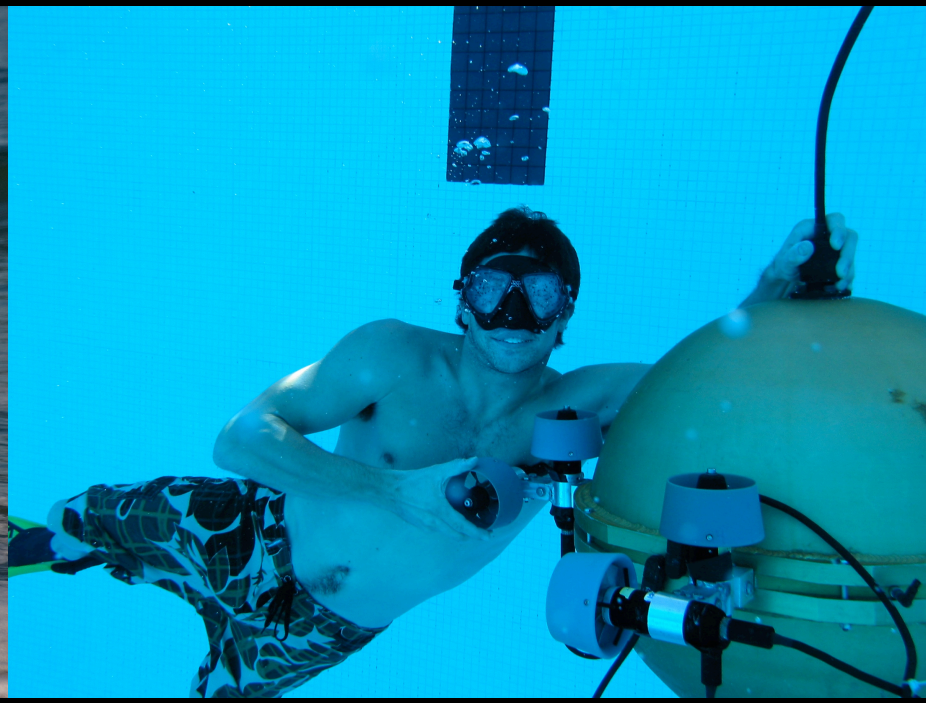
MRRSS
Marine Robotics Research Summer School

Autonomous Underwater Software Development



MRRSS – 2016

Ryan N. Smith

Topics

- Front End/Back End Software
  - MOOS
  - LCM
  - ROS
  - T-REX
- Issues in Development

MOOS

- Applications to control NMEA GPS, orientation and depth Sensors
- Allows state-based control of vehicles (pHelm)
- Sophisticated pose estimation framework for underwater and surface vehicles (pNav)

MOOS

"MOOS is a C++ cross platform middle ware for robotics research."

- CoreMOOS is a robust network based communications architecture
  - two libraries
  - lightweight communications hub called MOOSDB
- Essential MOOS is a layer of applications which use CoreMOOS. Common tasks include:
  - process control
  - logging.

MOOS

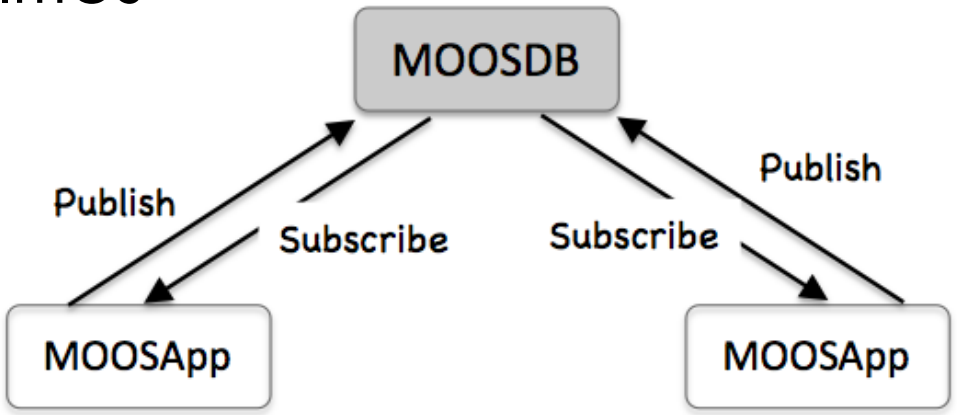
- Allows a live MATLAB session to join MOOS enabled processes
  - Allows visual debugging communication processes
  - Replay logged communications
- Applications to control NMEA GPS, orientation and depth Sensors
- Allows statebased control of vehicles (pHelm)
- A sophisticated pose estimation framework for underwater and surface vehicles (pNav)

MOOS-IvP

MOOS

- Interval Programming

- Platform Independence
- Module Independence
- Nested Capabilities



MOOS

Platform Independence:

The MOOS-IvP software typically runs on a dedicated computer

- "Backseat Driver" paradigm
- Two computers trade control

MOOS

Module Independence:

- MOOS is publish-and-subscribe middleware, and the IvP Helm is a behavior based architecture
- MOOS/IvP Helm provide architectures that enable autonomy/sensing system to be modular
- Benefits
  a. Modular autonomy/sensing system may be developed independently
  b. No one module is sacred—makes units replaceable

MOOS

Nested Capabilities:

- System architectures allow system to be extensible without core software modification
- Adaptable system

MOOS

Required to run:

- Windows, Mac, Linux, Ubuntu OS
- C++ language
- Terminal command line

LCM

- Provides a publish/subscribe message passing model
- Automatic marshaling/unmarshaling code generation with bindings
- Supported in a variety of programming languages

LCM

"LCM is a set of libraries and tools for message passing and data marshaling, targeted at real-time systems where high-bandwidth and low latency are critical."

- Provides a publish/subscribe message passing model
- Automatic marshaling/unmarshaling code generation with bindings
- Supported in a variety of programming languages

LCM

Supported Languages:

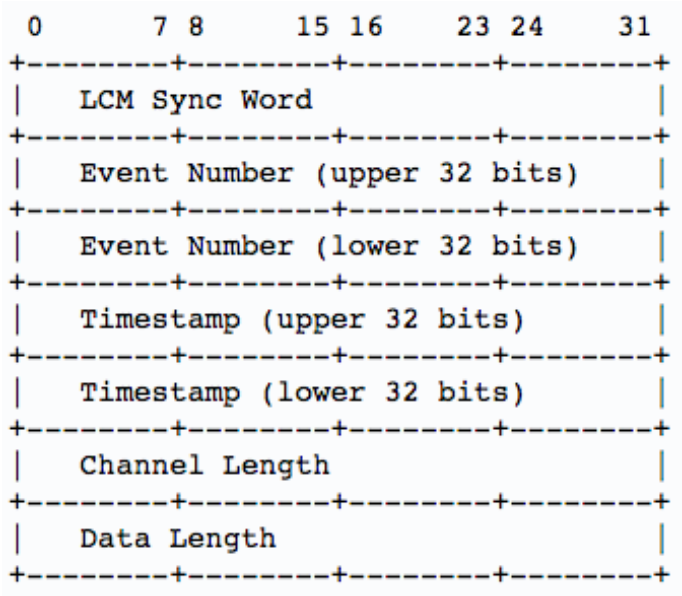
- C
- C++
- C#
- Java
- Lua
- MATLAB
- Python

LCM

Event Encoding Report:

Each event is encoded as a binary structure consisting of a header, followed by the channel and the data.

Example:



LCM

Required to run:

- Windows, Mac, Linux, Ubuntu OS
- Supported in many languages
- Terminal command line

ROS

- Publish/subscribe anonymous message passing
- Recording and playback of messages
- Request/response remote procedure calls
- Distributed parameter system

ROS

At the lowest level, ROS offers a message passing interface that provides inter-process communication, "middleware."

- publish/subscribe anonymous message passing
- recording and playback of messages
- request/response remote procedure calls
- distributed parameter system

ROS

Message Passing

- Anonymous publish/subscribe
- Clear interface between nodes (improves encapsulation and promotes code reuse)

ROS

Recording and Playback of Messages

- Data may be easily captured and replayed without changes to code
- Powerful design pattern that can significantly reduce development effort and promote flexibility and modularity

ROS

Remote Procedure Calls

- The ROS middleware provides synchronous request/response interactions between processes

ROS

Distributed Parameter System

- configure information through a global key-value store
  - Modify your task settings
  - Allows tasks to change the configuration of other tasks

ROS

Robot-specific capabilities that ROS provides:

- Standard Message Definitions for Robots
- Robot Geometry Library
- Robot Description Language
- Preemptable Remote Procedure Calls
- Diagnostics
- Pose Estimation
- Localization
- Mapping
- Navigation

ROS

Required to run:

- Windows, Mac, Linux, Ubuntu OS
- C++, Python, LISP languages
- Terminal command line

ROS Tutorials

- Go to: ros.org
- Click “Tutorials”
- Let’s “Get Started”

ROS Discussion and Example

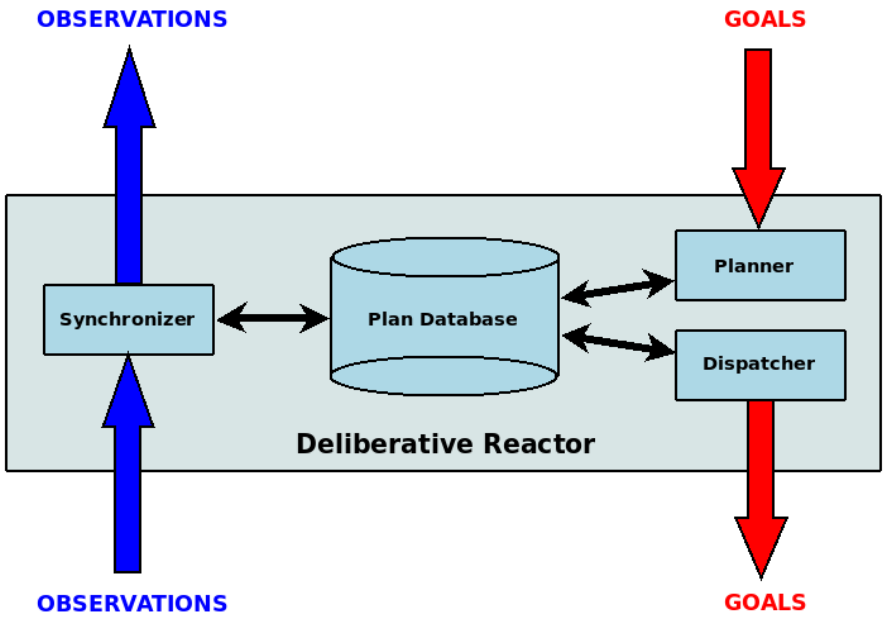
- Sandeep

Extending Autonomy

- At this point we can:
  - Plan
  - React
  - Re-plan
- What about Planning and Task Deliberation for multi-task missions?

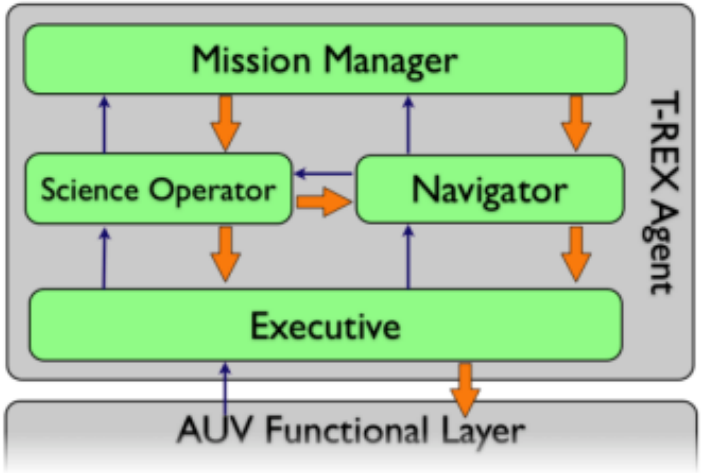
Teleo-Reactive Executive (T-REX)

- Sensing
- Planning
- Acting
  - "Deliberative Reactor"



T-REX

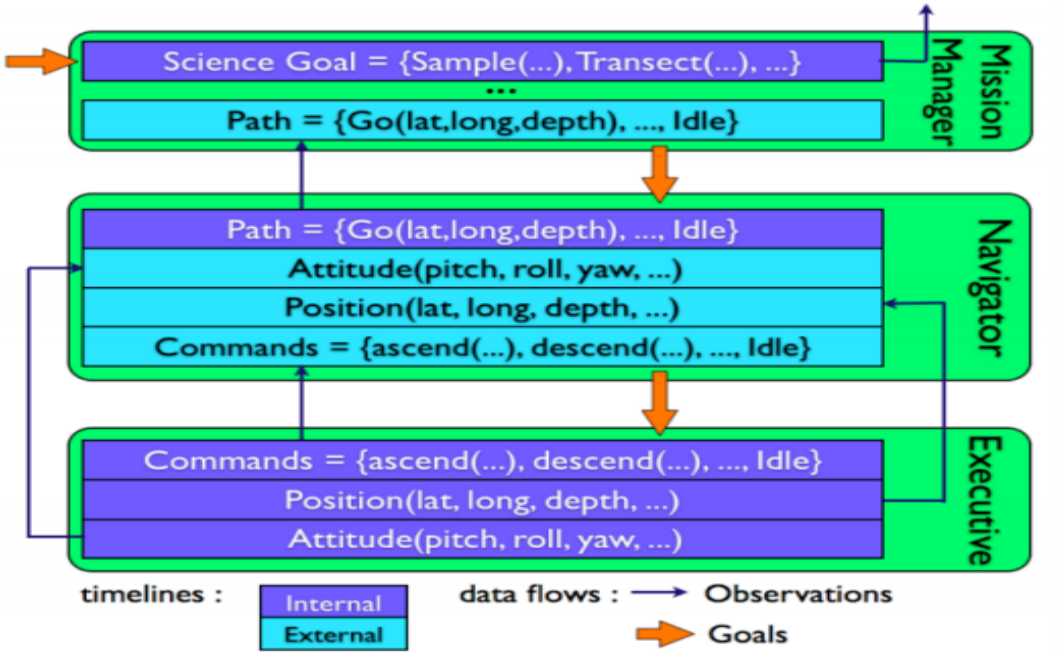
TREX is a hybrid executive that combines goal-driven and event-driven behavior in a unified framework based on temporal control and planning.



Goal-oriented feedback control system

- Green Boxes represent control transducers
- Orange arrows are goals
- Blue arrows are feedback

T-REX


Science Goal = {Sample(...), Transect(...), ...}
...
Path = {Go(lat,long,depth), ..., Idle}
Mission Manager
Path = {Go(lat,long,depth), ..., Idle}
Attitude(pitch, roll, yaw, ...)
Position(lat, long, depth, ...)
Commands = {ascend(...), descend(...), ..., Idle}
Navigator
Commands = {ascend(...), descend(...), ..., Idle}
Position(lat, long, depth, ...)
Attitude(pitch, roll, yaw, ...)
Executive
timelines :
Internal
External
data flows :
Observations
Goals

T-REX

Advantages of a Deliberation Reactor:

- Seamless integration of planning and execution
- Model-compliant execution
- High-level programming model
- Adjustable automated programming model
- High-impact on research advances

T-REX

Disadvantages:

- Steep Learning Curve for constraint-based temporal planning
- Problems can be hard to debug
- Heavyweight representation for some applications
- Too slow for some applications
- Automated planning often requires domain-dependent heuristics

Issues in Development

- Application Based
  - Survey
  - Monitor
  - Tracking

Examples:

- Mine Countermeasures (MCM)
- Anti-Submarine Warfare (ASW)

Issues in Development

- Uniqueness
  - Specific task in mind
  - High cost of development
  - Not very adaptable

Issues in Development

- Most software is beta
  - Expert user required
  - Command line
  - Kludgy software
  - Not modular

Example:

AUVs have the capability of accessing inhospitable and/or dangerous areas like beneath thick layers of ice or in marine minefields. In order to do so protocols must be designed which allow an AUV to navigate autonomously in an unknown environment and return at the end of deployment. Such systems are often unique to the application. Scenarios are so diverse that they are not adaptable. This is a serious disadvantage.

Underwater Operation

Below the surface AUV operation becomes more difficult

- Sophisticated algorithms required
- Multiple sensors
- Back-up/emergency protocol

Exercises

Exercise:

An AUV is needed to explore the Challenger Deep (10,900 meters).

1. Specify the type of vehicle.
2. What kind of program is needed?
3. What are some considerable challenges?

MRSS

Marine Robotics Summer School

A robot is needed to study Mayan ruins, 15 meters below Lake Atitlan, Guatemala.

1. Specify the type of device.
2. What kind of control structure should be used?
3. What are some considerable challenges?