MIT 2.S01 Introduction to Autonomous Underwater Vehicles

Lecture 5: Software and Hardware in the Loop Simulations

Supun Randeni

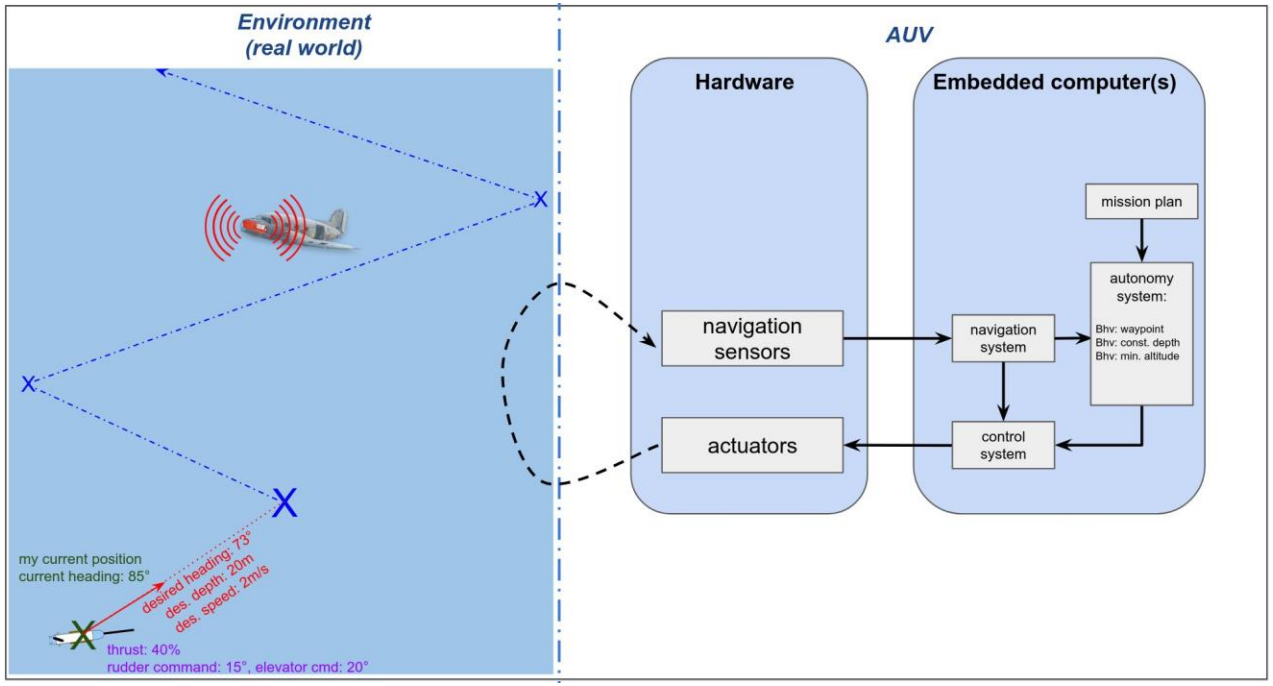
2026 Spring

MIT Sea Grant AUV Lab

MIT Marine Autonomy Lab

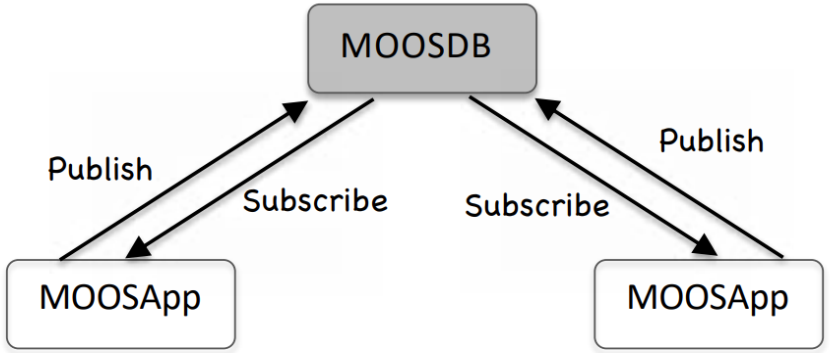
A recap: conceptual software architecture

- **Navigation sensors** measure parameters such as attitude & heading (AHRS), surface position (GPS), depth, etc.
- **The navigation system** uses these parameters to compute the underwater navigation solution (i.e., position, speed, etc.)
- **The mission plan software** keeps track of the overall goal of the mission and commands the current objective(s) to the helm.
- **The autonomy helm** makes the decision on the best desired course (e.g., desired heading, depth & speed) to follow, considering all currently active objectives.
- **The control system** computes the necessary actuator commands (e.g., control surface angles and thruster speed) to maintain the desired course commanded by the helm.
- **The actuators** (i.e., the control surfaces and thruster) provides forces and moments to move the AUV
- and repeat..



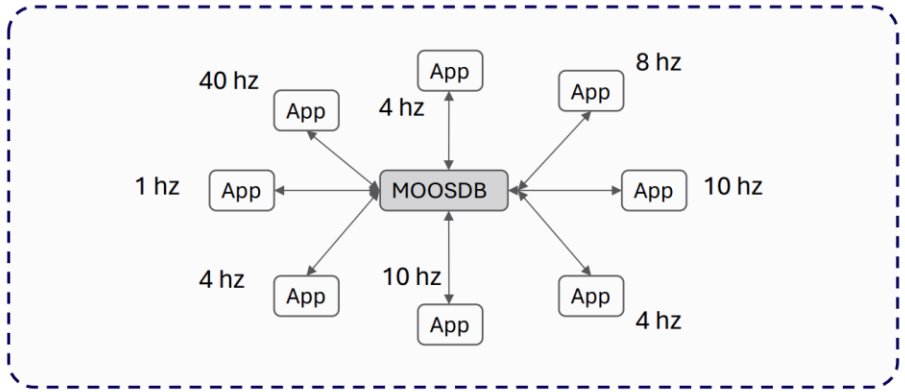
A recap: publish-subscribe protocol of MOOS middleware

- The primary role of a robot middleware is to facilitate communication between applications or programs.
- SeaBeaver AUVs use MOOS middleware, which has a publish-subscribe architecture.
- Each process is referred to as a "MOOS App", and the app handling the mail processing is an a called a MOOS Database, or MOOSDB for short.



- A MOOS Community is a collection of MOOS apps all connected to a single common MOOSDB
- A MOOS Community also typically has a name
- The frequency of the loop in each app can be configured by the user, and depends on the operation.

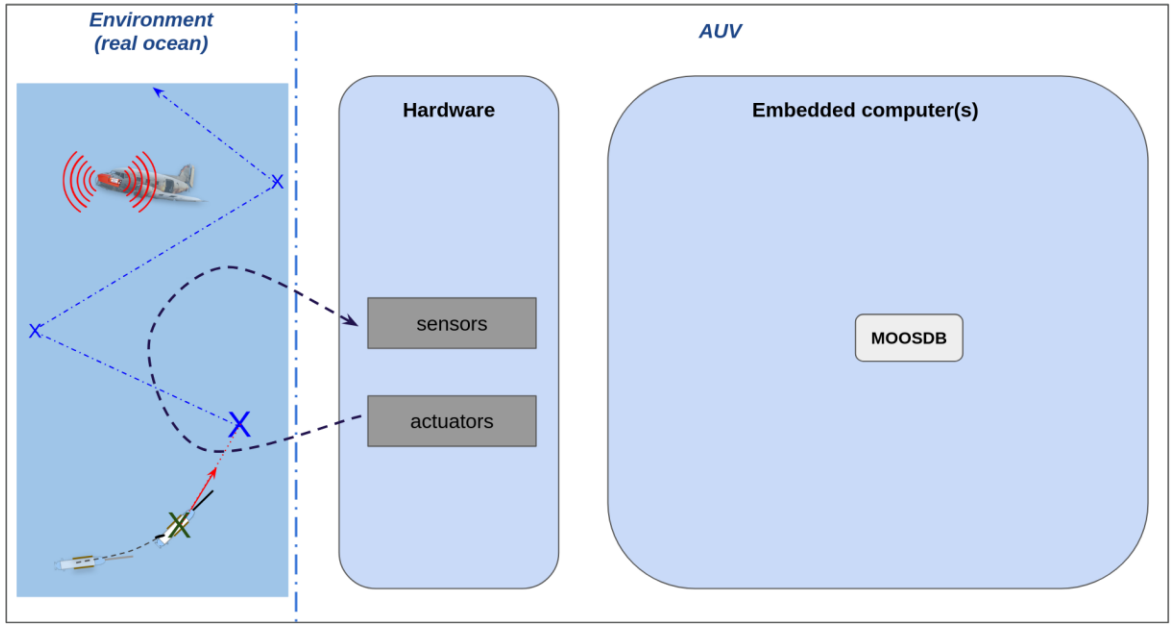
A MOOS community



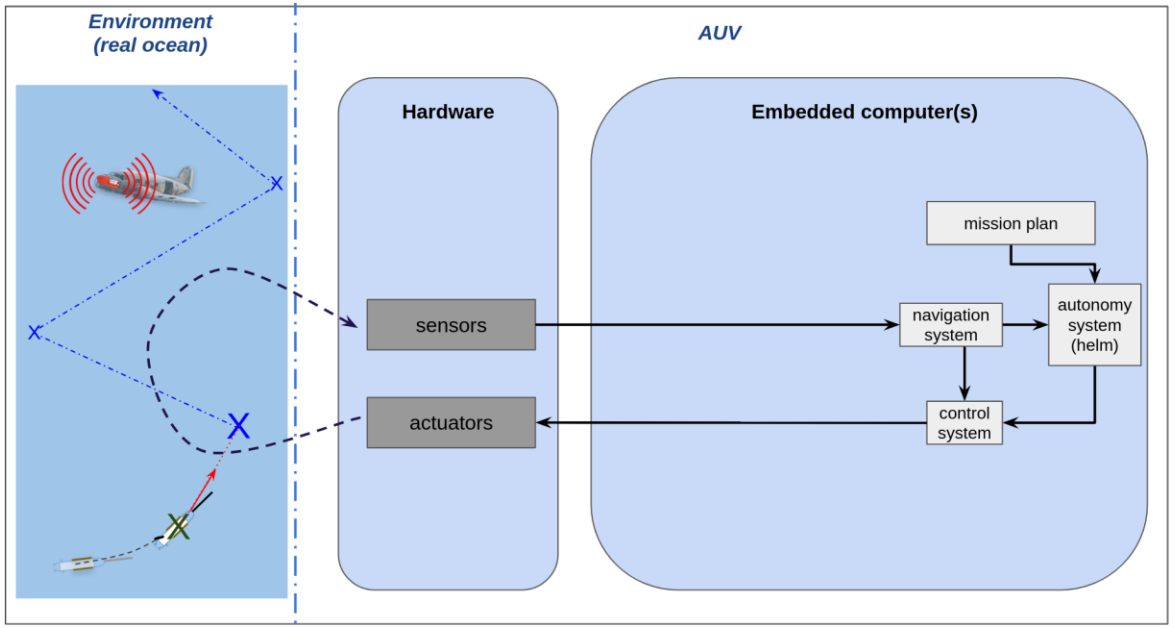
SeaBeaver software architecture

- SeaBeaver AUVs use *StackUxV* software project, together with MOOS-IvP, for sensing, navigation, control and autonomy.
- *StackUxV* uses MOOS middleware, and it consists of several MOOS applications.
- We will now introduce the actual MOOS applications within *StackUxV* that handles these conceptual tasks (i.e. sensing, navigation, control and autonomy)

Software architecture as a publish-subscribe diagram



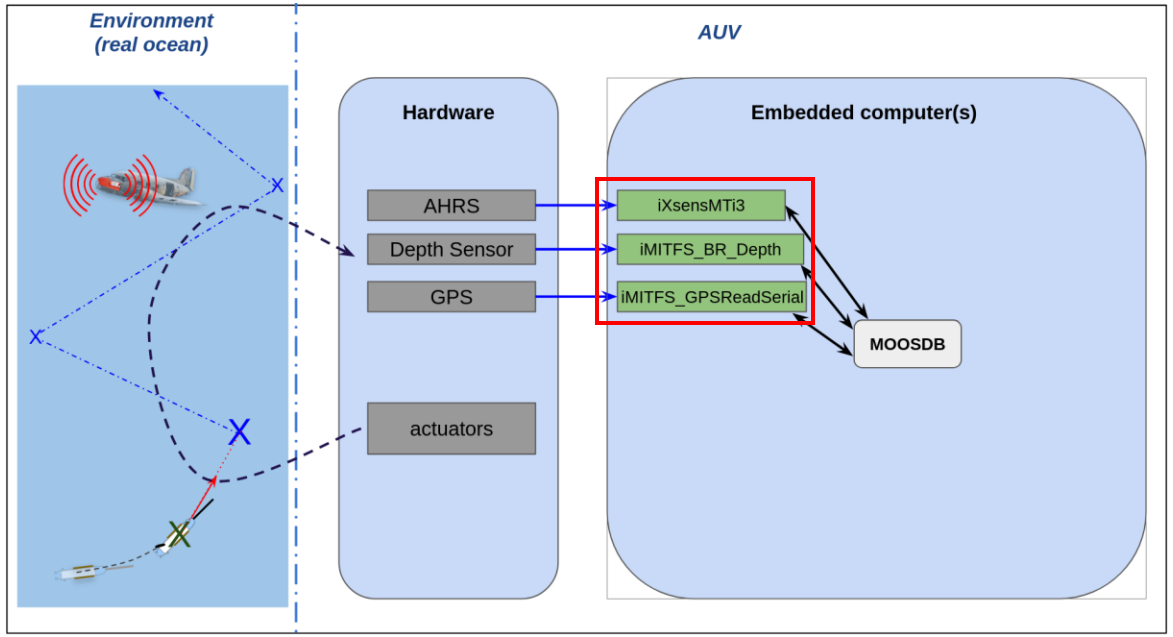
Software architecture as a data-flow diagram



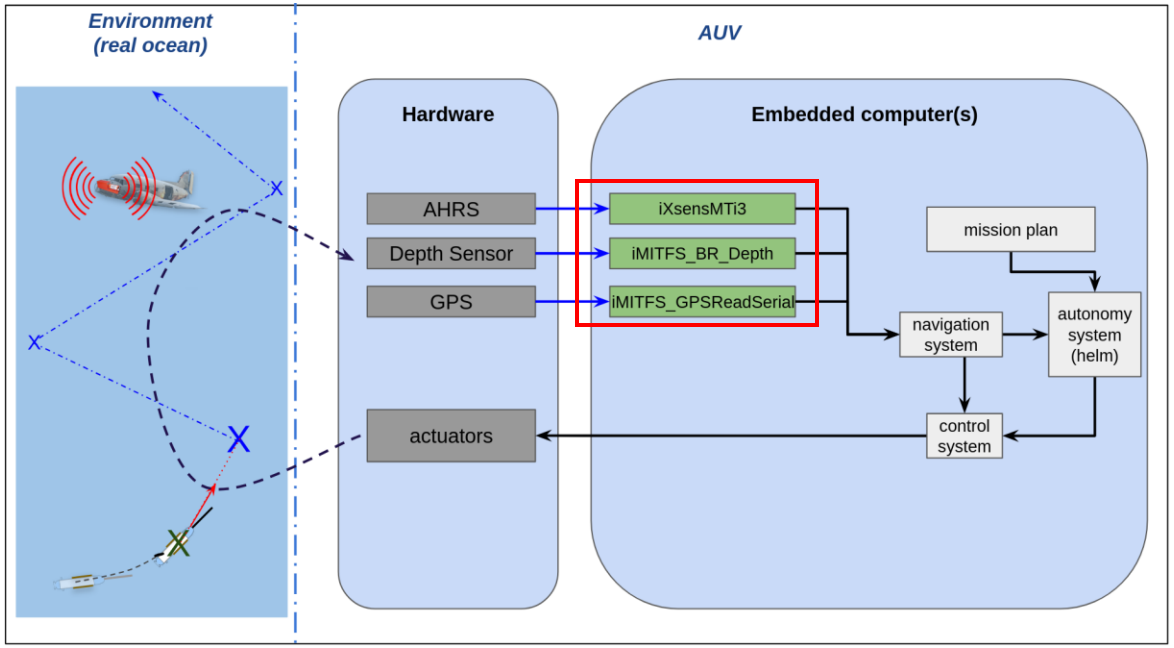
Sensor drivers

- SeaBeaver AUVs primarily use three driver apps from *StackUxV :*
  - ***iXsensMTi3*** – the MOOS app that interfaces with the XSens AHRS using UART communication
  - ***iMITFS_BR_Depth*** – the MOOS app that interfaces with the BlueRobotics depth sensor using I2C communication
  - ***iMITFS_GPSReadSerial*** – the MOOS app that interfaces with the GPS via UART communication
- These driver apps post the sensor data to the MOOSDB as MOOS messages:
  - RAW_IMU
  - RAW_GPS
  - RAW_DEPTH
- An example RAW_IMU message: time: 1711557931.4375551 raw_imu_data { sensor_id: 1 roll: 16.303578555463218 pitch: -5.9486337182577689 heading: 231.89094785947213 mag_declination: 0 is_true_heading: true accel_sensor_u: 2.8327710628509526 accel_sensor_v: 0.983140230178833 accel_sensor_w: -9.3787136077880859 roll_rate: -0.02950355596840382 pitch_rate: 0.983140230178833 heading_rate: 9.3787136077880859 mag_sensor_u: -0.23785543441772461 mag_sensor_v: 0.30035400390625 mag_sensor_w: 0.3094935417175293 sense_time: 1711557931.4375629 }

Software architecture as a publish-subscribe diagram



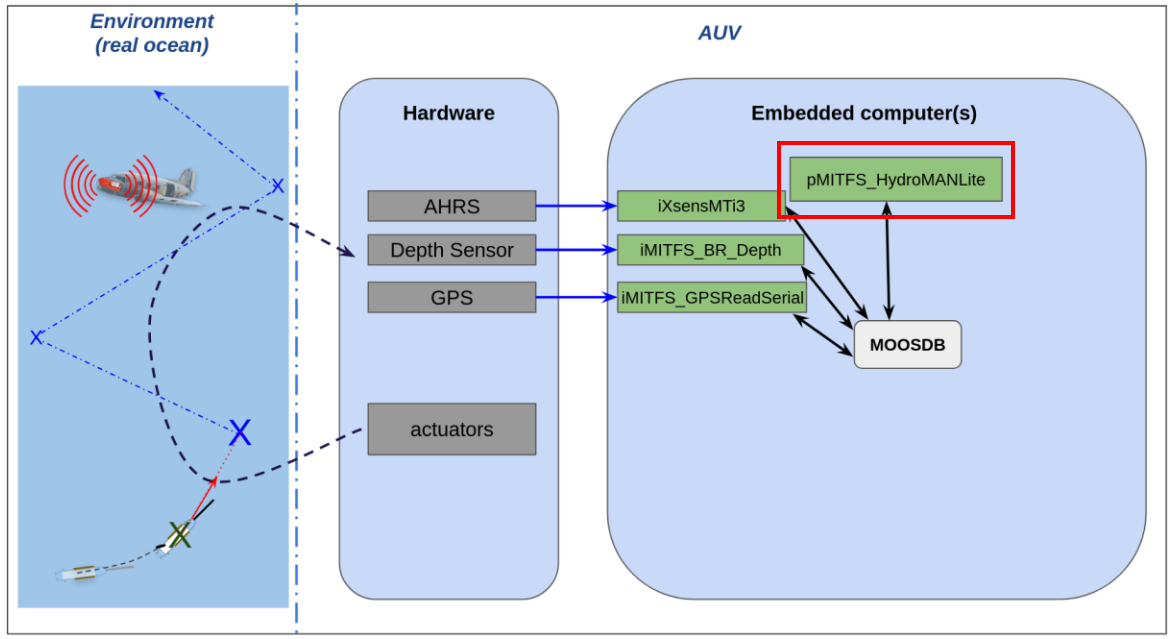
Software architecture as a data-flow diagram



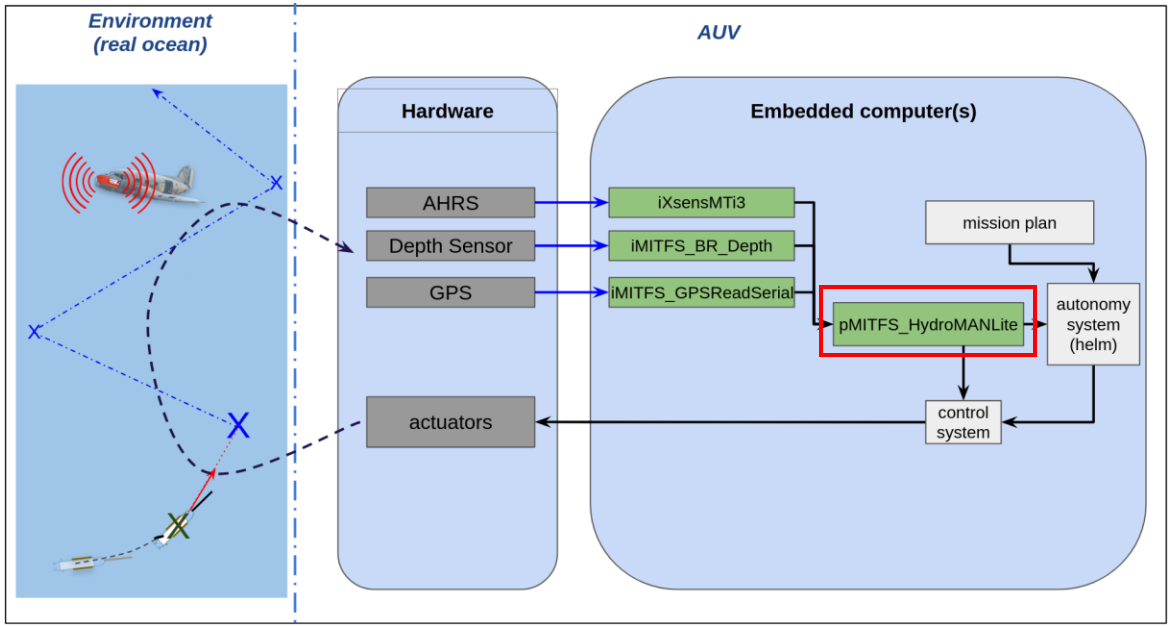
Navigation system (pMITFS_HydroMANLite)

- Reads raw sensor data from the AHRS, depth sensor and GPS by subscribing to:
  - RAW_IMU, RAW_DEPTH and RAW_GPS
- Processes the sensor data and computes a navigation solution while on the surface as well as underwater.
- Publishes the navigation solution to the MOOSDB with the message:
  - HYDROMAN_NAV_DATA
  - NAV_X, NAV_Y, NAV_LAT, NAV_LON, NAV_HEADING, NAV_DEPTH, NAV_SPEED
- An example HYDROMAN_NAV_DATA message: *time: 8566228785.3282547 nav_data { lat: 42.35808545498795 lon: -71.0869294526643 depth: 0.661040485039652 18 local_x: 42.012439333372733 local_y: -35.98125700273836 vel_xdot: 0.700593628482161 vel_ydot: -0.883719628620622 26 vel_ddot: 0.0053149055334286719 vel_u: 1.12775 vel_v: 0 vel_w: 0 imu_data { roll: -0.4982430346172943 pitch: -0.27002685246633218 heading: 141.59338724016959 } }*

Software architecture as a publish-subscribe diagram



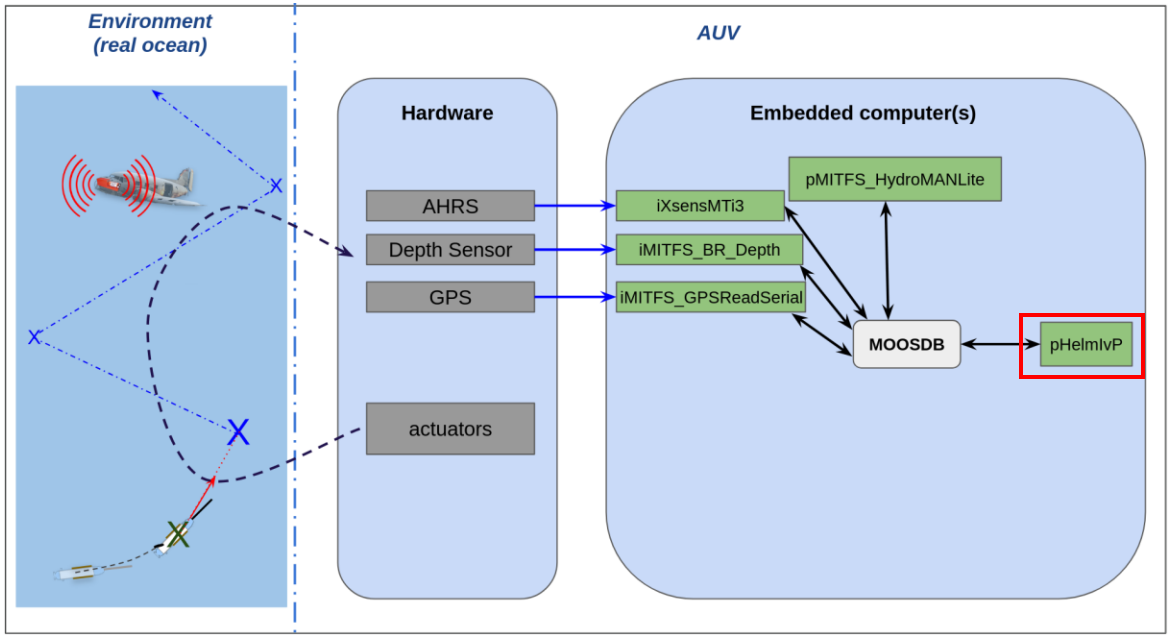
Software architecture as a data-flow diagram



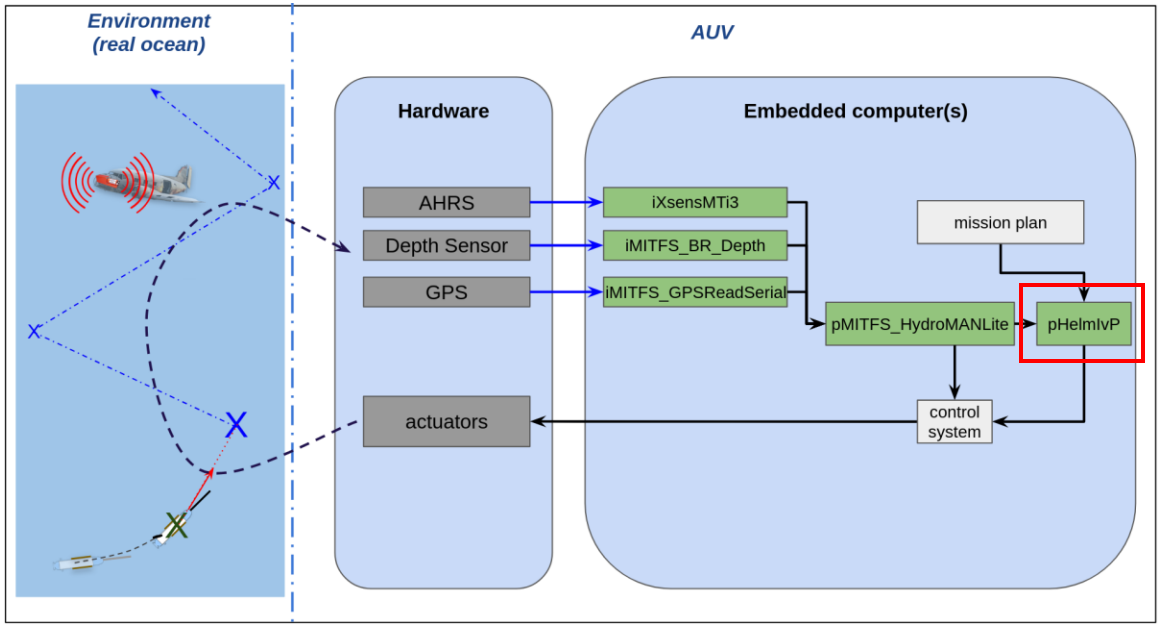
Autonomy helm (pHelmIvP)

- Reads the navigation solution by subscribing to NAV_*
- Computes the decision on the best desired course to follow, considering all currently active objectives.
- Publishes the final decision to the MOOSDB with the messages:
  - DESIRED_HEADING
  - DESIRED_DEPTH
  - DESIRED_SPEED

Software architecture as a publish-subscribe diagram



Software architecture as a data-flow diagram



Mission planning (pMITFS_MissionScript and pMITFS_MissionManager)

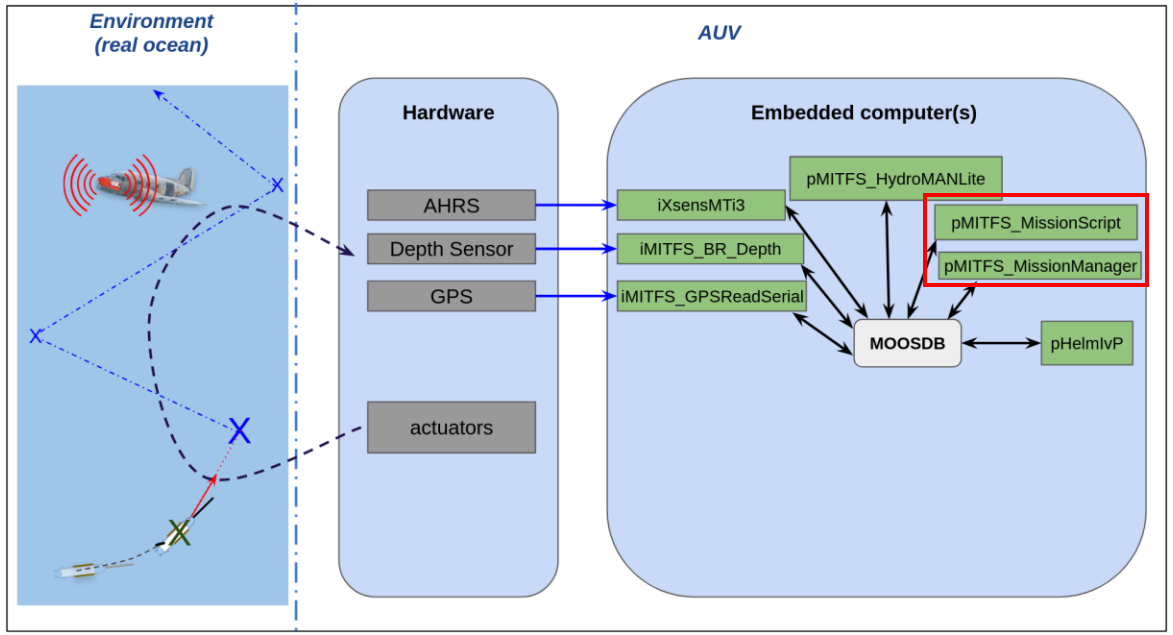
pMITFS_MissionScript

- Operator can script a list of behaviors to be executed as configuration parameters
- This app will inform *pMITFS_MissionManager* when the AUV needs to change mission parameters

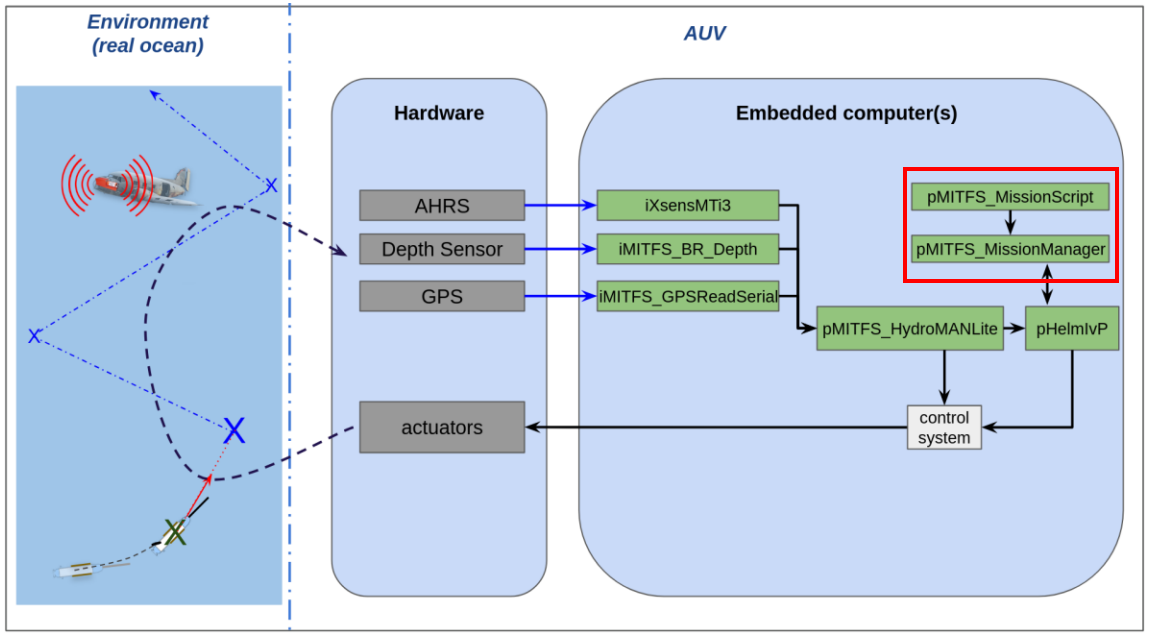
pMITFS_MissionManager

- Ensuring that the vehicle does not violate safety parameters, this app will command the current objective(s) to the helm

Software architecture as a publish-subscribe diagram



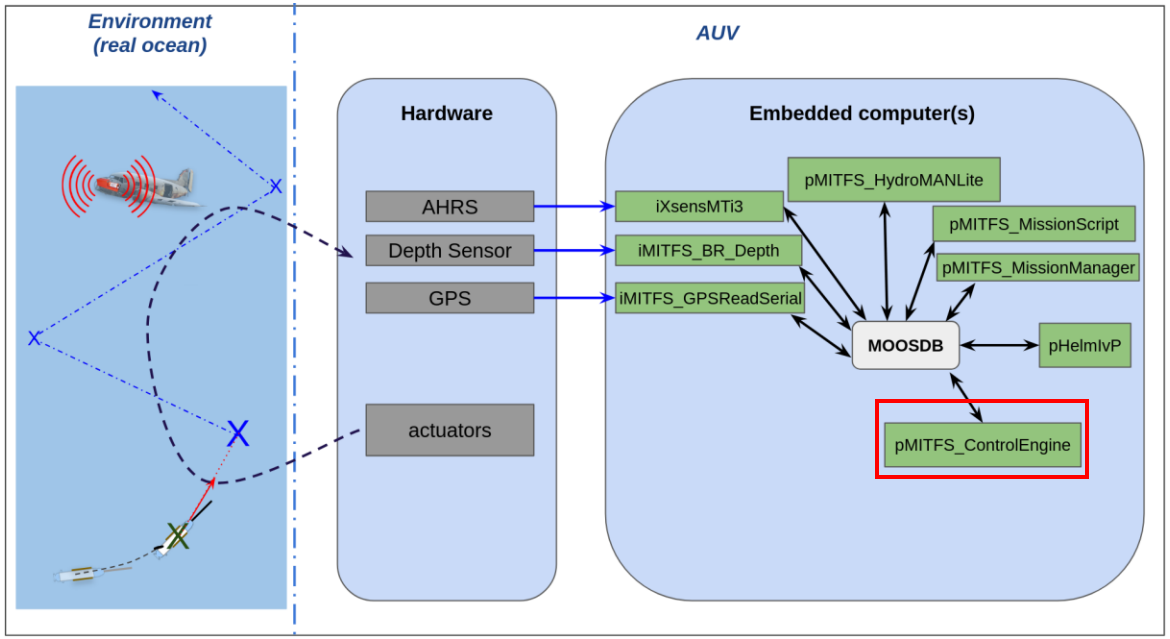
Software architecture as a data-flow diagram



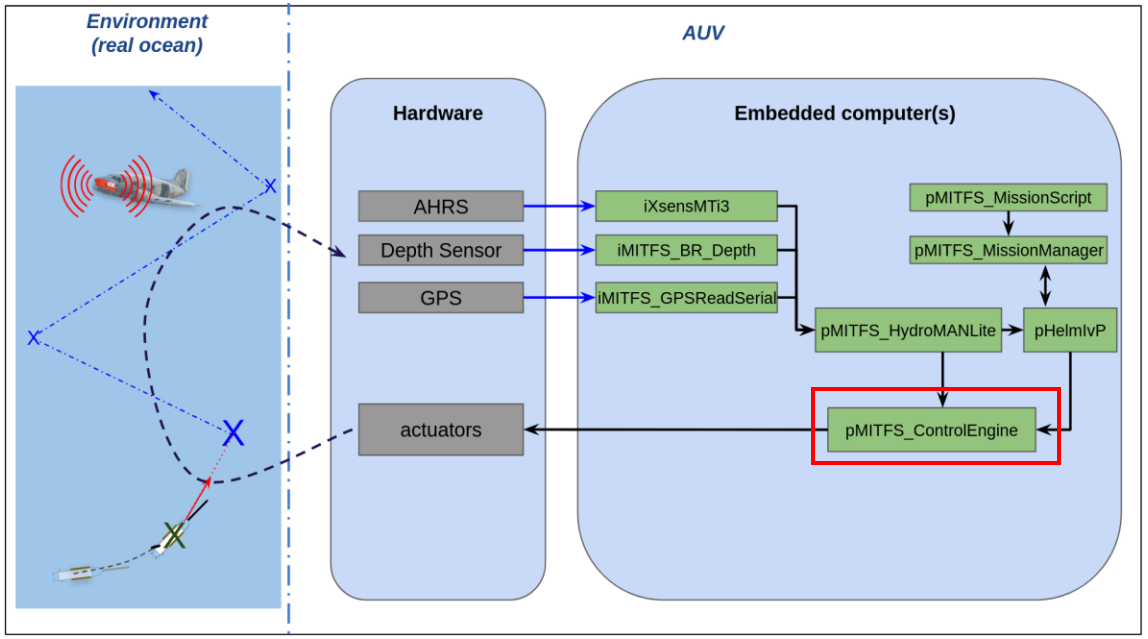
Control system (pMITFS_ControlEngine)

- Reads the course decisions made by *pHelmIvP* by subscribing to:
  - DESIRED_HEADING
  - DESIRED_DEPTH
  - DESIRED_SPEED.
- Computes the necessary control surface angle and thruster speed values to maintain the desired course.
- Publishes the control surface angle and thruster speed commands to the MOOSDB with the message:
  - ACTUATOR_COMMANDS

Software architecture as a publish-subscribe diagram



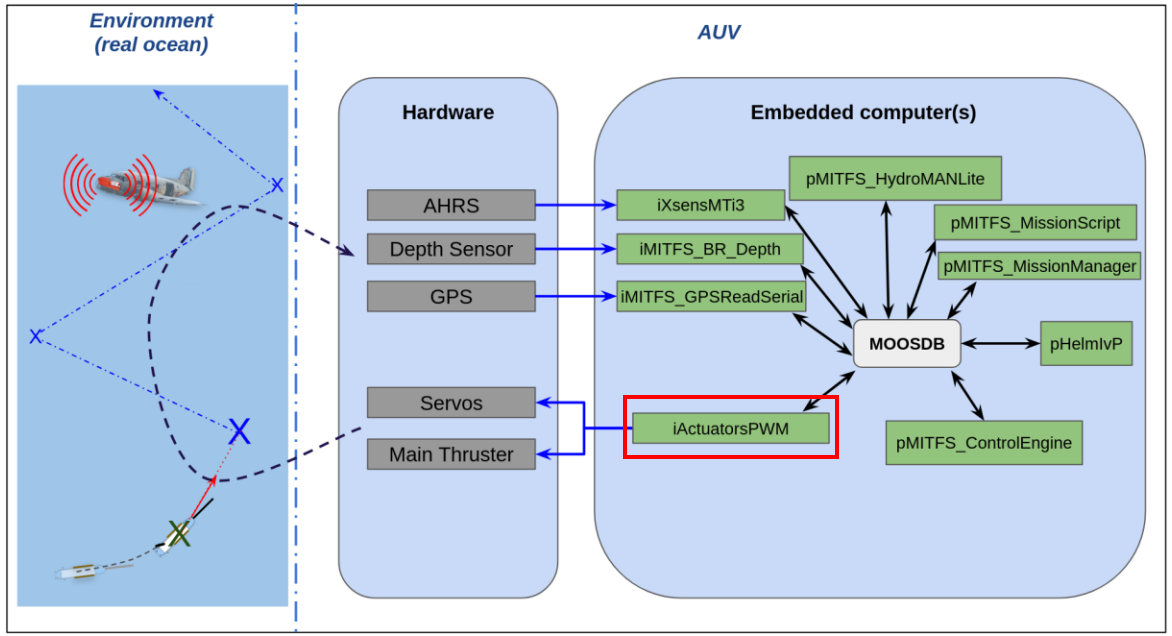
Software architecture as a data-flow diagram



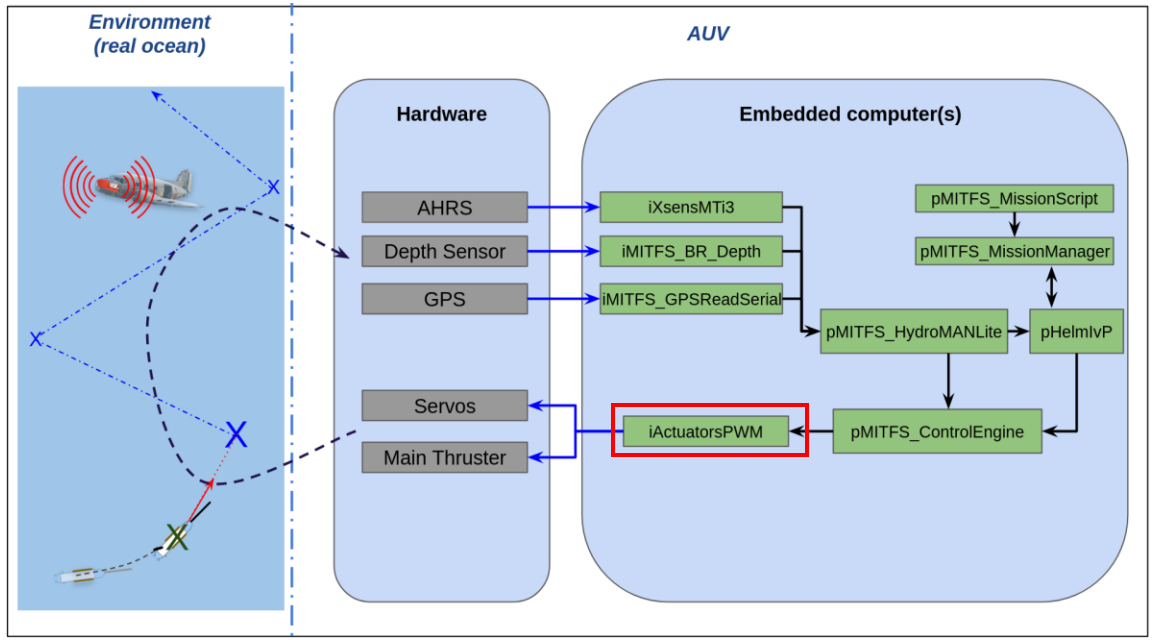
Actuator driver (iActuatorsPWM)

- The servo motors and thruster are driven by pulse-width modulation (PWM) signals.
- This app reads the ACTUATOR_COMMANDS message published by the *pMITFS_ControlEngine* app.
- Converts the commands to PWM values, and sends those PWM signals to the servos and thrusters.

Software architecture as a publish-subscribe diagram



Software architecture as a data-flow diagram



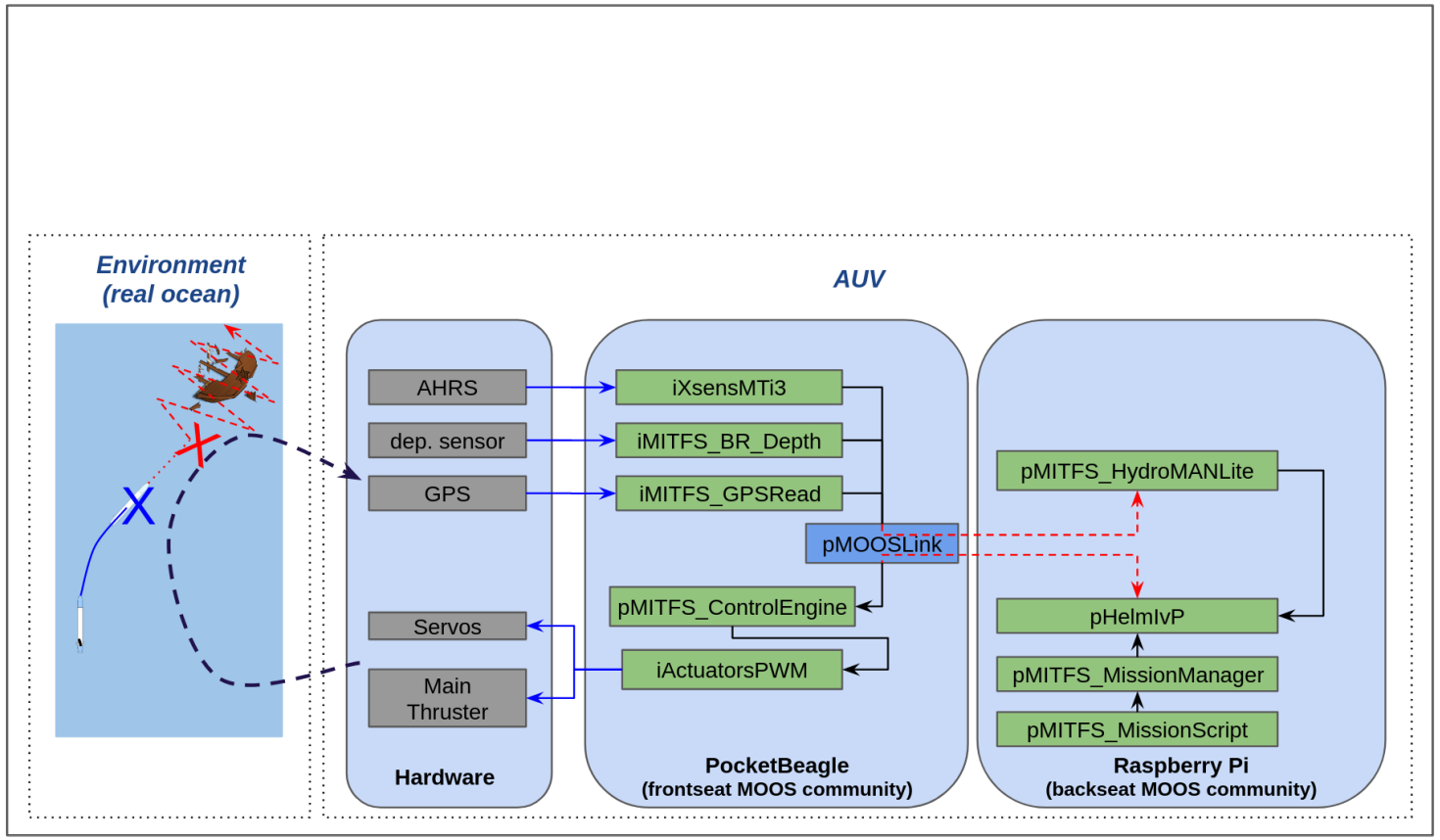
Frontseat and Backseat

- **Frontseat computer** (PocketBeagle):
  - Runs the hardware drivers.
  - Runs the low-level control system.

- **Backseat computer** (Raspberry Pi):
  - Runs higher-level autonomy and decision-making software.
  - Runs the underwater navigation software.

- **Two MOOS communities with two MOOSDBs**:
  - Frontseat MOOS community.
  - Backseat MOOS community.

- **Frontseat-Backseat interface**:
  - SeaBeaver uses a MOOS app called pMOOSLink to exchange MOOS messages between frontseat and backseat MOOS communities.

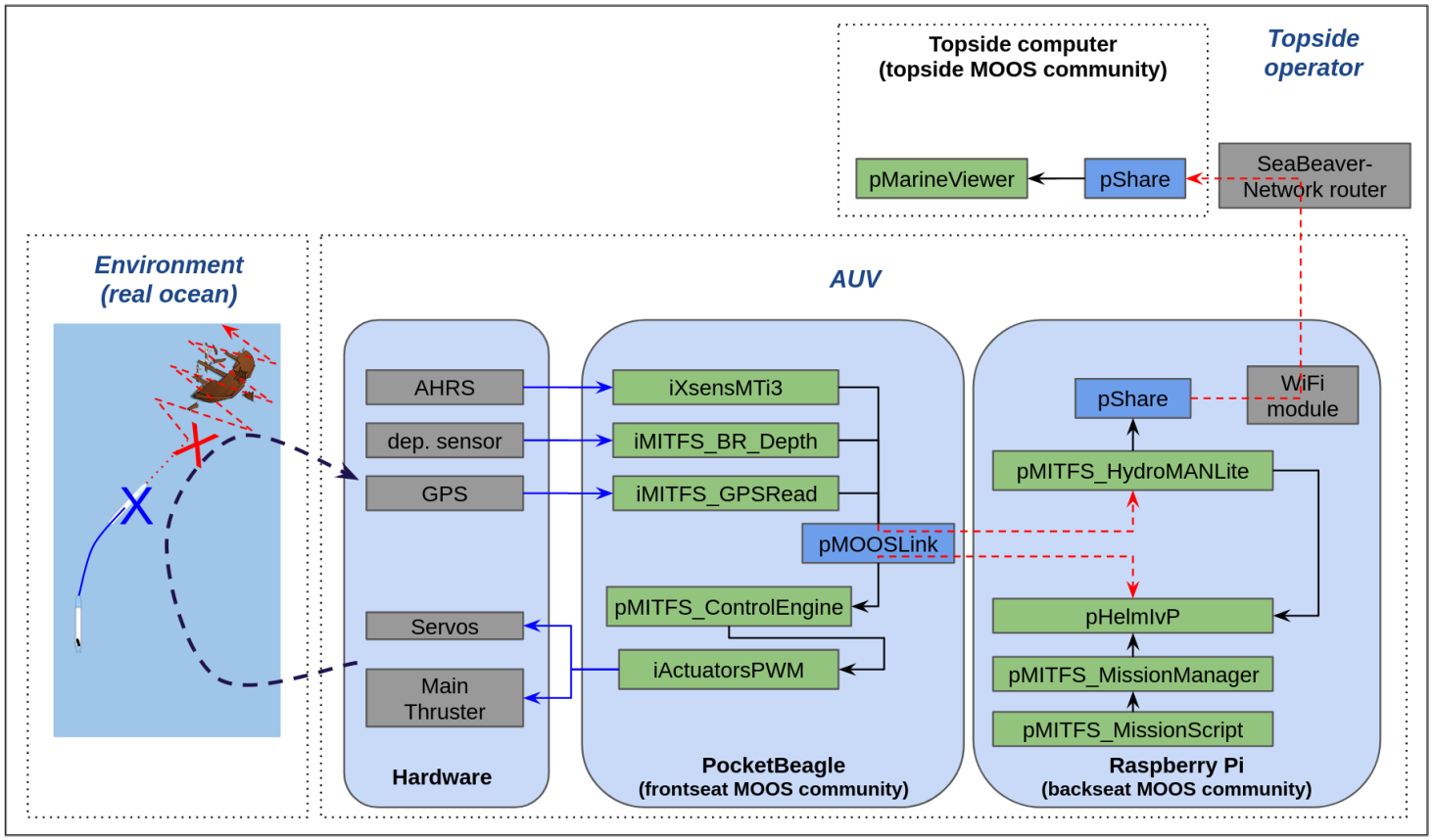


Topside MOOS community

- The AUV operator may choose to have *pMarineViewer* window on a topside computer to view the vehicle status when in proximity

- This will launch a third MOOS community called the "topside MOOS community"

- Vehicle-topside interface:
  - SeaBeaver uses a MOOS app called pShare to exchange messages between the backseat and topside MOOS communities.



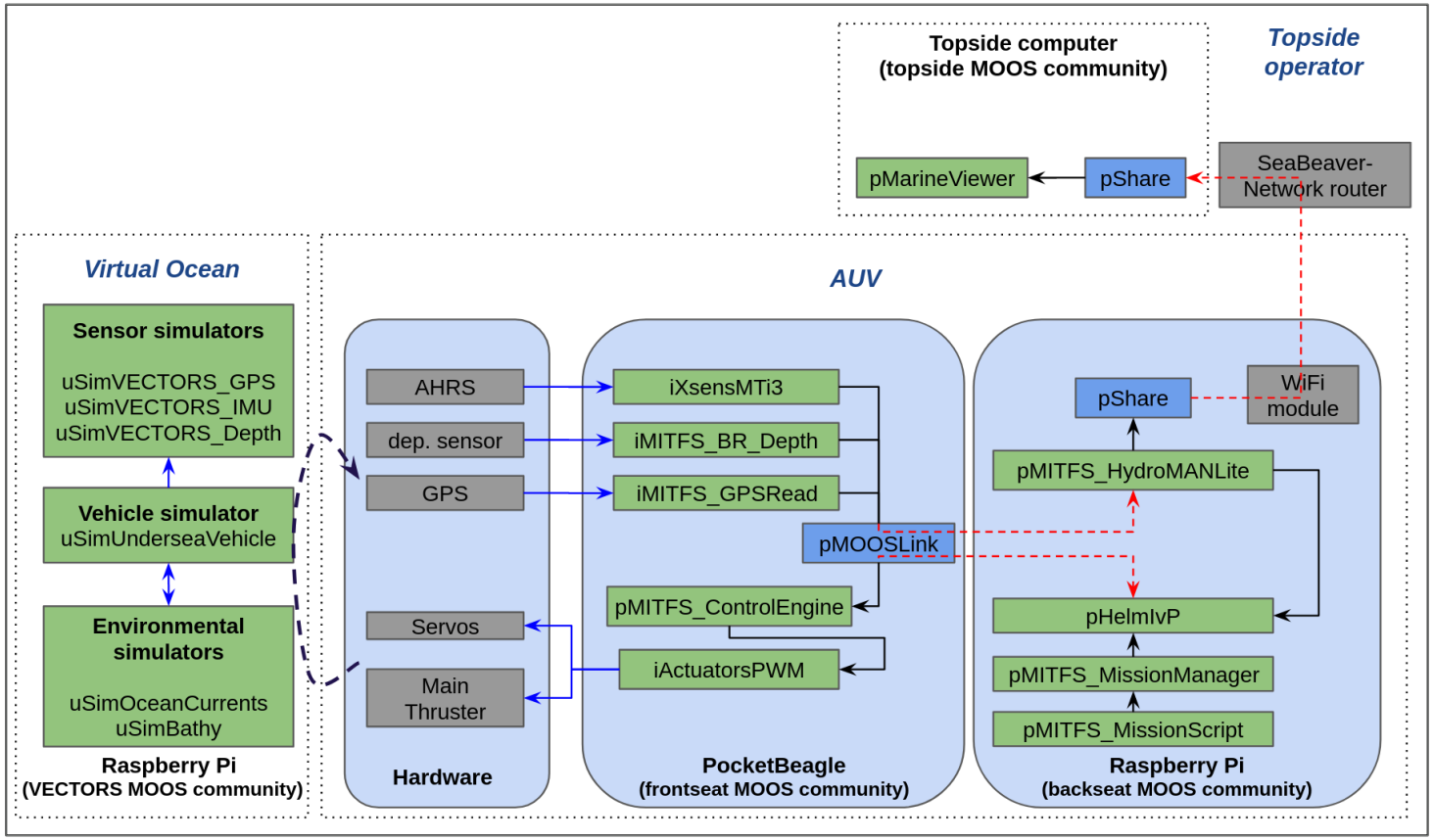
A virtual ocean!

- In-water field experiments are expensive (in terms of time and cost), and mistakes can be catastrophic.

- Risk and cost can be minimized by:
  - Extensively testing hardware
  - Extensively testing software
  - Testing both software and hardware together before the actual in-water experiment

- The virtual ocean simulates three key components:
  - The 6-degrees-of-freedom dynamics of the AUV
  - The dynamics of the surrounding ocean environment
  - The dynamics of the sensors

- SeaBeaver AUVs currently use a virtual ocean called VECTORS (Virtual Environment for Construction and Testing of Oceanic Robotics Systems)
  - A separate MOOS community named VECTORS

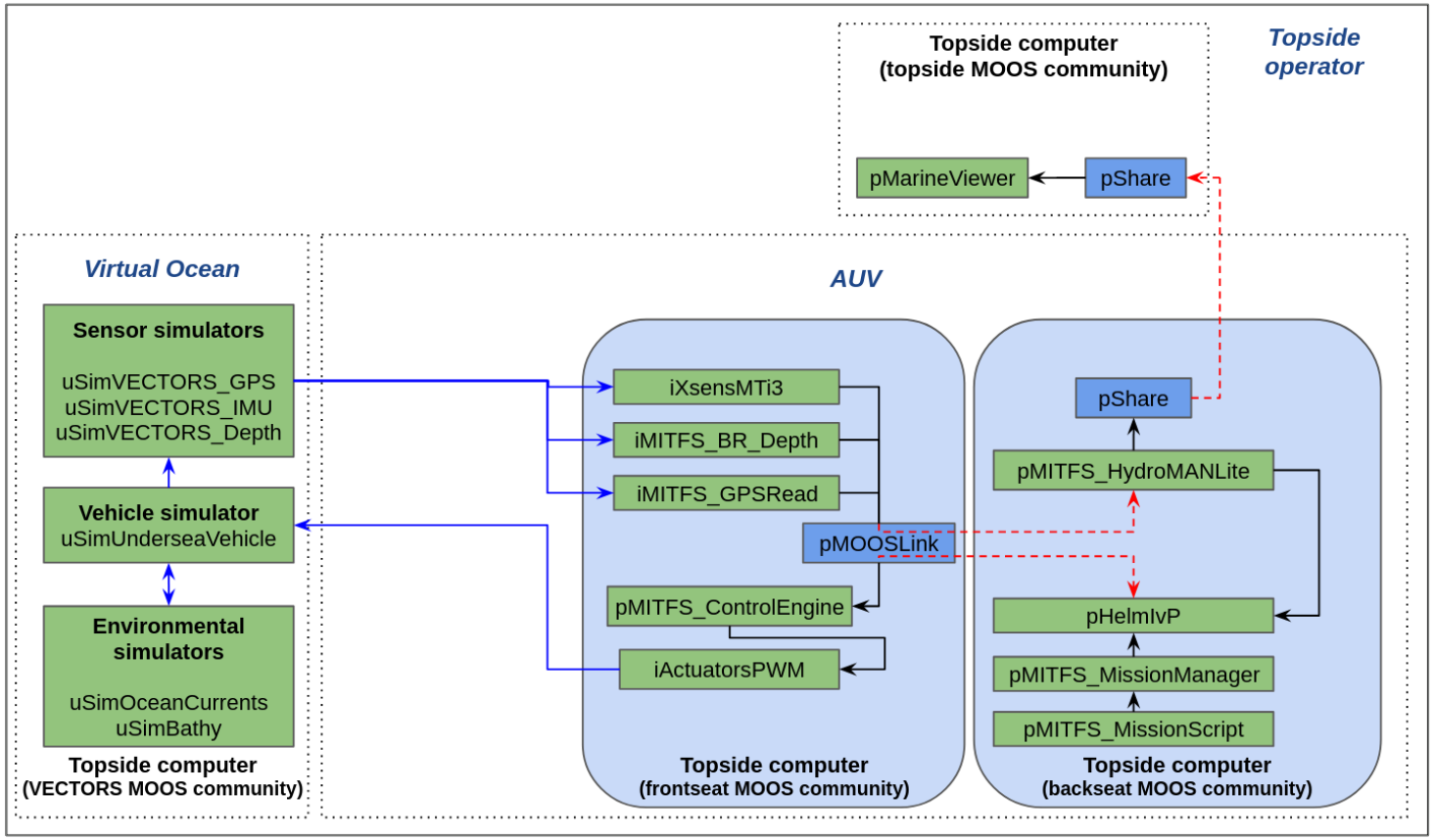


Software-in-the-loop (SITL) simulations

- SITL simulations allows us to rapidly test the actual embedded software components:
  - Simulations can be conducted faster than real-time (i.e., by using MOOS time warping)

- In this mode, four MOOS communities are launched on the topside laptop:
  - Frontseat MOOS community
  - Backseat MOOS community
  - Topside MOOS community
  - VECTORS MOOS community

- Hardware components (including embedded computers) are excluded from the test



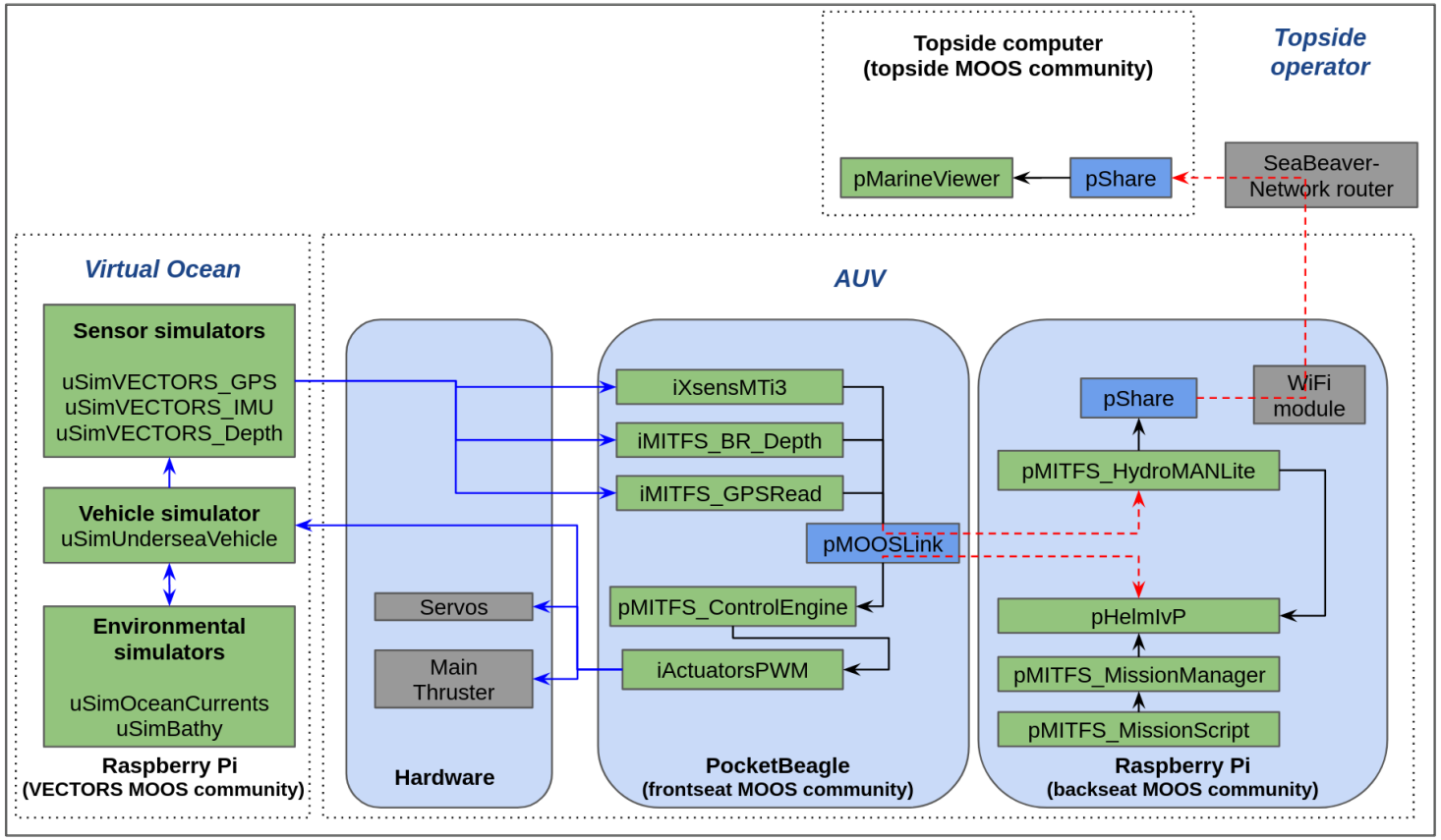
Hardware-in-the-loop (HITL) simulations

- HITL simulations allow us to test most of the hardware components along with the actual embedded software.

- Four MOOS communities are launched:
  - Frontseat MOOS community in the PocketBeagle
  - Backseat MOOS community in the Raspberry Pi
  - Topside MOOS community in the topside laptop
  - VECTORS MOOS community in the Raspberry Pi

- The hardware components that are being tested include:
  - Embedded computers (e.g. we will know if the processing power of the computers are not sufficient)
  - Actuators

- However, most sensors are excluded from the test:
  - GPS
  - AHRS
  - Depth sensor



End