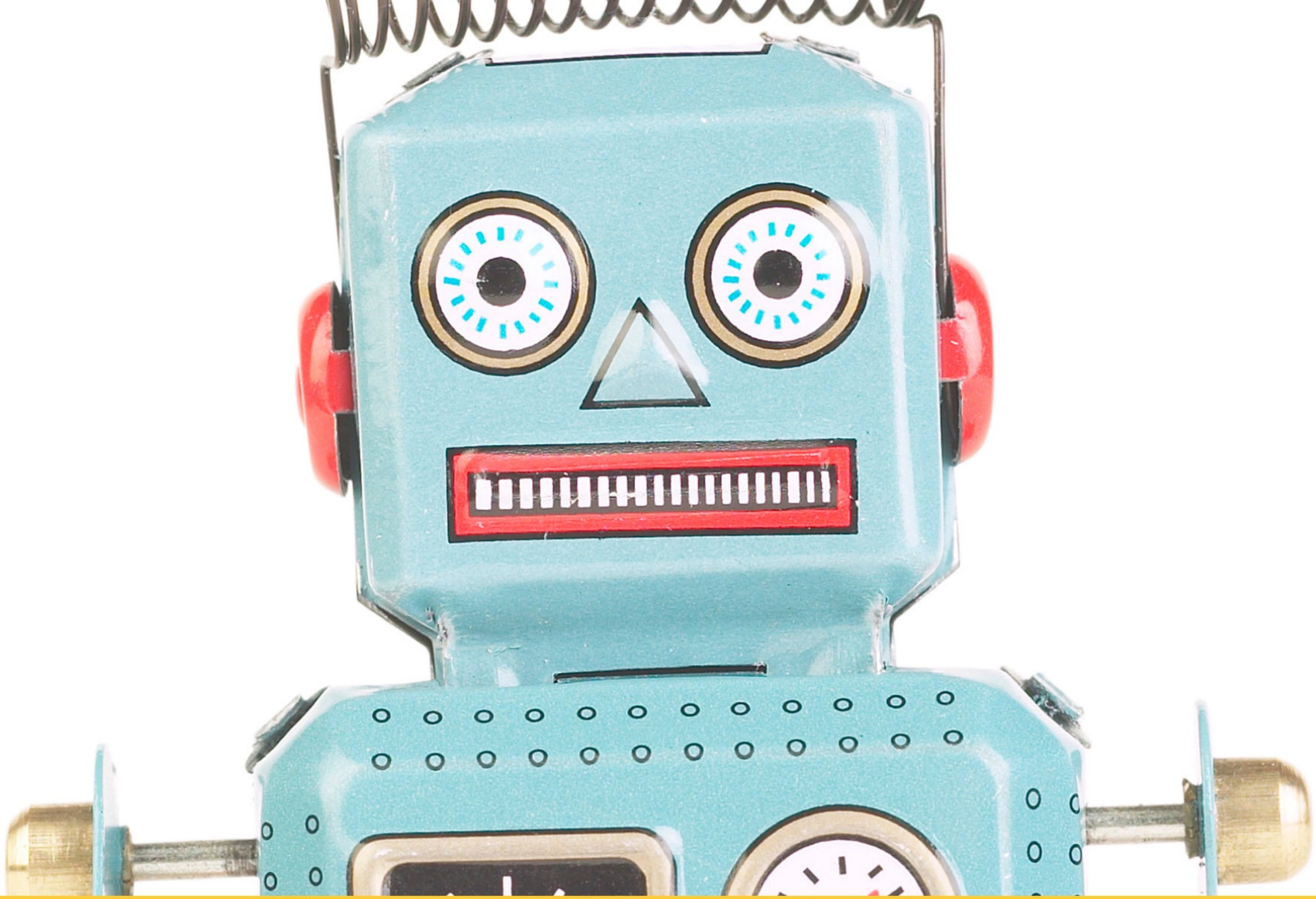


ROBOT OPERATING SYSTEM

AN INTRODUCTION TO ROS

AINSTEIN

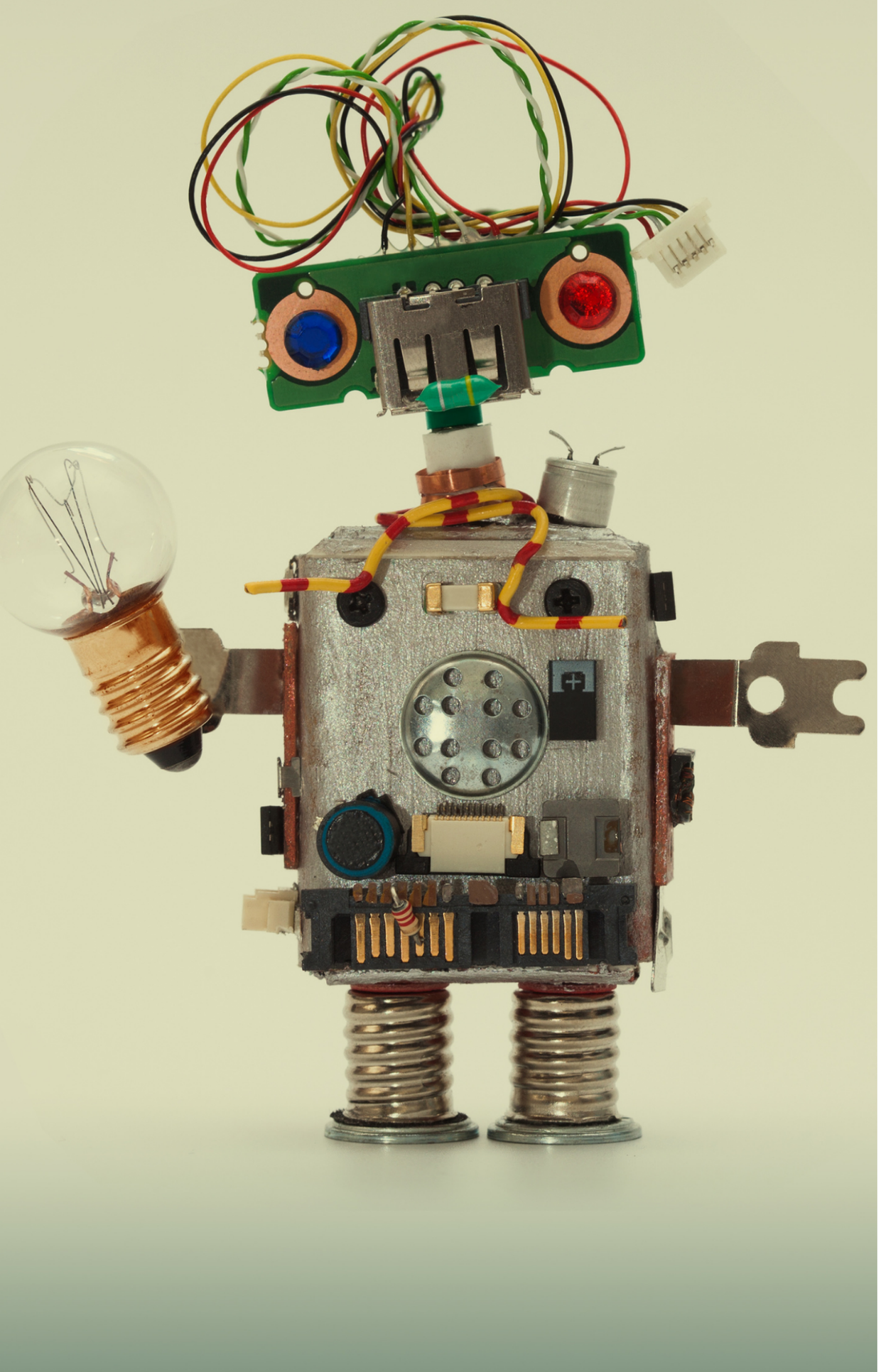
The History of ROS

Believe it or not, ROS has been around since 2006 after it was created by the Willow Garage*. This group was loosely tied to early Stanford attempts at standardizing robotics and matured through participation in the DARPA Grand Challenge (a self robot competition).

Willow Garage is famously known for their development of the PR2 robot, which has become a standard for robotics research. The PR2 is the flagship robot used for ROS. For example, 11 universities have this very robot in order to aid in learning and develop hardware standardization. Willow Garage also developed the Point Cloud Library (PCL) and maintained OpenCV** which has is critical for 2D and 3D perception.

Willow Garaged eventually evolved into the Open Source Robotics Foundation (2014 - present). They maintain and release ROS and the Gazebo Simulator which is used by companies and research labs across the globe as a simulator for Robotics.

* https://en.wikipedia.org/wiki/Willow_Garage
* ** https://opencv.org/



The basics of Robot Operating System

If you've ever used a computer or mobile device, you've used an operating system. Traditional operating systems for computers provide a lot of low level functionality.

For example:

- file systems
- device drivers
- networking
- memory management
- etc

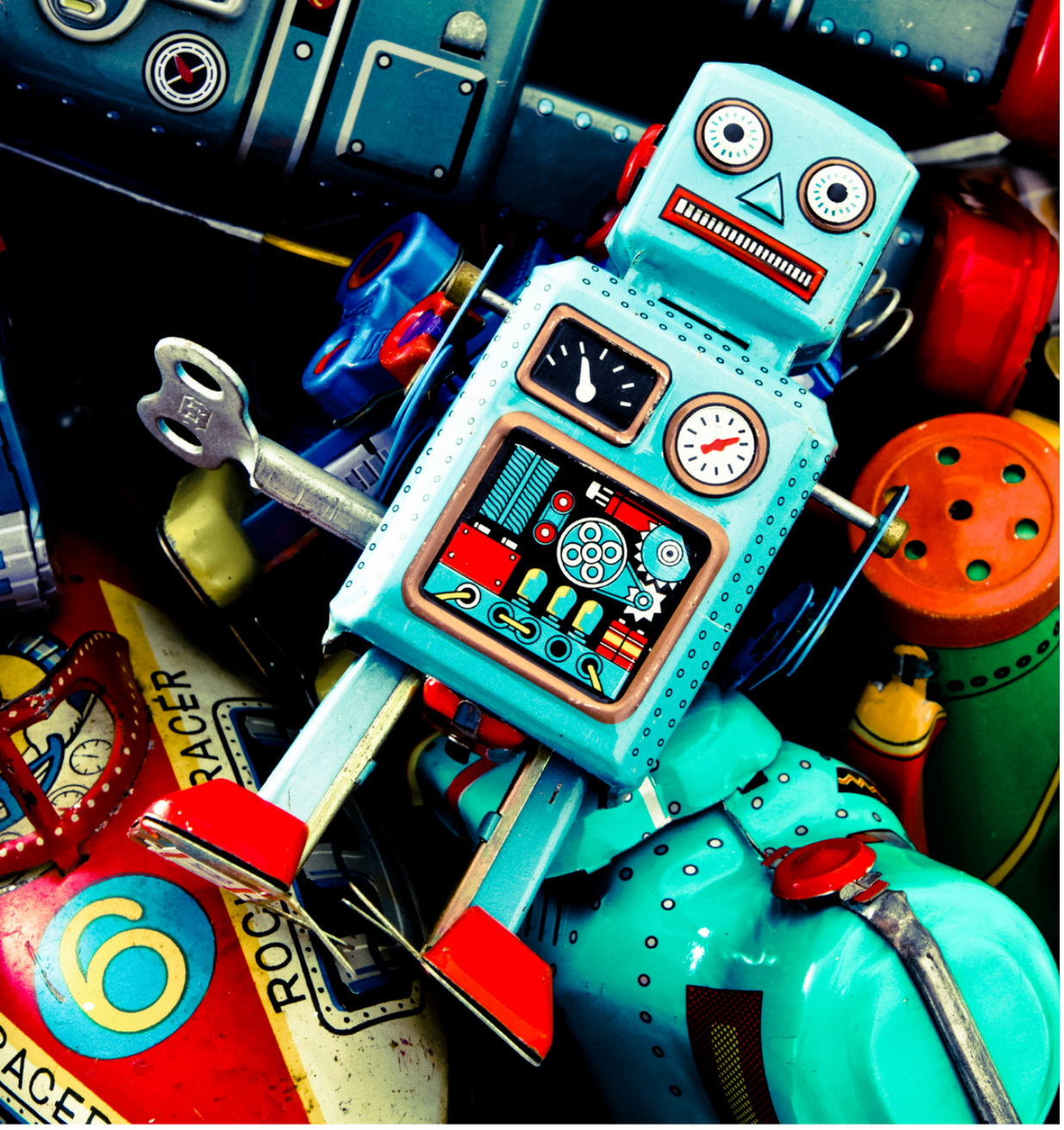
These are things you don't want to update or tweak each and every time you want to bring a new application to market.

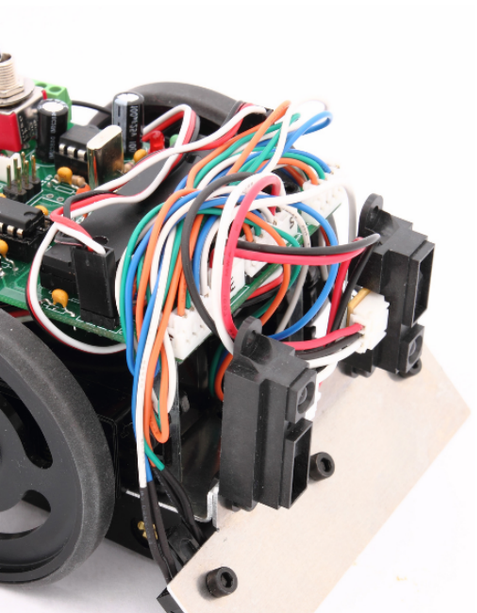
This is why we have Operating Systems (OS). The operating system does the work for you and lets you build on top of it saving you a ton of time.

In addition to the operating system, there are standards such as:

- Hardware: PCI bus, USB port, & FireWire
- Software: HTML, JPG, TCP/IP, & POSIX

The combination of these standards and operating systems are what help you write software and applications that will run on your systems easily.





OK. Cool. What about robots?

The idea of a robot, apart from immediately thinking SkyNet by Cyberdyne Systems, is pretty complex.

Robots have:

- Sensors
- Actuators
- Computing Hardware
- And they might even communicate to other robots

Robots also don't all look the same and come in all shapes and sizes.

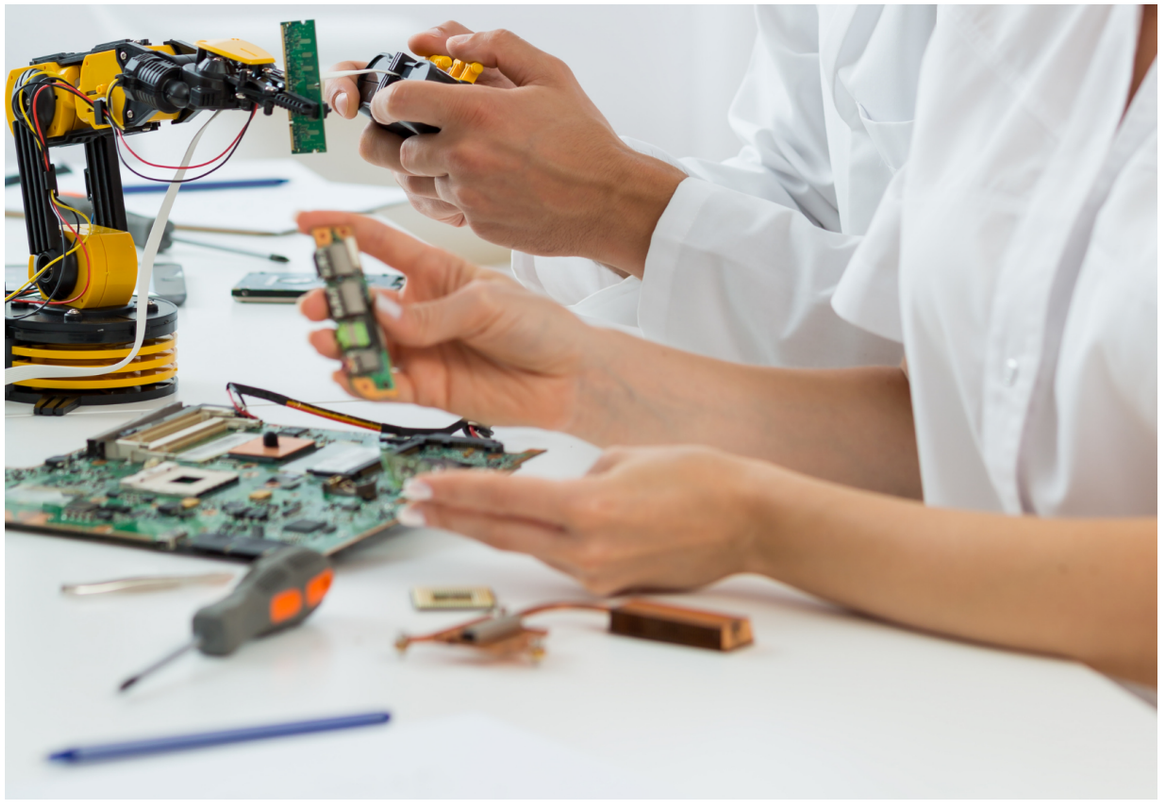
The main categories of robots are:

- Drones
- Medical robots
- Self-driving cars
- Flying robots
- Manipulators
- Underwater robots
- Ground robots

So how does one provide low level functionality to write applications?

PS: Basically if it has sensors, actuators, and computing hardware, you've got yourself a robot stew.

ROS is Middleware



ROS, is like an operating system in that it provides a lot of the lower level functionality that is needed to build applications on.

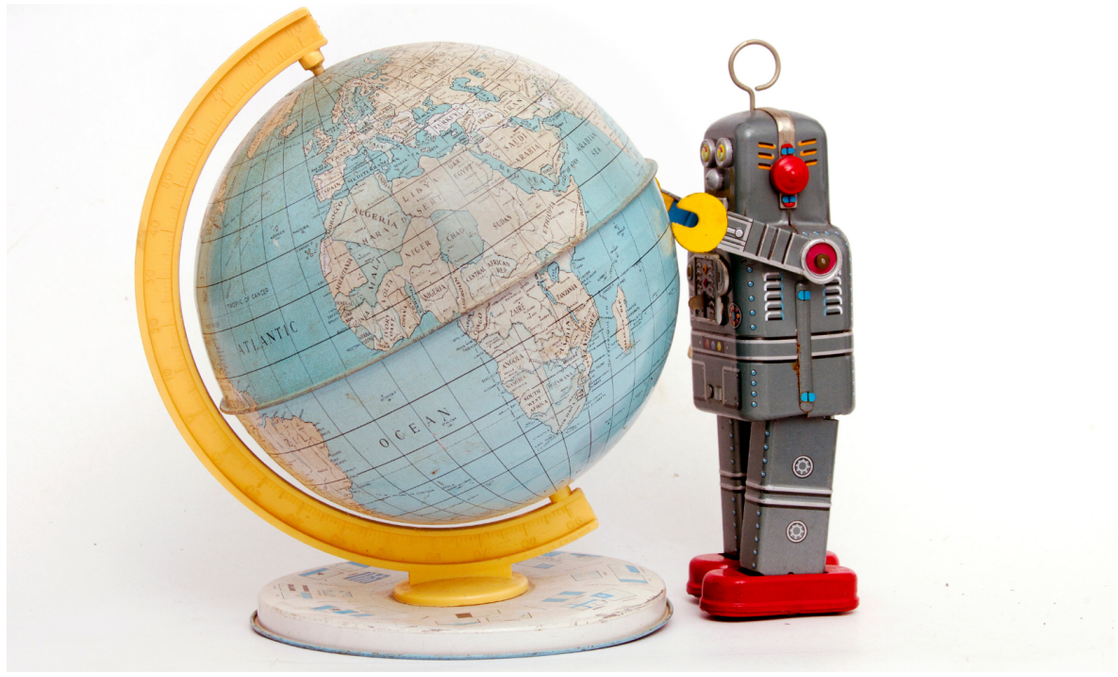
ROS may appear to be an OS, but it's really a **robot middleware package.** ROS is the plumbing that connects all the different sensors and actuators on a robot and connects different robots to each other.

ROS provides:

- Peer-to-peer communication between nodes (separate processes)
- Hardware abstraction
- Sensor and actuator "drivers"

ROS also provides:

- Visualization tools
- Robot simulators
- Data time synchronization/logging/playback
- A TON of application level packages such as drivers, SLAM, perception, planning, control, etc that are accessible to users across the globe!



ROS is less of an operating system and more of an **ECOSYSTEM.**

Ok. How does ROS work?

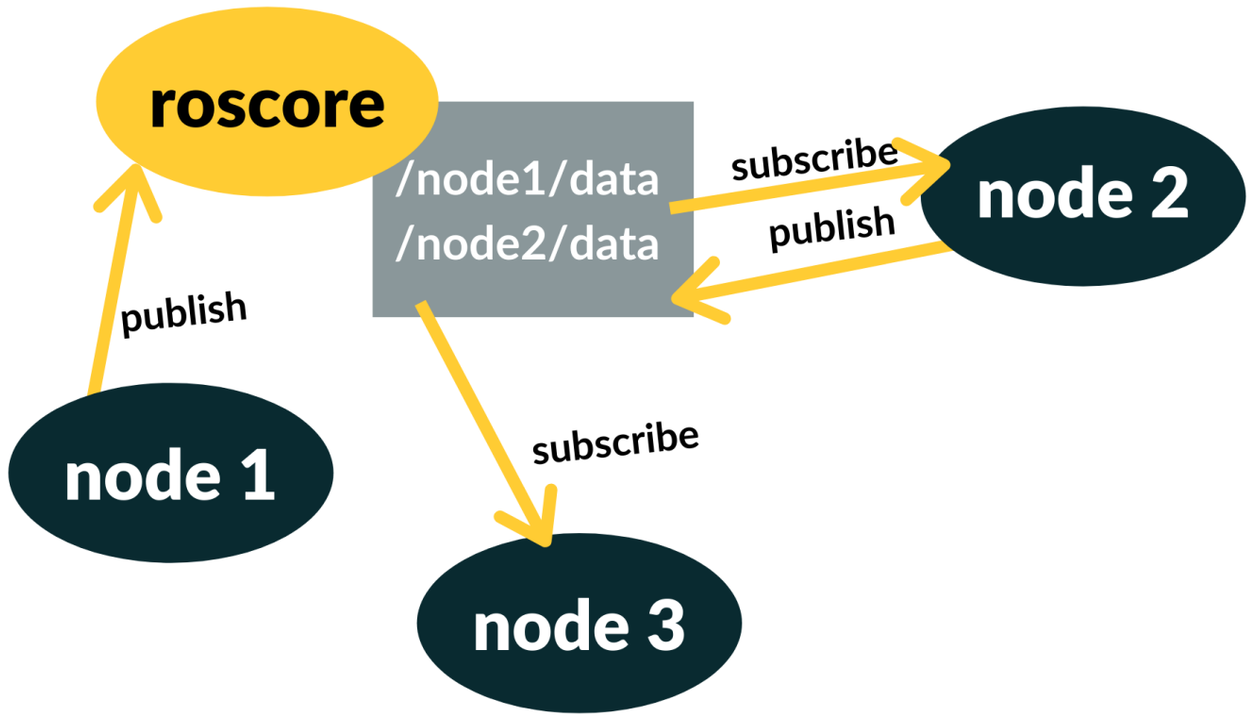
ROS is a peer-to-peer network using a master node called **roscore** to provide a name service to directly connect nodes. When a new node is run, it informs the master (i.e. roscore) about the topics it publishes and/or subscribes to. For example, this topic (/node1/data) could be a sensor driver for a radar that's getting data over ethernet and publishing it over ROS for other ROS nodes to consume.

Roscore now knows this node exists and the topic to which it's publishing its data

Subsequent nodes do the same, contacting the master (roscore) to form required connections. Roscore facilitates the name resolution. For example, if node3 says "I subscribe to node2's data topic" then roscore will connect the node2 to node 3

FYI. The messages themselves do NOT pass through master. It only connects the the nodes to the network and facilitates the transfer of information.

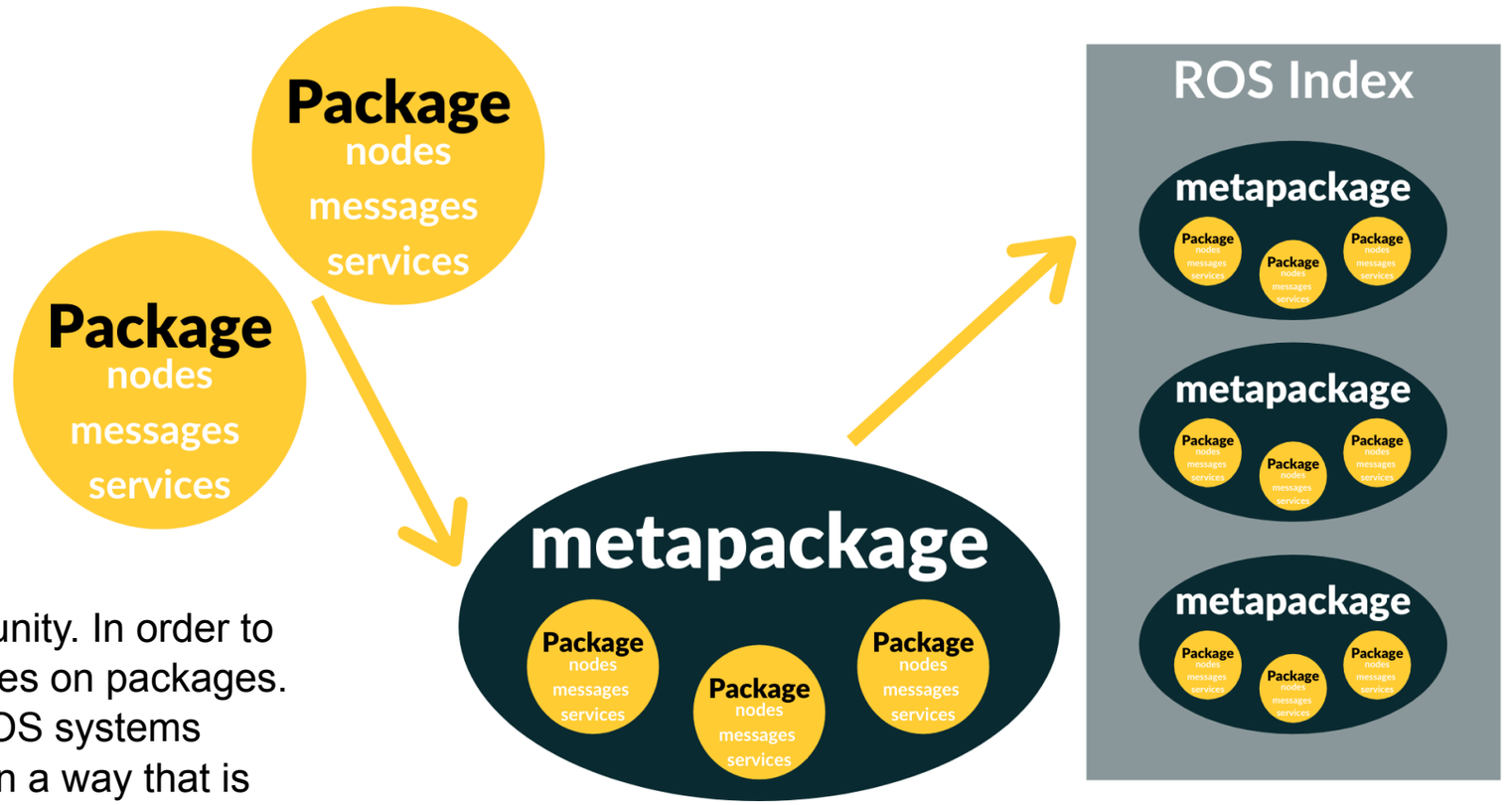
However there are all types of nodes that are highly specialized. For example, a node can record data messages time-synchronized to the same clock easily with a node from the rosbag package - and play it back later.



Furthermore, a rviz package can visualize by subscribing to any number of data topics. This enables data visualization through a plug-in (e.g. 3D Point Cloud visualization mapping).

Importantly, ROS can be set up as a **distributed** system across multiple machines. This means you can have multiple robots, ground stations, and servers located elsewhere. ROS is a very flexible structure to build your applications on.

How Packages facilitate code sharing/ reuse



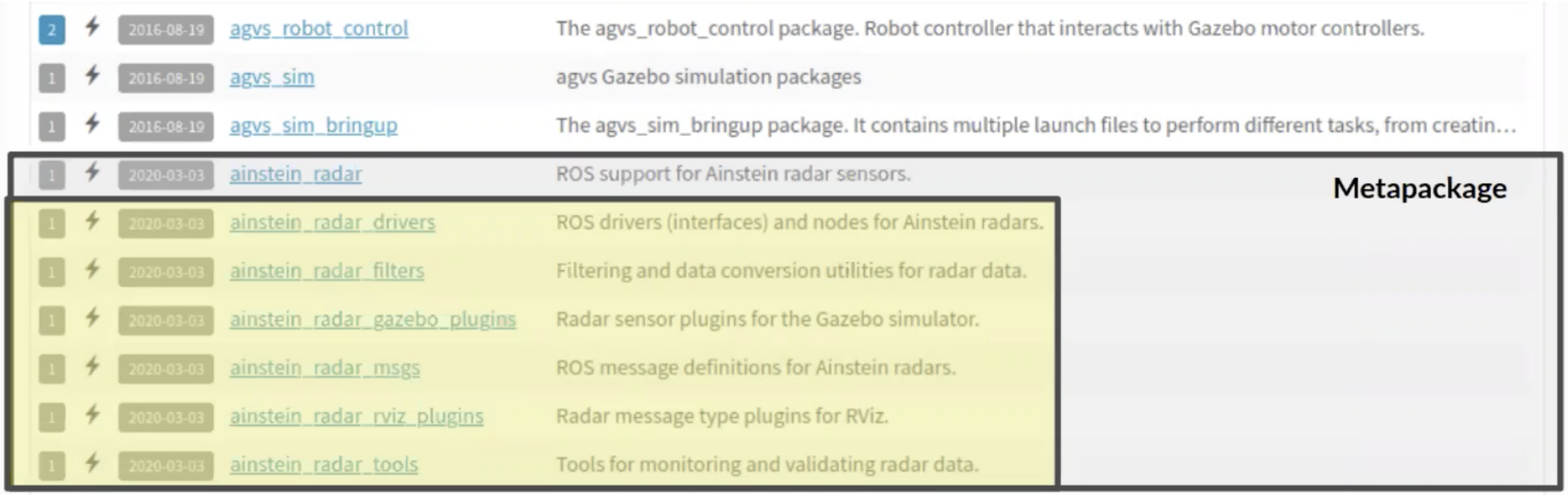
One of the core strengths of ROS is its community. In order to facilitate sharing and reuse of codes, ROS relies on packages. Think of packages as the building blocks of ROS systems providing enough complexity to be useful but in a way that is simple enough to repurpose in other applications.

NODES	MESSAGES	SERVICES
• Executable processes in C++/python (and others like java) • Subscribe to topics, perform processing, publish to other topics • Example: radar “driver”	• Describe data format for communication between nodes • Language and OS agnostic • Example: point cloud message, laser scan message, or radar target array message (like Ainstein does!)	• Blocking RPC (request-response) for simple actions • Example: camera calibration routine.

Nodes communicate via messages (to note, nodes and messages are the building blocks of packages)

Individual packages that share something in common are called metapackages. These metapackages are stored in the ROS index for all the users across the globe to use, search for, document, and so on.

So in reallife, if you search though the metapackages that are available on ros.org (1200+ available), it would look something like this:



These packages are released onto the internet for others to use. What you see above is the metapackage for our radar sensor and then, within the metapackage subpackages for drivers, filters, messages, generic tools, gazebo plugins, etc.

TL;DR: This is how the world shares the cool stuff their working on with other engineers. :-)

Benefits of being a ROS member? The indexing has helped our organization, Ainstein, be one of the top search results for ROS radar!

So what about package releases… is there a process?

- Develop on GitHub
- Index on ros.org
- Release with bloom (release automation tool)
- Install from package manager (e.g no cloning or manual building needed)

Naturally, you're going to need multiple packages

You'll often be running nodes from different packages simultaneously -some of which are interdependent. For example, you might want to have a couple of radars on a robot feed that information from the driver node into a SLAM node which provides localization to mapping for the robot which gets fed to a navigation node which plans a path and sends control commands down to the motors. ← That's how a bunch of different packages can be strung together in order to accomplish something.

That whole process get organized by a catkin workspace.

Catkin workspaces:
Packages are organized into workspaces. The catkin buildtool is a thin layer on top of seamake that traverses packages and looks for configurations to build them in order according to dependencies. You keep all of the packages for your application in catkin so they can be built simultaneously.

Launch files (i.e. scripts)
If you want to run nodes from different packages simultaneously you'll use the launch files. These are written in XML format and can run multiple nodes in sequential order. They also allow for simple topic remapping and parameter loading. Launch filed are essentially a high level interface for running multiple nodes at the same time from different ROS packages.

This example is of a few metapackages with their own packages and some other packages loosely tied together. The combination of the catkin workspace and the use of launch files helps make this operation much simpler to execute.

Another very useful tool ROS provides is urdf (Unified Robot Description Format) which is a language based in XML format for describing the geometry of a robot. URDF will describe how the different links of a robot are related to each other through different joints. For example, you could launch a sensor node if there's a connection seen. Through URDF, you can specify that it's on node 4 and the data then can be transformed over to any other link frame. This removes the need to do an unnecessary 3D geometry.

If you have a bunch of sensors on a robot, URDF is a huge time savor if used properly.

Also, URDF is not just for robots! Imagine a camera on a ceiling with a radar monitoring a robot on the floor doing navigation. Using URDF, you can tie the information from the robot, camera, and radar altogether.

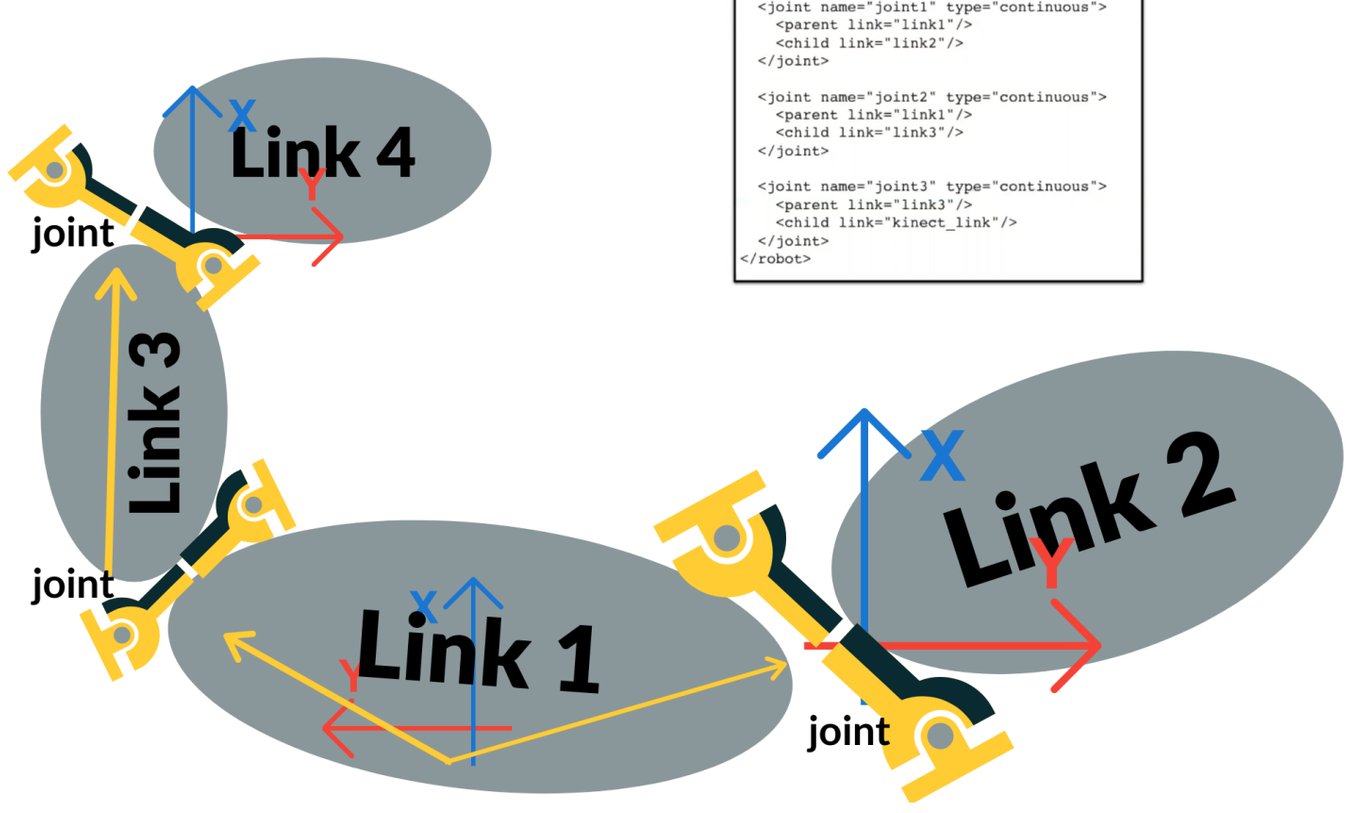
Another example would be to use a URDF for an entire building by tying different link frames to different rooms describing where you have different cameras to visualize the data. URDF is really just a way to manage distributed sensors among different frames.

```
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="kinect_link"/>
  </joint>
</robot>
```



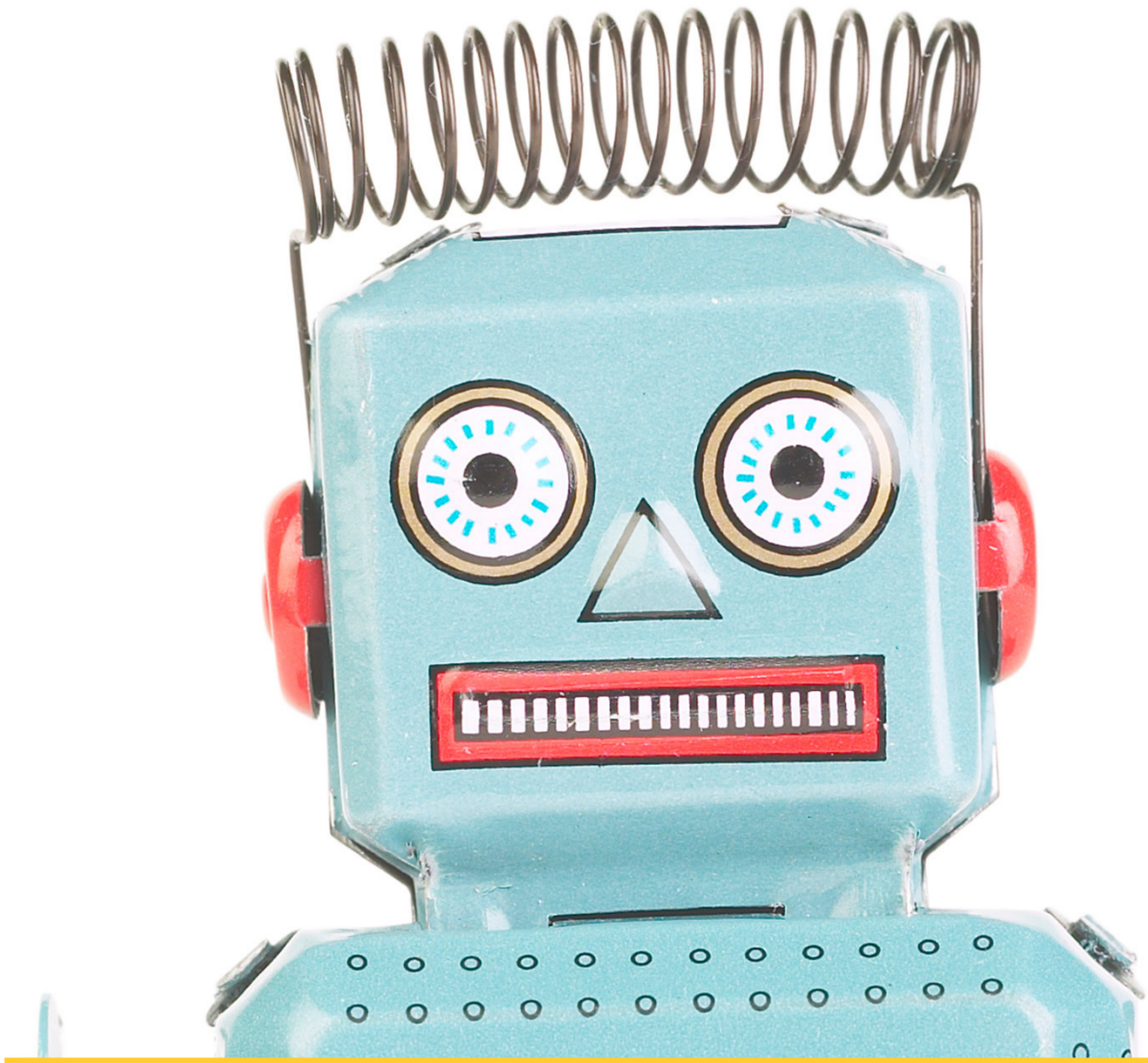
In Summary

- Fast prototyping
- No need to worry about time synchronization, data recording/playback, logging, etc. It’s all under the hood.
- Easily define complex coordinate frame structures, automatic frame conversions with tf package.
- Extensible debugging tools:
  - RViz for 3D visualization
  - RQt for error monitoring, logging, plotting
  - Dynamic parameter server for “live” tuning
- Native support for conversion to OpenCV, PCL, etc full 2D and 3D sensor processing stacks.

Ainstein’s ROS-compatible smart radar systems allow you to take control for streamlining integration of our radar sensors into your own custom applications.

ROBOT OPERATING SYSTEM

AN INTRODUCTION TO ROS



AINSTEIN

ADDRESS:
2029 Becker Drive
Lawrence, KS 66047 USA

EMAIL:
hi@ainstein.ai

PHONE: 785-856-0460

About Ainstein

Our mission is to enable safer driving, flying, working and living through radar-based technology. We are in the business of improving safety and protecting valuable assets through innovations in radar technology.

Ainstein makes radar systems smarter, more affordable and easier to deploy. We offer complete solutions for autonomous drones, advanced driver-assistance systems (ADAS), autonomous vehicles and industrial sensing – incorporating a combination of millimeter wave (mmWave) radar, sensor fusion and artificial intelligence (AI).

For years, cost, weight and performance constraints have hindered the wider adoption of radar. Ainstein makes radar systems accessible to everyone by overcoming these constraints. One recent innovation: we've developed the world's first UAV collision avoidance radar with 3D detection.

Radar systems and the sensor data processing intelligence are they keys to our autonomous future. We offer a deep scientific, mathematical and engineering expertise along with full spectrum portfolio (24GHz, 60GHZ, 76-81GHz) of hardware and software to support our customers in developing highly customized solutions with unmatched precision in unpredictable environments.

Our core team has more than a combined 100 years of experience I radar research and development with deep knowledge gained through projects funded by NASA, the U.S. National Science Foundation (NSF), the European Space Agency and others.

Other radar companies are at least two to three years behind Ainstein. Startups have been slow to market and are unable to produce at scale, while established companies are slow to adopt the newest technological innovations.

Ainstein products can be fully customized to specific application requirements, have unmatched precision in ALL weather conditions and surface types, and are a fraction of the price of competitive products.

Visit our website (www.ainstein.ai) for more information, or get in touch with Andrew Boushie, Vice President for Strategy and Partnerships, at andrew.boushie@ainstein.ai to arrange a phone call.

AINSTEIN