Ardalan Tajbakhsh Follow
Jan 25 · 9 min read · Listen

How to Become a Robotics Engineer? (Part 1/2)

Overview

There are many online resources on how to interview for software engineering positions and learn the foundational skills for industry roles. However, there is very little information on #roboticsengineering roles which makes it challenging for #newgrads looking to land their first job. I would like to start a series on this and take step towards closing this gap with the help of YOU !

Regardless of your area of interest, there are some foundational skills that every robotics engineer is expected to have a solid grasp on. Here is what I have found to be important:

- C++/Python programming
- Working knowledge of ROS
- Working knowledge of at least one simulator (Gazebo, PyBullet, Isaac Gym, etc.)
- Understanding of coordinate frames/transformations, kinematics, and dynamics of multi-body systems
- Basics of optimization, linear algebra, statistics and probability

With the foundations in your toolkit, the next step is to develop deeper knowledge of at least a single area/role. Here is a list of common industry roles:

1. Motion Planning Engineer
2. Controls Engineer
3. Perception Engineer

Note: Sometimes depending on the company/project motion planning/controls or perception/localization can be part of the same role.

Programming

C++ and Python are the most common languages used in most robotics projects. Some other languages like Go and Julia have also been getting some traction recently but I won't comment on those as I do not have much experience working with them. My recommendation is to start focusing on C++ first as it is used in most large-scale robotics projects and covers the majority of technical interviews.

First, get comfortable working with the fundamental data structures including vectors, queues, maps, graphs, sets, and heaps. Most algorithms can be implemented with a combination of these data structures. I suggest taking "Mastering data structures and algorithms using C and C++" course on Udemy for a comprehensive overview.

Second, learn to use the standard template library (STL) instead of re-implementing common operations on containers. Things like searching for an object in a vector and deleting it, or filtering a container
based on some criteria. Do not get overwhelmed trying to learn everything STL does in one go, instead every time you need to implement a common operation on a data structure, see if STL has an implementation for it already.

Note: C++20 has brought the ranges library which offers more functionality than STL with cleaner syntax, but it has a bit of a learning curve, so STL is a good place to start (More on this in a later post).

Third, start implementing known algorithms in your area of interest with C++. I've found this method to work best for me personally as it helps with learning the nuances of the language by doing. For example, if you are interested in motion planning, try implementing RRT in C++. If you got stuck, look up on GitHub for an existing


Fourth, write unit tests for every algorithm you implement. Learn to use Google test (gtest) framework for writing unit tests. There is plenty of online documentation on gtest and how to use all the functionality it offers.

At this stage, you should be comfortable implementing and testing algorithms in C++. Going from C++ to Python should not be difficult at all as you've already mastered the hard part. If you are doing machine learning / reinforcement learning work, Python may be more central to your dev process. Otherwise, I suggest using Python as a prototyping tool to quickly test ideas and analyze data before implementing your solution in C++.

Robot Operating System (ROS)

Learning ROS is particularly important as it teaches one to think at a macro-level about any given system. I suggest learning ROS even if the position you are targeting does not require it.

One common pitfall when learning ROS is trying to do too much at once. There are many different terms, concepts, and nitty gritty details which can be quite overwhelming for someone with no prior experience. "One bite at a time learning by doing" approach has been the most effective for me personally.

First, learn the alphabet of ROS by doing small demos. For example, setup your catkin workspace environment, write a ros node that prints something and get it to build. Then define a topic of a certain type and publish a message to the topic. Have another node that subscribes to the same topic. Now, write a launch file that launches both nodes. Log the messages passed inside the topic in a rosbag file. This way, you will see tangible progress every step of the way. The key concepts to start focusing on are ros project structure, nodes, messages, topics, services, and bags. This should give you the necessary foundation to contribute to ROS projects.

Second, start taking on small features within a ROS project in your research group, class


touch different parts of the project to understand the high-level connections better. You will be surprised how far this will take you if it is done consistently.

Note: Starting to contribute to ROS projects will also teach you how to ramp up on a new codebase, which is critical for industry roles.

Third, clearly define what you are trying to do before starting. It may sound obvious, but it is really easy to go down the rabbit-hole and jump around in documentation for hours. One thing that has helped me is to write down what I want my feature/program to do with great level of detail ahead of time. This keeps your attention focused on finding the info you need and eliminating the noise. A specific description can be something like "I want to write a node that takes in position measurements from motion capture, estimates velocity, and publishes that estimate to another topic".

Fourth, rinse and repeat ! No matter how long you have been using ROS, there are still things you may not know. Nothing replaces the hours you spend debugging and learning the details !

Kinematics and Dynamics

Up to this point, you should be able to develop algorithms with C++/Python and tie pieces of logic together with ROS. The next foundational skill is understanding coordinate frames and transformations, kinematics, and dynamics of multi-body systems. This is essential whether you are interested in motion planning, control, or perception.

First, if your school offers a comprehensive grad-level class covering these, consider taking it. Otherwise, I suggest taking the "Modern Robotics Specialization (Courses 1, 2, and 3) from Northwestern University" on Coursera. As a robotics engineer, you should have a solid grasp on configuration spaces, translational and rotational motion representations, forward kinematics, inverse kinematics, Lagrangian dynamics, and constraint representations.


kinematics). Write a function that computes the possible joint configurations given a desired end-effector position (Inverse kinematics). Now write a function that computes the accelerations in configuration space given some acting force at the end-effector (Forward dynamics). Integrating the acceleration will give you the velocity and position in the configuration space. This is all you need to simulate a robot ! Feel free to explore the same concepts with different platforms (Wheeled, aerial, etc.). These concepts are foundational in understanding robot motion. Take the time to understand them well before getting into more advanced concepts.

Third, once you are comfortable with these concepts, learn the common packages and libraries that allow you to perform these operations quickly. You probably do not want to derive them by hand every time. I recommend learning #sympy which is a powerful Python library for symbolic math operations (Checkout https://sajidnisar.github.io/posts/python_kinematics_sympy for a nice example). Also get comfortable with #numpy and #scipy (Specifically the integrate module) for implementing matrix operations and simulating dynamic systems. Once you are comfortable prototyping in Python, spend some time porting over your code to C++ ! You can leverage the Eigen library for implementing matrix operations in C++.

Note: MATLAB offers many tools with similar functionalities. However, Python is used significantly more in industry as it is free and has a massive support system. They key is learning the concepts, you can always use a different tool if necessary.

Simulation

This is one of the more controversial topics of this series as there are many options to consider. I would say that everything here is purely a reflection of my own experience and am completely open to hear your feedback in the comments. To become a robotics engineer, you need to be able to comfortably work with at least one simulator.

For single-agent applications with complex dynamics, #gazebo is a good place to start. I

better support for learning projects. I recently heard lots of great things about #isaacsim from NVIDIA and would love to learn about it more from you.

Unless you are targeting “Simulation engineer/dev”, having a basic knowledge of a simulator to the extent that you can test your code would be sufficient. There are normally people dedicated to developing and maintaining simulators in industry (A big shoutout to all of them !).

There are three steps you need to master to work well with a simulator.

1. **Environment setup**

As a developer, you will have to run many experiments to test your ideas and solutions. As a result, it is important to understand how to setup the simulator environment to match your expectation. This includes defining the environment map, rigid bodies, obstacles, sensors, friction parameters, external forces, and more. For example, if you are evaluating the robustness of your motion planning algorithm for an indoor application, you will probably need to simulate your agent with many different obstacle configurations. If you are testing a controller on a quadcopter, it helps to understand how to apply external disturbances. You will learn bits and pieces like these as you go depending on the project.

2. Understanding the interfaces

Every simulator has a set of APIs that allow you to interact with it. They define the expected data format from any external client. For example, gazebo uses #roscontrol as one way to interface between robot commands and the simulator.

3. Data logging and visualization

At the end of the day, the output of the simulator is what you care about. Depending on your use-case you may want to visualize the experiments as you are running them, or log the results and analyze after the fact. It is key to understand how this process works for your simulator. For instance, gazebo has its own GUI that lets you visualize topics or log


Soft Skills

A top notch robotics engineer must be an effective multi level thinker and communicator, master learner, and adaptable to constant change.

Multi-level thinker and communicator

One unique aspect of robotics engineering is it's very multidisciplinary nature. Navigation, perception, control, localization, software infra, product, hardware, etc come all together in one solution. Almost no part of a robotic system can be developed in isolation. This requires one to understand the project at different levels.

For example, in a dynamic sensing application navigation features are very coupled with perception (We have to move in a certain way to see what we need to see). This means that the navigation robotics engineer should understand the perception module, including it's capabilities and limitations. Same type of analogy goes for many other roles. The technical features can also have operational impacts that developers should be aware of. For example, using fleet of robots to replenish a warehouse completely changes the operational workflow. It is up to you to understand these nuances and communicate with the right people. One strategy that helps with developing this kind of thinking is to understand how your specific feature contributes to the overall system and what components it impacts.

Master learner

Regardless of background and experience, to become a robotics engineer you'll have to make learning a habit. This spans anything from dedicating a set amount of time to review papers to learning new software tools. I suggest spending at least 1 hr everyday on learning and making it a ritual. This time can be spent on learning something relevant to your current task/project or expanding your skill set into a new area. A little bit everyday goes a long way.

Adaptable

With constant change in development paradigms, tools and frameworks, and customer requirements, it is critical to adapt fast and be comfortable with uncertainty. As a robotics

time and all we can do is to learn from them and move on. What's important is to measure the reality objectively and adjust if necessary.

Now that you have the foundational skills under your belt, it is time to go through the roles in the robotics industry and the specific skills they require. Checkout part 2 https://medium.com/@ardalantj/how-to-become-a-robotics-engineer-part-2-2-9bdc3f15f60e.