Adaptive Sampling With Trigger Conditions

5/22/14 Brent Roman   brent@mbari.org

Traditional ESP Missions

- A sequence of “phases”, each with a prescribed start time
  - Actions predetermined by puck load
- ESP sleeps between phases. While “asleep”:
  - Still monitors contextual sensors
  - Still maintains radio context with shore
- All phases began at times prescribed in the mission script
  - Start times specified may be absolute or relative
    - Relative times specify the “sleep time” between phases
- No adaptive sampling was possible without hand coding it

Trigger Condition Overview

- Each start time is augmented by a list of trigger conditions
  - A phase starts when any of its trigger conditions is true
  - The start time can be thought of as the one required trigger condition
    - It determines the latest possible starting time for the phase
    - Triggers start phases before their scheduled times
      - Triggers cannot delay phases beyond their “start times”
      - Triggers ***cannot*** change the sequence of actions performed
        - » *Processing sequence is determined by puck load.*
- Each trigger condition is reevaluated whenever contextual sensors read
  - Sensible, as trigger conditions almost always evaluate sensor data
  - This is a convention
    (but, not difficult to circumvent if necessary)
- Each trigger condition runs in its own Ruby thread
  - Failure (e.g. exceptions raised) in any trigger will not affect the others
    - You can even patch the code and restart failed trigger conditions
    - Or, kill the trigger thread to ensure it does not trigger the phase

Basic Trigger Conditions

- Basic Trigger Conditions contain arbitrary true/false expressions
  - A threshold value is associated with each
    - CTD.temp < threshold
    - ISUS.no3 > threshold
    - CTD.depth > threshold[0] and CTD.fluor > threshold[1]
  - Thresholds need not be scalar values
  - Trigger expressions are reevaluated just after each time contextual sensors are read while the mission is awaiting conditions
- May be assigned names like Cold, Hot, Fresh, Salty
- Threshold values can be modified at any time
  - Via the script itself or the interactively via espclient
  - All modifications to thresholds are logged
- Very flexible, but also painfully verbose for complex triggers

Composite Trigger Conditions

- Two types
  - Trigger “all” means when all subordinate conditions are true
    - Trigger all: [Cold, DCM, HighNitrate]
    - Equivalent to: Cold[] and DCM[] and HighNitrate[]
    - Trigger all: []
      - is always true
  - Trigger “any” means when any subordinate condition is true
    - Trigger any: [Cold, DCM, HighNitrate]
    - Equivalent to: Cold[] or DCM[] or HighNitrate[]
    - Trigger any: []
      - is always false
- All subordinate conditions run in the same thread as the parent

Trigger.now!

- Not really a trigger condition, rather an action!
  - Causes the current mission phase to start immediately
  - Raises an exception if mission is not waiting
    - Exception is raised in caller's thread
    - The mission's processing is unaffected

- There need not be any trigger conditions associated with the waiting phase for Trigger.now! to work.
  - The phase may be just awaiting its start time

Trigger.replace or Trigger.restart

- Replace current phase's start time and/or trigger conditions
  - Affects only for the phase currently waiting to start
  - Raises an exception if mission is not waiting
- All arguments are optional
- First argument is the replacement phase start time
  - Specify nil to leave start time unchanged
- Other arguments are replacement trigger conditions
  - Omit other args to leave existing triggers in place
- Trigger.replace “+1.5 days”, Cold, Deep
  - Mission will continue waiting up to 36 more hours for the Cold ***or*** Deep condition to be satisfied

Trigger Range Conditions

- True if each listed measurement is within one of the associated ranges of interest
  - Represented as a Ruby hash mapping keys to values. In this case:
    - Keys are measurements, like:
      - CTD%:temperature
      - ISUS%:no3
      - Not *CTD.temperature* #NO!!!
        because that would check temperature once, when the condition was defined.
    - Values are ranges, like:
      - -3.3 .. 2.1
  - Trigger range:
    ```
    {CTD%:temperature => [-3.3..2.1, 5..7.21],
     CTD%:salinity => [ 33..33.4, 23..28.3, 35..35.5],
     ISUS%:no3 => 32.03..12.3}  #may omit [] for an array of one element
    ```
  - If first > last, as in 32.03..12.3 above, range check is logically negated
    Equivalent to: (no3 > 32.03 or no3 < 12.3)

Trigger Box Conditions

- True if each listed measurement is within *the same* associated box of interest
  - Represented as the same Ruby hash mapping used for Trigger Ranges
  - Trigger box:
    ```
    {CTD%:temperature     => [-3.3..2.1, 5..7.21],
     CTD%:salinity        => [ 33..33.4, 35..35.5]}
    ```
  - Read the boxes off the columns of the resulting matrix.
  - If temperature is in one column and salinity is in the other, the trigger condition is *false*
- Columns geometrically define a set of boxes in the space of sensor measurements

Trigger Box Corner Cases

- If measurements do not specify the same number of ranges:
  - Those that are missing ranges will be ignored

```
Trigger  box:
{CTD%:temperature => [-3.3..2.1, 5..7.21],
 CTD%:salinity      => [ 33..33.4 ]}
```

  - If the temperature is between 5..7.21, the trigger condition is true, regardless of salinity
- If a measurement specifies a single range (not an Array)
  - That range will be applied to all others
  - As though it had been repeated in an Array

```
Trigger  box:
{CTD%:temperature => [-3.3..2.1, 5..7.21],
 CTD%:salinity      => 33..33.4}
```

  - The salinity must always be in 33..33.4, regardless of temperature

Trigger Holdoffs

- Trigger holdoffs are a simple way to avoid false triggers
  - A form of glitch filtering
  - ESP logs show countdown when awaiting holdoffs
- All triggers have an associated holdoff in samples
  - condition must be true for at least holdoff+1 samples
  - nil is the default holdoff value
    - holdoff=nil, equivalent of holdoff=0
      - But holdoff nil is not displayed, whereas 0 is
  - holdoff of false disables that particular trigger condition
  - holdoff of true forces trigger on its next evaluation

Trigger Thresholds

- Each trigger optionally has an associated threshold value
  - Usually used to parametrize conditional expressions
    - But you may choose to compare to constants instead
  - Need not be scalar, only the expression interprets it
  - Not usually applicable to box or range conditions
    - Such thresholds would be vectors of ranges if used
- If your conditional expressions reference a threshold:
  - You must set it before the trigger is used
    - Cold.threshold = 4.3   #it's that easy!
  - The default threshold value is nil
    - CTD.fluor > nil #will generate an exception!

Trigger enable and disable

- Enable trigger monitoring with:
  - Trigger.enable
- Disable trigger monitoring with:
  - Trigger.disable
- Trigger monitoring is initially disabled
  - Use Trigger.enable as soon as contextual data starts making sense and all relevant thresholds are defined
- Triggers may be enabled/disabled at any time
  - Even while awaiting them
- Triggers are initially enabled during simulation!

Automatic Trigger Rearm

- Trigger monitoring may be disabled whenever a trigger condition causes a phase to start
  - If triggers remain enabled, rearm is said to be true
  - If triggers disable once one has fired, rearm is said to be false
- Set the rearm flag with:
  - Trigger.rearm = true
- Clear the rearm flag with:
  - Trigger.rearm = false
- Real missions start with rearm=false
  - You may change the Trigger.rearm flag at any time
  - You may want to combine it with Trigger.enable or Trigger.disable
- Simulation missions start with Trigger.rearm=true