Boxfish ROV

Telemetry, Command and Configuration Sentences

Boxfish Client Use Only – Do not publish

Version 0.8
26 Jan 2022

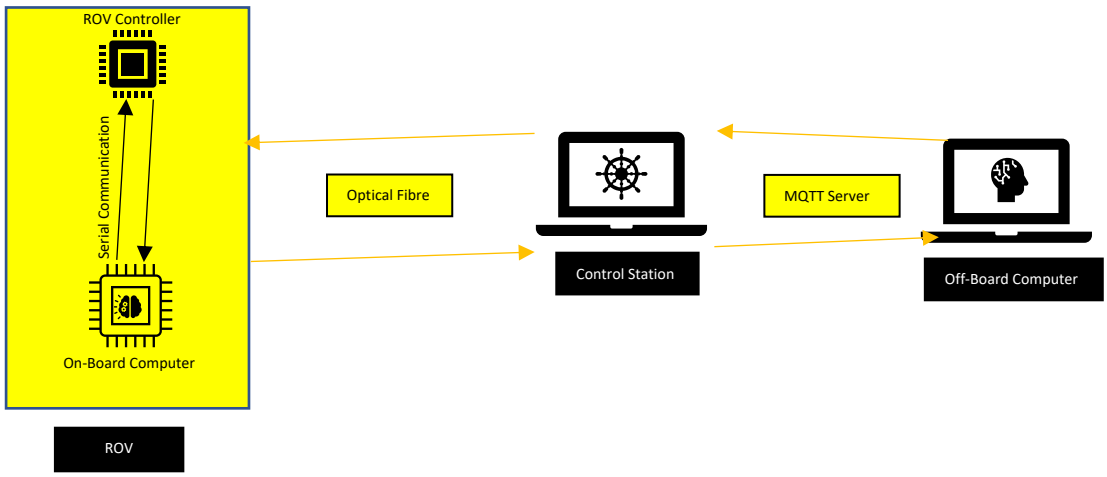
1 Introduction

The Boxfish software stack consists of the ROV controller software and the control station controller.

The control commands are sent by the Control Station Controller and the telemetry from the ROV is sent back to the Control station over optical fibre. On the control station, the incoming data is published over an MQTT based communication architecture.

Autonomous operations can be implemented in 2 ways:

1. **Onboard Solution**: A single board PC connected to the ROV controller via Serial port, allowing exchange of data and commands
2. **Off-board Solution**: A PC can be connected to the Control Station via Ethernet, which can send command to the ROV over MQTT after disabling the control station. The following diagram illustrates the flow of information in the system:



The autonomy controller can be written in any language supporting MQTT communication architecture. Python and C++ have been used in the past and are recommended.

2 Serial Port Specifications

The serial interface for the Boxfish ROV Controller (subsequently only referred to as Controller) telemetry operates at 1.8-5V TTL logic. The interface operates at 230400 baud, 8 data bits, 1 stop bit, no parity bits with neither software nor hardware flow control.

3 General Software Interface

The interface protocol design is based roughly on the National Marine Electronics Association's NMEA 0183 ASCII interface specification. The main differences are that the command characters can be more then 4 characters, the checksum is 32-bit instead of 8, and total sentence lengths can be up to 127 characters.

ROV sentences begin with the characters, "$PROV". The characters "$P" indicate that the sentence is a proprietary implementation and the characters and "ROV" indicate that it is from the ROV Controller. The character(s) that follow the characters "$PROV" uniquely identifies that particular proprietary sentence.

Sentence fields are fixed and new fields will only be added to the end of a sentence.

Most sentences are typically transmitted once per second. Exceptions are IMU data which is transmitted many times per second, and GPS data (if present) is not transmitted unless there is a fix.

In addition to this telemetry, commands sent to the controller are echoed.

All sentences transmitted by the Controller are terminated with <CR><LF>. A 8 hex digit checksum preceded by asterisk (*HHHHHHHH) is included just prior to the termination characters to allow for detection of errors in transmission. The checksum bytes (HHHHHHHH) are the hex ASCII representation of the 32-bit checksum of all the characters between the "$" and "*" characters, non-inclusive. The calculation of this checksum is documented in "Boxfish 32-bit Checksum".

4 Controller Telemetry Proprietary Sentences

Some fields are reserved and marked, NOT IMPLEMENTED, in which case the data transmitted will be integer or decimal 0.

4.1 **Controller Environment (PROVE)**

$PROVE,<1>,<2>,<3>,<4>,<5>,<6>*HHHHHHHH<CR><LF>

<1>	Temperature (Celsius)	23.4
<2>	Status, A means data for above field is valid	A
<3>	Relative Humidity (%)	68.2
<4>	Status, A means data for above field is valid	A
<5>	Pressure (mbar)	1014.1
<6>	Status, A means data for above field is valid	A

Temperature, humidity and pressure are of the internal environment.

4.2 **Marine Environment (PROVM)**

$PROVM,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,<10>,<11>,<12>,<13>,<14>,<15>,<16>*HHHHHHHH<CR><LF>

<1>	Water Temperature ©	23.4
<2>	Status, A means data for above field is valid	A
<3>	Depth (meters)	523.4
<4>	Status, A means data for above field is valid	A
<5>	Depth rate of change (meters/min)	2.43
<6>	Status, A means data for above field is valid	A

<7>	Salinity (%) NOT IMPLEMENTED	0.00
<8>	Status, A means data for above field is valid	F
<9>	Altimeter (meters)	1.54
<10>	Altimeter rate of change (meters/min)	8.65
<11>	Status, A means altimeter data is valid	A
<12>	Forward range (meters)	10.23
<13>	Forward rate of change (meters/min)	-42.18
<14>	Status, A means forward range data is valid	A
<15>	Is in water (1 = true)	1
<16>	Caps are on (1 = true)	1

4.3 **IMU (PROVI)**

$PROVI,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,<10>,<11>*HHHHHHHH<CR><LF>

<1>	Heading (magnetic, degrees)	175.1
<2>	Roll (degrees)	-1.72
<3>	Pitch (degrees)	2.34
<4>	Yaw (degrees)	23.9
<5>	Roll Rate (degrees / min)	-1.72
<6>	Pitch Rate (degrees / min)	2.34
<7>	Yaw Rate (degrees / min)	10.4
<8>	Status, A means data is valid	A
<9>	Time Stamp(sec)	2.515
<10>	Lights(%)	10
<11>	Compass Data Validity, A means data is valid	A

Pitch is positive for nose up. Roll is positive for clockwise rotation (facing forward), Yaw is positive for clockwise (viewed from top). 0 <= heading <= 360.0, -90.0 <= pitch <= 90.0, -180.0 < = roll <-= 180.0, -180.0 <= yaw <= 180.0. [Sorry]

4.4 **Thruster Status (PROVT)**

$PROVT,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,<10>,<11>,<12>,<13>,<14>,<15>,<16>*HHHHHHHH<CR><LF>

<1>	Thruster number (1 – N)	2
<2>	~~Thruster current (amps)~~	~~8.2~~
<3>	Status, A means data for above field is valid	A
<4>	Requested speed (% of max, - for reverse)	34
<5>	~~Measured RPM (I2C connected ESCs)~~	~~521~~
<6>	Status, A means data for above field is valid	A
<7>	~~Battery voltage (volts)~~	~~11.10~~
<8>	Status, A means data for above field is valid	A
<9>	~~ESC temp (C, I2C connected ESCs)~~	~~29.1~~
<10>	Status, A means data for above field is valid	A
<11>	Enabled (0 for disabled, 1 for enabled)	1
<12>	~~LED status (bitmask, 1=green, 2=red)~~	~~1~~
<13>	Status, A means data for above field is valid	A
<14>	~~ESC H/W Error Code (0-255)~~	~~0~~
<15>	Status, A means data for above field is valid	A
<16>	~~ESC Firmware Version (0.0 if not available)~~	~~1.1~~

One sentence sent for each thruster. Note that depending on the configuration, battery voltage may be the same for all thrusters (if all batteries are connected together), or different if different battery banks are used to supply power to different sets of thrusters.

4.5 **Control System Status (PROVC)**

$PROVC,<1>,<2>,<3>,<4>,<5>,<6>,<7>*HHHHHHHH<CR><LF>

<1>	Controller uptime (seconds)	534
<2>	Last command received (seconds)	1
<3>	Failsafe return to surface (0=off, 1=on)	0
<4>	Firmware Version (text)	1.5.5C
<5>	Valid command count	2943
<6>	Checksum errors	8
<7>	Invalid commands	0

Controller uptime is approximate, and will reset back to 0 if the controller if the controller reboots (this should not occur.)

4.6 **Control System Logging (PROVL)**

$PROVL,<1>,<2>,<3>,<4>,<5>*HHHHHHHH<CR><LF>

<1>	Message Type (I=info, W=warning, E=error)	W
<2>	Message ID (0-32767)	257
<3>	Sub Message ID (0-32767)	48
<4>	Controller uptime (seconds)	534
<5>	Message (0-48 chars)	Failed to initialise lighting

Message cannot contain dollar ($), comma (,), asterisk (*), semicolon (;), tab, newline <LF>, or carriage return <CR>.

4.7 **Internal Temperature Monitoring (PROVK)**

$PROVK,<1>,<2>,<3>*HHHHHHHH<CR><LF>

<1>	Device Serial Number (14 hex digits)	00112233445566
<2>	Temperature (C)	27.4
<3>	Status, A means data for above field is valid	A

Hardware for PROVK is not normally installed. PROVE provides temp at the ROV Controller PCBA.

4.8 **Control System Steering Vector (PROVCTRL)**

$PROVCTRL,<1>,<2>,<3>,<4>,<5>,<6>*HHHHHHHH<CR><LF>

<1>	Forward Thrust, in %. Negative for backward	0.1
<2>	Sideways Thrust, in %. Negative for left	0.0
<3>	Vertical Thrust, in %. Negative for down	-0.2
<4>	Roll Torque. Negative for left	0.0
<5>	Pitch Torque, Negative for down	0.0
<6>	Yaw Torque, Negative for left	0.05

$PROVCTRL is sent by the controller as a part of the telemetry and is the current thrust vector which is the steering vector (from PROVSTEER) after (any) adjustment by the control system.

Example: $PROVCTRL,0.80,0.00,0.00,0.0,0.00,0.05*ABCDEF01

4.9 **Control System Status (PROVSTAT)**

$PROVSTAT,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,9>,<10>,<11>,<12>,<13>,<14>*HHHHHHHH<CR><LF>

<1>	ROV Coordinate System (0=sea, 1=ROV)	1
<2>	Stabilisation (0=off, 1=on)	1
<3>	Depth Hold (0=off, 1=on)	0
<4>	Heading Hold (0=off, 1=on)	0
<5>	Altitude Hold (0=off, 1=on)	0
<6>	Return to Surface (0=disable, 1=enable)	1
<7>	Thrusters Suspended (0=normal, 1=suspended)	0
<8>	Target Depth (MSW). Valid if Depth Hold is on.	43.23
<9>	Target Heading (degrees). Valid if Heading Hold is on.	120.3
<10>	Target Altitude (MSW). Valid if Altitude Hold is on.	10.00
<11>	Target roll (degrees)	12.2
<12>	Target pitch (degrees)	-22.2
<13>	Target yaw (degrees)	0.0
<14>	~~Internal pitch control variable (unsupported)~~	~~0.4122~~

$PROVSTAT is sent by the controller once per second.

Example: $PROVSTAT,1,1,0,0,0,1,0,43.23,120,10.00,12.2,-22.2,0.0*ABCDEF01

4.10 **Power (PROVPWR)**

$PROVPWR,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8><9>,<10>,<11>,<12>,<13>,<14>,<15>,<16>,<17>*HHHHHHHH<CR><LF>

<1>	Controller 3.3V supply voltage (volts)	3.34
<2>	Controller 5V supply voltage (volts)	4.99
<3>	Controller battery voltage	15.10
<4>	Status, A means data for above field is valid	A
<5>	~~Aux 24V supply voltage~~	~~23.71~~
<6>	Status, A means data for above field is valid	A
<7>	~~Aux 12V supply voltage~~	~~12.01~~
<8>	Status, A means data for above field is valid	A
<9>	~~Critical 5V supply voltage~~	~~5.02~~
<10>	Status, A means data for above field is valid	A
<11>	~~Secondary 5V supply voltage~~	~~5.12~~
<12>	Status, A means data for above field is valid	A
<13>	Laser supply voltage	3.18
<14>	Status, A means data for above field is valid	A
<15>	Battery Percentage Remaining	52.3
<16>	Status, A means data for above field is valid	A
<17>	Sensor Power is On, A indicates true.	A

4.11 Other **Regularly echoed commands**

The commands PROVSTEER, PROVATT, and PROVTRIM are echoed back with their current values on regular intervals.

5 Controller Steering, Operations and Configuration Sentences

5.1 **No Operation (PROVNOOP)**

$PROVNOOP*HHHHHHHH<CR><LF>

	No parameters	

If the ROV Controller does not receive a command for 10 minute it will trigger emergency ascent. PROVNOOP can be used to reset this timeout without issuing a command.

Example: $PROVNOOP*ABCDEF01

5.2 **Change Coordinate System (PROVCOORD)**

$PROVCOORD,<1>*HHHHHHHH<CR><LF>

<1>	Turn ROV-based coordinate system on or off. Default is off.	0

The Boxfish ROV is able to interpret steering commands in two different coordinate systems: Sea or ROV. The default is Sea.

Sea: In this mode, the X- and Y-Axis are parallel with the horizon. The Y-Axis is vertical. For example, if the rover is tilted downward and forward thrust is applied, the rover will not descend at an angle as the X-Axis is not tilting with the rover but stays in parallel with the horizon.

Rover: In this mode the coordinate system is ‘attached’ to the rover and the axis pan and tilt with the rover. For example, if the rover is tilted downward and forward thrust is applied, the rover will descend at an angle as the X-Axis is now also pointing downward.

Example: Turn on ROV-based coordinate system: $PROVCOORD,1*ABCDEF01

5.3 **Steering (PROVSTEER)**

$PROVSTEER,<1>,<2>,<3>,<4>,<5>,<6>*HHHHHHHH<CR><LF>

<1>	Forward Thrust, in %. Negative for backward	0.1000
<2>	Sideways Thrust, in %. Negative for left	0.0000
<3>	Vertical Thrust, in %. Negative for down	-0.2000
<4>	Roll Torque. Negative for left	0.0000
<5>	Pitch Torque, Negative for down	0.0000
<6>	Yaw Torque, Negative for left	0.05000

$PROVSTEER sends a Thrust Control Vector in the currently active coordinate system. These commands time out after 2 seconds, so steering commands must be regularly sent to keep the ROV moving. 1.0000 = 100%. Start with low values for Roll and Yaw torque as the ROV is very responsive.

Example: Fast forward and slow turn to right: $PROVSTEER,0.80,0.00,0.00,0.0,0.00,0.05*ABCDEF01

5.4 **Direct Thruster Control (PROVDIRECT)**

$PROVSTEER,<1>,<2>,<3>,<4>,<5>,<6>*HHHHHHHH<CR><LF>

<1>	Thruster 1 (-1.000 to 1.000)	0.1000
<2>	Thruster 2 (-1.000 to 1.000)	0.1000
<3>	Thruster 3 (-1.000 to 1.000)	0.1000
<4>	Thruster 4 (-1.000 to 1.000)	0.1000
<5>	Thruster 5 (-1.000 to 1.000)	0.1000
<6>	Thruster 6 (-1.000 to 1.000)	0.1000
<7>	Thruster 7 (-1.000 to 1.000)	0.1000
<8>	Thruster 7 (-1.000 to 1.000)	0.1000

$PROVDIRECT controls thruster speeds directly. It is recommended to turn off Stabilisation and Depth Hold before using.

Example: Move forward: $PROVDIRECT,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1*ABCDEF01

5.5 **Turn Stabilisation on/off (PROVSTAB)**

$PROVSTAB,<1>*HHHHHHHH<CR><LF>

<1>	Turn stabilisation on or off. Default is off.	1

Example: Turn stabilisation off: $PROVSTAB,0*ABCDEF01

5.6 **Turn Depth Hold on/off (PROVDHOLD)**

$PROVDHOLD,<1>*HHHHHHHH<CR><LF>

<1>	Turn depth hold on or off. Default is off.	1

Example: Turn depth hold on: $PROVDHOLD,1*ABCDEF01

5.7 **Control Light (PROVLIGHT)**

$PROVLIGHT,<1>,<2>*HHHHHHHH<CR><LF>

<1>	Light #	1
<2>	Brightness, in % (unsigned integer)	30

Light 0 is the reverse lights, Light 1 and 2 both control the main lights, light 3 and 4 will control secondary lighting if installed. Example: Set light #2 to 50%: $PROVLIGHT,2,50*ABCDEF01.

5.8 **Control Laser (PROVLASER)**

$ PROVLASER,<1>,<2>*HHHHHHHH<CR><LF>

<1>	Laser #	1
<2>	Brightness, in % (unsigned integer)	80

Example: Set laser #1 to 80%: $PROVLASER,1,80*ABCDEF01

Note that current laser systems only support 0 or 100 for brightness.

5.9 **Calibrate Compass (PROVCCAL)**

$ PROVCCAL,<1>*HHHHHHHH<CR><LF>

<1>	Seconds (unsigned integer)	60

Following the reception of this command the ROV must be rotated at least 360 degrees through all 3 axes within the number of seconds passed as argument 1. The command is echoed when the calibration is complete (i.e. number of seconds passed as argument 1 has passed). Compass calibration will be impaired if rotation in all three axis is incomplete.

Example: Start compass calibration: $PROVCCAL,120*ABCDEF01

5.10 **Zero Depth (PROVDZERO)**

$ PROVDZERO[,<1>]*HHHHHHHH<CR><LF>

<1>	Optional parameter if set to 1 will force zeroing of depth sensor even when depth is greater than 5m.	1

Example: Zero depth sensor: $PROVDZERO*ABCDEF01

5.11 **Reset (PROVRESET)**

$PROVRESET*HHHHHHHH<CR><LF>

	No parameters	

$PROVRESET performs a software reset of the controller.

Example: $PROVRESET*ABCDEF01

5.12 **Thruster Enable/Disable (PROVTHRUST)**

$PROVTHRUST,<1>,<2>*HHHHHHHH<CR><LF>

<1>	Thruster Number (1-N)	2
<2>	Disable = 0, Enable = 1	0

$PROVTHRUST is used to disable a thruster so it is not used, thrusters once disabled can be re-enabled using this command.

Example: $PROVTHRUST,2,0*ABCDEF01

5.13 **Test Thrusters (PROVTSTT)**

$PROVTSTT*HHHHHHHH<CR><LF>

	No parameters	

$PROVTSTT is used to test the thrusters – it drives each thruster forward, in order, for a few seconds at a slow speed. It can be used to test that all thrusters are working, but also to check they are rotating the correct way and that they are connected up in the correct order.

Example: $PROVTSTT*ABCDEF01

5.14 **Attitude Angles (PROVATT)**

$PROVATT,<1>,<2>,<3>*HHHHHHHH<CR><LF>

<1>	Roll, Negative for left	0.1
<2>	Pitch, Negative for down	-30.0
<3>	Yaw, Negative for left	-0.2

$PROVATT sets angle offsets to what is considered forward. Roll, pitch and yaw must be in the range -180.0 to 180.0 degrees. Example: Pitch down 30 degrees and roll left 5: $PROVATT,0,-30,-5*ABCDEF01

5.15 **Return To Surface (PROVRTS)**

$PROVRTS,<1>*HHHHHHHH<CR><LF>

<1>	Disable = 0, Enable = 1. Default is 1	1

$PROVRTS enables or disables the automated Return To Surface functionality. The ROV switches into RTS mode when the RTS timeout interval after the last command or NOOP message has passed. See PROVTOUT for setting the timeout interval.

Example: $PROVRTS,0*ABCDEF01

5.16 **Timeout intervals (PROVTOUT)**

$PROVTOUT,<1>,<2>*HHHHHHHH<CR><LF>

<1>	Stall timeout (stops thrust if no PROVSTEER command is received in this many seconds). Default is 2.	2
<2>	Return to surface timeout. Default is 600	600

$PROVTOUT sets the timeout interval used by for Return-To-Surface and Steering logic.

Example: $PROVTOUT,10,600*ABCDEF01

5.16 **Trim adjustment (PROVTRIM)**

$**PROVTRIM**,<1>,<2>,<3>,<4>,<5>,<6>*HHHHHHHH<CR><LF>

<1>	Forward Thrust, in %. Negative for backward	0.1
<2>	Sideways Thrust, in %. Negative for left	0.0
<3>	Vertical Thrust, in %. Negative for down	-0.2
<4>	Roll Torque. Negative for left	0.0
<5>	Pitch Torque, Negative for down	0.0
<6>	Yaw Torque, Negative for left	0.05

$PROVTRIM sends trim offsets to the Thrust Control Vector in the currently active coordinate system.

Example: Apply a trim of 6% forward and 5% right: $PROVTRIM,0.06,0.05,0.00,0.0,0.00,0.00*ABCDEF01

5.17 **Override Thruster Shutdown Battery Voltage (PROVSHDV)**

$PROVSHDV,**<1>***HHHHHHHH<CR><LF>

<1>	Shutdown Voltage	13.8

The thruster shutdown battery voltage defaults to 14.1 volts, but may be overridden by this command. When the battery voltage drops below this voltage the thrusters will no longer rotate. This command should only be issued in emergencies were recovery of the ROV supersedes damage to the batteries. The ROV will power off at 13.7V regardless of this setting.

5.18 **Thruster Suspend (PROVTSUS)**

$PROVTSUS,<1>*HHHHHHHH<CR><LF>

<1>	Enable thruster suspend (1=on, 0=off).	1

Enabling thruster suspend prevents the thrusters from rotating while retaining the status of all modes. When thruster suspend is disabled, normal operation resumes.

5.19 **Environmental Settings (PROVENV)**

$PROVENV,<1>,<2>*HHHHHHHH<CR><LF>

<1>	Magnetic declination	19.5
<2>	Never-exceed dive depth (MSW)	1000

$PROVENV sets the environmental parameters like magnetic declination and maximum dive depth. Overriding the Never-exceed dive depth to greater than the ROVs and attached equipment designed depth is not recommended. This setting is not saved between ROV controller resets.

6 Dive Critical Commands:

Although the previous section has included all the commands for the ROV, this section highlights the most critical ones that are used in a dive mission. The same enumerated in a sequence which is followed during the deployment:

1. Thruster Test: This commands starts a thruster test to see if all the thrusters are working:

   Thruster Test: $PROVTSTT*

2. Arm thrusters: To arm the ROV and control it, the suspend mode is disabled. Note that the ROV can only be armed in the water.

   Arm: $PROVTSUS,0*

3. **Steering operations**: The ROV can be commanded a thrust vector, which takes translate x(forward), y(sideways), z(depth), roll, pitch, yaw, thrust:

   Steer: $PROVSTEER,x,y,z,roll,pitch,yaw*

   Note: All thrust magnitudes must be values between -1 and 1.

4. Quick stop / Disarm thrusters: To disarm the ROV :

Disarm: $PROVTSUS,1*

7 Sample Python Code

A. MQTT based Communication

1 Reading ROV Telemetry

The following python code opens the to the mqtt topic called ***'boxfish/telemetry/rov'*** created by the control station server over which the telemetry messages are relayed. It further parses the incoming data. It is recommended that a class object is created to store all the information coming in and the object is used by the autonomous controller. The same can be found in the file called Sample_Telemetry_code.py

```
import paho.mqtt.client as mqtt
import signal
import time


client_name = 'MBARI_TELEM_AUTO'
exit_process=0
```

```
class attitude:
    ## Class to log the attitude of the ROV
    def __init__(self,name):
        self.name=name
        self.cur=0
        self.des=0
```

```
class updateData:
    #create attitude objects
    roll=attitude("roll")
    pitch=attitude("pitch")
    yaw=attitude("yaw")
    heading=attitude("heading")
    depth=attitude("depth")
    altitude=attitude("altitude")
    time    =attitude("time")
    attitudes=[roll,pitch,yaw,heading,depth,altitude]
    #update them every time they are received
```

```
def handler(signum,frame):
    global exit_process
    print("Exiting")
    exit_process=1
```

```
## Instantiate the Data class object
rovTelemetry=updateData()
```

```
"""------------------connect to ROV--------------------"""
## Mqtt Functions
def on_log(client, userdata, level, buf):
    print("log: ",buf)
```

```
def on_connect(client,userdata, flags,rc):
    if rc==0:
        print ("connected ok")
    else:
        print("Bad connection Returned code=",rc)
```

```
def on_disconnect(client, userdata, flags, rc=0):
    print("Dis-connected result code"+str(rc))
```

```
def on_publish(client,userdata,result):        #create function for callback
    # print("data published \n")
    pass
```

```
def on_message(client,userdata,msg):
    topic=msg.topic
    ## Decode the message
    message=msg.payload.decode('utf-8')

    ## Parse the message into its' components
    splitMessage=message.split(",")
```

```
    if splitMessage[0]=="$PROVSTAT":
        ## Get the ROV attitude targets from $PROVSTAT
        rovTelemetry.depth.des    =splitMessage[8]
        rovTelemetry.heading.des  =splitMessage[9]
        rovTelemetry.altitude.des =splitMessage[10]
        rovTelemetry.roll.des     =splitMessage[11]
        rovTelemetry.pitch.des    =splitMessage[12]
        rovTelemetry.yaw.des      =splitMessage[13].split("*")[0]
# Be mindful of the '*' It gets parsed for the last parameter

    if splitMessage[0]=='$PROVM':
        ## Depth and Altitude(If Altimeter is installed)
        rovTelemetry.depth.cur    =splitMessage[3]
        rovTelemetry.altitude.cur =splitMessage[9]

    if splitMessage[0]=='$PROVI':
        ## Get the ROV current Attitude
        rovTelemetry.heading.cur  =splitMessage[1]
```

```
        rovTelemetry.roll.cur     =splitMessage[2]
        rovTelemetry.pitch.cur    =splitMessage[3]
        rovTelemetry.yaw.cur      =splitMessage[4]
        rovTelemetry.time.cur     =splitMessage[9]
```

```
    if splitMessage[0]=='$PROVSTEER':
        ## An example for tracking what thrust vector is operating on the ROV
        print("Yaw : {} , Downward thrust=
{}".format(splitMessage[6],splitMessage[3]))
```

```
broker="192.168.183.240" #MQTT broker is always at 192.168.183.240 (Control
station IP)
```

```
#Instantiate client
client=mqtt.Client("Logger+{}".format('{}'.format(client_name)))
#Assign functions to the client
client.on_connect=on_connect
# client.on_log=on_log
print("connecting to ",client)
client.on_message=on_message
client.on_publish=on_publish
client.connect(broker)
```

```
# Main Loop
while exit_process==0:
    client.loop_start()
    client.subscribe('boxfish/telemetry/rov')# Subscibing to the Topic
    time.sleep(0.1)
    signal.signal(signal.SIGINT, handler)
```

2 Sending commands to ROV

The following python code publishes commands to the ROV over mqtt topic called ***'boxfish/command/rov'***. A checksum is used to construct the commands, which is documented in the code provided in terms of functions. Sample_Publish_Commands.py.

```
#Publish commands to the ROV

import paho.mqtt.client as mqtt
import time


client_name='MBARI_AUTO_CMDS'
```

```
"""------------------ROV Checksum--------------------"""
```

```
## Checksum Calculations
def NmeaCRC32Value(i):
    crc32_polynomial = 0xEDB88320
    crc = i
    for j in range(0,8):
     if ((crc & 1) != 0):
      crc = (crc >> 1) ^ crc32_polynomial
     else:
      crc >>= 1
    return crc
def NmeaChecksumCalculate(command):
    checksum = 0
    last = len(command)
    for pos in range(0,last):
     temp1 = (checksum >> 8) & 0x00FFFFFF
     ulChar = ord(command[pos])
     temp2 = NmeaCRC32Value((checksum ^ ulChar) & 0x0000FF)
     checksum = temp1 ^ temp2
    return checksum
def calculate_checksum(command):
    command = command[1:-1]
    checksum = NmeaChecksumCalculate(command)
    return ("$"+command+"*"+str("%08X" % checksum)+'\r\n')
    # print("$"+command+"*"+str("%08X" % checksum)+'\r\n')
```

```
broker_address="192.168.183.240"
print("creating new instance")
client = mqtt.Client("{}".format(client_name))
print("connecting to broker")
client.connect(broker_address) #connect to broker
time.sleep(1) # wait
```

```
while True:
    client.loop_start() #start the loop
    ## Creating the PROVSTEER Command, the primary vector command to make the
ROV drive using the checksum generator functions

cmd=calculate_checksum("$PROVSTEER,{},{},{},{},{},{}*".format(0.2,0,0,0,0)) ##
make ROV drive forward
    print(cmd)
    client.publish("boxfish/command/rov",cmd)
    time.sleep(0.1)
```

B. Serial Port Communication

The onboard computer is linked to the ROV controller via a serial port which supports 230400 baud, 8 data bits, 1 stop bit, no parity bits settings. The following snippet of code describe how the serial library can be used to open the serial port for reading and writing commands to the ROV controller.

```
import time
from datetime import datetime
import serial
import serial.tools.list_ports as spl
```

```
class serialPort:
    controllerName =''
    port_detected = False
    def __init__(self):
        print("Connect Controller")
        while self.port_detected == False:
            ports = list(spl.comports())
            for port in ports:
                print(port.__dict__) # Enumrate all the USB connections to the
computer and select the ROV controller
                if (str(port.__dict__['manufacturer'])!= 'None'):
                    if str(port.__dict__['name'])=='ttyUSB2':# Will be
different for each computer
                        self.controllerName= port.__dict__['device']
                        self.ControllerPort= serial.Serial(port=self.
controllerName,baudrate=230400,timeout=5)
                        self.port_detected = True
                        print("Port Detected and connected")
                        break
                    else:
                        print("Port Not Detected yet")
                        time.sleep(1)
        self.reset_flag()

    def reset_flag(self):
        self.port_detected = False
```

```
ROVController=serialPort() # Initialise the ROV Controller and wait till it is detected and connected


# Read from the controller
Response = ROVController.ControllerPort.read(b'\n')
Response = response.decode(‘ascii’)
Message  = response.split(“,”)

# Write to the controller
## The message needs to be prepared using a checksum, which is discussed in the MQTT based communication section

ROVController.ControllerPort.write(bytes('{}\n'.format(message),encoding="ascii"))
```