# Software Considerations for Robots in a Multi-robot System

Kris Beevers
Rensselaer Polytechnic Institute
Algorithmic Robotics Laboratory
beevek@cs.rpi.edu

April 16, 2003

# Overview

- We're making a new mobile robot platform, to be used in multi-robot systems
- Relatively powerful computing onboard; to be capable of performing moderately difficult tasks autonomously
- It needs software!
  - Operating System
  - Low-level control of hardware
  - Higher-level 'API'
  - An "architecture" (e.g. reactive, deliberative, hybrid)
  - Bonus functionality (e.g. built-in mapping, localization, etc.)

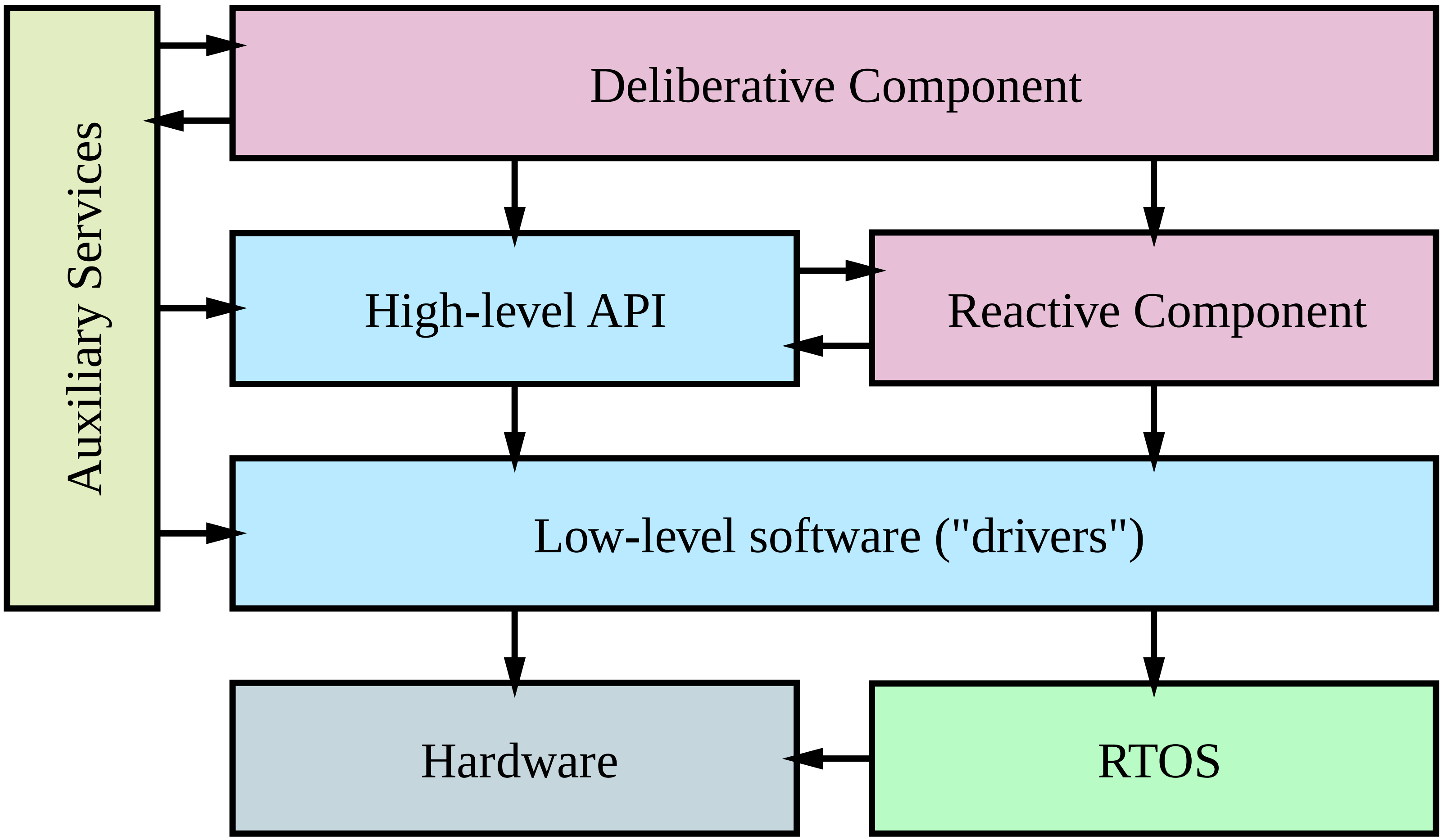
# Software Scope

- From the basics . . .
  - setting motor velocities
  - getting and reporting sonar readings
- To the not-so-basics
  - reactiveness to the environment
  - motion planning
  - task-level planning
  - mapping/localization
  - inter-robot communication and cooperation

# A Few Requirements

- A number of requirements drive our decisions; here are a few:
- Would like to be able to build in reactive "behaviors" that we can enable/disable
- Would like to be able to incorporate high-level "bonus functionality" (such as mapping) when desired, in a modular fashion
- Require extensibility:
  - new hardware devices
  - new reactive behaviors
  - new "bonus functionalities"

# An Early Proposal



# Operating Systems

- Hardware dependence: we can assume our platform will have an onboard x86, PowerPC or something approximately equivalent
- Need realtime responsiveness when talking to the hardware
  - collection of sensory data
  - motor control
- *Hard Realtime Operating Systems!*
  - offer guaranteed worst-case response times on hardware interrupts

# RTOS Commercial Alternatives

- QNX, vxWorks, TimeSys Linux, ...
- QNX, vxWorks:
  - proprietary microkernel
  - proprietary API (vxWorks at least has some POSIX compliance)
  - closed source
  - expensive!
  - but: lots of documentation and commercial support

# RTOS Commercial Alternatives (cont.)

- TimeSys Linux
  - modified Linux kernel with preemption
  - a number of proprietary kernel modules for performance, features, etc.
  - much better performance than default Linux, but not close to microkernel performance
  - free version available, missing features/support

# RTOS Free Alternatives

- Patched Linux kernel, RTLinux, RTAI

- Kernel patched for preemption—1-5 ms response time but not really a "hard" RTOS

- RTLinux, RTAI
  - separate high-performance microkernel that *runs the Linux kernel as its lowest priority process*
  - realtime code runs in the microkernel with an interrupt response time typically less than 15 $\mu$s on x86 (better on PowerPC)
  - communication with user-space applications (running in Linux) via FIFO buffers, semaphores, POSIX signals, etc.

# RTOS Free Alternatives (cont.)

- RTLinux is actually commercial but a free version with almost all of the functionality is available (support and some nice development tools can be bought if we want them)

- FSMLabs (RTLinux makers) also offer RTCore/BSD

- Why not tinker with all of the free ones and see which we like best?
  - TimeSys (can get commercial version/support)
  - RTLinux (can get commercial version/support)
  - RTAI

# Low-Level Control Of Hardware

- E.g. set motor velocity or get current sonar range
- Think of this as mostly being ‘drivers’ for devices
- What is/isn’t going to be done in hardware?
- This ‘layer’ will make use of RTOS: this makes it hard to debug, so keep it simple!
- It probably doesn’t matter *too* much how we implement this layer since it will rarely be seen by end users (except maybe by someone working on completely reactive stuff)

# Higher-Level API

- Provide a higher-level abstraction for controlling the hardware
- For example: methods that take a path and follow it
- But remember, "controlling hardware" doesn't just mean moving the robot around!
  - controlling sensors/collecting data (especially things like a camera, where there is definitely room for abstraction between direct control and the user-space application)
  - communication with other robots, with computers, etc.
- Provide functionality here that is independent of "architecture"
  - once we get to this stage, are we already defining the architecture?

## We Need An Architecture!

- Typically hear about reactive, deliberative, hybrid architectures
- Reactive architecture might act on a fairly low level (might want ability to run realtime code!)
  - hybrid architectures might demand this too!
- How much functionality should our architecture provide?
  - "go forward 5m at 0.5 m/s"
  - "go to $(x, y)$"
  - "go to $(x, y)$ without hitting anything"
  - "go get me a Coke"
- Probably want all of these, though we can get our own Cokes for now

## My Vote: Simple Hybrid Architecture

- Benefits of both reactive and deliberative architectures
- Keep It Simple Stupid: for both developers and end users
- Lots of crazy examples of architectures with "radical new ideas" but none seem to provide much improvement over a broad range of applications
- We can add bonus functionality (services) in a modular fashion *on demand* (like localization, mapping, vision stuff, etc.)

## Other Architectures to Look At: AuRA

- From R. Arkin, Georgia Institute of Technology
- Hybrid architecture
- Two "separate" systems (reactive/deliberative) that interface to each other
- Makes use of *a priori* information as well as dynamically-acquired data

# Other Architectures to Look At: Saphira

- From K. Konolige, Stanford Research Institute
- "Local Perceptual Space": geometric representation of space around robot
  - incorporates various levels of interpretation of sensor information
  - occupancy grids, analytic representations (such as linear surfaces), semantic descriptions ("door" or "wall")
- "Perception routines" and "action routines," connected by the "Procedural Reasoning System"
  - incorporates mapping, localization, topological planner, etc.

# Multi-robot Architectures

- We need to think about how multi-robot coordination fits into our plans!
- A few to look at:
  - L. Parker: ALLIANCE
  - R. Simmons: market-based cooperation
  - GRASP Lab: tightly-coupled cooperation
  - many others

## “Bonus Functionality”

- I.e. “services” to extend our architecture
- Mapping
- Localization
- Vision stuff
- Others have done people-tracking, gesture recognition, voice recognition, etc.
  - mostly stuff we probably don’t care about for now
  - but our system should still allow stuff like this to be incorporated!

# Mapping

- The standard approach: occupancy grids
  - still works! but lots of improvements have been developed
  - Dempster-Shafer model
  - K. Konolige: improvements to occupancy grids in specular/realtime environments
  - A. Zelinsky: 'certainty grid quadtree'
- Topological mapping
- 3D mapping (S. Thrun has lots of neat stuff)
- Multi-robot mapping

# Localization

- Kalman filtering
- Particle filtering (Monte-Carlo localization)
- Collaborative localization

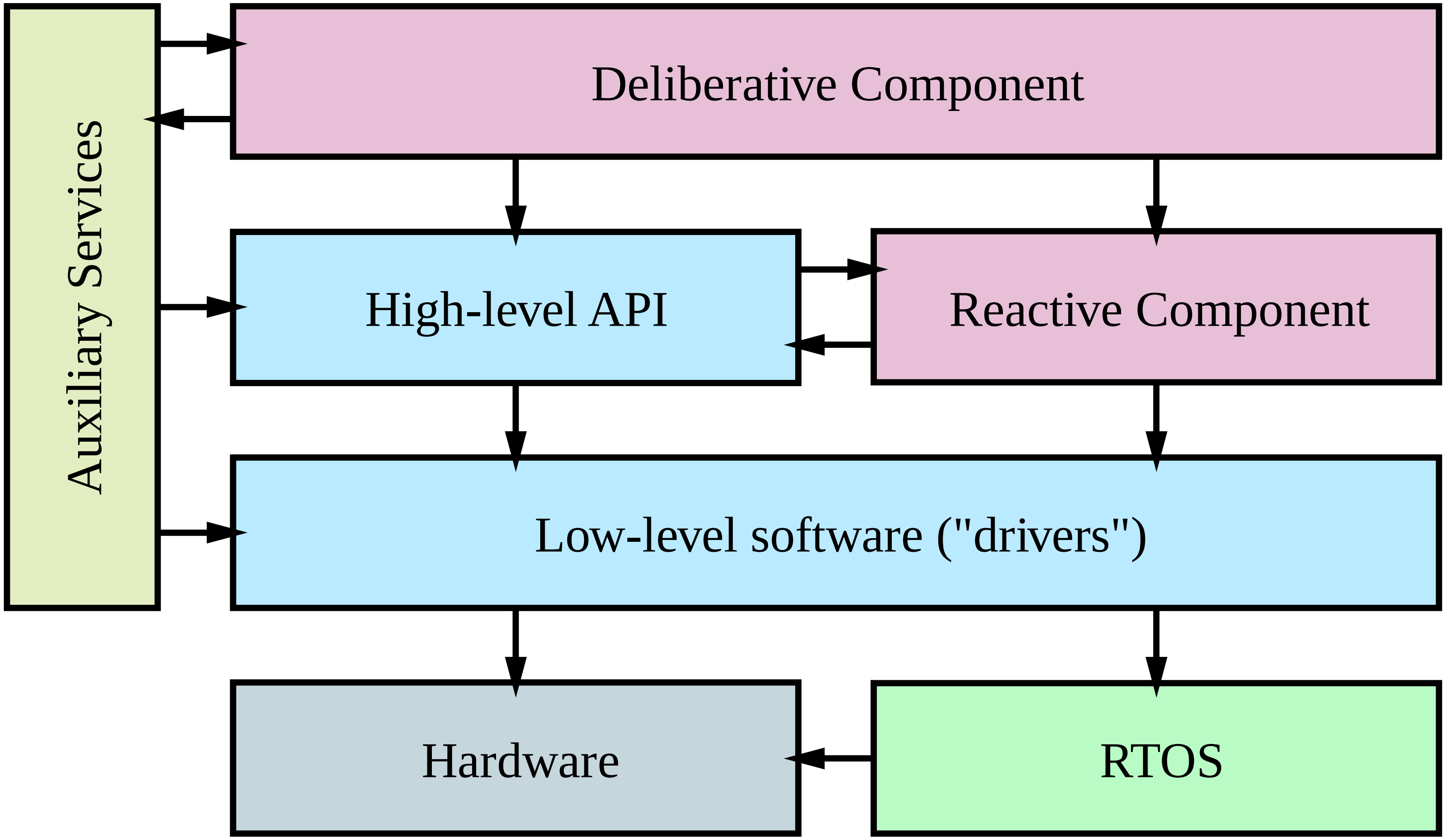
## Concurrent Mapping and Localization!

- Thrun, Fox, Burgard: alternate mapping and localization steps to improve map and location estimates

- J. Leonard, H. Feder: "decoupled stochastic mapping" (linear scaling of memory requirements with size of area being explored)

- Lots of others!

## The Early Proposal Again



# Where To Go From Here

- Need to make some choices
  - operating system
  - languages (how much C++?)
  - more concrete architecture decisions

- Get down to specifics
  - design documents
  - interface specifications
  - time estimates
  - more fun stuff like that!

# References

[1] R.C. Arkin and D. MacKenzie. Planning to behave: A hybrid deliberative/reactive control architecture for mobile manipulation. In *1994 International Symposium on Robotics and Manufacturing*, pages 5–12, Maui, Hawaii, August 1994.

[2] K. Konolige. Improved occupancy grids for map building. *Autonomous Robots*, 4(4), December 1997.

[3] K. Konolige, K.L. Myers, E.H. Ruspini, and A. Saffiotti. The Saphira architecture: A design for autonomy. *Journal of experimental & theoretical artificial intelligence: JETAI*, 9(1):215–235, 1997.

[4] J.J. Leonard and H.J.S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In J. Hollerbach and

D. Koditschek, editors, *International Symposium on Robotics Research*, pages 169–176, 1999.

[5] H.P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the 1985 IEEE International Conference on Robotics & Automation*, pages 116–121, 1985.

[6] D. Pagac, E.M. Nebot, and H. Durrant-Whyte. An evidential approach to map-building for autonomous vehicles. *IEEE Transactions on Robotics & Automation*, 14:623–629, 1998.

[7] L.E. Parker. ALLIANCE: an architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics & Automation*, 14(2):220–240, 1998.

[8] R. Simmons, T. Smith, M.B. Dias, D. Goldberg, D. Hershberger,

A. Stentz, and R. Zlot. A layered architecture for coordination of multiple robots. In A. Schultz and L. Parker, editors, *Multi-Robot Systems: From Swarms to Intelligent Automata*. Kluwer, 2002.

[9] T.G. Sugar and V. Kumar. Multiple cooperating mobile manipulators. In *Proceedings of the 1999 IEEE International Conference on Robotics & Automation*, pages 1538–1543, Detroit, Michigan, 1999.

[10] S. Thrun, D. Fox, and W. Burgard. Probabilistic mapping of an environment by a mobile robot. In *Proceedings of the 1998 IEEE International Conference on Robotics & Automation*, pages 1546–1551, 1998.

[11] A. Zelinsky. A mobile robot exploration algorithm. *IEEE Transactions on Robotics & Automation*, 8(6):707–717, 1992.