RaaS: Robotics as a Service
A Service Model for Cloud Robotics

Arjun Singh, Ben Kehoe, Ken Goldberg, Pieter Abbeel
University of California, Berkeley
Example Use Case

- Objects coming down conveyer belt
- Identify each object, pick it up, and place into correct bin
  - Low-level control
  - Object recognition
  - Grasp planning
  - Motion planning
How to Solve It with ROS

- Acquire computing hardware
- Install ROS and relevant packages
- Execute and maintain software and computing hardware

Drawing: Ketrina Yim
Pros and Cons of ROS

- Huge positive impact on robotics (especially sharing)
- Core great for messaging, low-level control – components that make a robot operating system
- Asking it to do too much
  - All code for everything robotics related

Packages required for barebones ROS installation

Packages required for full ROS installation
Pros and Cons of ROS

- Intimidating for non-roboticists
- Nontrivial to set up secure distributed networking with ROS -- need to understand VPNs, have control over network environment, etc.
- Dependencies can turn into a nightmare (especially with multiple ROS versions)
Computing Environment has Changed

- ROS design started in 2006 – a lot has changed!
- Cloud computing: Easy access to vast numbers of machines
- Software engineering: Service-oriented architectures and Software as a Service
  - E.g. Google Docs vs. Microsoft Office
Example Services – Motion Planning

Start State

Motion Planning Service

Goal State

Output Trajectory
Example Services – Object Recognition

Input Scene

Object Recognition Service

Detected Objects and Poses
Our Idea: Robotics as a Service (RaaS)

- Publish algorithms as services
- Consume services on your robot
- Collaborative data collection
Related Work

Cloud Robotics
- Rapyuta (www.rapyuta.org/)
- Rocon (www.robotconcert.org/)
- Goldberg et al. 1995
- Arumugam et al. 2010
- Ciocarlie et al. 2010
- Kuffner 2010
- Remy and Blake 2011
- Blake et al. 2011
- Waibel et al. 2011

Robotics Platform as a Service
- Higher-level orchestration
- Networked robotics
- Cloud computing for robots
- Big data for robotics
- Cloud-enabled robots

Service-oriented robotics
- Shared knowledge

Open-Source Tools
- Movelt! (Sucan and Chitta)
- GraspIt! (Ciocarlie and Miller)
- OpenCV (Bradski)
- OpenRAVE (Diankov)
- Ladon (www.ladonize.org)

Open-Source Tools
- Motion planning
- Grasp planning
- Computer vision
- Motion planning
- Web service framework

Commercial Products
- Mashape
- PiCloud

Commercial Products
- Index of service APIs
- Move computation into cloud
<table>
<thead>
<tr>
<th>Limited Resources</th>
<th>Code Sharing</th>
<th>Encapsulation</th>
<th>Parallelism</th>
<th>Common Interfaces</th>
</tr>
</thead>
</table>

Why RaaS?
Why RaaS?

- Some robotic platforms have limited onboard computation (e.g. Baxter)
  - Can’t run sophisticated robotics algorithms
  - Need offboard computation – use the cloud or buy computers
- Services give a straightforward way to move computation to the cloud
Why RaaS?

- Spend months working on your algorithm, finally finish it, and you want to share it
  - Need to figure out what all the dependencies are
  - Need to document how to install everything
  - Need to document how to use your code
  - Don’t have time for any of these
- Packaged service: only worry about API users interact with
Why RaaS?

- Robotics requires integrating many different components
  - Vision, NLP, control, messaging, planning, grasping, etc. – each alone is complex
  - Huge number of dependencies – something is bound to conflict
- Services force encapsulation of components
  - Use object detection and planning systems without worrying about conflicts

### Limited Resources
- Code Sharing
- Encapsulation
- Parallelism
- Common Interfaces

#### Object Detection:
- Dependency A 1.0
- Dependency B 2.1
- Dependency C 2.5

#### Motion Planning:
- Dependency B 2.1
- Dependency C 3.2

#### Motion Planning:
- Dependency A 1.0
- Dependency B 2.1
- Dependency C 2.5
- Dependency C 3.2
Why RaaS?

- Service: natural interface for parallelizing computation
- Insulates user from managing parallelism
- Automatically run multiple instances of a service on multiple machines

Limited Resources  Code Sharing  Encapsulation  **Parallelism**  Common Interfaces
Why RaaS?

- Easy benchmarking and comparison
- Use common interfaces when defining services
  - E.g. object recognition systems
    - Input: Image
    - Output: List of object identities
  - Can’t expect researchers to implement interfaces at a library level
- Swap out services
RaaS Workflow
RaaS Workflow – Algorithm Developers
RaaS Workflow – Algorithm Developers

- Write your usual code
- Wrap with service code
- Create Amazon Machine Image (AMI)
- Publish your service
RaaS Workflow – Algorithm Developers

- Make creating a web service as simple as possible
- Ignore HTTP, serialization, encoding, webservers, etc.
- Rich set of types available for service methods
  - Strings, floats, integers, binary blobs, raw files, timestamps, durations, poses, transformations, vectors, matrices, images, point clouds
  - Create your own
- Can serve ROS nodes as web services
RaaS Workflow – Algorithm Users

RaaS

Motion Planning 1
Grasp Planning
Object Recognition

Motion Planning 2
Grasp Planning
RaaS Workflow – Algorithm Users

- Choose services
- Launch machines in the cloud
- Connect to machine and use services
- Detect objects in images, plan motions, etc.

Service Definition:
```python
class ObjectDetectionService(Service):
    @method_input(Image, 'image')
    @method_output([String], 'detected_objects')
def detect_objects(self, **inputs):
    image = inputs['image']
    detected_objects = my_detector.detect_objects(image)
    return detected_objects
```

Client Code:
```
from object_detection_service import ObjectDetectionService

service = ObjectDetectionService()
```

Choose Amazon Machine Image (AMI) from one of the tabbed lists below:
RaaS Workflow – Algorithm Users

- Compare different algorithms implementing the same interface by changing a single URL
- Again, ignore serialization, encoding/decoding, HTTP, etc. – use like an ordinary function call
- Use web-service-based ROS Nodes as if they were local ROS Nodes
How to Solve It with RaaS

- No need to worry about computing hardware
- Install barebones ROS – (i.e. only messaging, low-level control, etc.)
- Use object recognition, motion planning, grasp planning services from RaaS
- RaaS complements ROS

Drawing: Ketrina Yim
Collaborative Data Collection – Example

- Ten camera rig with controllable turntable
- High quality 3d models from about 600 images in 5 minutes of human time
- Ship us an object, we upload model to cloud, you can recognize it in images
Roadmap

(Ar) Complete (pre-alpha)

- Service framework implemented for Python
- Service directory implemented for EC2
- Need to polish up rough edges and automate tedious steps

Future Work

- Linux containers rather than Amazon Machine Images
  - Can then run services anywhere – in your lab or in the cloud
- Infrastructure for other programming languages
- Data collection APIs

Interested? Email us! arjun@eecs.berkeley.edu