Interview Questions as Robotics Developer

Generally the 3 topics a robotics developer will be interviewed on will be google style data structure+algorithms questions, C++, and your robotics specialization.

I would say generally it is 60% google style, 10% C++ and 30% robotics. Though the exact numbers can vary from company to company. Since the google style can be found online. I'll just give you some of the robotics questions I have been asked in the past. Note that my specialization is SLAM. So the robotics questions tended toward that. You would of course get different questions if your background was in controls, or ML,... .

* Explain the parts of a SLAM system.
* What are the properties of a rotation matrix? \*What are the epipolar lines and the epipoles?
* Why is SLAM more accurate then odometry?
* Dense and sparse SLAM differences
* How to estimate scale?
* Should you include scale in your Kalman filter?
* Some discussion about error accumulation regarding computing position via the same landmarks
* How did you evaluate different slam systems
* If sensors are at different time steps how do you fuse them in a factor graph
* Some discussion about rolling shutter/distortion.
* discussion about superpoint
* What is a factor graph and how does it work
* Why would you pick a different decomposition method?
* What is EKF and UKF?
* Name the parts of a Kalman filter
* What are the different ways to store rotations, and the advantages/disadvantages
* Describe your SLAM system
* How do you marginalize in information form
* How to do it in a Kalman filter
* Explain what are the observability issues in SLAM and how to fix them
* Information or Covariance form
* What is the observability of the IMU biases?
* Write down the relationship you would use to estimate Camera-IMU extrinsics

**ROS for Robotics**

I use use ROS at work all the time I'd say like most things it's best to start with some small projects that interest you. Gazebo is an OK platform to test out some proof of concept ideas if you want simulations. For learning, this course I found to be fairly helpful in a general overview of the concepts and tools of ROS <https://www.youtube.com/playlist?list=PLRG6WP3c31_U7TFGduEIJWVtkOw6AJjFf>.

I'd say if you don't have a basic understanding of c++ and working in a Linux environment then to follow some online tutorials in those first. Pluralsight has many courses that can be free if you continuously sign up for free trials every 3 hours of lecture.

For full disclosure I took a ROS class in college senior year so I already was pretty proficient in robotics to begin with. But if I was to do it on my own I would probably buy a Raspberry pi or equivalent micro controller and look up ROS tutorials of people using them. As previously mentioned Gazeebo is really good for simulations, you can import models found online and have them subscribe to your topic names to control your motion. Also turtle Sim is a 2d super simple simulator for twist messages(basic motion).

Essentially how ros works is through publishers and subscribers. You make your robot subscribe to topics that you are publishing that tell it how to react. So what you want to make is a program that outputs those messages! For an easy proof of concept project write a program that publishes a twist message that continuously changes and hook it into turtle Sim. Ample documentation online is available for this project but if your having trouble finding some pm me and I'll shoot you some links.

Hope this helps if not let me know

My undergrad senior design was based on ROS. The OS is honestly kinda outdated. But there are a lot of benefits like the versatile packages that come with it so you don't end up doing much heavy lifting. So if you're interested, I would say some good prereqs are learning Linux (I wouldn't recommend the Windows version at all), Python or C/C++ and the ROS wiki tutorial. Don't skip the tutorial like I did! The biggest problem I had with ROS were the constant compatibility issues and configuring them for my project. You can also look into Nvidia Isaac which is a lot newer than ROS BUT very little documentation.

*ROS is not an OS. And it's not outdated, it's currently the best ecosystem for mobile robotics and getting an update every year.*

*There are some packages that are outdated, yes.*

An MPU is a motion processing unit. Its just a sensor with an accelerometer, gyro, and a compass (sometimes). A popular cheap one would be the MPU6050, I think it's like $10 for the breakout board. I2C is a hardware communication bus that uses two wires to transmit data. It's how your raspberry pi would talk to the MPU6050 and get information back. Other hardware communication busses are SPI, CAN, and a serial connection. Learning how to write code to talk to things is a great first step into robotics.

Yeah using a raspberry pi is how I learned to use Linux. I'd suggest installing a headless version of raspbian and ssh into it to work with it. But yeah there are a ton of simple spark fun or adafruit tutorials out there, I got started by doing ones I thought were interesting and eventually went on to compete in a robotics competition for underwater submarines. You should check out if your school has a robotics club. Getting hands on experience might be a better learning tool for you.

A combination of many things. Ros is probably fine for research, but it encourages really sloppy programming practices and is unsuitable for anything but prototypes. It's badly coded internally and is generally unreliable. Nothing else includes all of ROS's capabilities, because ROS does everything. In a more professional setup you'd probably use a combination of nng for networking and flatbuffers (or maybe protobuf) for serialization and messaging. Or maybe Cap'n proto if you wanna get esoteric and have really nice RPC functionality. ROS 2 is a bit better, but still not brilliant. ROS 1 is a bit of a trash fire IMO. Among other things:

* Having every node in it's own process means that it's nearly impossible to run your whole system under a debugger. It also means every node creates at minimum 6 thread, which causes tons of context switching and is terrible for performance
* The time library has got some subtle and awful bugs with time conversion. Aside from storing time as 2 32-bit numbers (seconds and nanoseconds separately) which just straight wastes 2 bits and causes some awful slowdown when doing arithmatic with time values, there are differences in the python implementation with how floating point rounding is handled that can cause the same value ready by python and C++ to produce different values in rare cases. Do yourself a favor and use std::chrono::nanoseconds or abseil's Time library
* Ros Message format does not support unions, enums, field deprecation, optional fields, or numbered fields to allow for forward-compatibility, unlike basically every other serialization format available (see, flatbuffers, protobuf, capn' proto). Support for message schema evolution is awful, the bag fixing system does not scale well long term. Also, in anything performance critical or large messages, you want a zero copy serialization scheme, which of these only flatbuffers and capn' proto allow for.
* Ros Transport is a mess, allowing messages to be dropped optionally but using a TCP backing even within a local machine which is the slowest option available. Also, It's full of threading issues. Internally they use volatile to mean atomic, which it doesn't. TSan complains a lot. in the places where they do use mutexes, the use them confusingly, sometimes incorrectly. I haven't been able to confirm it's because of threading issues, but in my experience, Ros will randomly drop 1 of every 100,000 messages or so, regardless of queue size
* Global state everywhere make startup and shutdown unreliable and non-deterministic. The order in which the nodes in your system come on line and start talking to eachother matters, and ROS doesn't give you a good way to control or order this.
* Params are overly limited. They're based of XMLRPC for some ungodly reason, meaning that you can't store a 64bit integer unless you store it as 2 separate 32 bit numbers or a string.

The list continues. TLDR: If you have to use ROS, use ROS2. Do yourself a favor and get a good book on design patterns, and use NNG & Flatbuffers for serialization & message transport, Abseil or Folly for general utility stuff, and other libraries you can find on the awesomecpp list for everything else. If you need a python binding, make one with pybind, don't just re-implement your protocols in python like ROS did.

edit: Also, and this is just icing on the cake, it irks me to no end that ROS is not an operating system and is not suitable for use in safety or performance critical robotics.