BeagleSystems Documentation

BeagleSystems GmbH

Jan 28, 2022

CONTENTS:

CHAPTER

ONE

HOWTOS

1.1 Install the BeagleSystems Software on a drone

To gain access to the repository, we first have to do some setup:

```
$ git config --global http.postBuffer 524288000
$ ssh-keygen -t rsa -b 4096 -C "your_email@beaglesystems.com"
$ cat ~/.ssh/id_rsa.pub
```

Upload the public key to your github settings to be able to download the repository.

```
$ mkdir Development
$ cd Development
$ git clone git@github.com:BeagleSystems/BeagleComrade -b develop
```

Create the file ~/.beaglerc with the following content.

```
#!/bin/bash

# !!! Double check that these paths are correctly set !!!
source /opt/ros/noetic/setup.bash
source /home/david/Development/beaglesystems/BeagleComrade/devel/setup.bash
export ROS_PACKAGE_PATH=/home/david/Development/beaglesystems/PX4-Autopilot:/home/
↪david/Development/beaglesystems/PX4-Autopilot/Tools/sitl_gazebo:$ROS_PACKAGE_PATH

# These exports are necessary if you try to run ROS debugging tools connected via␣
↪WiFi to the drone. Use the same exports on your local computer
#export ROS_HOSTNAME=192.168.8.100
#export ROS_IP=$ROS_HOSTNAME
#export ROS_MASTER_URI=http://$ROS_HOSTNAME:11311

export GST_PLUGIN_PATH=/usr/local/lib/aarch64-linux-gnu/gstreamer-1.0:/usr/lib/
↪aarch64-linux-gnu/gstreamer-1.0
export LD_LIBRARY_PATH=/usr/local/lib/aarch64-linux-gnu:$LD_LIBRARY_PATH
```

1.2 Install FT4232H

..code-block:: sh

> sudo apt-get install libconfuse-dev

1.3 Compile and install MAVSDK

```
sudo apt-get install build-essential cmake git
sudo pip3 install protoc_gen_mavsdk
cd ~/Development/beaglesystems
git clone git@github.com:BeagleSystems/MAVSDK -b develop --recursive
cd MAVSDK
cmake -Bbuild/default -DCMAKE_BUILD_TYPE=Release -H.
sudo cmake --build build/default --target install
```

If modifying the proto files, make sure to generate the corresponding .h and .cpp files:

```
cmake -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=OFF -DBUILD_MAVSDK_SERVER=ON -
↪Bbuild/default -H. && tools/generate_from_protos.sh && tools/fix_style.sh .
cmake --build build/default -j8 && sudo cmake --build build/default --target install
```

To compile an example (e.g. logfile_download), go to the ./examples/logfile_download directory and run the following:

```
cmake -Bbuild -H. && cmake --build build -j4 && ./build/logfile_download udp://:24541
```

To install MAVSDK-Python:

```
git clone git@github.com:BeagleSystems/MAVSDK-Python --recursive
cd MAVSDK-Python
sudo pip3 install -r requirements.txt
sudo pip3 install -r requirements-dev.txt
./other/tools/run_protoc.sh
cp ../MAVSDK/build/default/src/mavsdk_server/src/mavsdk_server mavsdk/bin/
MAVSDK_BUILD_PURE=ON python3 setup.py build
pip3 install -e .
```

1.4 Install Zephyr Apps on embedded devices like the Nucleo

Run the following installation process on an Ubuntu 20.04 computer:

```
sudo apt update
sudo apt install --no-install-recommends git cmake ninja-build gperf   ccache dfu-
↪util device-tree-compiler wget   python3-dev python3-pip python3-setuptools python3-
↪tk python3-wheel xz-utils file   make gcc gcc-multilib g++-multilib libsdl2-dev␣
↪screen
sudo pip3 install west

cd /tmp
wget https://github.com/zephyrproject-rtos/sdk-ng/releases/download/v0.13.2/zephyr-
↪sdk-0.13.2-linux-x86_64-setup.run
chmod +x zephyr-sdk-0.13.2-linux-x86_64-setup.run
```

(continues on next page)

(continued from previous page)

```
./zephyr-sdk-0.13.2-linux-x86_64-setup.run -- -d ~/zephyr-sdk-0.13.2

mkdir -p ~/Development/beaglesystems/zephyr
cd ~/Development/beaglesystems/zephyr
git clone https://github.com/BeagleSystems/zephyr
west init -m https://github.com/BeagleSystems/zephyr --mr develop
west update
west zephyr-export
pip3 install --user -r ~/Development/beaglesystems/zephyr/zephyr/scripts/requirements.
↪txt
sudo cp ~/zephyr-sdk-0.13.2/sysroots/x86_64-pokysdk-linux/usr/share/openocd/contrib/
↪60-openocd.rules /etc/udev/rules.d
sudo udevadm control --reload

echo "source ~/Development/beaglesystems/zephyr/zephyr/zephyr-env.sh" >> ~/.bashrc
source ~/.bashrc

cd ~/Development/beaglesystems
git clone git@github.com:BeagleSystems/zephyr_apps
cd zephyr_apps/eh2000_mavlink
./flash.sh
```

For debugging, you might want to see the debug output from the nucleo:

```
screen /dev/ttyACM0 115200 8N1
```

To quit the screen, just type ctrl+a :quit (a colon before quit).

1.5 Compile the documentation

We adhere to the recommendations described on https://www.writethedocs.org/guide/ and https://documentation.divio.com/reference/. Reference guides are kept in a similar style as http://mavlink.io/en/services/mission.html.

```
sudo pip3 install sphinxcontrib-mermaid
sudo pip3 install sphinx-jinja sphinxcontrib-napoleon sphinx-rtd-theme
```

Install sphinx-bootstrap-theme:

```
cd ~/Development/beaglesystems
git clone git@github.com:dayjaby/sphinx-bootstrap-theme
cd sphinx-bootstrap-theme
sudo python3 setup.py install
```

Modify the CSS:

```
cd ~/Development/beaglesystems
git clone git@github.com:dayjaby/bootstrap
cd bootstrap
npm install
npm install grunt
... modify files in ./less/ ...
./node_modules/.bin/grunt dist
cp -r dist/* ~/Development/beaglesystems/sphinx-bootstrap-theme/bootstrap/static/
↪bootstrap-3.4.1/
```

Create the documentation:

```
sphinx-apidoc -f -o source/mqtt_bridge ../BeagleComrade/src/mqtt_bridge/src/mqtt_
↪bridge
make clean && make html
```

Instead of `make html` you can create the documentation via

```
python3 -msphinx . _build
```

1.5.1 To generate a pdf using latex

```
sudo apt install texlive-full
sudo apt install texlive-latex-extra
make latexpdf
```

The pdf file is located in doc/_build/latex/

1.6 Debug PX4 topics

NuttX shell, list all available commands and list all uORB topics:

```
nsh> help
nsh> uorb top
q
```

If you have GPS problems, please check:

```
nsh> listener vehicle_gps_position
```

Is jamming indicator below 40? If not, make sure that all USB3.0 cables are far away from the GPS sensor.

In a SITL environment, we can use GDB to analyze. I assume that you run PX4 via a robot_upstarted launch file. Make sure that the PX4 ros node is commented out. First, we have to compile PX4 in GDB mode:

```
DONT_RUN=1 make px4_sitl_default gazebo___gdb
```

Next, we start the node as root.

```
$ sudo su -
# export HOME=/home/beagle
# source $HOME/.beaglerc
# gdb --args $HOME/Development/beaglesystems/PX4-Autopilot/build/px4_sitl_rtps/bin/
↪px4 /$HOME/Development/beaglesystems/PX4-Autopilot/ROMFS/px4fmu_common -s etc/init.
↪d-posix/rcS -t /$HOME/Development/beaglesystems/PX4-Autopilot/test_data
(gdb) handle SIGCONT noprint nostop
(gdb) run
```

One example how to break on certain mavlink messages, e.g. with mavlink message ID 212:

```
<ctrl-c> if GDB is running already
(gdb) break MavlinkReceiver::handle_message
(gdb) continue
Thread 75 "mavlink_rcv_if1" hit Breakpoint 1, MavlinkReceiver::handle_message␣
↪(this=0x7fff8c000b20, msg=0x7fffc6ffad90)
```

(continues on next page)

(continued from previous page)

```
 at ../../src/modules/mavlink/mavlink_receiver.cpp:132
 132     {
(gdb) x/20i $pc
=> 0x55555561d420 <MavlinkReceiver::handle_message(__mavlink_message*)>:        push ␣
↪ %rbp
   0x55555561d421 <MavlinkReceiver::handle_message(__mavlink_message*)+1>:      push ␣
↪ %rbx
   0x55555561d422 <MavlinkReceiver::handle_message(__mavlink_message*)+2>:      mov  ␣
↪ %rsi,%rbx
   0x55555561d425 <MavlinkReceiver::handle_message(__mavlink_message*)+5>:      mov  ␣
↪ %rdi,%rbp
   0x55555561d428 <MavlinkReceiver::handle_message(__mavlink_message*)+8>:      sub  ␣
↪ $0x8,%rsp
   0x55555561d42c <MavlinkReceiver::handle_message(__mavlink_message*)+12>:    ␣
↪movzbl 0xa(%rbx),%eax
   0x55555561d430 <MavlinkReceiver::handle_message(__mavlink_message*)+16>:    ␣
↪movzbl 0x9(%rsi),%esi
   0x55555561d434 <MavlinkReceiver::handle_message(__mavlink_message*)+20>:    ␣
↪movzbl 0xb(%rbx),%edx
   0x55555561d438 <MavlinkReceiver::handle_message(__mavlink_message*)+24>:     shl  ␣
↪ $0x8,%rax
   0x55555561d43c <MavlinkReceiver::handle_message(__mavlink_message*)+28>:     or   ␣
↪ %rsi,%rax
   0x55555561d43f <MavlinkReceiver::handle_message(__mavlink_message*)+31>:     shl  ␣
↪ $0x10,%rdx
   0x55555561d443 <MavlinkReceiver::handle_message(__mavlink_message*)+35>:     or   ␣
↪ %rdx,%rax
   0x55555561d446 <MavlinkReceiver::handle_message(__mavlink_message*)+38>:     cmp  ␣
↪ $0x8a,%eax
   0x55555561d44b <MavlinkReceiver::handle_message(__mavlink_message*)+43>:     je   ␣
↪ 0x55555561d8d0 <MavlinkReceiver::handle_message(__mavlink_message*)+1200>
(gdb) delete
(gdb) break *0x55555561d446 if $eax == 212
(gdb) info registers $eax
(gdb) continue
```

1.7 How to debug USB

```
sudo apt install libboost-dev libpcap-dev
git clone https://github.com/aguinet/usbtop
cd usbtop
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
sudo make install
sudo modprobe usbmon
sudo usbtop
```

Compare that with the USB devices found via lsusb.

USB devices can fail if the voltage drops below 5V. To check the voltage on Jetson Nano, run:

```
cat /sys/bus/i2c/drivers/ina3221x/6-0040/iio:device0/in_voltage0_input
```

CHAPTER

TWO

HARDWARE

2.1 Payloads

2.1.1 EH2000

Network configuration

```
sudo nmcli con add type ethernet ifname enx00e04c68020a con-name EH2000
sudo nmcli con mod EH2000 ipv4.address 192.168.42.1/16
sudo nmcli con mod EH2000 ipv4.method manual
sudo nmcli con mod EH2000 connection.autoconnect yes
sudo nmcli con up EH2000
```

Test commands for the NuttX shell

Get the device information and reported feedback from the gimbal:

```
listener gimbal_device_information
listener gimbal_device_attitude_status
```

Do a camera trigger test:

```
camera_trigger test
```

Do a continuous camera trigger test until stopped:

```
camera_trigger test_interval
camera_trigger test_interval stop
```

Make the camera look forward and follow yaw:

```
eh2000 test follow
```

Make the camera look down and follow yaw:

```
eh2000 test lookdown
```

Make the camera look left/right with a pitch of 45 degrees down:

```
eh2000 test lookleft
eh2000 test lookright
```

We provide commands to test the camera zoomed in (50mm) and zoomed out (16mm) and automatically focussed:

```
eh2000 test zoomin
eh2000 test zoomout
eh2000 test focus
```

Prepare the camera for precision landing, which includes the following commands:

- zoom out (MAV_CMD_SET_CAMERA_ZOOM)
- auto focus (MAV_CMD_SET_CAMERA_FOCUS)
- follow yaw (MAV_CMD_DO_GIMBAL_MANAGER_PITCHYAW)
- lookdown (MAV_CMD_DO_GIMBAL_MANAGER_PITCHYAW)

```
eh2000 test precland
```

We prepared some profiles for the camera:

```
eh2000 test profile_auto
eh2000 test profile_shutter
```

Be aware that these commands do certain other things: They flash the SD card and set the save path, so that images are written to the SD card.

As a fallback option, ssh to the drone and run these commands:

```
# Format the SD card
curl -G "http://192.168.42.108:80/cgi-bin/configManager.cgi?action=formatMedia"
# Switch to manual mode
curl -G "http://192.168.42.108:80/cgi-bin/configManager.cgi?action=shootMode&mode=5"
# Set aperture to F5.6
curl -G "http://192.168.42.108:80/cgi-bin/configManager.cgi?action=apertureMode&
↪mode=16"
# Set ISO mode to AUTO
curl -G "http://192.168.42.108:80/cgi-bin/configManager.cgi?action=isoMode&mode=0"
# Set shutter speed to 1/2500
curl -G "http://192.168.42.108:80/cgi-bin/configManager.cgi?action=shutterSpeedMode&
↪mode=18"
# Set exposure compensation to -0.3EV
curl -G "http://192.168.42.108:80/cgi-bin/configManager.cgi?
↪action=exposureCompensationMode&mode=4"
# Set zoom to 0 (completely zoomed out)
curl -G "http://192.168.42.108:80/cgi-bin/configManager.cgi?action=setZoomValue&
↪value=0"
# Save images to SD card
curl -G "http://192.168.42.108:80/cgi-bin/configManager.cgi?action=setSavePath&path=1"
# Do a single capture
curl -G "http://192.168.42.108:80/cgi-bin/configManager.cgi?action=capture&mode=0"
```

Video Streaming Setting

A. Set the destination IP Address

Go to launch folder and open foxtech_eh2000.launch file

```
cd ~/Development/BeagleComrade/launch/
vim foxtech_eh2000.launch
```

Locate the IP Address line

```
/dst_addr
```

Move the cursor to the IP Address value (use l or arrow keys) and type

```
ci"
```

Change the IP Address to designated IP Address, double check your IP address.

Exit from the insert mode by pressing ESC and save the file

```
# save changes and exit
:wq
# discard changes and exit
:q!
```

Restart the service

```
sudo systemctl restart beagle
```

B. Set up QGroundControl

1. Go to General Page under Application Setting.
2. Set the video stream to UDP H264..
3. Change the port to 8554.

CHAPTER

THREE

REFERENCE

3.1 ROS Messages

3.1.1 ROS message definitions

Table 1: Mapping of MQTT and ROS topics

ROS message	Data format
beagle_interfaces/RtcmData	string house_id string rtcm_id uint32 length string data

3.2 MQTT interface

Connecting to MQTT

The MQTT broker is hosted on 18.196.92.225:1883.

Table 2: Mapping of MQTT and ROS topics

MQTT topic	In/Out	ROS topic	Data format	Timestamp[1]
house/+/rtcm/+/raw	→	/rtk/rtcm	*beagle_interfaces/RtcmData*	

[1] The timestamp is an additional field in a message and is based on time.time() at the time of transmission of the data. Timestamps of the samples themselves are not included if not stated otherwise.

3.2.1 MQTT interface configuration

```
mqtt:
  client:
    protocol: 4      # MQTTv311
    client_id_from_mac: ["eth2", "eno1", "eth1", "eth0"]
  connection:
    host: "18.196.92.225"
    port: 1883
    keepalive: 10
  account:
    username: "beagle"
    password: "beagleB0o12"
  will:
    topic: ~/disconnected
    payload: "{}"
    qos: 2
  disconnect_on_shutdown: False
serializer: json:dumps
deserializer: json:loads
bridge:
  - factory: mqtt_bridge.bridge:MqttToRosBridge
    msg_type: beagle_interfaces.msg:RtcmData
    topic_from: house/+/rtcm/+/raw
    topic_to: /rtk/rtcm
    wildcards: ["house_id", "rtcm_id"]
```

3.3 mqtt_bridge

3.3.1 mqtt_bridge package

Submodules

mqtt_bridge.app module

`mqtt_bridge.app.`**`mqtt_bridge_node`**`()`

mqtt_bridge.bridge module

`class` `mqtt_bridge.bridge.`**`Bridge`**

Bases: `object`

Bridge base class

Parameters

- **`_mqtt_client`** (*`mqtt.Client`*) – MQTT client
- **`_serialize`** – message serialize callable
- **`_deserialize`** – message deserialize callable

`static is_service``()`

class mqtt_bridge.bridge.**MqttToRosBridge**(*topic_from*, *topic_to*, *msg_type*, *frequency=None*, *qos=2*, *queue_size=10*, *wildcards=None*, *latch=False*)

Bases: *mqtt_bridge.bridge.Bridge*

Bridge from MQTT to ROS topic

Parameters

- **topic_from** (*str*) – incoming MQTT topic path
- **topic_to** (*str*) – outgoing ROS topic path
- **msg_type** (*class*) – subclass of ROS Message
- **frequency** (*float|None*) – publish frequency
- **queue_size** (*int*) – ROS publisher's queue size
- **wildcards** (*list-of-str|None*) – list of wildcards. If it is not None, replace any + in topic_from with the values in this list.
- **latch** (*bool|False*) – whether to latch the message

class mqtt_bridge.bridge.**RosToMqttBridge**(*topic_from*, *topic_to*, *msg_type*, *frequency=None*, *qos=2*, *retain=False*, *delete_retained_on_shutdown=False*, *wildcards=None*, *drop=None*, *timestamp=False*, *bool_convert=None*)

Bases: *mqtt_bridge.bridge.Bridge*

Bridge from ROS topic to MQTT

Parameters

- **topic_from** (*str*) – incoming ROS topic path
- **topic_to** (*str*) – outgoing MQTT topic path
- **msg_type** (*class*) – subclass of ROS Message
- **frequency** (*float|None*) – publish frequency
- **qos** (*int|2*) – MQTT QoS
- **retain** (*bool|False*) – whether to retain the message
- **delete_retained_on_shutdown** (*bool|False*) – delete the message on shutdown if it was retained
- **drop** (*list-of-str|None*) – if it is not None, delete all values for the given keys

class mqtt_bridge.bridge.**RosToMqttServiceBridge**(*topic*, *msg_type*, *qos=2*)

Bases: *mqtt_bridge.bridge.Bridge*

Bridge from ROS topic to MQTT

Parameters

- **topic** (*str*) – incoming ROS topic path
- **msg_type** (*class*) – subclass of ROS Service
- **qos** (*int|2*) – MQTT QoS

static is_service()

`mqtt_bridge.bridge.`**`create_bridge`**(*factory*, *msg_type*, ***kwargs*)

bridge generator function

Parameters

- **`factory`** (*`str|class`*) – Bridge class
- **`msg_type`** (*`str|class`*) – ROS message type
- **`kwargs`** (*`dict`*) – a dictionary of arguments for the bridge class initialization

Return Bridge bridge object

mqtt_bridge.mqtt_client module

`mqtt_bridge.mqtt_client.`**`create_private_path_extractor`**(*mqtt_private_path*)

`mqtt_bridge.mqtt_client.`**`default_mqtt_client_factory`**(*params*, *private_path_extractor*)

MQTT Client factory

Parameters **`params`** (*`dict`*) – configuration parameters

Return mqtt.Client MQTT Client

mqtt_bridge.util module

`mqtt_bridge.util.`**`extract_values`**(*inst*)

`mqtt_bridge.util.`**`lookup_object`**(*object_path*, *package='mqtt_bridge'*)

lookup object from a some.module:object_name specification.

`mqtt_bridge.util.`**`populate_instance`**(*msg*, *inst*)

Returns an instance of the provided class, with its fields populated according to the values in msg

Module contents

CHAPTER

FOUR

PRECISION LANDING

4.1 Precision Landing State Machine

4.2 Precision Landing Integration with Gimbal Camera

PYTHON MODULE INDEX

INDEX