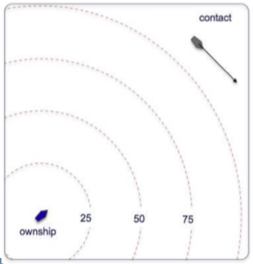
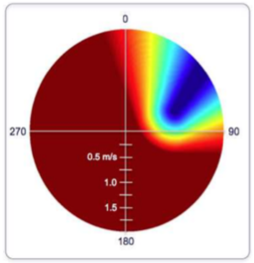
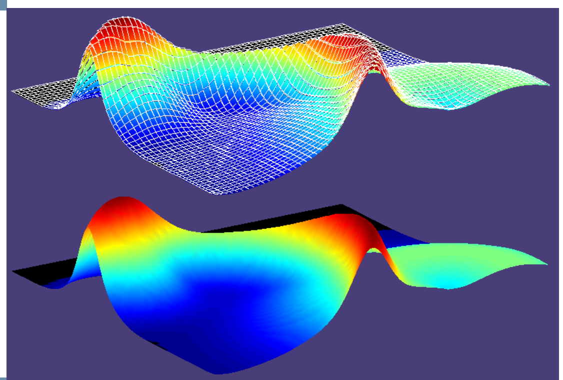




Writing Behaviors for the IvPHelm – Basic Overview and Summary of Tools



Michael R. Benjamin

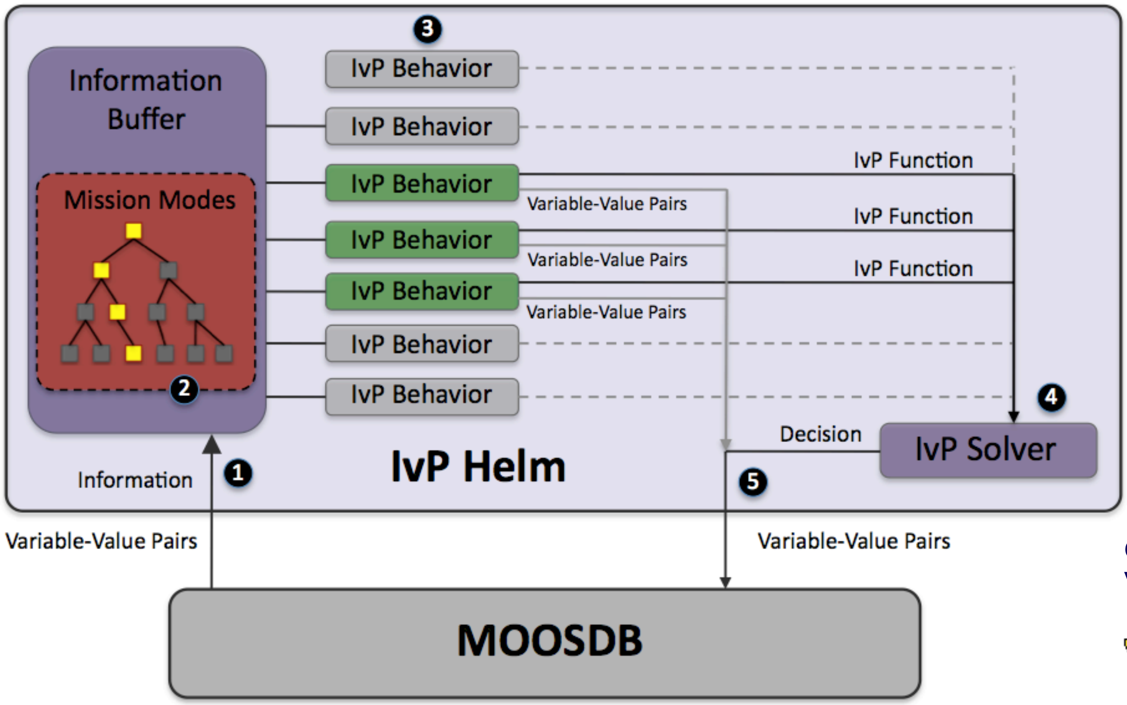
Dept. of Mechanical Engineering, MIT
Computer Science and Artificial Intelligence Lab

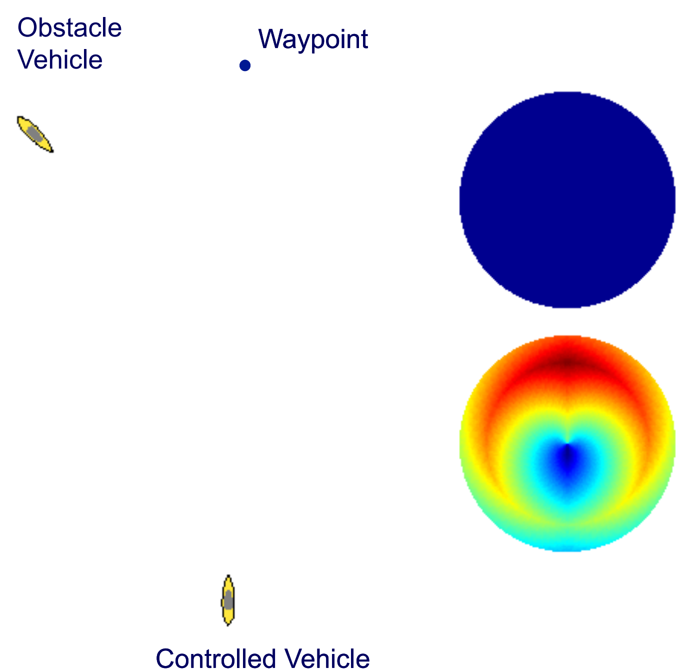
mikerb@csail.mit.edu

Outline

- The IvP Behavior Interface
- Writing Your First Behavior and Augmenting the Helm
- Overview of IvP Functions
- The Reflector Tool
- The ZAIC Tool
- Rendering IvP Functions

How Behaviors Fit in the Helm

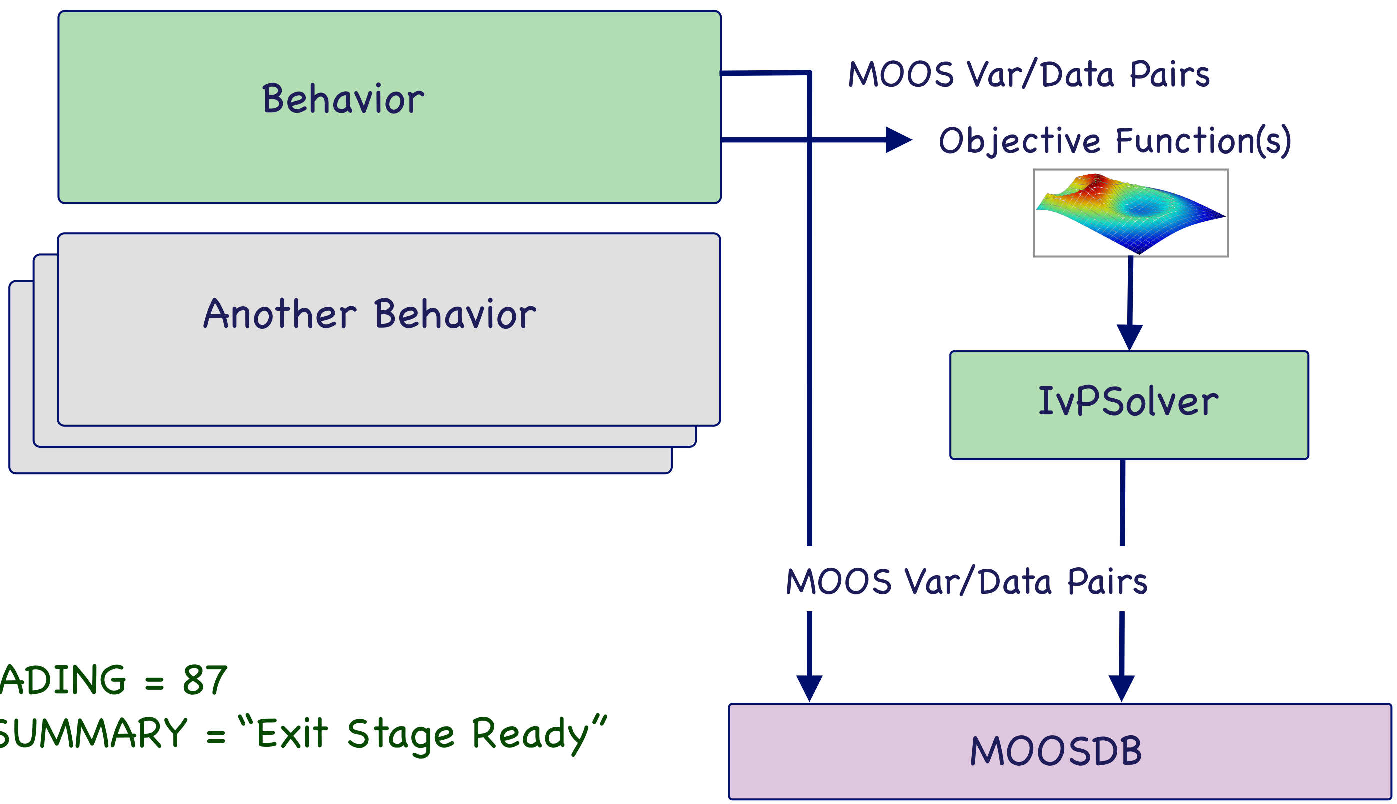




1. Mail is read.
2. Helm mode is determined.
3. Behaviors generate their output.
4. Competing behaviors are resolved.
5. The Helm posts its results

Behavior Output

- Each behavior produces two kinds of output (to the helm):
  (1) MOOS Var/Data Pairs
  (2) Objective Function(s)
- The Helm produces one kind of output – MOOS Var/Data Pairs.
  (It is afterall simply one other MOOS application)



For Example:

DESIRED_HEADING = 87
PROGRESS_SUMMARY = "Exit Stage Ready"

Behavior Run Conditions

- Conditions are set in the helm configuration for each behavior.

  condition = COLLISION_AVOIDANCE == true

  condition = STATION_KEEP == true

  condition = (DEPLOY==true) and (SURVEYING == false)

Example: The Loiter behavior in the Charlie Example Mission

```
Behavior = BHV_Loiter
{
  name       = loiter
  priority   = 100
  condition = MODE==LOITERING

        speed = 1.3
     clockwise = false
        radius = 4.0
     nm_radius = 25.0
       polygon = format=radial, x=0, y=-75, radius=40, pts=6
}
```

Behavior Run Conditions Determine Behavior Run States

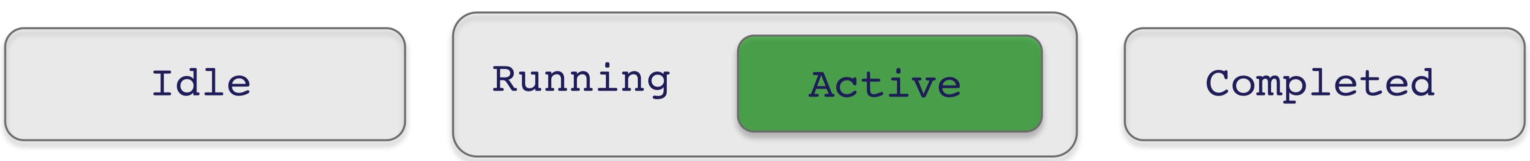
- Upon each helm iteration the behaviors' *run-state* is evaluated.

  (1) Running (conditions have been met)

  (2) Active (A running behavior that produces an objective function)

  (3) Idle (conditions have NOT been met)

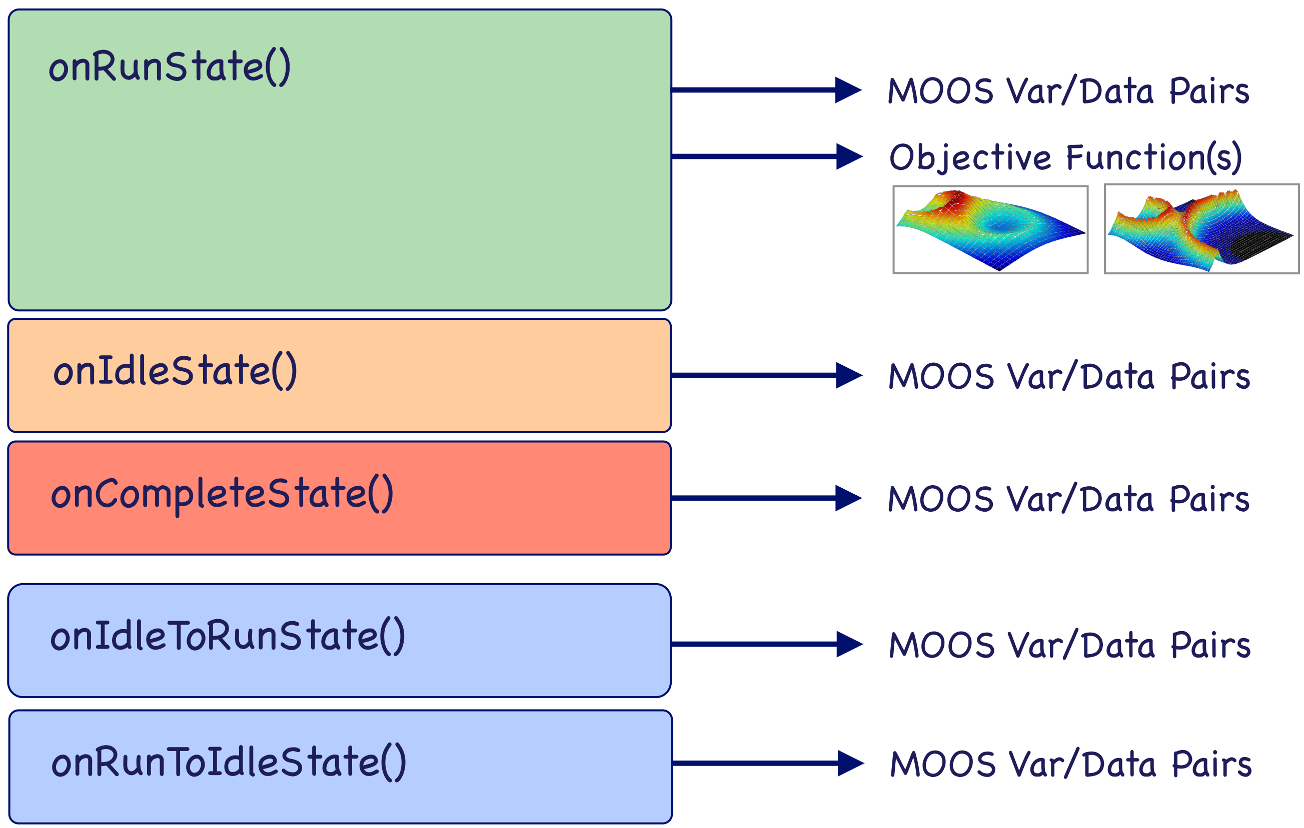
  (4) Completed (A behavior that completed previously)



Behavior State and Behavior Output

On each iteration of the helm, the helm operates on each behavior:

(1) Determines the vehicle state (whether its run conditions have been met).
(2) Depending on the state (and state on previous iteration) calls one or more standard functions:

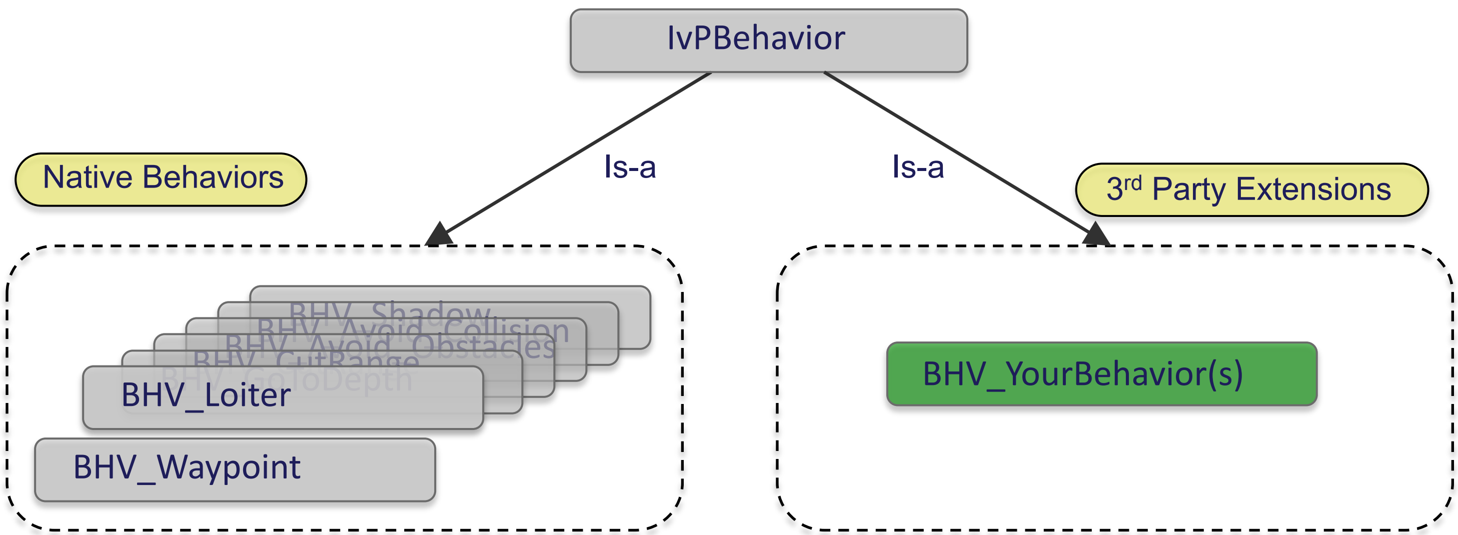


Outline

- The IvP Behavior Interface
- Writing Your First Behavior and Augmenting the Helm
- Overview of IvP Functions
- The Reflector Tool
- The ZAIC Tool
- Rendering IvP Functions

Extending the Helm's Behaviors

- All behaviors are a subclass of the IvP parent class.
  (Just like all MOOS apps are a subclass of the MOOSApp parent class.)



- Two Issues in Adding New Behaviors:
  (1) What's involved in building one?
  (2) How to augment the helm and use it once its built?

  Extending a MOOS-IvP Autonomy System and Users Guide to the IvPBuild Toolbox
  MIT CSAIL Technical Report TR-2009-37

Behavior Overloaded Functions

- The IvPBehavior has six key virtual functions.
- The primary work of the behavior author is to overload these functions with the functionality unique to the author's design.

IvPBehavior

```
virtual BehaviorReport onRunState()
virtual void onIdleState()
virtual void onComplete()
virtual void onRunToIdleState()
virtual void onIdleToRunState()
virtual bool setParam(parameter, value)
```

Subclass

BHV_YourBehavior

```
BehaviorReport onRunState()
void onIdleState()
void onComplete()
void onRunToIdleState()
void onIdleToRunState()
bool setParam(parameter, value)
```

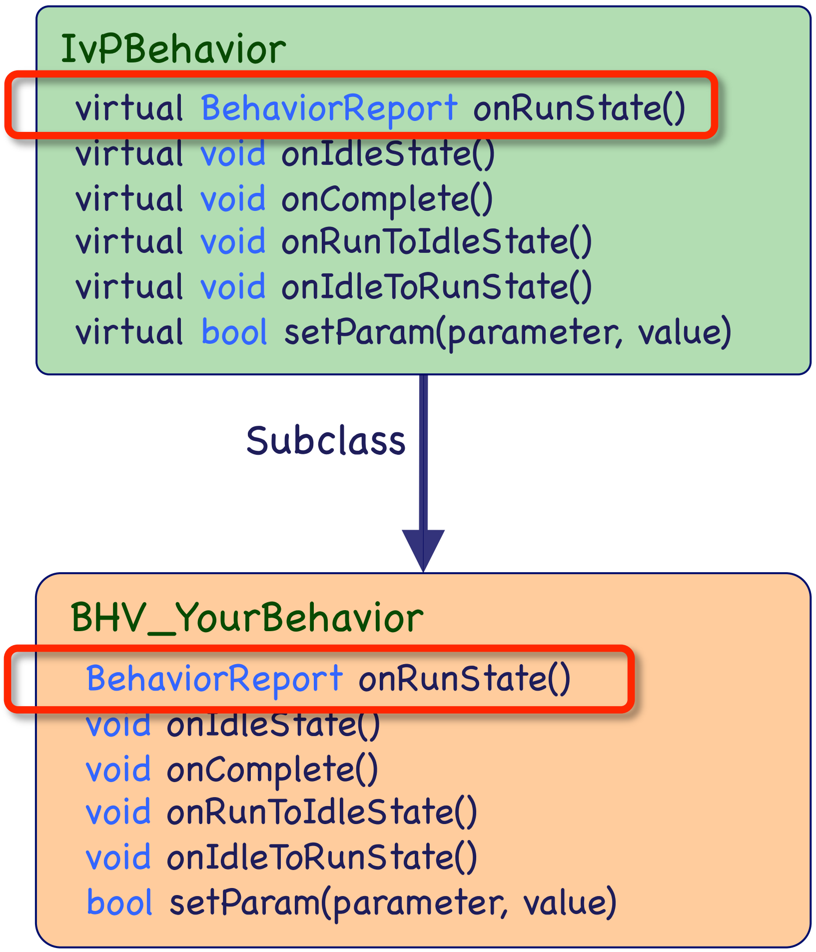
- These are not "pure" virtual functions
- Behavior author has the option to not implement them. Defaults exist.

BHV_YourBehavior

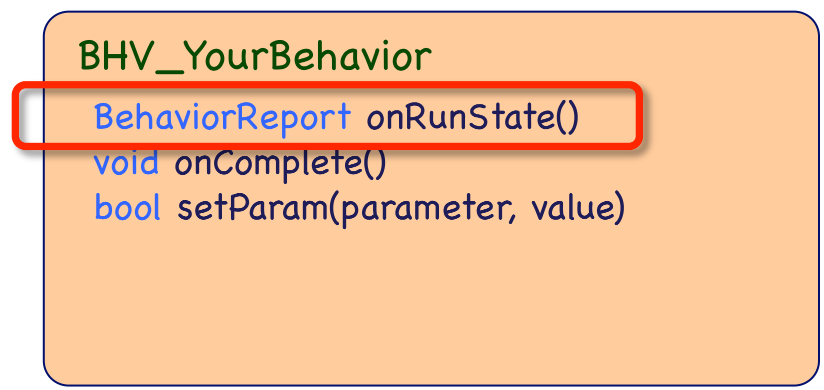
```
BehaviorReport onRunState()
void onComplete()
bool setParam(parameter, value)
```

Behavior Overloaded Functions

- The IvPBehavior has six key virtual functions.
- The primary work of the behavior author is to overload these functions with the functionality unique to the author's design.



- These are not "pure" virtual functions
- Behavior author has the option to not implement them. Defaults exist.



The OnRunState() Function

- The "onRunState()" function holds the primary implementation of the behavior.
- It builds and returns an IvPFunction (Objective Function) to the helm.

```
IvPBehavior
  virtual BehaviorReport onRunState()
  virtual IvPFunction* onRunState()
  virtual void onIdleState()
  virtual void onComplete()
  virtual void onRunToIdleState()
  virtual void onIdleToRunState()
  virtual bool setParam(parameter, value)
```

- An objective function is a mapping from possible helm decisions to a utility value.

$$f(x_1, \ldots, x_3) = \text{utility}$$

$$f(\text{heading}, \text{speed}, \text{depth}) = \text{utility}$$

- An IvP objective function is objective function of a particular form.

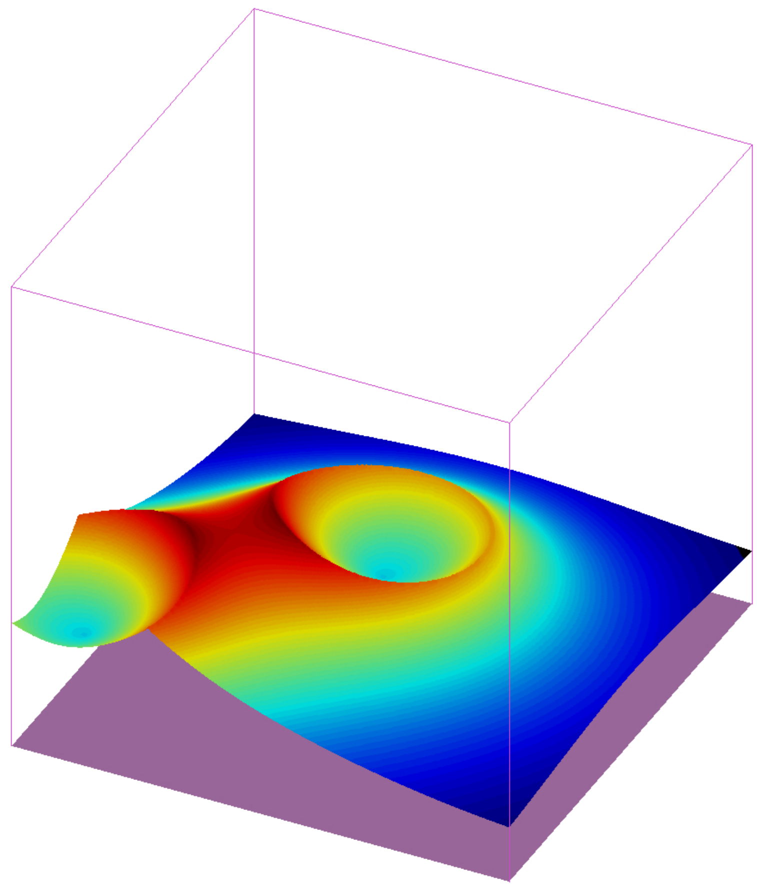
Outline

- The IvP Behavior Interface
- Writing Your First Behavior and Augmenting the Helm
- Overview of IvP Functions
- The Reflector Tool
- The ZAIC Tool
- Rendering IvP Functions

Objective Functions and IvP Functions

An *objective function* is a function where the domain is "decision space", and the range represents utility to the decision-maker's goals or objectives.

Underlying Function

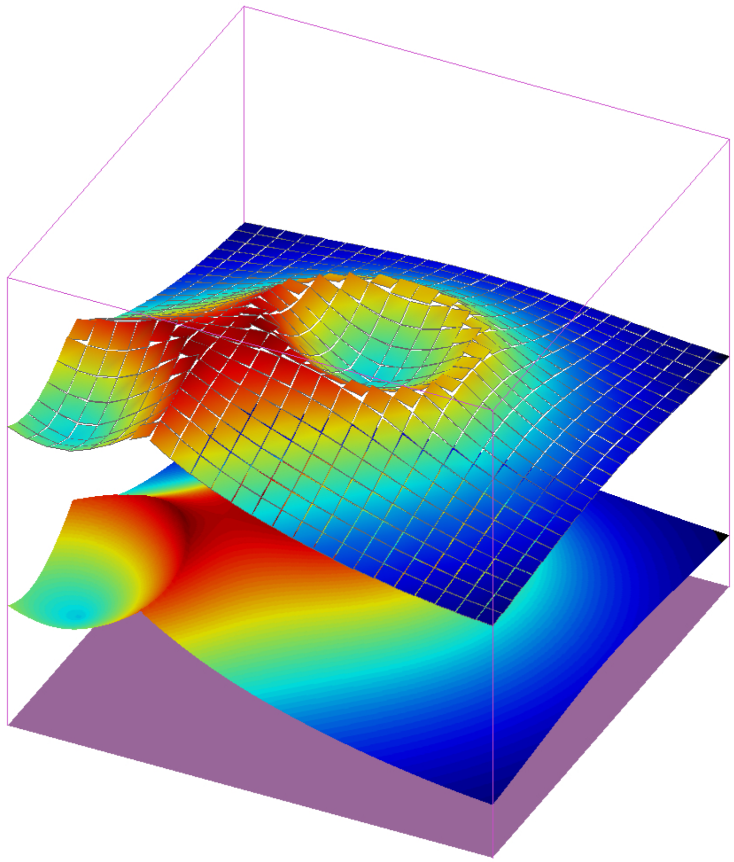


$$f_i(x,y) = ((1 - \frac{|\sqrt{(x-250)^2 + (y-250)^2} - 100|}{2500})^8 * 200) - 100, + ((1 - \frac{|\sqrt{(x-50)^2 + (y-50)^2} - 100|}{2500})^8 * 200) - 100$$

Objective Functions and IvP Functions

An *IvP function* is a piecewise linear approximation of an objective function, over a discrete decision space.

Piecewise Linear Approximation 525 Pieces

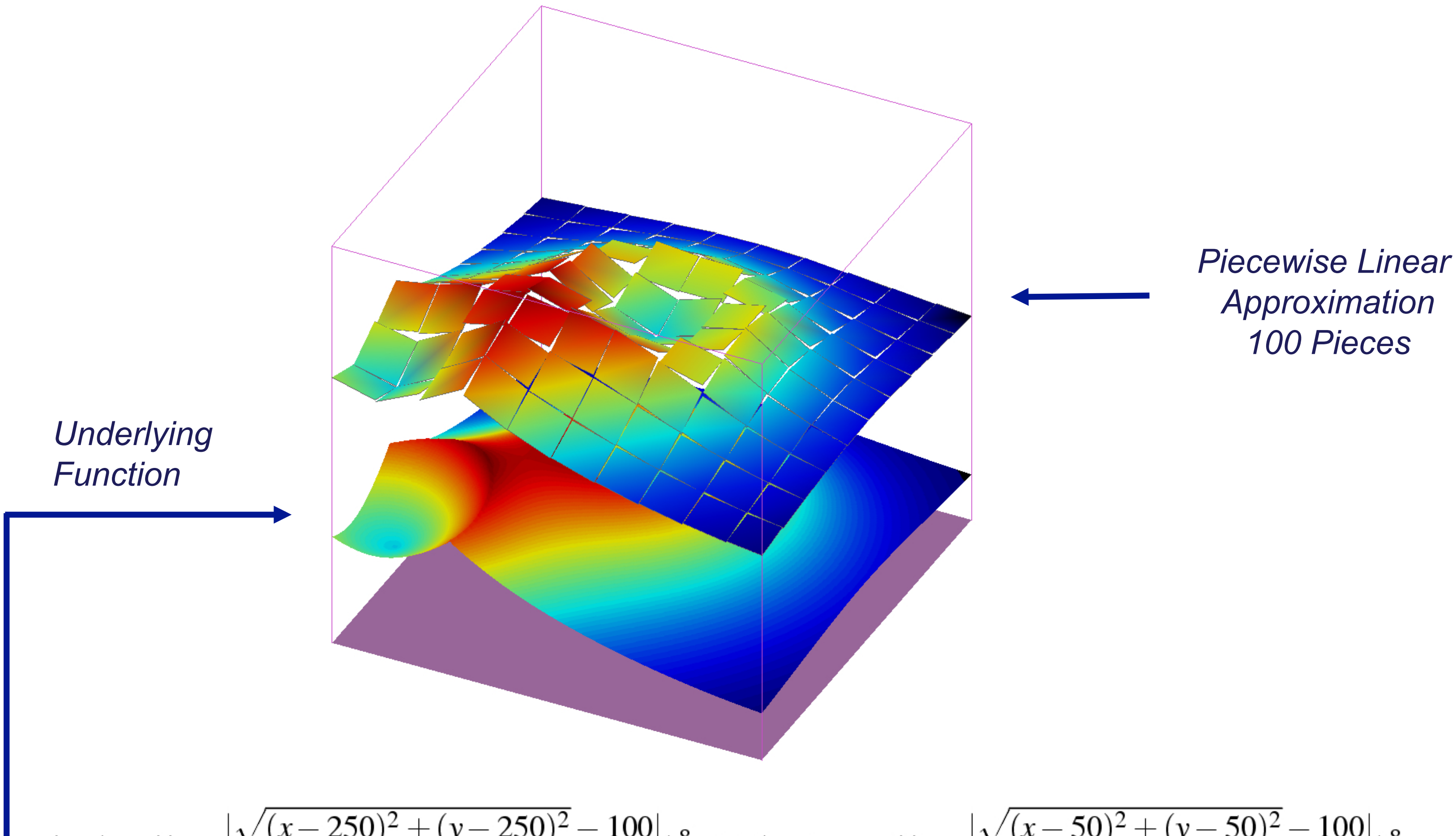


Underlying Function

$$f_i(x,y) = ((1 - \frac{|\sqrt{(x-250)^2 + (y-250)^2} - 100|}{2500})^8 * 200) - 100, + ((1 - \frac{|\sqrt{(x-50)^2 + (y-50)^2} - 100|}{2500})^8 * 200) - 100$$

Objective Functions and IvP Functions

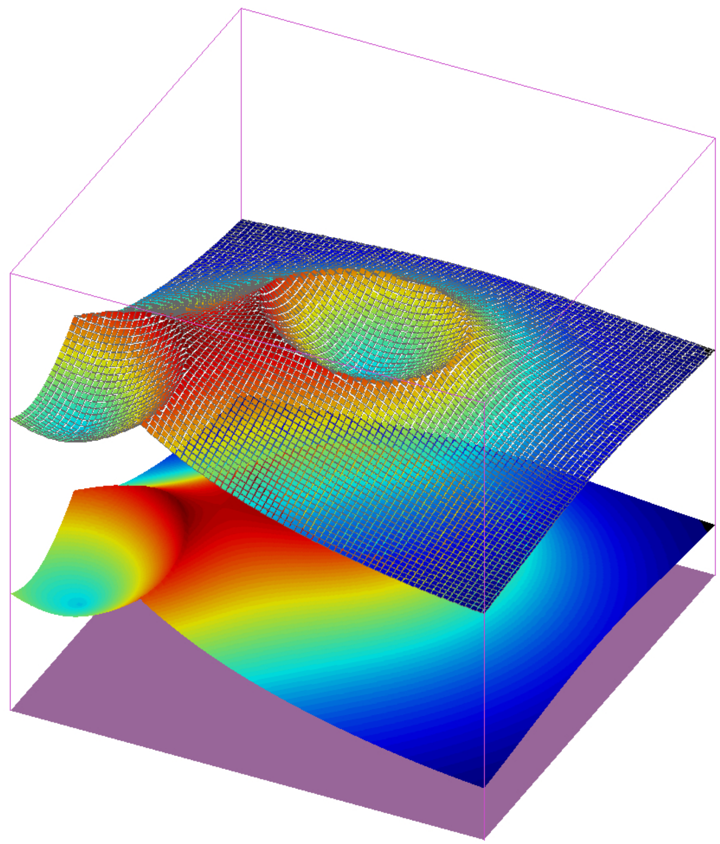
An *IvP function* is a piecewise linear approximation of an objective function, over a discrete decision space.



$$f_i(x,y) = ((1 - \frac{|\sqrt{(x-250)^2 + (y-250)^2} - 100|}{2500})^8 * 200) - 100, +((1 - \frac{|\sqrt{(x-50)^2 + (y-50)^2} - 100|}{2500})^8 * 200) - 100$$

Objective Functions and IvP Functions

An *IvP function* is a piecewise linear approximation of an objective function, over a discrete decision space.



Piecewise Linear Approximation 10,000 Pieces

Underlying Function

$$f_i(x,y) = ((1 - \frac{|\sqrt{(x-250)^2 + (y-250)^2} - 100|}{2500})^8 * 200) - 100, +((1 - \frac{|\sqrt{(x-50)^2 + (y-50)^2} - 100|}{2500})^8 * 200) - 100$$

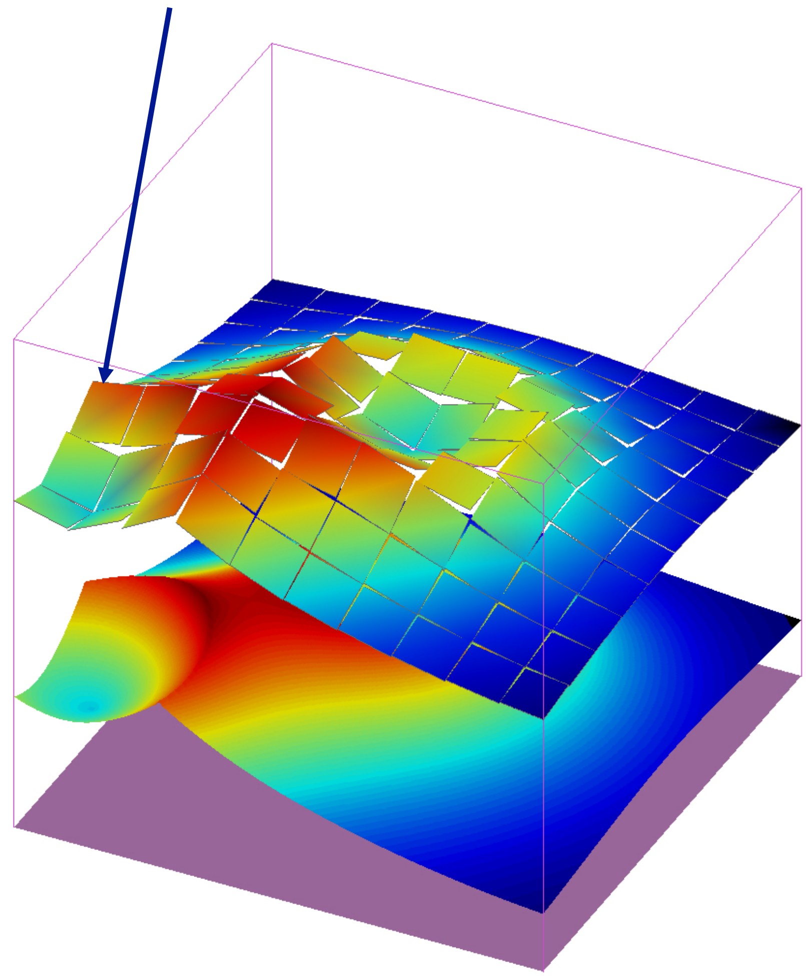
Piecewise Linear Functions in IvP

Piecewise linear (IvP) functions:

- Each point in the decision space belongs to exactly one piece.
- Each pieces has an interval boundary and a linear interior function.

Advantages:

- Any underlying function can be represented.
- Pieces need not be uniformly distributed.
- Extends to *n* dimensions.
- Syntax can be exploited by the solution algorithms.



The IvPBuild Toolbox

Q: How are IvP Functions built?

A: The IvPBuild Toolbox

The IvPBuild Toolbox is a

- C++ Library,
- Distributed with the MOOS-IvP tree.
- A set of tools for building IvP functions from a user's underlying objective function.
- Meant to be invoked from within a behavior implementation – from within the onRunState() function.

The IvPBuild Toolbox contains two basic tools:

- The ZAIC tool – for building 1D objective functions.
- The Reflector tool – for building IvP Functions in N dimensions.

Outline

- The IvP Behavior Interface
- Writing Your First Behavior and Augmenting the Helm
- Overview of IvP Functions
- The Reflector Tool
- The ZAIC Tool
- Rendering IvP Functions

The Reflector Tool
(Pure Uniform)

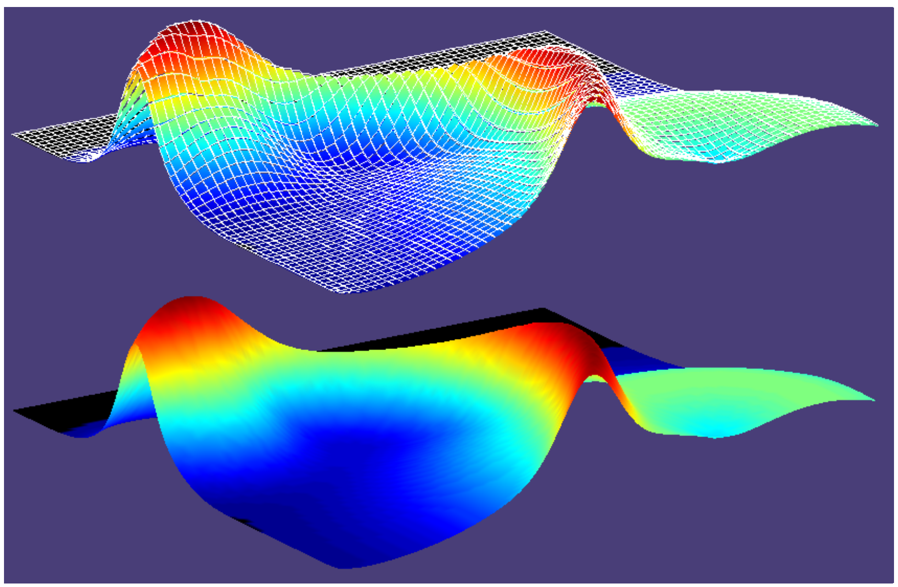
The Reflector Tool builds an IvP function from a given underlying function by sampling the underlying function.

Three variations discussed:

→ (1) Pure Uniform
(2) Uniform with Prioritized Augmentation
(3) Uniform with Focused Augmentation

Algorithm Design Criteria
(1) Minimize error.
(2) Minimize pieces generated.
(3) Minimize generation time.
★ (4) Minimize complexity of use for the user.
★ (5) Minimize restrictiveness of the tool.

Method 1 - *Pure Uniform*



Basic idea:
- Function is composed of uniform piecewise linearly defined pieces.

Pros:
- Simple to use.
- Requires no insight into underlying function.
- Can explore time, size, accuracy tradeoff space.

Cons:
- Treats all areas of the underlying function equally.
- Does not capitalize on insight into underlying function.

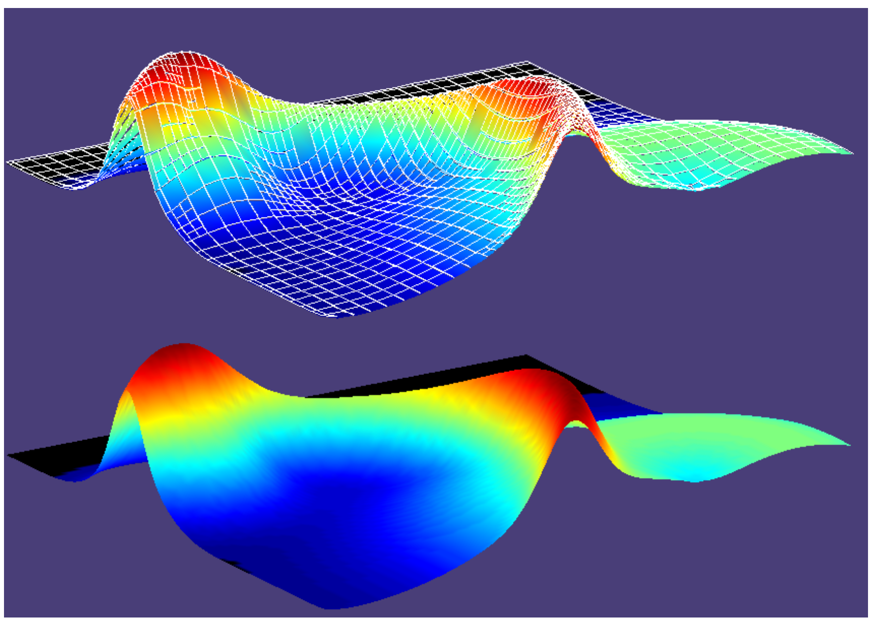
The Reflector Tool
(Uniform with Prioritized Augmentation)

The Reflector Tool builds an IvP function from a given underlying function by sampling the underlying function.

Three variations discussed:
- (1) Pure Uniform
- → (2) Uniform with Prioritized Augmentation
- (3) Uniform with Focused Augmentation

Algorithm Design Criteria
- ★ (1) Minimize error.
- ★ (2) Minimize pieces generated.
- (3) Minimize generation time.
- ★ (4) Minimize complexity of use for the user.
- (5) Minimize restrictiveness of the tool.

Method 2 - Uniform with Priority-Based Augmentation



Basic idea:

Start with a uniform function and further refine the pieces that have the worst error (prioritized during first linear regression phase).

Pros:
- Simple to use. No insight into underlying function required
- Can explore time, size, accuracy tradeoff space.

Cons:
- Does not always catch the pieces with worst error.
- Does not capitalize on insight into underlying function.

The Reflector Tool

(Uniform with Prioritized Augmentation)

The Reflector Tool builds an IvP function from a given underlying function by sampling the underlying function.

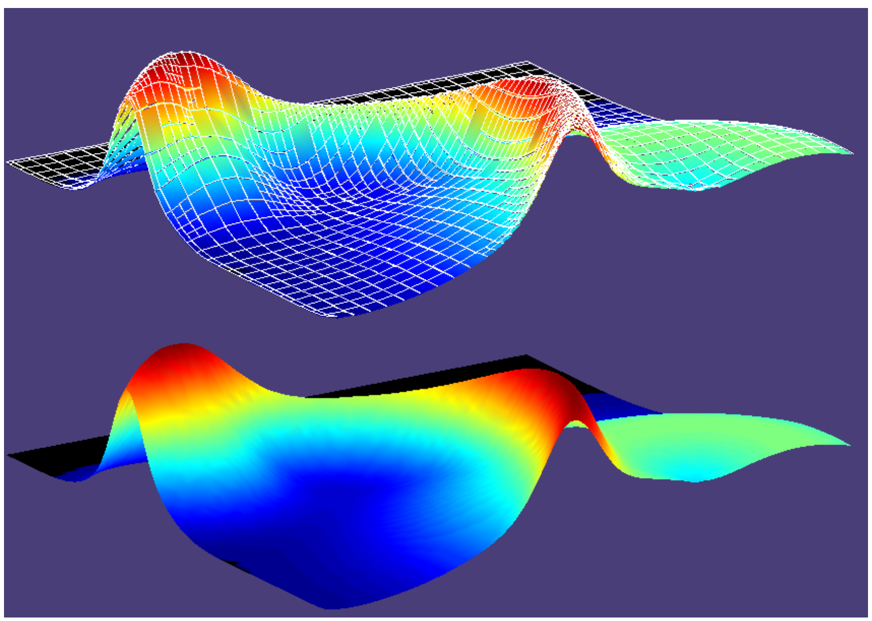
Three variations discussed:

(1) Pure Uniform
→ (2) Uniform with Prioritized Augmentation
(3) Uniform with Focused Augmentation

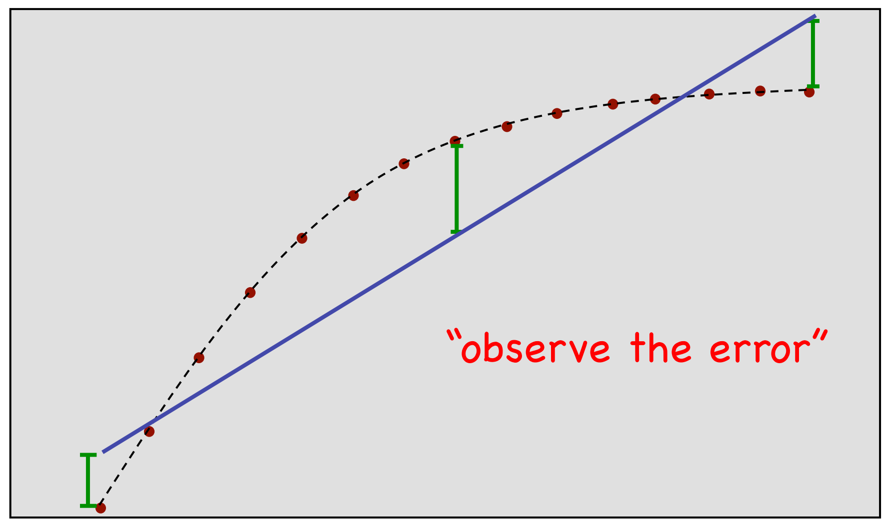
Algorithm Design Criteria

- ★ (1) Minimize error.
- ★ (2) Minimize pieces generated.
- (3) Minimize generation time.
- ★ (4) Minimize complexity of use for the user.
- (5) Minimize restrictiveness of the tool.

Method 2 - Uniform with Priority-Based Augmentation



As a linear approximation is calculated, keep track of the observed error.
Store pieces in a fixed-length priority queue for later revisit.



* function ranges from 0-100

The Reflector Tool

(Uniform with Prioritized Augmentation)

The Reflector Tool builds an IvP function from a given underlying function by sampling the underlying function.

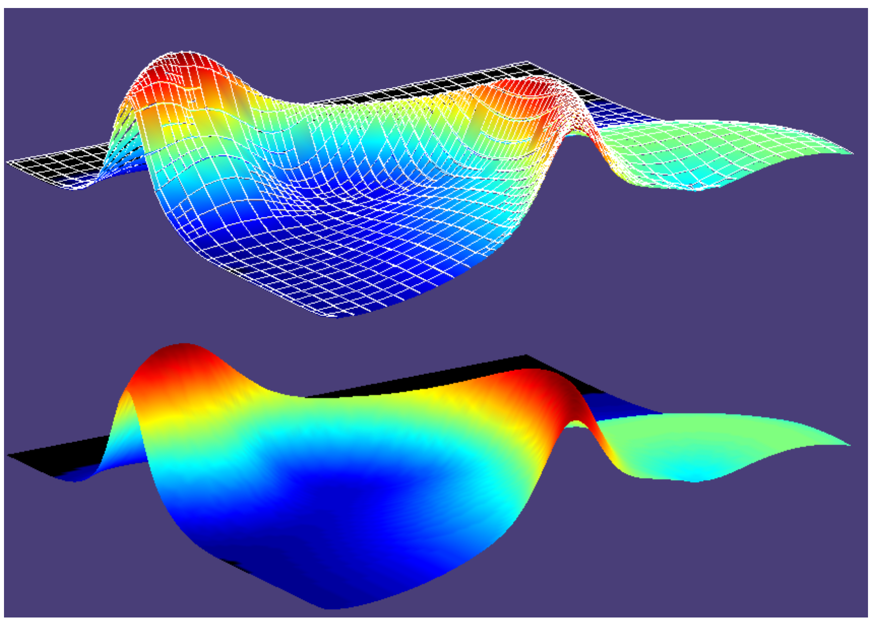
Three variations discussed:

(1) Pure Uniform
→ (2) Uniform with Prioritized Augmentation
(3) Uniform with Focused Augmentation

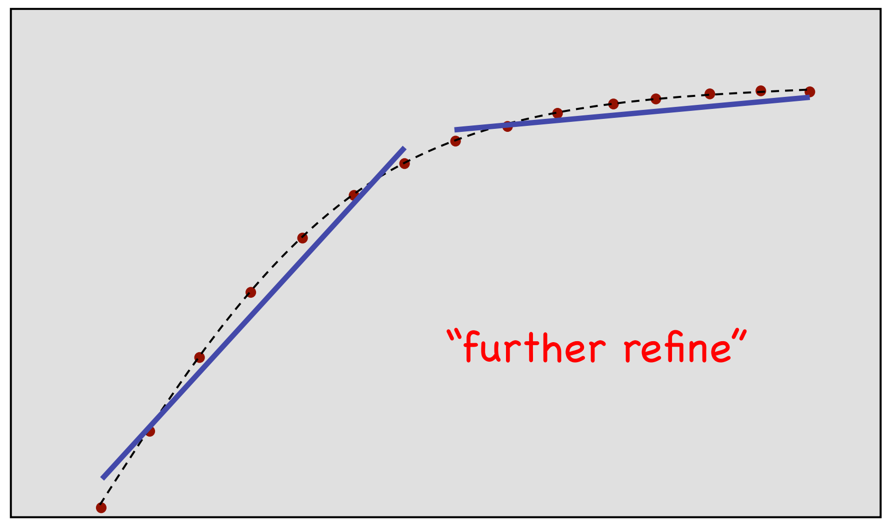
Algorithm Design Criteria

★ (1) Minimize error.
★ (2) Minimize pieces generated.
(3) Minimize generation time.
★ (4) Minimize complexity of use for the user.
(5) Minimize restrictiveness of the tool.

Method 2 - Uniform with Priority-Based Augmentation



As a linear approximation is calculated, keep track of the observed error.
Store pieces in a fixed-length priority queue for later revisit.



* function ranges from 0-100

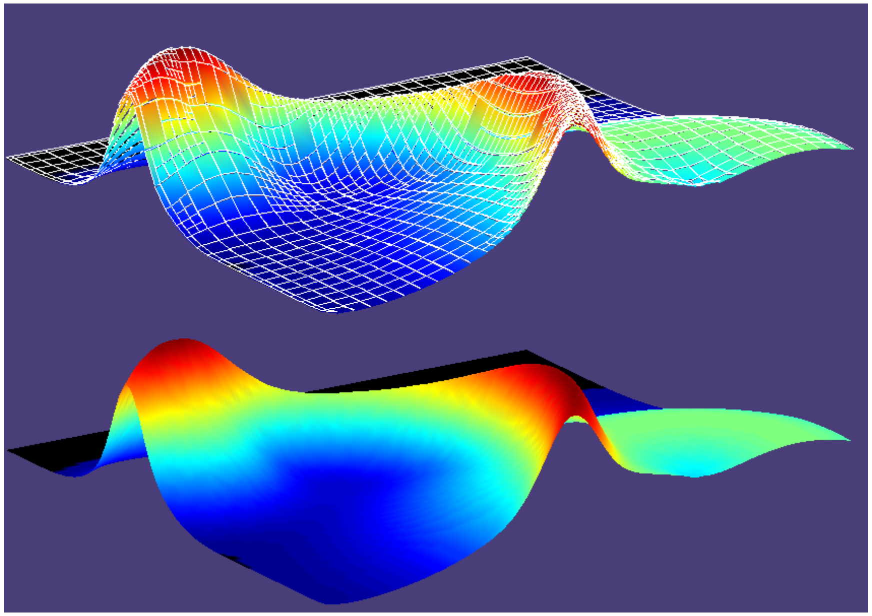
The Reflector Tool
(Uniform with Prioritized Augmentation)

The Reflector Tool builds an IvP function from a given underlying function by sampling the underlying function.

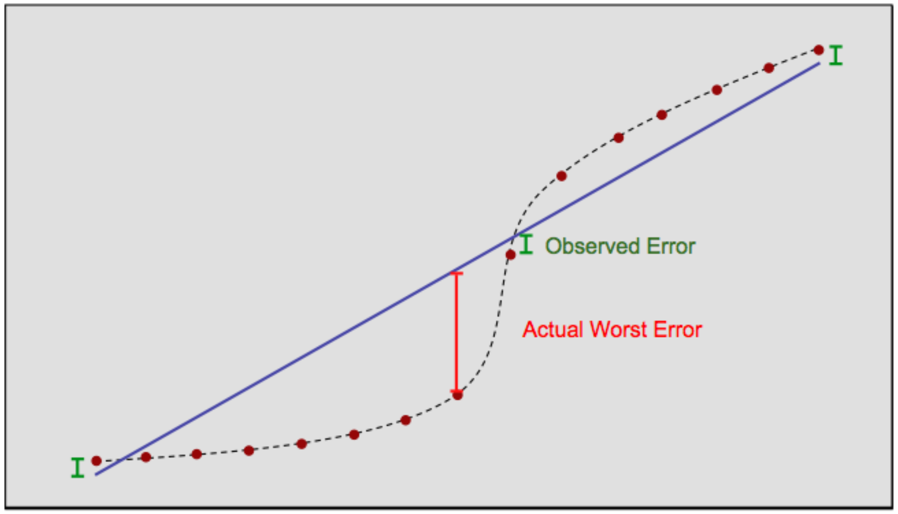
Three variations discussed:
- (1) Pure Uniform
- → (2) Uniform with Prioritized Augmentation
- (3) Uniform with Focused Augmentation

Algorithm Design Criteria
- ★ (1) Minimize error.
- ★ (2) Minimize pieces generated.
- (3) Minimize generation time.
- ★ (4) Minimize complexity of use for the user.
- (5) Minimize restrictiveness of the tool.

Method 2 - Uniform with Priority-Based Augmentation



As a linear approximation is calculated, keep track of the observed error.
Store pieces in a fixed-length priority queue for later revisit.



The Reflector Tool

(Uniform with Prioritized Augmentation)

The Reflector Tool builds an IvP function from a given underlying function by sampling the underlying function.

Three variations discussed:

(1) Pure Uniform

→ (2) Uniform with Prioritized Augmentation

(3) Uniform with Focused Augmentation

Algorithm Design Criteria

★ (1) Minimize error.

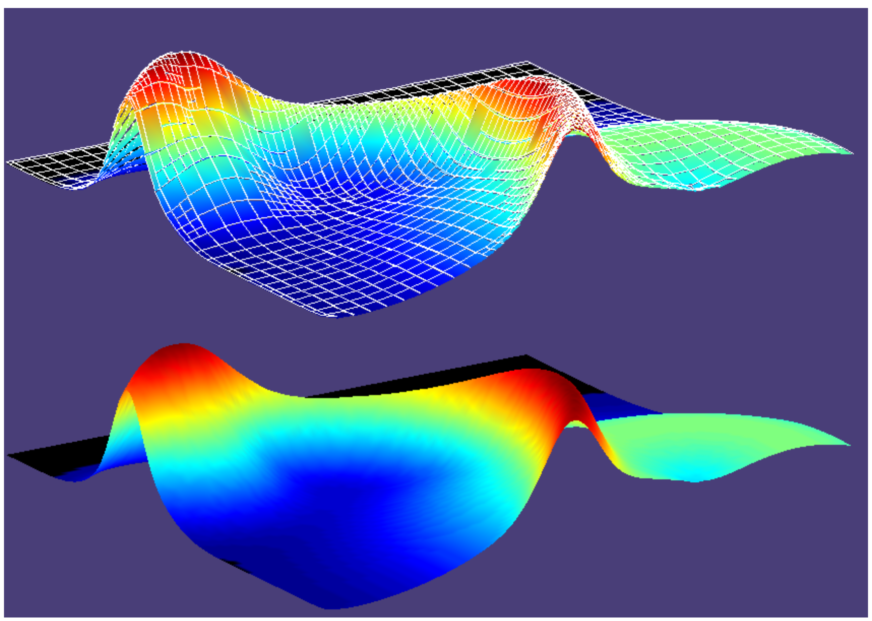
★ (2) Minimize pieces generated.

(3) Minimize generation time.

★ (4) Minimize complexity of use for the user.

(5) Minimize restrictiveness of the tool.

Method 2 - Uniform with Priority-Based Augmentation



Basic idea:

Start with a uniform function and further refine the pieces that have the worst error (prioritized during first linear regression phase).

Pros:

- Simple to use. No insight into underlying function required
- Can explore time, size, accuracy tradeoff space.

Cons:

- Does not always catch the pieces with worst error.
- Does not capitalize on insight into underlying function.

The Reflector Tool

(Uniform with Focused Augmentation)

The Reflector Tool builds an IvP function from a given underlying function by sampling the underlying function.

Three variations discussed:

(1) Pure Uniform

(2) Uniform with Prioritized Augmentation

→ (3) Uniform with Focused Augmentation

Algorithm Design Criteria

★ (1) Minimize error.

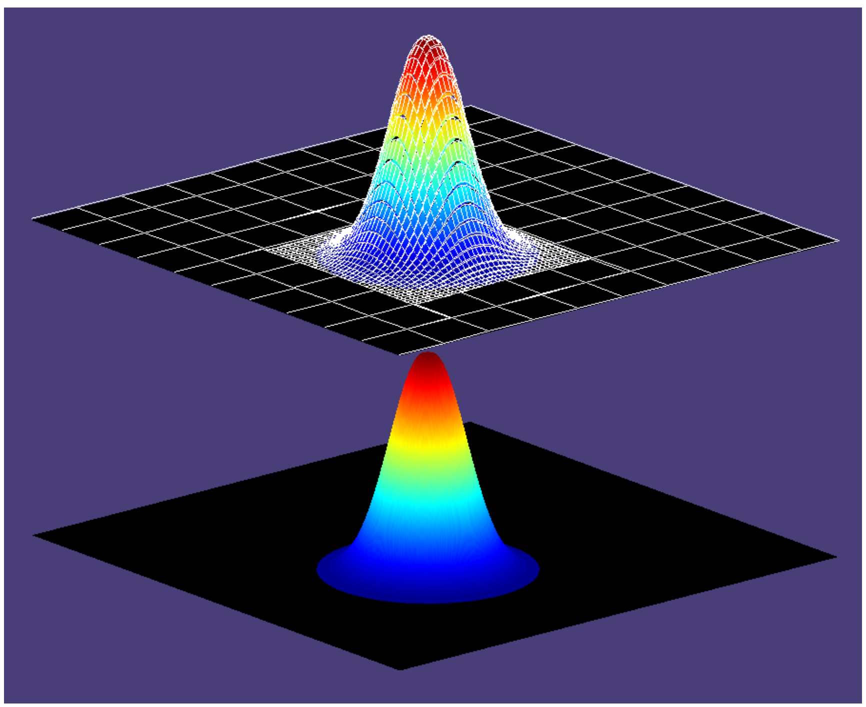
★ (2) Minimize pieces generated.

★ (3) Minimize generation time.

(4) Minimize complexity of use for the user.

(5) Minimize restrictiveness of the tool.

Method 3 - Uniform with Focused Augmentation



Basic idea:

Start with a uniform function and further refine the pieces in areas thought to need more pieces for error reduction

Pros:

- Simple to use. Capitalizes on insight of underlying function.
- Can explore time, size, accuracy tradeoff space.

Cons:

- Not all functions have area suitable for focused refinement.
- Requires insight into underlying function.

The ZAIC Tool

The ZAIC Tool builds an IvP function from a given underlying function by sampling the underlying function.

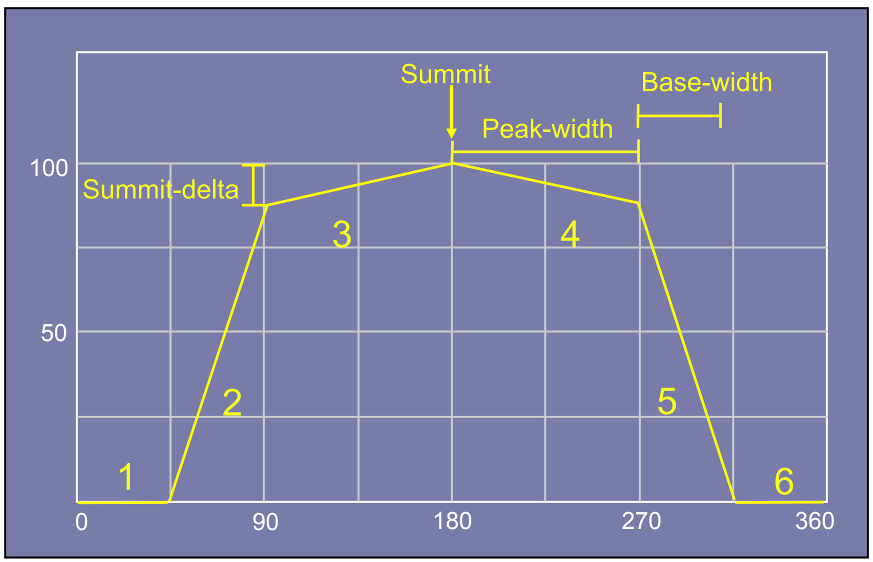
Three variations:

(1) ZAIC_PEAKS
(2) ZAIC_LEQ
→ (3) ZAIC_HEQ

Algorithm Design Criteria

- ★ (1) Minimize error.
- ★ (2) Minimize pieces generated.
- ★ (3) Minimize generation time.
- ★ (4) Minimize complexity of use for the user.
- ★ (5) Minimize restrictiveness of the tool.

Method 4 - ZAIC Peaks



Basic idea:
1D Functions with one or more peaks. Identify the peak properties and the IvP function is generated.

Pros:
Simple to use. Very few pieces.
As many peaks as desired.

Cons:
Only suitable for 1D objective functions.

IvP Function:

(1)	$0 \le x \le 45$	$y = 0$
(2)	$46 \le x \le 90$	$y = 1.89x - 85$
(3)	$91 \le x \le 180$	$y = 0.17x + 70$
(4)	$181 \le x \le 270$	$y = -0.17x + 130$
(5)	$271 \le x \le 315$	$y = -1.89x + 595$
(6)	$316 \le x \le 359$	$y = 0$

Outline

- The IvP Behavior Interface
- Writing Your First Behavior and Augmenting the Helm
- Overview of IvP Functions
- The Reflector Tool
- The ZAIC Tool
- Rendering IvP Functions

Rendering IvP Functions

Two tools for rendering IvP Functions

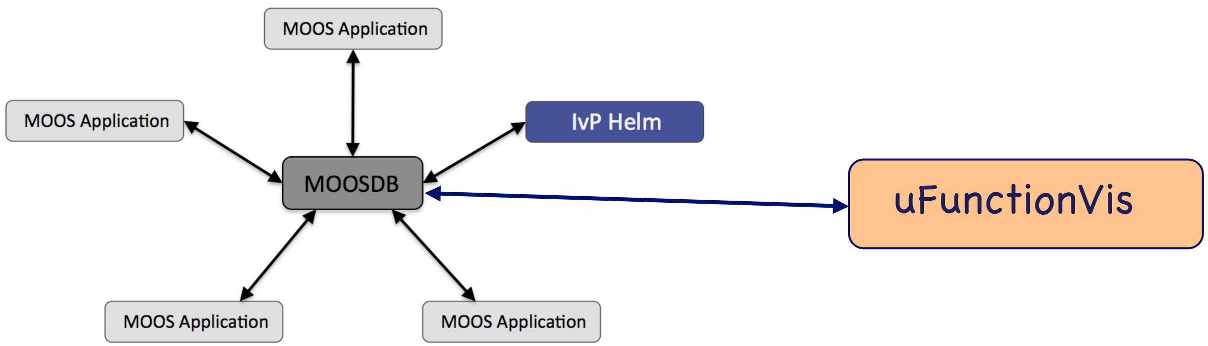
(1) uFunctionVis: For rending IvP Functions during missions (typically sim)

(2) alogview: For rendering IvP Functions post-mission from log files.

The uFunctionVis Tool

The uFunctionVis Tool:

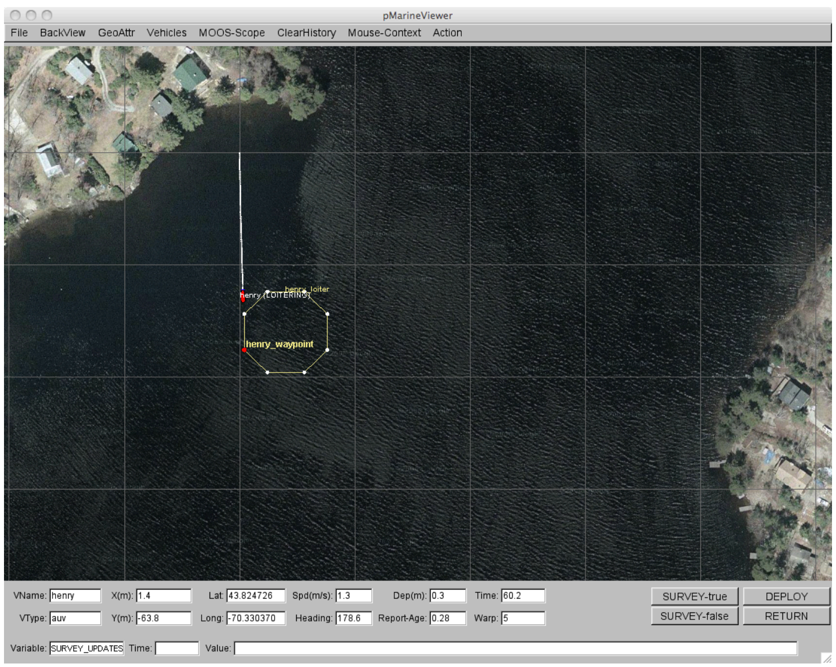
- A separate MOOSApp that subscribes for IvP Functions posted by the helm
- The IvPHelm posts all objectives functions to the MOOSDB for rendering and debugging purposes. All posted to the variable BHV_IPF.

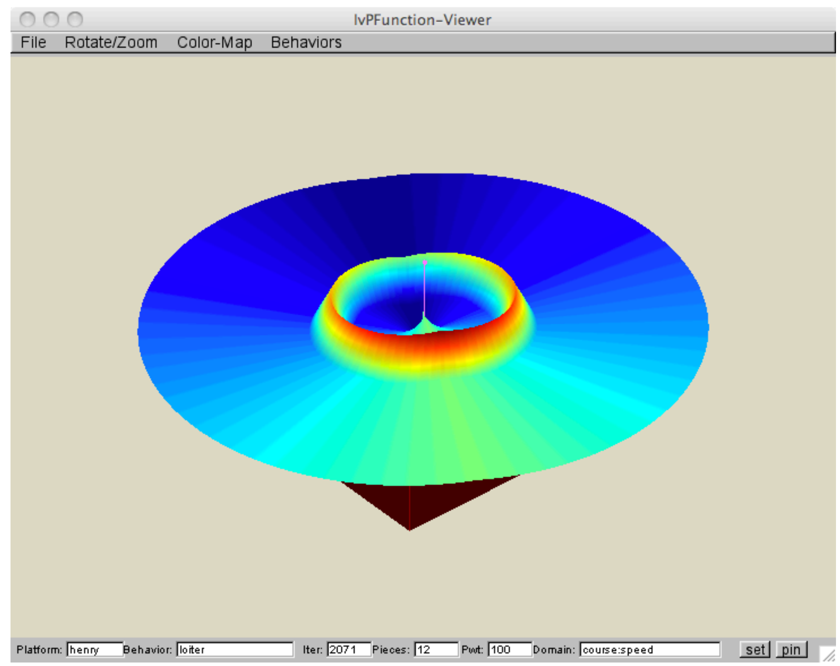


The uFunctionVis Tool

The uFunctionVis Tool:

- A separate MOOSApp that subscribes for IvP Functions posted by the helm
- The IvPHelm posts all objectives functions to the MOOSDB for rendering and debugging purposes. All posted to the variable BHV_IPF.

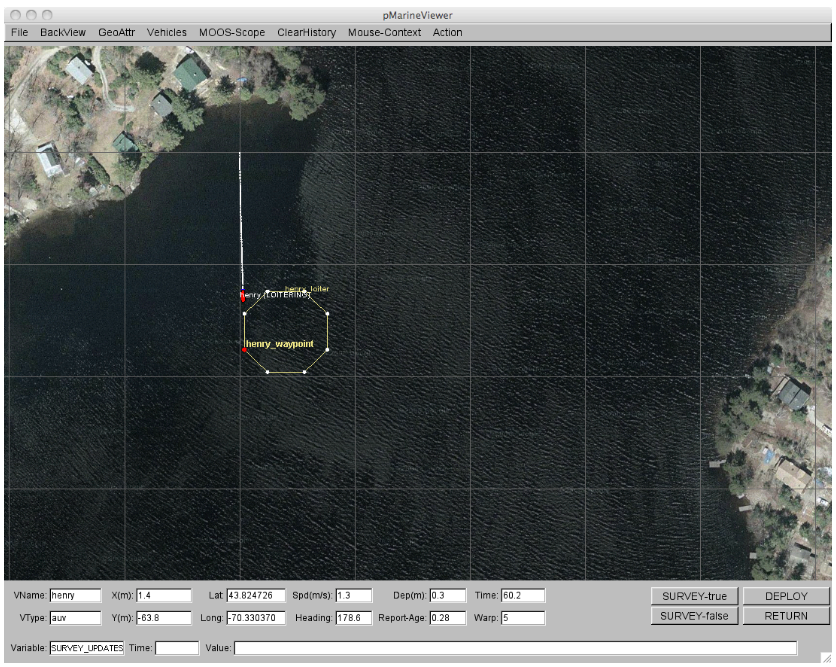


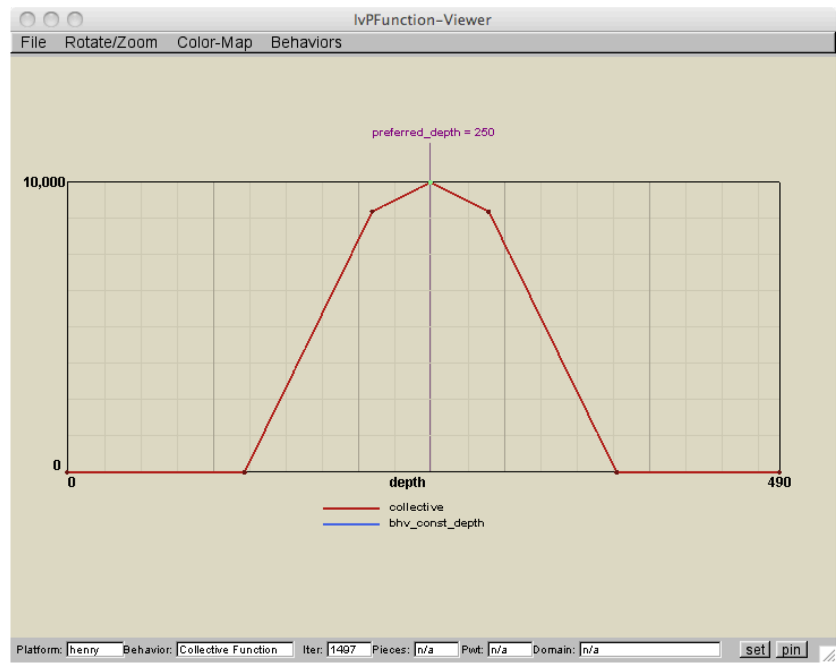


The uFunctionVis Tool

The uFunctionVis Tool:

- A separate MOOSApp that subscribes for IvP Functions posted by the helm
- The IvPHelm posts all objectives functions to the MOOSDB for rendering and debugging purposes. All posted to the variable BHV_IPF.





The alogview Tool
(Part of the AlogToolbox)

The alogview Tool:

- An off-line (non-MOOS) tool for rendering alog files.
- Contains native capability for rendering IvP functions from the helm.