

Experiences from Robotic Autonomy Software

From research to planetary flight robots

Dr. Issa A.D. Nenas

Supervisor, Robotic Mobility

Principal Technologist

Jet Propulsion Laboratory,

California Institute of Technology

Lecture 1

Lectures at **LASER 13th LASER Summer School on Software Engineering**

Software for Robotics

September 9–17, 2017

© 2017 California Institute of Technology. Government Sponsorship Acknowledged.

Clearance: CL#14-2459

Overview

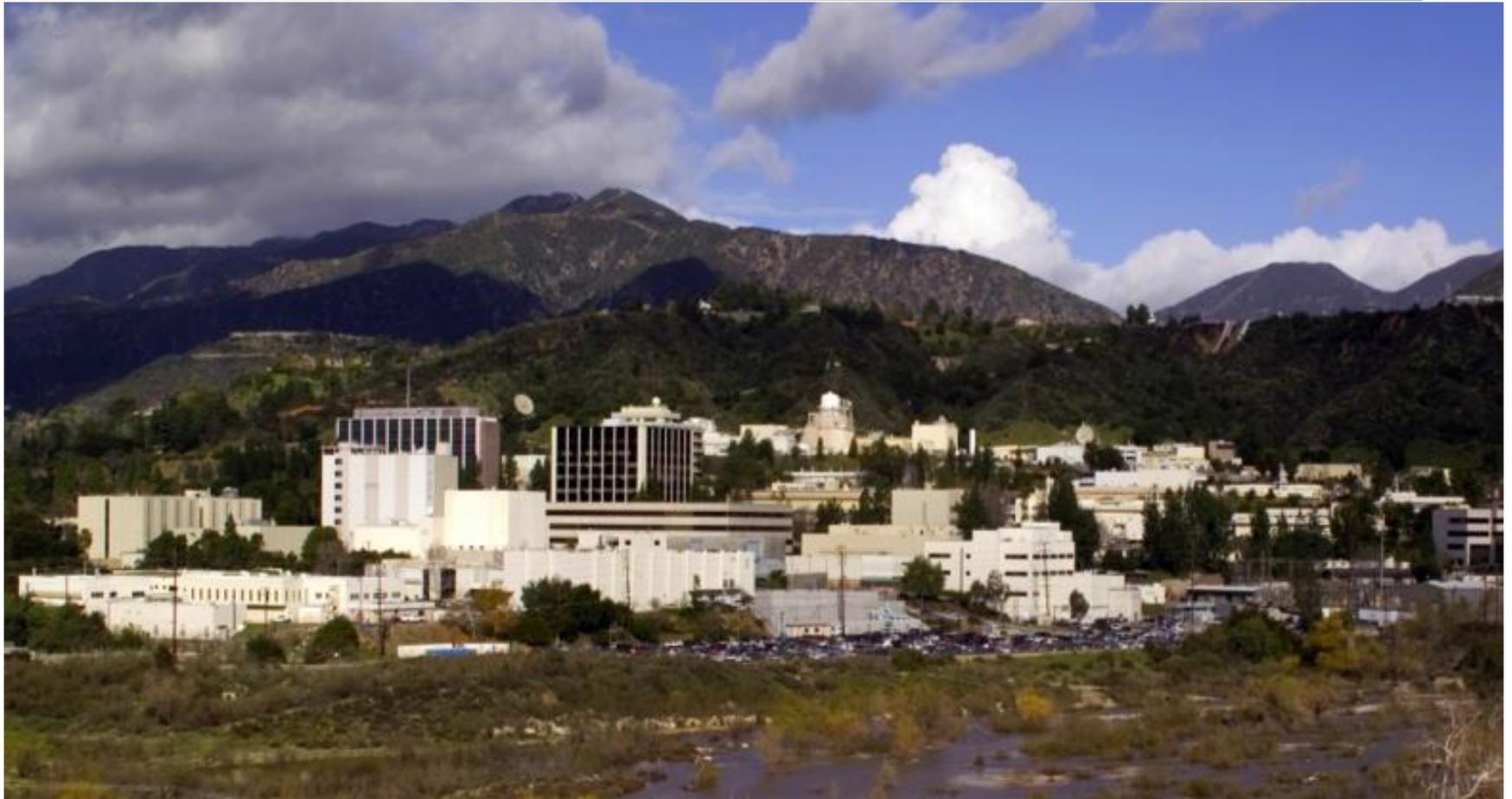


- NASA's Jet Propulsion Laboratory (JPL)
- My background
- Objective of the Lectures
- Lecture 1 – Overview
 - Lectures overview
 - Autonomy
 - Robot heterogeneity
 - Flight software environment
 - Robotic research software
 - Rules for safety critical software

INTRODUCTION TO NASA/JPL

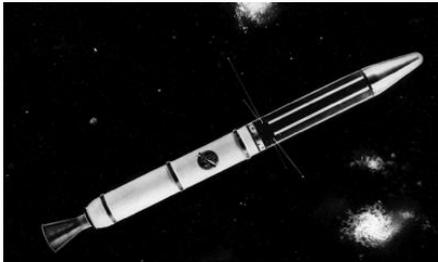


JPL Overview



- Pasadena, California
- One of 10 NASA centers
- Founded in the 1930s

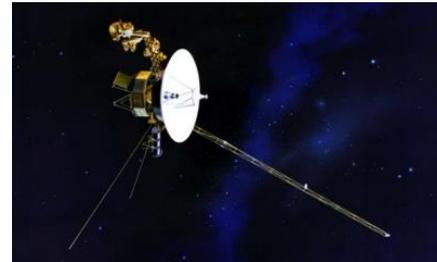
JPL Responsible for Many Space Firsts



1st U.S. satellite
1958 – Explorer 1



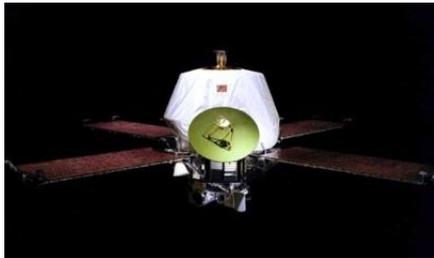
1st Close-up images of another planet
1964 – Mariner 4 / Mars



1st Fly-bys of Neptune and Uranus
1986, 1989 – Voyager 2



1st U.S. Spacecraft to the moon
1964 – Ranger 7



1st orbiter at another planet
1971 – Mariner 9 / Mars



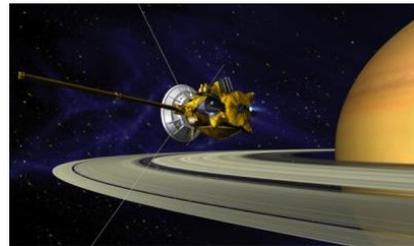
1st orbiter at Jupiter
1979 – Galileo



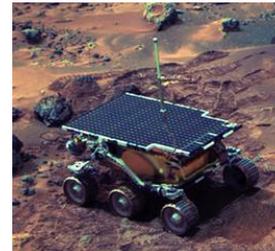
1st planetary mission
1962 - Mariner 2 /Venus



1st gravity assist mission
1974 – Mariner 10/ Venus



1st orbiter at Saturn
2004 – Cassini

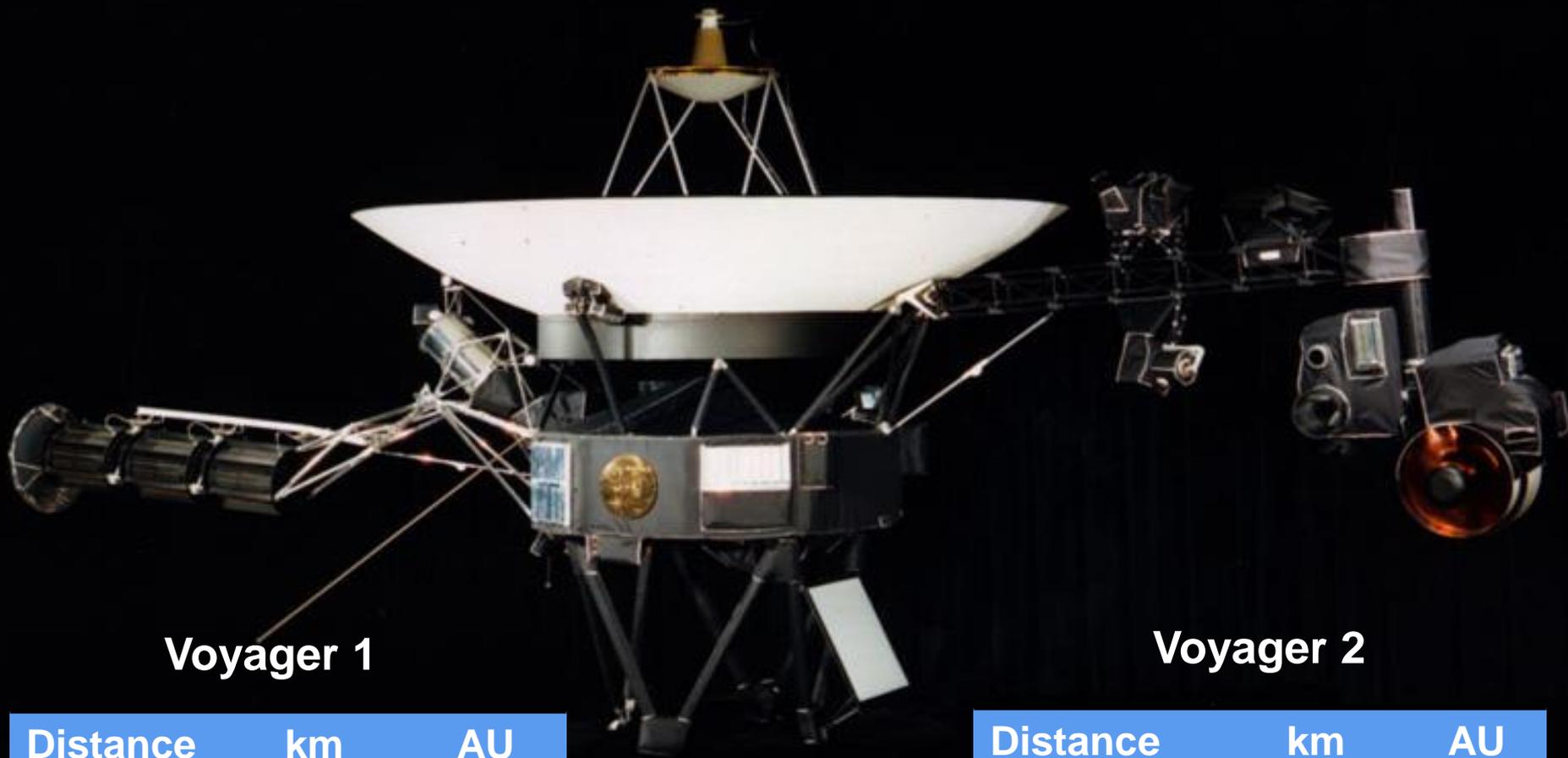


1st rover on Mars
1997 – Pathfinder

Currently operating 22 spacecraft, 2 rovers and 10 instruments in space

Voyager

(40 years and still going)



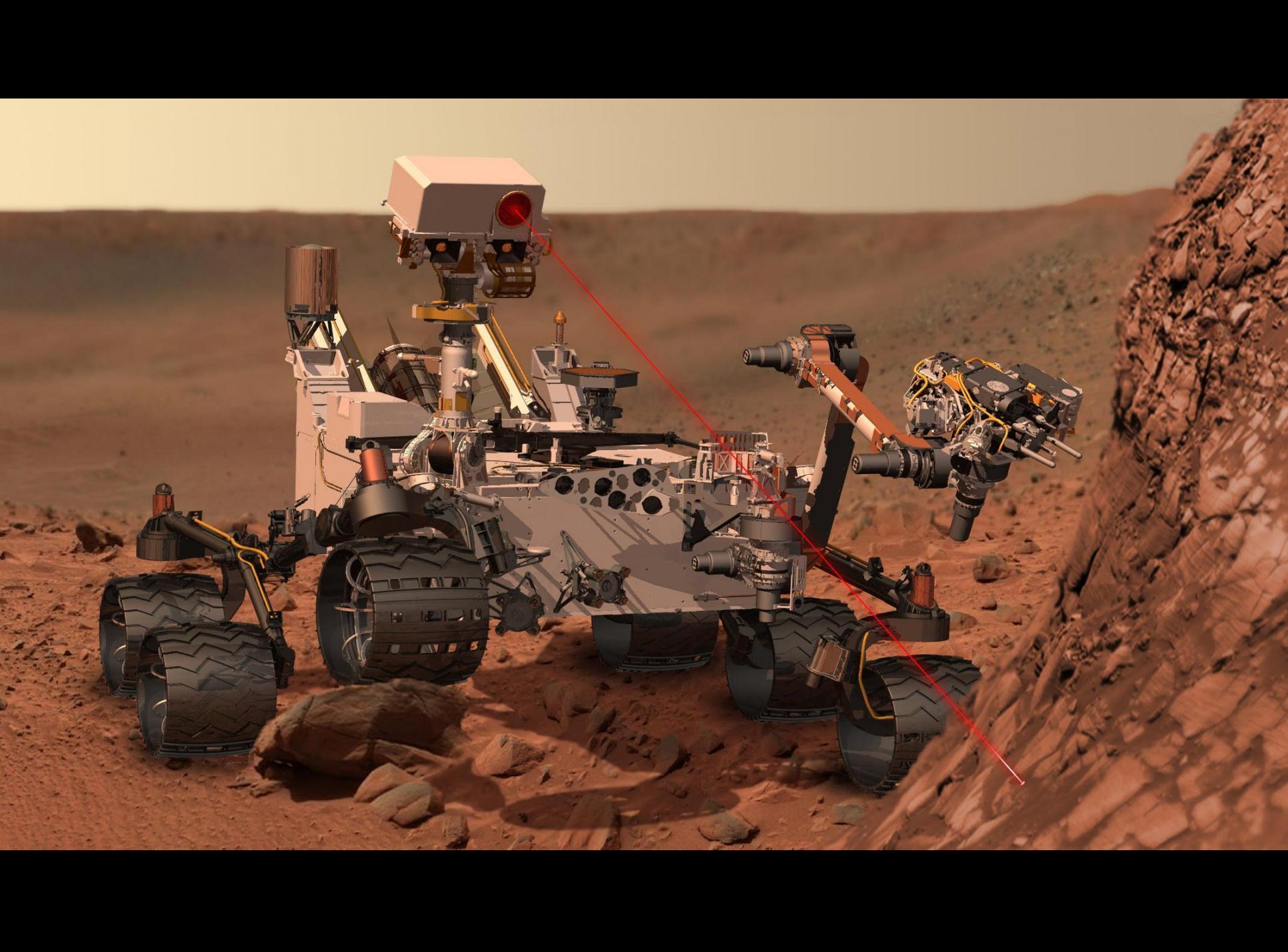
Voyager 1

Voyager 2

Distance from	km (billions)	AU
Earth	20.901	139.71
Sun	20.898	139.70

Distance from	km (billions)	AU
Earth	17.192	114.9
Sun	17.258	115.4

Cassini Grand Finale

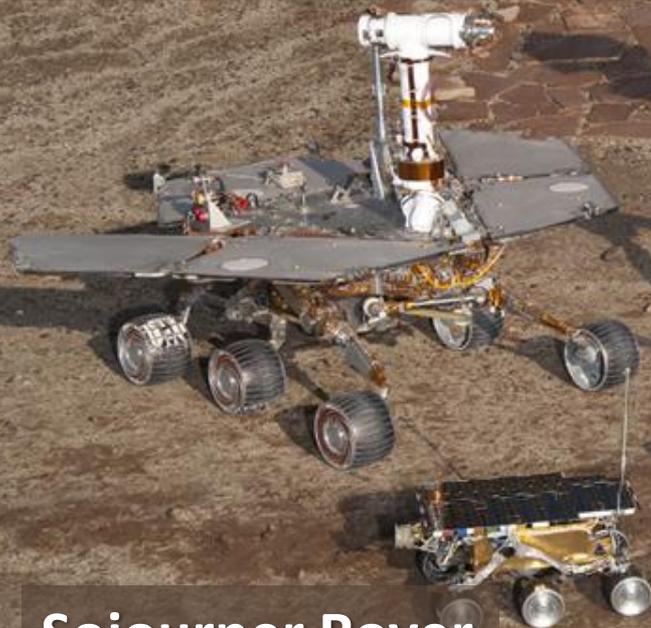


Mars Rovers



Mars Exploration Rover

1.6 meters 174 kg



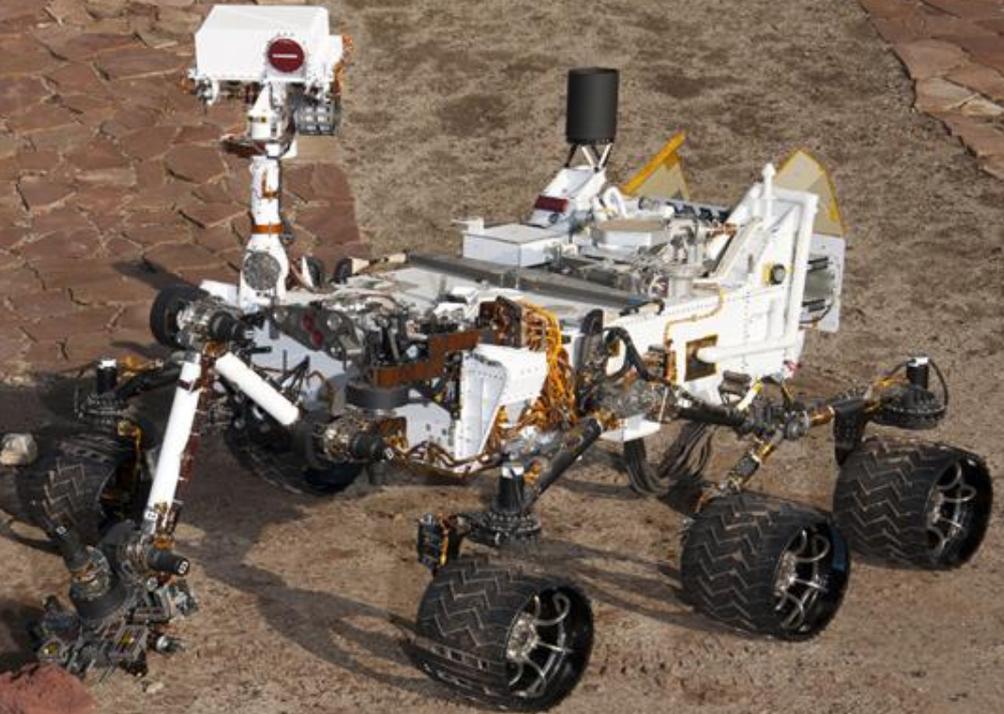
Sojourner Rover

65 cm 11.5 kg



Mars Science Laboratory

3.0 meters 900 kg



My Background and Experience



Experience

- **20 years in Space Robotics**
Research and flight – NASA/JPL
 - Leads robotic mobility (4 years)
 - Led robotics software (8 years)
- **2.5 years in Industrial Robotics**

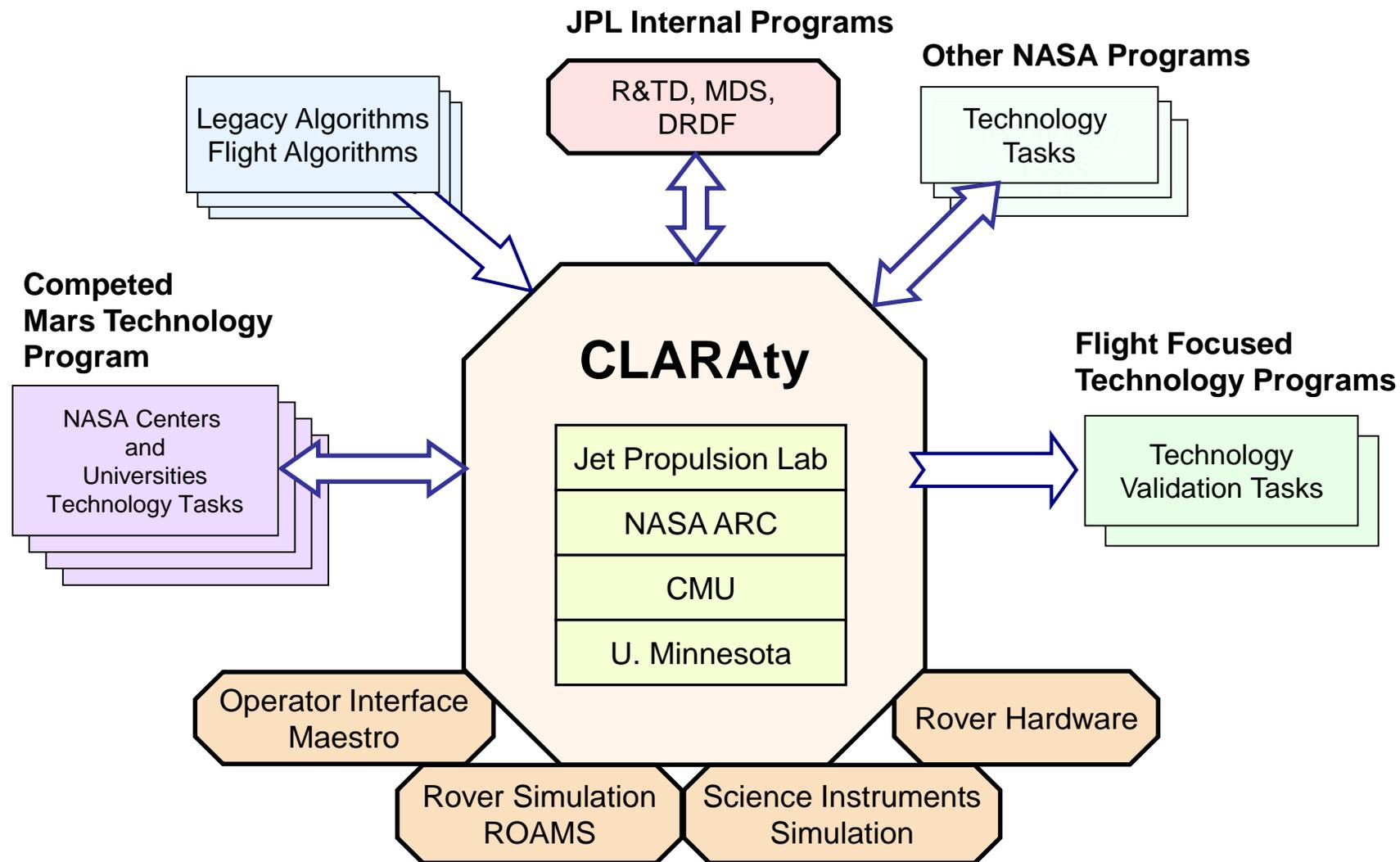


Education

- **Extensions** in Computer Science
- **Ph.D.** and **M.S.** in Mechanical Engineering Robotics
- **B.E.** in Electrical Engineering



A Robotics Autonomous Software



Autonomous Navigation in Rough Terrain

Goal:

- Rover control
- Rover navigation
- Path planning with continuous replanning
- Terrain Traversability analysis
- Multi-stereo data fusion
- Visual odometry
- Stereovision
- Inertial sensing and estimation
- Manipulation (mast)
- Locomotion
- Mechanism model
- Rover/mast kinematics
- Trajectory generation
- Servo (PID control)
- I/O control

15 m straight forward



DuAxel rover concept with rappelling capability

Microgravity Mobility

How to Explore the Surface
of
Comets and Asteroids



Jet Propulsion Laboratory
California Institute of Technology

COURSE OBJECTIVES



Course Objectives



Become familiar with some of the:

- **Space environment**
- **Challenges** of reusable robotic software
- **Approaches** to architecting robotic software
- **Architectural themes** from practical developments
- Aspects of **system state, uncertainty** and **models**
- Aspects of **software interoperability**
- Influence of **autonomy** on robotic software
- Influence of **system health management**

Work in some detail through examples on:

- Rover **Mobility**
- Rover **Navigation**

Course Overview



- **Lecture 1: Introduction**
 - Background on space robotics
 - Challenges due to robotic heterogeneity
 - Impact of autonomy
 - Understanding the space environment
 - Flight and research robotics software
 - State of the practice
 - Migration of software to flight
 - Summary



- **Lecture 2: Architectural review**
 - Review of Robotic Software Architectures with closer look at:
 - NASREM -> 4D-RCS, ControlShell, Mission Data Systems, LAAS, CLARAty and ROS
 - Architectural and design elements
 - Layered
 - Blackboard
 - Component-connector
 - Object-oriented design
 - Design patterns
 - Data flow patterns (synchronous and asynchronous)
 - Event-based programming, and finite state machines

Course Overview



- **Lecture 3: Architectural themes**
 - Common architectural themes
 - Reflections: advocacy and criticism
 - Lessons learned
- **Lecture 4: Commanding, state and health**
 - Common architectural themes (continued)
 - Robot commanding (sequences vs. task networks)
 - System health management
 - System state

Course Overview



- **Lecture 5: Navigation example**
 - Perception (orbital and rover)
 - Traversability and hazard assessment (geometric and non-geometric hazards)

- **Lecture 6: Navigation example (continued)**
 - Traversability and hazard assessment (continued)
 - Motion planning
 - Navigation architecture and interoperability
 - What lies ahead
 - Summary
 - Concluding thoughts

BACKGROUND

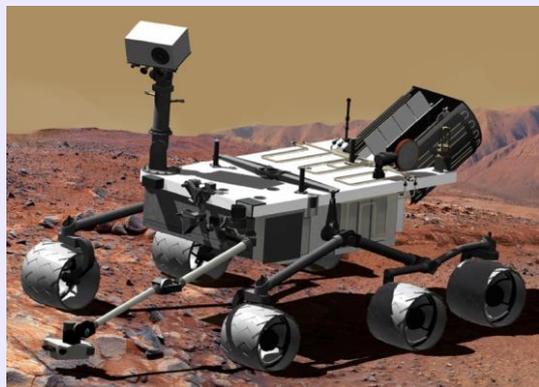
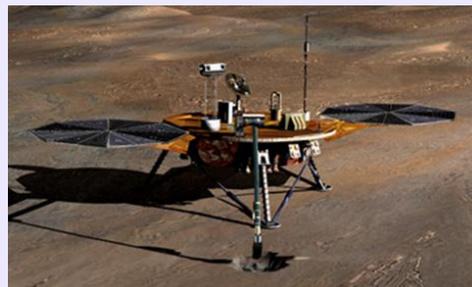
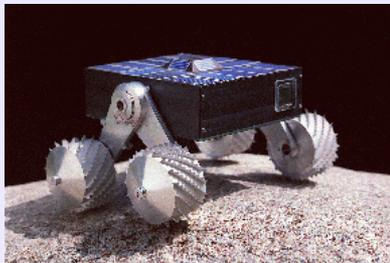
SPACE ROBOTICS



Examples of Space Robotic Systems



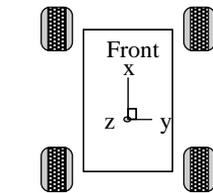
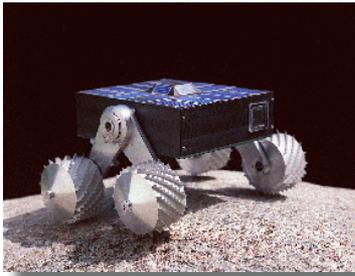
Flight Robots



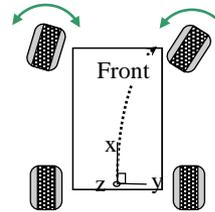
Research Robots



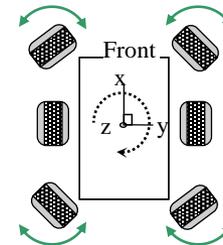
Variations Even With a Family



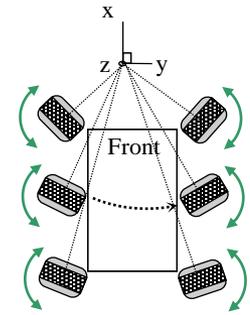
(a)
Skid Steerable
(no steering wheels)



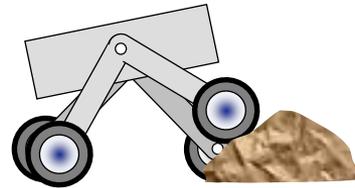
(b)
Partially steerable



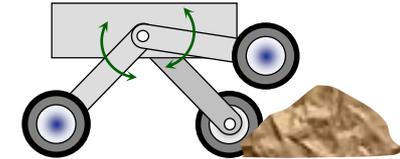
(c)
Partially steerable



(d)
Fully-steerable

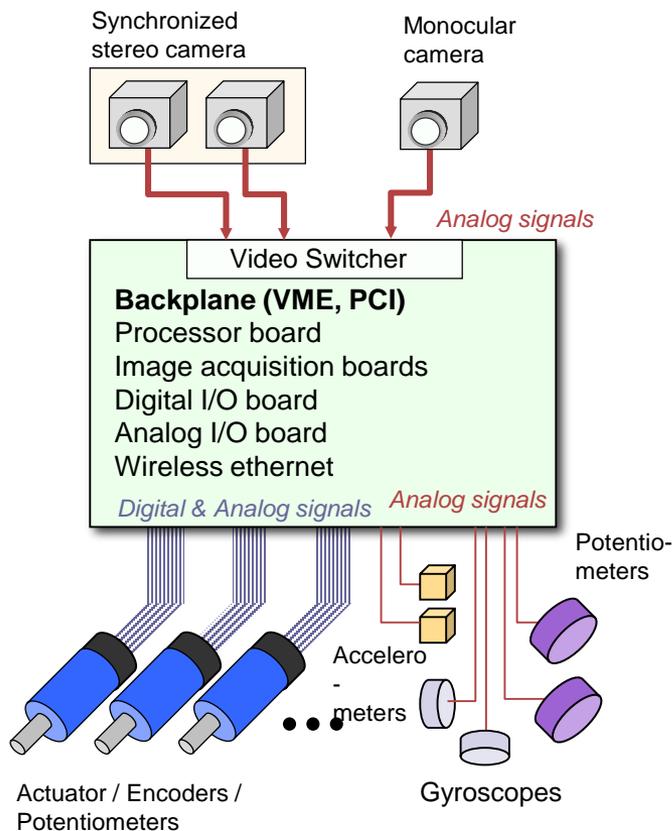


(e)
Passive Suspension (complies to terrain)

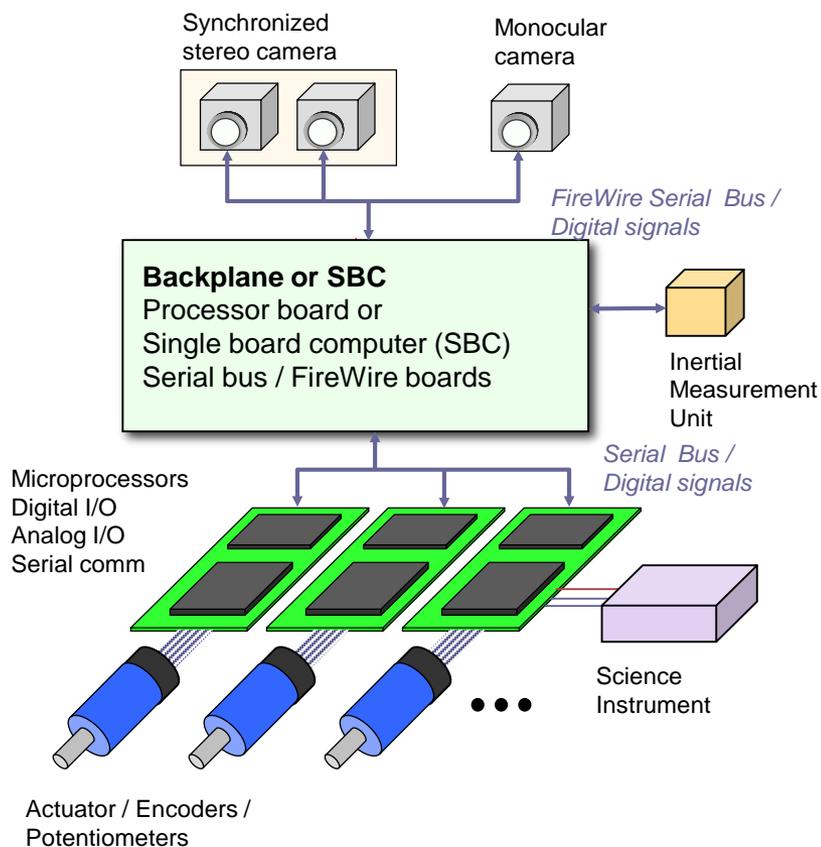


(f)
Active Suspension (actuated links)

Centralized vs. Distributed Architectures

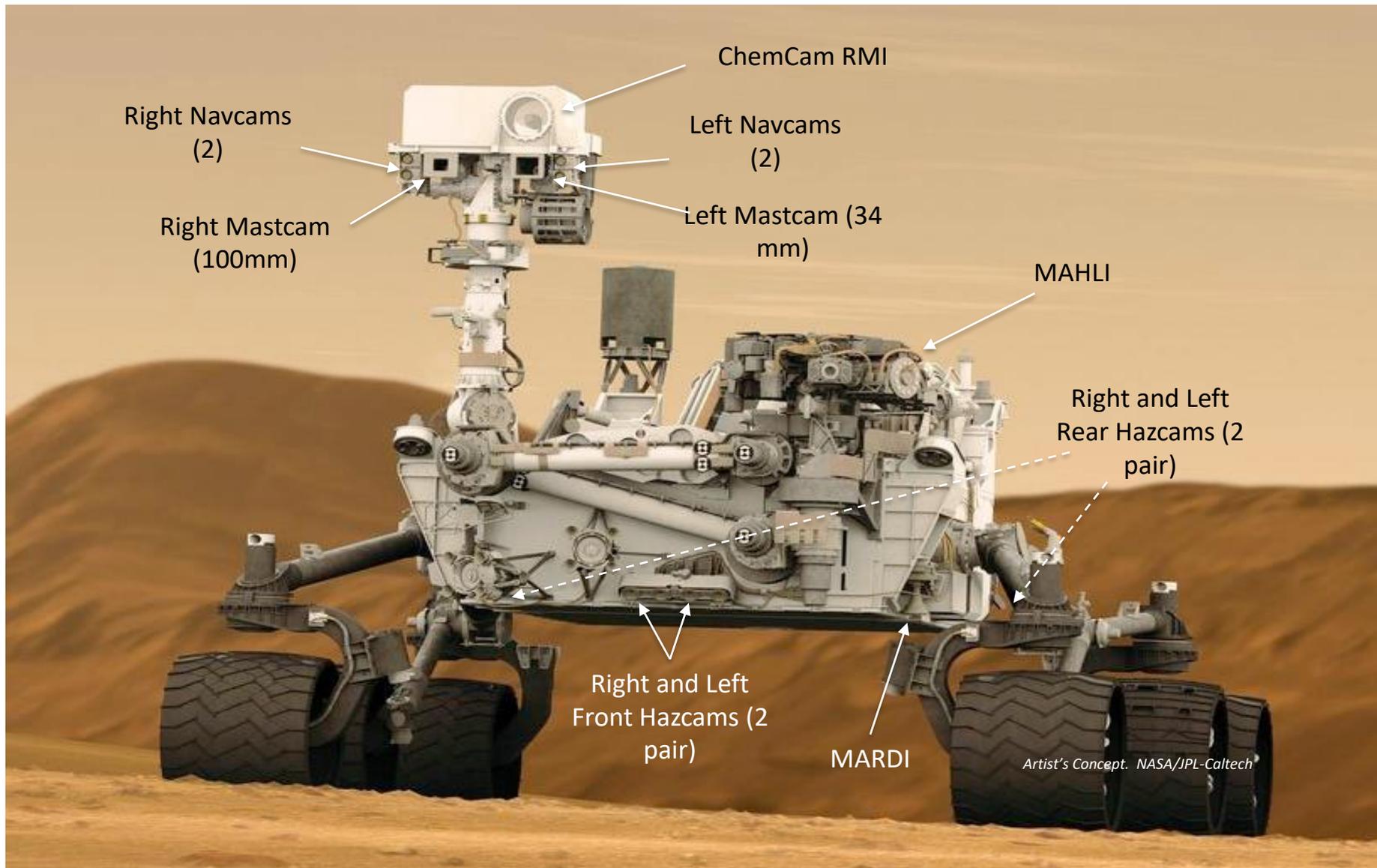


FIDO



Athena

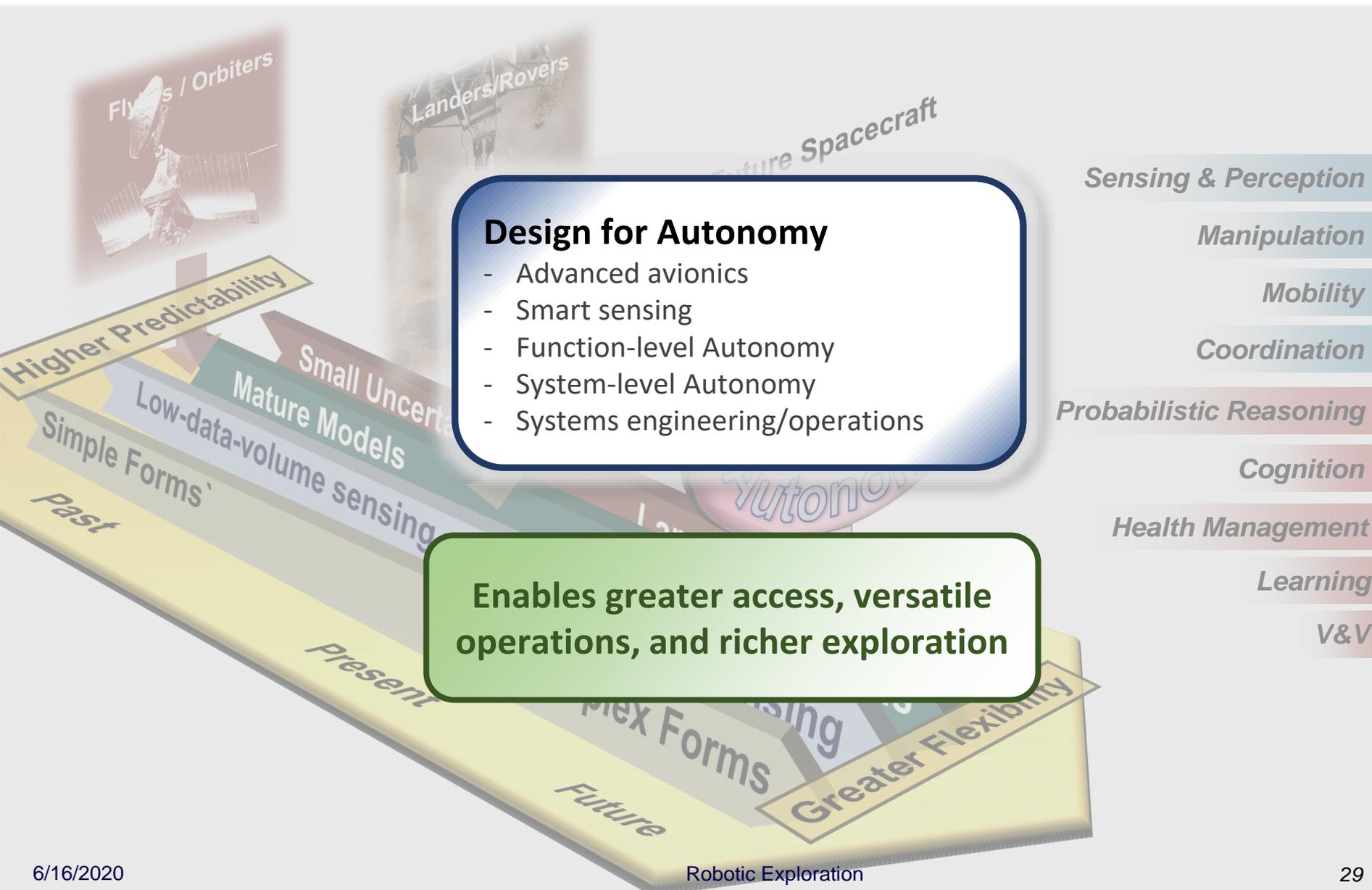
Complexities and Constraints



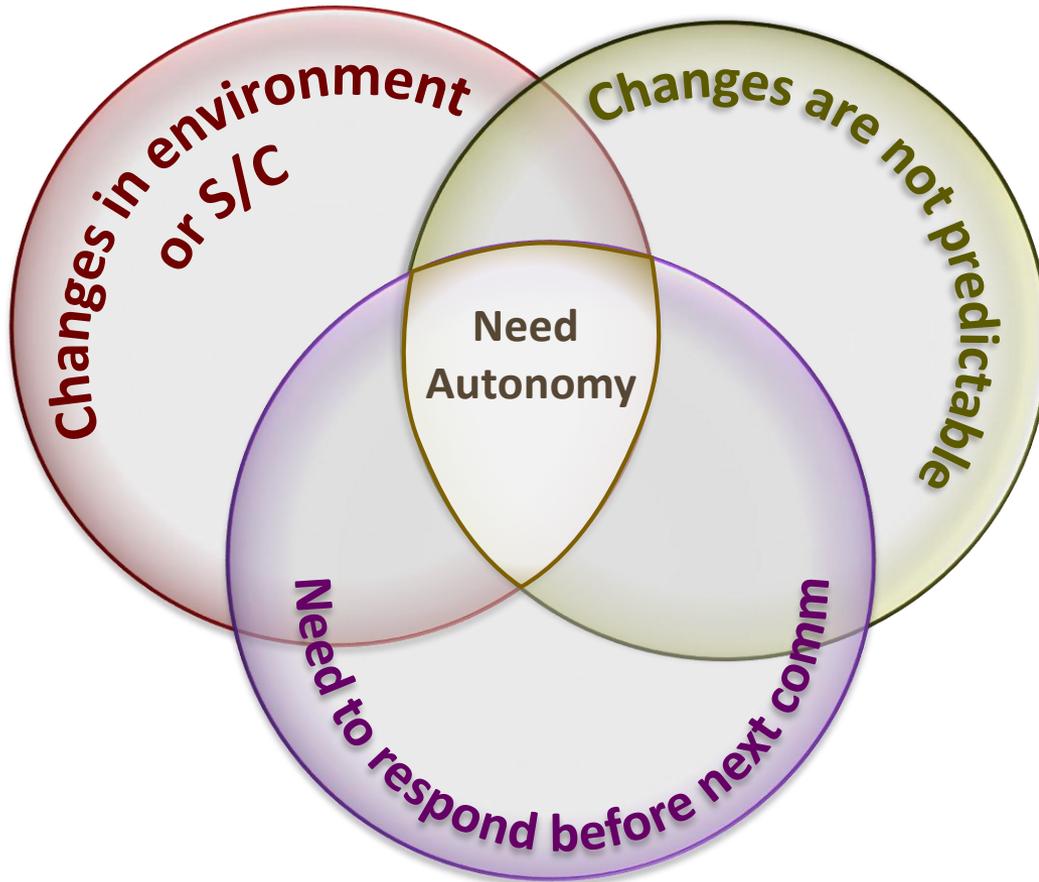
AUTONOMY



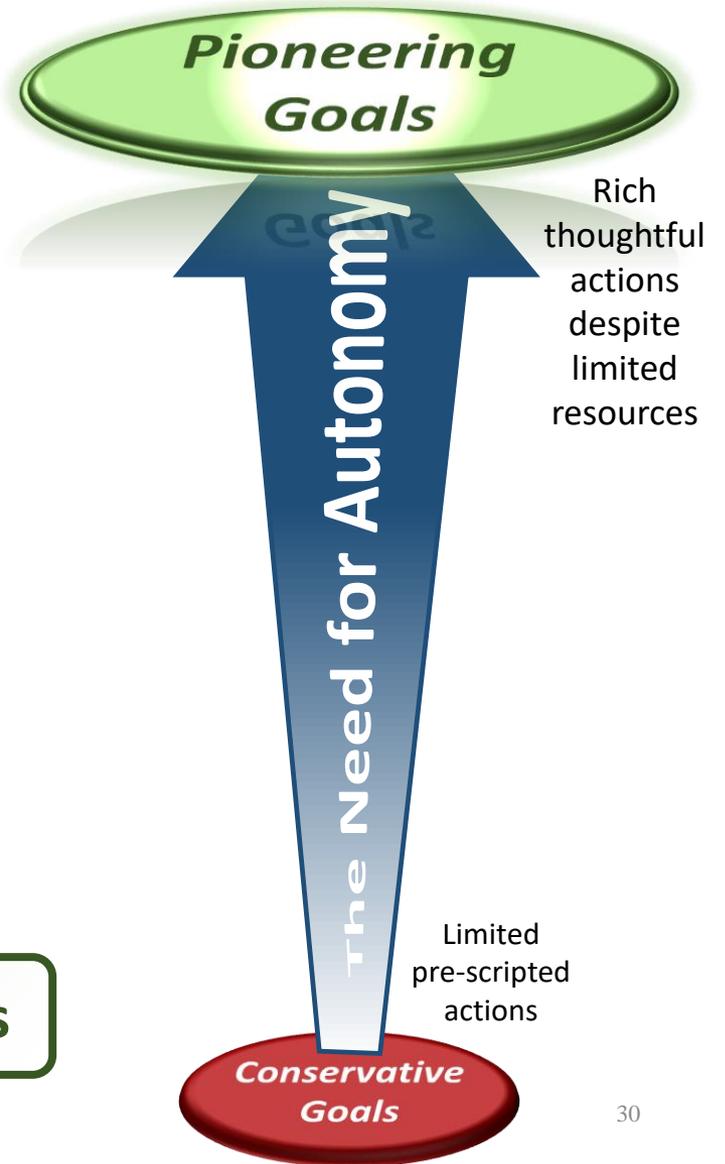
Autonomy for Future Exploration



The Need for Autonomy



Autonomy enables pioneering missions



FLIGHT SOFTWARE



Challenges for Flight Robotic Software



- **Space environment**

- Remote – communication windows and time delays
- Radiation
 - Single-event upsets
 - Total dose
- Unknown environment – in particular surface and sub-surface
- Limited sources for energy
- Limited mass (limits power)

- **Computation**

- Limited processing and memory due to radiation hardened parts

- **Robustness**

- Need to always know the state of the spacecraft

Example of Computing Environment



Item	MER	MSL
Radiation-hardened CPU	RAD6000 (PowerPC)	RAD750 (PowerPC)
Clock Speed	20 MHz	133 MHz
On-board RAM	128 Mbytes	128 Mbytes
Real Time Operating System	VxWorks 5.3.1	VxWorks 6.7
Addressable Code RAM	32 MB	32 MB
FSW + RTOS Code Size	10 MB	21 MB
Additional RAM	n/a	512 Mbytes SDRAM (half for RAMFS)
Per-Task Memory access	Shared Memory	Shared Memory
C/Embedded C++ compiler	Green Hills MULTI 3.5	GCC 4.1.2

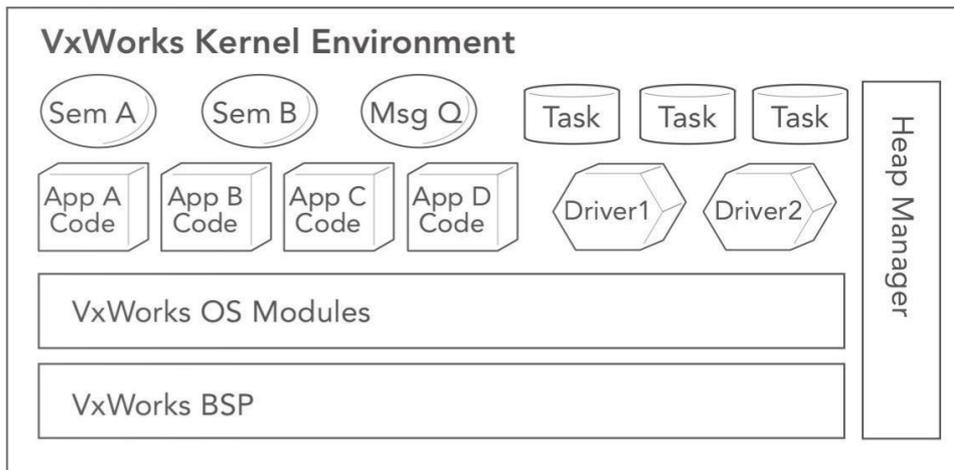
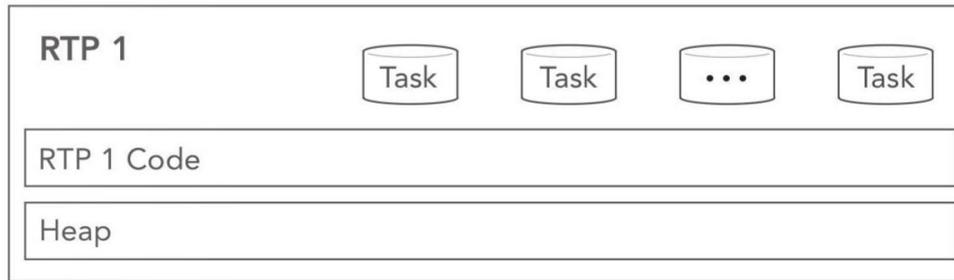
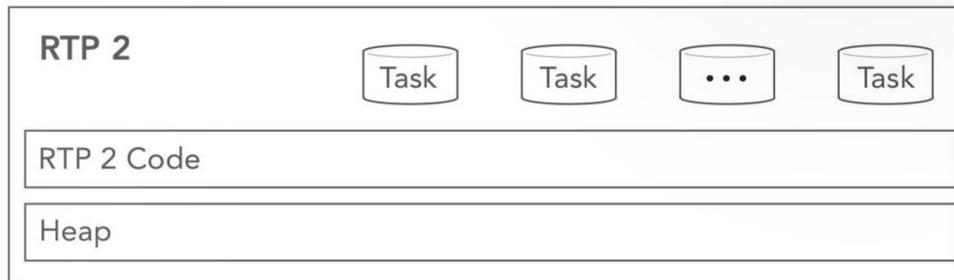
Credit: *M. Maimone*

VxWorks Task Model



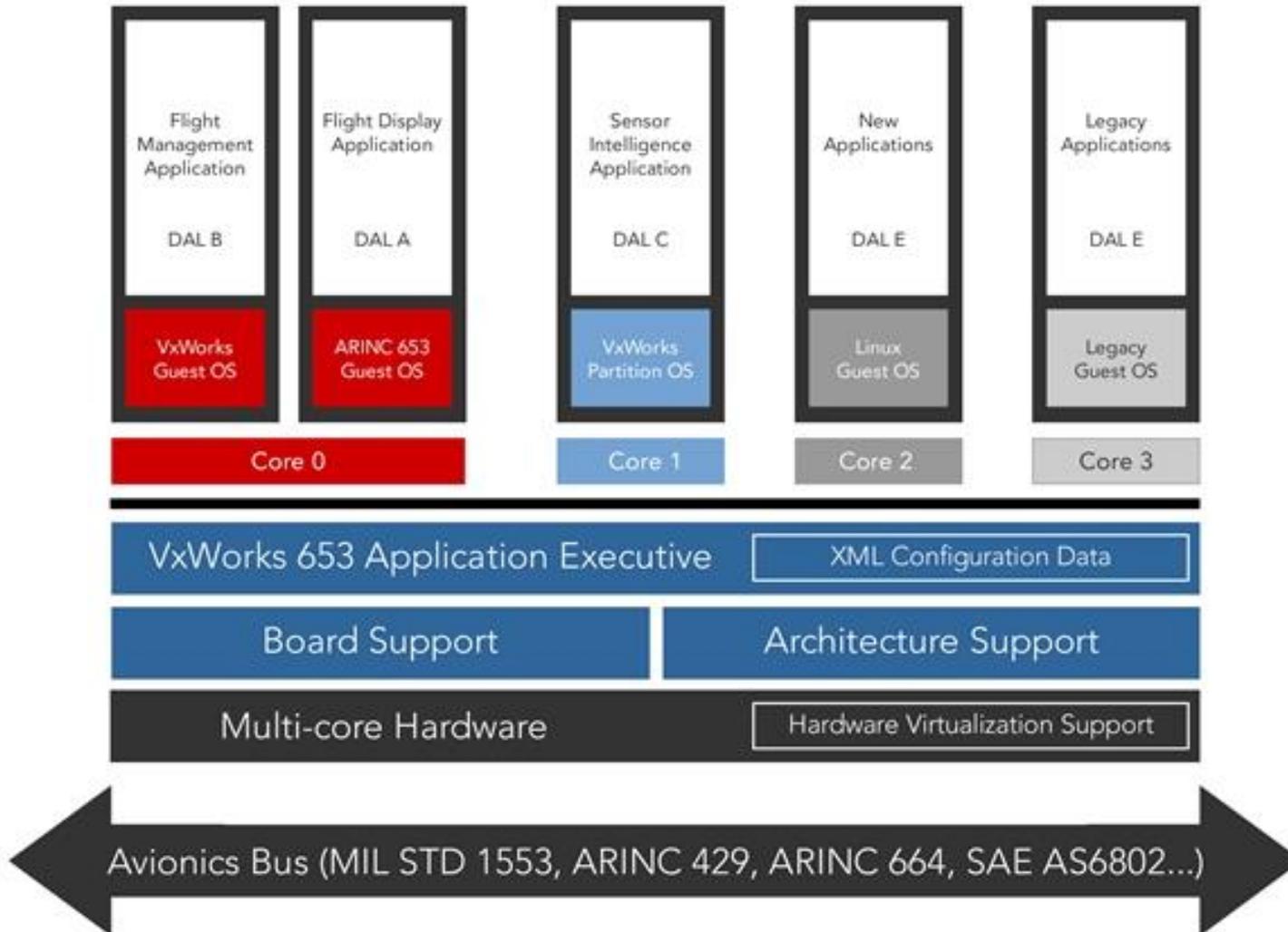
- **Real-time OS** – meets timing guarantees
- **Pre-space-time partitioning** (requires hardware support)
 - Memory
 - Flat without paging
 - Tasks
 - Are a hybrid between a process and a thread
 - Calls to kernel have low-latency and low-system overhead (context switching)
 - Share memory space
 - Accessible from console (during development)
 - Task coordination
 - Semaphores
 - Message queues

VxWorks Architecture



Credit: Wind River

Modern VxWorks Architecture





- **Key considerations**
 - Long-lived missions spanning decades (e.g. Voyager 40 years)
 - Software that is analyzable (e.g. static analysis, code coverage)
 - Software and functionality that can be verified and validated
- **State of the practice**
 - **Operating System:** real-time OS
 - Older mission flew custom OS
 - VxWorks now flies on most missions
 - Other OS – e.g. Ada: language and OS flying on Cassini
 - **Programming Language:**
 - C for rover missions with C++ exception for surface navigation
 - C/C++ for other missions

ROBOTIC RESEARCH SOFTWARE



Robotics Research Software



- **Platforms:** heterogeneous fleet
- **Operating System:** primarily real-time until recently
 - VxWorks for > 20 years on research rovers
 - QNX on some projects
 - Linux with real-time extensions currently
- **Languages:**
 - Largely C++, C, Python but also worked with Sun on real-time Java
 - Deployed object-oriented software under VxWorks on heterogeneous platforms
- **Tools:**
 - RTI's Control Shell/NDDS for 7 years
 - VxWorks tools
 - Linux tools



Robotics Research Software



Key drivers

- **Flexibility:** to support new tech capabilities
- **Affordability:** for research budgets
- **Efficiency:** to maximize test coverage
- **Extendibility:** to reduce need for re-architecting
- **Commonality:** to share functionality across platforms
- **Maintainability**



Migration to Flight



- Risk
- Flight heritage
- Product challenges
 - Cost
 - Maturity
 - Limited driver support
 - Custom Board Support Packages (BSP)
- Platform specific vs. generalized reusable solutions
- Impact of closed eco-systems
- Established knowledge base (staff background)
- Security

What does that mean for software?



- Programming: need safety critical software
 - Evaluate and understand all nuances of a language. More complex languages are more challenging to static analyze
 - Predict behavior of software over extended periods of time
 - Eliminate memory fragmentation by enforcing dynamic memory allocation at initialization only
 - Use patterns for software (e.g. tasking and message passing) to prevent deadlocks
 - Ensure all software compiles with no warnings
- The Mars Exploration Rovers were the first to fly C++
- Other projects have since used embedded C++

Ten Rules for Safety Critical Software

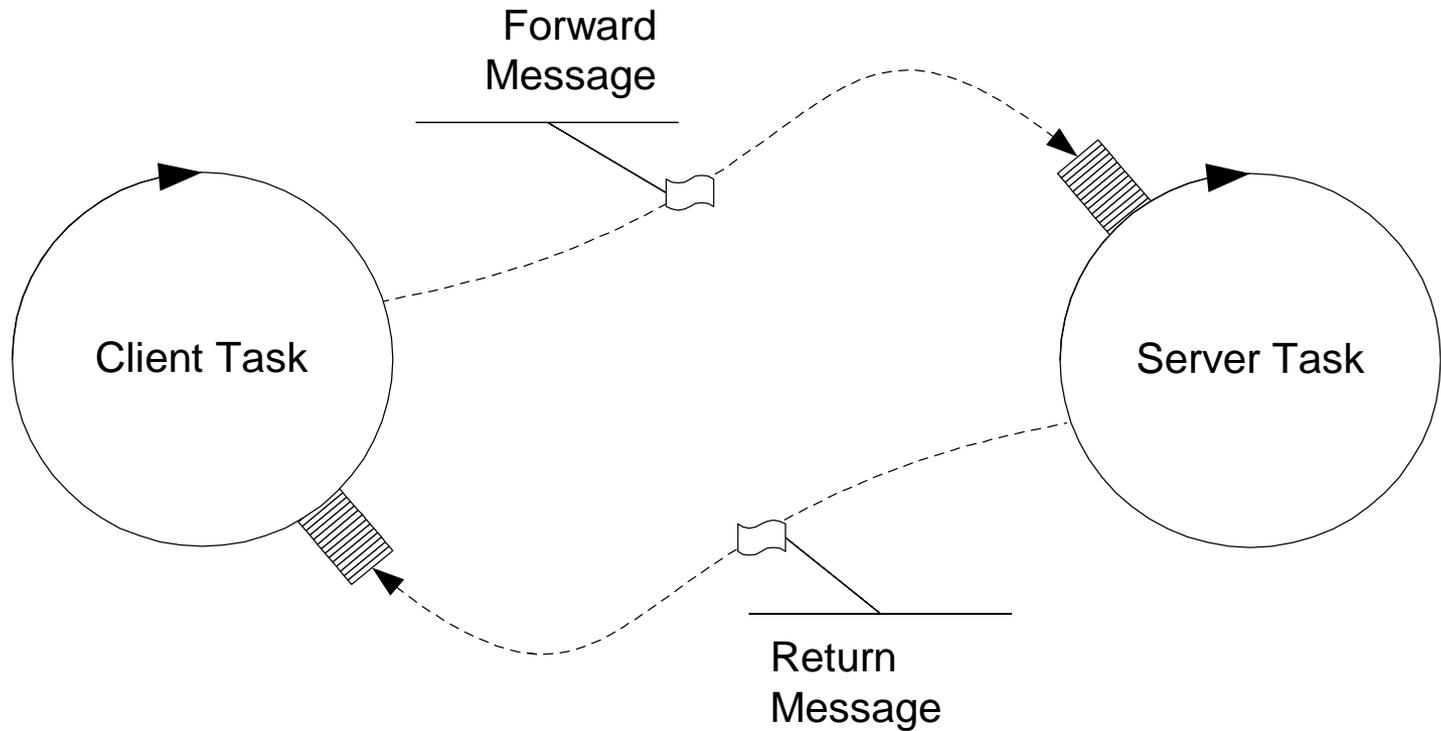


1. Avoid complex flow constructs, such as [goto](#) and [recursion](#).
2. All loops must have fixed bounds. This prevents runaway code.
3. Avoid [heap memory allocation](#).
4. Restrict functions to a single printed page.
5. Use a minimum of two [runtime assertions](#) per function.
6. Restrict the scope of data to the smallest possible.
7. Check the return value of all non-void functions, or cast to void to indicate the return value is useless.
8. Use the [preprocessor](#) sparingly.
9. Limit pointer use to a single [dereference](#), and do not use [function pointers](#).
10. Compile with all possible warnings active; all warnings should then be addressed before release of the software.

Courtesy of Gerard Holzmann

https://en.wikipedia.org/wiki/The_Power_of_10:_Rules_for_Developing_Safety-Critical_Code

IPC Message Passing and Queues in MER FSW



Basic Message/Reply Design

Summary



- Overview of JPL and a small snapshot of on-going robotics activities
- Covered an overview of the course (six lectures)
- Examined the heterogeneity from real-world examples
- Described need and impact of autonomy on robotic software
- Examined environment and constraints of flight and research software
- Examined the rules of safety-critical software
- Looked at migration of software to flight

LECTURE 2

ARCHITECTURE AND DESIGN



Presentation Overview



- Featured Video: deployment example
 - Autonomous navigation on heterogeneous rovers
- A review of robotic architectures
- Architectural styles
- Summary

Applicability of this structured decomposition is applicable outside software domain





**How do you architect
a robotic system?**

Key Considerations



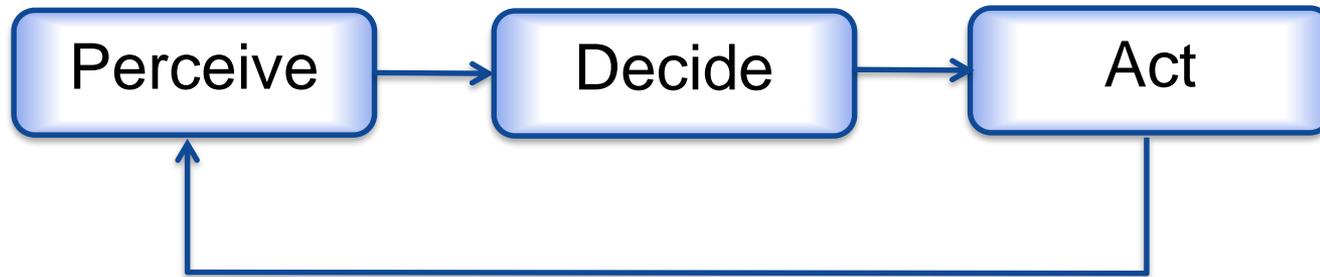
- Articulate the driving requirements
 - Is it a one-off or a family of platforms? How different are they?
 - How is the system envisioned to evolve over time?
 - Does the benefit of generality outweigh the cost?
- Understand implications of driving requirements
- Understand your system's abstractions at all levels
- Understand usage
- Consider verification and validation at requirements
- Don't aim for 100% from the 1st cycle
 - Define right balance in upfront architecting through design, prototyping, implementation and deployment
 - Evolve and mature over time

Challenges



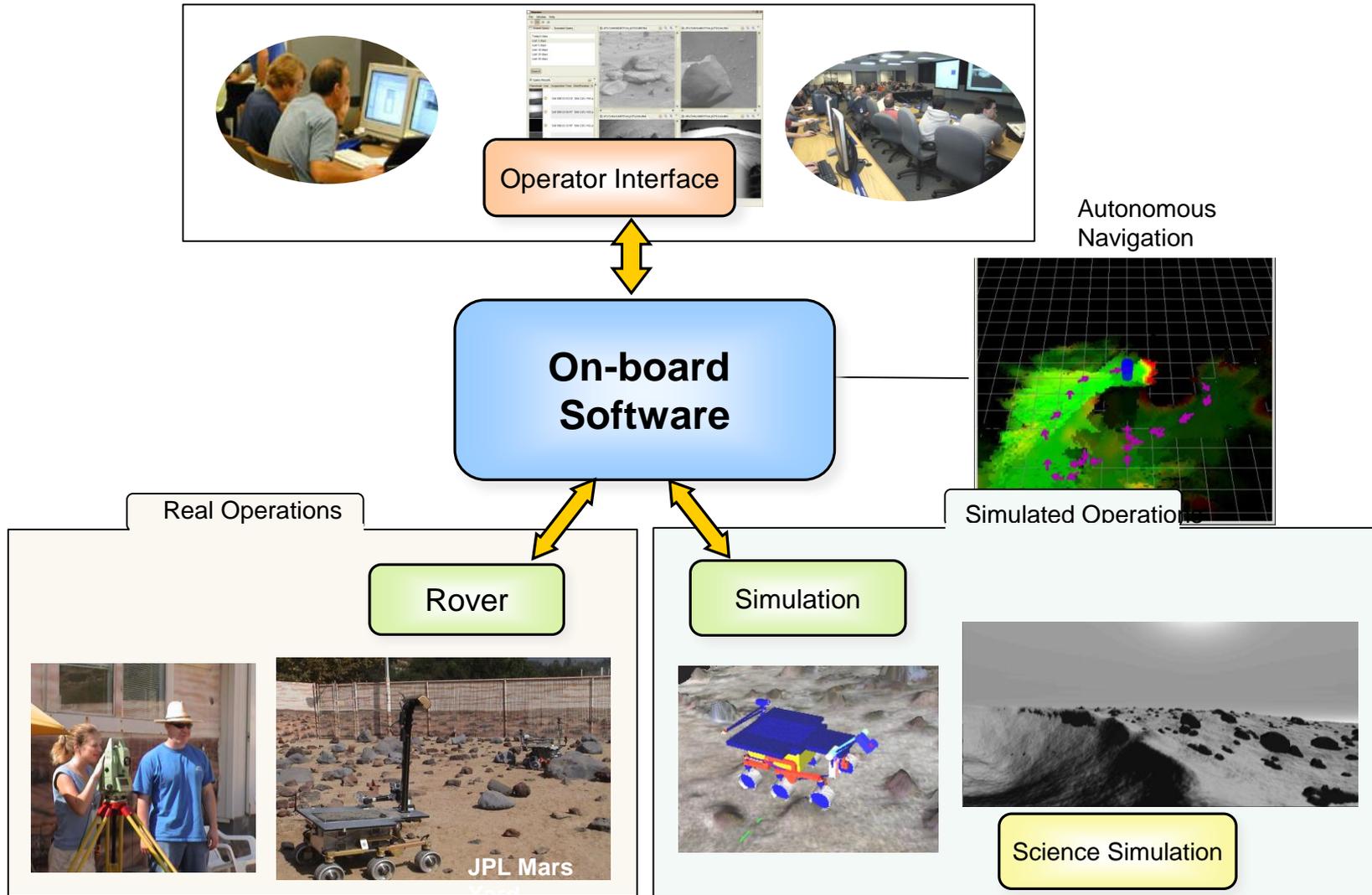
- Different physical characteristics
- Different hardware architectures
- Platform availability
- Contributions from other institutions
- Flexibility for innovation
- Handling restrictions (IP and ITAR)
- Supporting legacy software
- Scalability to complete systems

What is Robotic Autonomy?



Autonomy: *To make decisions and take actions, in the presence of uncertainty, to execute the mission and respond to internal and external changes without human intervention.*

End-to-End Robotic Systems



A BRIEF REVIEW OF ROBOTIC ARCHITECTURES



ARCHITECTURES AND FRAMEWORKS

Robotic Architecture Review



- Started in the **1980s – 1990s**

- In U.S.: **NASREM** (NASA Reference Architecture)  NIST (J. Albus)

- Morphed into the 4D-RCS

- In U.S.: RTI's **ControlShell**  RTI's Constellation

- Focused on the framework rather than interfaces

- In U.S.: RTI's **NDDS**  Object Management Group DDS standard

- Data Distribution Service for Real-Time Systems is machine-to-machine middleware for scalable, real-time, dependable, high-performance and interoperable data exchanges between publishers and subscribers.
- Deployed into: robotics, financial trading, air traffic control, smart grid applications

- In U.S.: JPL's **Mission Data Systems** for JPL flight projects

- State-based architecture for safety critical remotely operated system

- In U.S.: **JAUS** (Department of Defense) Joint Architecture for Unmanned Systems (1998)

- Scope: all unmanned military vehicles
- Component-based high-level message set / passing architecture

- Abstraction levels: sense, think, act
- Spatial, temporal hierarchy
- Multi-level access
- Shared memory
- Controller module as building block
 - Finite state machine with data buffers that communicate through global memory
 - Non-blocking I/O
 - Cyclical sampling rather than interrupts
- Synchronous control at low levels and asynchronous control at higher levels

Albus, James S., H. McCain, and Ronald Lumia. "NASA/NBS standard reference model for telerobot control system architecture (NASREM)." *Technical Note (NIST TN)-1235* (1989).

4D-RCS



Some themes persist:

- Abstraction levels
- Spatial, temporal hierarchy
- Multi-level access
- Controller module as building block

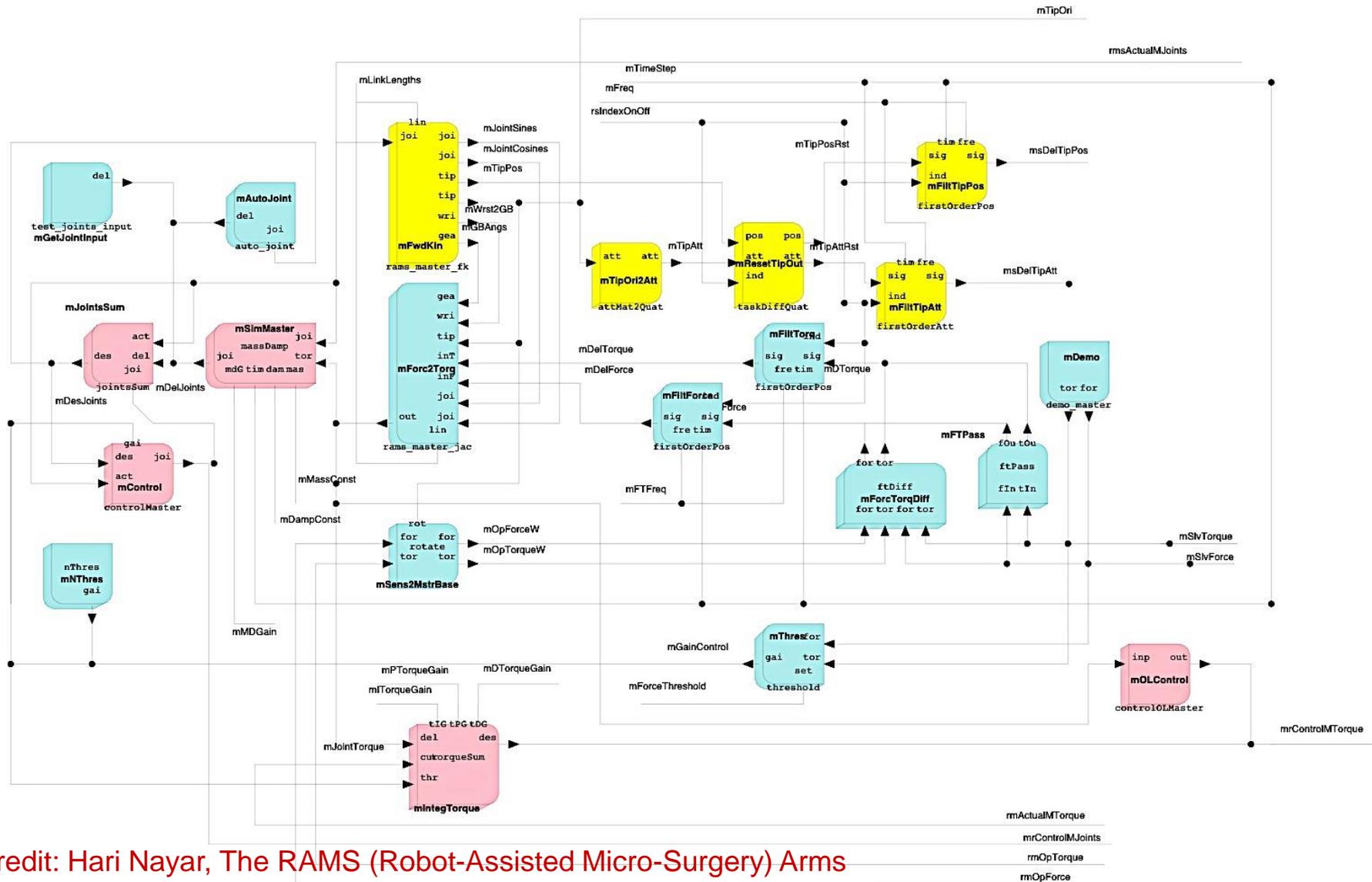
New themes: each node is:

- Goal-driven, model-based, and closed-loop
- Can decompose goals into actions (i.e. planning and execution local to each node)
- Local world models

Criticisms, according to Balakirsky (2003): because planning is performed on world model rather than on actual world, the validity of plans is questioned given planning delays [1]

[1] https://en.wikipedia.org/wiki/4D-RCS_Reference_Model_Architecture

ControlShell Data Flow



Credit: Hari Nayar, The RAMS (Robot-Assisted Micro-Surgery) Arms

ControlShell FSM



- 1992–1997

- Data Flow Elements (DFE)
 - Inputs and outputs: integers and float matrices
 - Data flows by copy
 - Run-time configurable
- Finite State Machines (FSMs)
 - State transitions
- Component scheduler
- Network Data Distribution System (NDDS)

- 1997-1999

- Hierarchical Components (Cog)
 - Contains DFEs and FSMs
 - Uses connectors (primitives and user defined interfaces)
- NDDS

Credit: <https://www-robotics.jpl.nasa.gov/systems/system.cfm?System=9>

Robotic Architecture Review



- **2000s – 2010s**

- In France, the **LAAS-CNRS** system architecture (1990s and 2000s)
 - Layered decomposition of Functional Level / Decision Level

- In U.S.: **Player/Stage** (2000)

- Developed at USC; client server architecture; supports multiple COTS rovers; most recognized; indoor robots

- In U.S.: **CLARAty** (JPL, CMU, NASA ARC, U. Minnesota) (2000-2007)

- Developed generic interface to enable interoperable software
- Support integration and deployment of competed technology for the Mars Technology Program

- In Europe, **OROCOS** (2001)

- Funded by in part by the EU and led by K.U. Leuven
- Provides CORBA-based real-time tool kit, bayesian filtering library and kinematics and dynamics

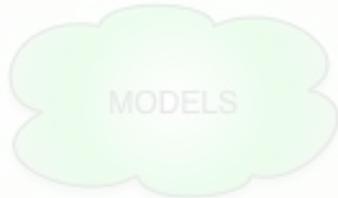
- In U.S.: Intel's **RETF** (Robotics Engineering Task Force) (2002)

- Modeled after IETF (Internet Engineering Task Force)
- Defining standardized robotics interfaces

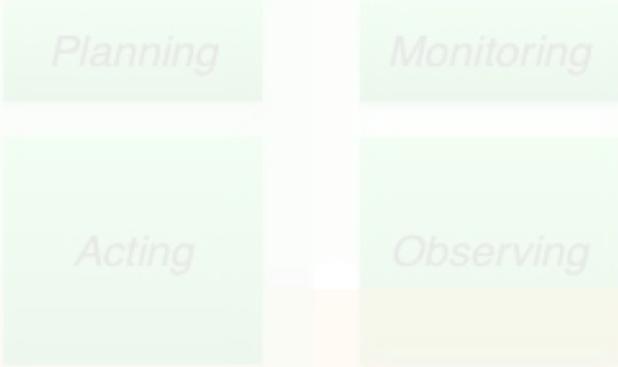
LAAS



Decisional Level



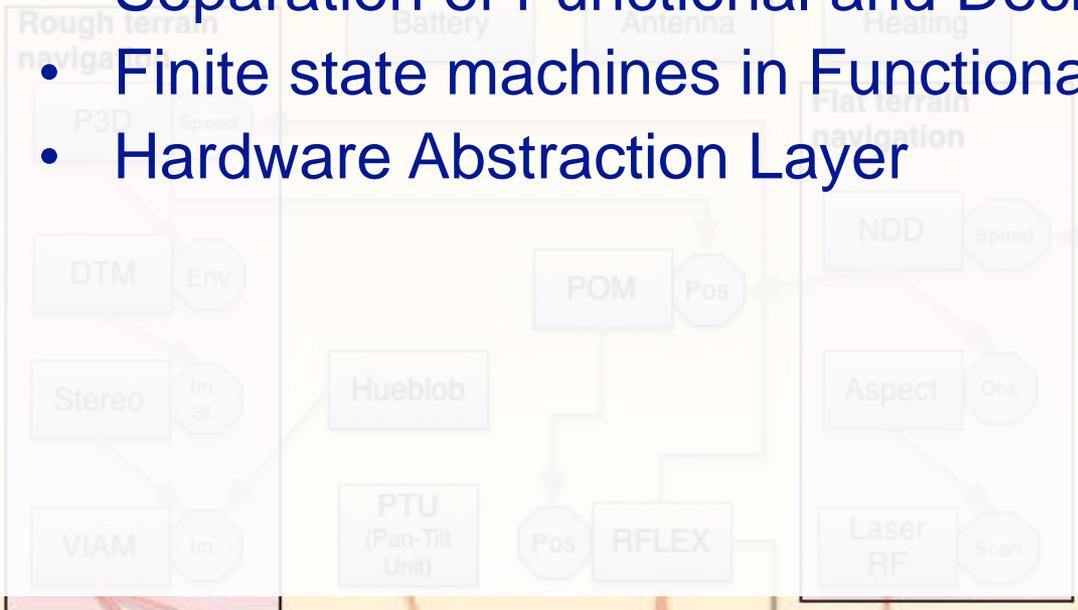
FAPE



Functional Level

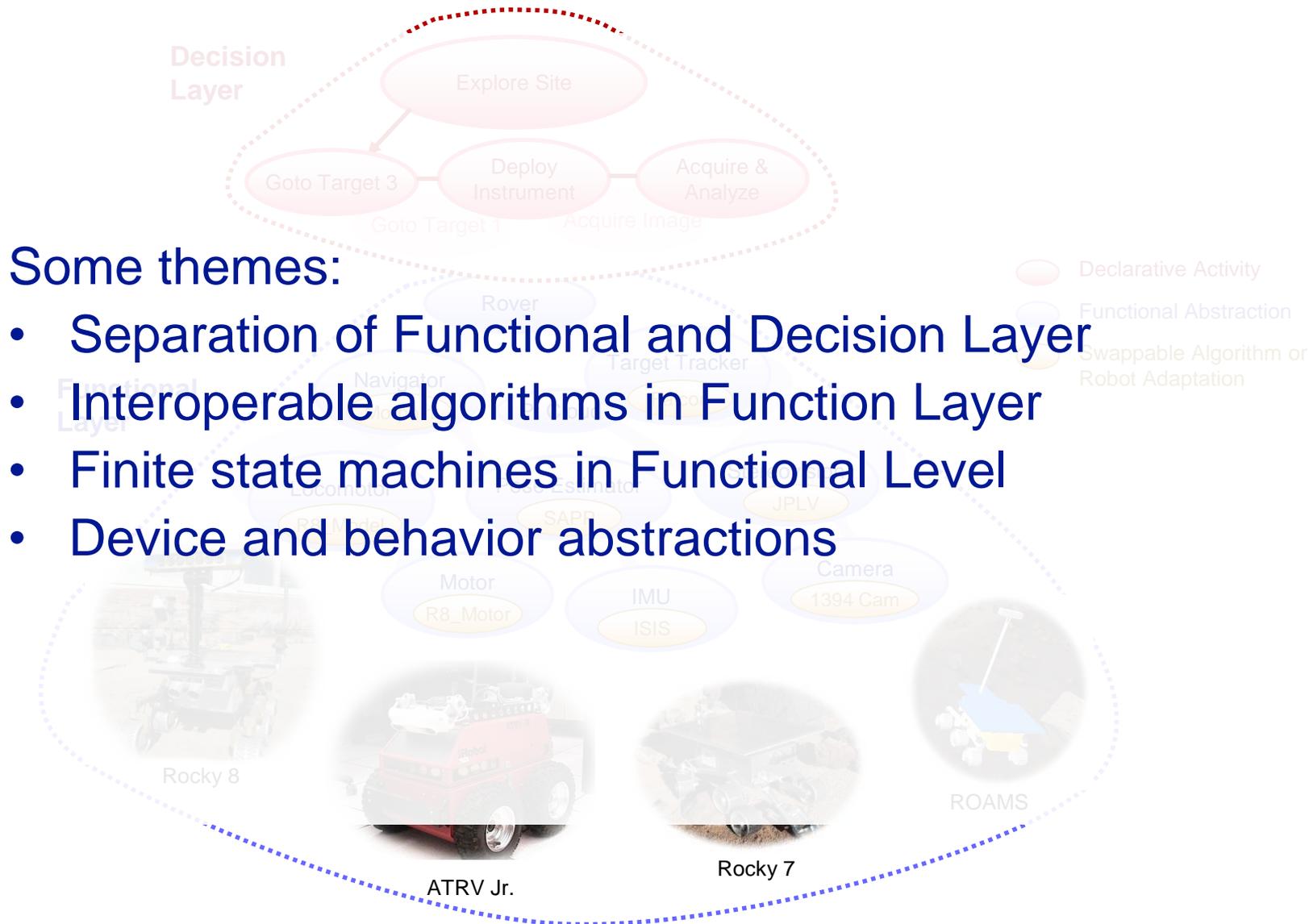
Some themes:

- Separation of Functional and Decision Level
- Finite state machines in Functional Level
- Hardware Abstraction Layer



Credit: Felix Ingrid, LAAS

CLARAty Architectural Concept



Some themes:

- Separation of Functional and Decision Layer
- Interoperable algorithms in Functional Layer
- Finite state machines in Functional Level
- Device and behavior abstractions

CLARAty Abstractions/Components



Decision Layer

- Planners
 - Activity, Plan

Functional Layer

- Executives
- Behaviors
 - Locomotor, Pose_Estimator, Manipulator, Navigator, etc.
- Models
 - Motor_Model, Camera_Model, Mechanism_Model, etc.
- Devices
 - Motor, Camera, IMU, 3D sensor, etc.
- Data structures
 - Array, Matrix, Image, Map, Message, Resource, etc.

Generic and specialized



I.A. Nesnas, et.al., "[CLARAty: Challenges and Steps Toward Reusable Robotic Software](#)," International Journal of Advanced Robotic Systems, Vol. 3, No. 1, pp. 023-030, 2006.



- **2000s – 2010s (continued)**
 - In U.S.: NASA's **JTARS** (multiple centers and universities) (2005)
 - In U.S.: Microsoft **Robotics Studio** (2007)
 - Service-oriented architecture; XML based message passing; supports heterogeneous programming languages
 - In Germany: RoSTA (2007)
 - Funded by the EU and led by Fraunhofer Institute
 - In U.S.: Willow Garage's **ROS** (2008)
 - In Europe: **BRiCS** (Best Practice in Robotics) (2009)
 - In Japan: AIST's **Open-R** for humanoid robots
 - In U.S.: **CARMEN** at Carnegie Mellon for Robot Navigation
 - In Canada: **MARIE** Sherbrooke University
 - In France: **URBIE** (robotic programming language)
 - In U.S.: **Aware** from IRobot

Robotic Architectural Review



- **2010s – Present**

- In U.S.: Open Source Robotics Foundation **ROS** (2013 –)

- Probably the largest repository of robotics software
 - Publish subscribe model for interoperability
 - Focus on indoor robotics

- In U.S.: NASA/JPL Software and Robotics Frameworks (on-going)

And many others:

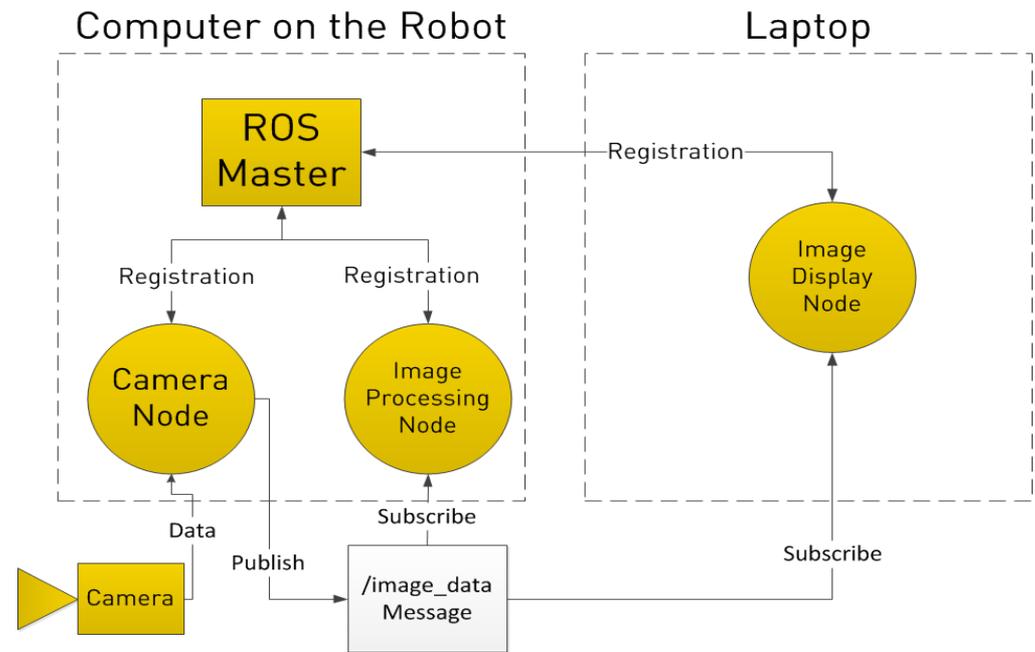
- Miro (for the Robocup competition; Corba-based real-time framework), ESRP from Evolution, ROCI from U. Penn, OSCAR from U. Texas, ARIA from MobileRobots

What is ROS exactly?

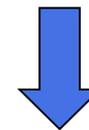
- **Plumbing:** publish-subscribe messaging for distributed computing
- **Tools:** for configuring, introspecting, debugging, visualizing, logging, testing, and managing distributed computing.
- **Capabilities:** functional libraries (mobility, manipulation, and perception)
- **Ecosystem:** a community with a focus on integration and documentation.

Adapted from B. Gerkey post on:
<https://answers.ros.org/question/12230/what-is-ros-exactly-middleware-framework-operating-system/>

Check and replace if picture is copyright restricted
<http://robohub.org/wp-content/uploads/2014/01/ros101-3.png>



Coupled
plumbing + functionality



Separated
plumbing from functionality
(e.g. OMPL and PCL)

ARCHITECTURAL STYLES



APPROACH AND CHALLENGES

Architectures and Frameworks



- Reusable Framework
 - Domain agnostic
- Reusable/Interoperable Components
 - Domain specific



Robotics

- Layered (2, 3)
 - Deliberative
 - Reactive
(subsumption)
- State-based

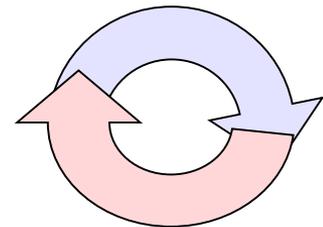
Software

- Object-oriented
- Component-based
- Event-based
- Publish-subscribe
- Service-oriented

Technical Approach (for Domain-Specific)



1. Capture **requirements** from domain experts
2. Use **global perspective** across domains (motion, perception, estimation, navigation, autonomy)
3. Identify **recurring patterns** and **common infrastructure** therein
4. Use **domain expert** to guide design
5. Define **appropriate interfaces** for each subsystem
6. Develop **generic framework** to support various implementations
7. Adapt **legacy implementations** to validate framework
8. **Encapsulate** when re-factoring is not feasible or affordable
9. Develop **regression tests**
10. **Test** on multiple robotic platforms and **study limitations**
11. **Feed** learned experience **back** into the design
12. **Review** and **update** to address limitations



After several iterations one hopes to have achieved a truly reusable infrastructure

Algorithm Infusion Challenges

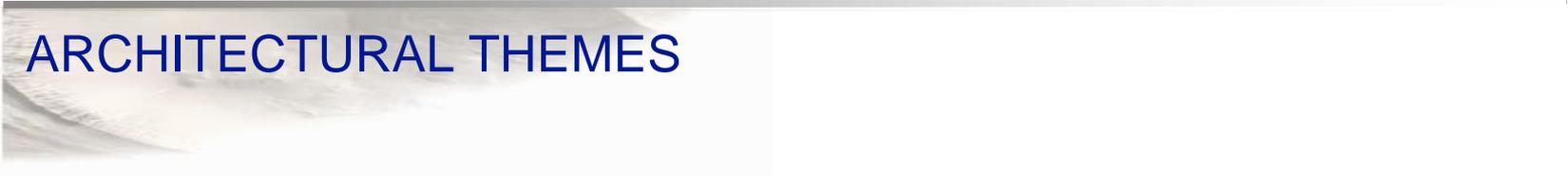


The *new* algorithms to be integrated may:

- Have architectural mismatches with the framework
- Include multiple orthogonal functionalities
- Make implicit assumptions about the platform
- Duplicate functionality in the framework
- Use incompatible data structures
- Be complex and hard to tune
- Depend on specific platform
- Require highly specialized domain expertise
- Be poorly implemented

LECTURE 3

ARCHITECTURAL THEMES



Presentation Overview



- Featured Video: deployment example
 - Autonomous navigation on heterogeneous rovers
- Common architectural themes (part 1)
- Reflections on architectural themes: advocacy and criticism (part 1)
- Architectural styles
- Summary

Applicability of this structured decomposition is applicable outside software domain

ARCHITECTURES AND FRAMEWORKS

ARCHITECTURAL THEMES



Acknowledgement: CLARAty Developers (Core Team)



- NASA Ames Research Center
 - Lorenzo Flueckiger
 - Clay Kunz
 - Randy Sargent
 - Hans Utz
 - **Anne Wright**
- Carnegie Mellon
 - David Apfelbaum
 - **Reid Simmons**
 - Nick Melchior
 - Chris Urmson
- University of Minnesota
 - Stergios Roumeliotis
 - Anastasios Mourikis
 - Nikolas Trawny
- Jet Propulsion Laboratory
 - Khaled Ali
 - Ian Baldwin
 - Kelly Breed
 - Max Bajracharya
 - Antonio Diaz Calderon
 - Daniel Clouse
 - Jeffrey Edlund
 - Tara Estlin
 - Enrico Ferrentino
 - Dan Gaines
 - Won Kim
 - Richard Madison
 - Michael McHenry
 - Jack Morrison
 - Hari Das Nayar
 - **Issa A.D. Nesnas**
 - Kyohei Otsu
 - Michael Paton
 - Mihail Pivtoraiko
 - Richard Petras
 - Venkat Rajagopalan
 - Luca Randazzo
 - Rob Reid
 - Jacek Sawonwienicz
 - I-Hsiang Shu
 - Robert Steele
 - Richard Volpe

For the complete list of key former developers and contributors see:

<http://claraty.jpl.nasa.gov> -> Project -> Team

Architectural Themes

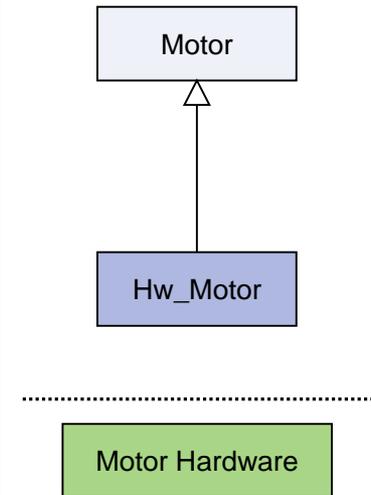
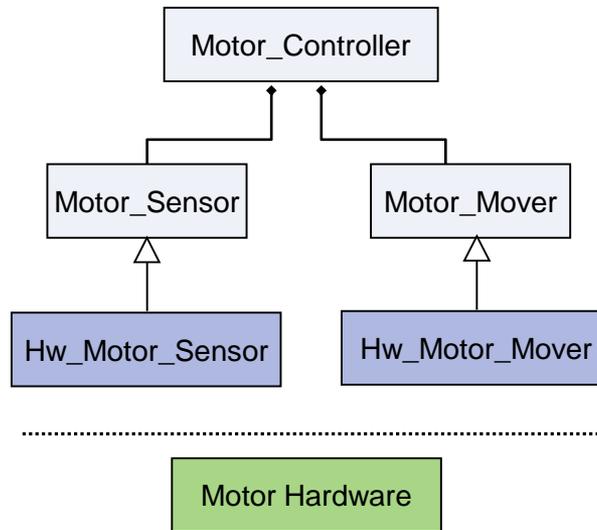
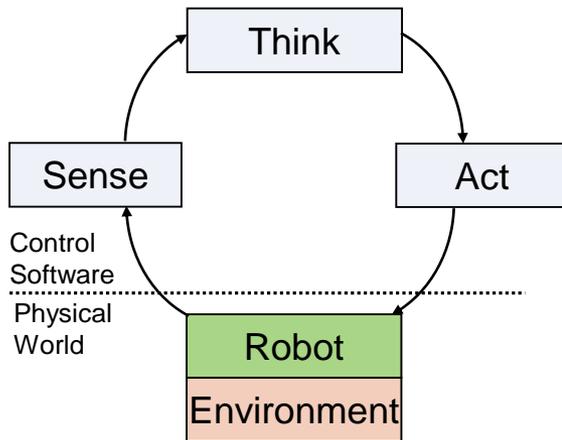


1. Abstraction hierarchy
 2. Multiple programming paradigms
 3. Multi-level access
 4. Common data structures
 5. Interoperable transformations
 6. Unified mechanism model
7. Separation of concerns
 - Estimation from control
 - Models from control
 - Logical from physical hierarchies
 - Interface from implementation
 8. Run-time encapsulation

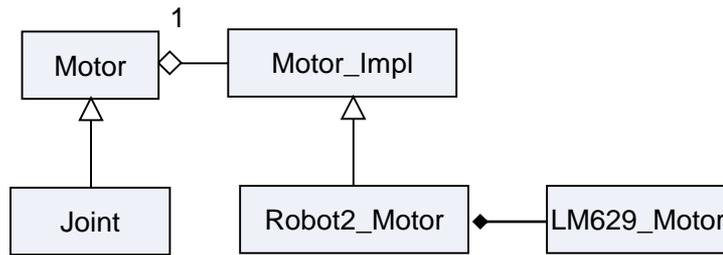
*Some recurring and
some from CLARAty*

I.A. Nesnas, "[CLARAty: A Collaborative Software for Advancing Robotic Technologies](#)," *NASA Science and Technology Conference*, University of Maryland University College, Adelphi, MD, June 2007

Theme 1: Use Abstraction Hierarchy

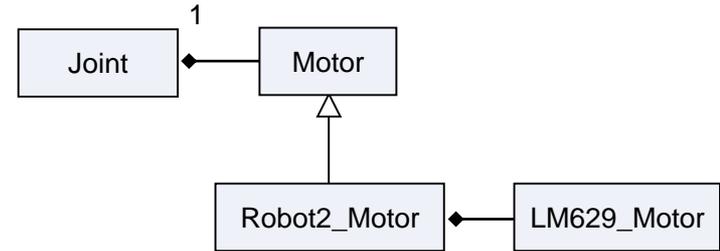


Theme V: Separating Logical from Physical Hierarchies



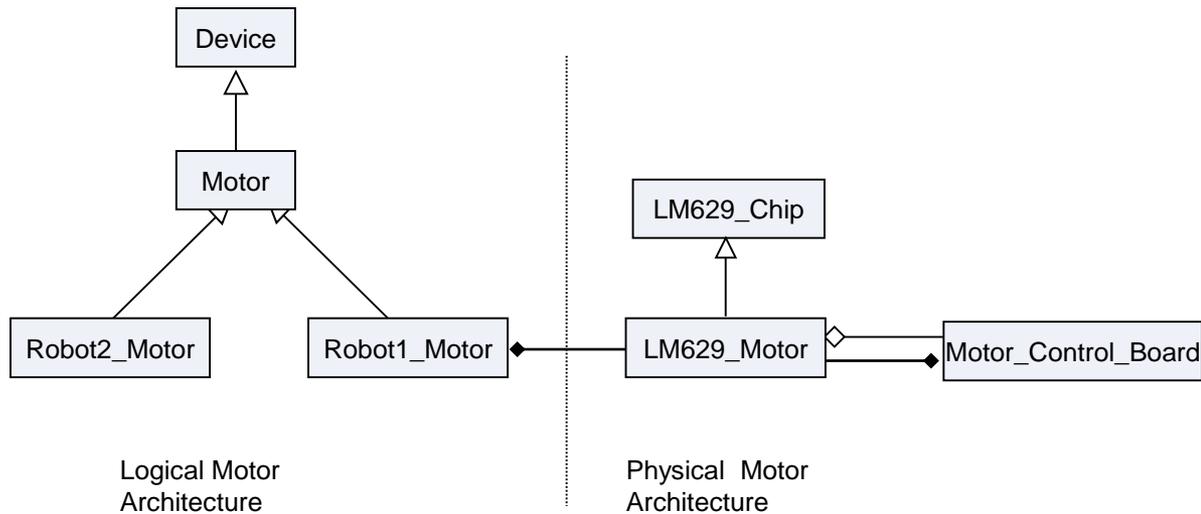
(a) Joint is a Motor

Revision 2



(b) Joint has a Motor

Revision 3



Logical Motor Architecture

Physical Motor Architecture

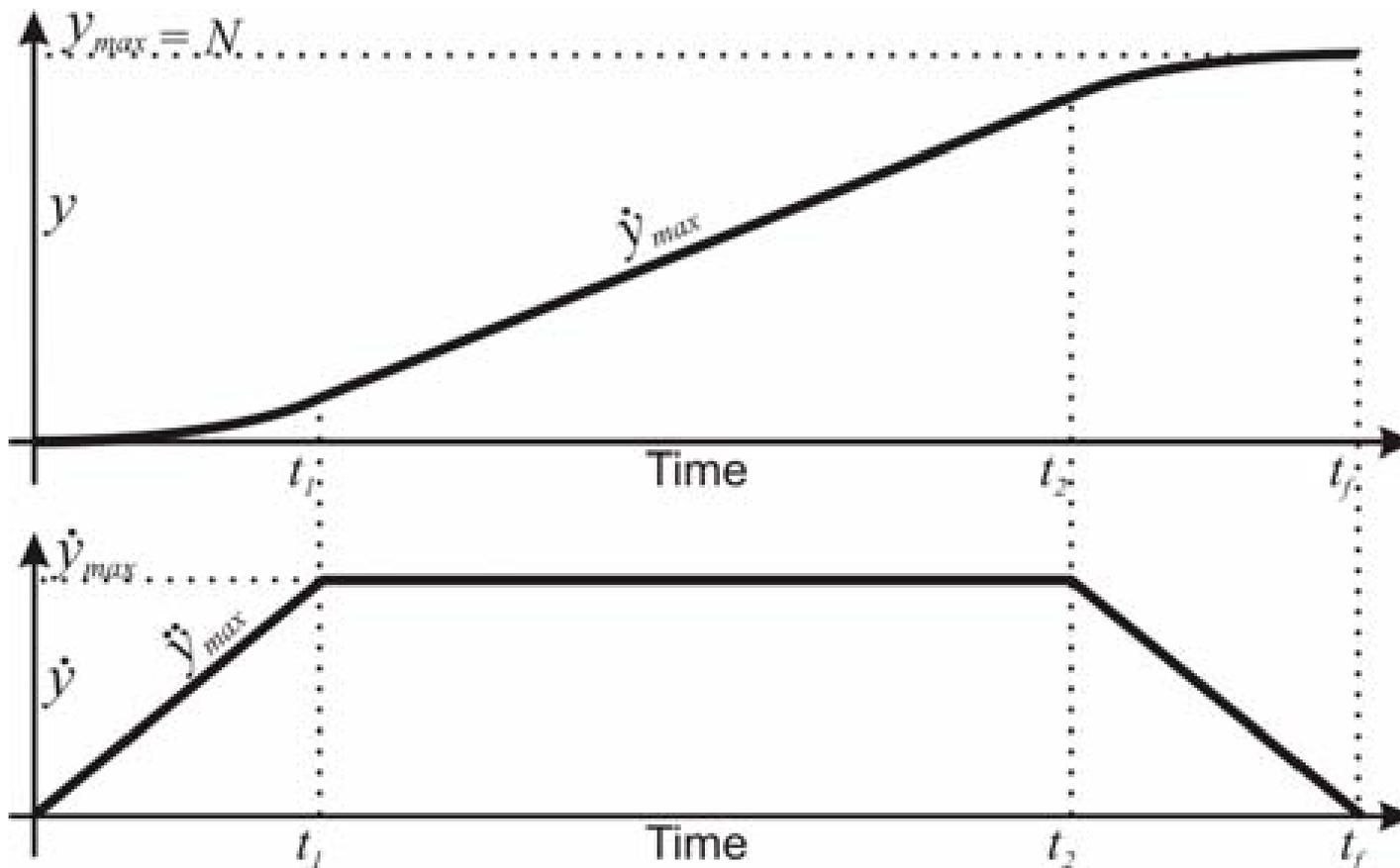
Example: Motor Generic API



```
// C++ Sample Code  
  
Motor  a_motor;  
  
a_motor.move(pos);
```

What is the problem with this code?
What is missing?

Example: Motor



M. Pivtoraiko, I.A. Nesnas, H.D. Nayar, "[A Reusable Software Framework for Rover Motion Control](#)", *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Los Angeles, CA, February 2008

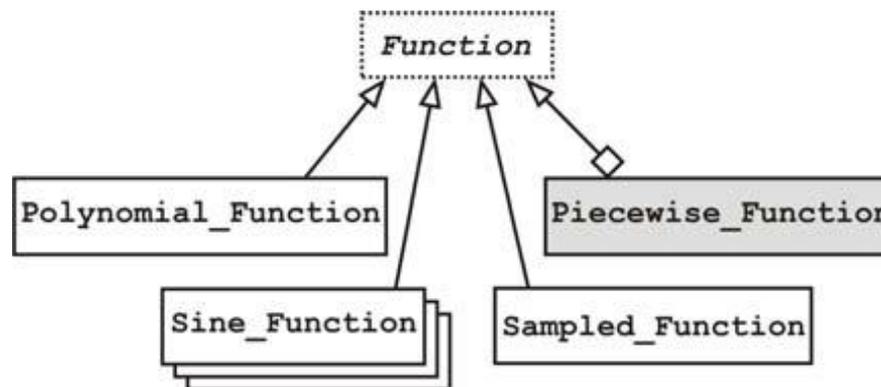
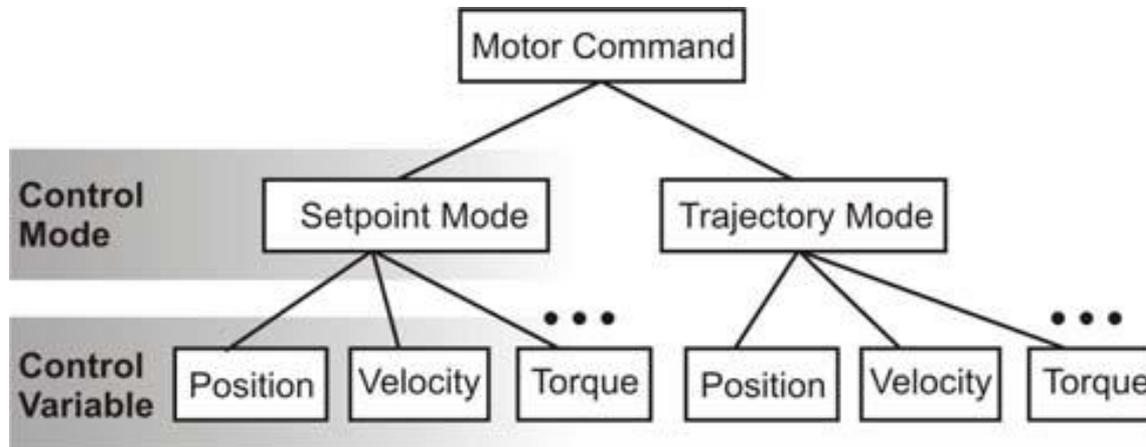
Example: Motor



```
// C++ Sample Code  
  
Motor  a_motor;  
  
a_motor.move(delta_pos, max_vel, accel, decel);
```

**So what is the problem with this code now?
What is missing?**

Example: Motor



M. Pivtoraiko, I.A. Nesnas, H.D. Nayar, ["A Reusable Software Framework for Rover Motion Control"](#), *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Los Angeles, CA, February 2008

Example: Motor



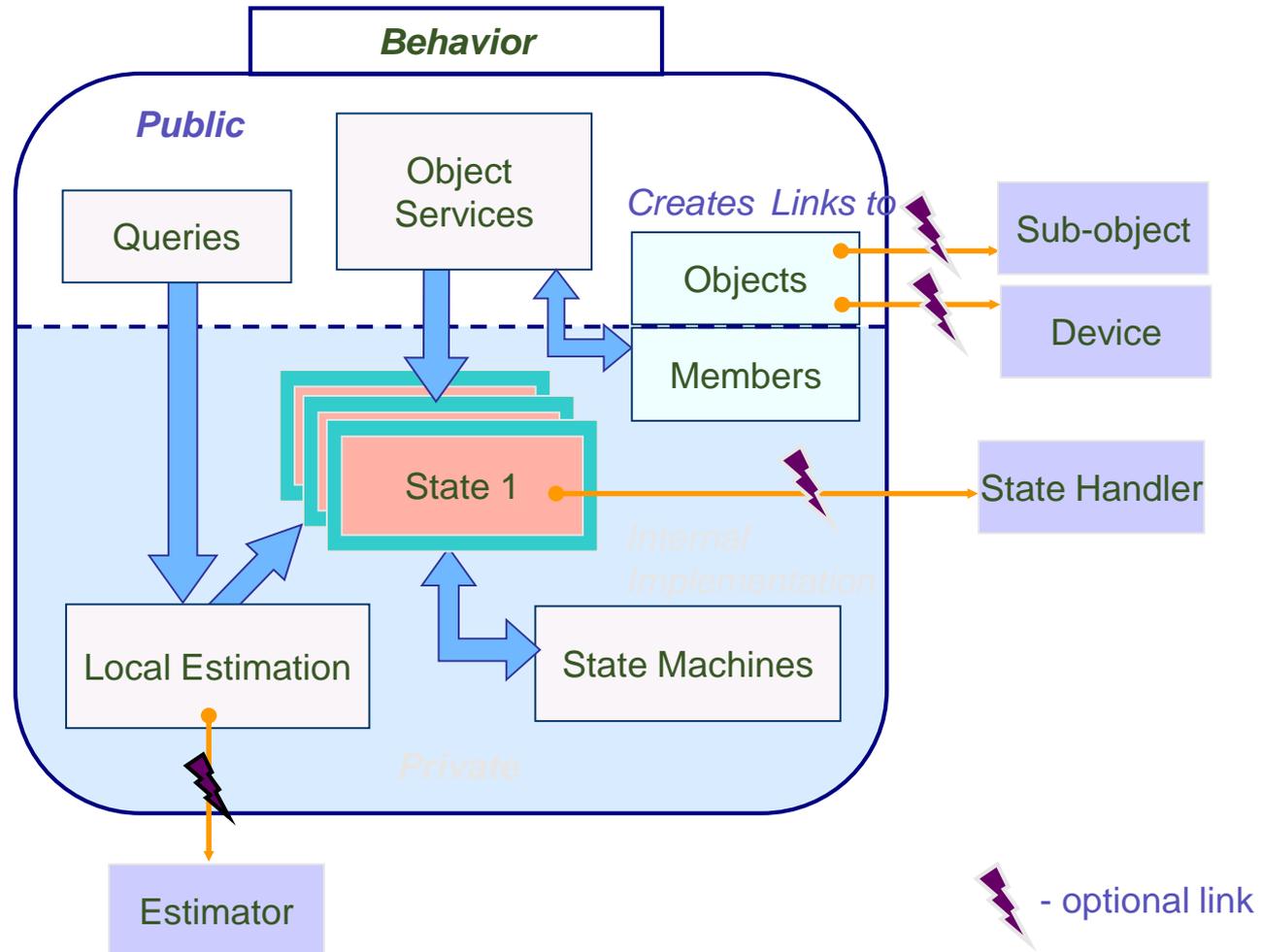
```
// C++ Sample Code

Motor  a_motor;

a_motor.use_trajectory_mode();
a_motor.move(Trapezoidal_Traj_Params( delta_position,
                                     max_velocity,
                                     accel,
                                     decel ));
```

**Important to consider implicit assumptions
of your API**

Theme 1: Use Abstraction Hierarchies



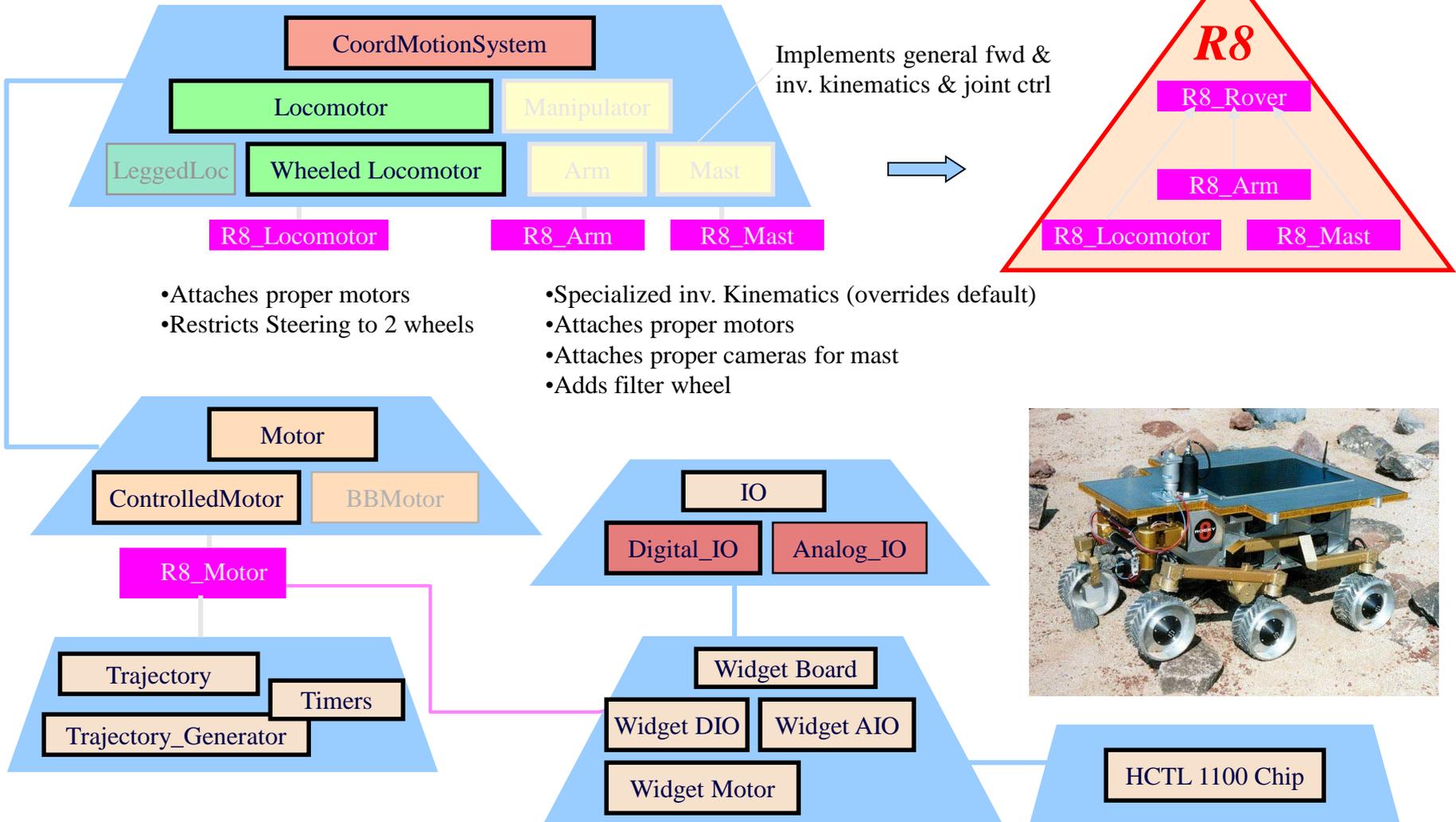
Example: Rocky 8 Rover



Non reusable Code



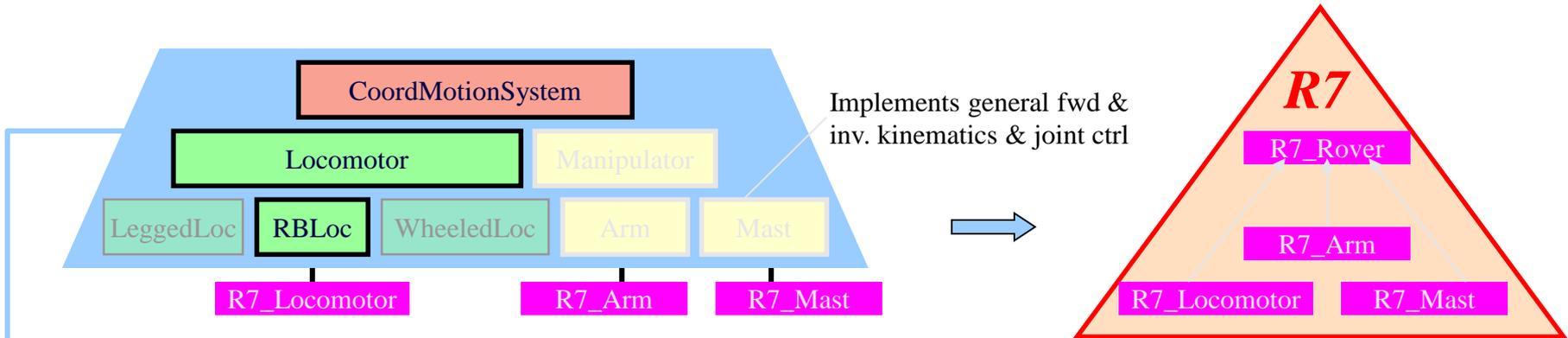
Reusable Code



Example: Rocky 7 Rover

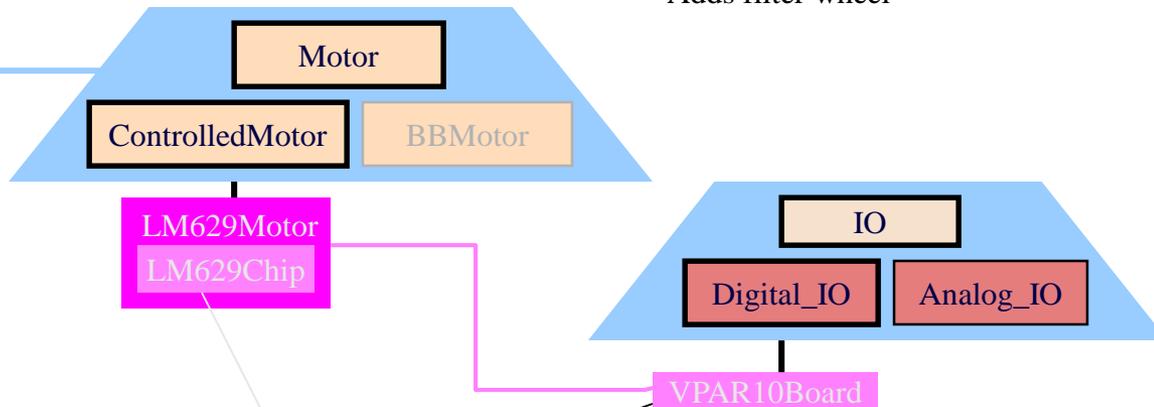


Non reusable Code
 Reusable Code



- Attaches proper motors
- Restricts Steering to 2 wheels

- Specialized inv. Kinematics (overrides default)
- Attaches proper motors
- Attaches proper cameras for mast
- Adds filter wheel



Device Drivers



Theme 1: Use an Abstraction Hierarchy



Advocacy

- **Manages complexity**
- **Enables interoperability**
- **Enables multi-level access**

Criticism

- **Imposes too much structure**
(cross-domain coupling can occur in flight)
- **Encapsulates data/state**
- **Creates strong coupling**
(compile time)



Declarative Programming

```
Rover.navigate_from_to(Loc1, Loc2)
```

```
Preconditions:    near(Loc1,Loc2)  
                  rover.has_power(Loc1,Loc2)  
                  rover.has_time(Loc1,Loc2)
```

```
Effects:        rover.is_at(Loc2)
```

Procedural Programming

```
If near(Loc1,Loc2) AND  
    rover.has_power(Loc1,Loc2) AND  
    rover.has_time(Loc1,Loc2) AND  
Then:  
    rover.navigate_from_to(Loc1,Loc2)
```

T. Estlin, D. Gaines, C. Chouinard, F. Fisher, R. Castano, M. Judd, R. Anderson, and , I. Nesnas, "[Enabling Autonomous Rover Science Through Dynamic Planning and Scheduling.](#)" *Proceedings of the 2005 IEEE Aerospace Conference, Big Sky, Montana, March 2005.* pdf (12 pages, 0.4MB)

Theme 2: Multiple Programming Paradigms



Advocacy

- Allows flexible ordering of activities
- Maximizes activities given resource constraints

Criticism

- Difficulty in predicting time to generate plan
- May not generate a plan
- Requires all constraints to be explicit
- Emergent behavior

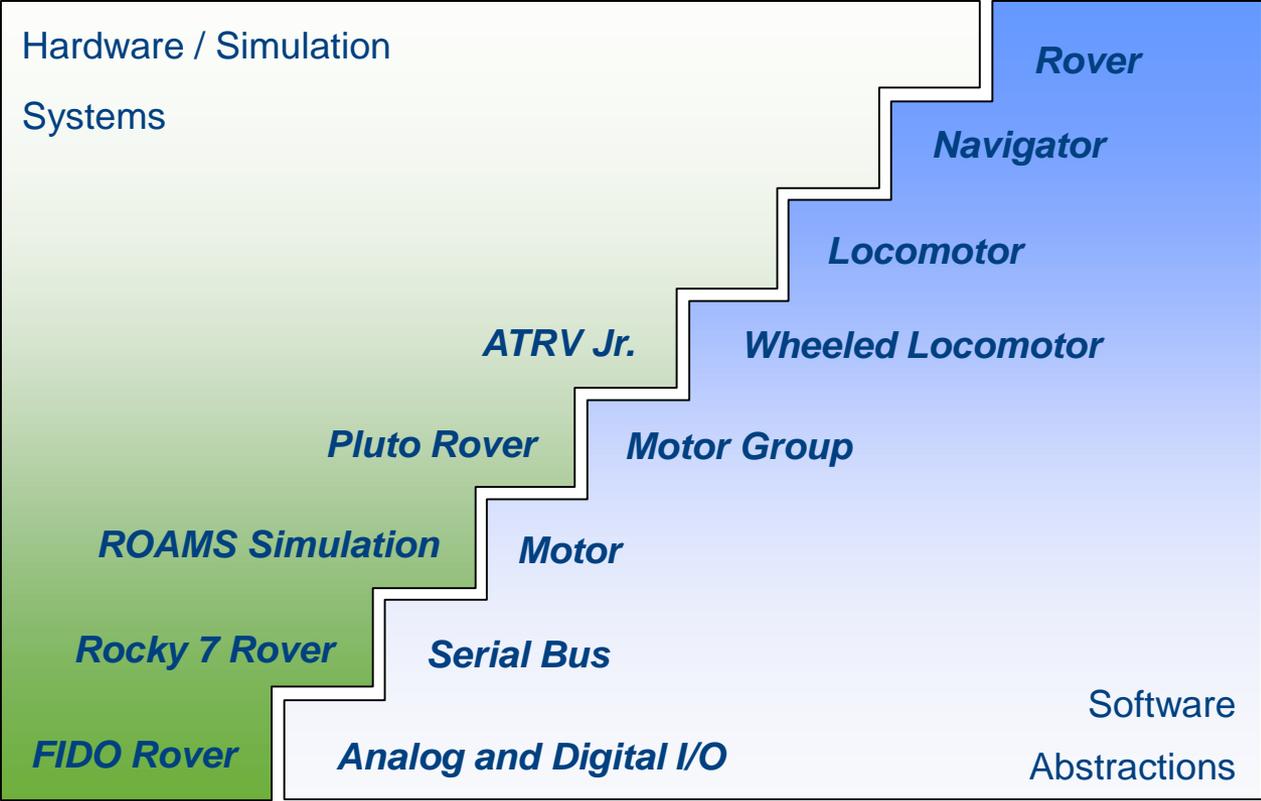
Declarative

Procedural

- Does not require explicit constraints (terse)
- More predictable

- Over-constrains order of activities

Theme 3: Provide Multi-level Access



Multi-level mobility abstractions



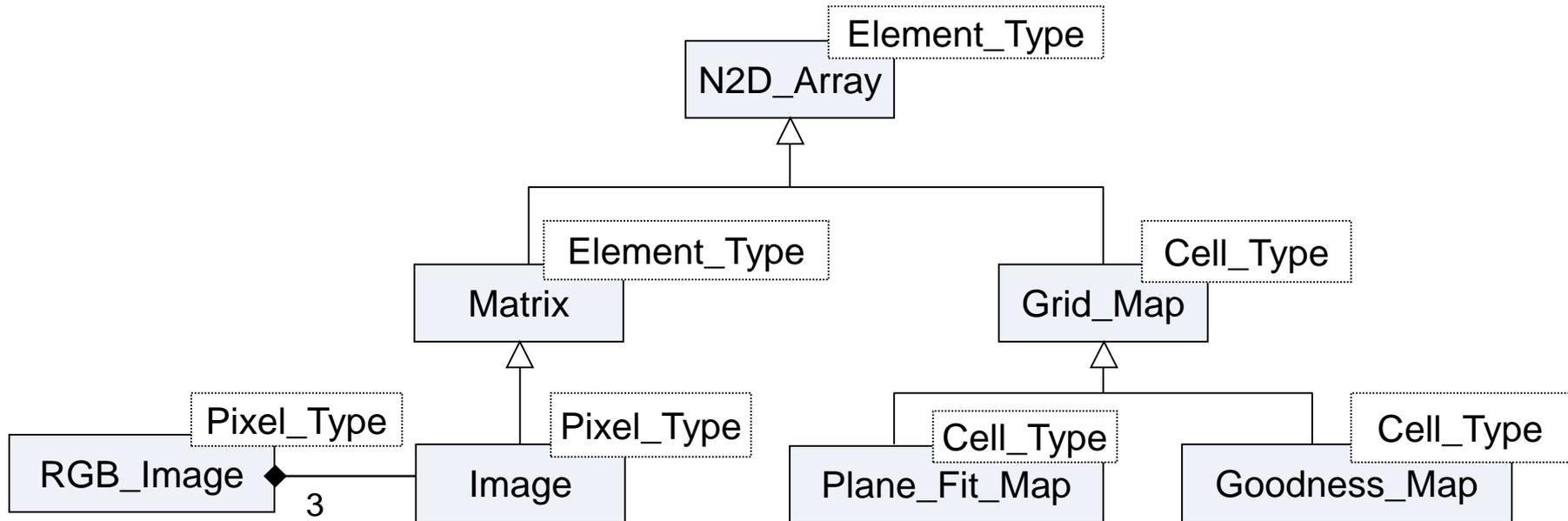
Advocacy

- Allows integration of new technologies at any level
- Allows migration of functionality between software and hardware

Criticism

- Requires arbitration among multiple masters

Theme 4: Use Common Data Structures



Theme 4: Use Common Data Structures



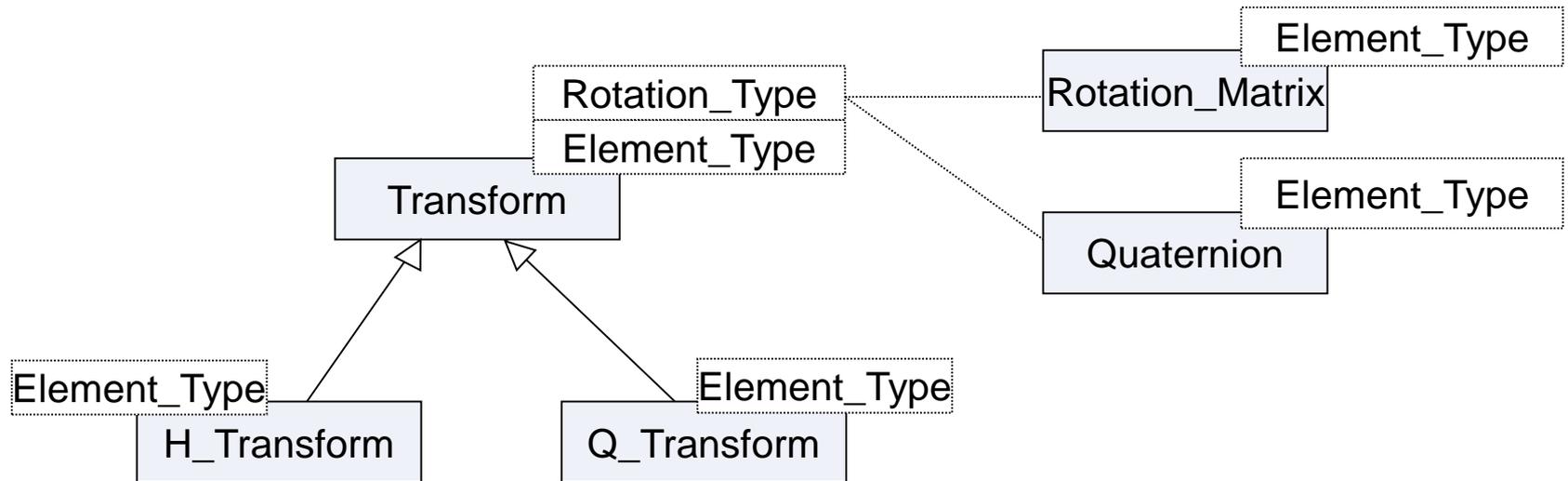
Advocacy

- Reduces unnecessary duplication (cost and maintenance)
- Allows deeper dives for debugging
- Reduces architectural mismatches

Criticism

- Creates dependencies on common data structures
- Modifications of data structures ripples through system

Theme 5: Use Interoperable Transformations





Advocacy

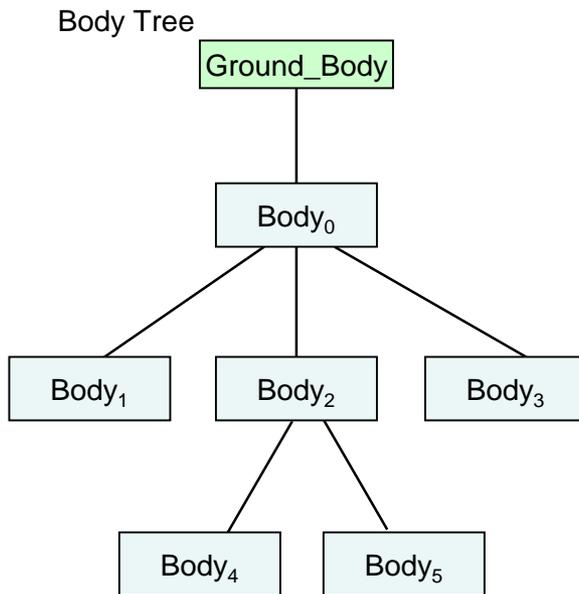
- Enables interoperability
- Reduces errors in coordinate transformation conversions
- Increases consistency and understandability

Criticism

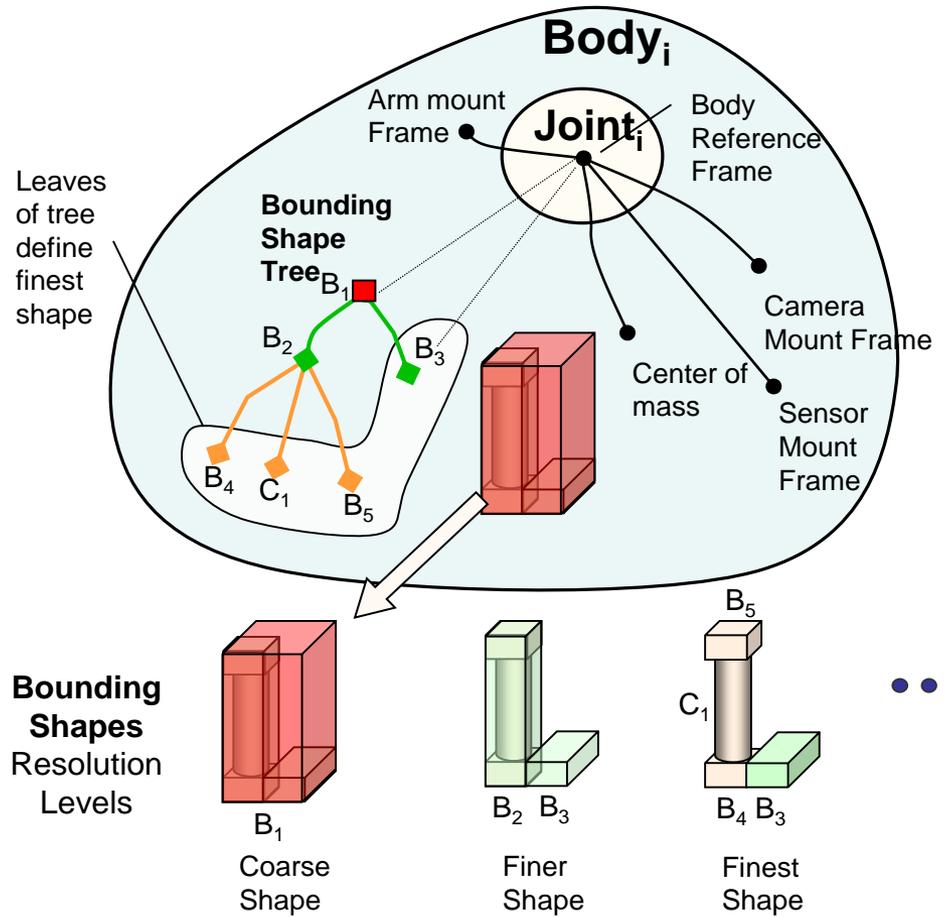
- Creates dependencies on common data structures
- Modifications of data structure ripple through system

Bodies and Joints

Unifying mechanism model



Mechanism Tree



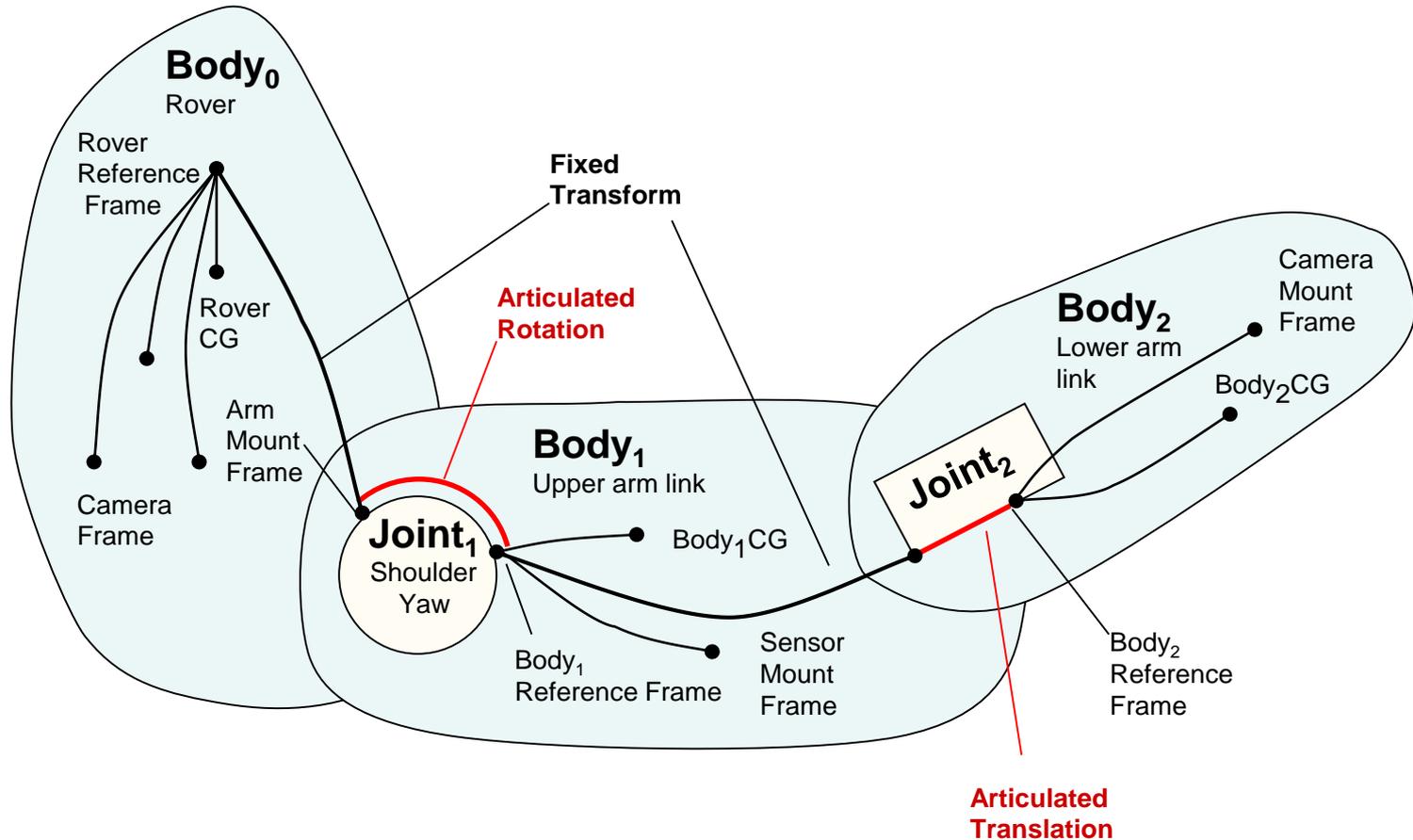
- A. [Diaz-Calderon, "Towards a Unified Representation of Mechanisms for Robotic Control Software,"](#)
- B. [International Journal of Advanced Robotic Systems, Vol. 3, No. 1, pp. 061-066, 2006.](#)

Theme 6: Unify Mechanism Model



Unifying mechanism model

Make stateless



- A. [Diaz-Calderon, "Towards a Unified Representation of Mechanisms for Robotic Control Software,"](#)
- B. [International Journal of Advanced Robotic Systems, Vol. 3, No. 1, pp. 061-066, 2006.](#)

Theme 6: Unify Mechanism Model



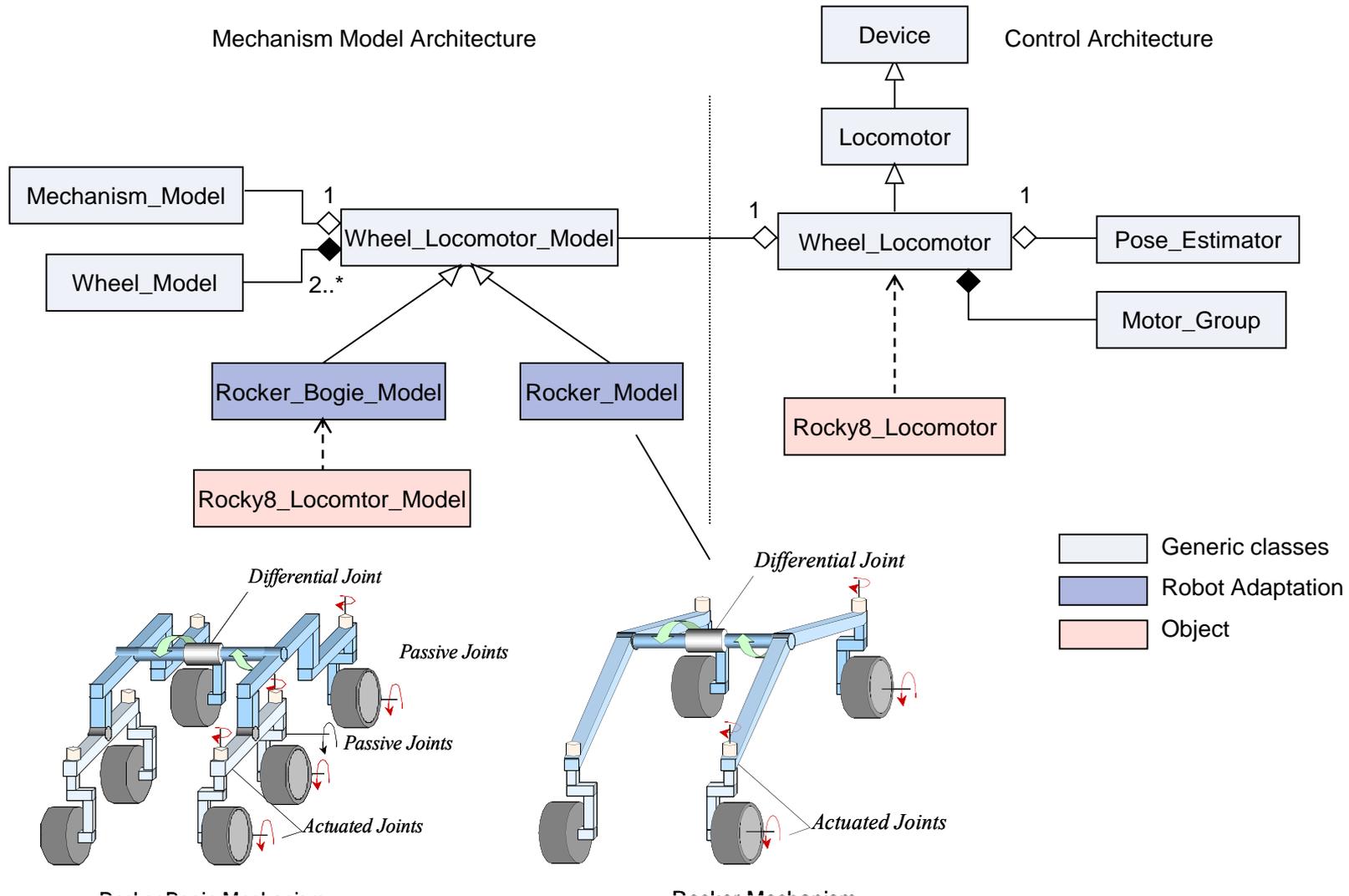
Advocacy

- Enables integrated motions increasing robotic workspace (e.g. mobile manipulation)
- Offers consistent representation
- Improves interoperability
- *Supports planning (what if?)*

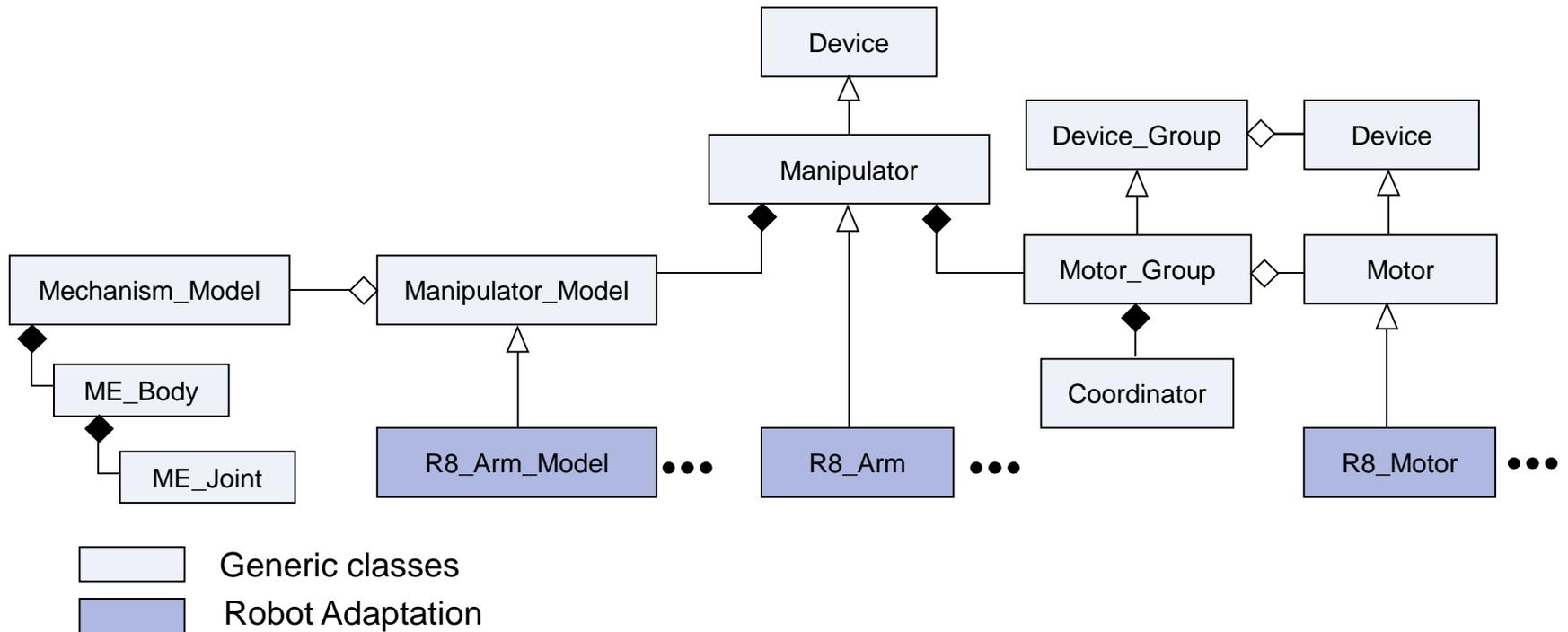
Criticisms

- Imposes structure
- Adds overhead

Theme 7(a): Separate Models from Control



Theme 7(a): Separate Models from Control



Theme 7(a): Separate Models from Control



Advocacy

- *Supports planning (what if?)*

Criticisms

Concluding thoughts



- Engaging domain experts is critical
- Starting with the end in mind helps steer the effort
- Developing robotics standards and reusable robotic software is hard because of the hardware/software heterogeneity
- Common infrastructure reduces accidental complexity and saves resources but adds constraints that could stifle innovation
- Generalized software increases complexity
- Interoperable software/hardware is challenging for autonomous robotics systems
- Developing and evolving Themes is critical
- Handling non-technical challenges (ITAR / IP) is important and requires significant effort

Summary



- Reviewed decades of robotics architectures
- Presented process and challenges
- Shared architectural themes
- Shared reflections on advocacy and criticisms

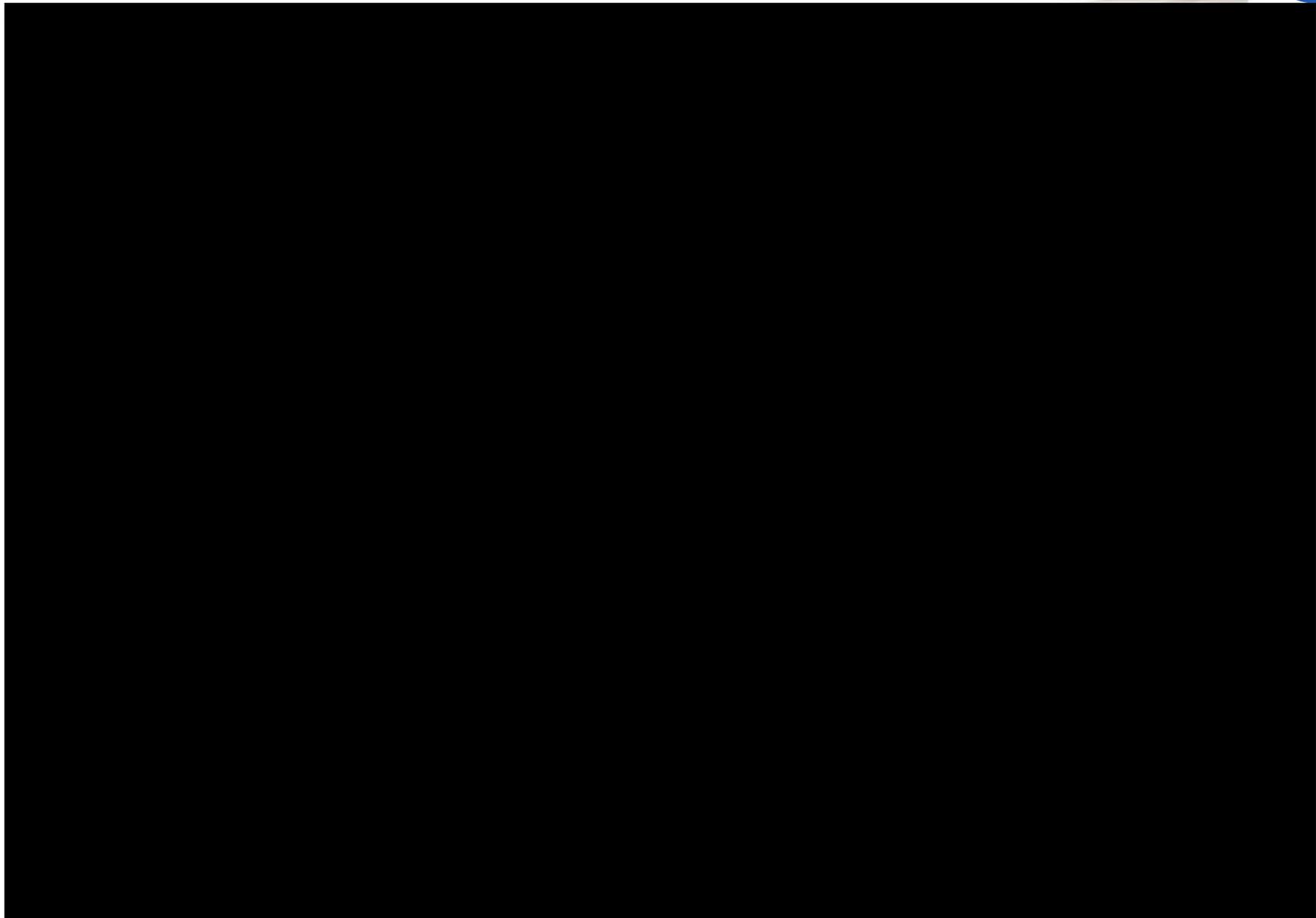
H.D. Nayar, I.A. Nenas, "[Measures and Procedures: Lessons Learned from the CLARAty Development at NASA/JPL](#)," *International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, October 2007

LECTURE 4

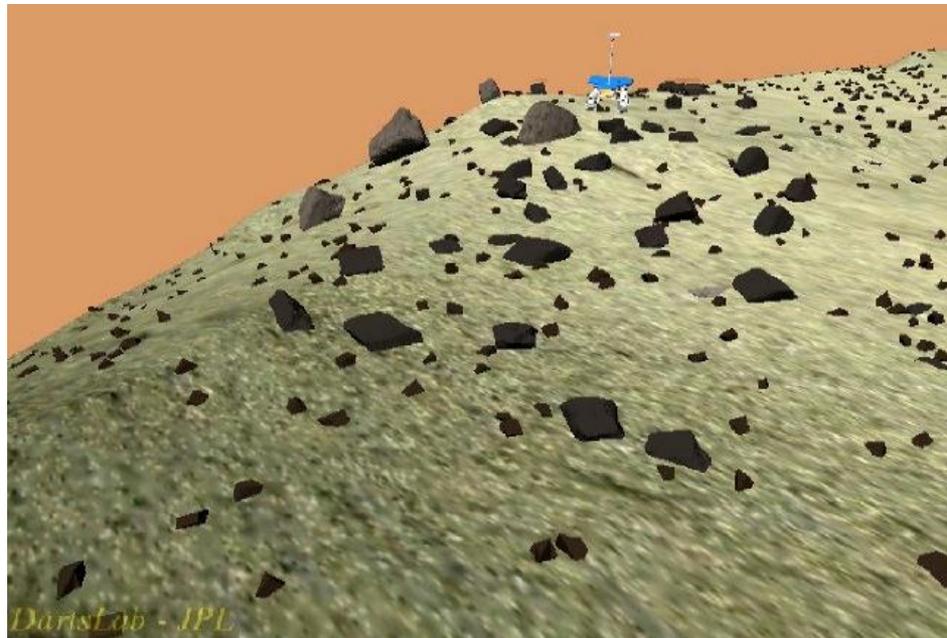
MORE ARCHITECTURAL ELEMENTS



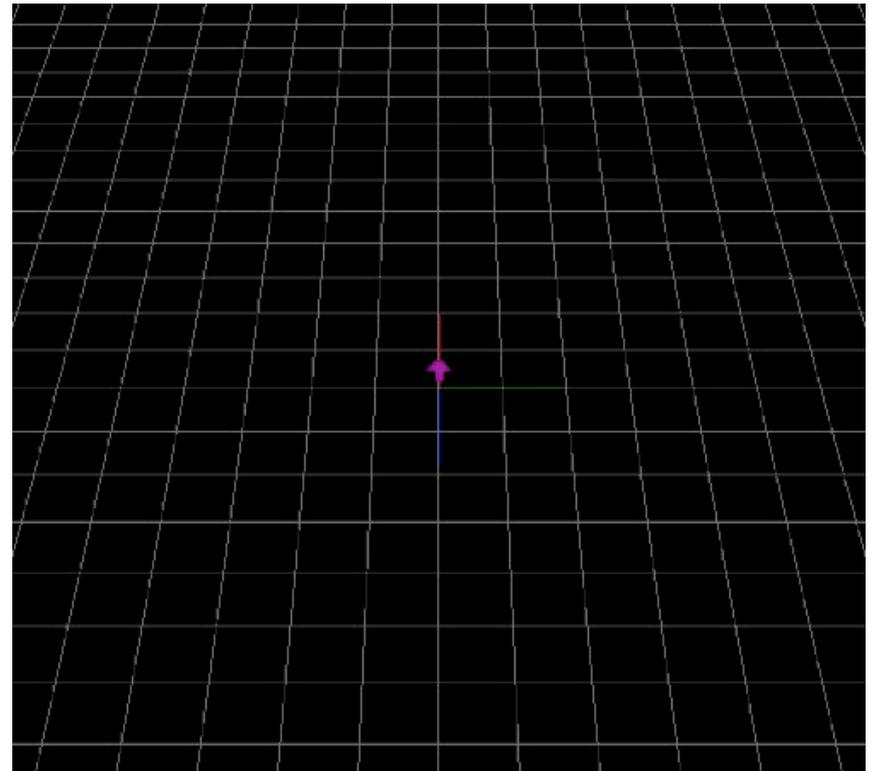
Example: Autonomous Approach and Measure



Navigating in a Simulated Environment



ROAMS



CLARAty Morphin
Navigator GUI

Courtesy of SOOPS task

Innovative Technologies Infused



- Over 50 technologies from two dozen technology providers integrated into the framework.
- Several technologies were formally and independently validated
- Several were infused into the MER and later MSL mission:
 - Autonomous navigation (Morphin/GESTALT) (CMU/JPL)
 - Visual target tracking of designated targets (ARC/JPL)
 - Long-range global navigation (Field D* - CMU)
 - Autonomous science observations (JPL)
 - Autonomous manipulation (JPL)
 - 6DOF Extended Kalman Filter pose estimation (U. Minnesota)

Architectural Themes

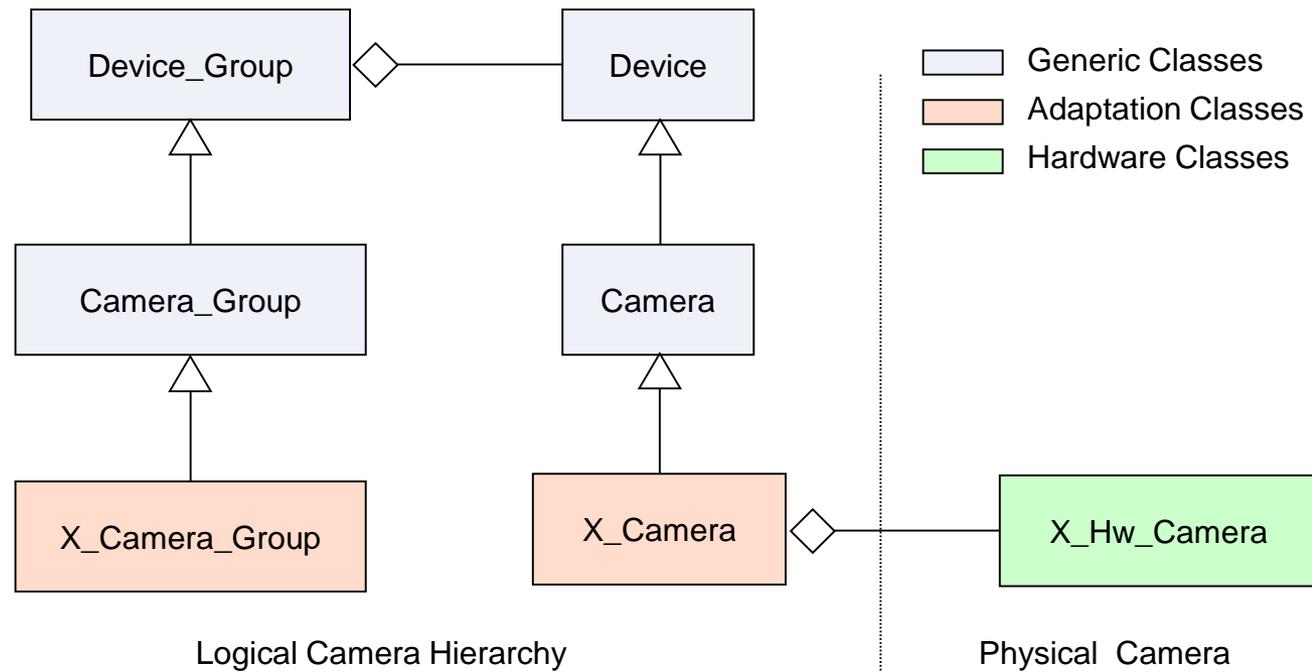


1. Abstraction hierarchy
2. Multiple programming paradigms
3. Multi-level access
4. Common data structures
5. Interoperable transformations
6. Unified mechanism model

*Some recurring and
some from CLARAty*

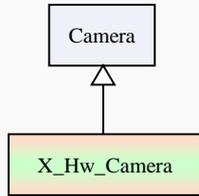
7. Separation of concerns
 - a) Models from control
 - b) Logical from physical hierarchies
 - c) Interface from implementation
 - d) Estimation from control
8. Run-time encapsulation

Theme 7(b): Separate Logical from Physical Hierarchies



D.S. Clouse, I.A. Nesnas, C. Kunz, ["A Reusable Camera Interface for Rovers,"](#) IEEE International Conference on Robotics and Automation, Workshop on Software Development and Integration in Robotics, Rome, Italy, April 2007.

Historical Evolution Leading to Separating Logical and Physical Hierarchies

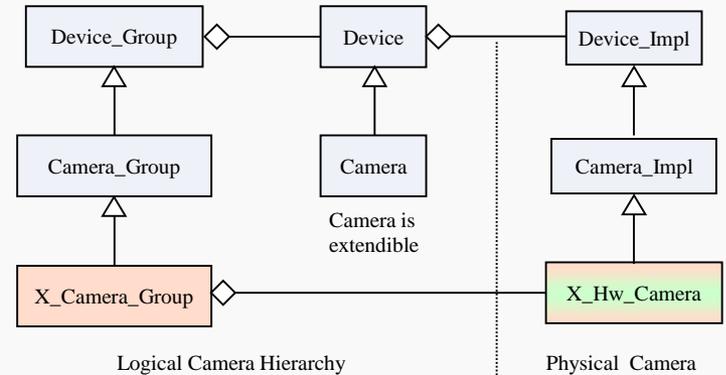


Revision I: (1999)

1. Simple hierarchy
2. Camera functionally not extendible
3. Hardware adaptations dependent on CLARy API
4. Only two camera synchronization. Done in the adaptation
5. Uses Image abstraction

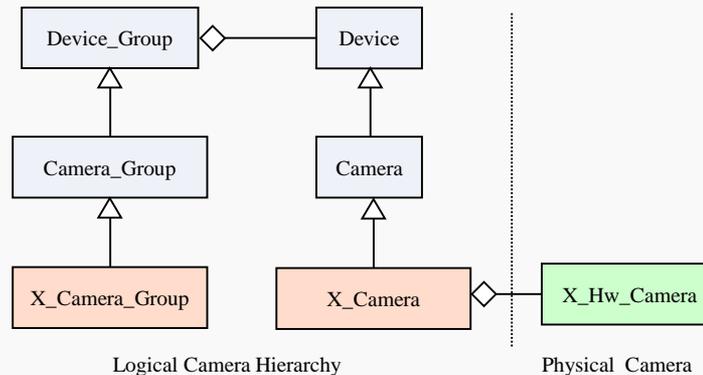
Revision II: (2002)

1. Complex hierarchy
2. Camera functionally extendible
3. Hardware adaptations dependent on CLARy API
4. Generic camera grouping
5. Multi-camera synchronization
6. Multi-client/thread support
7. Camera power management
8. Uses Image abstraction



Revision III: (2007)

1. Moderate complexity
2. Camera functionally not extendible
3. Separation of logical and physical hierarchies
4. Hardware adaptations independent of CLARy API
5. Generic camera grouping
6. Multi-camera synchronization
7. Multi-client/thread support
8. Camera power management
9. Supports capturing image properties by using Camera_Image abstraction



- Generic Classes
- Adaptation Classes
- Hardware Classes

Camera Use Cases from (SCIP):

- ✓ Monocular multi-resolution nav imaging for target tracking
- ✓ Synchronized binocular stereo from navcams for target ranging and fine pointing
- ✓ Synchronized quad stereo imaging from nav and hazcams for target hand-off
- ✓ Synchronized binocular stereo from hazcams for obstacle avoidance

I.A. Nenas, ["The CLARy Project: Coping with Hardware and Software](#)

[Heterogeneity,"](#) book chapter to appear in the *Software Engineering for Experimental Robotics*, Springer Tracts on Advanced Robotics, edited by Davide Brugali, 2006.

A Bad Solution for Task Safety



```
Image<uint8_t>    img1;
X_Camera          cam1(hw_cam1);

cam1.lock();
cam1.set_brightness(0.35);
cam1.acquire(img1);
cam1.unlock();
```

Problems

- User must reestablish parameter settings for every lock.
- Assumes user's will write cooperative code.
- Deadlock / starvation are possible.

D.S. Clouse, I.A. Nesnas, C. Kunz, ["A Reusable Camera Interface for Rovers,"](#) IEEE International Conference on Robotics and Automation, Workshop on Software Development and Integration in Robotics, Rome, Italy, April 2007.



Logical/Physical Cameras for Task Safety

Physical camera object represents a piece of hardware.

- Interface is specific to the camera hardware
- Parameter change happens immediately
- Acquire uses current hardware parameter settings

Logical camera object maintains a single user's view.

- Base class defines a common interface for all logical cameras.
- Interface may be extended to support hardware-specific functions.
- Parameter values are cached
- Acquire atomically sets params in hardware and acquires image.

No special user code is required for task safety.

```
Image<uint8_t>  img1, img2;  
X_Hw_Camera    hw_cam1(id_unique_to_hw);  
X_Camera       cam1(hw_cam1);  
  
cam1.set_brightness(0.35);  
cam1.acquire(img1);  
cam1.acquire(img2);
```

Definition of Logical Camera Class



```
class Camera : public Device {
public:
    virtual bool        set_contrast(double gain_percent) = 0;
    virtual double      get_contrast() const = 0;
    virtual bool        set_brightness(double offset_percent) = 0;
    virtual double      get_brightness() const = 0;
    virtual bool        set_exposure(double seconds) = 0;
    virtual double      get_exposure() const = 0;

    enum IMAGE_FORMAT { MONO8, YUV411, ..., RGB8, MONO16, RGB16 };
    virtual bool        set_format(IMAGE_FORMAT format, int width, int height);
    virtual IMAGE_FORMAT get_format() const = 0;
    virtual int         get_width() const = 0;
    virtual int         get_height() const = 0;

    virtual void        acquire(Image<uint8_t> & image,
                               Time* timestamp = NULL,
                               Feature_Map* feat_map = NULL) = 0;
    virtual void        acquire(Image<uint16_t> & image,
                               Time* timestamp = NULL,
                               Feature_Map* feat_map = NULL) = 0;
};
```

Camera_Group Code Example



```
X_Hw_Camera hw_cam1(id1), hw_cam2(id2);
X_Camera cam1(hw_cam1), cam2(hw_cam2);
X_Camera_Group grp(cam1, cam2);

cam1.set_brightness(0.35);
cam2.set_brightness(0.35);
Vector<Image<uint8_t> > images(2);
grp.acquire(images);
```

D.S. Clouse, I.A. Nesnas, C. Kunz, ["A Reusable Camera Interface for Rovers,"](#) IEEE International Conference on Robotics and Automation, Workshop on Software Development and Integration in Robotics, Rome, Italy, April 2007.



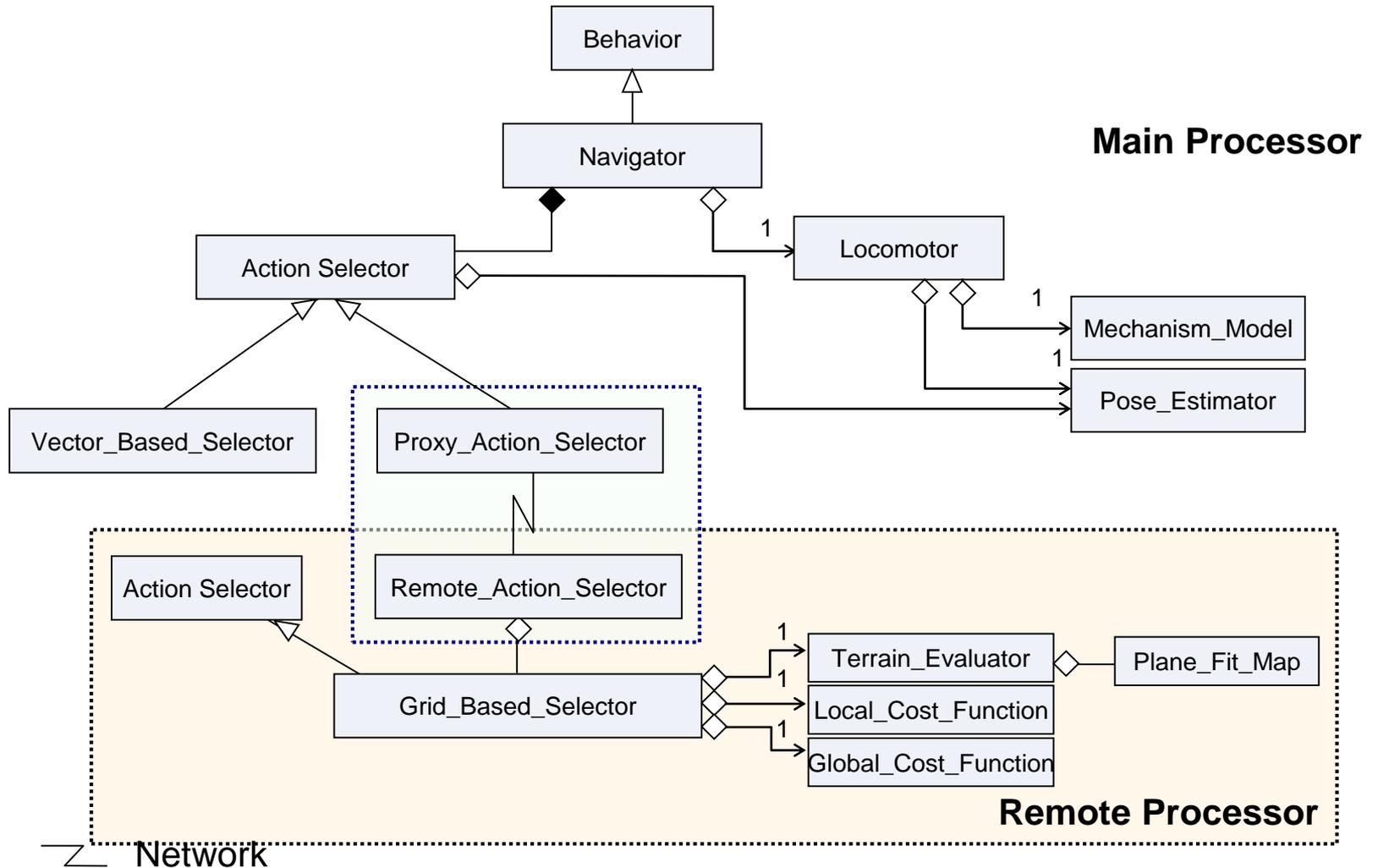
Advocacy

- Decouple hardware architecture from functional/logical constraints
- Provide clear mapping between “*what the software needs*” and “*what the hardware can do*”

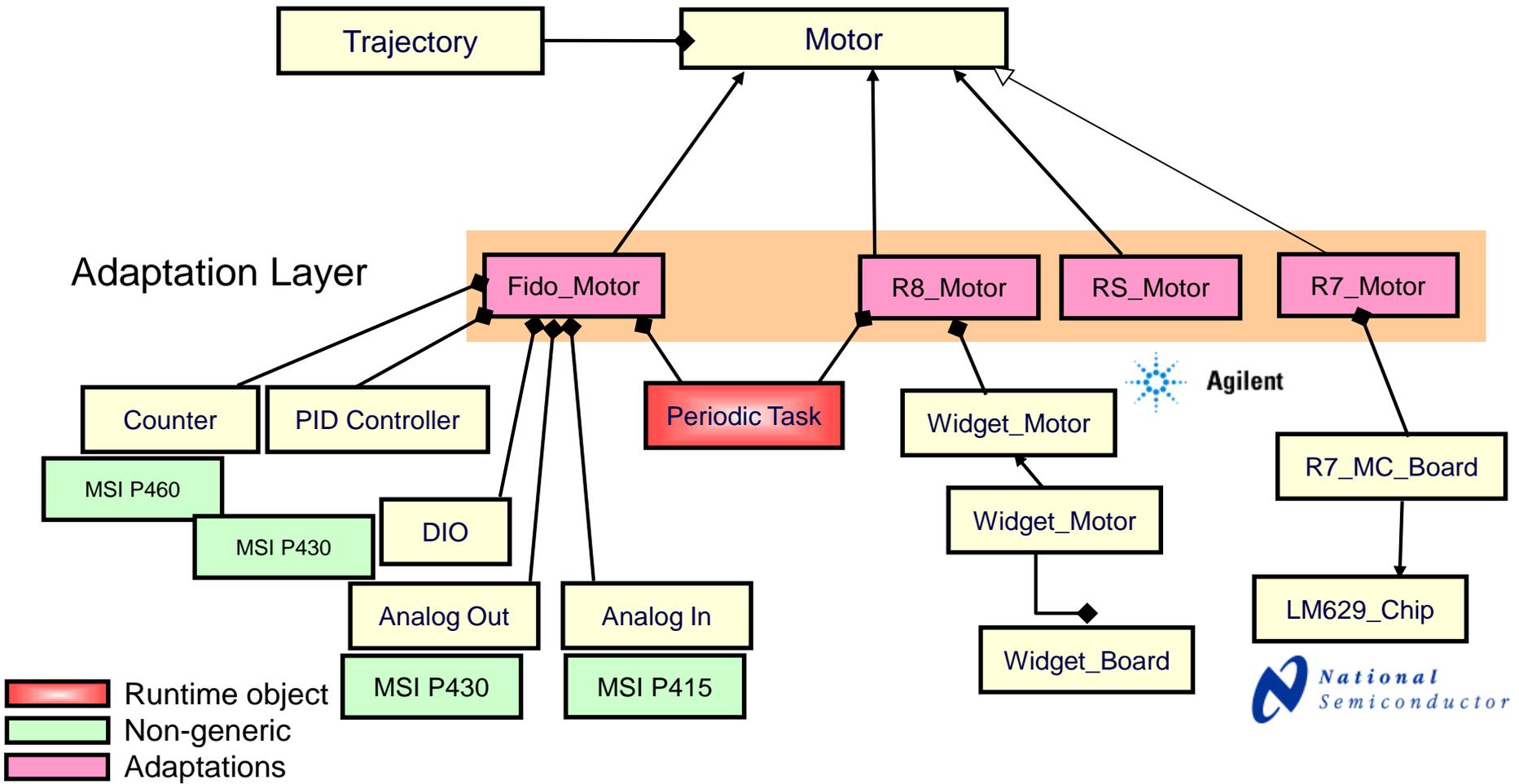
Criticisms

- Increased complexity for interfacing to hardware

Theme 7(c): Separating Interface from Implementation for Remote Processing



Theme 8: Encapsulate Non-Generic Run-time Models





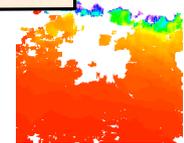
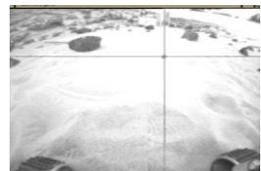
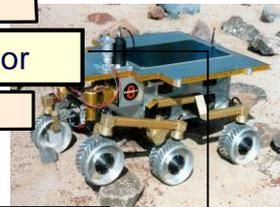
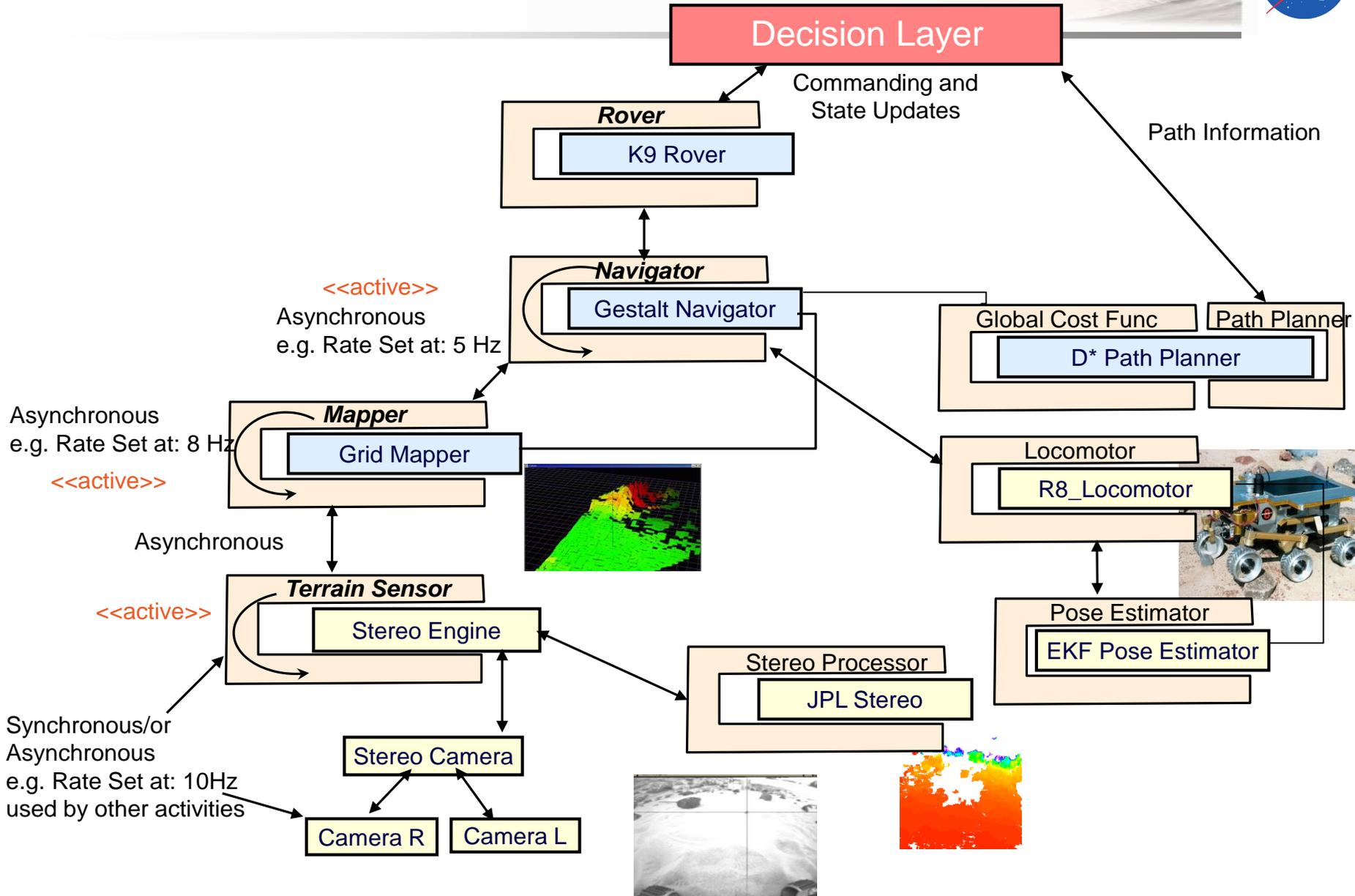
Advocacy

- **Manages complexity**
- **Enables interoperability**
- **Enables multi-level access**

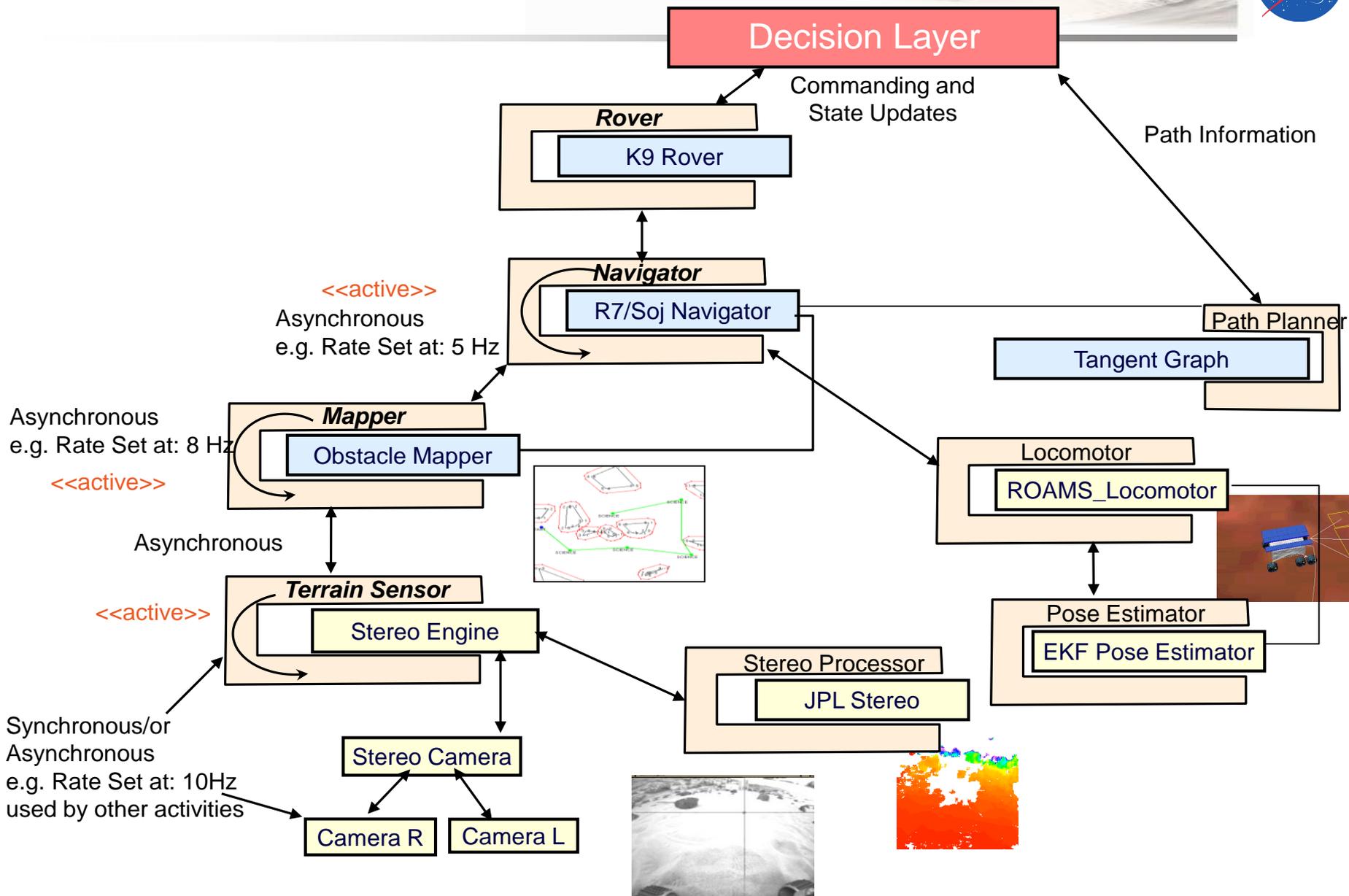
Criticisms

- **Hides run-time elements**
- **Run-time element could impact performance**

Putting it All Together - Swapping Navigation Algorithms



Putting it All Together - Swapping Navigation Algorithms



ControlShell FSM



- 1992–1997

- Data Flow Elements (DFE)

- Inputs and outputs: integers and float matrices
 - Data flows by copy
 - Run-time configurable

- Finite State Machines (FSMs)

- State transitions

- Component scheduler

- Network Data Distribution System (NDDS)

- 1997-1999

- Hierarchical Components (Cog)

- Contains DFEs and FSMs
 - Uses connectors (primitives and user defined interfaces)

- NDDS

Credit: NASA The RAMS (Robot-Assisted Micro-Surgery) Arms
<https://www-robotics.jpl.nasa.gov/systems/system.cfm?System=9>



Advocacy

- Flexibility
- Run-time re-configurability
- Structured environment
- Looser component dependency

- *Graphical representation*
- *Auto-generated FSM code*

Criticism

- Concrete interfaces
(*no abstraction*)
- Performance
(*multiple data copies*)
- Run-time type checking
- Scalability
- *Difficult to manage graphical representation*
- *Multiple ways to edit (graphics, code, auto-code)*
- *Debugging*
- Inter-operability
- Availability (cost)

Lessons Learned



- **Success:** many many attempts, little traction ...
- **Timing:** is critical: has the field matured enough and ready for standardization?
 - Too soon risks stifling innovation
 - Too late results in accidental complexity and challenges of inter-operability
- **Scope:**
 - Efforts with a **narrow** technical **scope** but large application potential seem to succeed (IETF, OMG DDS)
 - Efforts with **broad scope** and large **heterogeneity** face challenges (e.g. JAUS)
- **Hardware/software:** efforts need to engage both hardware and software providers and supplies
- **Outlook:** autonomous systems is a rapidly growing field with the possibility for several disruptive innovations
- **Process:** need an evolvable process that enables fielding new innovations and evaluating success prior to incorporation into the standard



Logical/Physical Cameras for Task Safety

Physical camera object represents a piece of hardware.

- Interface is specific to the camera hardware
- Parameter change happens immediately
- Acquire uses current hardware parameter settings

Logical camera object maintains a single user's view.

- Base class defines a common interface for all logical cameras.
- Interface may be extended to support hardware-specific functions.
- Parameter values are cached
- Acquire atomically sets params in hardware and acquires image.

No special user code is required for task safety.

```
Image<uint8_t>  img1, img2;  
X_Hw_Camera    hw_cam1(id_unique_to_hw);  
X_Camera       cam1(hw_cam1);  
  
cam1.set_brightness(0.35);  
cam1.acquire(img1);  
cam1.acquire(img2);
```

Definition of Logical Camera Class



```
class Camera : public Device {
public:
    virtual bool      set_contrast      (double gain_percent)      = 0;
    virtual bool      set_brightness    (double offset_percent)    = 0;
    virtual bool      set_exposure      (double seconds)           = 0;

    virtual double    get_contrast      () const = 0;
    virtual double    get_brightness    () const = 0;
    virtual double    get_exposure      () const = 0;

    enum IMG_FORMAT { MONO8, YUV411, ..., RGB8, MONO16, RGB16 };

    virtual bool      set_format(IMAGE_FORMAT format, int width, int height);
    virtual IMG_FORMAT get_format() const = 0;
    virtual int       get_width()  const = 0;
    virtual int       get_height() const = 0;

    virtual void      acquire(Image<uint8_t> & image,
                               Time* timestamp = NULL,
                               Feature_Map* feat_map = NULL) = 0;
    virtual void      acquire(Image<uint16_t> & image,
                               Time* timestamp = NULL,
                               Feature_Map* feat_map = NULL) = 0;

};
```

Camera_Group Code Example



```
X_Hw_Camera hw_cam1(id1), hw_cam2(id2);  
X_Camera cam1(hw_cam1), cam2(hw_cam2);  
X_Camera_Group grp(cam1, cam2);  
  
cam1.set_brightness(0.35);  
cam2.set_brightness(0.35);  
Vector<Image<uint8_t> > images(2);  
grp.acquire(images);
```

Component Architecture (ControlShell)



Advocacy

- Flexibility
- Run-time re-configurability
- Structured environment
- Looser component dependency
- More amenable to system health management

- *Graphical representation*
- *Auto-generated FSM code*
- *Monitoring state transitions during execution*

Criticism

- Concrete interfaces (*no abstraction*)
- Inter-operability
- Performance (*multiple data copies*)
- Debugging (distributed)

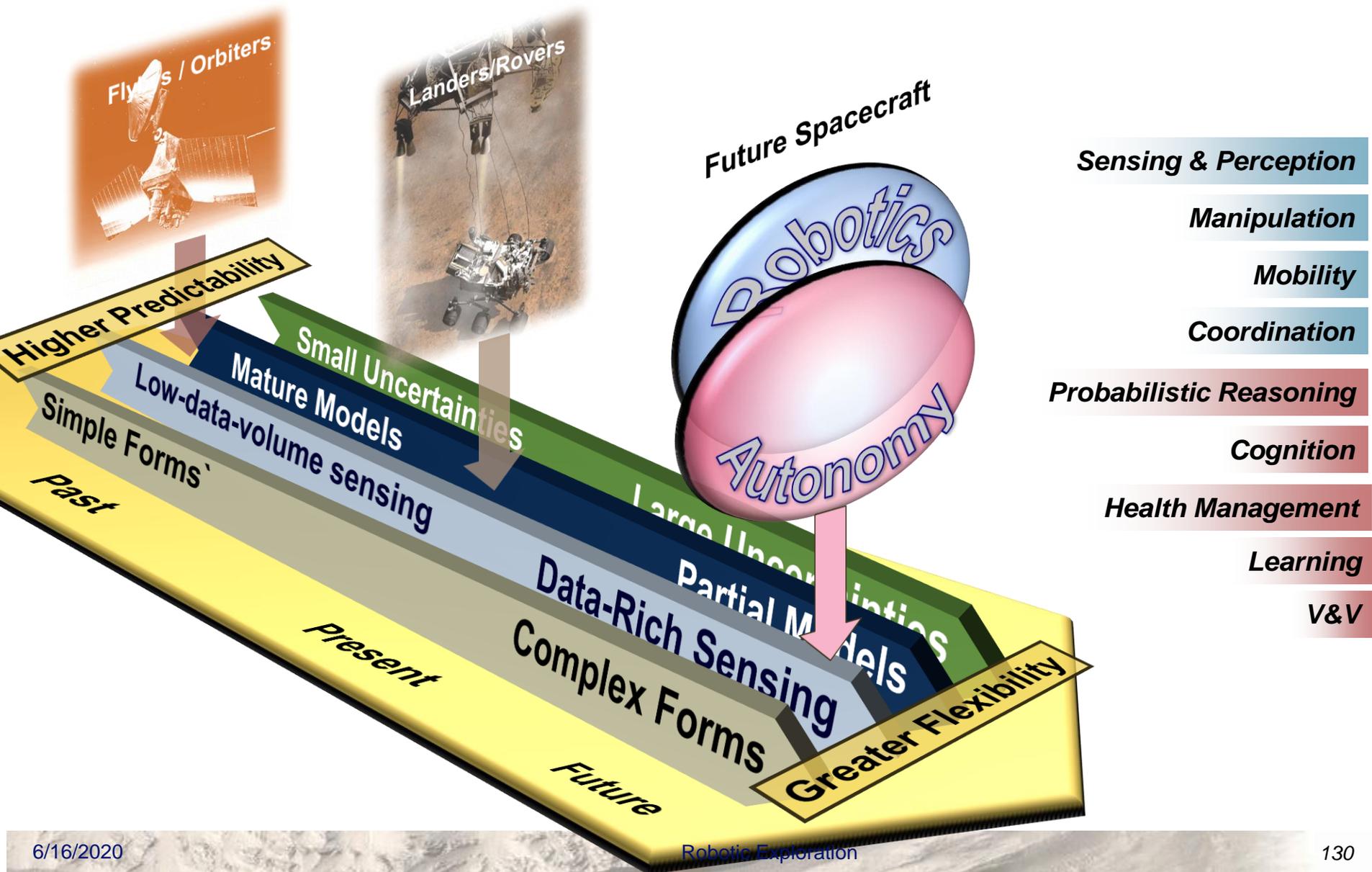
- *Difficult to manage graphical representation*
- *Multiple ways to edit (graphics, code, auto-code)*
- *Availability (cost)*

COMMANDING THE ROBOT

SEQUENCES VS. TASK NETWORKS



Autonomy for Future Exploration



Command Sequencing



- **A basic operations paradigm**
 - Appeared in early days of space exploration
 - Requires little processing power or memory
 - Fundamentally open loop control
 - Works well for predictable scenarios
 - Responds to faults by going into safe-mode
- **New paradigm**
 - Maximize onboard resource usage
 - Onboard response to faults

Adapted from: Slides by D. Dvorak

D Dvorak, R Rasmussen, G Reeves, A Sacks, [“Software architecture themes in JPL's Mission Data System”](#) - Aerospace Conference ..., 2000 - ieeexplore.ieee.org

Sequences vs. Task Networks



Old Paradigm

Command Sequencer

- Commands issued at absolute and relative times
- Works well for predictable, fault-free scenarios
- When a fault occurs, does not know what activities are affected
- Requires Earth-in-the-loop for recovery

New Paradigm

Task Sequencer

- Task = command + success criterion
- Network captures task dependencies
- Task network holds recovery options for task failures
- Unaffected tasks continue running

Sequences vs. Task Networks



Command sequencer

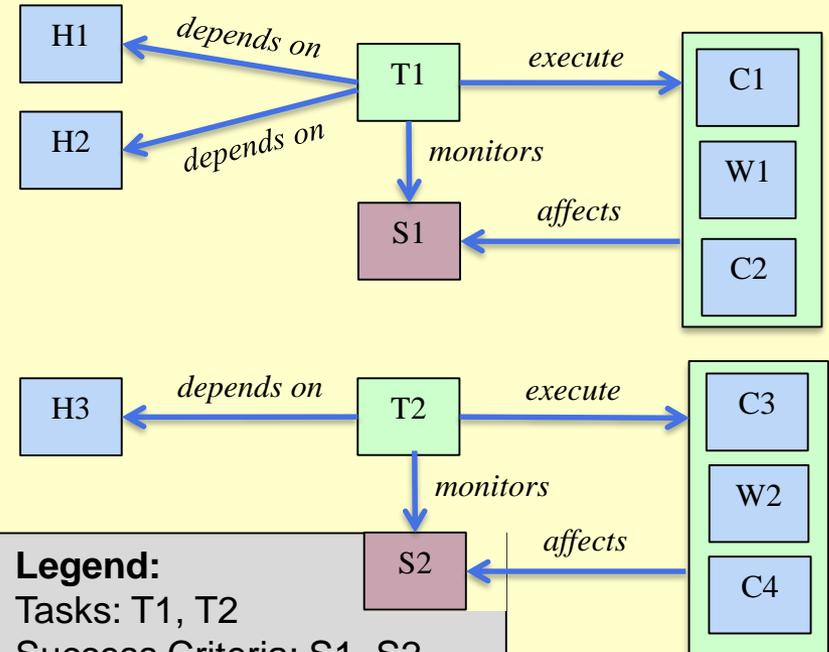
Start of sequence

- Issue command C1
- Wait W1 seconds
- Issue command C2
- Issue command C3
- Wait W2 seconds
- Issue command C4

End of sequence

- Success/failure unknown until telemetry analyzed on Ground
- Fault protection often must be safe because dependencies not known onboard

Task Sequencer



- Task success/failure monitored onboard
- Fault recovery more localized because dependencies known onboard
- Unaffected tasks continue to run

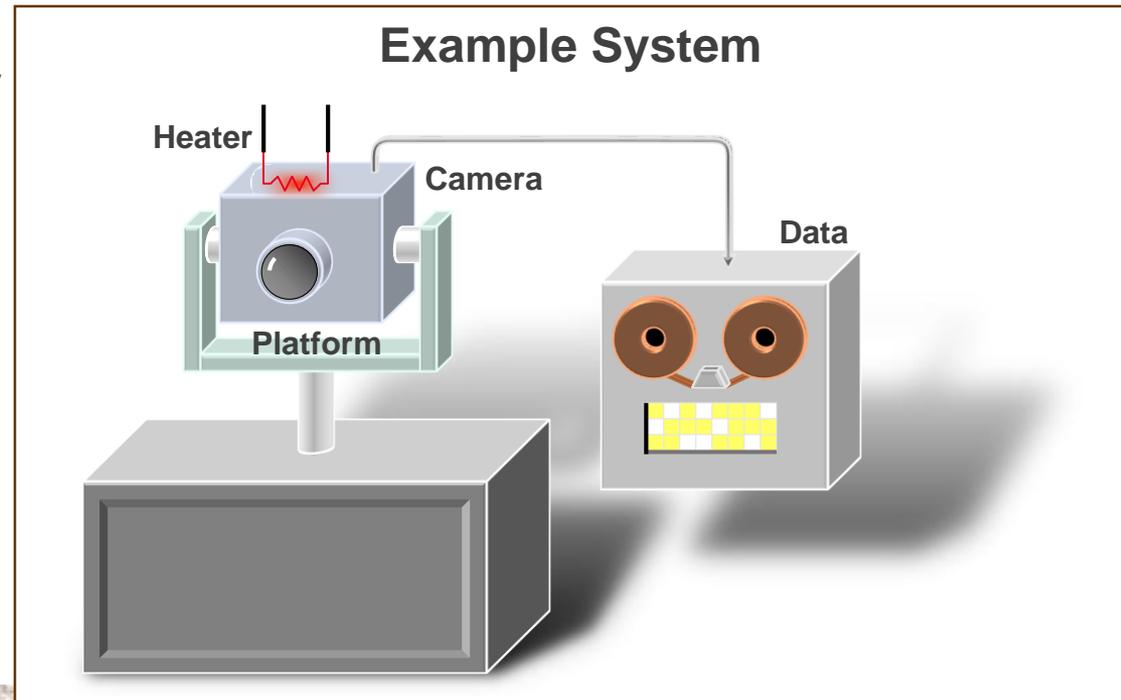
Adapted from: Slides by D. Dvorak

Example: Camera on a Scan Platform



- The camera rotates on the gimbaled platform to point at a target
 - Picture data from the camera is stored separately
 - When the camera is OFF, a heater can keep it warm
- Since control is about change, we need a way to talk about change
 - This is accomplished with the notion of

State



Exposing the Sequence Structure

Taking a Picture

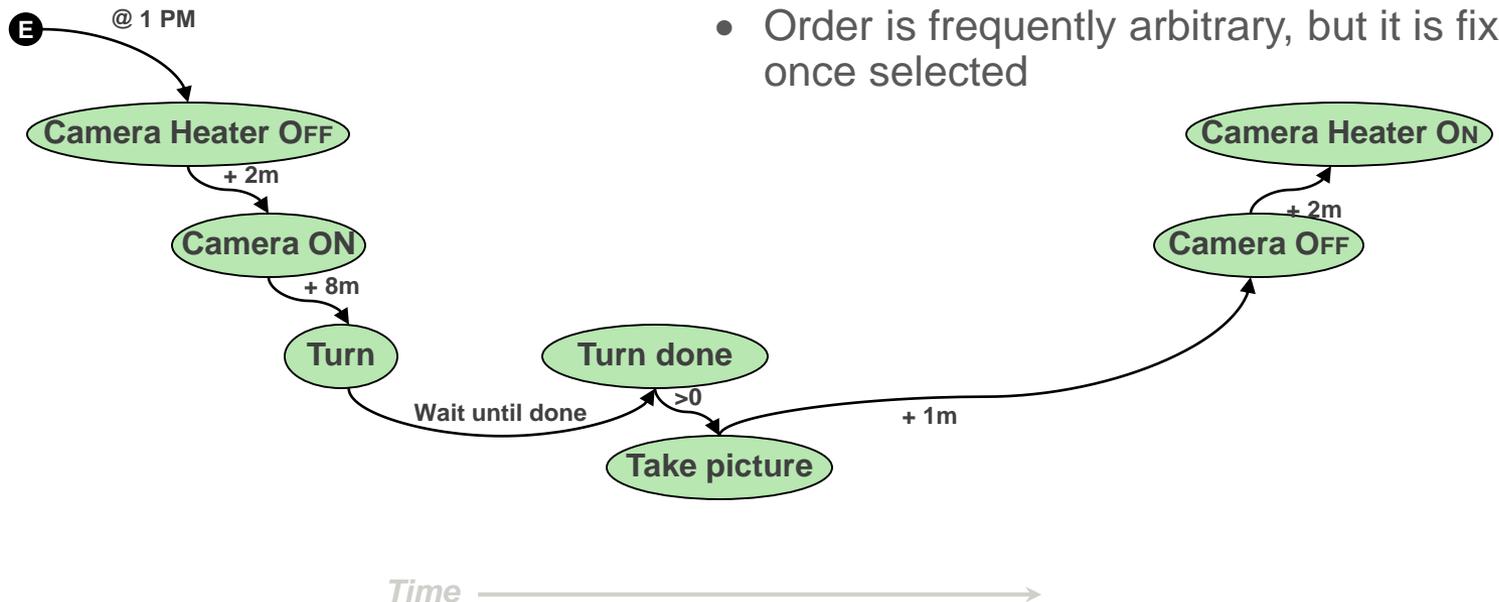
Operator's View:

1:00 PM	Camera Heater OFF
+ 2m	Camera ON
+ 8m	Turn platform to target
Turn done	Take picture
+ 1m	Camera OFF
+ 2m	Camera Heater ON

- Sequences are just that...

sequential!

- Commands are strung along a single **command time line**
 - Absolute and relative times, plus an occasional event
 - All relationships among commands must be captured only in these consecutive time relationships
 - Order is frequently arbitrary, but it is fixed, once selected





Checking the Sequence

Taking a Picture

Operator's View:

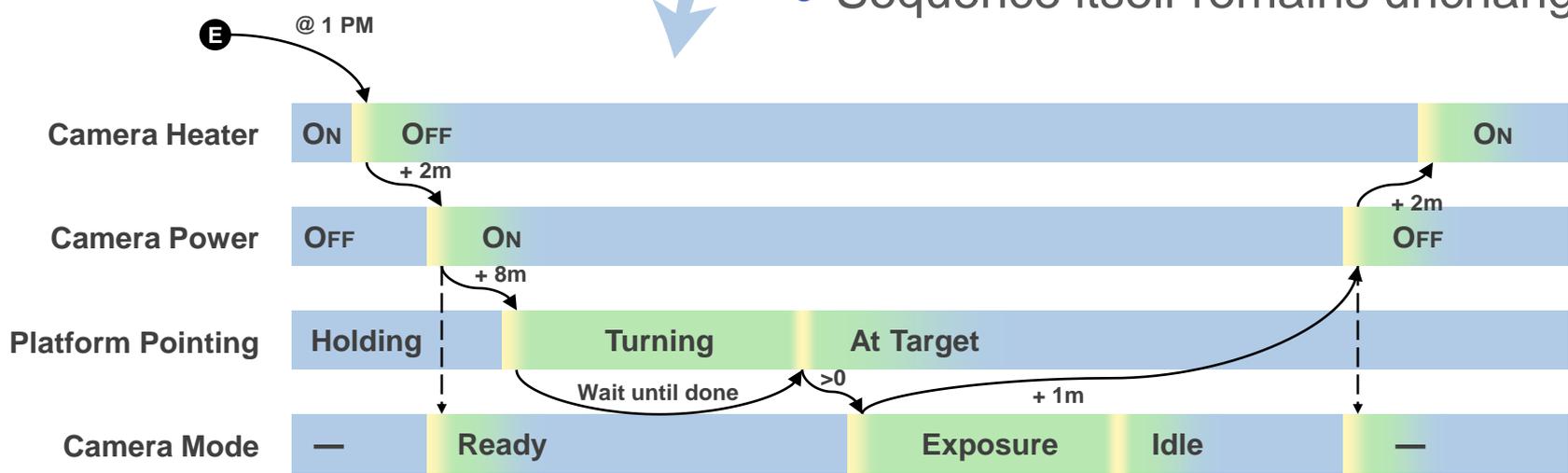
1:00 PM Camera Heater OFF
 + 2m Camera ON
 + 8m Turn platform to target

 Turn done Take picture

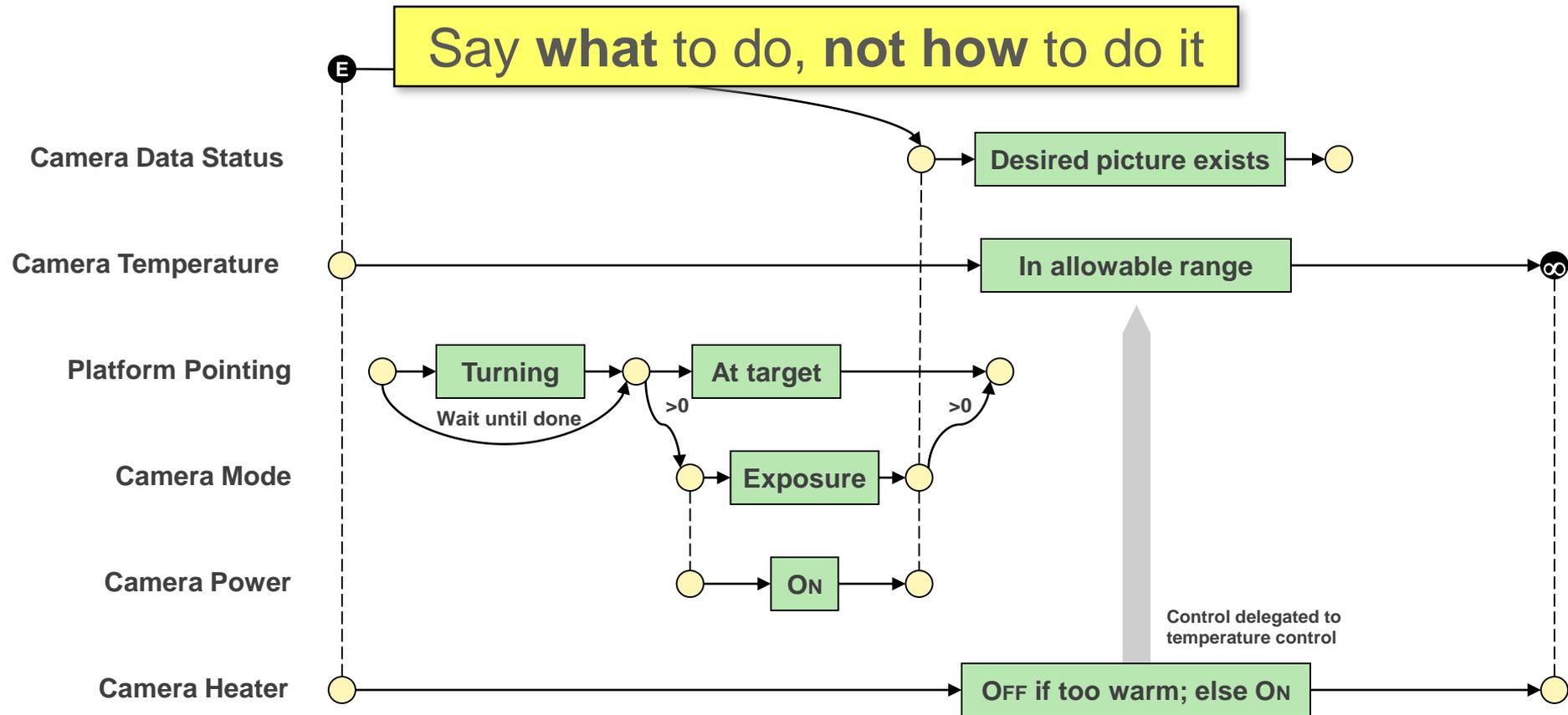
 + 1m Camera OFF
 + 2m Camera Heater ON

- Manage boundary conditions
 - Assume start states
 - Impose end states
- Use models to predict times, state gaps, and side effects
- Check rules, such as...
 - Enough time for exposure
 - Stationary during entire exposure
- Sequence itself remains unchanged

State Representation:



Elaboration of the Example



- The resulting constraint network is still complete, but more flexible than the sequence-based version
- Parent–child relationships among elements can be included in the network, providing an even more information to aid manipulation

SYSTEM HEALTH MANAGEMENT



Model-Based System Health Management



- Provides an estimate of system behavior with respect to health through the use of system models
- Models typically feed into Diagnostic Engines
 - Estimated behavior from models is compared to sensor/command data
- Some systems require explicit fault models (e.g. Livingstone, HyDE)
- Others only require models of *nominal* system behavior for fault diagnosis (e.g. MONSID)

K Kolcio, L Fesq, R Mackey, [“Model-based approach to rover health assessment for increased productivity.”](#)
- Aerospace Conference, 2017.

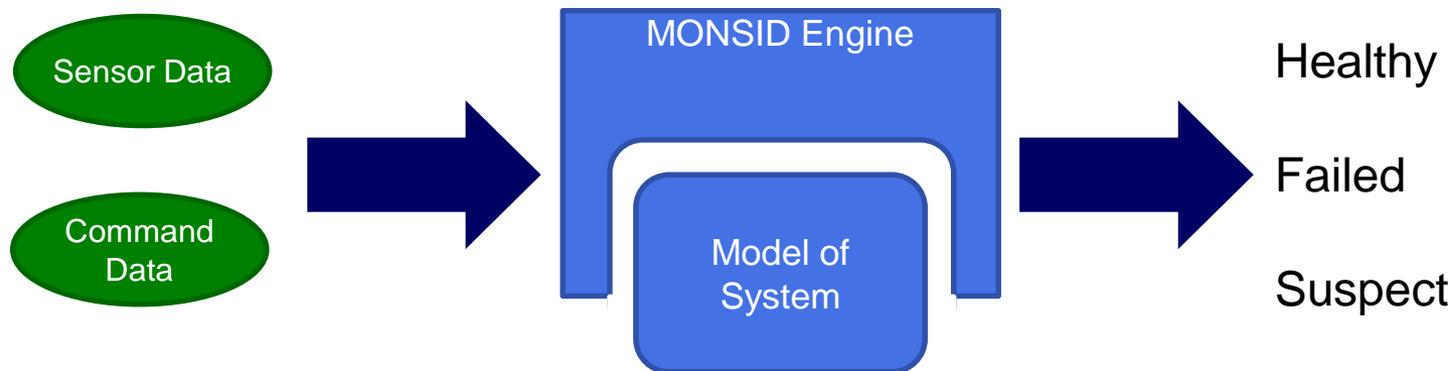
139

Credit: Adapted from slides by Ksenia Kolcio

MONSID System Health Management



- MONSID: **M**odel-based **O**ff-Nominal **S**tate **I**solation and **D**etection
 - Off-nominal includes anomalous, degraded, and failed conditions
- Provides fault detection and isolation
- Provides inputs to response and recovery actions

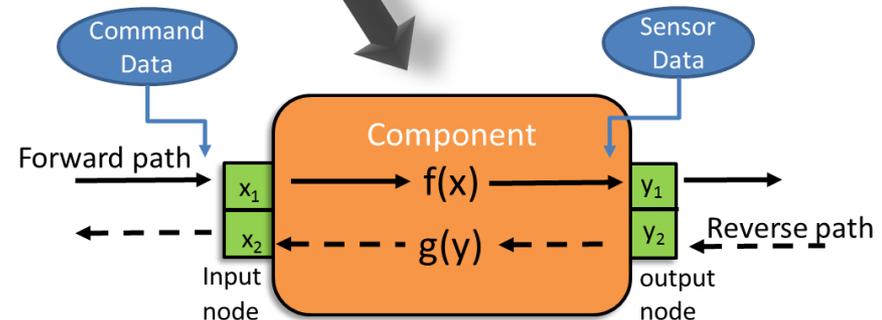
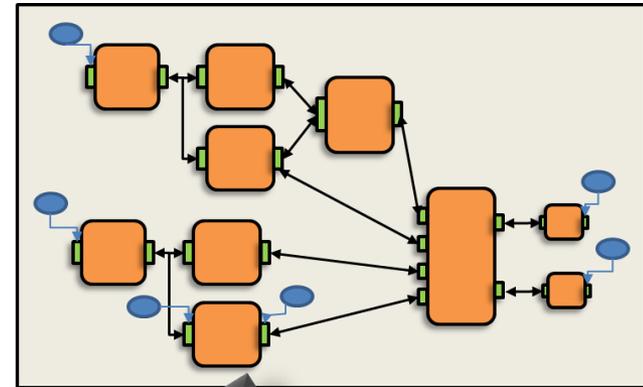


Credit: Adapted from slides by Ksenia Kolcio

MONSID Models



- Interconnected components representing hardware nominal behavior
- Sensor and command data propagated through model in forward and reverse directions
- Nominal behavior *constrained* by input-output relationships



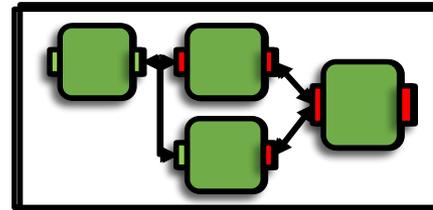
Credit: Adapted from slides by Ksenia Kolcio

How the MONSID Engine Works



Fault Detection

- Sensor & command data propagated through model
- Check consistency at nodes
- Node consistency checks pass
- **Node consistency checks fail**

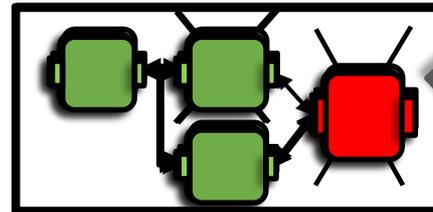


Node Violations!



Fault Isolation

- Iteratively suspend components
 - Suspend a component and check nodes

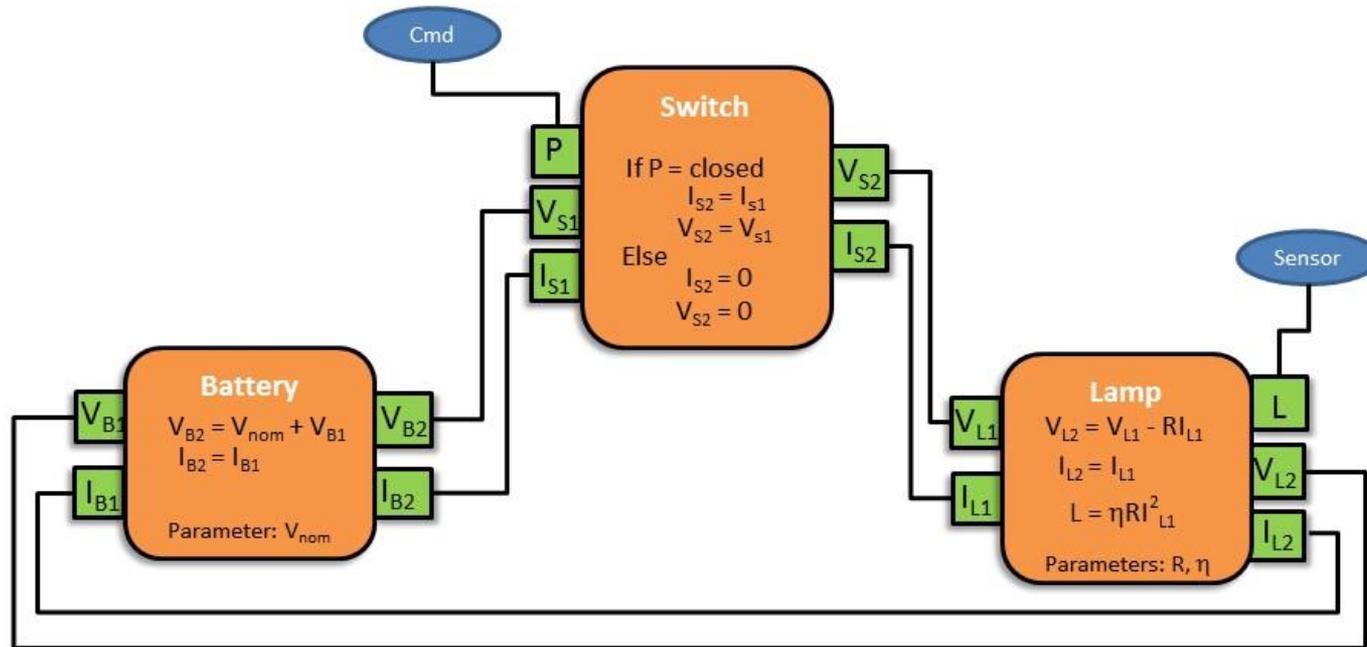


Node Violations!
Resuspend independent
components and check nodes
Component isolated!



Credit: Adapted from slides by Ksenia Kolcio

Simple Flash Light Example



Battery reverse constraints:

$$V_{B1} = V_{B2} - V_{nom}$$

$$I_{B1} = I_{B2}$$

V_{nom} = nominal battery voltage

Switch reverse constraints:

$$\text{If } I_{S2} \neq 0$$

$$I_{S1} = I_{S2}$$

$$V_{S1} = V_{S2}$$

$$P = \text{closed}$$

$$\text{Else}$$

$$I_{S1} = 0$$

$$V_{S1} = 0$$

$$P = \text{open}$$

Lamp reverse constraints:

$$V_{L1} = \sqrt{L/\eta} + V_{L2}$$

$$I_{L1} = \sqrt{L/R\eta}$$

R = resistance
 η = efficiency
 L = lumens

Forward constraints are in component boxes.

* J.F. Castet *et al*, AIAA SciTech 2015

A STATE-CENTRIC THEME

STATE-CENTRIC APPROACH

Architectural Themes



Background

- Space systems are always tightly coupled
- Highly constrained resources demand it
- A key software role is to make this coupling manageable

Key Themes

- State and state uncertainty; estimates are not facts
- Use explicit models
- Express domain knowledge in models (no program logic)
- Use goal-directed operation
- Use real-time reaction to changes in state (closed-loop)
- Use real-time resource management
- Use integral system health management
- Instrument the software to gain visibility into its operation

More Architectural Elements (from MDS)



- Goals vs. Commands
 - Goals are **closed loop** while commands are **open-loop**
- A goal
 - Specifies an objective (intent) as a success criterion to be checked during execution
 - Specifies what to achieve, not how to achieve it
 - During execution a goal knowingly succeeds or fails
 - Permits flexibility in achievement
- State is central
- Goal is a constraint on state
- Goal Network vs. Command Sequence
- Estimates vs. measurements
- Controllers achieve constraint on state

<http://mds.jpl.nasa.gov>

Layered Architecture for Control Systems

Presentation Layer

- Operator interface and tools
- Human decisions & planning
- Longest time-scales

Planning Layer

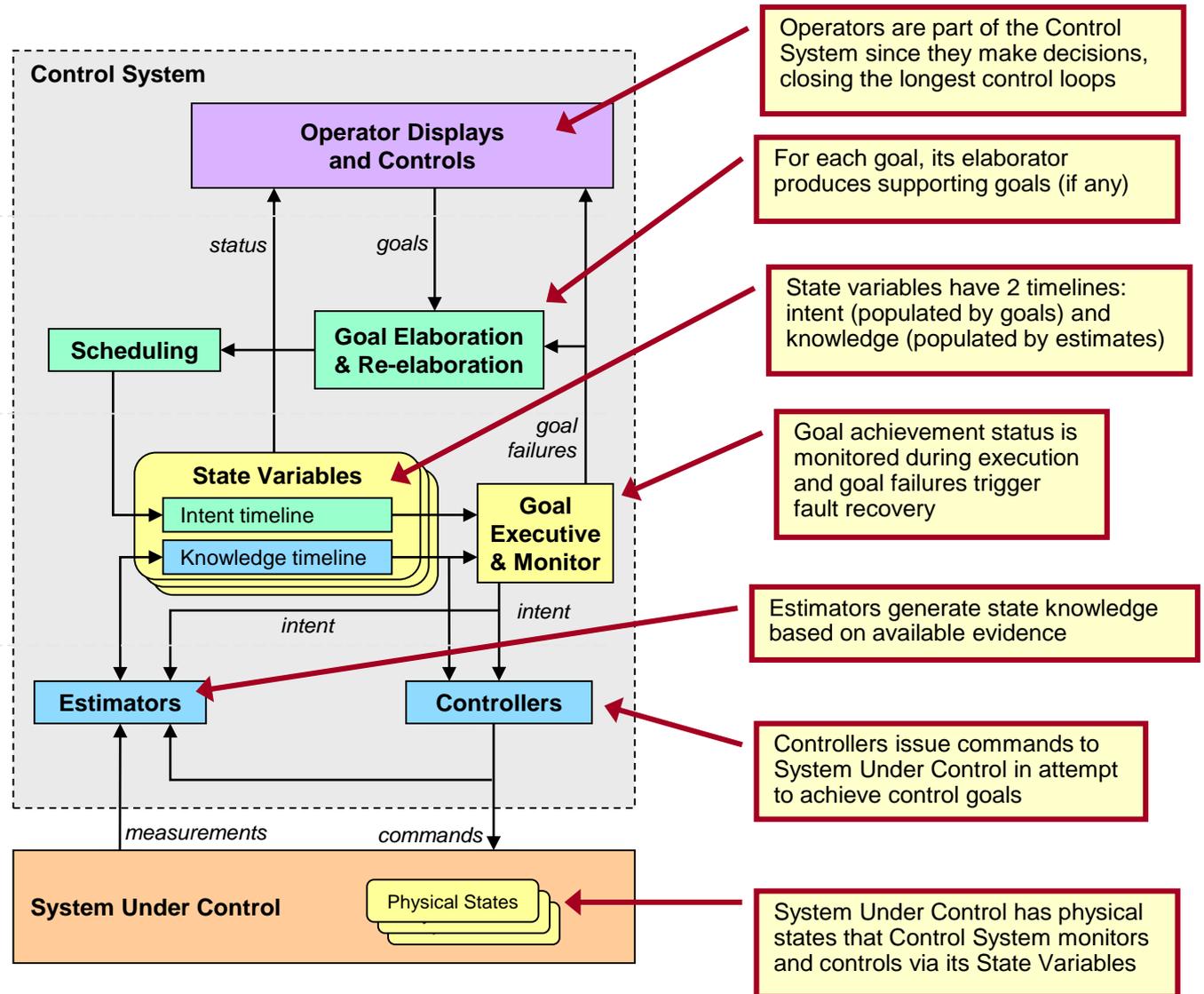
- Deliberative planning
- Long time-scale control loops
- Applies alternate tactics
- Progressive problem escalation

Execution Layer

- Executes plan on intent timeline
- Monitors goal achievement
- Detects control failures
- May handle some contingencies

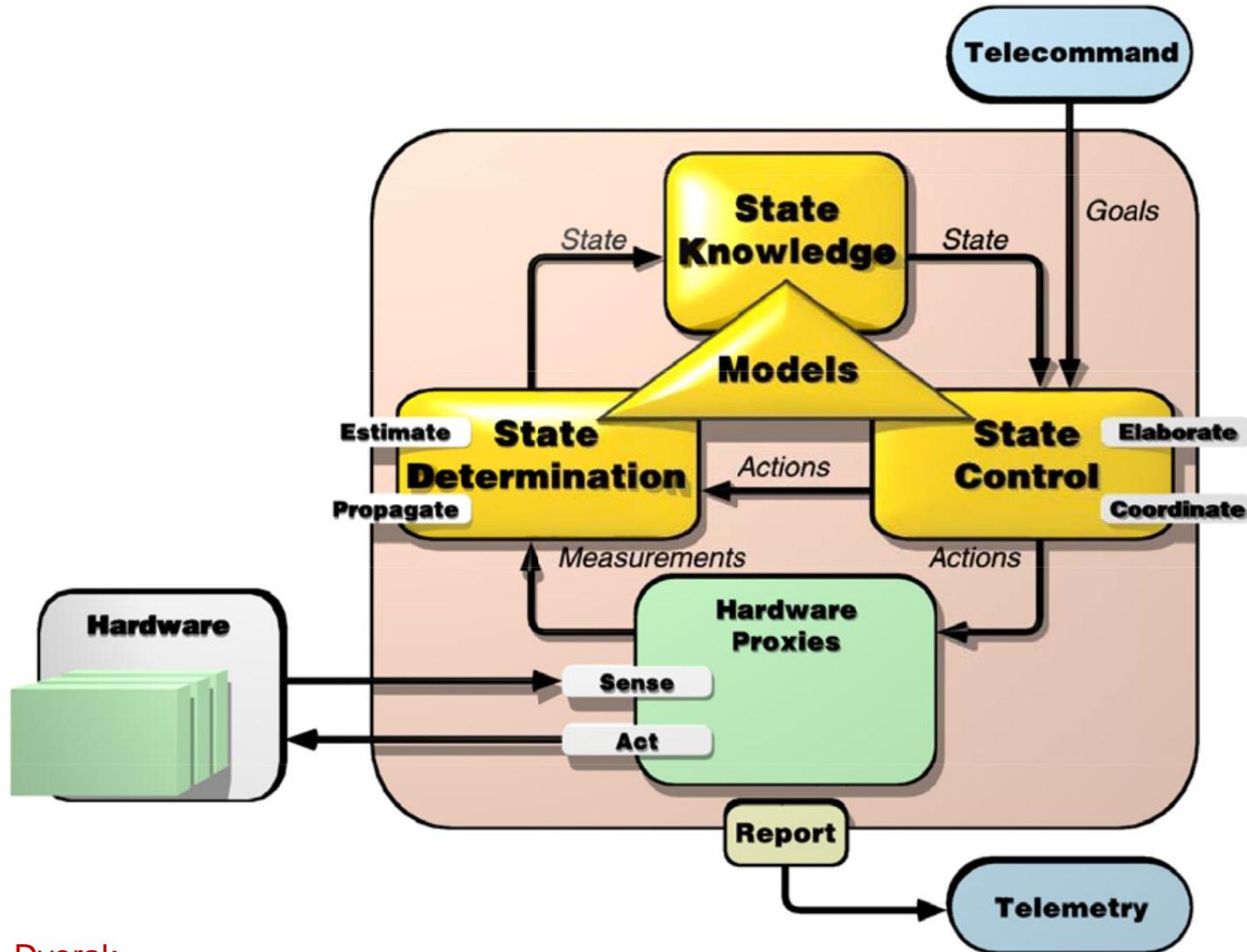
Control Layer

- Achieves goals
- Highly reactive behavior
- Short time-scale control loops



Credit: D. Dvorak

State-Centric Architecture



Credit: D. Dvorak



Advocacy

- Designed to be resilient for space environment
- Enables capturing unusual coupling (e.g. thermal impact on functional capabilities)
- Provides explicit representation of intent

Criticism

- Handling complex states (e.g. surface representation)
- Scalability (explicit states could explode in numbers)
- Separation of estimation and control could be conflict with certain hardware (e.g. controlled motor)
- Always expressing goals as constraints on state can be challenging for some situations

Lessons Learned



- Think about design: implement and re-implement
- Flexibility comes at a cost: know where you need it
- **Hardware/software:** efforts need to engage both hardware and software providers and supplies
- **Outlook:** autonomous systems is a rapidly growing field with the possibility for several disruptive innovations
- **Process:** need an evolvable process that enables fielding new innovations and evaluating success prior to incorporation into the standard

Summary



- Continued architectural themes
- Sequencing vs. Task Networks
- State-Centric approaches

LECTURE 5

NAVIGATION EXAMPLE

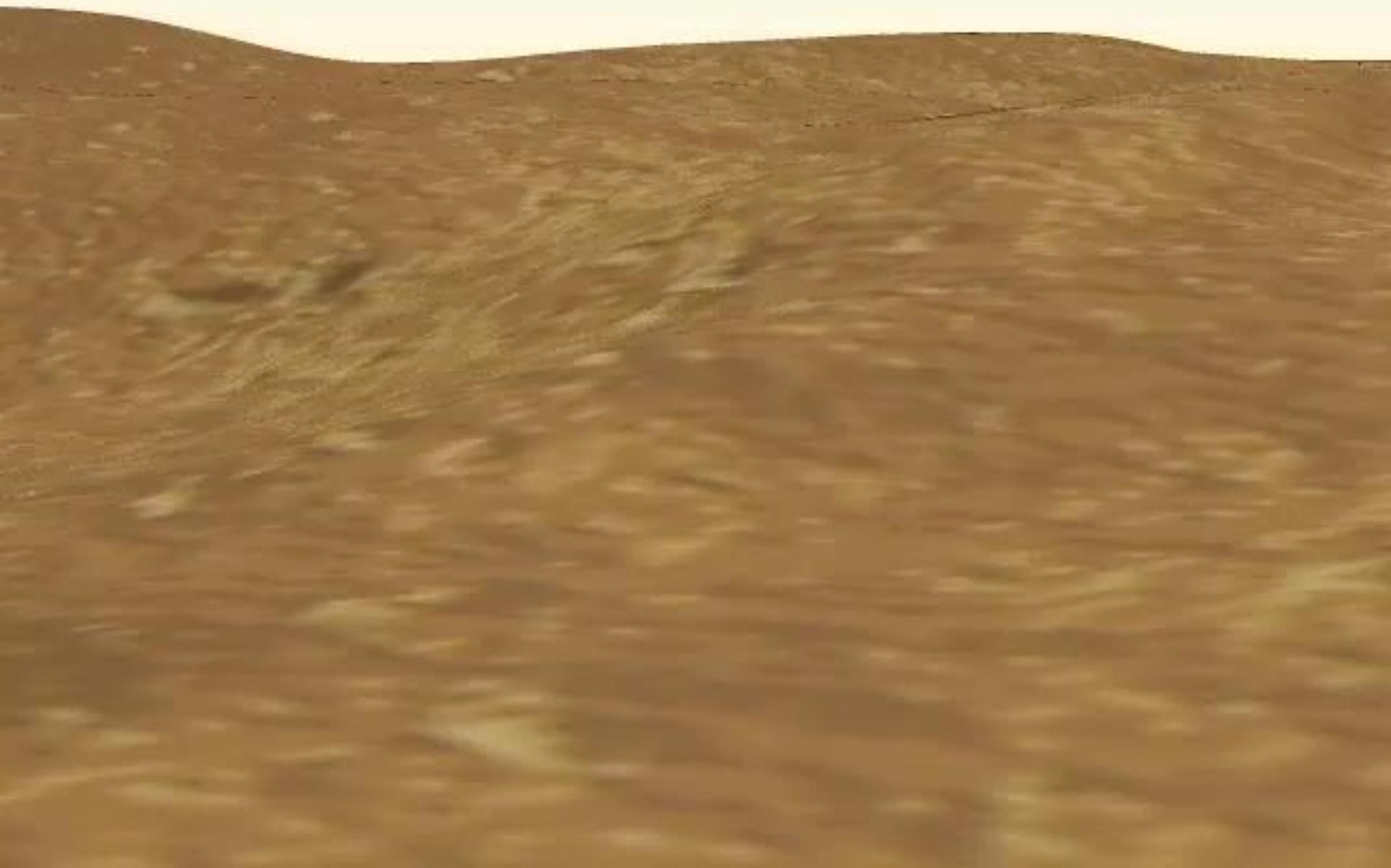


Presentation Overview



- Featured Video: Rover Navigation 101
- Rover navigation on flight rovers
- Navigation framework for testing different algorithms

Rover Navigation: Fun with Spirit and Opportunity



ROVER NAVIGATION

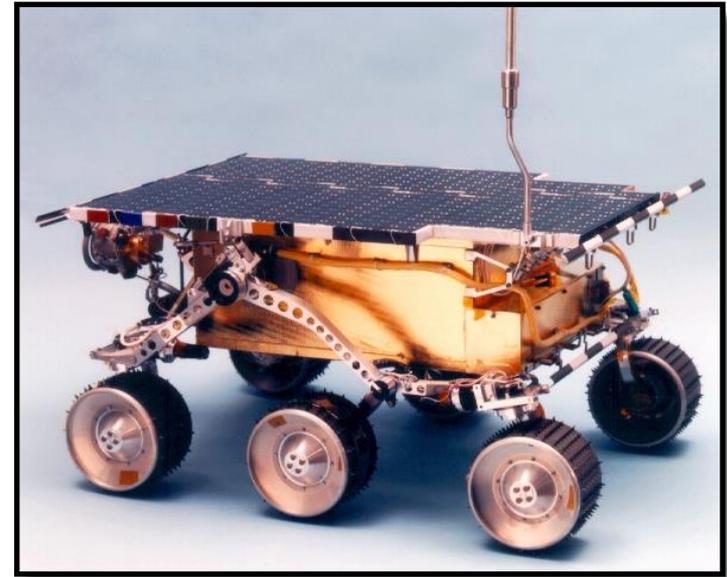


FLIGHT ROVERS

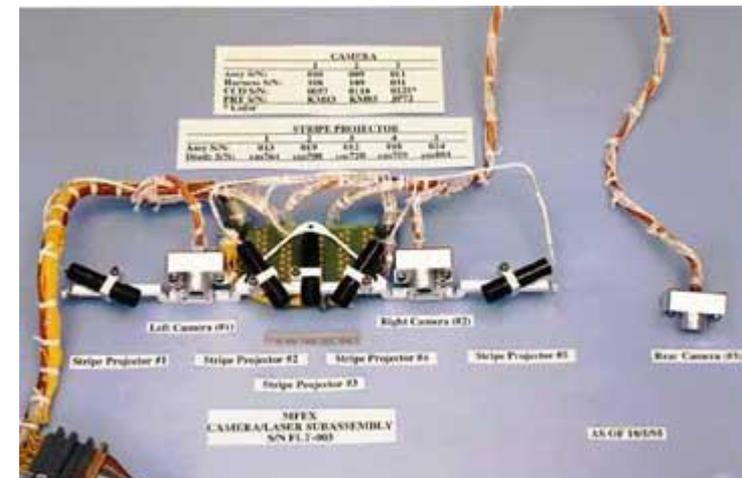
Rover Autonomy on Mars



- **Control:**
 - Bang-bang (on/off switching)
 - Motor encoder (drive) or potentiometer (steering)
- **Hazard detection and avoidance**
 - Uses laser striping and camera
 - Determines presence of obstacles
 - Steers autonomously to avoid obstacles
 - Continues to goal
 - Uses averaged odometry and gyro when stopped to update distance traveled to estimate of progress toward goal



Sojourner (1997)





- **Autonomous navigation:** surface traverse with hazard detection and avoidance (obstacles, sinkage, ...); multi-sol driving
- **Visual odometry (VO):** visual means to estimate relative traverse distances. Used to detect and estimate slip
- **Visual target tracking*:** to approach or view designated target of interest from a distance
- **Autonomous manipulation*:** detecting and avoids obstacles in the arm's workspace
- **Autonomous sampling*:** reacting to drilling or coring anomalies
- **Opportunistic Science:** detecting unique visual features to acquire further measurement

* Demonstrated by used sparingly



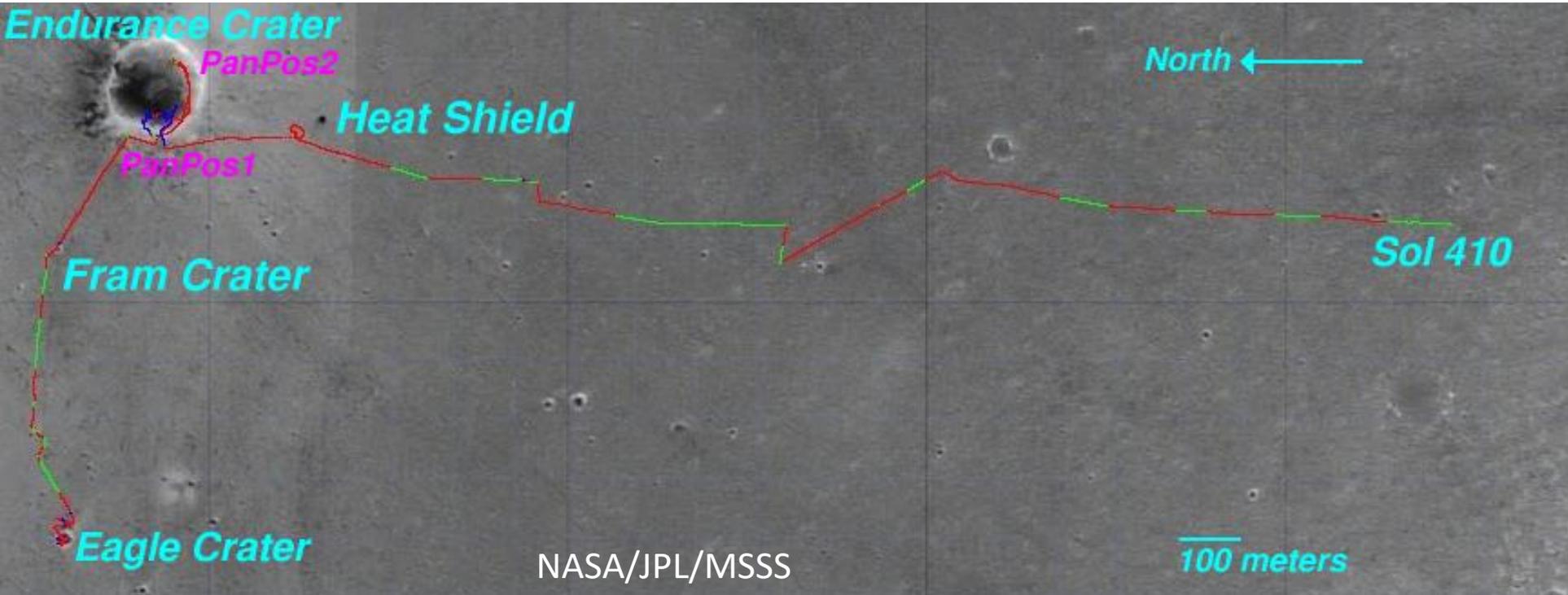
Spirit (2004)

Opportunity (2004)



Curiosity (2011)

Opportunity Traverse (through Sol 410)



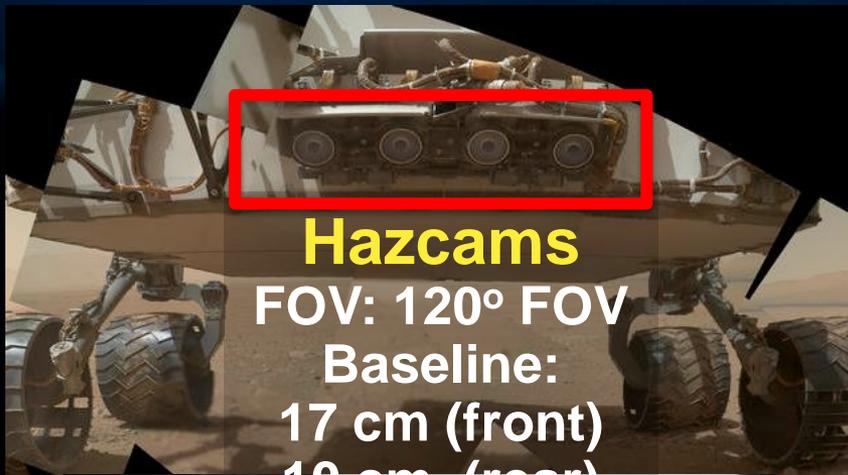
Driving Modes:

Adapted from a Slide by M. Maimone

- **Blind Drive** (planned by ground)
- **Autonav** (uses on-board perception and terrain analysis)
- **Visodom** (uses on-board perception to detect slip)



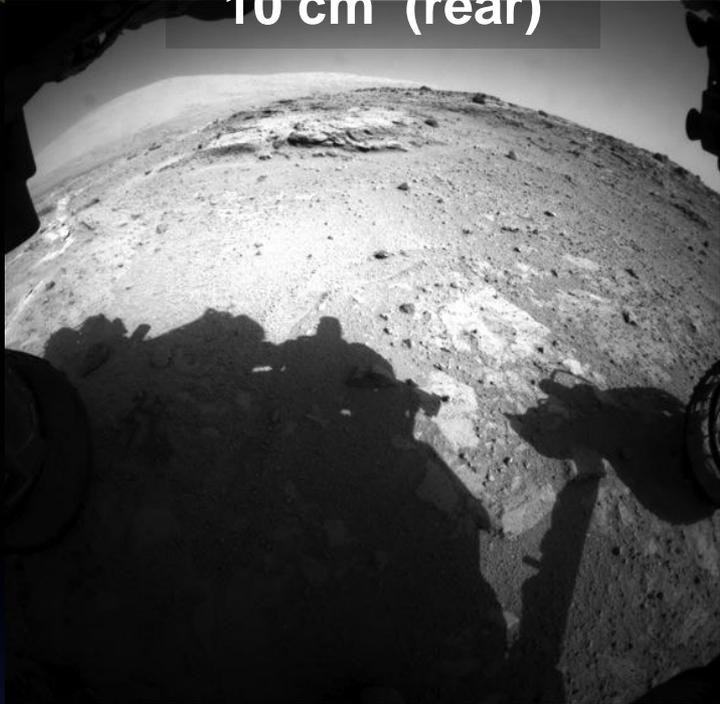
Rover Perception



Hazcams
FOV: 120° FOV
Baseline:
17 cm (front)
10 cm (rear)



Navcams
FOV: 45° FOV
Baseline:
42 cm (front)
~2 m off ground



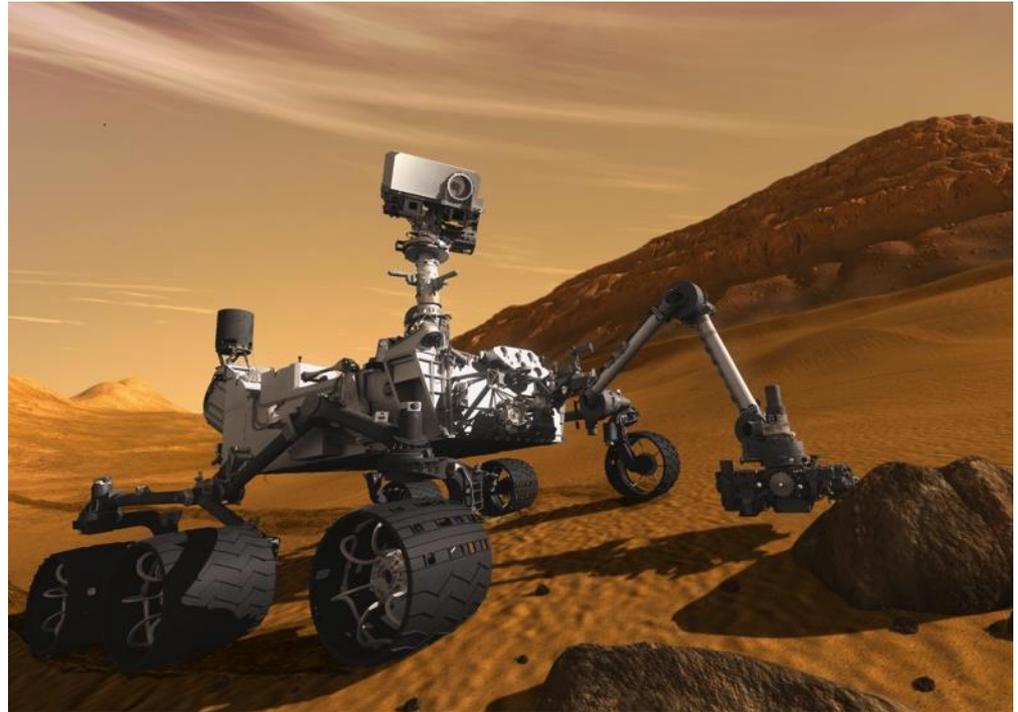
ROVER NAVIGATION



Orbiter and Rover



Artist's Concept. NASA/JPL-Caltech



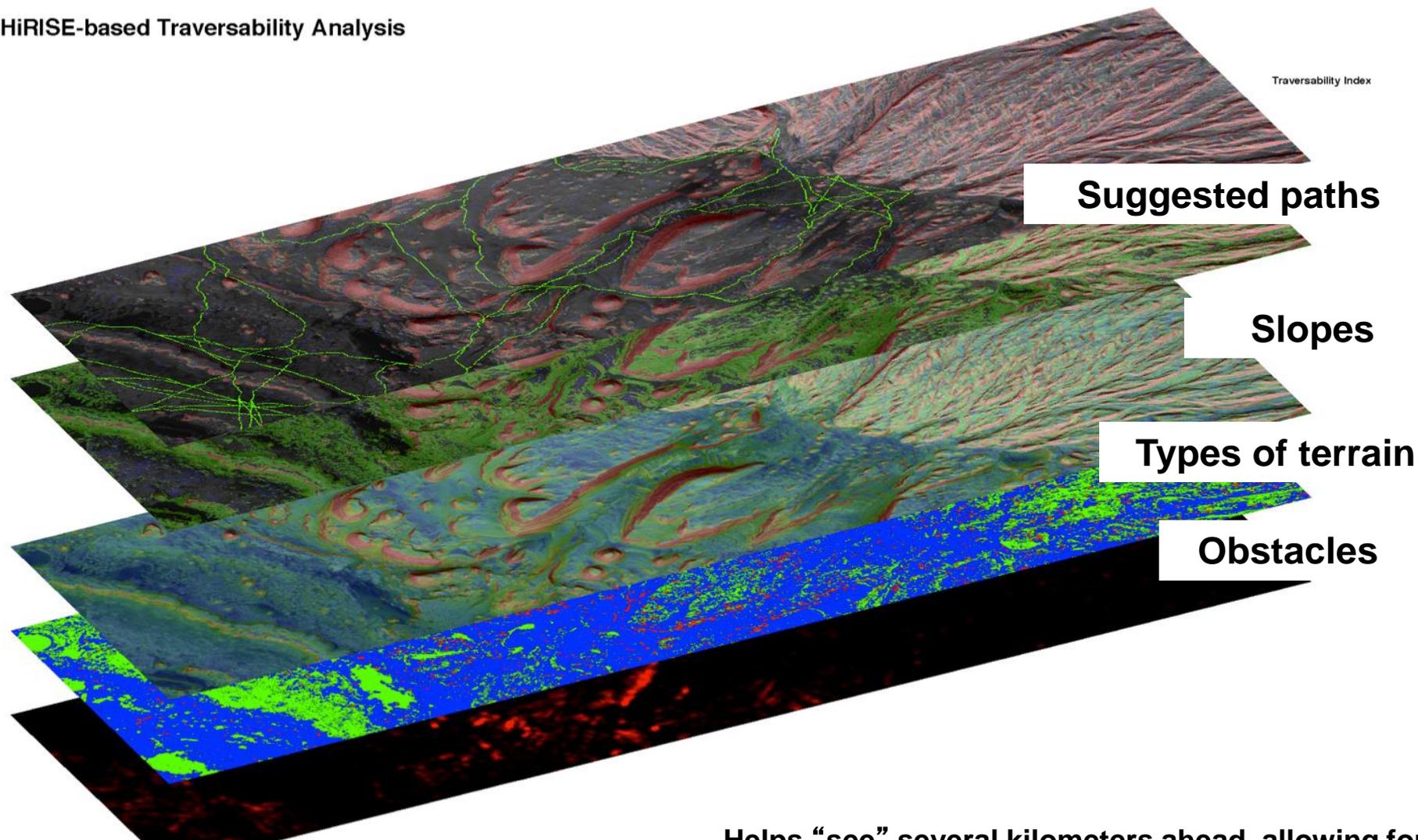
Artist's Concept. NASA/JPL-Caltech

Mapping and communication

Data from the Mars Reconnaissance Orbiter

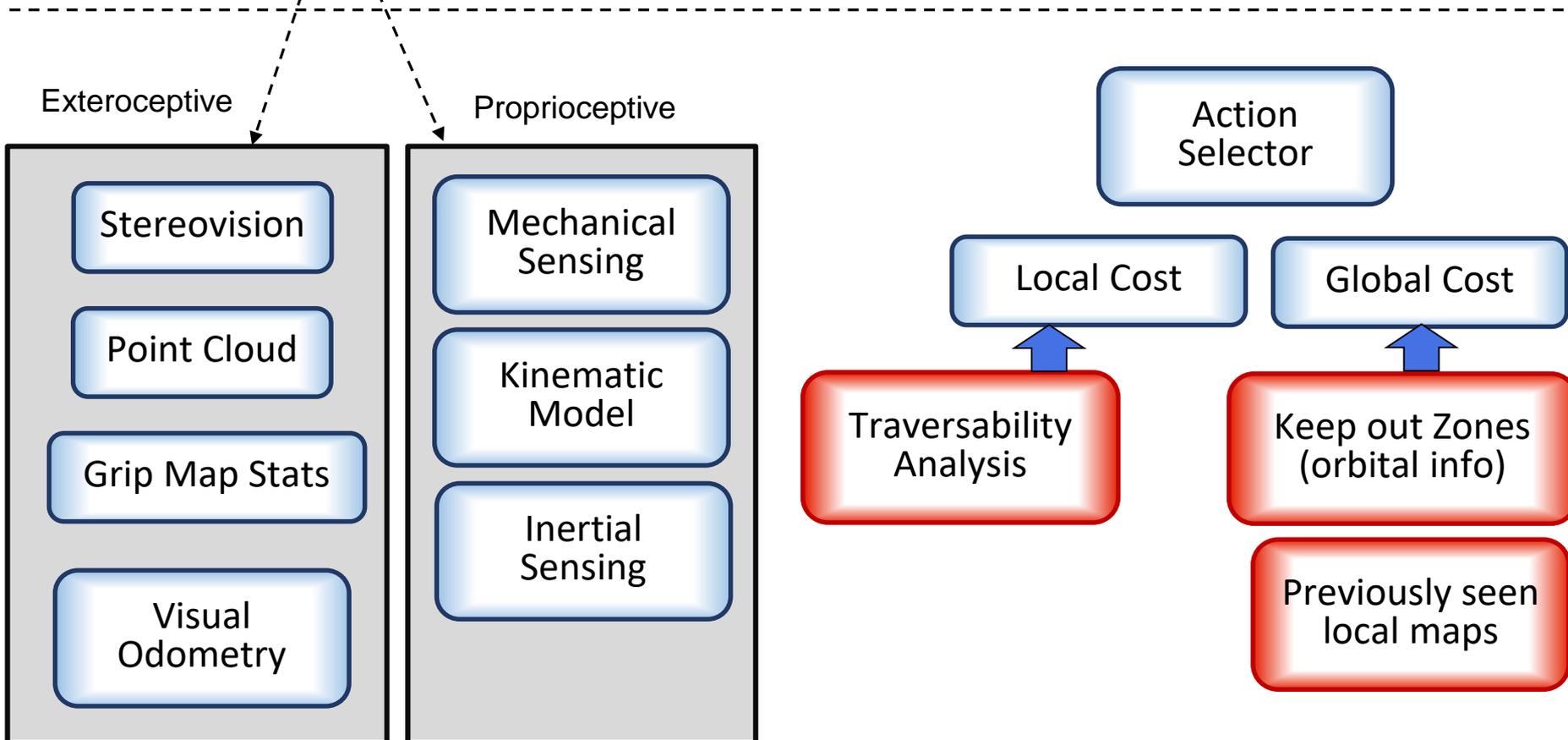
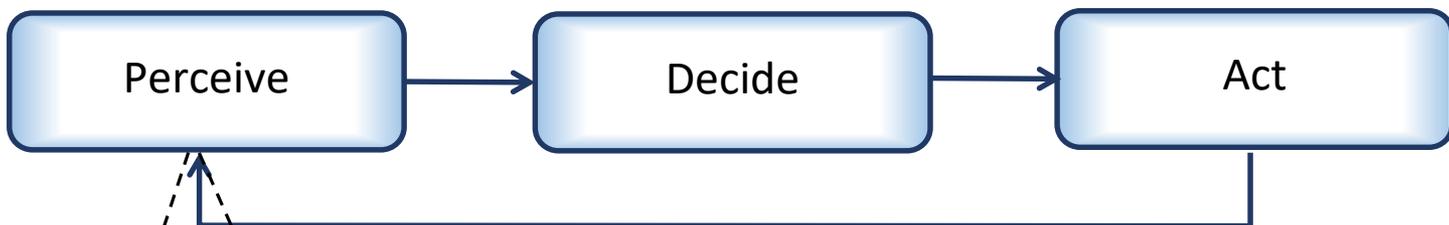


HiRISE-based Traversability Analysis



Helps “see” several kilometers ahead, allowing for these different modes of driving.

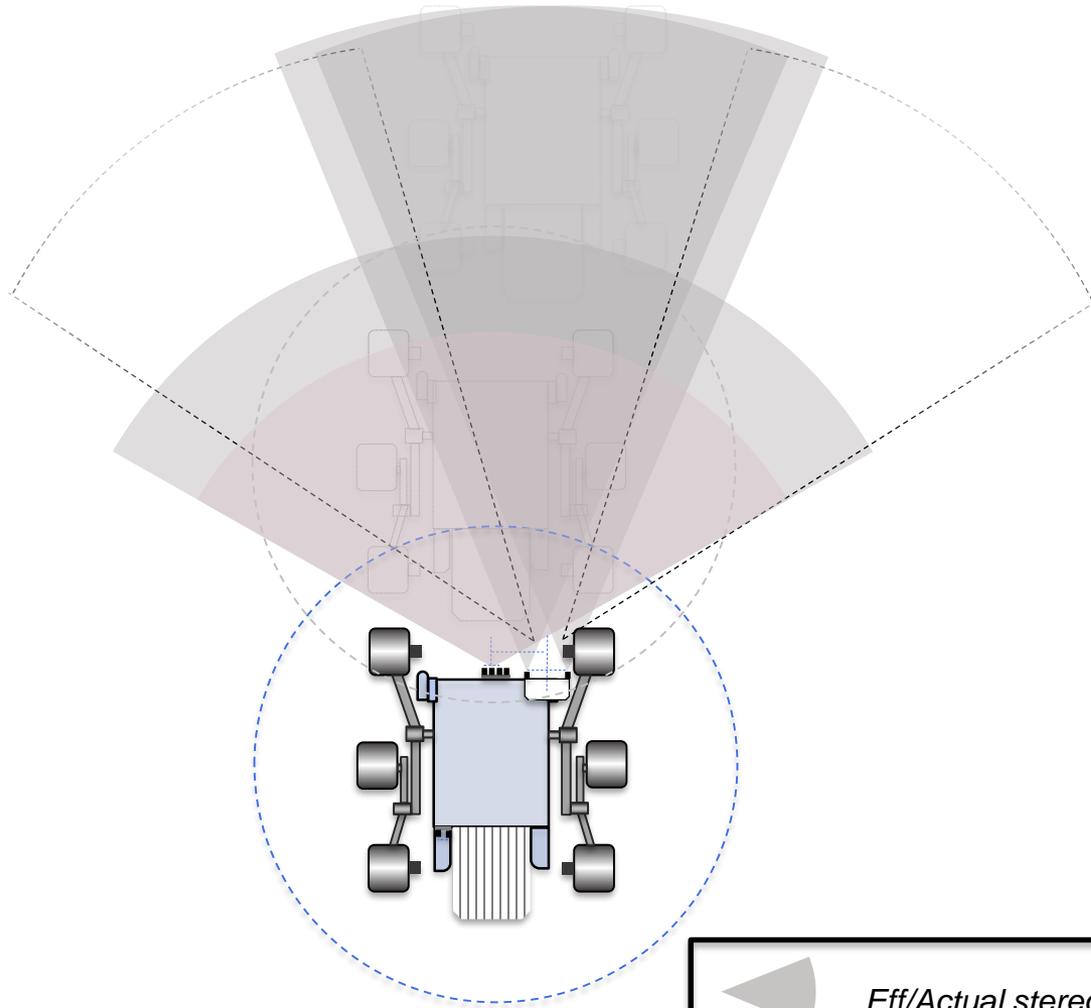
Autonomous Navigation



ROVER NAVIGATION

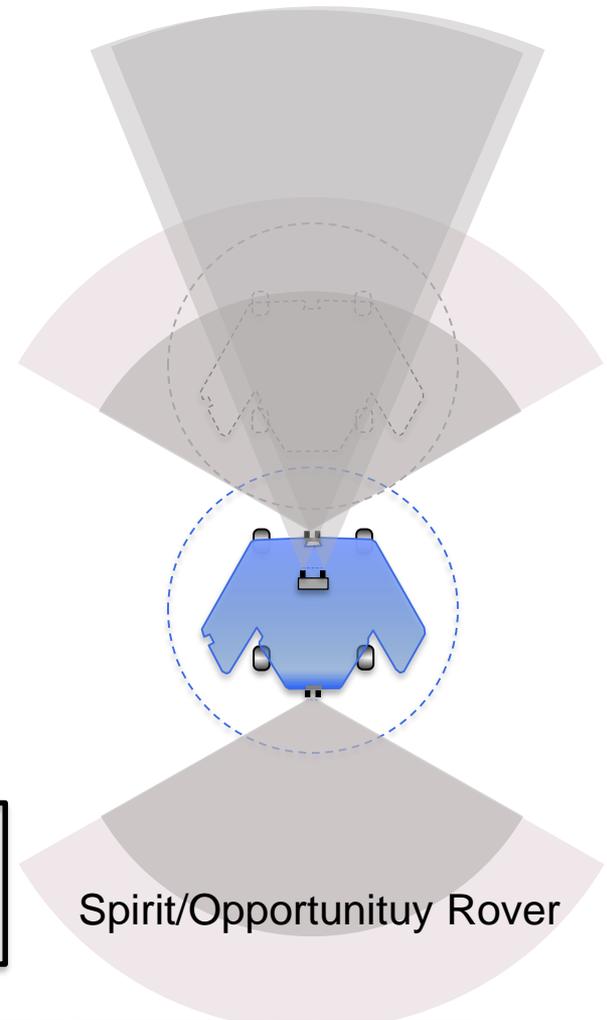
- PERCEPTION
- TRAVERSABILITY ANALYSIS
- ACTION SELECTION

Perception Envelope



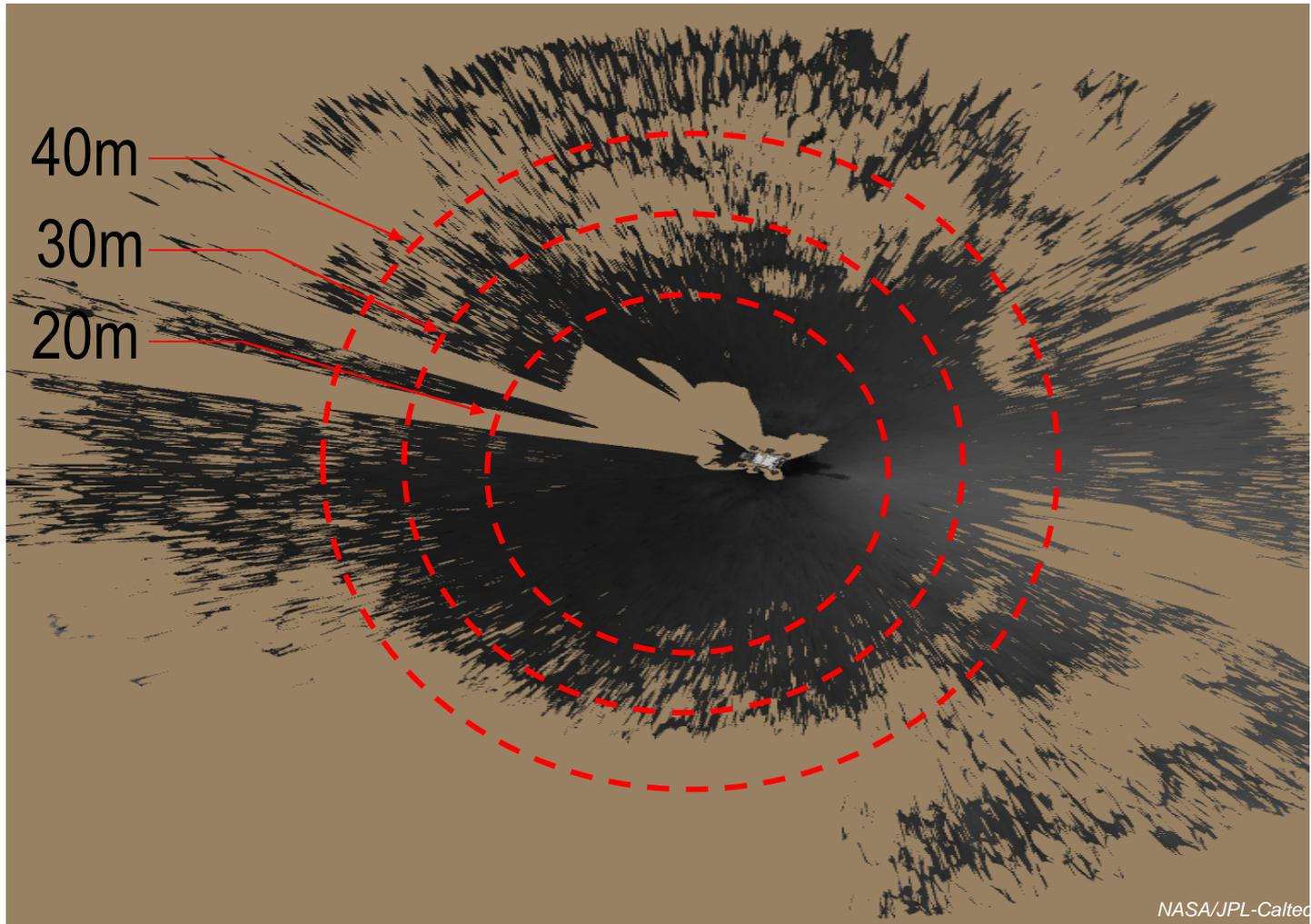
Curiosity Rover

 *Eff/Actual stereo range*
All geometries are properly scaled



Spirit/Opportunity Rover

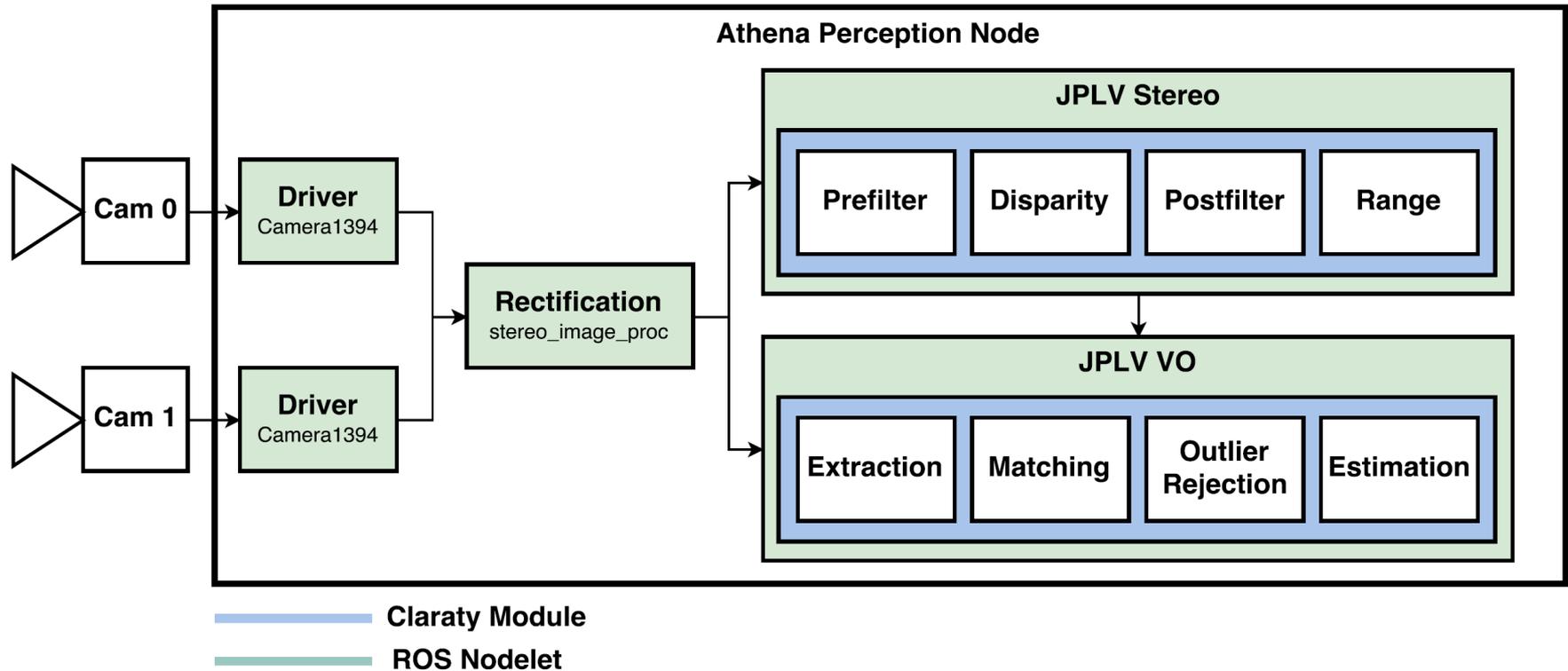
Typical Data from Navcams



NASA/JPL-Caltech

The dark areas closest to the rover are the safest places to drive

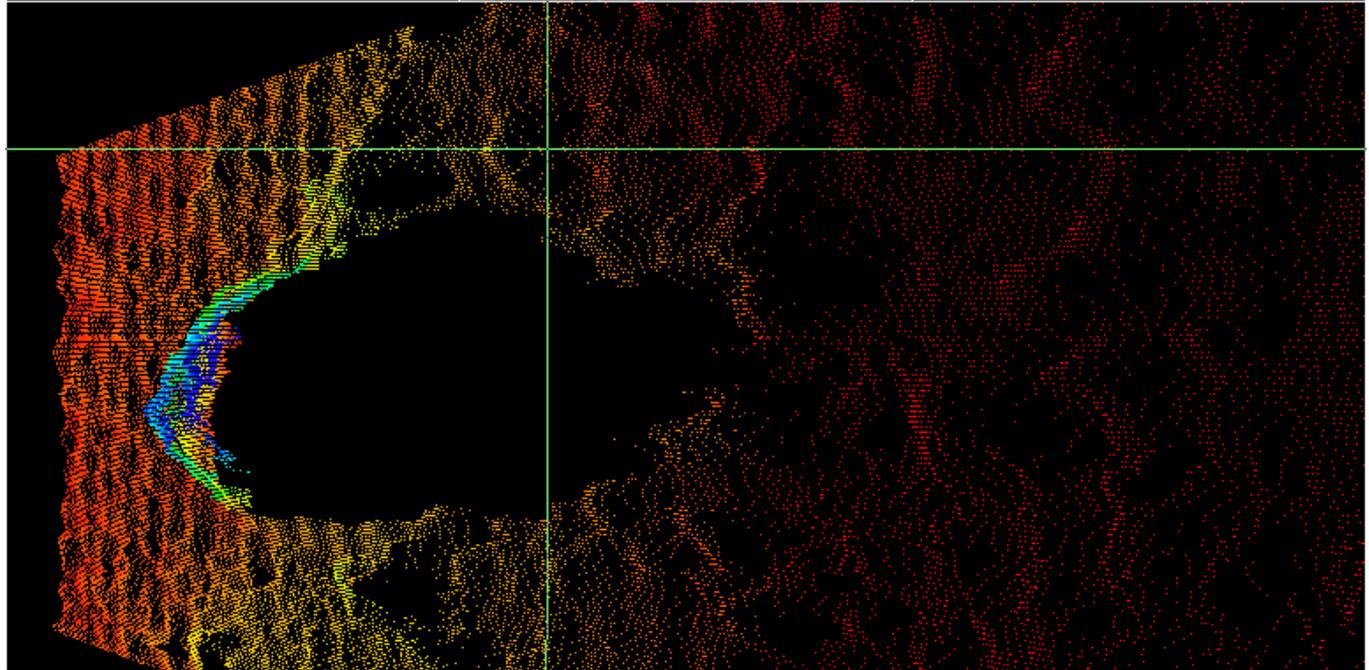
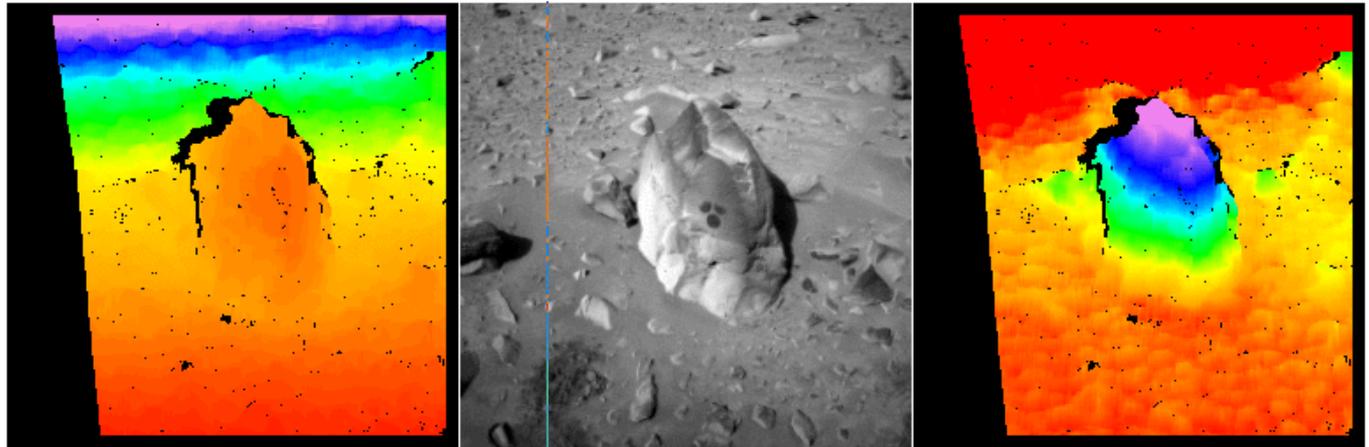
Modular Pipeline



Credit: adapted from slide by Michael Paton

I. Nenas, "Claraty: A collaborative software for advancing robotic technologies," in NASA Science and Technology, 2007 Conference on, 2007.

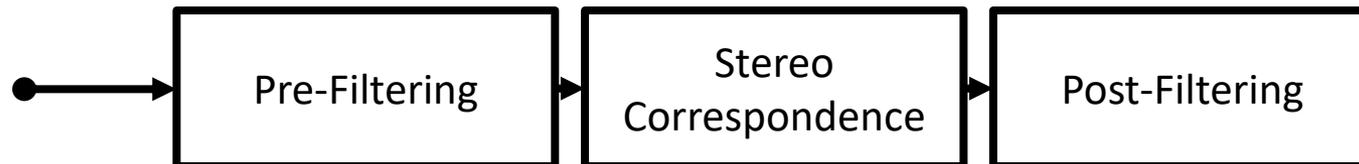
Point Cloud



Credit: Larry Matthies,
Todd Litwin, Mark Maimone

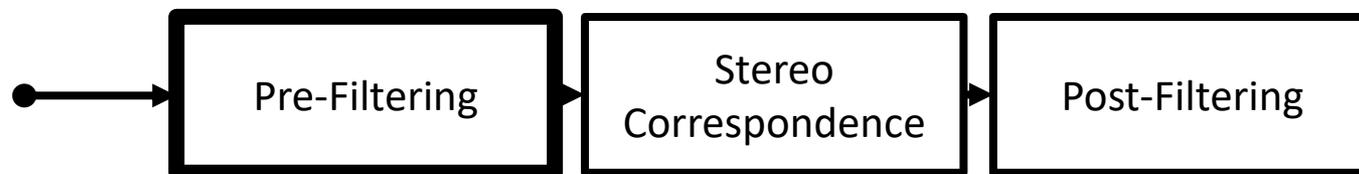


Stereo Disparity Pipeline



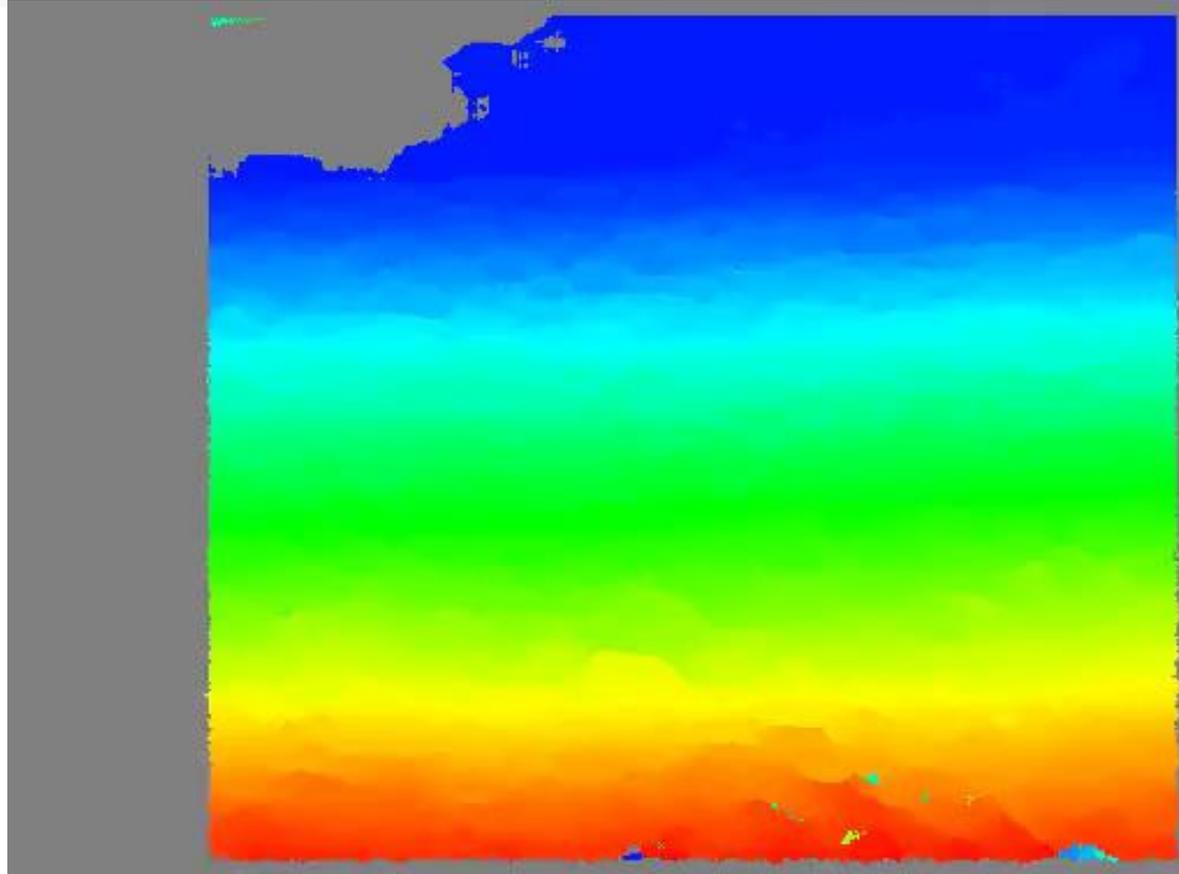
Credit: adapted from slide by Michael Paton

Stereo Disparity Pipeline



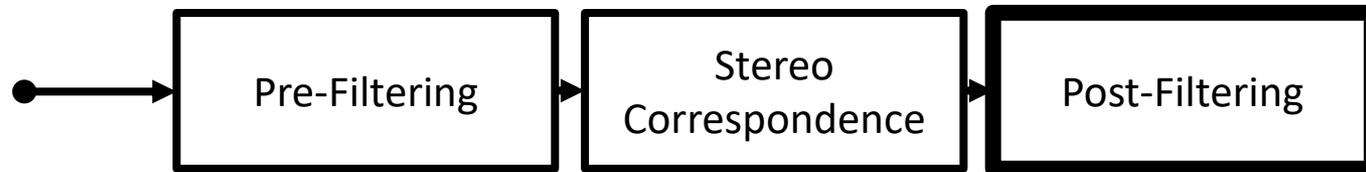
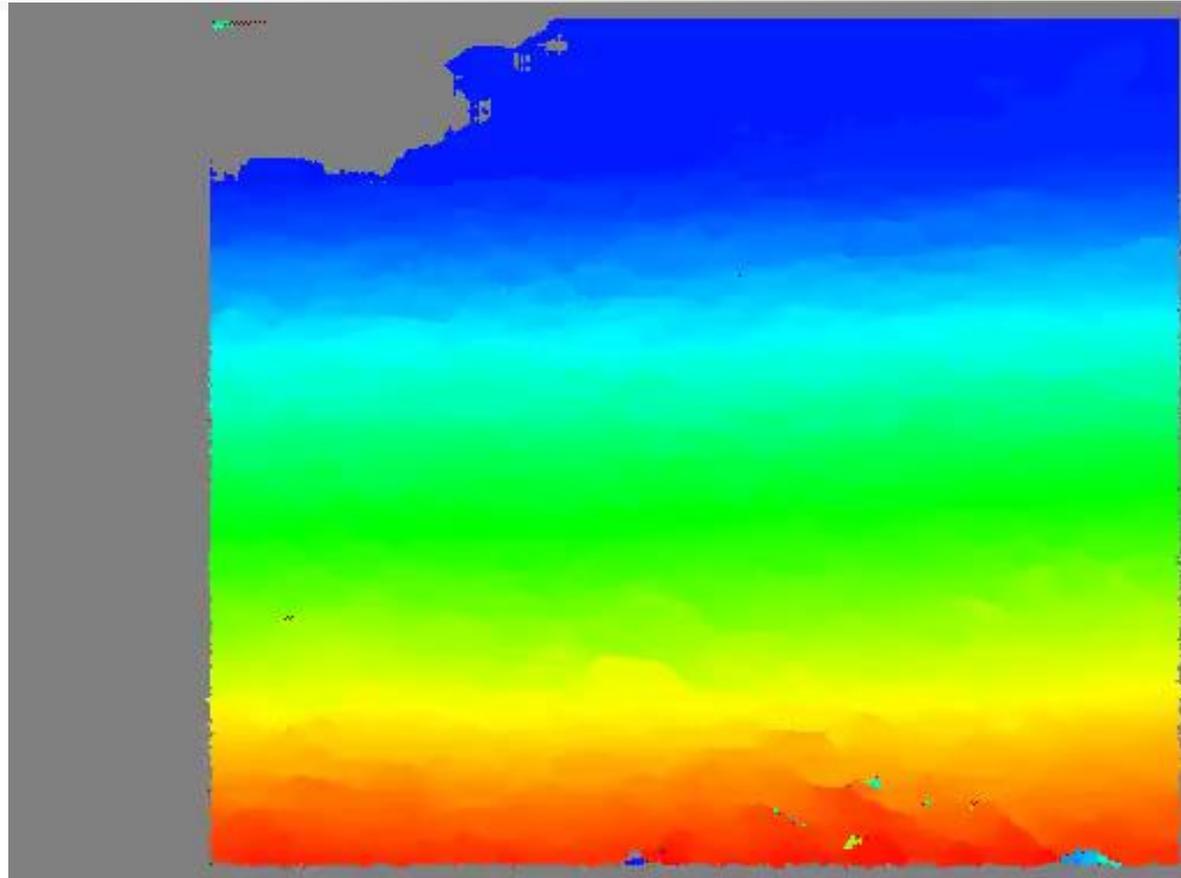
Credit: adapted from slide by Michael Paton

Stereo Disparity Pipeline



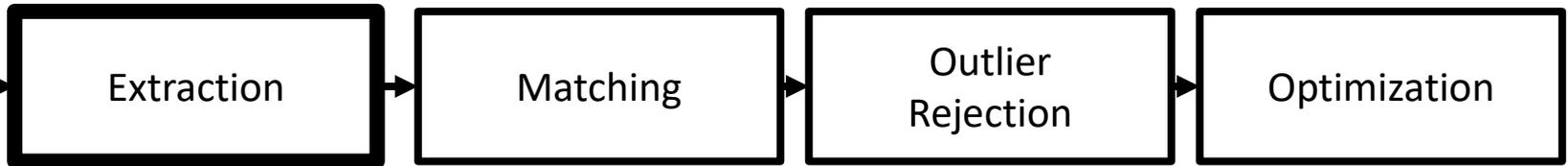
Credit: adapted from slide by Michael Paton

Stereo Disparity Pipeline



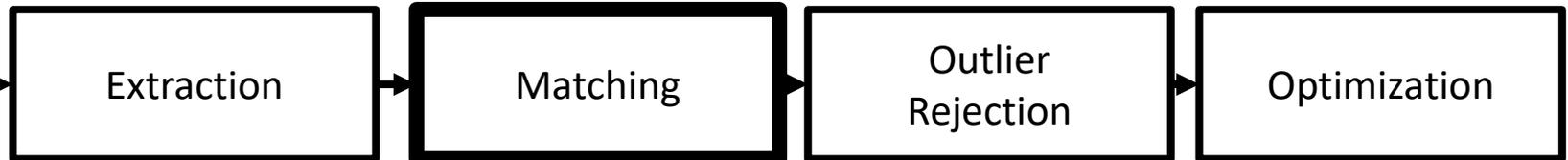
Credit: adapted from slide by Michael Paton

Visual Odometry Pipeline



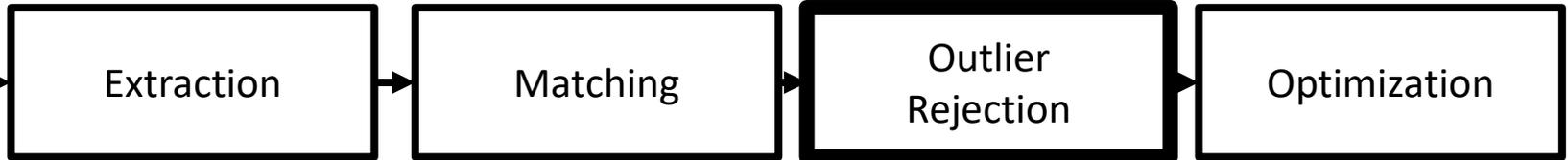
Credit: adapted from slide by Michael Paton

Visual Odometry Pipeline



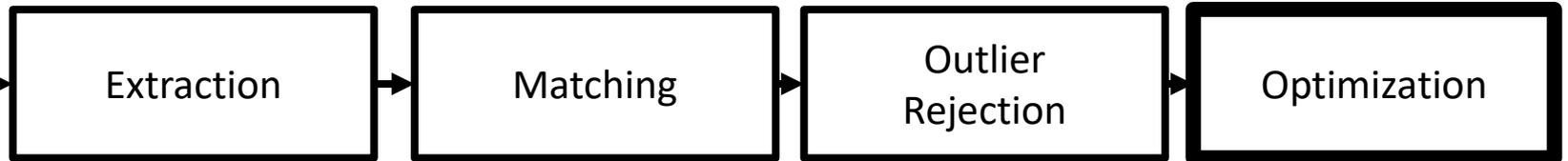
Credit: adapted from slide by Michael Paton

Visual Odometry Pipeline



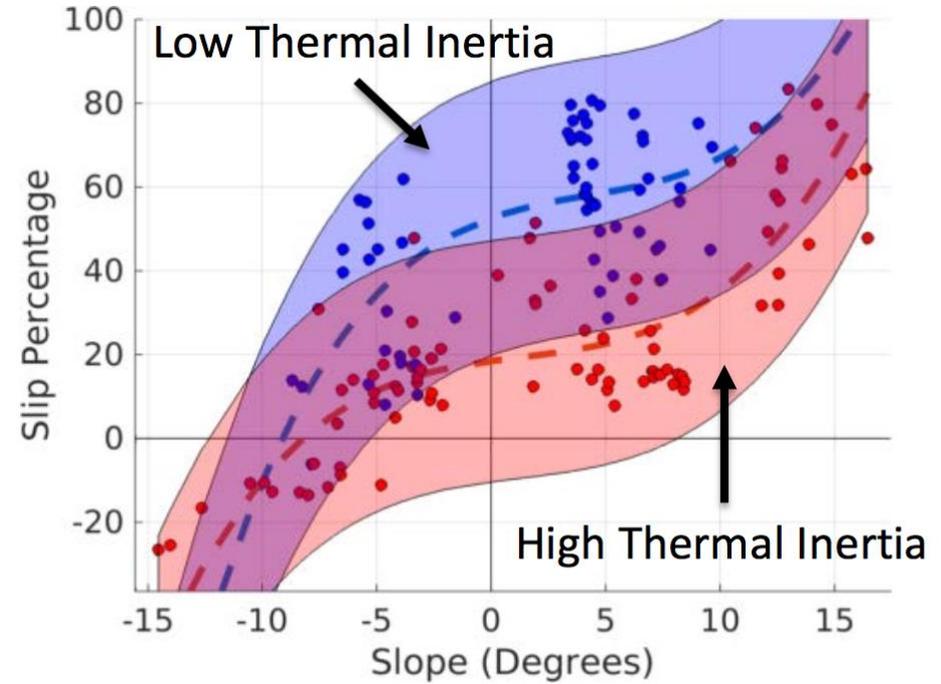
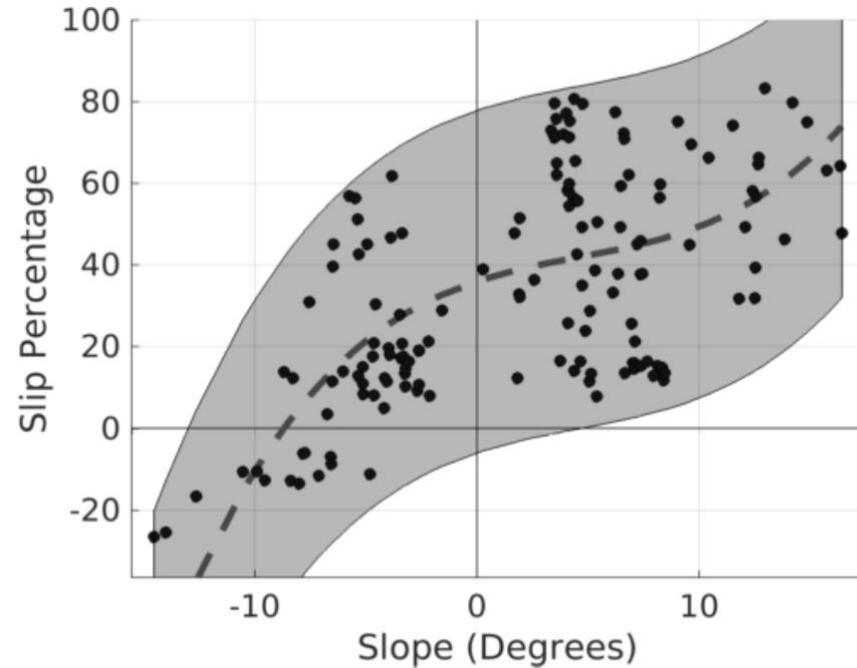
Credit: adapted from slide by Michael Paton

Visual Odometry Pipeline



Credit: adapted from slide by Michael Paton

Correlating Thermal Inertia and Slip

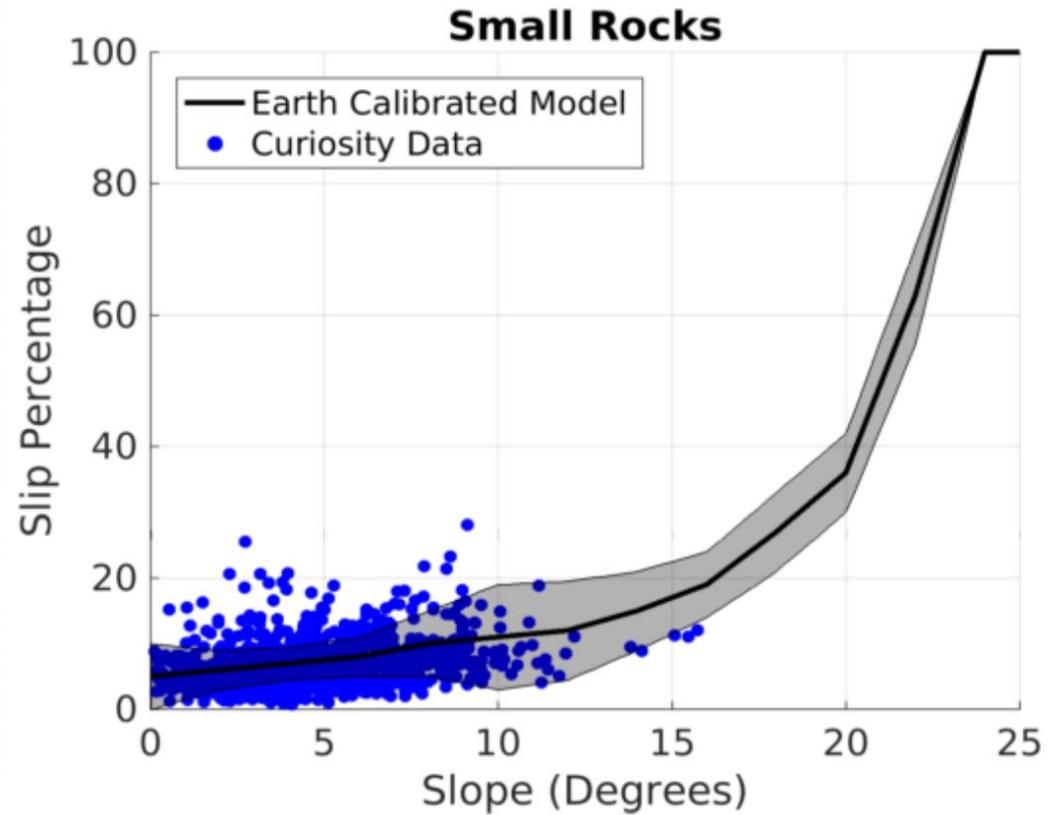
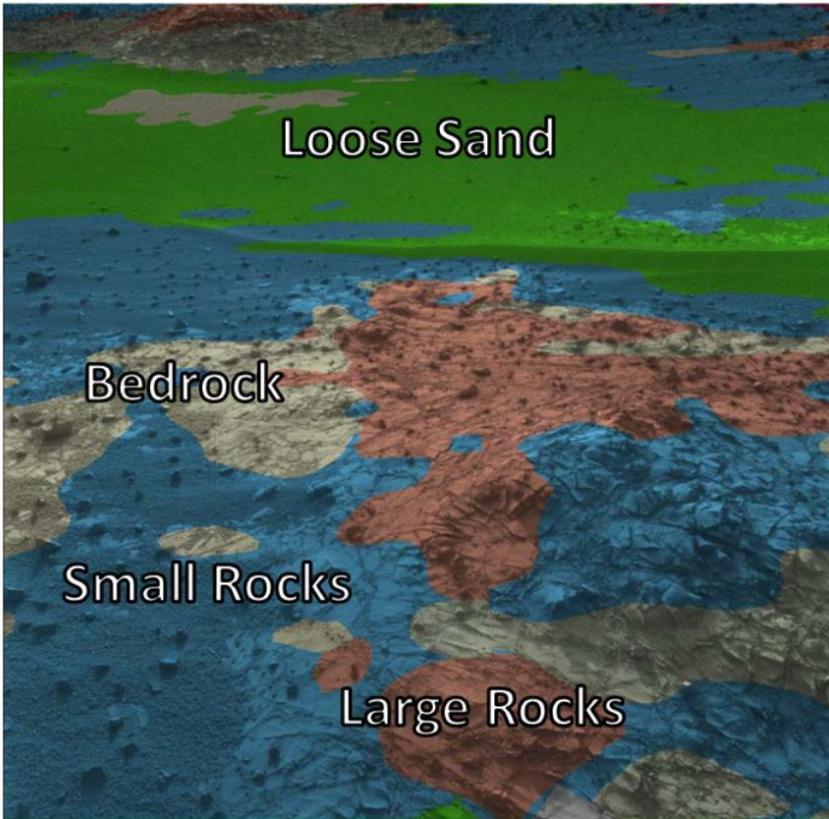


Model	Cross-Validation Error
No Thermal	24.0
Thermal Inertia	17.5

$$p(\text{slip} | I, \text{slope}) = \sum_{i=1}^2 \pi_i(I) p(\text{slip} | \text{slope})$$

Credit: Chris Cunningham

Terrain Classification Helps



Credit: Chris Cunningham

LECTURE 6

SURFACE NAVIGATION EXAMPLE



TANGO ON RACING QUADROTORS

Onboard Vision-based Flight

15 May 2017



Jet Propulsion Laboratory
California Institute of Technology

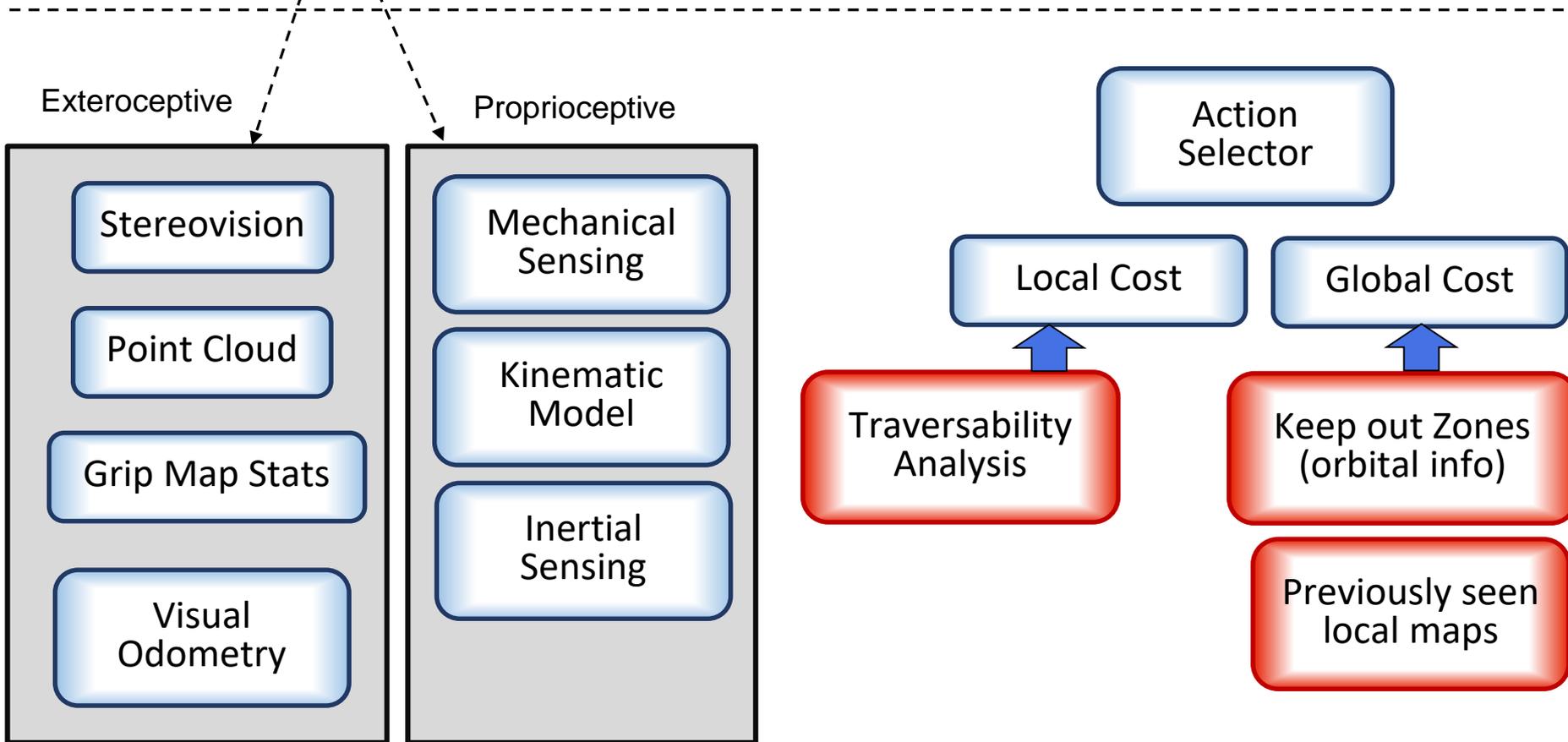
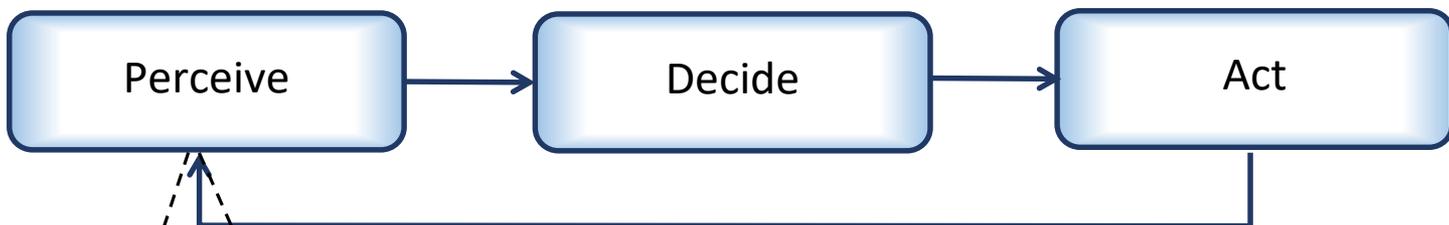
Gene Merewether
Robert G. Reid
Theodore Tzanetos

Presentation Overview



- Recap rover navigation example
 - Perception
 - Traversability analyzer
 - Mobility
 - Software architecture
 - Deployments
- Future directions
 - Cassini Grand Finale(video)
 - Open Ocean Worlds

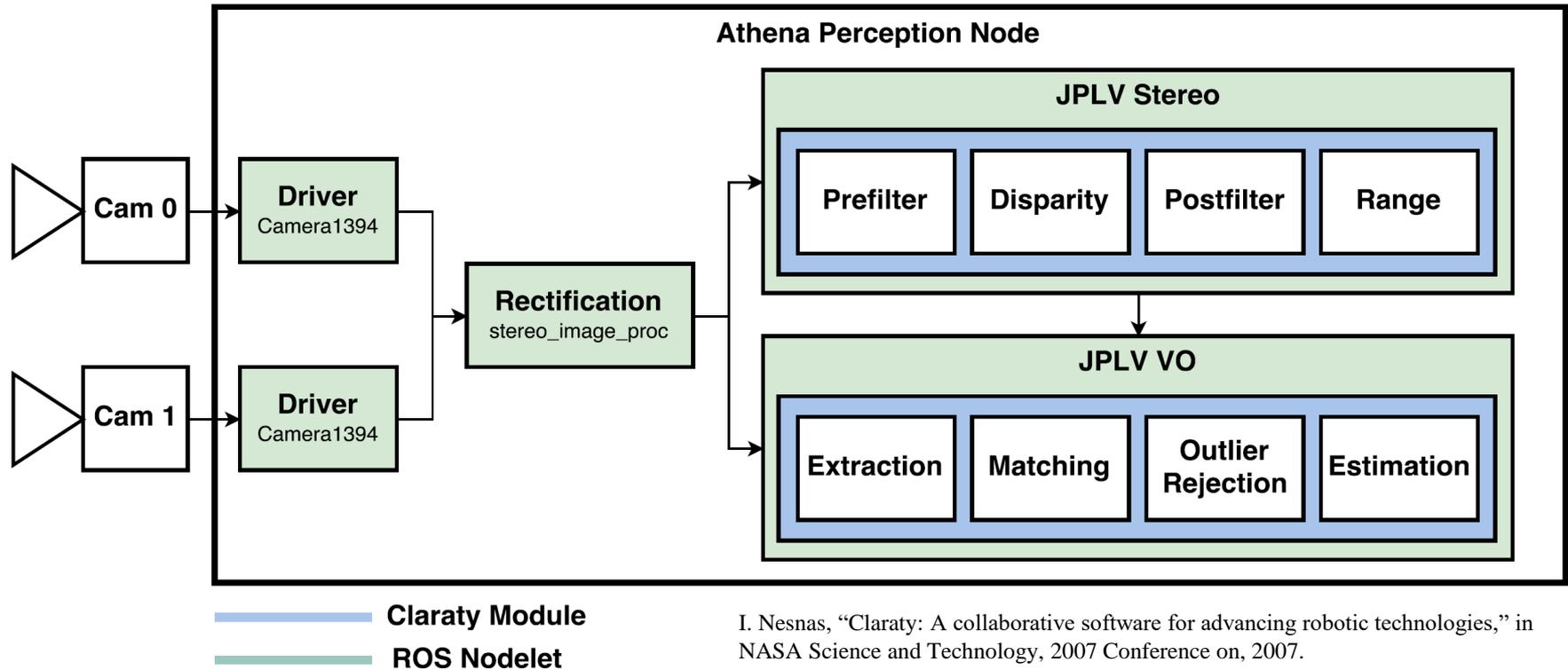
Autonomous Navigation



ROVER NAVIGATION

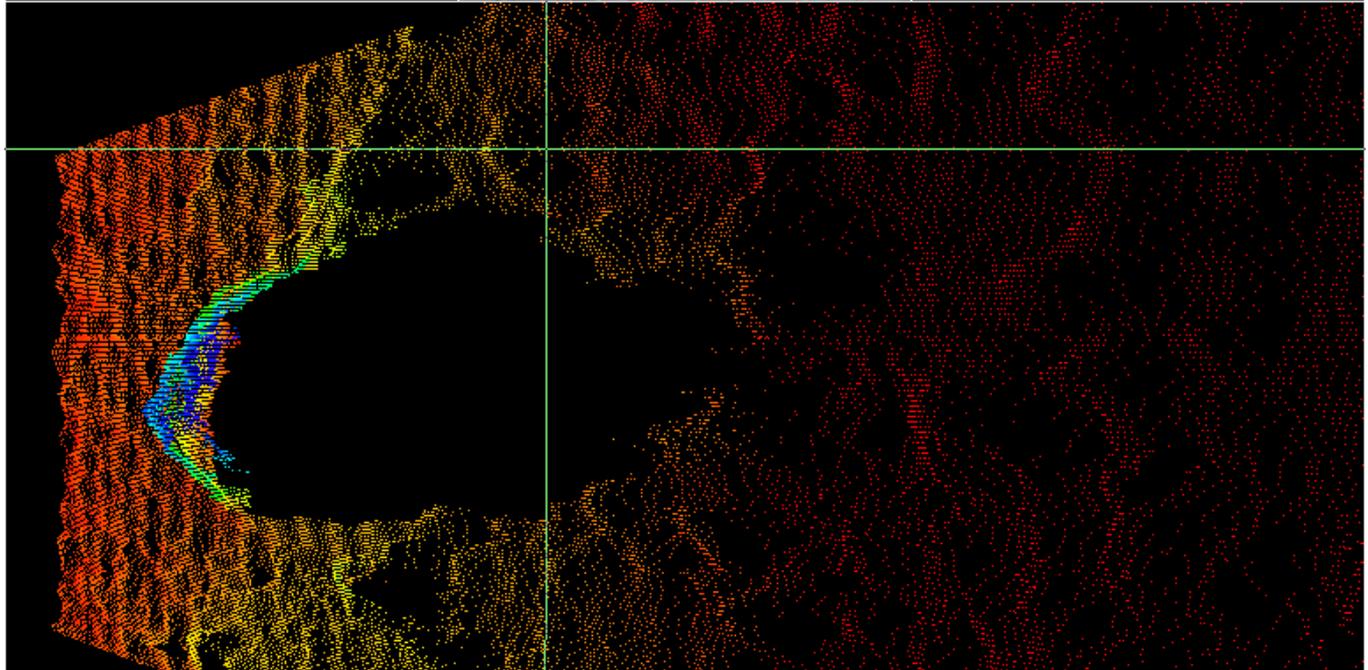
- PERCEPTION
- TRAVERSABILITY ANALYSIS
- ACTION SELECTION

Modular Pipeline



I. Nesnas, "Clarity: A collaborative software for advancing robotic technologies," in NASA Science and Technology, 2007 Conference on, 2007.

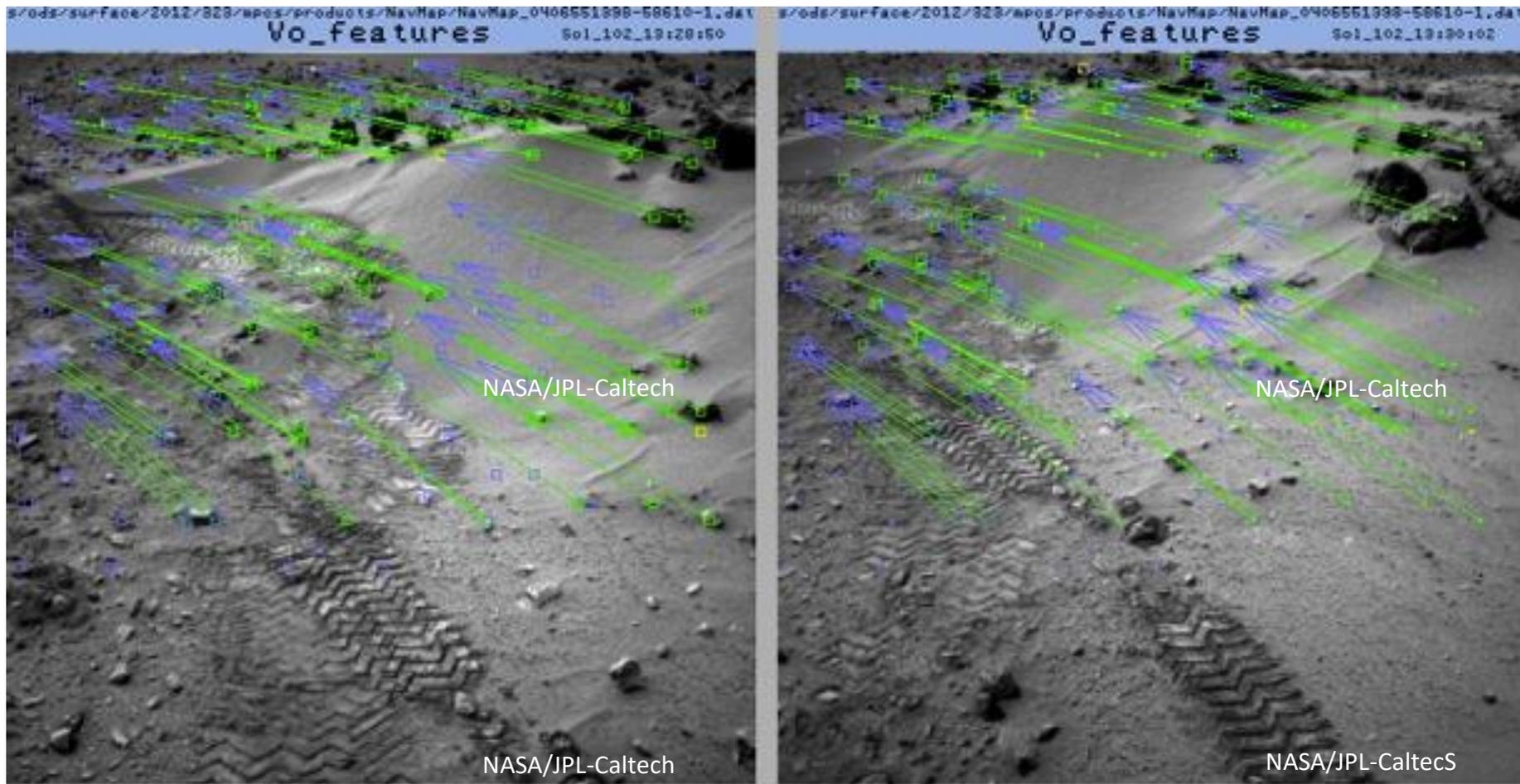
Point Cloud



Credit: Larry Matthies,
Todd Litwin, Mark Maimone



Pose Estimation Using Visual Odometry



Unlike terrestrial robots, Curiosity drives as far as possible between VO images

Credit: Mark Maimone

ROVER NAVIGATION

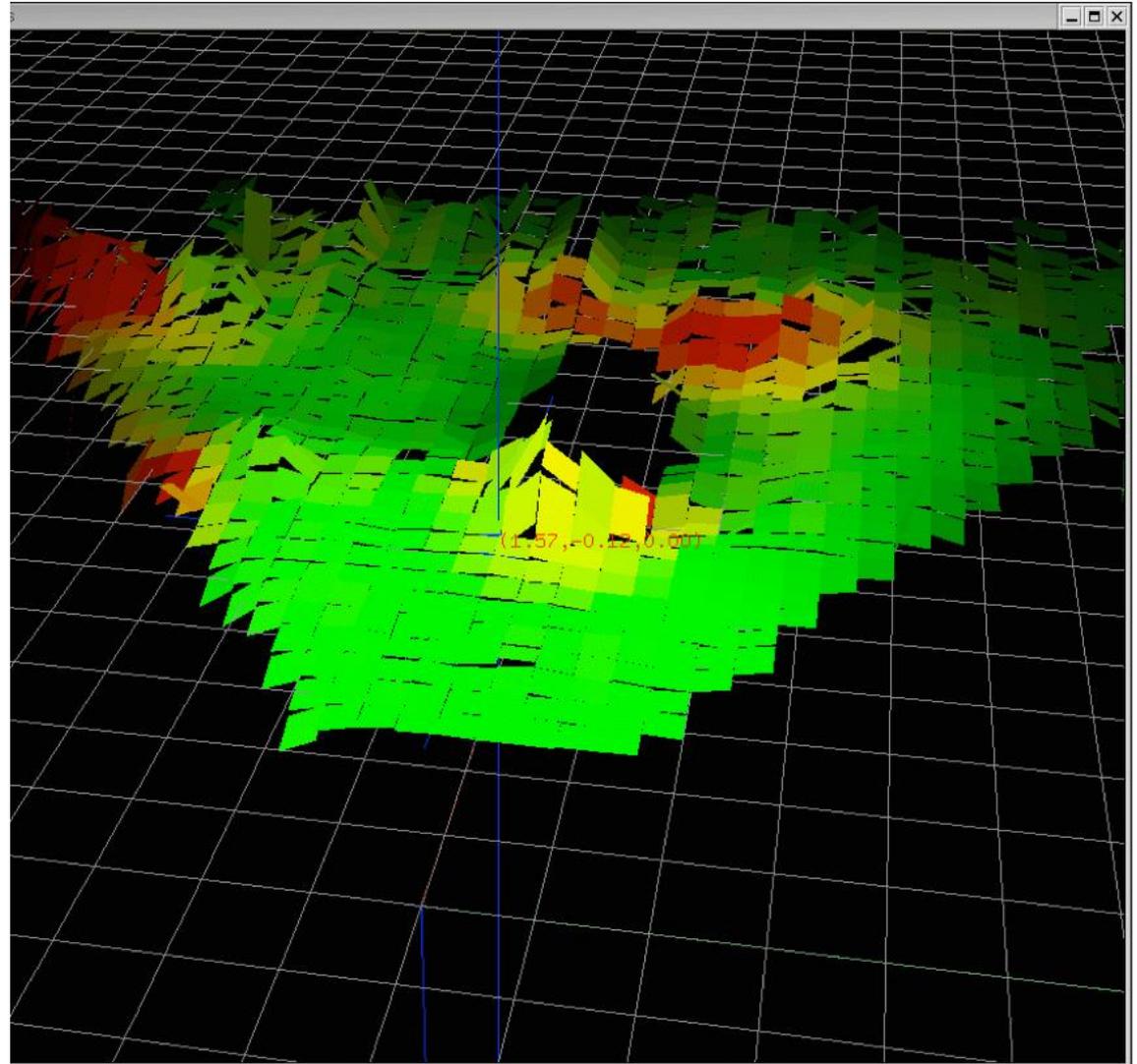
- 
- PERCEPTION
 - **TRAVERSABILITY ANALYSIS**
 - ACTION SELECTION

Point Cloud Statistics

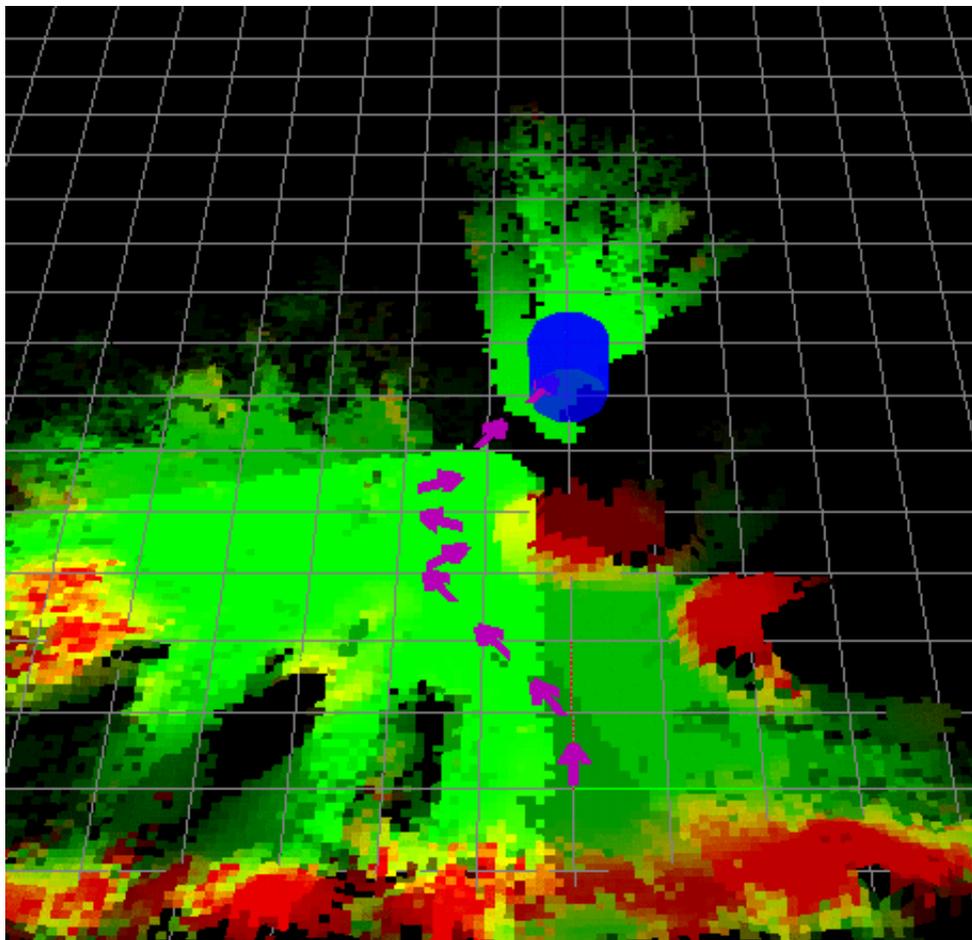


Traversability:

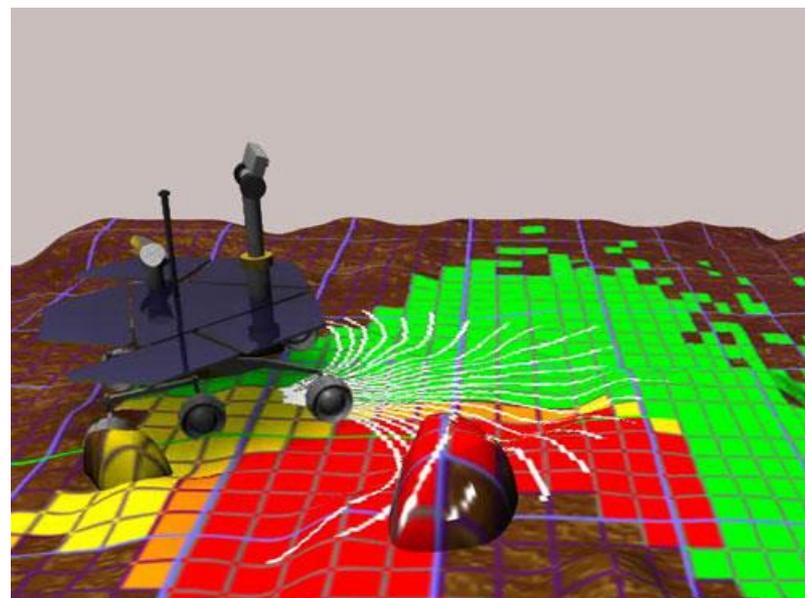
F (
terrain,
mobility,
safety,
)



Terrain Analysis and Hazard Detection

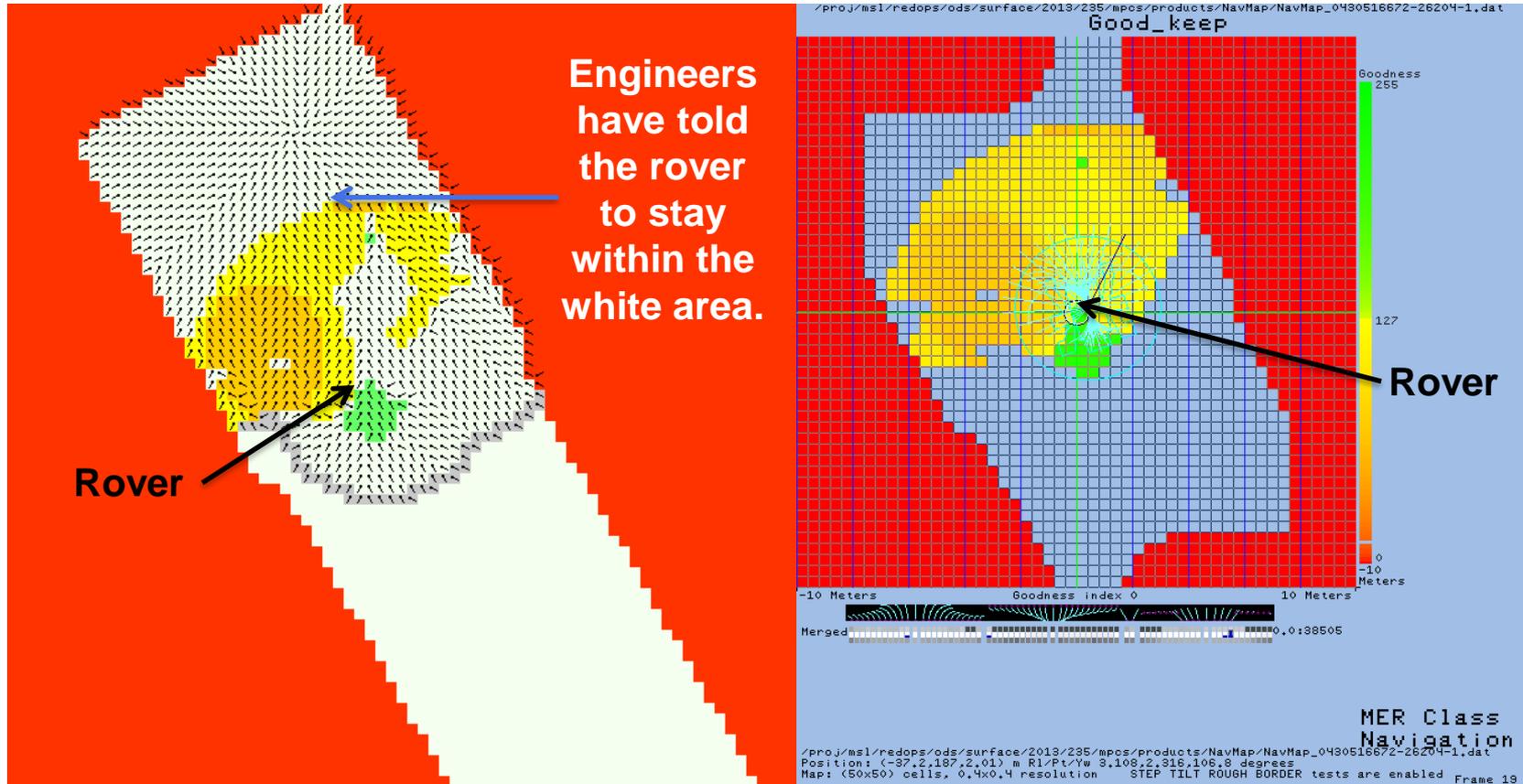


Credit: CLARAty - JPL/Carnegie Mellon – C Urmson, et al.



Credit: JPL/GESTALT navigation – Mark Maimone

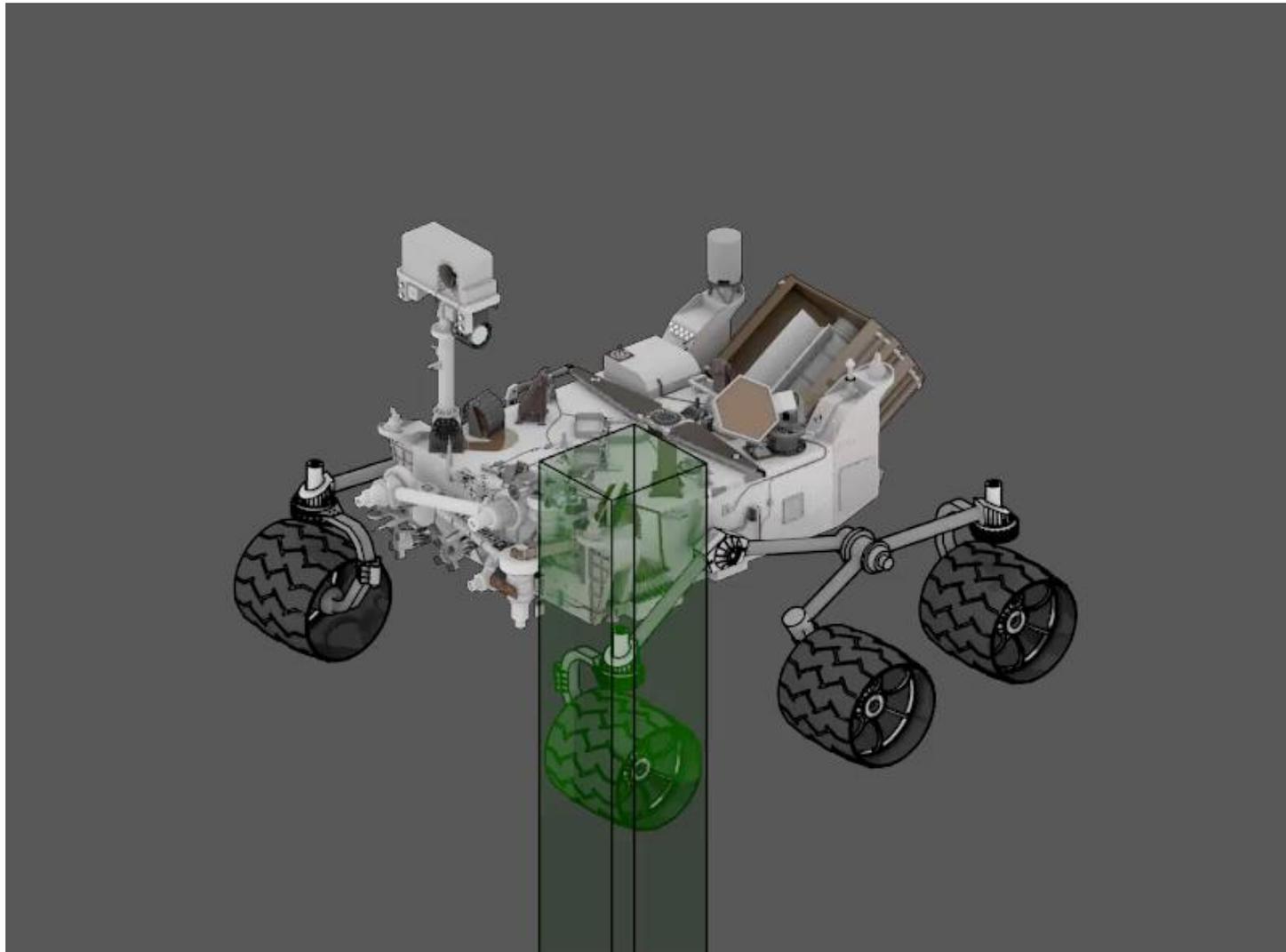
Safety: Constraining the Search with Keep-in Zones



Yellow means drive carefully, just like on Earth.

Credit: Mark Maimone

Alternative Algorithm: ENav Approx. Clearance Eval. (ACE)

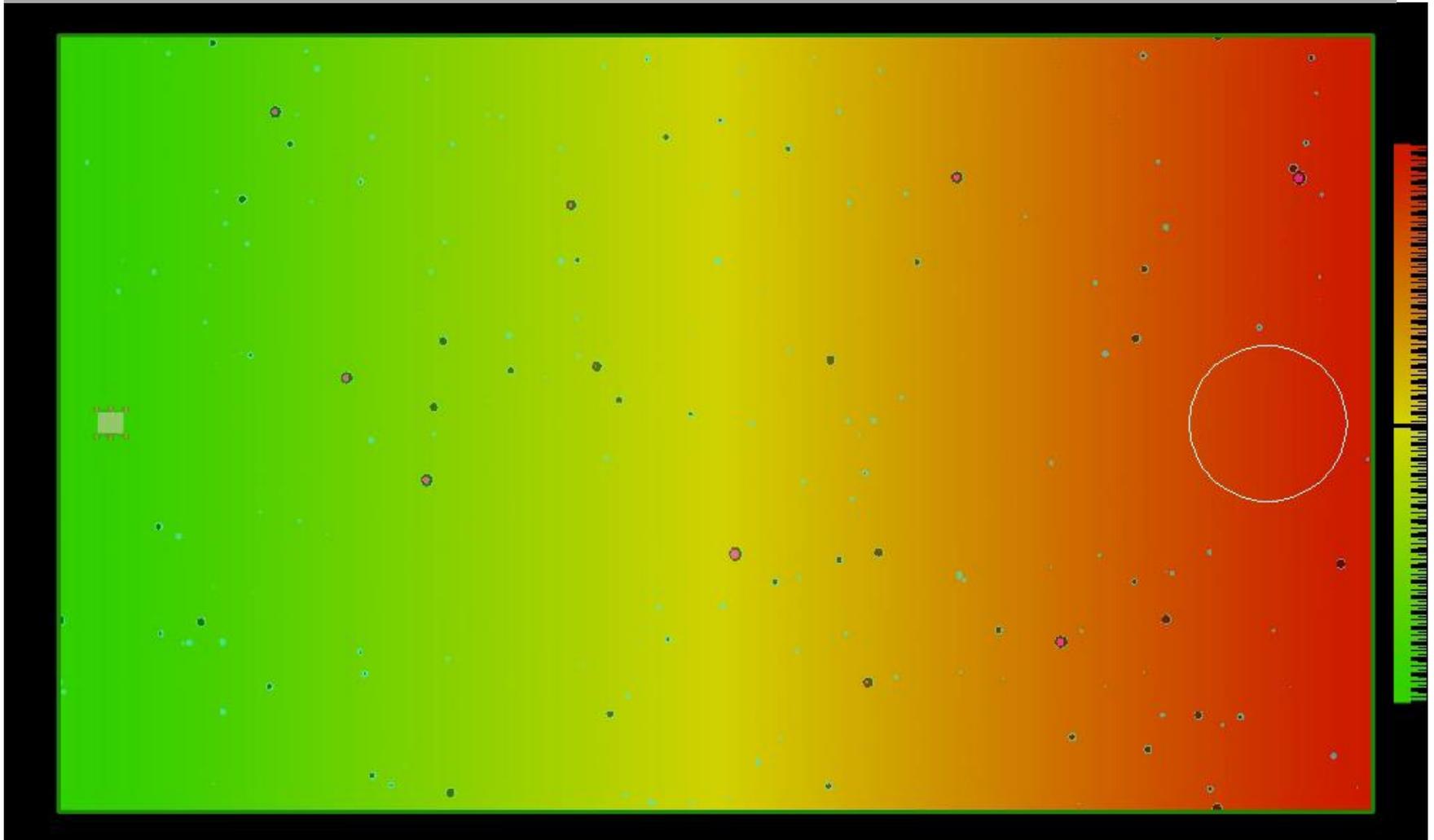


Credit: Guillaume Matheron, Olivier Toupet, Tyler Del Sesto, Hiro Ono, Michael McHenry

ENav Results



Color legend: wheel drop, low clearance, occlusion, high local tilt, high global tilt, KIZ/KOZ

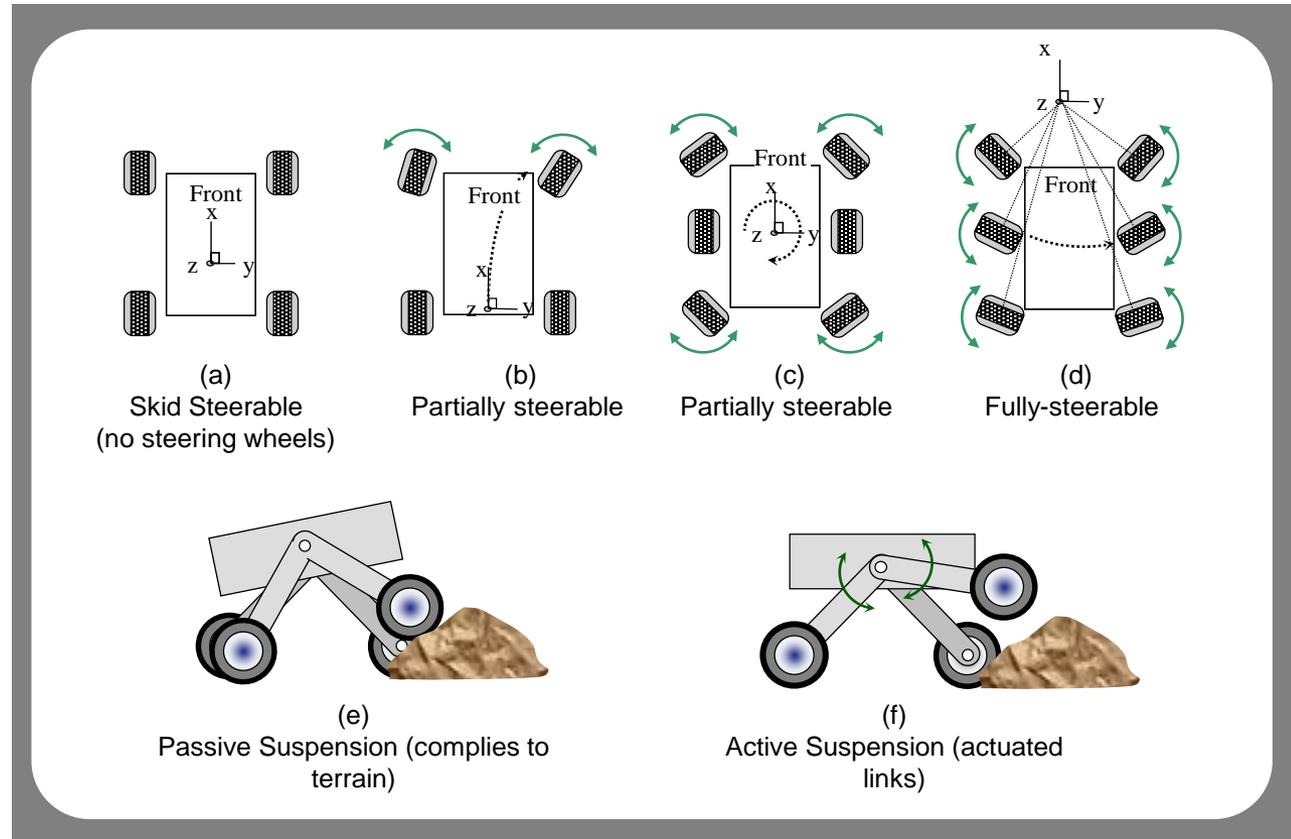
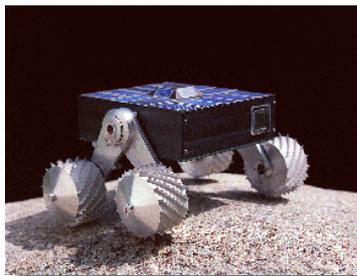


Credit: Guillaume Matheron, Olivier Toupet, Tyler Del Sesto, Hiro Ono, Michael McHenry

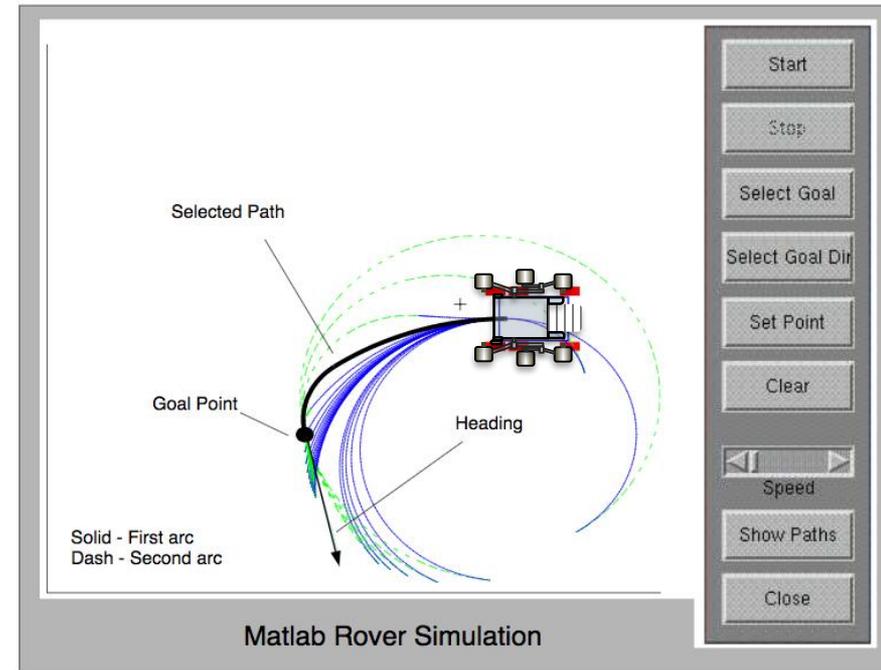
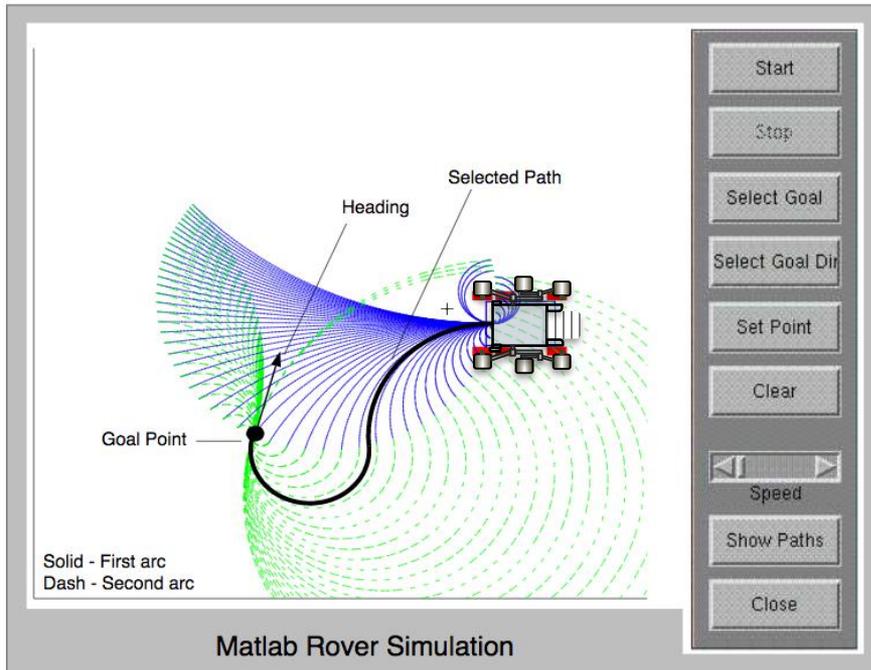
ROVER NAVIGATION

- 
- PERCEPTION
 - TRAVERSABILITY ANALYSIS
 - MOBILITY
 - ACTION SELECTION

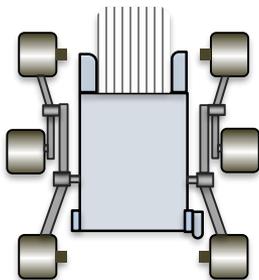
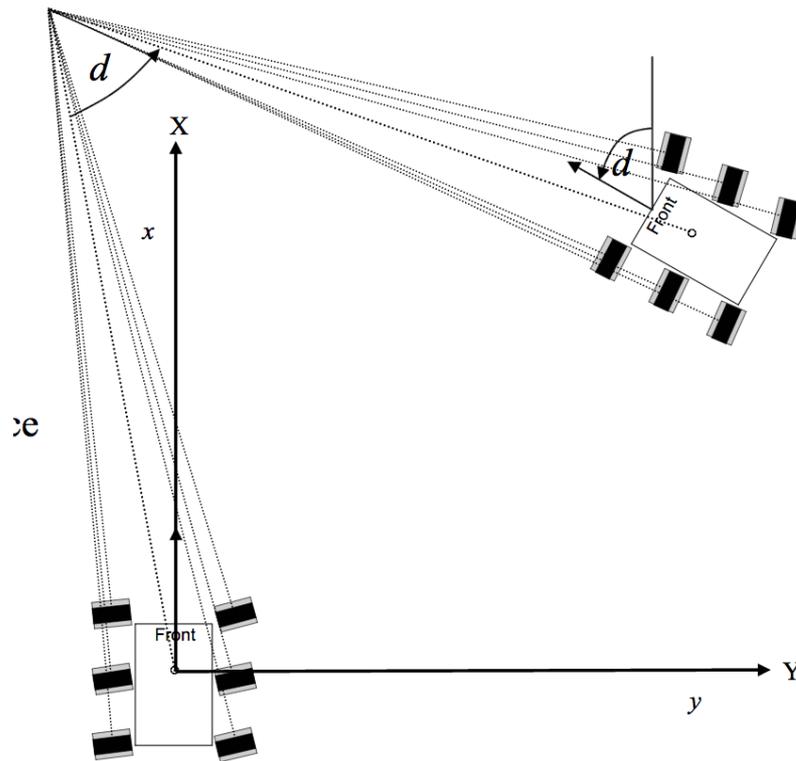
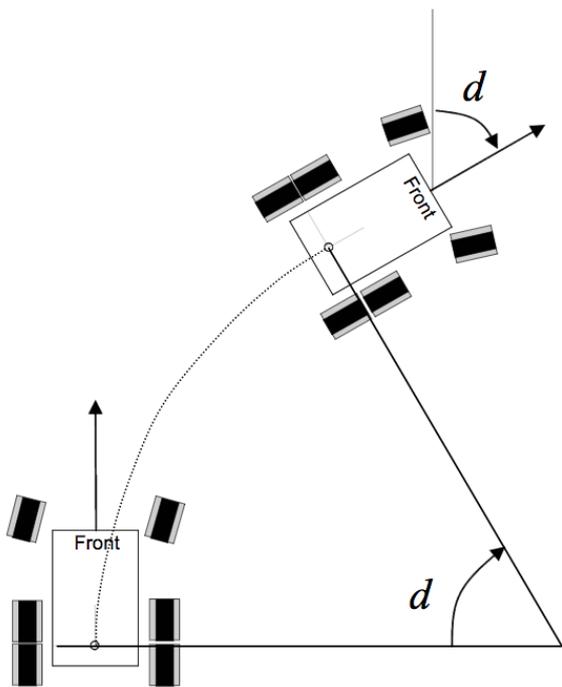
Option 1: Rover Mobility – Flat Terrain Approx.



Rover Mobility – Parallel Parking Maneuvers

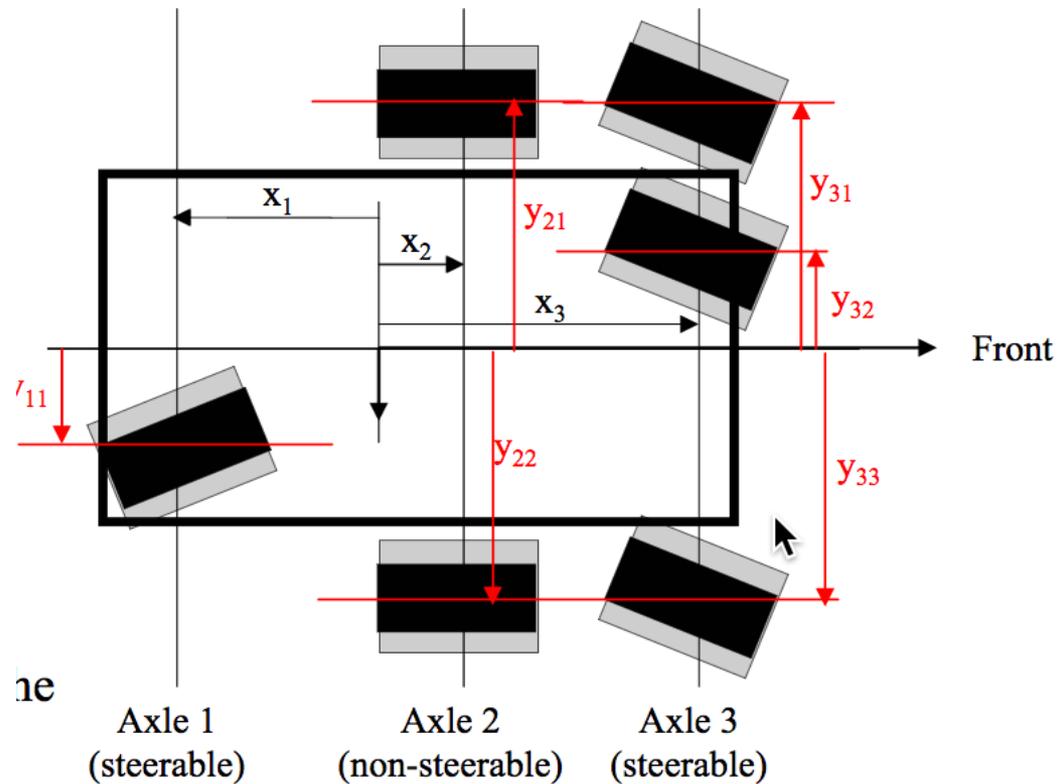


Rover Mobility – Parallel Parking Maneuvers



Credit: adapted from slide by H. Nayar

Rover Mobility – Generalized to any wheeled vehicle



Credit: adapted from slide by H. Nayar

TRAJECTORY GENERATOR CLARATY INTEGRATION FIELD EXPERIMENTS

AUGUST 2006

CARNEGIE MELLON TECHNICAL LEAD: THOMAS M. HOWARD
JPL TECHNICAL LEAD: DR. ANTONIO DIAZ-CALDERAN



Jet Propulsion Laboratory
California Institute of Technology

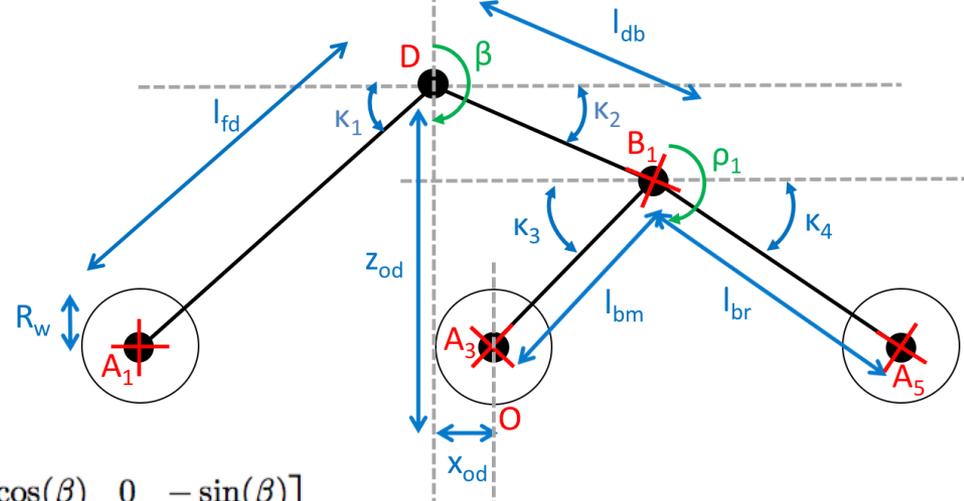


Option 2: Rocker Bogie Kinematics



- Use kinematics to relate the wheel rates to the attitude and suspension angles & rates:

$${}^{\mathcal{F}}\vec{v}_{A/I} = {}^{\mathcal{F}}\vec{v}_{B/I} + {}^{\mathcal{F}}\vec{\omega}_{R/I} \times {}^{\mathcal{F}}\vec{BA}$$



$$\eta_1 \vec{v}_{A_1} = \eta_1 R_{w_1} {}^{w_1} R_{r_{k_1}} {}^{r_{k_1}} R_{b_d} {}^{b_d} \vec{v}_{A_1}$$

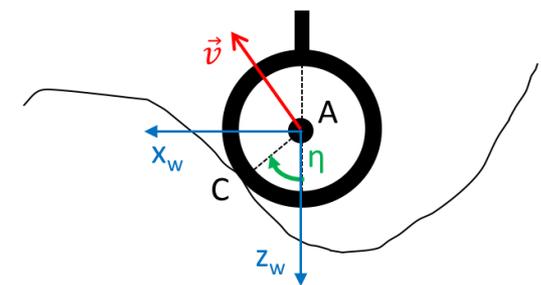
$$= \begin{bmatrix} \cos(\eta_1) & 0 & -\sin(\eta_1) \\ 0 & 1 & 0 \\ \sin(\eta_1) & 0 & \cos(\eta_1) \end{bmatrix} \begin{bmatrix} \cos(\psi_1) & \sin(\psi_1) & 0 \\ -\sin(\psi_1) & \cos(\psi_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

$$\left(\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} + \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times {}^{b_d} \vec{OD} + \begin{bmatrix} \omega_x \\ \omega_y + \dot{\beta} \\ \omega_z \end{bmatrix} \times {}^{b_d} \vec{DA}_1 \right)$$

$$\eta_1 v_{A_1}^x = R_w \omega_1^y$$

$$\omega_1^y = \dot{\theta}_1 + \zeta_1^y \text{ with } \vec{\zeta}_1 = \eta_1 R_{w_1} {}^{w_1} R_{r_{k_1}} {}^{r_{k_1}} R_{b_d} \begin{bmatrix} \omega_x \\ \omega_y + \dot{\beta} \\ \omega_z \end{bmatrix}$$

$${}^{b_d} \vec{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \sin(\phi) \cos(\theta) \\ 0 & -\sin(\phi) & \cos(\phi) \cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$



Credit: Olivier Toupet, Jeffrey Biesiadecki

Capabilities of Wheel Locomotor



- Type of maneuvers:
 - Straight line motions (fwd / bkwd)
 - Crab maneuvers
 - Arc maneuvers
 - Arc crab maneuvers
 - Rotate-in-place maneuvers (arc turn $r=0$)
- Driving Operation
 - Non-blocking drive commands
 - Multi-threaded access to the Wheel_Locomotor class – e.g. one task can use Wheel_Locomotor for driving while the other for position queries
 - Querying capabilities during all modes of operation. Examples include position updates and state queries
 - Built-in rudimentary pose estimation that assumes vehicle follows commanded motion

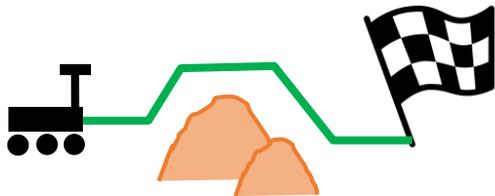
ROVER NAVIGATION

- PERCEPTION
- TRAVERSABILITY ANALYSIS
- ACTION SELECTION

Option 1: Action Selection (M2020 Enav)

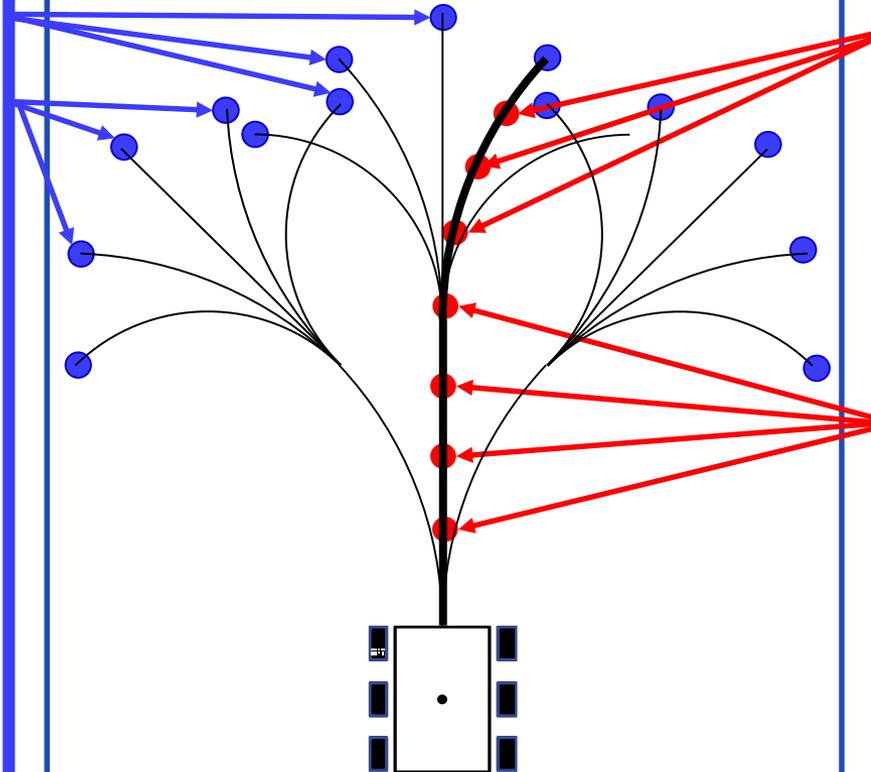


Global Planner



- Gives cost from the end of tree to goal
- Routes computed on 200m x 200m map
- 1 m resolution
- Considers slope, roughness, keep-out zones

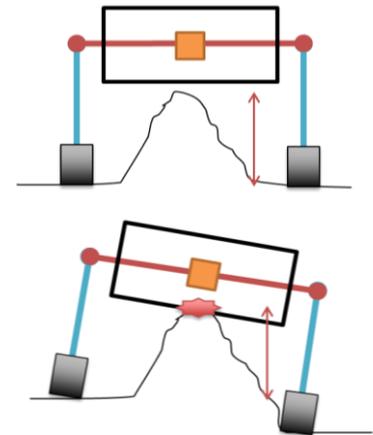
Local Planner



- Selects best path for the next 6m

ACE

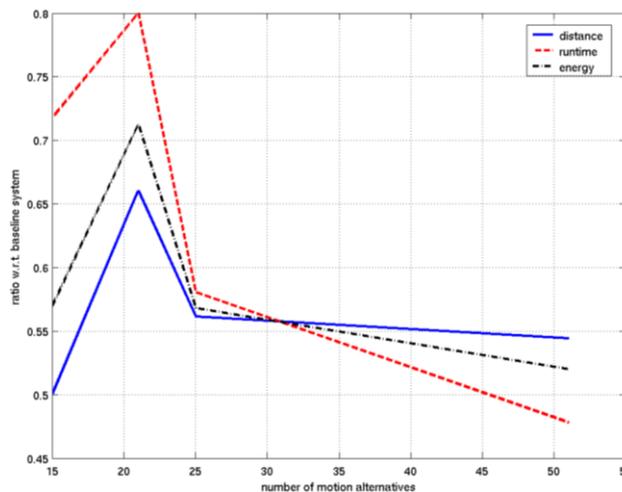
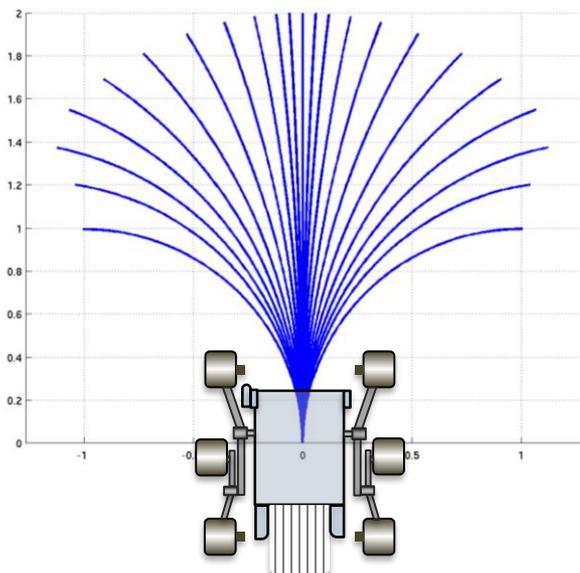
(Approx. Clearance Est.)



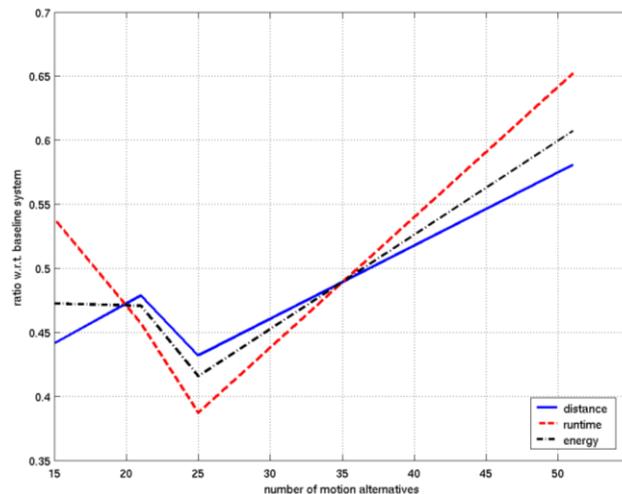
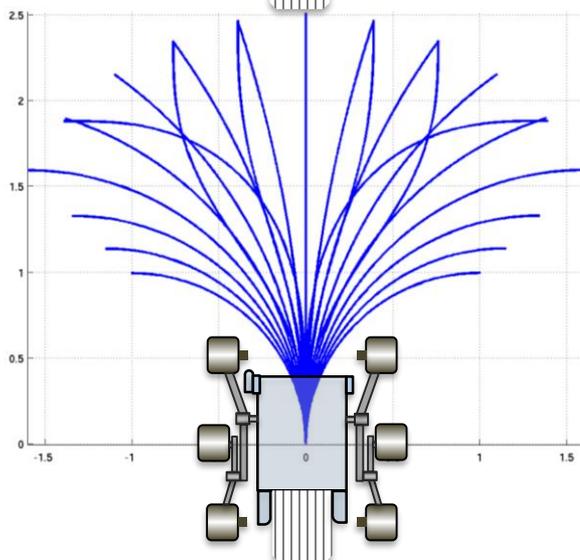
- Runs every 25cm
- Checks clearance, tilt, suspension and attitude limits, wheel drop

Credit: Olivier Toupet, Hiro Ono, Michael McHenry, Tyler Del Sesto

Option 2: More Diverse Potential Action Space



- Irregular sampling of fixed-curvature arcs
- Reduced maximum heading change



- Variable curvature paths and fixed-curvature arcs
- All three performance metrics exhibit improvement over uniform, constant arc length

Credit: Thomas Howard, Mihail Pivtoraiko

ROVER NAVIGATION



SOFTWARE ARCHITECTURE (RESEARCH)

Navigation Architecture

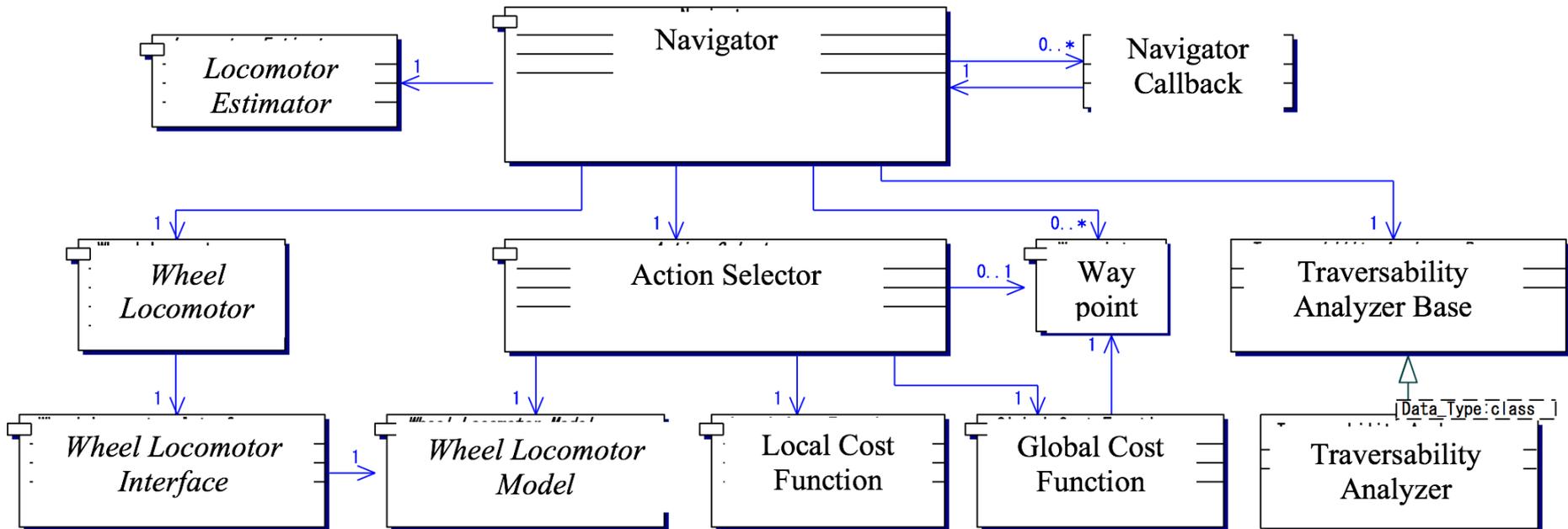


Figure 1. The structure of the navigation framework.
Italicized names are external modules.

C. Urmson, R. Simmons, I. Nenas, "A Generic Framework for Robotic Navigation," Proceedings of the IEEE Aerospace Conference, Big Sky Montana, March 2003.

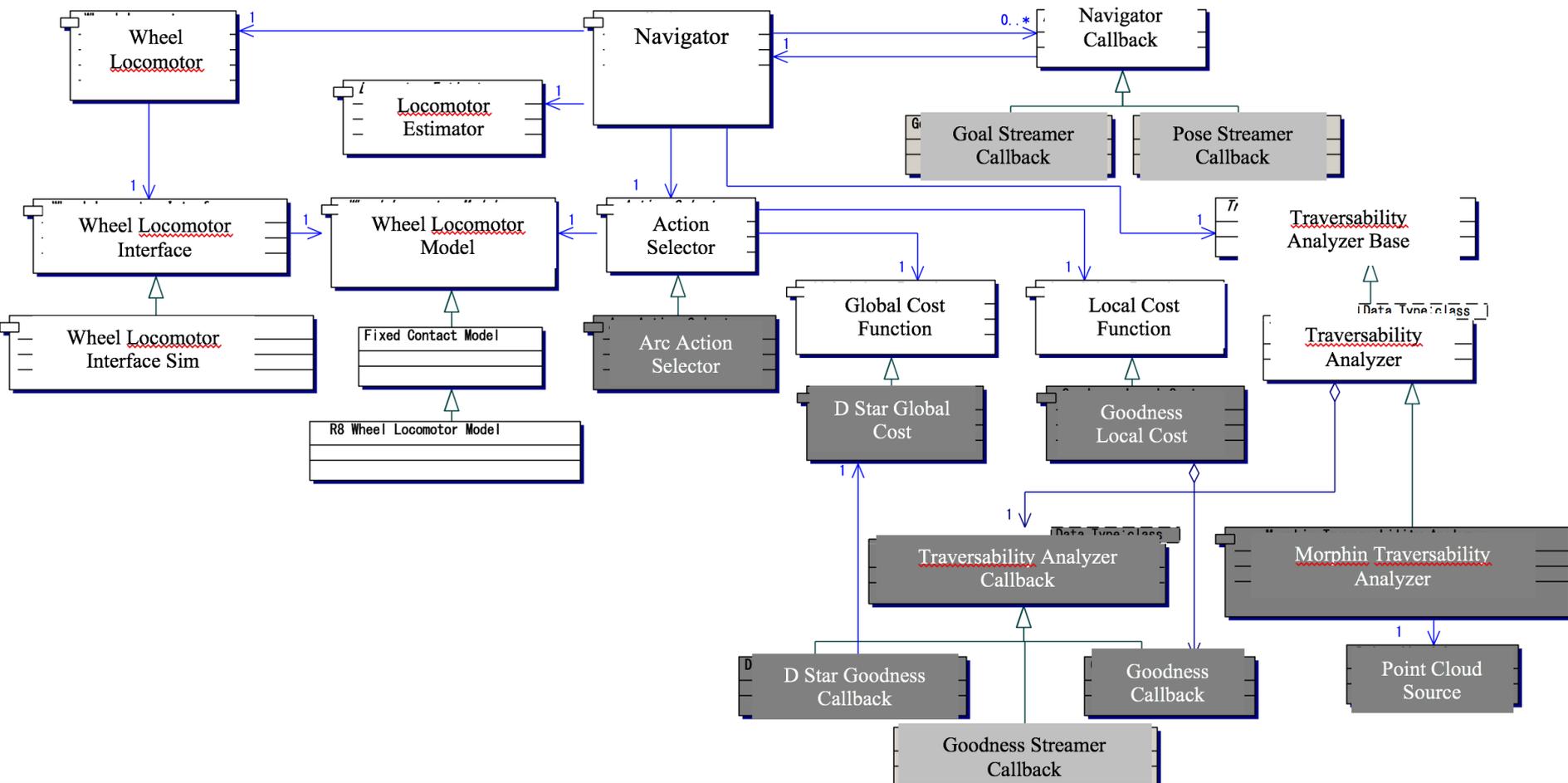
Navigation Architecture



```
repeat {
  update pose;
  execute callbacks;
  if (cur_waypoint->achieved()) {
    if (waypoint_queue->is_empty()) {
      break;
    } else {
      cur_waypoint = waypoint_queue->pop();
    }
  } else {
    traversability_analyzer->update();
    action = selector->get_best_action();
    locomotor->execute(action);
  }
}
```

...
C. Urmson, R. Simmons, I. Nenas, "A Generic Framework for Robotic Navigation," Proceedings of the IEEE Aerospace Conference, Big Sky Montana, March 2003.

Navigation Architecture



C. Urmson, R. Simmons, I. Nenas, "A Generic Framework for Robotic Navigation," Proceedings of the IEEE Aerospace Conference, Big Sky Montana, March 2003.

Athena Rover Testbed



Surface Navigation Research



Goal: 15 m straight forward

Recent Results



Credit: Venkat Rajagopalan, Jacek Sawoniewicz, Kyohei Otsu, Travis Brown, Issa Nesnas

WHAT LIES AHEAD



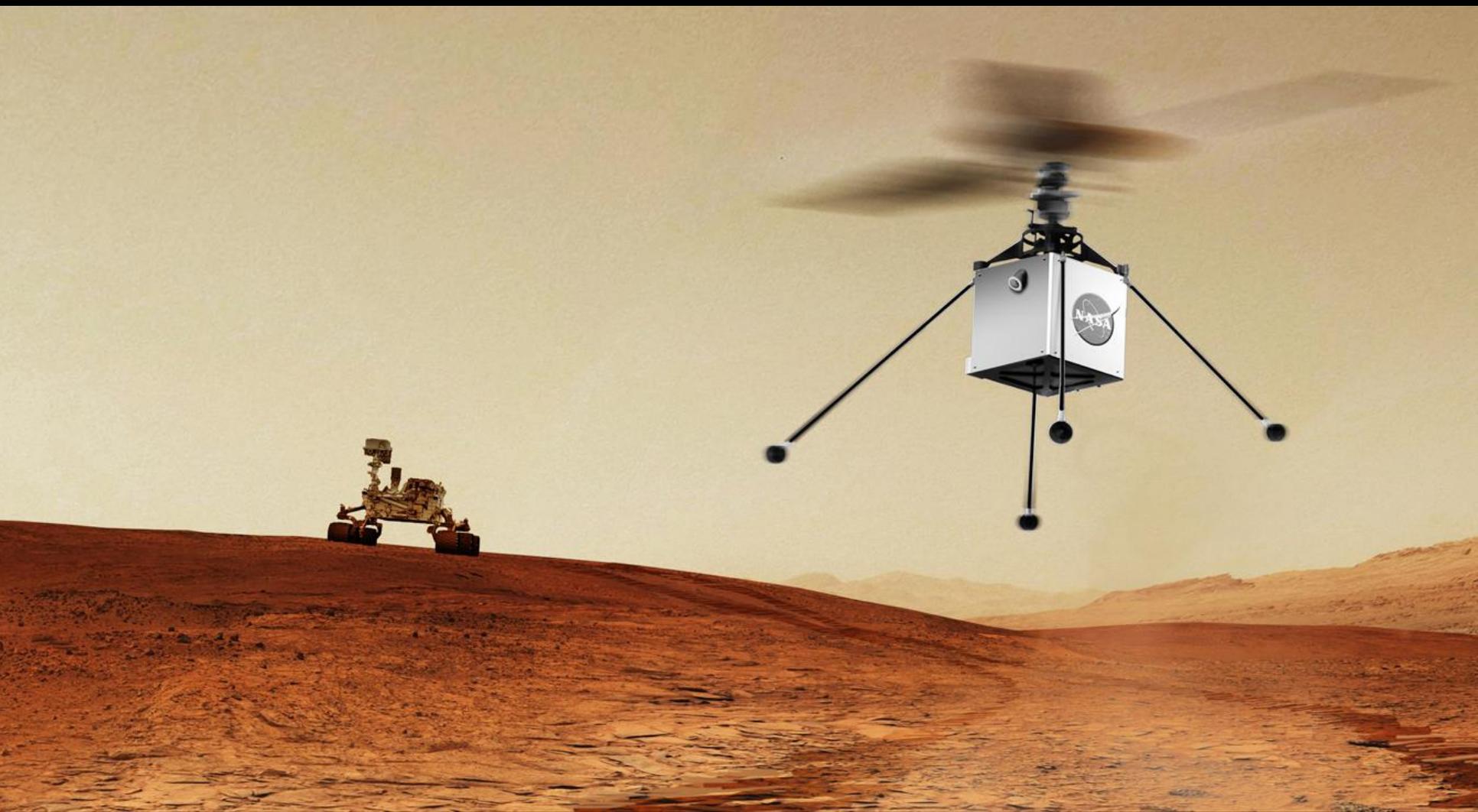
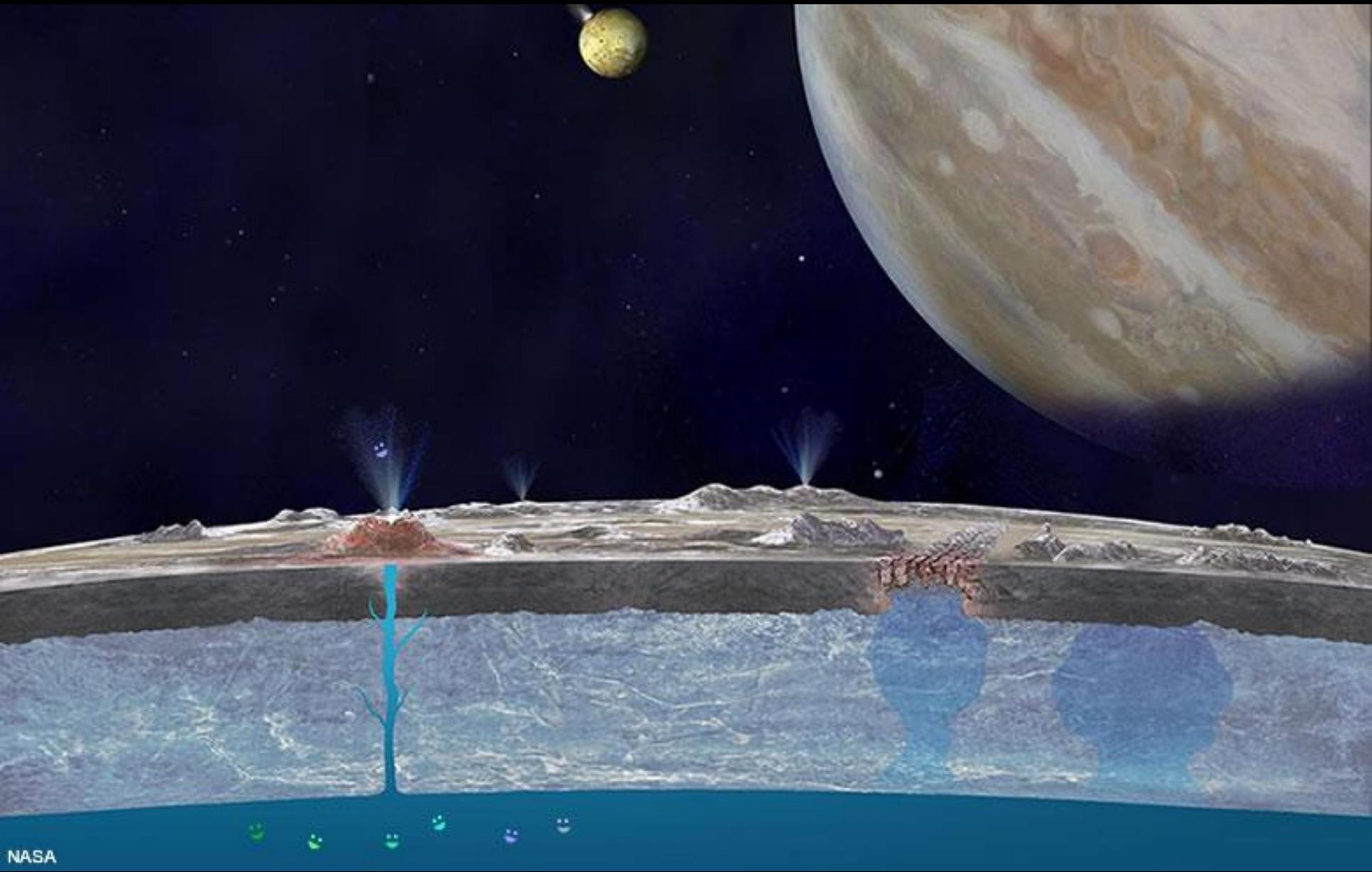


Illustration of Mars 2020 on Mars with a proposed Mars helicopter

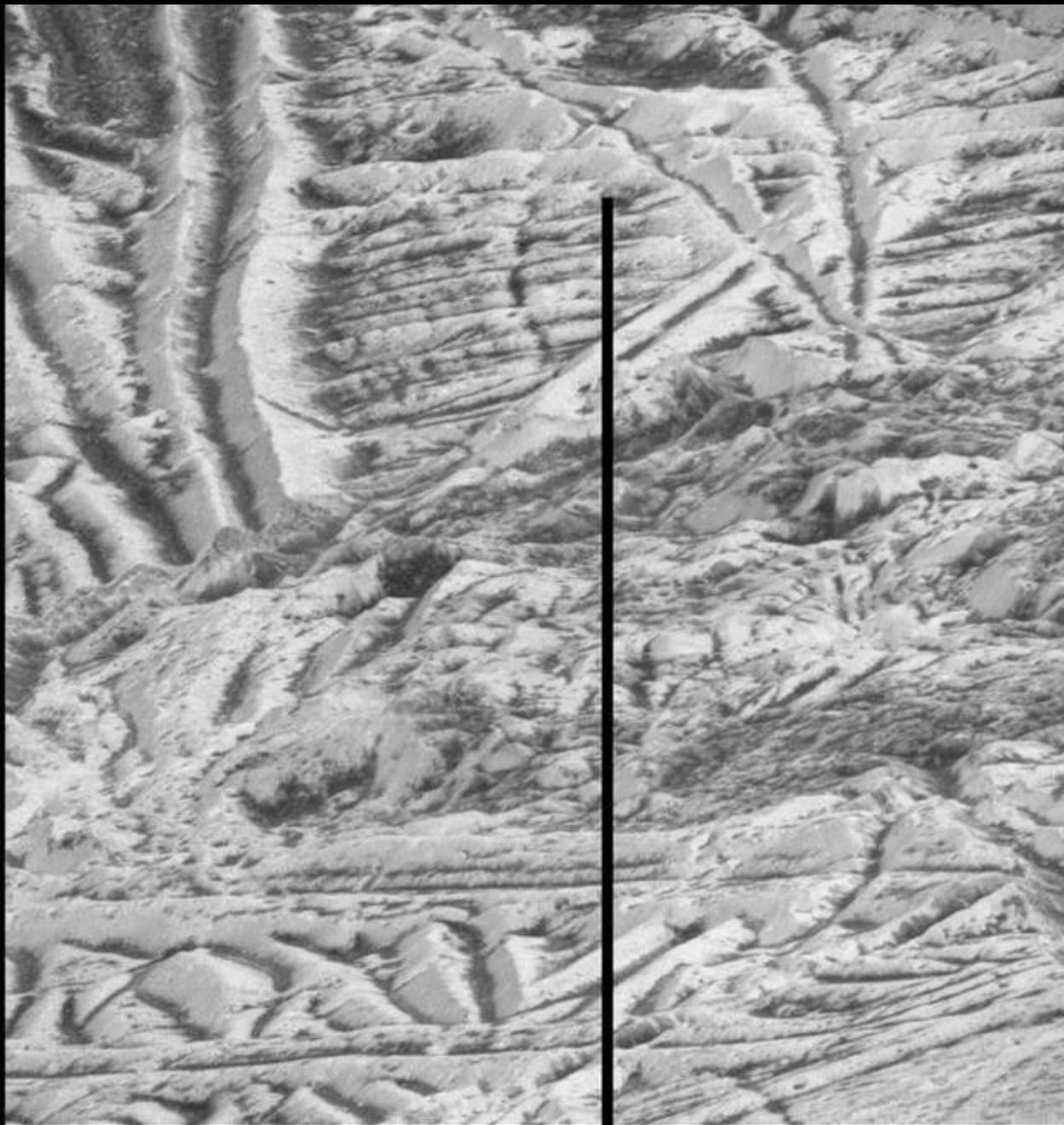
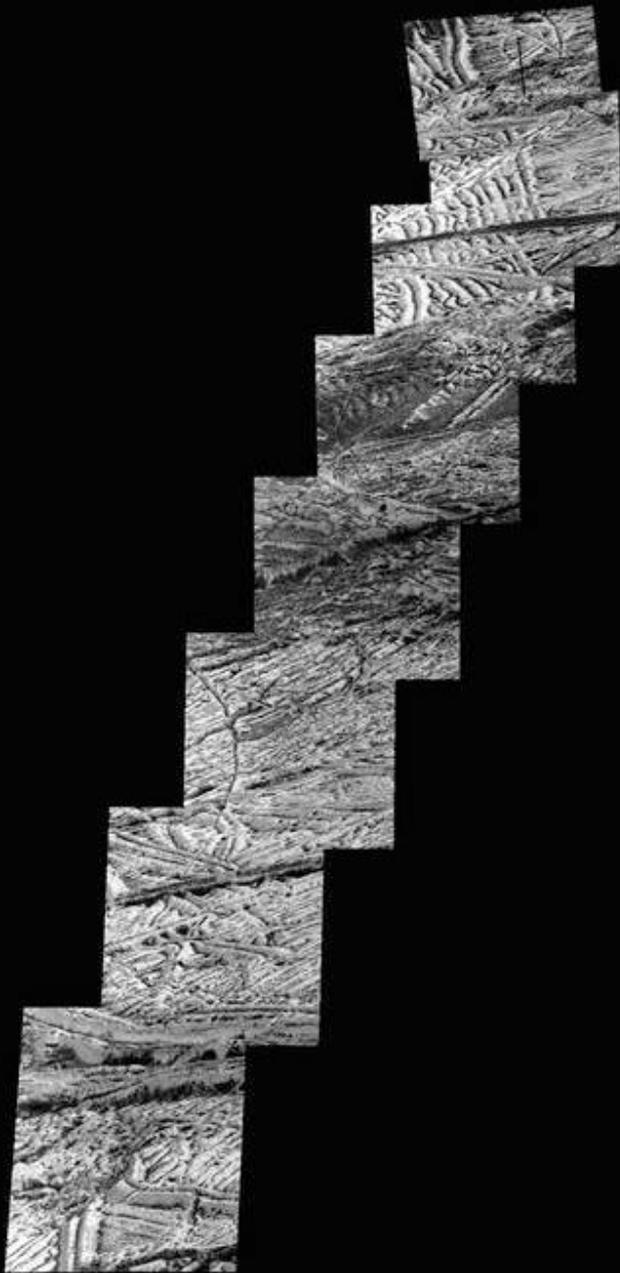


Artist's concept of a Europa Plume



NASA

Artist's concept Exploring Europa's Interior



CONCLUDING THOUGHTS



SUMMARY

Lectures Summary



- Motivated architecting robotics software with a focus on space applications
- Shared constraints and challenges from that domain
- Reviewed architectural styles across 3 decades
- Shared experiences about architectural themes
- Investigated in detail one example: *surface navigation*
- Peered into future challenges

Summary



- Developing scalable and reusable robotic software is hard because of the hardware/software heterogeneity
- Architectural styles, object-orientedness, and design patterns provide flexibility and help manage complexity
- Abstractions and polymorphism help handle heterogeneity
- Concurrency is critical to robotic applications
- Often times, robot software architecture is a hybrid of multiple styles
- Reusability comes at a cost
- Software interoperability has been demonstrated on complex platforms
- Architectural themes recurred from multiple experiences across multiple projects

Concluding Thoughts



- Advancing robotic autonomy software would continue to play a critical role in space exploration
- Environments will continue to become more challenging
- Some of the most interesting sites are currently inaccessible to state-of-the-art mobility platforms
- However, with new ideas and approaches to advance the art, we would overcome such challenges
- Prototyping and field-testing would be a critical

BACKGROUND SLIDES



SUMMARY

Generic Technologies & Algorithms



- Technologies that are generic by design should not be constrained by the software architecture & implementation
- Non-generic technologies should be accommodated on the appropriate platforms
 - Example (**Generic**): if you are working in navigation, you would not care about H/W architecture difference among different rovers
 - Example (**Specific**): if you are doing wheel/terrain interaction research, you might require specific hardware which one of the vehicles would support
- Assumptions are made explicit

Dynamics Simulation (DARTS)



Objectives

- Advanced high-fidelity, physics-based modelling and simulation for autonomous robotic platforms and environments
- Used for the development, test and operation of autonomous robotic platforms

