Guide to iOceanServerComms

Scott R. Sideleau
Naval Undersea Warfare Center (NUWC), Division Newport

May 11, 2010

Preface

Special thanks to Joe Burke (UMass-D ATMC) for the initial development and documentation efforts on the *iOceanServerComms* MOOS module.

The following document aims to extend the original documentation efforts to account for changes in the Frontseat Helm (i.e. UVC) of the OceanServer Iver2 Autonomous Undersea Vehicle (AUV).

Contents

1 State of the MOOS Module

Since late 2007, I have taken ownership of the original code base for this MOOS module and have worked closely with OceanServer Technology, Inc. to resolve several bugs in the serial communication architecture of the Frontseat Helm. As such, the *iOceanServerComms* module should be considered an "early beta release" that is currently undergoing significant re-design.

That being said, I am always looking for user feedback on what has been developed to this point and will – as is customary – provide as much support as I reasonably can to others attempting to use the *iOceanServerComms* application. Please refer to the **Support** section of this document for contact details.

2 Conventions

The following is a description of the conventions used throughout the *iOceanServer-Comms* MOOS module. The file "iOceanServerComms.h" declares the class, functions, and variables about to be discussed.

2.1 A Note on Hungarian Notation

As coded now, the *iOceanServerComms* MOOS module employs Hungarian notation; that is, the name of all variables are preceded by their intended use or type or both. From studying the code, the followed scheme appears to be primarily Systems Hungarian based – using prefix letters to indicate the variable type at the system level – as opposed to Apps Hungarian based, where the prefix letters would indicate the variable's purpose.

Regardless of the pros and cons of Hungarian notation in software engineering, I have opted to maintain the source code in its current form and will continue to extend the given nomenclature. As such, the following notes should be made to clarify the naming convention as it is understood now:

- `m_` – corresponds to an external (i.e. MOOS) variable. This is an item usually obtaining its data from the `MOOSdb`.
- `b` – indicates a variable of type *Boolean*.
- `df` – indicates a variable of type *double precision floating point*.
- `s` – indicates a variable of type *C++ string*.
- `n` – indicates a variable of type *integer*.

2.2 `CMOOSOceanServerComms` Class Variables

This is the primary class employed by the iOceanServerComms module, which contains the member variables responsible for interacting with the `MOOSdb`. Said private variables are discussed further, by group, in the following subsections.

2.2.1 Compass Variables

The compass variables are filled via a serial communication request with the Frontseat Helm application.

- `m_dfPitch`
  The current vehicle pitch angle, in decimal degrees, as reported by the Frontseat Helm. It is currently unclear if this is the *raw* pitch (i.e. the instantaneous value reported by the hardware compass) or the *adjusted* pitch (i.e. the pitch value geometrically calculated at the center of the vehicle).

- `m_dfRoll`
  The current vehicle roll angle, in decimal degrees, as reported by the Frontseat Helm.

- `m_dfMagHeading`
  The uncorrected vehicle heading, in decimal degrees, as reported by the Frontseat Helm.

- `m_dfDepth`
  The vehicle depth, transformed into meters, as reported by the Frontseat Helm. Please note that the frontseat reports and records this value in feet; hence, the need for the unit transformation.

2.2.2 GPS Variables

The GPS variables are filled via a serial communication request with the Frontseat Helm application. Please note that the Frontseat Helm **must** be running a mission in order for valid GPS messages to be passed.

- `m_dfLatOrigin`
  The origin latitude, specified in decimal degrees, usually set in a MOOS-IvP configuration file (i.e. a *.moos* file) and stored in the `MOOSdb`. The Frontseat Helm is unaware of this value and its meaning, which is to assist in the formation of the UTM Grid Coordinate Plane.

- `m_dfLonOrigin`
  The origin longitude, specified in decimal degrees, usually set in a MOOS-IvP configuration file (i.e. a *.moos* file) and stored in the `MOOSdb`. The Frontseat Helm is unaware of this value and its meaning, which is to assist in the formation of the UTM Grid Coordinate Plane.

- `m_dfLatNow`
  The most recently received latitude, in decimal degrees, from the Frontseat Helm. GPS data is polled at a 1-Hz rate.

- `m_dfLonNow`
  The most recently received longitude, in decimal degrees, from the Frontseat Helm. GPS data is polled at a 1-Hz rate.

- `m_dfMagVar`
  The most recently received magnetic variation, in decimal degrees, from the Frontseat Helm. This value is relative to True North.

- `m_sWarning`
  The most recently received flag to accompany the GPS data that indicates its validity. Because `pNav` is currently not used, all received GPS messages – when a Frontseat Mission is running – should be valid, as the Frontseat Helm currently predicts GPS location when underwater via the principle of dead reckoning. It is currently unclear if set and drift has been accounted for at this time.

2.2.3 State Variables

The State variables are filled via a serial communication request with the Frontseat Helm application.

- `m_bControl`
  This value is determined by the last "bit" of the State message (the N, M, or A). In order to send servo or primitive commands to the Frontseat Helm, this value must be set to **true**, else the vehicle is in "Autonomous" mode and the Frontseat Helm is driving.

- `m_bOverride`
  This value toggles the ability to send primitive commands to the Frontseat Helm. If **true**, primitive commands are sent to the Frontseat Helm. Primitive commands differ from servo commands by dictating the actual movement of the fins instead of simply commanding the Frontseat Helm to complete a desired action (i.e. dive to 1-meter). At this time, the sending of primitive commands is not operational as bugs are being worked out on the Frontseat and Backseat side.

- `m_dfMaxSpeed`
  The maximum speed of the configured Iver2 vehicle. As of now, this value is not used; rather, safety rules implemented in the Frontseat Helm are being used to cap surface and dive speeds on the vehicle.

- `m_nMaxPitch`
  Depending on vehicle operation mode, this value will vary from 25- to 90-degrees. When the user has control of the vehicle via iRemote or pHelmIvP, the "freewheeling" value of 90-degrees is used to allow greater control. When the vehicle is operating under Backseat Helm autonomy, the value is capped at 25-degrees to ensure safer operation of the vehicle. When the vehicle is in Frontseat Helm autonomy, the value has no effect. Regardless of vehicle operating mode, the safety rules on the Frontseat Helm take precedence.

2.2.4 YSI Variables

The YSI variables are filled via a serial communication request with the Frontseat Helm application.

- `m_bYSI`
  This value indicates whether or not YSI data will be polled for on the Frontseat Helm. If **true**, then the additional parameter is added to the data request and the YSI NMEA string will be parsed accordingly.

2.2.5 Control Variables

The following variables are used to store control parameters that will be formed into NMEA strings and transmitted to the Frontseat Helm via serial communications.

- `m_dfDesiredDepth`
  The desired next depth, specified in meters, which is later transformed into feet to account for the Frontseat Helm's use of standard units in the SendServoCommand() method.

- `m_dfDesiredSpeed`
  The desired next speed, specified in meters per second (m/s), which is later transformed into nautical miles per second (knots) to account for the Frontseat Helm's use of standard units in the SendServoCommand() method.

- `m_dfDesiredHeading`
  The desired next heading, specified in radians, which is later transformed into decimal degrees to account for the Frontseat Helm's use of standard units in the SendServoCommand() method.

- `m_nDesiredThrust`
  The desired next thrust setting, specified as a signed percentage. This is used by SendPrimitiveCommand(), which is currently under re-design.

- `m_nDesiredRudder`
  The desired next rudder setting, specified as a signed decimal degree. This is used by SendPrimitiveCommand(), which is currently under re-design.

- `m_nDesiredElevator`
  The desired next elevator setting, specified as a signed decimal degree. This is used by SendPrimitiveCommand(), which is currently under re-design.

- `m_nTimeout`
  The desired timeout to use before returning control to the Frontseat Helm. Current operating procedure dictates that should the Frontseat Helm reach the timeout limit without receiving another command from

the Backseat Helm, then the Frontseat Helm with take control and drive the vehicle to its next waypoint (usually a park point).

- `m_nWaypointNum`
  The current waypoint the autonomous Frontseat Helm was heading towards. This could be used to orchestrate jump-to-waypoint maneuvers initiated in the backseat. As of now, the Frontseat Helm's jump-to-waypoint feature is untested and, as such, the Backseat Helm's jump-to-waypoint methodology is under review and re-development.

2.3 `CMOOSOceanServerComms` Class Functions

This is the primary class employed by the iOceanServerComms module, which contains the functions responsible for interacting with the `MOOSdb` and the Frontseat Helm. Said private functions are discussed further in the following subsections.

2.3.1 `OnStartUp()` Method

Called when the class is started by the MOOS mainframe (i.e. `pAntler`). This function employs the `MissionReader` to parse the mission file and

- initialize the serial port connection to the Frontseat Helm,
- set the `LatOrigin` based on the mission file,
- set the `LongOrigin` based on the mission file,
- initialize the local UTM grid coordinate system,
- set the `CommTimeout` based on the mission file,
- check for the presence of the YSI sensors based on the mission file,
- and call `DoRegistrations()` to complete initial registration of variables with the `MOOSdb`.

2.3.2 `OnConnectToServer()` Method

This is an overloaded function taken from the CMOOSApp class that is called when iOceanServerComms has made a connection to the `MOOSdb`. Its primary purpose is to call `DoRegistrations()` to register specific variables in the `MOOSdb`.

2.3.3 `DoRegistrations()` Method

This method registers our initial set of desired variables in the `MOOSdb`. The variables are directly linked to the class variables described earlier in Section 2.2; please see said section for further comments.

- DESIRED_SPEED
- DESIRED_HEADING
- DESIRED_DEPTH
- VEHICLE_WPT_INDEX
- MOOS_MANUAL_OVERIDE
- DESIRED_RUDDER
- DESIRED_THRUST
- DESIRED_ELEVATOR
- ZERO_RUDDER
- ZERO_ELEVATOR
- VEHICLE_UNDERWAY

2.3.4 `OnNewMail()` Method

This function is called when any of the registered variables of interest are published to the `MOOSdb`. The registered variables, described in the previous section, are stored locally while `NEXT_WAYPOINT` is published and control is handed to `Iterate()`.

2.3.5 `Iterate()` Method

Called at an interval of 1/Apptick. This is the main application loop for the class and is responsible for initiating the appropriate Frontseat Helm communication functions (e.g. `SendServoCommand()`).

2.3.6 `RequestInfo()` Method

Forms a valid `$OSD` message in NMEA format to communicate to the Frontseat Helm via the serial connection. Depending on the value of `m_bYSI`, the message requests the following data:

- `C` - Compass
- `G` - GPS
- `S` - State
- `P` - Power
- `Y` - YSI

2.3.7 `SendServoCommand()` Method

Forms a valid `$OMS` message in NMEA format to communicate to the Frontseat Helm via the serial connection. The message contains the following *double precision floating point* data converted to *strings* for transmission:

- `DESIRED_COURSE`
- `DESIRED_DEPTH`
- `DESIRED_MAXPITCH`
- `DESIRED_SPEED`

2.3.8 `SendPrimitiveCommand()` Method

Forms a valid `$OMP` message in NMEA format to communicate to the Frontseat Helm via the serial connection. The message contains the following *integer* data converted to *strings* for transmission:

- `DESIRED_RUDDER`
- `DESIRED_ELEVATOR`
- `DESIRED_THRUST`

Please note that this method is under re-evaluation and re-development and is, therefore, not used.

2.3.9 `IdentifyMessage()` Method

Determines what message was received from the Frontseat Helm and hands said message off to the appropriate parsing method for further processing. It is important to note that the Compass message, as received from the Frontseat Helm, is in the old PNI format, whereas all other messages are in the newer NMEA format.

2.3.10 `ParseCompass()` Method

Parses the received PNI-formatted Compass message from the Frontseat Helm. It publishes the following *double precision floating point* values into the `MOOSdb`:

- `COMPASS_HEADING`
- `COMPASS_YAW`
- `COMPASS_ROLL_DEG`
- `COMPASS_PITCH_DEG`
- `NAV_DEPTH`
- `COMPASS_TEMPERATURE`

2.3.11 `ParseGPS()` Method

Parses the received NMEA-formatted GPS message from the Frontseat Helm. It publishes the following *double precision floating point* values into the `MOOSdb`:

- `GPS_LATITUDE`
- `GPS_LONGITUDE`
- `GPS_X`
- `GPS_Y`
- `GPS_HEADING`
- `GPS_YAW`
- `GPS_SPEED`
- `GPS_MAGNETICVARIATION`

And publishes the following Boolean if the GPS data is bad:

- `GPS_WARNING`

2.3.12 `ParseState()` Method

Parses the received NMEA-formatted State message from the Frontseat Helm. It publishes the following *double precision floating point* values into the `MOOSdb`:

- `NAV_LAT`
- `NAV_LONG`
- `NAV_X`
- `NAV_Y`
- `NAV_SPEED`

2.3.13 `ParsePower()` Method

Parses the received NMEA-formatted Power message from the Frontseat Helm. It publishes the following *integer* values into the `MOOSdb`:

- `BATTERY_PERCENT`
- `BATTERY_TIME`
- `BATTERY_WATTS`
- `BATTERY_VOLTS`
- `BATTERY_AMPS`

- `BATTERY_WATTHRS`

And the following *string* value:

- `BATTERY_STATE`

2.3.14 `ParseYSI()` Method

Parses the received NMEA-formatted YSI message from the Frontseat Helm. It publishes the following *double precision floating point* values into the `MOOSdb`:

- `YSI_TEMP`
- `YSI_SPCOND`
- `YSI_SALINITY`
- `YSI_DEPTH`
- `YSI_TURBIDITY`
- `YSI_ODO%`
- `YSI_ODO`
- `YSI_BATTERY`

And the following *string* values:

- `YSI_DATE2`
- `YSI_TIME`

2.3.15 `issueNextWaypoint()` Method

Forms a valid `$OJW` message in NMEA format to communicate to the Frontseat Helm via the serial connection. The message simply dictates to the frontseat what waypoint the vehicle is at and/or unto which the operator would like the vehicle to navigate.

3 Recommended AUV Setup

In the following section, NUWCDIVNPT's operational procedure for the OceanServer Iver2 AUV is outlined. This is what has worked for us, but feel free to pursue other methods and let us know to improve this section further.

3.1 Frontseat Helm Operation

Although it is possible to design a Frontseat Helm mission with OceanServer's VectorMap software and run the vehicle via the UVC2 application, our particular use case dictates that the Backseat Helm should control the vehicle fully. As such, the Frontseat Helm is provided a mission with two waypoints: a start point, and a park point. Vehicle operation is as follows:

1. Place the Iver2 vehicle in the water.
2. Establish WiFi connectivity and launch vehicle on two-point mission.
3. As vehicle heads to first point, toggle the `VEHICLE_UNDERWAY` flag in the Backseat Helm to give control to *iOceanServerComms*.
4. After Backseat Helm mission has completed, watch Frontseat Helm execute remainder of two-point mission, bringing the vehicle to a park point.
5. *untested* Use the jump to waypoint (`$OJW`) command to reset the Frontseat Helm and repeat Step 2 to execute a new Backseat Helm mission.

3.2 Backseat Helm Operation

The Backseat Helm is setup to startup MOOS and any necessary applications on system boot to ensure that they are not closed when the vehicle is out of WiFi range. Since we are running Debian "etch" Linux, modifying `/etc/rc.local` was necessary to automatically startup MOOS automatically. Please note that, depending on your Linux distribution, the methodology that you follow to automatically launch `pAntler` on your .moos file will vary.

iOceanServerComms is currently configured to wait for the `VEHICLE_UNDERWAY` flag to be toggled. Thus, a login to the vehicle is initiated to launch `iRemote`, where the "manual mode" can be toggled by pressing the 'o' (for operator control) button followed by the 'y' (for yes) button to confirm release of the vehicle. Afterwards, whatever mission was planned and placed in the .bhv file will be executed.

Because *iOceanServerComms* is starting up on vehicle boot, it is necessary to restart the Backseat Helm computer whenever a change to the .moos or .bhv file to be executed have been changed via the `sudo shutdown -r now` command.

4 Support

Below, you will find contact information for the respective resources that can be used whenever a problem is encountered.

4.1 Getting Help with the *iOceanServerComms* Module

To get help with the *iOceanServerComms* MOOS module, please contact:

- Scott R. Sideleau
  Naval Undersea Warefare Center (NUWC), Division Newport
  Code 2511, Tactical Control and Contact Management
  Tel: +1 401-832-3129
  Email: Scott.Sideleau@navy.mil

4.2 Getting Help with the Frontseat Helm

To get help with the Frontseat Helm or the Iver2 AUV itself, please contact:

- Technical Support
  OceanServer Technology, Inc.
  Tel: +1 508-678-0550
  Email: support@ocean-server.com
  FAQ: http://www.ocean-server.com/faq.html

4.3 Getting Help with the MOOS-IvP Helm

To get miscellaneous help with the MOOS-IvP Helm and its related software, please use the following official mailing list:

- Mailing List: moosivp@lists.csail.mit.edu
- Sign-up Site: https://lists.csail.mit.edu/mailman/listinfo/moosusers

4.4 Getting Help with the MOOS Core

To get miscellaneous help related to MOOS (i.e. the 'Core' components), please use the following official mailing list:

- Mailing List: moosusers@lists.csail.mit.edu
- Sign-up Site: https://lists.csail.mit.edu/mailman/listinfo/moosivp