Key considerations when choosing a robot's operating system

August 2018

Introduction

Robots have been the subject of science fiction films for decades, but with technological advances, including the rise of internet connected devices, a real robotic revolution is beginning to emerge. Converging with the growth of artificial intelligence and machine learning, robots are being adopted in every sector imaginable, including consumer, industrial, healthcare, and retail. According to IDC, worldwide robotic spending, encompassing hardware, software and related services, is set to reach US$230.7bn by 2021.

Due to the sizeable market potential, demand, and evolution of technology, companies are seizing the opportunity to build robots. However as with many other initiatives in the wider Internet of Things (IoT) industry, there are key decisions that robotic manufacturers need to make from the development stage onwards to ensure they are building a sustainable, future-proof, and secure robot.

One of the most important decisions to be made is that of the robot's operating system. Unfortunately, this importance is not always obvious until the company is too invested to change it, which can lead to a slew of delays or issues to overcome. The operating system that's perfect for hacking things together may be impossible to maintain once the robot reaches production. Similarly, the build-your-own option means maintaining the entire operating system (backporting upstream security updates, etc.) for the lifetime of the robot. The rock-solid, stable option may end up having versions of dependencies that are too old to use. That snazzy new one created by a niche startup that seems to tick all the boxes is less appealing if they suddenly go out of business.

With all the choices available, which considerations are needed to ensure the best is selected? While there are certainly some very low-level technical considerations to make (e.g. footprint constraints or performance requirements), many of the factors influence further up the chain than just the development process. The operating system can contribute to a company's monetisation route, support offerings and security strategy. This whitepaper discusses how this decision can guide the wider strategy, and covers several key considerations that change as the product moves from development through to production and maintenance.

Development factors

The initial development of the robot is a critical time when it comes to technology selection, both hardware and software. Unless a company is in the habit of burning money, the operating system used on the robot during this stage should also be the one used in production (or at least closely related). There's a balancing act to be had here: the engineering team will naturally want the operating system that best supports whatever they're developing, but there are more factors to be considered.

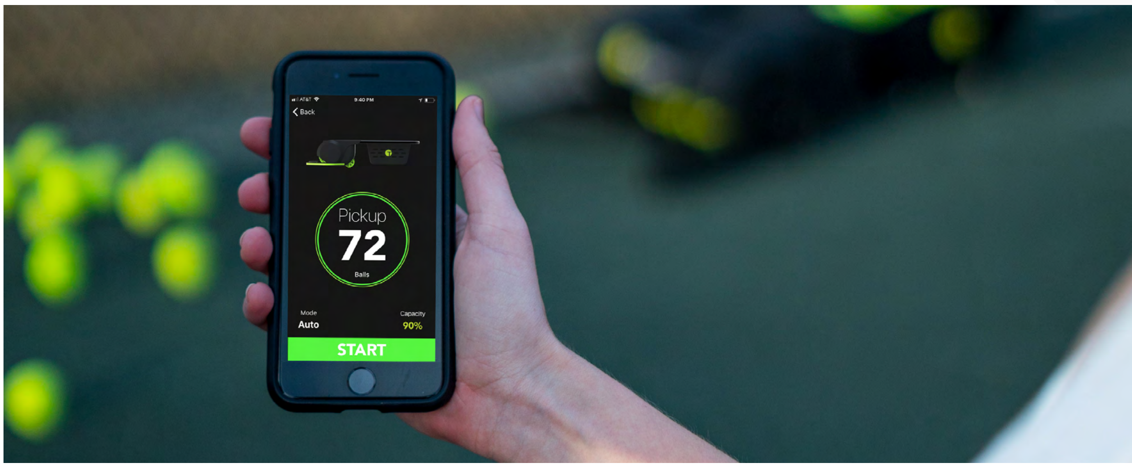
Ubuntu -

Ubuntu is an open source operating system for the desktop, cloud, and IoT devices, including robots. Ubuntu is the leading cloud guest OS, running most public cloud workloads today on Azure, AWS, Google and IBM. According to RightScale's latest State of the Cloud report, the majority of the cloud market (64%) runs on Amazon Web Services (AWS). Of all cloud instances on AWS, 56% are Ubuntu. In addition, 40% of Linux users run Ubuntu or Ubuntu Core on their IoT solution according to Eclipse's 2018 Developer Survey.

Learn more about Ubuntu **and** where it is used**.**

Software stack compatibility

Regardless of other reasons used when deciding on a given operating system, selecting one that isn't compatible with the technology required (libraries, frameworks, etc.) is a recipe for disaster. The engineering team will spin its wheels while competitors hit the market first. Perhaps required libraries are written specifically for certain Linux distributions, or the team has settled on using some middleware to enable faster development. For example, 49% of the roboticists we polled were using the Robot Operating System (ROS). If potential operating systems are evaluated with this in mind, it will help lead to a very streamlined development process.

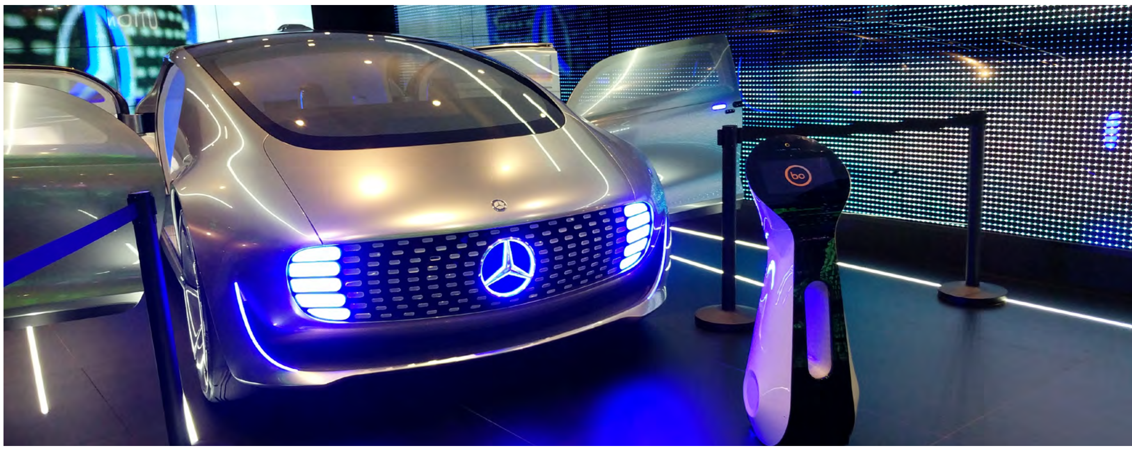


Haitham Eletrabi, CEO of Tennibot, who has developed the world's first robotic tennis ball collector, factored this in during their development:

"The compatibility with software like ROS and OpenCV makes the implementation and testing of Tennibot's algorithms an easy task. [...] In addition, Ubuntu is so versatile with different sensors and components that it really makes it the more superior option for us."

Hardware compatibility

Hardware compatibility should also be a primary concern, for much the same reason as software: a significant chunk of time will be spent ensuring components to work together before any real progress can be made on the robot itself. For example, it's not unusual to find hardware that requires drivers only written for specific Linux distributions, or to work with vendors that have limited exposure to Linux in general.



Adrian Negoita, CTO of BotsAndUs, who are pushing the boundaries of customer service robotics, realised this early on:

"Ubuntu ticked every box. It has the outstanding hardware support we were looking for, and thanks to Ubuntu's popularity in the market, most vendors have plenty of experience in integrating with Ubuntu-based systems. This makes it much easier for us to select additional components and solutions[...]."

Development team familiarity

Speed is a big deal when developing any product; market leaders can easily be determined simply by who gets there first. When a team is considering programming language options for a new project, the decision is heavily influenced by the team's familiarity with said language. This isn't necessarily because the team is resistant to change, but because they know they can produce higher-quality work in less time if they can use a familiar language. A similar consideration must be made when looking at operating systems: if the engineering team isn't already familiar with it, time to market will be delayed as they learn its ins and outs.

Snaps and Snapcraft -

Snaps are containerised software packages designed for Linux systems across cloud, desktop and IoT devices. They are safe to run, bundle their dependencies, and feature automatic updates. Snapcraft is the command line tool for writing and publishing your software as a Snap.

Learn more about Snaps and Snapcraft**.**

Ease of system integration

A robot is rarely a standalone device: it often needs to seamlessly interact with other devices. The other device may be as simple as a digital twin for hardware-in-the-loop testing, but in general, off-device computation is getting more popular in robotics. Cloud robotics, speech processing, and machine learning are all use-cases that can benefit from processing information in a server farm instead of on a resource-constrained robot. If possible, it makes a lot of sense to use the same operating system on the robot as in the cloud. It prevents division of domain knowledge, and keeps processes the same, decreasing development time of both the client and server components.

The Small Robot Company, for example, are creating a range of farming robots. Rather than make one large, expensive robot capable of a number of tasks, they're making a number of small, specialised robots each of which is responsible for specific tasks. This makes the process of taking care of a field into a coordinated dance of multiple systems that need to communicate and work together. If anyone understands how important it is to have a smooth integration process between systems, it's their Chief Roboticist, Joe Allnutt:

"At The Small Robot Company we use Ubuntu on our farming robots, on their on-farm bases (kennels), on our development machines and have instances running in the cloud on AWS. Ubuntu provides a consistent, secure and well understood foundation for our systems."

Ubuntu Core -

Ubuntu Core is a lightweight, transactional version of Ubuntu for IoT devices such as robotics. It uses the same kernel, libraries and system software as classic Ubuntu and is easy to transition Ubuntu based development projects to Ubuntu Core for production.

Made entirely with Snaps from the ground up, Ubuntu Core offers automatic updates ensuring that even unattended devices can receive the latest security patches. If an update fails, it will roll back to the last known working version to minimise downtime.

Learn more about Ubuntu Core**.**

Core

Support availability

One of the many joys of engineering is problem-solving. However, every engineer gets to the point where they need some help. Where do they turn? Of course, it depends on the problem. If they need help with the operating system, they often turn to the community around that distribution, where more often than not, others are experiencing the same issues. For example, Ubuntu has community support through its security notices, AskUbuntu, the Ubuntu forums, Launchpad, as well as various IRC channels and mailing lists.

Matt MacLeod, Lead Software Architect at BotsandUs, states how this community support has been used to their advantage:

"Ubuntu's extensive community support makes it a great choice for our technical team. They're able to seek advice and find solutions to issues quickly, which helps us accelerate the development process even further."

Community support is one of the best things about Linux, but it's not always perfect. Sometimes no one responds, no one knows, or no one has time to figure it out 'right now'. What happens then? The ability to get reliable, predictable, professional support for the operating system of choice is important, from assistance in the product design all the way to extended software support.

IoT app stores -

Canonical's IoT app store enables robot manufacturers to create either a public or private marketplace where applications (Snaps) can be managed and revenue can be generated from updates, support services, new features, and even new applications.

Learn more about IoT app stores**.**

On the other hand, what if the issues have nothing to do with the operating system at all? The operating system can still be an important part of obtaining support. For example, ROS only hosts package repositories for Debian and Ubuntu, which means that the use of those operating systems are essentially assumed whenever one asks for help.

Production factors

Once a robot starts the move from development to production and maintenance, new factors come into play. If this part of the picture wasn't taken into account prior to reaching this stage, there will be pain as the company either just ships what they already have, or scramble to engineer solutions to problems that could have already been solved. This is likely to result in a delayed entry into the market, lost profits, higher support costs, and probably its own set of technical problems.

Software update process

It doesn't take long to find examples of companies who started shipping devices without considering the need to update them. The Internet of Things market is flooded with them. With the rush to get devices to market, it's not at all rare to find devices with hard-coded credentials, development keys, various security vulnerabilities, and no update path.

The Mirai botnet is an excellent example of this. It infected so many devices that it amassed the bandwidth to take internet giants like Twitter, Paypal, and Spotify offline back in 2016. How did it infect these devices? By using default, hard-coded credentials. Did the companies who created them react quickly and release updates to all those devices in order to secure them? No, in the absence of an update plan, they were forced to recall them. The cost of figuring out a good update story up front pales in comparison to that of a mass recall.

There's also the less dramatic example of customers hitting a bug, or the company wanting to enable a new feature. Being able to do this quickly and cheaply without causing customer frustration is paramount. There are a few ways to achieve this:

- First of all, if the number of customers is small, it is feasible to use a field engineer, traveling to customers and updating them. This provides the utmost control, but it doesn't scale well, and quickly becomes astronomically expensive as either the number of updates or the number of customers rise.

- A second, more common method is firmware downloads: bundling all updates into a single downloadable image which is then flashed to the device. This scales better with the number of updates or customers, but it requires the engineering of infrastructure to host the update, as well as software to robustly apply it. Also, what happens if the update bricks the robot? A recovery plan also needs to be developed. All of these things take time and money.

- The optimum solution is to select an operating system that has this functionality built into it, like those that support Snapcraft. A company using Snaps for their software immediately solves all these problems at once: they're fast and easy, use infrastructure that already exists (the Snap Store), update automatically when an update is released, and have rollback/disaster recovery built in. Ideally, though, these advantages would cover the entire operating system, not just the software the company puts on top. There's only one operating system that uses Snaps for everything, including the inner-workings of the operating system itself: Ubuntu Core.

Long-term support

In addition to considering how operating system updates are delivered, one must consider for how long those updates will be delivered. Specific versions of operating systems are typically only supported for a set amount of time. For example, long-term support releases of Android Things are supported for three years, whereas Ubuntu and Ubuntu Core are supported for five years (or even longer with Extended Security Maintenance). If the supported lifespan of the operating system is shorter than the anticipated lifespan of the robot being produced, it will eventually stop getting updates, potentially falling prey to the Mirai botnets of tomorrow. At the very least, it will force the company to devise a new plan for security updates incurring additional expense and time that wasn't accounted for during development.

Ensuring a profitable lifespan

Actually shipping and maintaining robots might be the final story for some, but it shouldn't be, and it certainly isn't the strategy necessary to retain customers and extend the robot's lifespan. How does it stay relevant as competitors come out with newer products?

The iPhone serves as a good example, here. The first iPhone, released in 2007, ran iPhone OS 1, which did not support third-party applications. How long would it have stayed relevant if they continued down that path? After the newness wore off, a vast number of users would likely have jumped to something else. However, it wasn't long before iPhone OS 2 came out, which included the app store. The rest is history.

This isn't something that will apply to all robotic platforms, but it's something that should be considered. Depending on the purpose of the robot, one could actually open it up completely, allowing control or sensor use via APIs, and then support a third-party app store of some kind (such as the Snap Store). This can increase the longevity of the robot by essentially allowing third parties to have another vision for it, but this depends on the robot being pretty general-purpose.

Even if the robot isn't general-purpose (they rarely are), an app store can open up alternative revenue streams, where new functionality can be provided in exchange for a fee, or on a subscription basis. If a company adopts a private store (e.g. a IoT app store), it could open it up to select partners as well, opening the door to collaboration and mutual benefit.

Take Rigado, for example. They create edge gateways for large-scale commercial IoT deployments. Every customer has slightly different requirements, so these gateways can actually be customised using a curated list of applications in their app store. Not only does this allow the device to be used in different industries, but Rigado is planning to extend this capability to allow business partners to provide further-customised app stores to their customers.

Conclusion

The field of robotics is moving in the same direction as other devices that make up the IoT: smarter and always connected. The importance of the operating system running on a platform like this cannot be understated. Throughout the entire lifecycle of the robot, it can be there to either help or hinder. Which one depends on whether or not the company considers the entire picture before making a decision.

If you're interested to learn more about running your robot on Ubuntu, please click here and contact us.

Further reading:

Webinar: Building a commercial robot with Open Source

Blog series: ROS prototype to production on Ubuntu Core
(A five-part blog and video series focusing on the technical details of why and how to use Snaps and Ubuntu Core for robotics.)

Case study: BotsandUs

Case study: Robotcheers

Blogs: Tennibot | SeaMachines

ubuntu®
Delivered by Canonical