Complexity of robotic software development (1)

- Complex heterogeneous distributed systems
  - Complex algorithms but limited computation and memory
  - Requires data sharing mechanism (middleware)
  - Concurrent execution of processes/threads
- The robot is "embodied" in the real world
  - The world is uncertain: sensor readings, action outcomes, unexpected events, etc.
  - Physics cannot be delayed: strict time constraints
- Challenging problems
  - Trial and error
  - Method/algorithm tuning
  - Fast prototyping

Complexity of robotic software development (2)

- Different hardware devices
  - Different protocols and interfaces
- Inherently complex setting
  - e.g., visualizing sensor data requires:
    - Connect to the sensor
    - Understand and implement the protocol
    - Build a GUI
- A lot of system engineering
  - Challenging programming
  - Debugging and testing
- Heterogeneous Team of developers

Software frameworks

- A software framework is an abstraction in which common code providing generic functionalities that can be selectively overridden or specialized by user code
  - Frameworks are similar to software libraries in that they are reusable abstractions of code wrapped in a well-defined API.
  - Unlike libraries, however, the overall program's flow of control is not dictated by the caller, but by the framework. This inversion of control is the distinguishing feature of software frameworks [Wikipedia].
Modularity
- Divide-et-impera approach
- Common engineering method
- A complex problem can be often subdivided in simpler sub-problems
- Module-level tests and debugging
- Local search space for bugs
- Key features
  - Abstraction and common interfaces
  - Code reuse
  - Encapsulation
  - Decoupling

Example of module decomposition in robotics

Key features
- Abstraction and common interfaces
- Code reuse
- Encapsulation
- Decoupling

Software frameworks for robotics
- Aims
  - Promote standard design techniques
  - Code reusability (components)
  - Ready-to-use design techniques
  - Rapid prototyping
  - Concurrent engineering
- Examples
  - OROCOS (EURON project), CLARAty (NASA), OpenRTM-aist (Japanese project), YARP, MS Robotic Studio,
  - Oro, Player/Stage, MARIE, MOAST, ...
  - ROS (Willow Garage)
  - OpenRDK (DIS RoCoCo ...)  
- Main elements of a framework
  - Concurrency model
  - Information sharing model
  - Libraries and tools
  - Interoperability

Concurrency model
- Threads features
  - Are easy to write
  - Information sharing is fast
  - Less robust than processes
  - Need facilities for concurrent data access

Frameworks that use threads:
- OpenRDK, (OROCOS), (OpenRTM)
Software development in robotics

**Information sharing model**
- Data ports
- Blackboard
- Services
  - Use of third-party middleware: e.g., CORBA, ICE, OMG DDS
  - Ready-to-use
  - Different goals
  - Developing ad-hoc middleware: It's a complex task
  - Allows for application-oriented tuning

Frameworks that use data ports:
- OROCOS
- OpenRTM
- Orca
- ...

Frameworks that use a blackboard:
- OpenRDK
- MIRO

**Libraries and tools**
- Libraries reduce programming time
  - Frameworks include libraries for geometrical computations, control, filesystem utilities, operations on maps, images, etc.
  - Often external libraries are used (e.g., OpenCV, libgsl, libxml, ImageMagick, etc.)
- Tools speed-up development and debugging phases
  - Graphical tools for debugging and inspection
  - Simulators or connection to simulators
  - Logging and replaying
  - Profiling

**Interoperability**
- No "best" framework, it depends on
  - Different applications (service robotics, multi-robot teams, etc.)
  - Different focus (low-level real-time control, high-level artificial intelligence, machine learning, image processing, etc.)
  - Different people (students, researchers, control experts, etc.)
  - Different set of robot compatibility (mobile robots, robotic arms, frameworks for a single model of robot)

- Interoperability
  - The ability of heterogeneous systems to suitably exchange information, using common protocols and abstractions

**An example of frameworks comparison**
- OpenRDK
  - Real-time not supported
  - Compatibility with a large set of robots
  - Multi-threaded process
  - Module development guided/constrained
  - Pluggable inter-process communication (rdk, dds, http, file, etc.)
  - Core libraries for robotic applications
  - Small community
  - GPLv3 license
  - ...
- ROS
  - Real-time not supported
  - Compatibility with a large set of robots
  - One module = one process
  - Free module development
  - Proprietary protocol for inter-process communication
  - Large set of libraries for robotic applications
  - Large community
  - BSD license
  - ...
A short history of OpenRDK

- SPQR-RDK (first commit to the CVS repository)
  - April, 2nd 2003

- SPQR-RDK 2
  - September, 30th 2005

- OpenRDK (SourceForge)
  - February, 25th 2008

Questions

Questions?

We are on SourceForge

http://opendk.sourceforge.net