Ardalan Tajbakhsh Follow
Jan 25 · 8 min read · Listen

How To Become a Robotics Engineer? (Part 2/2)

In the last part of this series (https://medium.com/@ardalantj/how-to-become-a-robotics-engineer-d7a9678fc1f8), we covered the foundational skills every robotics engineer is expected to have and how to acquire them effectively. In this article, we will focus on the current industry roles and dive deeper into their specific requirements.

Motion Planning

Robotics engineers with focus on motion planning are responsible for delivering algorithms that generate behaviors for robots. Planning layer is essentially the story teller that comes up with a sequence of desired actions given the task definition, description of the environment, kinematics and dynamic limits of the robot(s), and state definition.

For motion planning roles, in addition to mastery of C++ data structures (Graphs in particular), deep knowledge of search-based (A*, D*, etc.), sampling-based (RRT, RRT*, PRM, etc.), and trajectory optimization methods (Direct collocation, multiple shooting, etc) is necessary. You need to understand the application and limitation of each method and be able to implement these algorithms for a specific problem. To develop a better intuition on various planning algorithms, I suggest reviewing planning literature in diverse applications (For example, legged locomotion, aerial navigation, autonomous driving, etc). By doing this consistently, you will start to see which approach makes sense

asked to perform a coding task in C++. This can be either live or online to evaluate your ability to write code on the spot. The next step is either a take-home assignment or a phone interview with a senior engineer. The take home assignment may be a mixture of literature review and some implementation of a planning algorithm. This is to evaluate your depth of knowledge and ability to perform non-trivial tasks within a given timeframe. After the take home assignment, there will be a panel interview where you will have to present your solution to the assignment and answer detailed questions about the assignment or specific planning scenarios.

The panel interview can be quite intense as you may be asked a variety of planning problems. The key is to recognize that you are not expected to know the answer to every problem, but the company is evaluating your approach when facing a novel problem. My recommendation is to make the panel interview a collaboration. Propose your solution, get feedback, refine it further, clarify the details, and refine some more. You are trying to demonstrate that you can solve problems and iteratively improve your solution, not that you know everything.

Control

A robotics engineer with controls expertise is responsible for development, testing, and deployment of algorithms that allow the robot to track desired motion plans within the physical limits of the hardware while maintaining stability. Controls engineering requires strong foundation in multi-body dynamics, optimization theory, linear systems theory, and hardware implementation.

Multi-body dynamics

We have covered this in the robot motion foundations, but a robotics controls engineer needs to be comfortable computing equations of motion and be familiar with model simplification techniques. Part of this comes from developing a strong theoretical foundation through cousework, and another part from practicing dynamics modeling on

systems theory. Specifically, linearization of multi-input multi-output systems, solutions to linear time invariant (LTI) differential equations, properties of linear dynamical systems (Controllability, observability, and stability), and design of feedforward/feedback and optimal controllers and observers (Pole placement, LQR, Kalman filter, etc). Most of these concepts are covered in a first year grad-level course in linear systems.

Optimization

Most modern controllers like model predictive control (MPC) are optimization based. It is critical for a robotics controls engineer to have a deep understanding of formulation, implementation, and debugging of various types of optimization problems (Linear/nonlinear,
stochastic/deterministic, convex/non-convex, etc). Like the programming part, I suggest learning to implement optimization problems in Python (Pyomo) or MATLAB (Quadprog) and then transition to C++ (IPOPT). I suggest checking out Stephen Boyd's convex optimization course for a comprehensive review.

Hardware

Having extensive experience working with hardware may not be required, but it certainly helps in landing industry roles. Understanding how to tune a controller based on the hardware response, limitations in update rate and communication, and behaviour of various motors and actuators are incredibly valuable and rare skills.

Computer Vision

Computer vision is growing very rapidly and industry roles are becoming more specialized as the scope of perception tasks increase. However, the foundational skills are relatively consistent across different projects and companies. Perception engineers are expected to have a solid understanding of different types of cameras, camera calibration, image processing techniques (denoising, image restoration, filtering, etc.), object detection, object classification, semantic segmentation, and object tracking.

foundations and understand applications of neural networks (CNNs, GANs, RNNs) to the mentioned vision tasks. I think everyone agrees that Andrew Ng's Deep learning specialization is the way to go here !

Once you are familiar with these concepts, start doing small projects with a Raspberry Pi and slowly add more complexity. For example, start with an out-of-the-box object detection solution from Github and get it working with your camera. Then change the lighting condition to break the solution and try to fix it with additional image preprocessing or training with more data. Next, you can try tracking an object and estimating its 3D pose. Doing small projects early will solidify the concepts that have been learned in your coursework.

Most industry roles in perception require working knowledge of a deep learning framework like Pytorch or Tensorflow, so start learning these as you are doing small projects. There are tons of online resources that cover either of them in great detail. The goal is to master the process of developing a training pipeline for a given task and improve the accuracy over time while maintaining robustness to different conditions the model is expected to work in.

Note from a friend in CV: Real-world perception is not just about running pre-trained models. Most of the problems arise in handling edge cases, various lighting conditions, camera vibration, etc.

Note from another friend in CV: Read research papers frequently ! You may not directly use a paper, but understanding the underlying methodology helps with thinking about your own problem from different angles.

Sensor Fusion

State estimation and localization (Referred to as "sensor fusion" in job postings). These roles are less common compared to the other three as estimation can be part of control or

State estimation in a nutshell is about generating accurate hypotheses of robot states using the system model and sensor measurements. A robotics engineer working in state estimation needs to have a strong foundation in statistics and probability theory, optimization theory, knowledge of core state estimation frameworks, and experience with common sensors used in robotics.

Statistics and Probability

Solid understanding of probability density functions, random variables, distributions and sampling, and Bayes rule come up many times in state estimation literature. If you feel a bit rusty about these topics, I suggest taking a refresher course that covers the basics. Once you master these concepts, you will be able to read various literature in state estimation and develop a higher-level understanding. The next step would be a more advanced grad-level class in SLAM that covers major algorithms and use cases.

Optimization

It is important to recognize that estimation and control almost exactly solve the same problem mathematically. As a result, most of the optimization machinery used in control theory applies to estimation as well. Most modern SLAM algorithms solve an optimization problem over a factor graph. In proprioceptive estimation, modern methods solve a QP over the history of state measurements to generate the most likely estimate of the current state. So, spend time mastering formulation, setup, implementation, and debugging optimization problems. They are everywhere !

Estimation Frameworks

With the mathematical foundation under your belt, you can start learning the core algorithms that are commonly used for state estimation in various applications. These include Kalman filter and its variants (Extended and unscented KF), particle filter, Fast SLAM, and ORB SLAM to name a few. Again, implementing a few of these by yourself is the best learning strategy in my opinion. One thing that helped me with the understanding of state estimation was implementing a Kalman filter on a small segway-style robot using an Arduino, wheel encoders, and an IMU to estimate the orientation for balance control.


Software Infrastructure

Software infrastructure is the back-bone of every robotics project. Without software infrastructure, none of the algorithms we have discussed so far will be realized on hardware. A robotics engineer with focus on infrastructure should be very strong with C++ libraries and frameworks, software architecture design, design patterns, clean coding practices, and communication.

Deep C++ knowledge

C++ is one of the key tools in every robotics engineer's toolbox. However, software infra engineers will need to have mastery over nuances of the language, common libraries, and complex system debugging. You are expected to help integrate new libraries into the codebase, identify areas of refactoring, dockerize services, improve the usability of the software, maintain CI/CD pipeline and many more. Getting ramped up on a new codebase helps a ton with understanding many of these tools and processes. You can start with contributing to an open source project related to your area of interest and go from there.

Architecture Design / Design Patterns

Understanding common software design patterns is critical in large scale projects as the codebase is constantly being extended which requires the underlying structure to support it. Without design patterns, developers need to write more code than necessary and it becomes very difficult to maintain the project. Refactoring guru is a great resource for understanding fundamental design patterns in C++ (https://refactoring.guru/design-patterns/cpp). To learn design patterns, I suggest trying to refactor some old written code using one of the common patterns (For example, factory, builder, etc). Another important skills here is separating out the business logic (Algorithms that do stuff) from interfaces and base types so that they can be managed and extended somewhat separately.

Software Maintenance

Every robotics software project requires constant maintenance as the codebase grows


keeping track of these (They are often overlooked !) and communicate any necessary actions to other developers. The less time you spend maintaining your codebase, the longer it will take to develop new features !