Event-based Systems with ROS: Examples from the STAIR Project

Morgan Quigley
Stanford University

Joint work with:
Stanford: Blake Carpenter, Adam Coates, Quoc Le, Ellen Klingbeil, Andrew Ng, **many others**
Willow Garage: Eric Berger, Ken Conley, Brian Gerkey, **many others**

Motivation

- Personal robotics: a general-purpose robot in every home and workplace
- Long-standing AI dream









Overview: STAIR Project

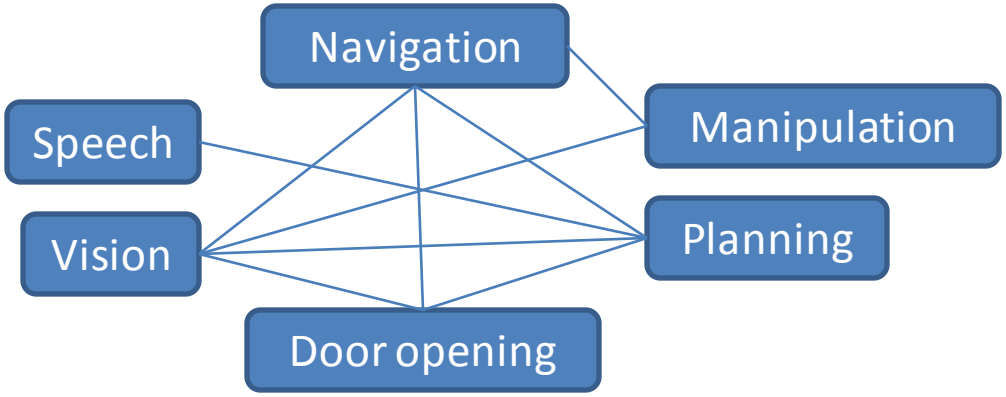
- "STAIR, please fetch the stapler from my office"
  - Speech recognition
  - Navigation: driving, doors, elevators
  - Vision: target objects, grasp points
  - Manipulation
- One team for each major component
- Components need to work seamlessly on robot
- How to best integrate them?

Experiments: 2006

- Framework with static connections, synchronous top-level executive
- 5 machines
- ~25 modules
- Explicit links
- Hard to modify

Dilemma

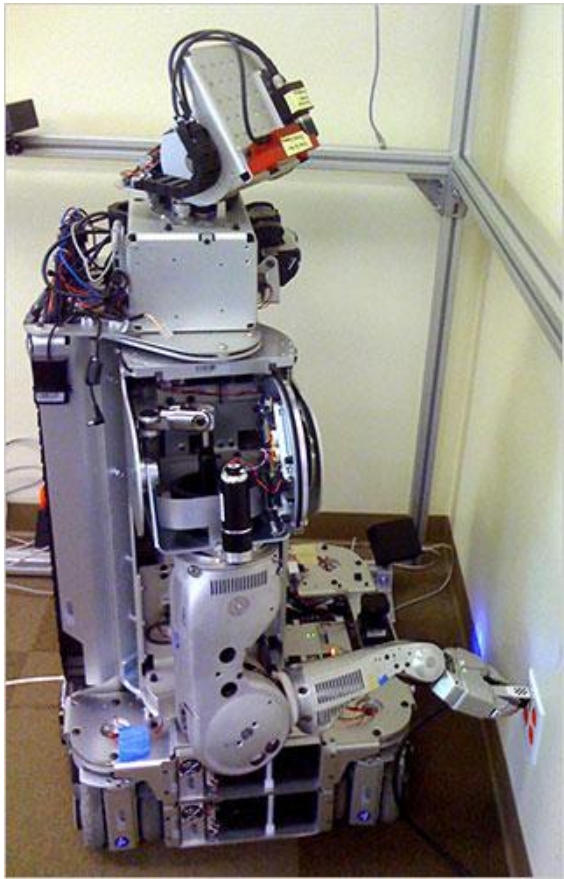
- More components can allow more applications



- More components can lead to nasty bugs and versioning nightmares

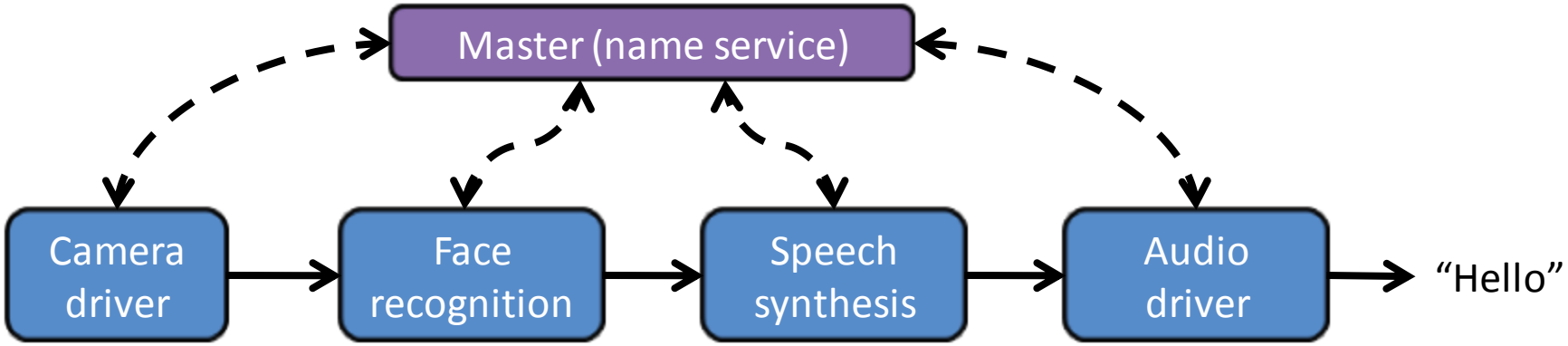
ROS: a Robot Operating System

- A framework for robot software:
  - finding, getting, writing, debugging, running
- UNIX-inspired
  - command-line friendly
  - many small tools
  - cross-language, cross-platform
  - fully open-source (BSD)
- Efficient: runs entire robot
- Streams vs. Events vs. Data Flow?



ROS: high-level

- Peer-to-peer
  - small programs connect to each other at runtime
  - “master” (registrar) node provides name service
- Runtime system: graph analogy
  - processes = nodes
  - P2P connections = directed edges



ROS: high-level

- Minimalist interface definition language (IDL)
- Native message objects generated from IDL
- Serialization, deserialization, helpers, etc.
- Actual IDL files:

JointState.msg	MapMetaData.msg	OccupancyGrid.msg

`Header header`
`string[] name`
`float64[] position`
`float64[] velocity`
`float64[] effort`
	`time map_load_time`
	`float32 resolution`
	`Uint32 width`
	`Uint32 height`
	`Pose origin`
		`MapMetaData info`
		`Byte[] data`

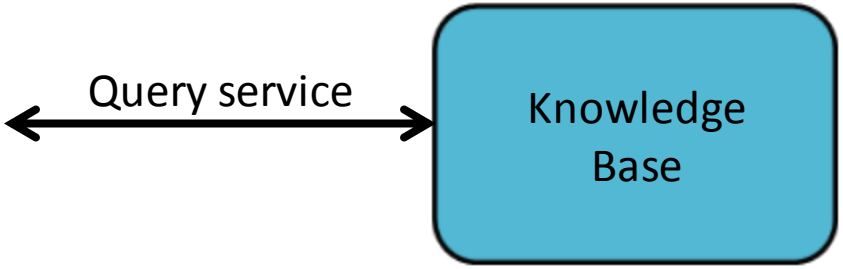
241 lines of C++ 256 lines of Python 192 lines of LISP	150 lines of C++ 132 lines of Python 112 lines of LISP	191 lines of C++ 218 lines of Python 71 lines of LISP

Publish/Subscribe

- Connections of strongly-typed topics, not nodes
- Nodes don't know/care about their peers
- Dynamic:
  - Anyone can publish at any time to any topic
  - Anyone can subscribe at any time to any topic
- ROS manages the plumbing (sockets, etc.)

Services

- Alternative communications model: RPC
- Can simplify code in some cases
- Can create bottlenecks in some cases
- We prefer publish/subscribe, but don't want to be overly dogmatic
- Use case: knowledge base query



Command-line Tools

- Debugging is a huge part of building robots
- Our opinion: command-line tools are ideal
  - Simple scripts can build more complex tools
  - Easy to run on headless machines, small overhead
- Samples:

```
rostopic list
rostopic echo TOPIC_NAME[S]
rosrecord TOPIC_NAME[S]
rosplay RECORDING_FILE
rosnode info NODE_NAME
roswtf
```

Command-line ``Remapping''

- Topic and service names are hard-coded
- Makes source code easy to read
- Optional: override names on command line
- OLD_TOPIC := NEW_TOPIC
- Allows configuration without recompiling

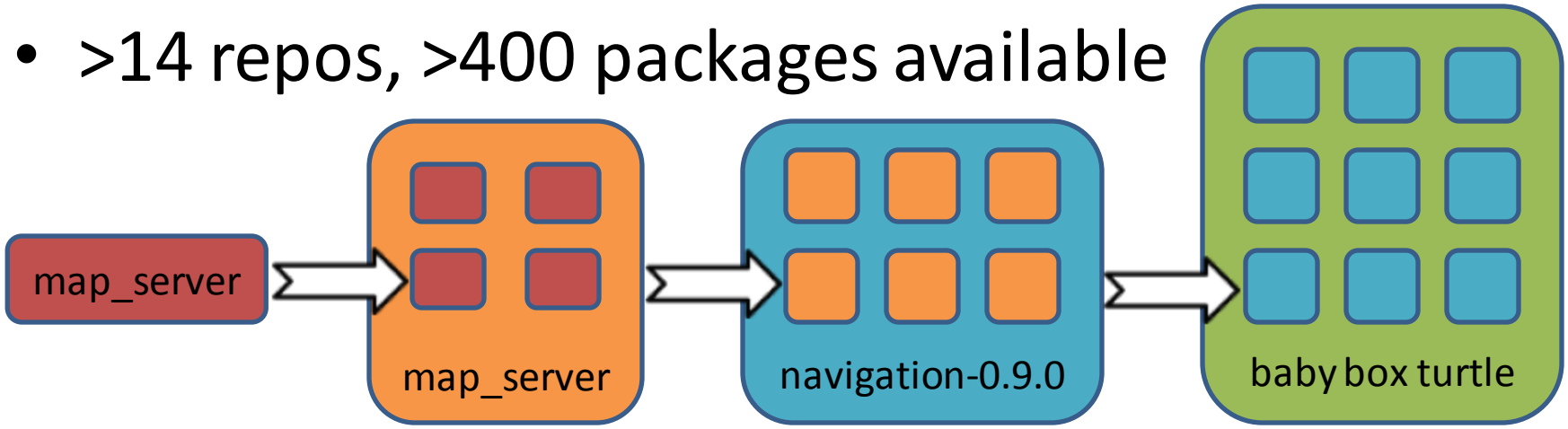
```
./hokuyo_node   scan:=base_scan
./hokuyo_node   scan:=tilt_scan
```

Launch Files

- Command-line execution is great for debugging
- Become tiresome once the system works
- Launch files: XML to automate startup/teardown
- Run many programs from a single shell
- Kill them all with one Ctrl-C
- Typically, launch files for drivers, low-level nodes, high-level nodes, and work in progress
- Easily create unit (or small-group) test suites

Code Organization

- Package: build system unit
  - Just directories in repositories; little structure
  - Recursive build tool: **rosmake**
- Stacks: groups of packages
- Distributions: collections of stacks
- >14 repos, >400 packages available



Live Demo: 2-d Navigation Simulator

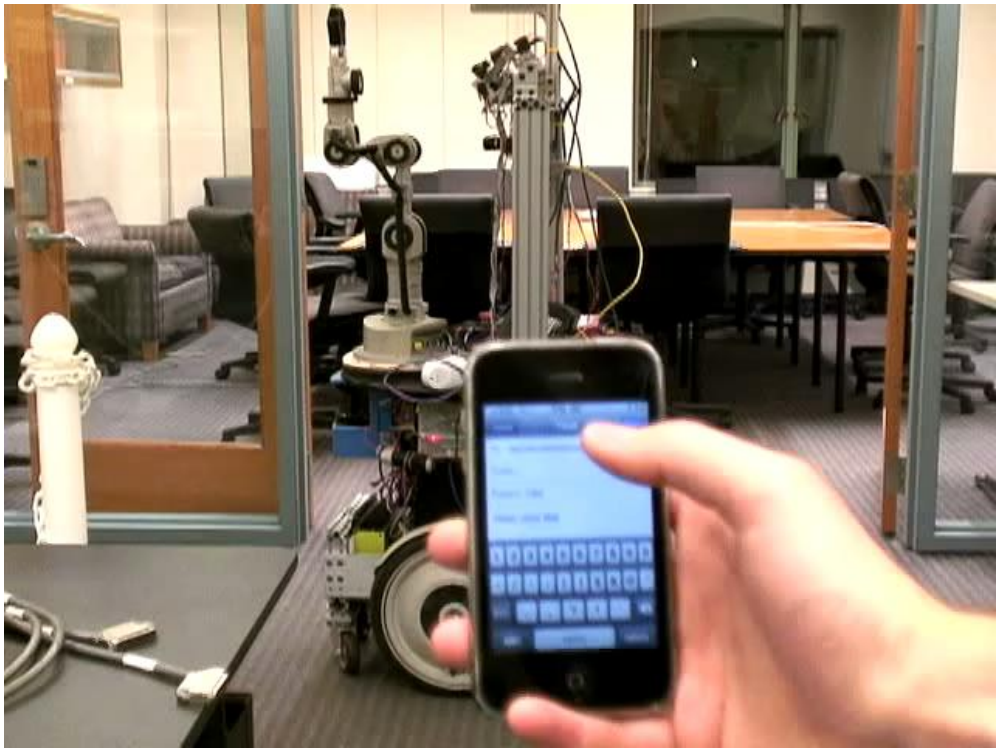
- ROS wrappings of the Stage simulator
- See ros.org for installation instructions
- Tools demonstrated:
  - rxgraph: live view of process interconnections
  - rostopic: print message streams to the console
  - rosnode: print publications/subscriptions of nodes
  - But wait, there's more! See ros.org

Experiments: 2008

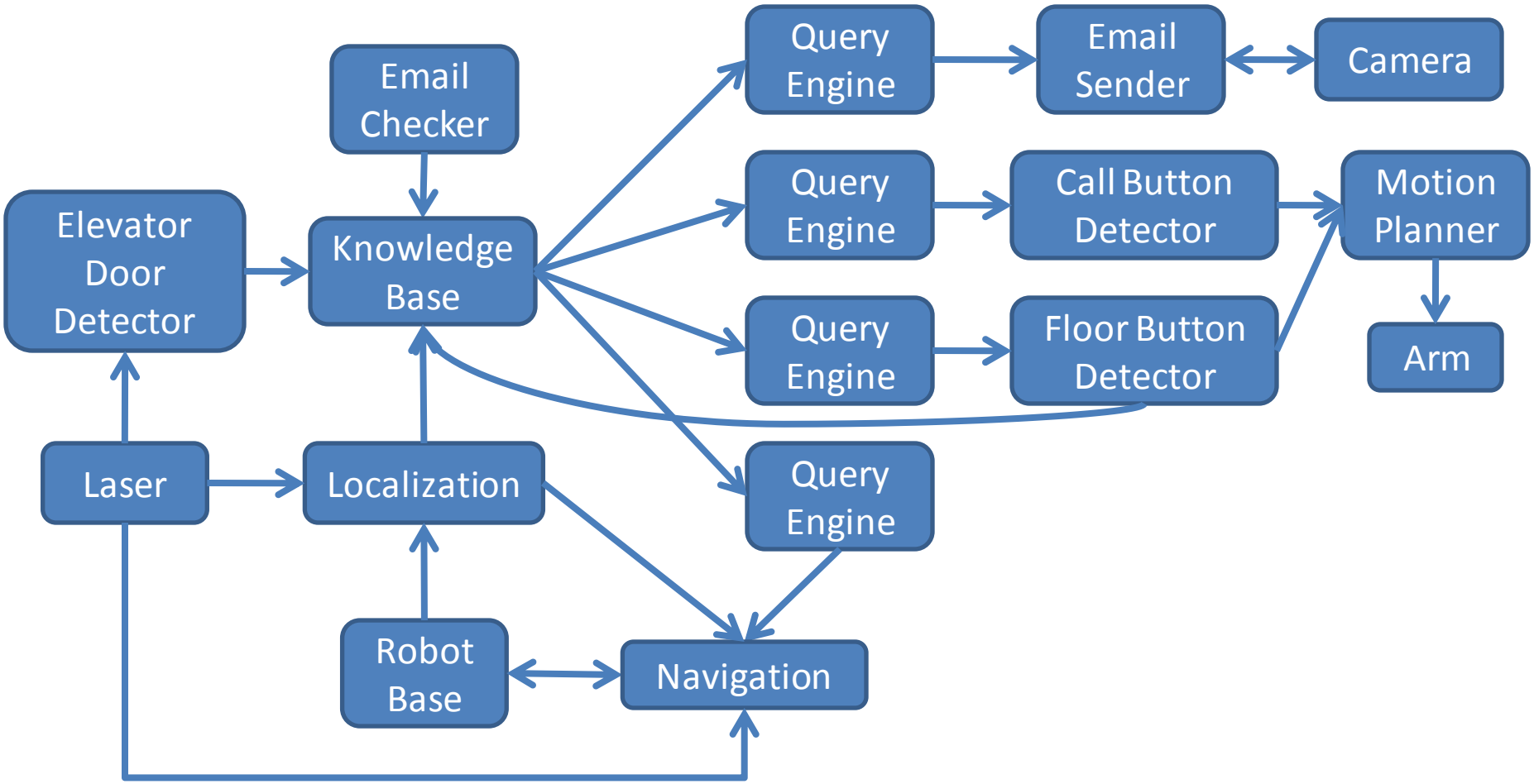
- Asynchronous connections via ROS
- Synchronous top-level executive (ruby script)
- Large backend

Experiments: 2009

- Asynchronous connections via ROS
- Coarse world model updated asynchronously
- Executive as functions of world model
- Email to send tasks



Building-Monitor Application



Lessons Learned

- Given sufficient hacking time, anything can work
- But, event-based systems scale more gracefully
  - Less painful to run and debug
  - Less painful to make more robust
  - Fewer assumptions hard-coded: less brittle
  - Code re-use is easier

Conclusion

- ROS supports publish/subscribe messaging
- Very few assumptions built into ROS
- Any number of systems can be built on top of it
- Much, much more available than discussed here

ros.org

Event-based Systems with ROS: Examples from the STAIR Project

Morgan Quigley
Stanford University
Joint work with:
STAIR: Quoc Le, Ellen Klingbeil, Blake Carpenter, Andrew Ng
ROS: Eric Berger, Ken Conley, Brian Gerkey, and many others