CS 309: Autonomous Robots
FRI I

Introduction to ROS

Instructor: Justin Hart

http://justinhart.net/teaching/2020_spring_cs309/

ROS

Robot Operating System

- Not an OS
- Middleware providing communication
- A collection of utilities relevant to robotics

Before ROS

- Robots mostly ran highly-customized software
  - Many still do
- Some things LIKE ROS were in the wild
  - YARP – Yet Another Robot Platform
- Robots require many pieces of specialized software
  - Before ROS, this often meant
    - Digging through other groups' software libraries
    - Having many specialists in your group who understand these pieces
    - Piecing these things into your robot's custom software
- As a result
  - Fewer schools had robots
  - There were fewer robot companies
  - The cost of entry was higher

Willow Garage

- Founded in 2006
  - Robotics company
  - Technology incubator

- First two projects
  - DARPA Urban Challenge
  - Solar-powered boat

STAIR – Stanford Artificial Intelligence Robot

- Stanford had STAIR 4 robots
  - What if they all had the same basic starter software?
  - This was the start of ROS

- 2007 - Stanford AI Lab makes the first ROS release

- 2008 - Two concepts pitched to Willow Garage
  - Build a common robotics hardware platform
    - The Personal Robot 1 (PR1)
  - Build a common robotics software platform
    - ROS

- Willow Garage
  - Hires a bunch of research scientists & other research personnel
  - Kicks off a bunch of internal projects

- 2010
  - ROS had grown significantly
  - The PR2 goes for sale
    - Price ~$400,00 each
  - 11 schools included in the beta program get theirs for free

- PR2 beta program schools must open source software developed on the PR2

- A large collection of ROS software becomes available for the PR2

- This creates a robotics ecosystem with ROS and the PR2 at the center

- ROS becomes the closest thing to a “starter kit” for robotics in existence

- ROS becomes the dominant technology for robotics in academia and in certain industry sectors

Communications

- Topics
- Services
- actionlib

ROS Topics

- Publish/Subscribe

- A **node** may publish a **topic**
  - Example: A 3D camera may publish a point cloud

- Another node may **subscribe to** a topic
  - An object detector may subscribe to video data

- Many nodes may concurrently subscribe to each topic

ROS Services

- Remote Procedure Call
  - Allows a node (program) to call a function on another node
- Useful if one program should exclusively handle some type of request
  - Only one node drives the robot
    - But many may take turns telling it it where to go
  - Similar things happen for
    - The arm
    - Maintaining the robot's map
    - Recording and accessing the database of the robot's knowledge

ROS Actionlib

- RPC + Feedback

- Used when status updates about progress matter
  - “Is the robot still navigating, or is there a problem?”
  - “Is the robot still talking, or did it fail?”

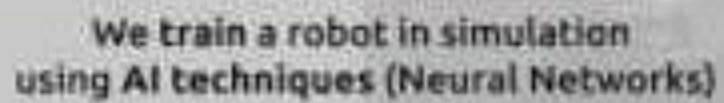
- Generally, this is if an action might take a while to complete

A few basic tools

- Gazebo
- rviz
- MoveIt!

Gazebo

- A 3D robot simulator
  - Software can be written and tested in Gazebo, then run on a robot with few modifications
- Works with ROS software
  - Publishes topics
  - Services actionlib and service calls
- Users can download robot models or build them
- Users can download or build models of places for the robots to move around in
  - We have a GDC simulator in Gazebo

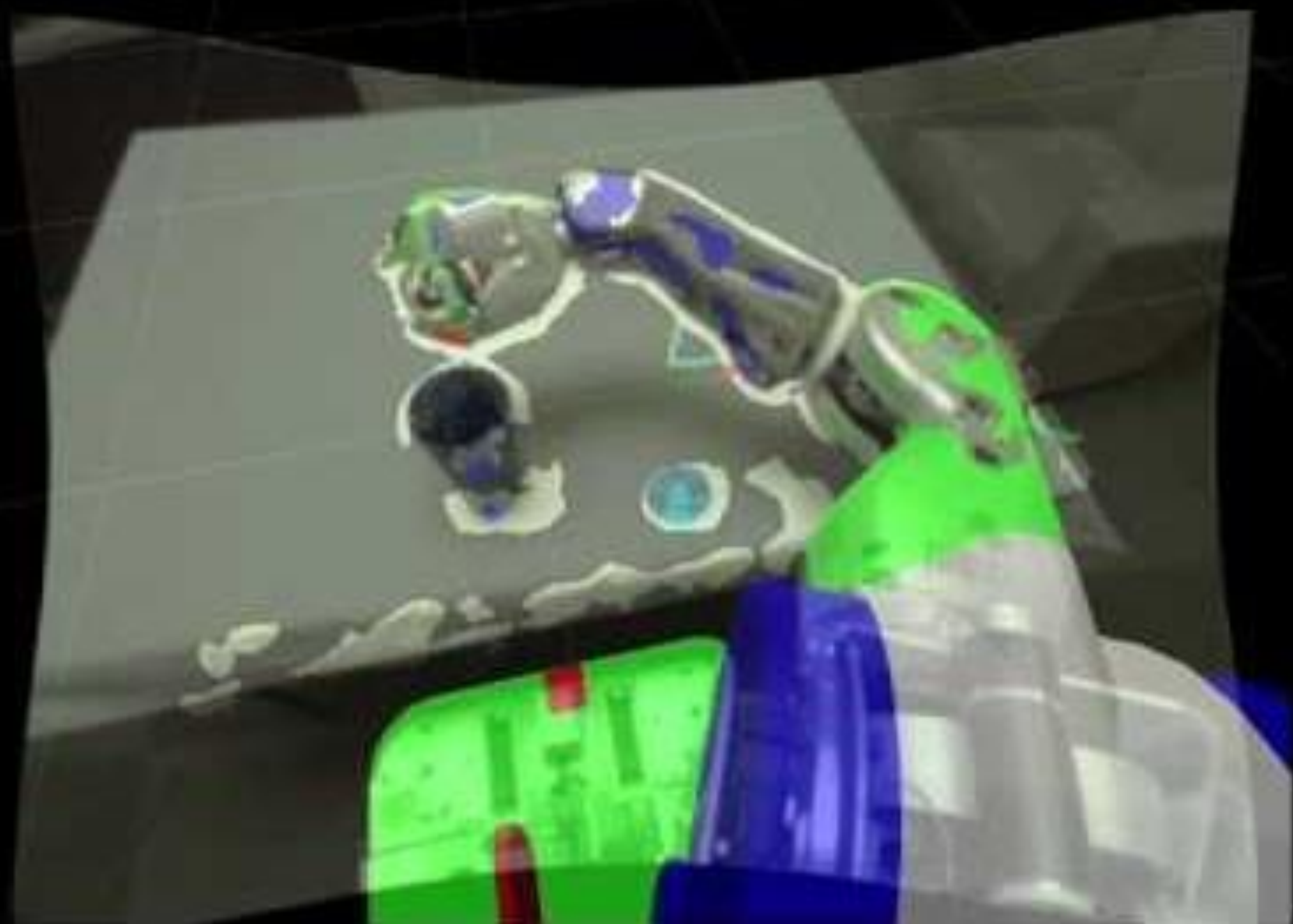

We train a robot in simulation
using AI techniques (Neural Networks)

rviz

- ROS Visualizer

- Visualizes many kinds of data
  - TF (transform) frames
    - Where an object is, and what direction it is facing
  - URDF – Universal Robot Definition File
    - 3D models of robots
  - Point Cloud
    - 3D vision data
  - Camera images

- Seeing what the robot “sees” makes it easier to write and debug software



MoveIt!

- Simplifies complex motion planning, especially for arms
- Pipeline
  - Perceptual data
  - Kinematic solvers
    - How the arm can move
  - Motion planners
- Customization for all of this
- Simulation and visualization

Rviz
x1

Packages and Stacks

- There is a large collection of ROS software
- Related software is distributed as a **package**
- Nodes are linked together into **stacks**
  - MoveIt! is a ROS stack

- Packages include
  - Sensing – Running sensors
  - Perception – Finding objects, shapes, places
  - Navigation – Most ROS robots can drive without modification
  - Many others..

Now, we will jump to how we start the robots in the lab

Now, let’s jump back in and make ROS practical

ROS Topics

- Publish/Subscribe
- A **node** may publish a **topic**
  - Example: A 3D camera may publish a point cloud
- Another node may **subscribe to** a topic
  - An object detector may subscribe to video data
- Many nodes may concurrently subscribe to each topic

Communications

- Topics
- Services
- actionlib

ROS Topics

- Publish/Subscribe
- A **node** may publish a **topic**
  - Example: A 3D camera may publish a point cloud
- Another node may **subscribe to** a topic
  - An object detector may subscribe to video data
- Many nodes may concurrently subscribe to each topic

ROS Services

- Remote Procedure Call
  - Allows a node (program) to call a function on another node
- Useful if one program should exclusively handle some type of request
  - Only one node drives the robot
    - But many may take turns telling it it where to go
  - Similar things happen for
    - The arm
    - Maintaining the robot's map
    - Recording and accessing the database of the robot's knowledge

ROS Actionlib

- RPC + Feedback

- Used when status updates about progress matter
  - “Is the robot still navigating, or is there a problem?”
  - “Is the robot still talking, or did it fail?”

- Generally, this is if an action might take a while to complete

Packages and Stacks

- There is a large collection of ROS software
- Related software is distributed as a **package**
- Nodes are linked together into **stacks**
  - MoveIt! is a ROS stack

- Packages include
  - Sensing – Running sensors
  - Perception – Finding objects, shapes, places
  - Navigation – Most ROS robots can drive without modification
  - Many others..

PR2
PR2
PR2
PR2
PR2
PR2
PR2
PR2
PR2

What you should not do with ROS

- ROS is not magic
- It does not reduce research to the practice of combining good ROS packages
  - Pre-existing software
- You should not rossify every function call
  - We will learn about this later
  - This runs very slowly
- It is often cheaper and easier to use non-rossified code

What you should do with ROS

- Use ROS to *wrap* functionality
    - Make your code available to other components
    - Treat data like this
        - What needs to come in from the rest of the system
        - What needs to go out to the rest of the system
        - If it's only used internally by your node or application, don't make it a ROS call.
- Others have built ROS nodes, use them if they help you
    - Do not use them if they do not help you
    - They probably did nothing magic with their code.
        - You may be better off calling their code normally.
- Because the components use standardized protocols and formats, they are generally compatible with each other

What you should do with ROS

- Use ROS to *wrap* functionality
  - Make your code available to other components
  - Treat data like this
    - What needs to come in from the rest of the system
    - What needs to go out to the rest of the system
    - If it's only used internally by your node or application, don't make it a ROS call.

- Others have built ROS nodes, use them if they help you
  - Do not use them if they do not help you
  - They probably did nothing magic with their code.
    - You may be better off calling their code normally.

- Because the components use standardized protocols and formats, they are generally compatible with each other

Installing ROS

Despite giving you these instructions, I would really encourage you to use the lab in GDC 3.414, or the 3rd floor computer lab.

Installing ROS on your laptop or home machine is optional. We have machines for you to use, and the department machines on the first floor have ROS.

Installing ROS

- Ubuntu 16.04
  - Follow install instructions from Ubuntu website
  - Dual boot. VMs will run ROS very poorly
    - Back up your files first!
    - Disk resizing works very poorly!
    - Make a partition for Windows and one for Linux
  - If you need help, see the mentors

Installing ROS

- ROS installation instructions
  - Follow these only if installing on your personal machine
    - These require root! Don't do this to a department or lab machine!
    - http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment
    - Pick "Kinetic Kame"
      - Ubuntu
      - AMD64
    - Set up sources.list
      - sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
    - Add cryptographic keys identifying this as a trusted source
      - sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
    - Use APT to install
      - Sudo apt-get update
      - Sudo apt-get install ros-kinetic-desktop-full

Installing ROS

- APT will tell you if there are problems
  - It will also tell you, generally, how to fix those problems, just read the error
  - If it does not, copy and paste the error into Google
  - If that fails, then ask a mentor, then me.

Sidebar.. Who to ask when there's a problem

1. Yourself. You're training to become the person that others would ask, so you need to learn sometime.
2. The compiler or software, it's probably giving you an error message.
3. Google
4. A peer-mentor. That's their job.
5. Me.

If Google or the tool have the answer, then that's the best route! It's not that we don't want to help you. It's that you're trying to develop a skill for when nobody is there to help you. IE, when you are the person giving the help.

ROS machines

- If you are using a lab machine, it has Kinetic installed
  - Both the undergraduate lab and the BWI lab
- If it involves sudo, you probably should not be doing it on a lab machine.
  - You will need sudo on your personal machine, though.

Installing ROS

- Setting up rosdep
  - sudo rosdep init
  - rosdep update

- rosdep is used to set up dependencies
  - For instance, you download a package, but it requires another package to run properly. This makes this tracking and setup automatic.

Configuring your ROS Environment

- http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment
- You can do this manually, but you probably will not want to
  - source /opt/ros/<distro>/setup.bash
  - In this case, <distro> is kinetic
- Configuring your environment sets up “environment variables”
  - $PATH – Where the computer looks for programs
  - $ROS_PACKAGE_PATH – Where ROS looks for packages
  - $ROS_MASTER_URI – Where the roscore instance is running
  - Many others

Creating a ROS Workspace

- mkdir –p ~/catkin_ws/src
- cd ~/catkin_ws/src
- catkin_init_workspace
  - This is different from the guide. Both work.
- catkin build
  - Please use catkin build rather than catkin_make
    - We have been standardizing on this as a lab.

Navigating ROS

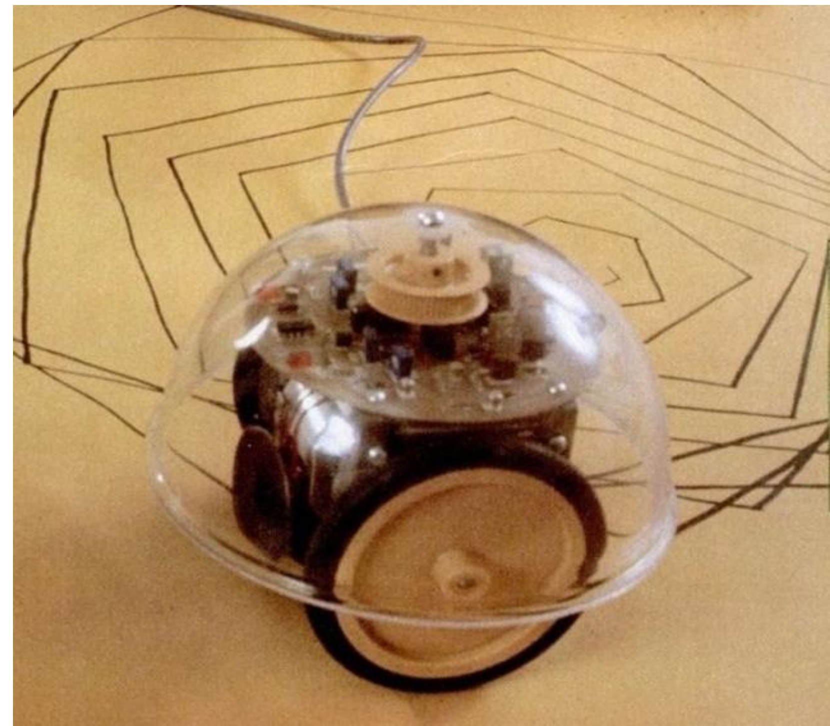
- http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem
- We will just follow this tutorial

Cmake

- In HW2 we used “make”
- ROS uses cmake – cross-platform make
- Makefiles in large systems (like ROS) can become complicated
  - Cmake is intended to manage this

A quick primer on ROS nodes

- http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes
  - roscore
    - Manages communications between nodes
  - turtlesim_node
    - Connects to roscore in order to communicate



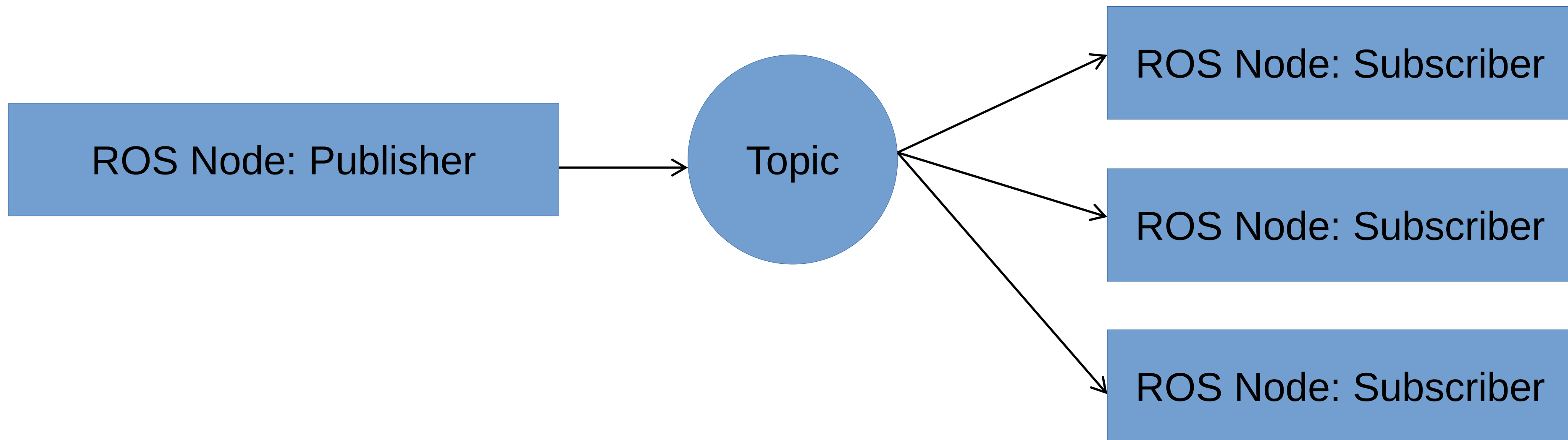
Understanding ROS topics

- http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics
  - Topics run a publish/subscribe architecture
    - A “publisher” provides a stream of data
    - A “subscriber” listens to the stream
  - The first demo shows
    - turtlesim_node – a subscriber
    - turtlesim_teleop_key – a publisher
  - If we run ros_topic_echo, we can see the keys telling the turtle what to do

Publish/Subscribe

- ROS topics use a publish/subscribe architecture
  - Publishers
    - Broadcast data on a “topic”
    - One publisher per topic
  - Subscribers
    - Read data from a topic
    - Multiple subscribers can subscribe to the same topic
  - Example
    - rviz with Kinect v2 data

A simplified diagram



roscore

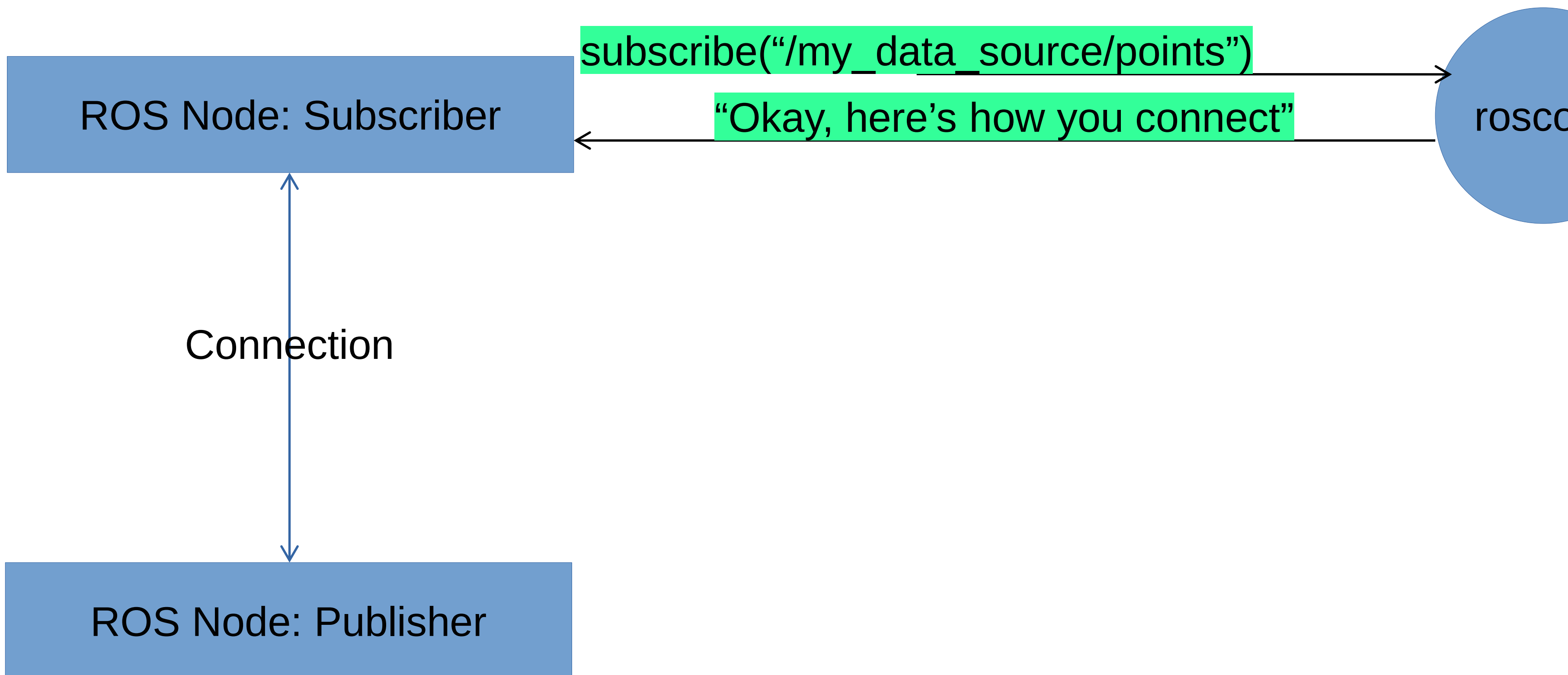
- Coordinates ROS communications
  - When a node publishes a topic, it registers this topic on roscore with a name
  - Other nodes can subscribe using this name

roscore, in practice

ROS Node: Publisher

publish(“/my_data_source/points”)

roscore, in practice



ROS Topics

- Connect over a network socket
  - $ROS_MASTER_URI – Points to roscore
  - Default is http://localhost:11311
  - In class we will use one robot
    - But for RoboCup@Home, we use a single ROS master to control a network of computers doing different things for the robot
    - $ROS_MASTER_URI provides the roadmap for our programs

ROS Message Types

- ROS has a special compiler for messaging
- You can specify what type of message is published on a topic
- Message types are specified in a very simplified language
- The compiler outputs the code for the target programming language (if supported) (usually C++ or Python)

ROS Message Types

- To use a ROS topic, you do not really need to understand networking
- As we use it, each message type is stored in a C++ class
  - The compilers write this class
    - You do not
    - You write a simple text file saying what goes into the message

ROS Message Types

- For most common message types, ROS already has a specified message
  - This allows programs to interoperate, by using the commonly-supported message
  - If you use standard message types, your nodes will generally work with other existing ROS tools
    - For example, if you publish point cloud data
      - sensor_msgs/PointCloud
      - Now you can see your point cloud in rviz!
      - Now you can use your published data with other existing programs!

ROS Message Formats

- Topics look like this

```
string a
int64 b
```

CMakelists.txt & package.xml

- CMakelists.txt
  - Describes how to build the software
  - Works with catkin_make or catkin_build
- package.xml
  - Package manifest
  - Tells ROS and catkin what to do with your software
  - Describes
    - Dependencies
    - Licensing
    - Contact info
  - If you run catkin_create_package, it will give you a template that you can fill in!

Let’s look at a simple ROS message

/opt/ros/kinetic/std_msgs/String.msg

Catkin workspace & build tools

- catkin_make & catkin build
  - catkin build works a little better, but most tutorials use catkin_make
- catkin_init_workspace
  - Top-level script that initializes your catkin workspace
- catkin_create_package
  - Shortcut script to create a ROS package for your nodes

catkin_create_package

- catkin_create_package <package_name> roscpp rospy std_msgs <other dependencies if you need them>
  - Note that this makes a TEMPLATE, you will need to edit CMakeLists.txt and package.xml
  - For HW3, this will read

    catkin_create_pkg hw3 roscpp rospy std_msgs sensor_msgs cv_bridge image_transport

CMakeLists.txt

- Used to build your software
  - The version created by catkin_create_pkg is only a template, you will need to uncomment and modify the lines that you need

```
# add_executable(${PROJECT_NAME}_node src/hw3_node.cpp)

# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )
```

  - Possibly edit in other places

package.xml

- Version made by catkin_create_pkg is probably correct.
  - Your implementation may vary
- package.xml is your manifest file
- It tells ROS how to treat your package
- Name
- License
- Maintainer
- If it requires other packages in order to build or run it

rosbag

- rosbag records of plays back pre-recorded data from ROS topics
- rosbag play -l three_cups.bag
- -l makes it run the bag in a loop

Writing our first publisher

- String.msg is used in the ROS publisher/subscriber tutorial
  - http://wiki.ros.org/ROS/Tutorials/CreatingPackage
  - http://wiki.ros.org/ROS/Tutorials/BuildingPackages
  - http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

Writing our first subscriber

- Event-driven programming
  - Core idea
    - Wait for something to happen then act on what happened
    - The thing that happened is an event
  - Main is generally replaced with a method called a main loop

Setting up ROS in a node

- ros::init()
  - Sets up ROS
- ros::NodeHandle
  - Manages the connection to roscore

ROS Topic Setup

- Publish/Subscribe
  - Publisher
    - advertise()
    - publish()
  - Subscriber
    - subscribe()

ROS Event Processing

- ROS main loop processing calls
  - ros::spin()
  - ros::spinOnce()
  - ros::MultiThreadedSpinner spinner(4)
    - Start 4 threads
    - Spin in each thread
  - ros::AsyncSpinner spinner(4)
    - Another multithreaded spinner, does not synchronize the 4 threads
- These maintain a “callback queue”
  - Like this:
    - A message is received
    - It is stored in the callback queue
    - The callback queue calls the callback (the registered function) on the message in the order it was received
  - Every event in ROS goes into the callback queue

Writing our first subscriber

- Main loop
  - Checks repeatedly to see if something happened
  - ros::spinOnce() does the checking
  - Subscribers check to see if a message has been published on a topic
- At the start of the program a "callback" is "registered" with the event framework
  - In our case, that framework is ROS
  - Whenever the event happens, the callback is called.
- This will make more sense when we try it out!
  - http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

ROS Message Types

- Services look kind of like this

```
int64 a
int64 b
---
string retVal
```

Using custom messages

- Example
    - TwoInts.msg
        - Changes to CMakeLists.txt
        - Changes to package.xml
    - sendint.cpp
    - recint.cpp

ROS RPC Event Processing

- RPCs like ROS service calls and actionlib work the same way
  - Traditionally, this is across a network, but on the same machine is fine
  - Also use roscore
  - Also serviced by the event driven model

- In RPCs
  - Message comes in with a “request”
  - The request is serviced
  - The node sends a message back with the return value

ROS Service Setup

- Remote Procedure Call
  - Server
    - advertiseService()
  - Client
    - call()

Using services

- Example
  - add_two_ints
  - And the client!!

Putting it all together

- Let’s compute Fibonacci numbers with ROS topics and services
  - two_ints_pub
  - add_two_ints
  - fib_node