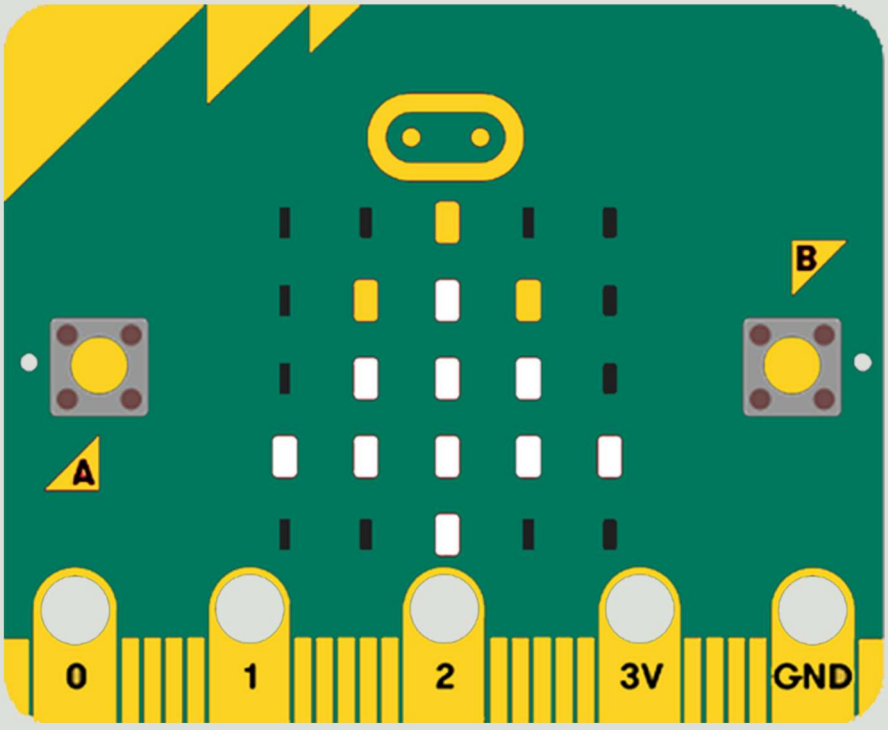
Introduction to Programing and Robotics with Micro:bit

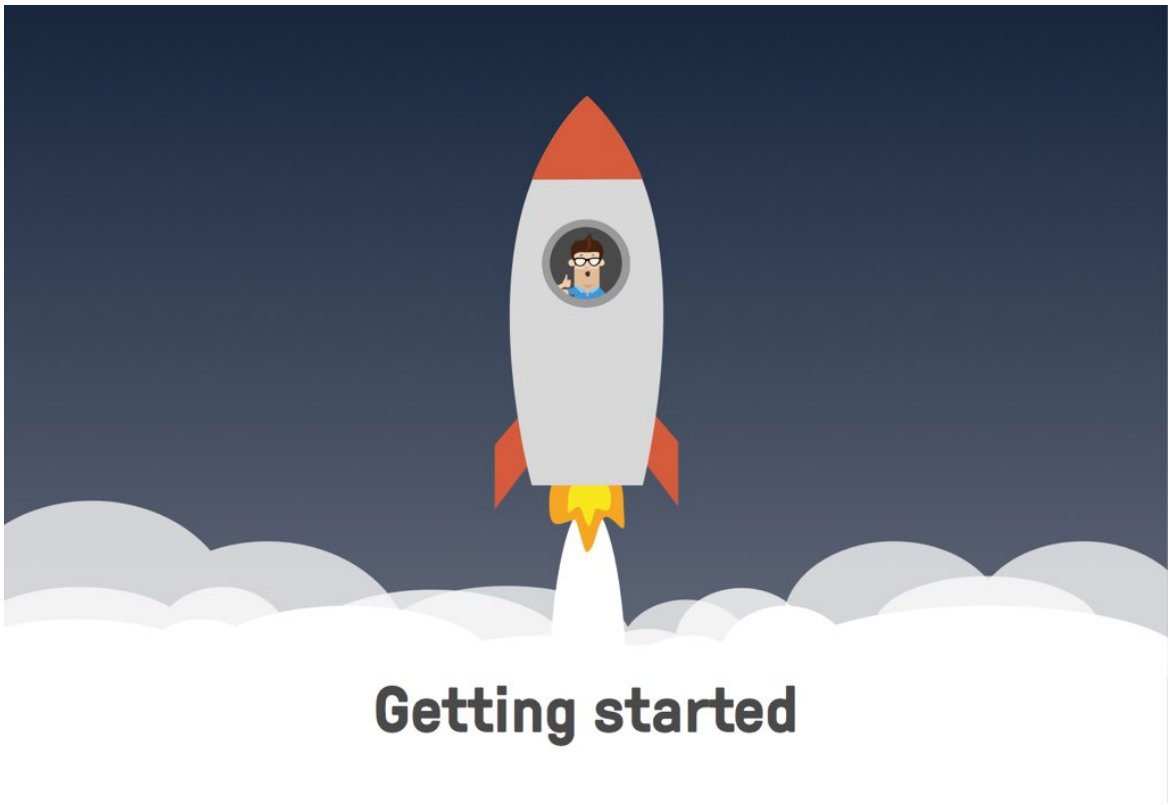
Handbook



MEET

Macau Edinburgh Exchange Tour

Getting Started



Welcome on board, joining our journey through the world of Computer Science. In these couple of weeks, we will do our very best to assist you and guide you through challenges you will face. Do not worry if you have not done any programming before because everything will be taught along the way. We encourage you to be curious in the upcoming weeks, we will try to find the best answer to all of your questions!

This handbook serves as:

- Supplementary learning material
- Future reference for ideas after the project ends

All the files related to our project and more can be found on the course website: macau.computer-science.party

During our project, we will be using the BBC micro:bits – an ARM-based micro-controller used for computer science education around the world.

Introduction to Programming with Python

Brief History of Computers

Before the 1920s, *computers* (sometimes *computors*) were human clerks that performed computations. They were usually under the lead of a physicist. Many thousands of computers were employed in commerce, government, and research establishments. Some performed astronomical calculations for calendars, others created ballistic tables for the military.

After the 1920s, the expression *computing machine* referred to any a machine that performed the work of a human computer, especially those in accordance with effective methods of the *Church–Turing thesis*. The thesis states that a mathematical method is effective if it could be set out as a list of instructions able to be followed by a human clerk with paper and pencil, for as long as necessary, and without ingenuity or insight.

Machines that computed with continuous values became known as *analog*. They used machinery that represented continuous numeric quantities, like the angle of a shaft rotation or difference in electrical potential. Digital machinery, in contrast to analog, could render a state of a numeric value and store each individual digit. Digital machinery used difference engines or relays before the invention of faster memory devices. The phrase *computing machine* gradually gave way, after the late 1940s, to just *computer* as electronic digital machinery became common. These computers could perform the calculations that were performed previously by the human clerks. Since the values stored by digital machines were not bound to physical properties like analog devices, a logical computer, based on digital equipment, was able to do anything that could be described "purely mechanically." The theoretical *Turing Machine*, envisioned by Alan Turing, is a hypothetical device theorised to study the properties of such hardware.

Programming Languages

Whenever you give a task to a machine it will not have any clue what to do by itself, you have to tell it exactly what to do. Over the time computers seem to have become smarter and now have some understanding of your work but that is just because someone instructed it that this is the way our world works. For instance, if you ask your school calculator (which is a computer too) to divide 5 by 0 you will get an error. It won't try to calculate the result because someone (a programmer) told him that division by 0 is not a valid action in the world of mathematics and informed the machine what to do in a situation like this.

When communicating with the computer, we will not only encounter the limits of its understanding, but a difficulty with simple communication to tell it what to do. In a ideal kind of world, computers would be able to understand human language (English, Cantonese, Swahili etc.). Unfortunately, this kind approach would have two major problems:

- **Human spoken languages are ambiguous**

Often in human languages, a sentence may be interpreted/understood in different ways: For instance, the sentence "Alice saw Bob with a telescope" could have two meanings:

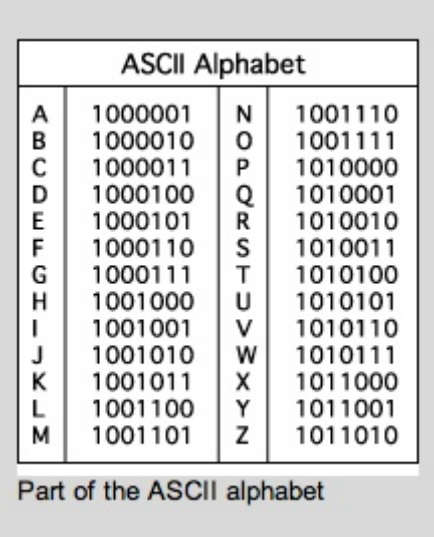
- Alice saw Bob who was carrying a telescope
- Alice saw Bob through a telescope

Both interpretations are valid. The computer would not know which one to choose in this case. There are many such sentences in English, sojust listing what todo in each situation (i.e. context) cannot be done. Besides that, there would be another problem to maintain a machine's understanding of English: Because it is actively developing, the number of valid words and grammar expressions change constantly.

- **Computers by itself cannot understand these languages.**

Under the hood of a modern computer, it only understands two states: high voltage and low voltage, its native language is binary. Binary is a special language represented by the two symbols 0 and 1. Computers understand high voltage in the

chip as 1 and low voltage as 0. From these two symbols (similar to Morse code) more complicated stuff can be built, for instance using 8 binary digits (called 8 bit or 2 bytes) you can represent all English language characters using the so called *ASCII* system.



ASCII Alphabet			
A	1000001	N	1001110
B	1000010	O	1001111
C	1000011	P	1010000
D	1000100	Q	1010001
E	1000101	R	1010010
F	1000110	S	1010011
G	1000111	T	1010100
H	1001000	U	1010101
I	1001001	V	1010110
J	1001010	W	1010111
K	1001011	X	1011000
L	1001100	Y	1011001
M	1001101	Z	1011010

Part of the ASCII alphabet

Using these binary symbols, every machine has its own instruction set for which each instruction is represented by 32 bits (64 bit for newer computers). Any of these instruction is very simple like “adding two numbers” or “store a value in the memory” but by just using these you can already create very complex programs.

The biggest drawback of a machine language is that it cannot be understood by humans. To check manually what this instruction `01001011010110010010011110010101` does for example, would mean tediously looking it up in a table. For this reason, assembly languages were introduced. Assembly language has a one-to-one correspondence with a machine language (in other words, for each of the binary instructions there is a human readable command). The main difference between it and machine language is that it encodes the meaning of the instruction which cannot be seen in machine language. The translation from an assembly language to machine language is done by a special program called *assembler* in a process called assembling (a program is writing another program, isn’t that cool?). Each different computer with its own set of instructions would need it’s own assembler.

Assembly language was a first step to a solution. It was realised that just to produce a simple task like taking two numbers from the user input, adding them up, and printing the result takes a whole bunch of assembly instructions. Hence a new level of abstraction was introduced to encapsulate the common stuff done by everyone. This led to the development of *high-level* programming language such as *C*, *FORTRAN*, and *Pascal* which are still used today. The translation from high level language such as *C* to an assembly language is done by yet another program called compiler in the process called compiling (seems familiar?)

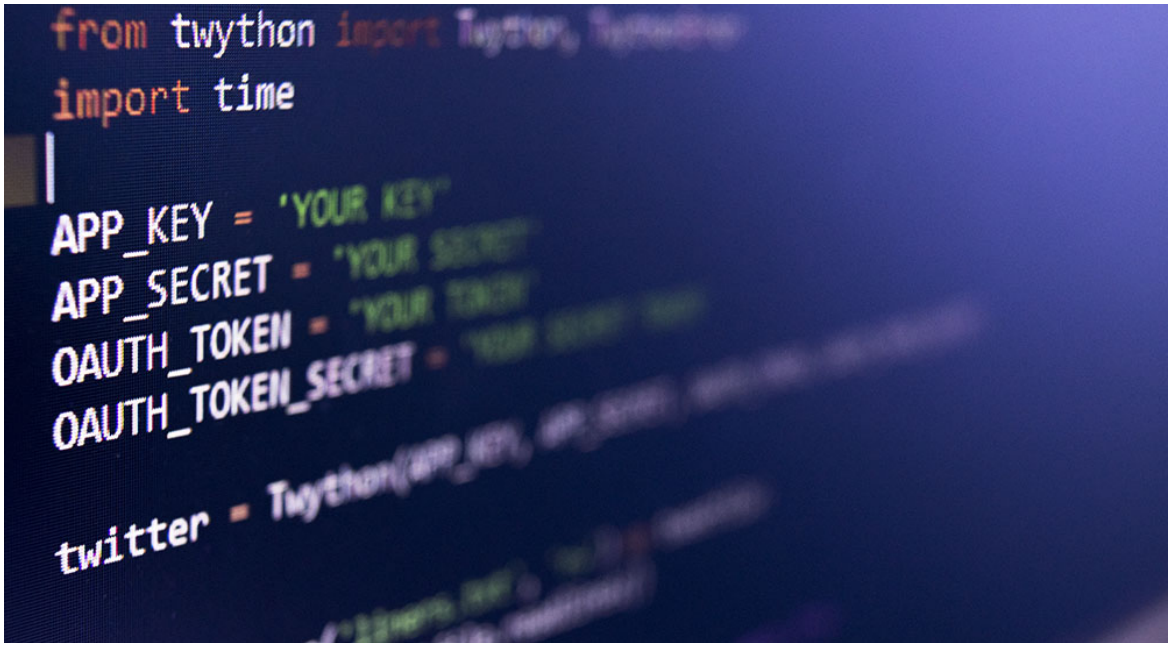
Of course, the abstractions and solutions given by one language sometimes do not offer the solutions needed for everyone. More specialised and convenient

programming language are created every day. Currently, there are thousands of programming languages, each suited more for one purpose or another, and providing different kinds of functionality with its own trade-offs. Currently, the most popular and widely used programming languages in the world are *JavaScript*, *Java*, *C/C++*, and *Python* which we are going to explore in our project!

Overview of Python

Python is a powerful, high-level, object-oriented programming language created by Guido van Rossum. It has simple and easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time. It is a general-purpose language which has a wide range of applications from web development (e.g. *Django*, *Flask*, *Bottle*), scientific and mathematical computing (*Pandas*, *NumPy*, *scikit-learn*) to desktopgraphical user Interfaces (*Pygame*, *Panda3D*). The syntax of the language is clean and length of the code is relatively short. It's fun to work in Python because it allows you to think about the problem rather than focusing on the syntax. At the same time, it is a very powerful language. Using *TensorFlow* for example, you can write state of the art machine learning programs.

Writing Python



In this course, we are going to use a website, specifically for the micro:bit. But if you want to program in python on your own, follow the instructions below.

To write Python on any computer and to interpret it you need to install Python on your device. You can do it by going to www.python.org and following instructions for the specific operating system you are using. After installing Python on your device, you will need a text editor in which you will be writing your code to compile. In real-world application other, more sophisticated text editors such as *Atom* or even IDE's (integrated development environments) like *PyCharm* or *Visual Studio* can be used.

Learning Python

The detailed course material we will be covering together can be found on the project website macau.computer-science.party

There are plenty of resources to expand your understanding and skills in Python after this project. Here are some good ones to start out from:

- docs.python.org official Python tutorials created by the Python Software Foundation
- Learn Python the Hard Way: A very Simple Introduction to the Terrifyingly Beautiful World of Computer Code by Zed A. Shaw (free PDF available online)
- Python programming tutorial in the single video by Derek Banas available on YouTube.

Introduction to Robotics with the Bit:Bot

History of Robotics

Many sources attest to the popularity of automatons in ancient and Medieval times. Ancient Greeks and Romans developed simple automatons for use as tools, toys, and as part of religious ceremonies. Predating modern robots in industry, the Greek God Hephaestus was supposed to have built automatons to work for him in a workshop.

In the Middle Ages, in both Europe and the Middle East, automatons were popular as part of clocks and religious worship. Many other automata were created that showed moving animals and humanoid figures that operated on simple cam systems, but in the 18th century, automata were understood well enough and technology advanced to the point where much more complex pieces could be made. Automata were so popular that they travelled Europe entertaining heads of state such as Frederick the Great and Napoleon Bonaparte.

The Industrial Revolution and the increased focus on mathematics, engineering and science in England in the Victorian age added to the momentum towards actual robotics. Charles Babbage (1791-1871) worked to develop the foundations of computer science in the early-to-mid the nineteenth century, his most successful projects being the difference engine and the analytical engine.

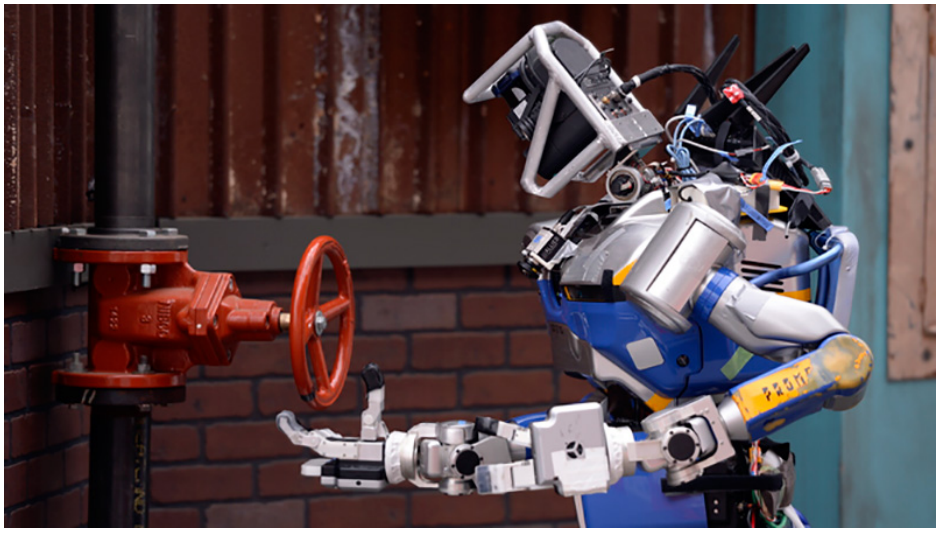
Automata continued to provide entertainment during the 19th century, but coterminous with this period was the development of steam-powered machines and engines that helped to make manufacturing much more efficient and quick. Factories began to employ machines to either increase workloads or precision in the production of many products.

In 1920, Karel Capek published his play R.U.R. (Rossum's Universal Robots), which introduced the word "robot." It was taken from an old Slavic word that meant something akin to "monotonous or forced labour." However, this was thirty years before the first industrial robot went to work. In the 1950s, George Devol designed the Unimate, a robotic arm device that transported die castings in a General Motors plant in New Jersey, which started work in 1961.

Robotics became a burgeoning science and more money was invested. Robots spread to Japan, South Korea and many parts of Europe over the last half century.

Additionally, robots have found a place in other spheres, as toys and entertainment, military weapons, search and rescue assistance, and many other jobs. Essentially, as programming and technology improve, robots find their way into many jobs that in the past have been too dangerous, dull or impossible for humans to achieve. Indeed, robots are being launched into space to complete the next stages of extraterrestrial and extrasolar research.

Why Is Robotics Still Hard?



Making robots is no easy task. If you talk to roboticists, they will tell you that it took years before the last robot they built or programmed was any good at performing a specific task. And although you may see videos of impressive robotic feats, the reality is often more sobering. So why is it difficult to make robots? Here's a breakdown looking at why robotics still requires years of research before seeing them in our everyday life.

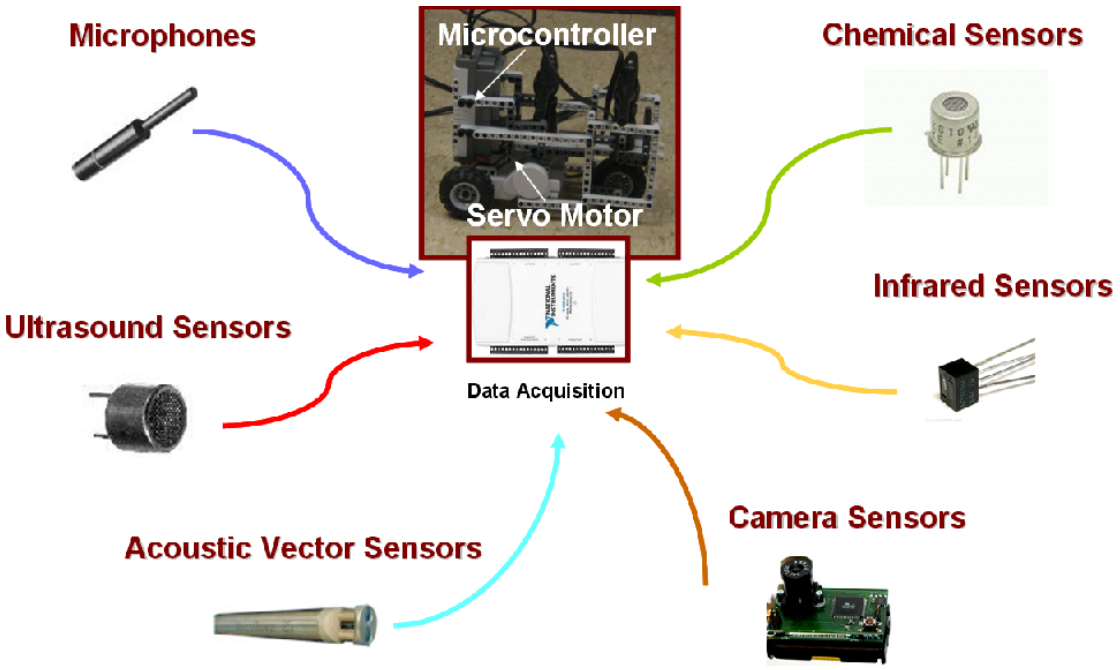
Power

Most robots are required to operate without being plugged into a power socket. This means they need to carry their own energy source, be it a battery pack or gas tank. Small drones can typically operate for less than 1 hour, which is also the battery life of most advanced humanoids such as ATLAS from Google's Boston Dynamics. So by

the time the robot has walked out the door and made a few steps, it's time for a power recharge.

Progress is being made, and a push for batteries that allow our laptops and cell phones to work for days on end is also powering the increase in robot run time. The main challenge is that robot motion is often power hungry. Most drones will use the largest portion of their energy powering their propellers rather than computation, sensing, and communication combined. Larger batteries could give a robot more power, but will also make it heavier, which then requires more energy to move the robot. The reality is that robots are often docked to a charging station.

Sensing

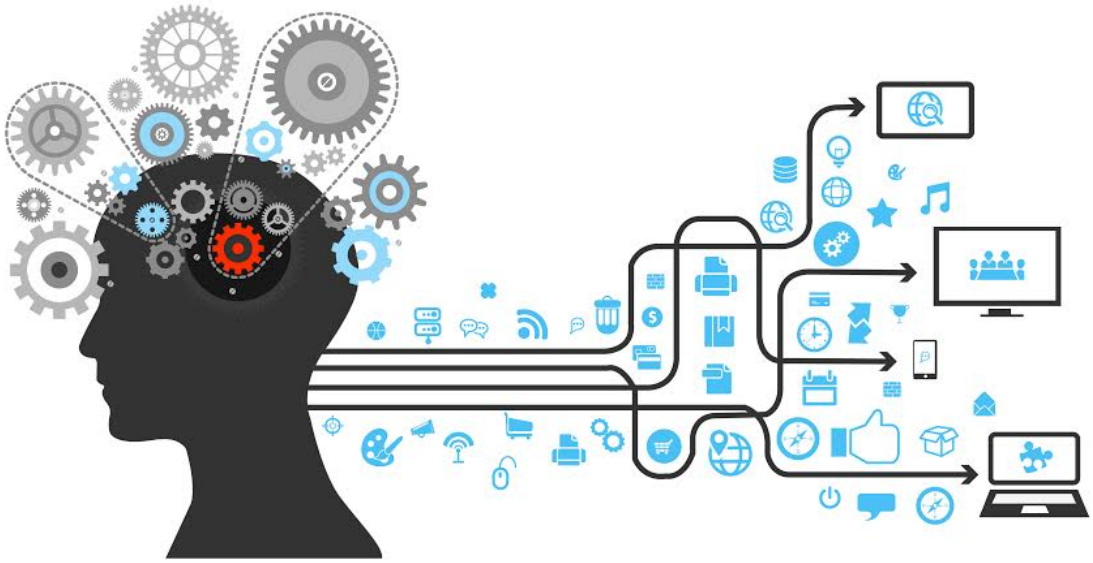


Did you ever wonder why most demos show robots manipulating objects with bright colours? Robots still have a hard time recognising everyday objects. Even though various algorithms have proven to be effective to label images with labels such as "black cat" or "white rose", robots need to know how the object is used and how you can interact with it. A fuchsia shirt, striped jacket, or a pair of trousers will all look quite different for a laundry robot and each of them will require a different sequence of motion. Cameras are helpful but image processing is still a burden. Beyond vision, touch and sound are still rarely used in robotics.

Manipulation

Industrial robots are very successful at manipulating specific pre-defined objects in a repetitive manner. Manipulation outside of these constrained environments is one of the greatest challenges in robotics. There is a reason most successful commercial robots for the home environment, including telepresence robots, vacuum cleaners, and personal robots, are not built to pick up objects. Companies such as Shadow Robot are trying to capture the fine motor control that allows us to interact with everyday objects in a robotic hand – using these manipulators often requires precise planning.

Cognition



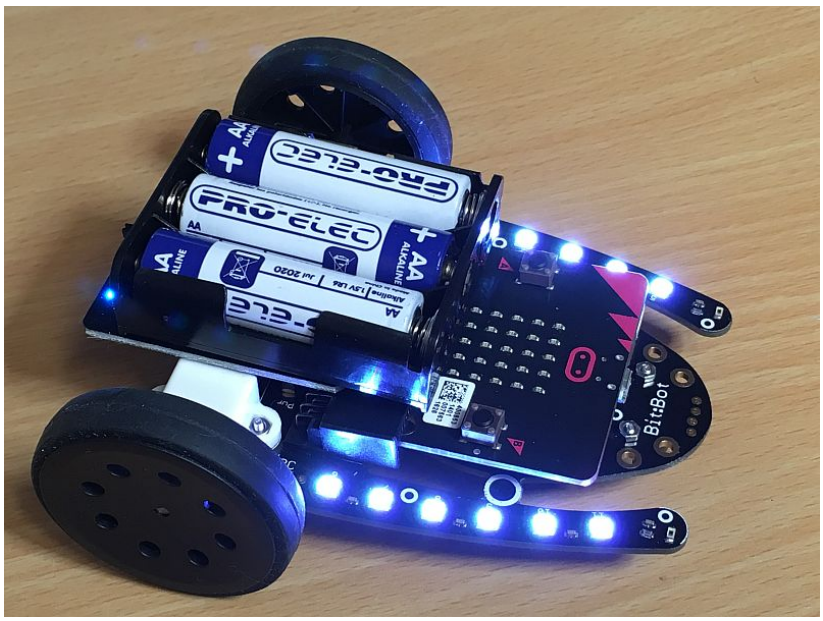
Current robots typically use well-determined algorithms that allow them to complete specific tasks, for example navigating from point A to point B, or moving an object on an assembly line. Designing collaborative robots or robots for the home will increasingly require them to understand new environments and learn on the job. What seems like a simple task to us, could turn into a complex cognitive exercise for a robot.

Whatever the learning embedded in the robot, it's important to realise that we are still far from anything that resembles human intelligence or understanding. The forest trail navigation mostly crunches the data from lots of forest trail images and performs the correct motor commands in response. This is closer to a human learning to balance a poll on the palm of their hand through practice, rather than the development of a real understanding of the laws of physics.

Unstructured environments

The world is a messy place, and for most robots, operating in unstructured environments is difficult. That's why commercial robots have been most successful in factories, on warehouse floors or roads, in the open air, and underwater. On the other side, there are very few robots that operate autonomously in the home environment, other than vacuum cleaner robots.

Overview of Bit:Bot



A great way to engage with both robotics and Python is through the usage of the bit:bot – a robotic car powered by the micro:bit. Bit:bot has the following features:

- 2 micro metal gear motors. Both fully controllable in software, for both speed and direction
- Wheels with rubber tyres for maximum grip
- Really smooth metal ball front caster
- 12 mini neopixels in 2 sets of 6 along the arms either side.
- 2 digital line following sensors.
- 2 analog light sensors
- Buzzer
- Powered from integrated 3xAA battery holder
- Extension port for additional neopixels
- Expansion connections at the front for additional sensors