Heading Formation Path Control+ Desired Heading Net-Catch Coordinator
	FormPos,FormCoord	
Common	EstimatedLocalState	Local State (Local State Producer)
	CoordConfig	

Table 1.1: Transport layers (do not use vehicle names here)

The source of each local message, the distribution of the tasks and the computation power is discussed in more detail in the implementation chapter of the thesis. This guide mainly explain how to configure the DUNE program.

1.3 Common

The common layer defines the tasks running on all vehicles in the interconnected system. Firstly all vehicles must transform its states to the common reference system, this is done in `[Transports.LocalStateTransport]`, hence, that is why all transport layers tasks must filter on `Local State`. Furthermore, the common transport layer with its corresponding layer is defined.

```
# uav/netrecovery/common.ini
[Transports.LocalStateTransport]
Enabled             = Always
Entity Label        = Local State
```

1.4 Formation

The formation configuration is added to all multirotors and holds the force control containing the formation control as well. `[Control.Formation.Force]` defines the actual force control, for the formation to run the coordinator `[Control.Formation.Coordinator]` must be enabled, this one where not implemented in this thesis, but is necessary for formation control. The centroid states are calculated in `[Transports.Formation.Centroid]` based and the states on all vehicles. The filtration on entity label on desired heading and desired linear is necessary as there exists multiple IMC::DesiredHeading and IMC::DesiredLinearState with different labels in the system (hence, these labels must correspond to labels of the active path controller). Lastly the formation transport layer and the discovery task is set here, such that all multirotors run these (`[Transport.UDP/Formation]` and `[Transport.DiscoverVehicle/..]`).

```
# uav/netrecovery/formation.ini
[Control.Formation.Force]
Entity Label                          = Formation Controller
EstimatedLocalState Entity Label      = Formation Centroid
Desired Heading Entity Labels         = Desired Heading Formation Path Control,
                                        Desired Heading Net-Catch Coordinator
Desired Linear Entity Labels          = Desired Linear Formation Path Control,
                                        Desired Linear Net-Catch Coordinator


[Transports.Formation.Centroid]
Entity Label                          = Formation Centroid
Desired Heading Entity Labels         = Desired Heading Formation Path Control,
                                        Desired Heading Net-Catch Coordinator

[Control.Formation.Coordinator]
Entity Label                          = Formation Coordinator
```

1.5 Summary

In total each header of the vehicles must include the following headers

Master

The master vehicle must also run the tasks as specified below the header, some of the parameter to the tasks are specified. The reader is referred to the source code for the complete parameter lists.

```
# uav/netrecovery/master.ini
[Require common.ini]
[Require formation.ini]

# Recovery coordinator
[Control.NetCatch.Coordinator]
Entity Label                    = Net-Catch Coordinator
# Use the centroid states as feedback
EstimatedLocalState Entity Label = Formation Centroid

# Payload transport
[Control.Formation.Path]
Entity Label                    = Formation Path Control
# Use the centroid states as feedback
EstimatedLocalState Entity Label = Formation Centroid

# Recovery maneuver,
# interacts with the recovery coordinator
[Maneuver.NetRecovery]

# Formation configuration transport layer,
# dispatches the desired formation parameters
[Transports.Formation.Configuration]
Entity Label = Coordination Configuration
Vehicle List = ntnu-hexa-004, ntnu-hexa-003
```

Slave

```
# uav/netrecovery/slave.ini
[Require common.ini]
[Require formation.ini]
```

Fixed-Wing

```
# uav/netrecovery/fixedwing.ini
[Require common.ini]
```

Chapter 2

SITL setup

For the *Software-In-The-Loop* (SITL) setup multiple simulators are required to run simultaneously. Follow the guide on [DIY, 2015c] closely, however some modifications are required. Here a setup for Linux will be explained (based on [DIY, 2015b]), first off all, install all dependencies as required as specified in the guide.

2.1 ArduCopter

Clone the ardupilot repository from the NTNU-UAV-lab server[1] and checkout the `copterdev/3.3` branch. Here the repository is cloned to the folder `~/uavlab`.

```
~/uavlab $ git clone git@uavlab.itk.ntnu.no:uavlab/ardupilot.git
~/uavlab $ cd ardupilot
~/uavlab/ardupilot $ git checkout copterdev/3.3
```

2.2 ArduPlane

For the ArduPlane (APM:Plane) the official APM:Plane code [DIY, 2015a] will be used. The simulator environment will be given by the JSBSim simulator [JSBSim, 2015]. Still one should follow the guide on [DIY, 2015c] with some modifications.

[1] `http://uavlab.itk.ntnu.no:88/` (requires access)

2.2.1 JSBSim

To use the X8 JSBSim simulator, follow the guide on `http://uavlab.itk.ntnu.no:88/uavlab/x8_jsbsim`.

2.2.2 Run the simulators

Now assuming the folders is located on the following structure

```
~/uavlab/ardupilot
~/uavlab/external/jsbsim
~/uavlab/external/ardupilot
```

for short the following abbreviations will be defined

```
UA=~/uavlab/ardupilot
EJ=~/uavlab/external/jsbsim
EA=~/uavlab/external/ardupilot
```

In order to run the simulators one must set the paths correctly. As default the uavlab-ardupilot simulators should be used such that, the following should be added to your `~/.bashrc` file.

```
export PATH=$PATH:$HOME/uavlab/external/jsbsim/src
export PATH=$PATH:$HOME/uavlab/ardupilot/Tools/autotest
export PATH=/usr/lib/ccache:$PATH
```

then source it

```
. ~/.bashrc
```

Then to start a copter-simulation

```
UA/ArduCopter $ sim_vehicle.sh -I<port> -L <location> --console --aircraft <log>
```

`<log>` will be located in the current folder. The location is specified in the `ardupilot/Tools/autotest/locations.txt` with the name corresponding to `<location>`. Lastly the `<port>` gives which port it will listen to in order to connect with DUNE, it is crucial that each simulator runs on separate ports corresponding to the desired DUNE vehicle. `<port>` will be a number from 0 to k were the port-number will be $5760 + 10k$.

Then to start a fixed-wing UAV simulation the path to the new `sim_vehicle.sh` must be given.

```
EA/ArduPlane $ export PATH=$HOME/uavlab/external/ardupilot/Tools/autotest:$PATH
EA/ArduPlane $ sim_vehicle.sh -f jsbsim:X8 -I<port>
               -L <location> --console --aircraft <log>
```

Table 2.1: Assigning ports

Vehicle	Name	Port	Instance
Fixed-Wing	ntnu-x8-002	5760	-I0
Multirotor (Master)	ntnu-hexa-004	5770	-I1
Multirotor (Slave)	ntnu-hexa-003	5780	-I2

2.3 DUNE

Each vehicles configuration file must have the following entry in order to communicate with the others

```
[Control.UAV.Ardupilot/AP-SIL]
TCP - Port                              = <5760+10k>
```

2.4 Example, 2 multirotors - 1 fixed-wing

then, the main file should be set as seen in Section 1.1.

Then to run DUNE

```
~/uavlab/build $ ./dune -c ntnu-hexa-003 -p AP-SIL
~/uavlab/build $ ./dune -c ntnu-hexa-004 -p AP-SIL
~/uavlab/build $ ./dune -c ntnu-x8-002   -p AP-SIL
```

and simulators

```
UA/ArduCopter $ sim_vehicle.sh -I1 -L Breivika --console --aircraft copter
UA/ArduCopter $ sim_vehicle.sh -I2 -L Breivika --console --aircraft copter
EA/ArduPlane  $ sim_vehicle.sh -f jsbsim:X8 -I0 -L Breivika --console --aircraft X8
```

Takeoff copters

```
$ mode guided
$ arm throttle
$ takeoff <height above ground>
$ rc 3 1500
```

then upload and start the desired plan from Neptus.

Takeoff fixed-wing

Upload and start desired plan from Neptus, then, in the fixed-wing UAV simulator terminal

```
$ mode auto
$ arm throttle
```

start the plan again.

Bibliography

DIY (2015a). Copter | Multirotor UAV.
URL https://github.com/ArduPilot/ardupilot.git

DIY (2015b). Sitl setup on linux.
URL http://ardupilot.org/dev/docs/setting-up-sitl-on-linux.html

DIY (2015c). SITL Simulator (Software in the Loop) | Developer.
URL http://dev.ardupilot.com/wiki/sitl-simulator-software-in-the-loop/

JSBSim (2015). JSBSim Open Source Flight Dynamics Model.
URL http://jsbsim.sourceforge.net/