Payload UUV Autonomy
(3 Architecture Principles)

Architectural Principle #1
Payload Autonomy

Decouple the Procurement of Hardware and Software
Payload UUV Autonomy
(3 Architecture Principles)

Payload Computer

Main Vehicle Computer

MOOS Middleware
MOOS Applications

IvP Helm

Behavior-Based
Modular HELM

IvP Helm Overview

Michael Benjamin, Henrik Schmidt, © 2018
MIT Dept of Mechanical Engineering
The IvP Helm

- The IvP Helm is a MOOS App, known as pHelmIVP
- The IvP Helm works with other MOOS Apps, performing sensor-processing, planning, communications.

The IvP Helm Execution Loop

1. Mail is read in the MOOS OnNewMail() function and applied to a local buffer.
2. The helm mode is determined, and a set of running behaviors determined.
3. Behaviors do their thing – posting MOOS variables and an IvP function.
4. Competing behaviors are resolved with the IvP solver.
5. The Helm decision and any behavior postings are published to the MOOSDB.
Mail is read in the MOOS OnNewMail() function and applied to a local buffer.  
The helm mode is determined, and set of running behaviors determined.  
Behaviors do their thing – posting MOOS variables and an IvP function.  
Competing behaviors are resolved with the IvP solver.  
The Helm decision and any behavior postings are published to the MOOSDB.
Example: Waypoint Behavior

- Start-Up Configuration Parameters
- Run-Time Input:
  - Sensor Values
  - Human Commands
- IvP Functions
- MOOS Postings

Behaviors Generally

- Configuration Parameters
  - List of waypoints
  - Desired speed
  - Capture radius
- Run-Time Input:
  - Sensor Values
  - Human Commands
- IvP Functions
- MOOS Postings
Behaviors Generally

- List of waypoints
- Desired speed
- Capture radius

Configuration Parameters

Run-Time Input

NAV_X = 237.98
NAV_Y = 1900.0

IvP Functions

MOOS Postings

---

Behaviors Generally

- List of waypoints
- Desired speed
- Capture radius

Configuration Parameters

Run-Time Input

NAV_X = 237.98
NAV_Y = 1900.0

IvP Functions

MOOS Postings

WPT_INDEX = 2
TRAVERSALS = 7
Competing Objective Functions

• An example of competing behaviors (1) Transiting and (2) Collision Avoidance

Competing Objective Functions

• Each vehicle is running two behaviors
• Each produces its own objective function

Transiting Objective Function

Collision Avoidance Objective Function
Competing Objective Functions

- Each vehicle is running two behaviors
- Each produces its own objective function

\[
\begin{align*}
\text{Behavior 1:} & \quad f_1(x_1, x_2, \ldots, x_n) \\
\text{Behavior 2:} & \quad f_2(x_1, x_2, \ldots, x_n) \\
\text{Behavior 3:} & \quad f_3(x_1, x_2, \ldots, x_n)
\end{align*}
\]

\[
x^* = \arg\max_x \sum_{i=1}^{k} (w_i \cdot f_i(x))
\]

Action

IvP Solver
Competing Objective Functions

\[ x^* = \arg\max_x \sum_{i=1}^{k} (w_i \cdot f_i(x)) \]

Interval Programming

- IvP is Interval Programming
- It is a format for representing objective functions
- It is a solver that capitalizes on that format – fast, globally optimal

IvP Functions are piecewise linear

Piece distribution need not be uniform
The Alpha Mission

To launch yourself:

```
$ cd moos-ivp/ivp/missions/s1_alpha
$ ./launch.sh 10
```
Alpha Mission Has Two Behaviors

Three questions discussed next:

• How is this mission configured?
• What initiates this mission?
• How does the helm transition to return?
Configuring the Helm for a Mission

A mission is configure with two files:

- **alpha.moos** – configures all MOOS apps, including general helm parameters
- **alpha.bhv** – configures all Helm behaviors

To launch yourself:

```
$ cd moos-ivp/ivp/missions/s1_alpha
$ ./launch.sh 10
```

### Behavior Files

Helm configuration file structure:

```
Behavior = <behavior_name>
{
  parameter = value
  . . .
  parameter = value
}
```

Helm configuration file structure: **file.bhv**

- **Variable Initializations**
- **Behavior Configuration**

Michael Benjamin, Henrik Schmidt, © 2018  
MIT Dept of Mechanical Engineering
### Alpha Mission Behavior File

```plaintext
initialize: DEPLOY = false
initialize: RETURN = false

Behavior = BHV_Waypoint
{
  name = waypt_survey
  pwt = 100
  condition = RETURN = false
  condition = DEPLOY = true
  endflag = RETURN = true

  speed = 4
  capture_radius = 5.0
  slip_radius = 15.0
  polygon = 60,-40:60,-160:150,-160:180,-100:150,-100:150,-40
  repeat = 1
}

Behavior = BHV_Waypoint
{
  name = waypt_return
  pwt = 100
  condition = RETURN = true
  condition = DEPLOY = true
  endflag = DEPLOY = false

  speed = 2.0
  capture_radius = 2.0
  slip_radius = 8.0
  points = 0,-2
}
```

### Helm Initial MOOSDB Pokes

```plaintext
initialize: DEPLOY = false
initialize: RETURN = false

Behavior = BHV_Waypoint
{
  name = waypt_survey
  pwt = 100
  condition = RETURN = false
  condition = DEPLOY = true
  endflag = RETURN = true

  speed = 4
  capture_radius = 5.0
  slip_radius = 15.0
  polygon = 60,-40:60,-160:150,-160:180,-100:150,-100:150,-40
  repeat = 1
}

Behavior = BHV_Waypoint
{
  name = waypt_return
  pwt = 100
  condition = RETURN = true
  condition = DEPLOY = true
  endflag = DEPLOY = false

  speed = 2.0
  capture_radius = 2.0
  slip_radius = 8.0
  points = 0,-2
}
```

When pHelmIVP launches, it will write to the MOOSDB:

- DEPLOY = false
- RETURN = false
Three Architectures

IvP Helm

Overview

Alpha Mission

Behavior Files

Behavior Conditions

Behavior States

Behavior Flags

Michael Benjamin, Henrik Schmidt, © 2018

MIT Dept of Mechanical Engineering

Behavior Types vs. Names

alpha.bhv file

initialize: DEPLOY = false
initialize: RETURN = false

Behavior = BHV_Waypoint
{name = waypt_survey
pwt = 100
condition = RETURN = false
condition = DEPLOY = true
endflag = RETURN = true

speed = 4

capture_radius = 5.0
slip_radius = 15.0
polygon = 60,-40:60,-160:150,-160:180,-100:150,-40
repeat = 1
}

Behavior = BHV_Waypoint
{name = waypt_survey
pwt = 100
condition = RETURN = true
condition = DEPLOY = true
endflag = DEPLOY = false

speed = 2.0

capture_radius = 2.0
slip_radius = 8.0
point = 0,-2
}

Both behaviors are the same type.

Each behavior has a unique name.

Waypoint Behavior Points

alpha.bhv file

initialize: DEPLOY = false
initialize: RETURN = false

Behavior = BHV_Waypoint
{name = waypt_survey
pwt = 100
condition = RETURN = false
condition = DEPLOY = true
endflag = RETURN = true

speed = 4

capture_radius = 5.0
slip_radius = 15.0
polygon = 60,-40:60,-160:150,-160:180,-100:150,-40
repeat = 1
}

Behavior = BHV_Waypoint
{name = waypt_survey
pwt = 100
condition = RETURN = true
condition = DEPLOY = true
endflag = DEPLOY = false

speed = 2.0

capture_radius = 2.0
slip_radius = 8.0
point = 0,-2
}

The waypoint behavior accepts either:
- a polygon
- a single point

Michael Benjamin, Henrik Schmidt, © 2018

MIT Dept of Mechanical Engineering
Behavior Conditions

- Each condition involves one or more MOOS variables
- A behavior may have more than one condition
- If multiple conditions, all conditions need to be satisfied.

Example:

```
condition = RETURN = false
condition = DEPLOY = true
```

- Both RETURN and DEPLOY are MOOS variables
- Both are of type string (not double)
- The condition is true if the current variable value matches the string
Behavior Conditions

- Each condition involves one or more MOOS variables
- A behavior may have more than one condition
- If multiple conditions, all conditions need to be satisfied.

```plaintext
initialize DEPLOY = false
initialize RETURN = false

Behavior = BHV_Waypoint
{
    name = waypt_survey
    pwt = 100
    condition = RETURN = false
    condition = DEPLOY = true
    endflag = RETURN = true

    speed = 4
    capture_radius = 5.0
    slip_radius = 15.0
    polygon = 60,-40:60,-160:150,-160:180,-100:150,-40
    repeat = 1
}
```

The Helm Information Buffer

- The helm maintains an information buffer, a cache of MOOS Variable Values
- It is updated by reading MOOS mail on each iterate loop
- Behavior Conditions are checked against this buffer
The Helm Information Buffer

- The helm maintains an information buffer, a cache of MOOS Variable Values
- It is updated by reading MOOS mail on each iterate loop
- Behavior Conditions are checked against this buffer

Behavior Logic Conditions

Simple logic condition with one component

- \( \text{condition} = \text{RETURN} = \text{false} \)
  - true if the MOOS variable \( \text{RETURN} \) has the string value “false”

- \( \text{condition} = \text{DEPLOY} \neq \text{true} \)
  - true if the MOOS variable \( \text{DEPLOY} \) has a string value other than “true”

WARNING: this condition is fail if the MOOS variable \( \text{DEPLOY} \) has never been written to.
Disjunctive (OR) Logic Conditions

A logic condition may have more than one component

\[
\text{condition} = ((\text{RETURN} = \text{false}) \text{ or } (\text{DEPLOY} \neq \text{true}))
\]

True if

- the MOOS variable RETURN has the string value “false”, OR
- the MOOS variable DEPLOY has a string value other than “true”

WARNING: this condition will fail if the MOOS variable DEPLOY has never been written to – even if the first component (RETURN = false) is true

Simple Example: “Double Loiter”

Mission Synopsis:
Upon receiving a deploy command, transit to and loiter at region A for a fixed duration and then to region B. Periodically switch between regions until recalled home.

```
Behavior = BHV_Loiter
{
    name = loiter_a
    condition = (DEPLOY=true) and (REGION=A) and (RETURN=false)
    speed = 1.8
    radius = 4.0
    polygon = format=radial,x=0,y=-75,radius=40,pts=8
}

Behavior = BHV_Loiter
{
    name = loiter_b
    condition = (DEPLOY=true) and (REGION=B) and (RETURN=false)
    speed = 1.8
    radius = 4.0
    polygon = format=radial,x=160,y=-75,radius=40,pts=8
}

Behavior = BHV_Return
{
    name = return
    condition = (DEPLOY=true) and (RETURN=true)
    speed = 1.8
    radius = 4.0
    point = 80,40
}
```
Simple Example: “Double Loiter”

Mission Synopsis:
Upon receiving a deploy command, transit to and loiter at region A for a fixed duration and then to region B. Periodically switch between regions until recalled home.

Launch and return position

Simple Example: “Double Loiter”

Mission Synopsis:
Upon receiving a deploy command, transit to and loiter at region A for a fixed duration and then to region B. Periodically switch between regions until recalled home.
Simple Example: “Double Loiter”

Mission Synopsis:
Upon receiving a deploy command, transit to and loiter at region A for a fixed duration and then to region B. Periodically switch between regions until recalled home.

```
set MODE = ACTIVE {
  DEPLOY = true
} INACTIVE

set MODE = RETURNING {
  MODE = ACTIVE
  RETURN = true
}

set MODE = LOITER_A {
  MODE = ACTIVE
  REGION = A
} LOITER_B
```

Launch and return position

Launch and return position

Launch and return position

Launch and return position

Launch and return position

Launch and return position

Launch and return position

Launch and return position
Simple Example: “Double Loiter”

Mission Synopsis:
Upon receiving a deploy command, transit to and loiter at region A for a fixed duration and then to region B. Periodically switch between regions until recalled home.

```
set MODE = ACTIVE {
    DEPLOY = true
} INACTIVE
set MODE = RETURNING {
    MODE = ACTIVE
    RETURN = true
}
set MODE = LOITER_A {
    MODE = ACTIVE
    REGION = A
} LOITER_B
```

Question: Why define the “Active” mode? Why not just have:

```
LOITER_A
LOITER_B
RETURNING
```

Mission Synopsis:
Upon receiving a deploy command, transit to and loiter at region A for a fixed duration and then to region B. Periodically switch between regions until recalled home.
Simple Example: “Double Loiter”

Mission Synopsis:
Upon receiving a deploy command, transit to and loiter at region A for a fixed duration and then to region B. Periodically switch between regions until recalled home.

```plaintext
set MODE = ACTIVE {
  DEPLOY = true
} INACTIVE
set MODE = RETURNING {
  MODE = ACTIVE
  RETURN = true
}
set MODE = LOITER_A {
  MODE = ACTIVE
  REGION = A
} LOITER_B
```

Mission Synopsis:
Upon receiving a deploy command, transit to and loiter at region A for a fixed duration and then to region B. Periodically switch between regions until recalled home.

MIT Prototype Autonomy Modes

IVP-Helm Hierarchical Mode Declarations

Launch
AllStop
Mission
Diving
Surfacing
Deploy
Return
Prosecute
Detection
Tracking
Recovery
Hold-at-Risk
Area Clearance
Low Power Alert
Communicating
Helical
Elevator
Transit
Anchored
Elevator
Detection
Hold-at-Risk
Area Clearance
Low Power Alert
Communicating
Helical
Elevator
Transit
Anchored
Elevator
Behaviors may be in one of four states:

- **Idle**: a behavior has not met its run condition, as defined by the `condition` parameter.
- **Running**: a behavior has met its run conditions
- **Active**: a behavior is running state and is producing an objective function
- **Completed**: completion is specific to a behavior, or may be due to a `duration` timeout defined generally for all behaviors.
Active vs. Running States

The running state: behavior has met its run conditions.
The active state: behavior is running and producing an objective function.

The helm’s primary job is to produce a helm decision. A behavior is participating in that decision only if it is producing an objective function.

A behavior may participate in the helm decision based on:

1. The run conditions (mostly dependent on an external decision process)
2. The behavior’s own logic (a local decision based on a more nuanced understanding of the situation).

Active vs. Running States

What is the difference between running and active?

The running state: behavior has met its run conditions.
The active state: behavior is running and producing an objective function.

The helm’s primary job is to produce a helm decision. A behavior is participating in that decision only if it is producing an objective function.

The choice to participate in the helm decision is made at two points:

1. The run conditions (mostly dependent on an external decision process)
2. The behavior’s own logic (a local decision based on a more nuanced / domain-expert understanding of the situation).
Behavior Flags

- Flags are MOOS Pokes triggered by behavior state
- They are mission configuration parameters (not behavior source code)
- They are critical tools for structuring a mission
Behavior Flags

- Flags are MOOS Pokes triggered by behavior state
- They are mission configuration parameters (not behavior source code)
- They are critical tools for structuring a mission

**Flags**

- **idleflag**: posted when the behavior is in the idle state.
- **runflag**: posted when the behavior is in the running (or active) state.
- **endflag**: posted when the behavior completes.
- **activeflag**: posted when the behavior is in the active state.
- **inactiveflag**: posted when the behavior is not in the active state.

**End Flags**

- End Flags are posted when a behavior completes
- An endflag may trigger the condition of another behavior
- Alpha mission as an example. The end of the survey behavior triggers the start of the return behavior.
Alpha Mission End Flag Example

**End Flag Example**

1. **survey waypoints completes**
2. **endflags posted**
   - `RETURN=true`
3. **return waypoint behavior begins**

**Behavior = BHV_Waypoint**

```plaintext
Behavior = BHV_Waypoint
{
  name = waypt_survey
  pwt = 100
  condition = RETURN = false
  condition = DEPLOY = true
  endflag = RETURN = true
  speed = 4
  capture_radius = 5.0
  slip_radius = 15.0
  polygon = 60,-40:60,-160:150,-160:180,-100:150,-40
  repeat = 1
}
```

**Behavior = BHV_Waypoint**

```plaintext
Behavior = BHV_Waypoint
{
  name = waypt_return
  pwt = 100
  condition = RETURN = true
  condition = DEPLOY = true
  endflag = DEPLOY = false
  speed = 2.0
  capture_radius = 2.0
  slip_radius = 8.0
  point = 0,-2
}
```

**survey waypoints completes**

**END**
Behavior Completion

Behaviors states:

- Idle
- Running
- Active
- Completed

**Completion** is defined by the behavior. For example:
- A *waypoint* behavior completes when it has visited all its waypoints.
- A *loiter* behavior never completes.

Even behaviors that don’t normally *complete*, may complete when configured with a prescribed *duration*, e.g., duration=60 // seconds

By default, a completed behavior simply ceases to exist once it is completed. No chance for participation ever again in the helm.

Unless… the behavior is configured with *perpetual=true*. 