Introduction tot Advanced sensors

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Advanced sensors	Date:	Friday, 8 December 2023, 4:35 PM
Book:	Introduction tot Advanced sensors		

Table of contents

1. What is a sensor?

2. Application of sensors in manufacturing

3. IoT networking

4. System- and application software

5. Connections to the web

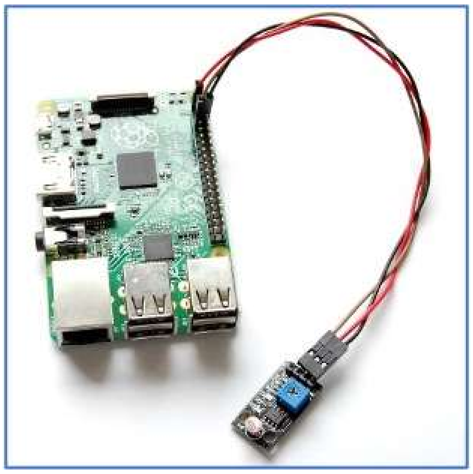
6. Data storage

7. Linux

1. What is a sensor?

A **sensor** is an artificial implementation of what in biology is called a sense. Most sensors are electronic or mechanical, but there also are software and virtual sensors. A sensor helps a machine perceive the environment or collect information that can be used to control processes in industry and computer science.

A sensor measures a physical quantity in any of the following domains: radiation, pressure, temperature, magnetism, level, movement, light intensity and chemistry. Sensors convert the measured quantity into a standardised 0/4-20 mA or 0-10 V control signal for further processing, for example via an analogue-digital converter to a **Programmable Logic Controller** (PLC) or **Distributed Control System** (DCS). In this course, we focus on the **Raspberry Pi** to collect and process data (see unit 2).

Figure 1:  Pi with sensor

Sensors can be made using the same technology as for **microprocessors** and memories in PCs, but also with other technologies. This will be discussed in more detail later.

2. Application of sensors in manufacturing

Sensors are key devices in manufacturing environments. Processes in manufacturing can be **automated** by means of sensors. This will enable making the right choice at the right time. To achieve this, sensors collect **data**. This can be different kinds of data, such as pressure, temperature, magnetism, height, movement and light.

Indicators that can be checked in the process are established in advance. These are called **Key Performance Indicators** (KPIs). KPIs are an operational tool to monitor the performance of a process with the aim of improving it.

There are two main types of KPIs: Input & Output. **Output KPIs** are the most important KPIs, as they relate to the result you are looking for. These are, for example, the number of products that a manufacturing process produces. With **Input KPIs** you look at actions that you take to realise a certain output. Think, for example, of checking whether a certain raw material has reached the correct temperature for processing.

3. IoT networking

Thirty billion things provide trillions of gigabytes of data. How can they work together to enhance our decision-making and improve our lives and our businesses? Enabling these connections are the networks that we use daily. These networks provide the foundation for the Internet and the digitised world.

The methods that we use to communicate are evolving continually. Whereas we were once limited by cables and plugs, breakthroughs in wireless and digital technology have significantly extended the reach of our communications.

Forming the foundation of the digitised world, networks come in all shapes and sizes. They can range from simple networks consisting of two computers to networks connecting millions of devices.

Simple networks in homes enable connectivity to the Internet. They also make it possible to share resources, such as printers, documents, pictures and music, between a few local computers.

In businesses and large organisations, networks can provide products and services to customers through their connection to the Internet. Networks can also be used on an even broader scale to provide consolidation, storage and access to information on network servers. Networks enable email, instant messaging and collaboration among employees. In addition, networks enable connectivity to new places, making machines more valuable in industrial environments.

The Internet is the largest network in existence and effectively provides the "electronic skin" that surrounds the planet. In fact, the term "internet" means "network of networks". The Internet is literally a collection of interconnected private and public networks. Businesses, small office networks and home networks connect to the Internet.

Watch the following 5-minute video on "Networks":

0:00

NETWORK TYPES

There are many different types of modern networks, characterised by their geographic size, the number of devices or networks that they connect, and whether they support mobile devices or not. Networks can also be characterised by their function and purpose.

Personal Area Network (PAN)

Personal area networks are small networks where connected wireless devices are within personal reach. Connecting your smartphone to your car using Bluetooth is an example of a PAN.

Local Area Network (LAN)

LANs are typically networks in a small or local geographic area, such as a home, small business or department within a large corporation. LANs can connect two or more devices, including computers, printers and wireless devices. LANs provide access to larger wide area networks (WANs) and the Internet.

Wide Area Networks (WANs)

The term WAN typically refers to a collection of LANs that provides inter-LAN and Internet connectivity for businesses and governments.

The Internet is a multi-layer global network system that connects hundreds of millions of computers. The Internet is not owned by any one person or organisation. This large system comprises multiple local and global networks serving private, public, business, academic and government purposes. It allows for the exchange of data between more than a hundred Internet-linked countries worldwide. This makes the Internet an enormous carrier of diverse information resources and services. These include text and multi-media data, email, online chat, VoIP, file transfer and file sharing, ecommerce and online gaming.

Wireless Networks

Wireless networks are computer networks that use electromagnetic waves instead of wires to carry signals over the various parts of the network. Wireless networks can be described as PANs, LANs, or WANs, depending on their scope.

Because browsing the Internet is considered a normal daily activity, wireless access points have become commonplace in today's communication infrastructure. Public Internet-connected places include libraries, airports, coffee shops, hotels, and specialised Internet cafés. Thanks to Wi-Fi technology, the Internet can now be accessed by every person with a laptop, tablet or smartphone.

The Cloud

The term "cloud" is used in many ways. The cloud is not as much a type of network as it is a collection of data centres or groups of connected servers that are used to store and analyse data, provide access to on-line applications and provide backup services for personal and corporate use. Cloud services are provided by different organisations.

The Edge

The edge refers to the physical "edge" of a corporate network.

Fog Computing

With the rising number of sensors used by the Internet of Things, there is often a need to store the sensor data securely and closer to where the data can be analysed. The analysed data can then be used quickly and effectively to update or modify processes within the organisation. The fog is located at the edge of a business or corporate network. Servers and computer programs allow the data to be pre-processed for immediate use. The pre-processed data can then be sent to the cloud for more in-depth computing if required.

The **Internet of Things** (IoT) is the connection of millions of smart devices and sensors connected to the Internet. These connected devices and sensors collect and share data for use and evaluation by countless organisations. These organisations include businesses, cities, governments, hospitals and individuals. The IoT has been made possible in part due to the advent of cheap processors and wireless networks. Previously inanimate objects such as doorknobs or light bulbs can now be equipped with an intelligent sensor that can collect and transfer data to a network.

Researchers estimate that over 3 million new devices are connected to the Internet each month. Researchers also estimate that in the next four years, there will be over 30 billion connected devices worldwide.

Perhaps a third of connected devices will be computers, smartphones, tablets and smart TVs. The remaining two-thirds will be other "things": sensors, actuators and newly invented intelligent devices that monitor, control, analyse and optimise our world.

Some examples of intelligent connected sensors are: smart doorbells, garage doors, thermostats, sports wearables, pacemakers, traffic lights, parking spots, and many others. The variety of different objects that could become intelligent sensors is limited only by our imagination.

IoT devices connected to the network

A sensor needs to be connected to a network so that the gathered data can be stored and shared. This requires either a wired Ethernet connection or a wireless connection to a controller. Controllers are responsible for collecting data from sensors and providing network or Internet connectivity. Controllers may have the ability to make immediate decisions, or they may send data to a more powerful computer for analysis. This more powerful computer might be in the same LAN as the controller or might only be accessible through an Internet connection.

Sensors often work together with a device called an actuator. Actuators take electrical input and transform that into physical action. If, for example, a sensor detects excess heat in a room, the sensor sends the temperature reading to a microcontroller. The microcontroller can then send the data to an actuator, which turns on the air conditioner.

Many new devices such as fitness wearables, implanted pacemakers, air meters in a mine shaft and water meters in a farm field all require wireless connectivity. Because many sensors are "out in the field" and are powered by batteries or solar panels, power consumption is an important consideration. Low-powered connection options must be used to optimise and extend the availability of the sensor.

A sensor needs to be **connected** to a network so that the gathered data can be stored and shared. This requires either a wired Ethernet connection or a wireless connection to a controller. Controllers are responsible for collecting data from sensors and providing network or Internet connectivity. Controllers may have the ability to make immediate decisions, or they may send data to a more powerful computer for analysis. This more powerful computer might be in the same LAN as the controller or might only be accessible through an Internet connection.

Sensors often work together with a device called an actuator. Actuators take electrical input and transform that into physical action. If, for example, a sensor detects excess heat in a room, the sensor sends the temperature reading to a microcontroller. The microcontroller can then send the data to an actuator, which turns on the air conditioner.

Many new devices such as fitness wearables, implanted pacemakers, air meters in a mine shaft and water meters in a farm field all require wireless connectivity. Because many sensors are "out in the field" and are powered by batteries or solar panels, power consumption is an important consideration. Low-powered connection options must be used to optimise and extend the availability of the sensor.

Chapter 3 "Using a device to work with sensors" describes different devices that can work with sensors and communicate measurements with databases and other devices.

Automation through sensors

Automation is any process that is self-driven and reduces, then eventually eliminates, the need for human intervention.

Automation was once confined to the manufacturing industry. Highly repetitive tasks such as automobile assembly were turned over to machines and the modern assembly line was born. Machines are excellent at repeating the same task without getting tired and without the errors that humans are prone to make in such jobs. This results in greater output, also because machines can work 24 hours a day without breaks. Machines also provide a more uniform product.

The IoT opens up an entirely new world in which tasks previously requiring human intervention can be automated. As we have seen, the IoT allows the collection of vast amounts of data that can be quickly analysed to provide information that can help guide an event or process.

As we continue to embrace the benefits of the IoT, automation becomes increasingly important. Access to huge amounts of quickly processed sensor data made people think about how to apply the concepts of machine learning and automation to everyday tasks. Many routine tasks are being automated to improve their accuracy and efficiency.

Automation is often related to the field of robotics. Robots are used in dangerous conditions such as mining, firefighting and cleaning up industrial accidents, reducing the risk to humans. They are also used in such tasks as automated assembly lines.

We now see automation everywhere, from self-service checkouts at shops and automatic environmental controls in buildings to autonomous cars and planes. How many automated systems do you encounter in a single day?

Chapter 2 "Sensors" describes different sensors for measuring different values.

4. System- and application software

There are two common types of computer software: **system software** and **application software**.

Application software programs are created to accomplish a certain task or collection of tasks. For example, Cisco Packet Tracer is a network simulation program that allows users to model complex networks and ask "what if" questions about network behaviour.

System software works between the computer hardware and the application program. It is the system software that controls the computer hardware and allows the application programs to function. Common examples of system software include Linux, Apple OSX and Microsoft Windows.

Both system software and application software are created using a **programming language**. A programming language is a formal language designed to create programs that communicate instructions to computer hardware. These programs implement algorithms that are self-contained, step-by-step sets of operations to be performed.

Some computer languages compile their programs into a set of machine-language instructions. C++ is an example of a compiled computer language. Others interpret these instructions directly without first compiling them into machine language. Python is an example of such an interpreted programming language. In this module we will focus on Python.

When the programming language is determined and the process is diagrammed in a flow chart, program creation can begin. Most computer languages use similar program structures.

Chapter 4 "Programming IoT with Python on a Raspberry Pi" shows the steps for getting started with programming IoT solutions.

5. Connections to the web

Before you can start writing Python programs to connect to the Internet, you need to understand a bit about how the Internet works. It is basically a giant computer network, but one that follows certain rules and uses certain protocols, and we need to utilise those protocols in order to do anything on the Web.

Web Communication Protocols

Most common web traffic is encapsulated in the HyperText Transfer Protocol Secure (**HTTPS**) format. A protocol is simply an agreement between two communicating parties (in this case, computers) as to how that communication is to take place. It includes information such as how data is addressed, how to determine whether errors have occurred during transmission (and how to handle those errors), how the information is to travel between the source and destination and how that information is formatted. The "https" in front of most URLs (Uniform Resource Locators) defines the protocol used to request the page. Other common protocols used are **TCP/IP** (Transmission Control Protocol/Internet Protocol), **UDP** (User Datagram Protocol), **SMTP** (Simple Mail Transfer Protocol), and **FTP** (File Transfer Protocol). Which protocol is used depends on factors such as the traffic type, the speed of the requests, whether the data streams need to be served in order and how forgiving of errors those streams can be.

When you request a web page with your browser, a lot is happening behind the scenes. Let's say you type https://www.dtamproject.eu into your location bar. Your computer, knowing that it is using the HTTPS protocol, first sends www.dtamproject.eu to its local DNS (Domain Name System) server to determine to what Internet address it belongs. The DNS server responds with an IP address—let's say, 168.119.138.10. That is the address of the server that holds the web pages for that domain. The Domain Name System maps IP addresses to names, because it is much easier for you and me to remember "www.dtamproject.eu" than it is to remember "168.119.138.10".

Now that your computer knows the server's IP address, it initiates a TCP connection with that server using a three-way "handshake". The server responds and your computer asks for the page index.html. The server responds and then closes the TCP connection. Your browser then reads the coding on the page and displays it. If there are other parts of the page it needs, such as PHP code or images, it requests those parts or images from the server and displays them as well.

6. Data storage

Data storage

Depending on the IoT application, IoT devices may be required to store data for a period of time before sending it out for processing. This scenario is common with devices that do not maintain constant connections to their gateways or controllers. A good example of this situation is vehicle trackers installed in transport trucks. The system may be built to tolerate periods when the trackers are out of range and cannot communicate to transmit the location of the truck. If this happens, the trackers will store the data in the device itself. As another example, consider a connected car. If it is damaged beyond repair, the insurance company may choose to auction it, along with the owner's data still stored in the car's storage devices. In both examples the data should be kept encrypted to avoid tampering or data theft.

Desktop and laptop storage devices (hard drives and SSDs) have included support for built-in encryption for a while now. Known as self-encrypting drives, these storage devices stand out because the encryption capability is built into the drive controller, allowing the drive itself to encrypt and decrypt, independent of the operating system.

While it is not included in every IoT device, manufacturers are beginning to release new devices with self-encrypting flash memory. Be it in thumb drive or SD card format, the concept of self-encrypting storage devices is very important for IoT.

Cloud services often rely on servers to provide service. Data stored in these servers must also be encrypted to avoid data tampering or theft. Regular backups are mandatory to minimise losses in case of an emergency.

Data transmission

IoT devices are often small, inexpensive devices, with little to no security. Although such devices are computers, they rely on constrained memory and computing resources and may not support complex and evolving security algorithms. Modern encryption algorithms may require more processing power than is available in the IoT device. In addition to physical security, the IoT device must be able to protect its own firmware and the data it transmits. If data is not properly secured through encryption, it can be intercepted, captured or manipulated while in transit. Any of these actions may compromise system confidence and make the data unreliable.

To mitigate this problem, it is important to ensure that IoT devices are running the latest version of their firmware and protocols. Also ensure that communication uses protocols that provide secure encryption by default. The encryption algorithm must be strong, with older algorithms tending to present exploitable weaknesses. Regardless of the encryption method chosen, make sure all endpoints agree on the most secure parameters available. A common attack is to trick devices to agree on sub-optimal security parameters under which the connection can be exploited. It is also important to use and verify digital certificates. This is often a challenge with small IoT devices because of their limited memory and CPU capacity.

7. Linux

Servers and cloud endpoints should also be secured and use strong encryption algorithms before communicating with IoT devices. If the IoT device relies on an intermediary device such as a gateway or controller, this intermediary device must also use strong encryption. Naturally, intermediary devices should also be kept up to date with the latest software to keep the device from becoming the weak link that breaks the chain.

Linux

Linux is an operating system dating back to 1991. It was created, and is currently maintained, by a community of programmers. Linux is open source, fast, reliable and small. It requires very little hardware resources to run, and is highly customisable. Linux is part of several platforms and can be found in anything from "wristwatches to supercomputers". Linux is also a very popular choice in IoT devices.

Another important aspect of Linux is that it is designed to be on the network. Network operations and connections are simple in Linux, making it a good choice for network professionals and administrators. Anyone can get the kernel's source code, inspect it, modify it and re-compile it at will. Companies and users are free to modify, repackage and run the software. They are able to redistribute the program with or without charges.

Linux distribution is the term used to describe different Linux packages created by different companies. Linux distributions (or distros) include the Linux kernel, plus a number of customised tools and software packages. While some of these will provide and charge for Linux support (geared towards Linux-based businesses), the majority of them also offer their distribution for free without support. Debian, Red Hat, Ubuntu, Slackware and Mint are just a few examples of Linux distributions.

Raspbian is a Linux operating system distribution created specifically for the Raspberry Pi. Raspbian is a Debian Linux variation and, as such, maintains its Linux structure.

Linux shell

The Linux operating system can be divided into kernel and shell. The kernel can be thought of as the OS itself, while the shell is just a program that runs on the OS and offers user-OS interaction functionality.

To interact with the machine's hardware, the user interacts with the shell, which interacts with the kernel, which in turn interacts with the hardware.

The shell is a command interpreter and therefore the terms shell, terminal, console and CLI are often used interchangeably. This course uses the term terminal to refer to shell. When a user logs in on the system, the login program checks the username and password; if the credentials are correct, the login program calls the shell. From this point on, an authorised user can begin interacting with the OS through text-based commands.

Historically, when no graphical interface was available, the shell was text-based with no support for a mouse or advanced visual features. Also called a Command Line Interface (CLI), the text-based shell was the only way for the user to interact with the OS. Today, the Linux shell is still extremely important, as it provides low level access to the system. While many modern Linux distributions include support for graphical user interfaces (GUIs), the shell is still considered the most efficient method of interacting with the system. This is especially true when the interaction is related to system maintenance or troubleshooting.

As an example of the user-shell-kernel interaction, suppose a user wants to delete a file named myFile. The user types rm myFile. rm is the command to remove files and myFile is the name of the file to be deleted. The shell searches the file system for the rm program and then asks the kernel, via system calls, to execute the rm program with the parameter myFile. While the process is running, the shell will be unavailable to the user, presenting no prompt to signify that status. When the rm myFile process has finished running, the shell displays the prompt to the user, indicating that it is ready and waiting for further commands

Different kinds of sensors and their operation

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Advanced sensors	Date:	Friday, 8 December 2023, 4:36 PM
Book:	Different kinds of sensors and their operation		

Table of contents

1. Overview

2. Temperature/Humidity/Air pressure/Gas sensors

3. Motion sensors

4. Navigation modules

5. Raspberry Pi Sensors – Wireless / Infrared (IR) / Bluetooth

6. Motors

7. Analogous Raspberry Pi Sensors

8. Power/Current Supply

9. Displays

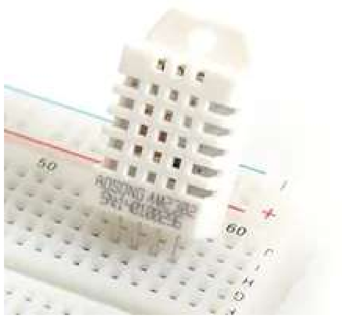
10. Other Modules, Components and Raspberry Pi Sensors

1. Overview

The described Raspberry Pi sensors, modules and components are divided into the following categories:

- Temperature / Humidity / Air Pressure / Gas
- Motion Sensors
- Navigation Modules
- Wireless / Infrared (IR) / Bluetooth
- Motors
- Analogue Sensors
- Current Supply
- Displays
- Other Modules, Components and Sensors

2. Temperature/Humidity/Air pressure/Gas sensors



The DHT11 and DHT22 sensors measure humidity as well as temperature. Only one GPIO is used. The difference between the two is mainly the measuring range and accuracy. The white DHT22 can measure all humidity ranges from 0-100% with an accuracy of 2%. By comparison, the DHT11 (blue) is only able to measure areas of 20-90% humidity and, above all, the accuracy is significantly worse, at 5%.

The DS18B20 and DS18S20 represent very simple sensors. These Raspberry Pi sensors are addressed via the so-called 1-wire bus. An advantage is that many different 1-wire components can be connected in series and read out by a single GPIO. However, these modules cannot measure additional information such as humidity or air pressure. The DS18B20 is particularly suitable for outdoor use, as there are also water-resistant versions available. With a measuring range from -55°C to +125°C it is well suited even for non-everyday applications.

Barometer:

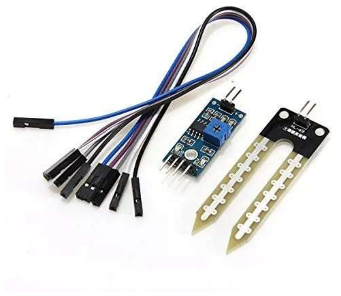
Measuring the air pressure can be meaningful in weather stations and similar projects. This is best done using the BMP180, which is controlled via I2C on the Raspberry Pi. In addition to the air pressure, it reads out the temperature and the altitude. However, this last value is not very accurate. If you need the height, you should read the values with a GPS receiver.

Gas sensor:

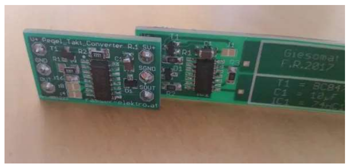
The MQ gas sensors can detect different gases at room temperature. Other gases are supported depending on the model. The MQ-2 recognises methane, butane, LPG and smoke, the MQ3 detects alcohol, ethanol, smoke and more. You can find a list of all MQ sensors and their supported gases here.

These sensors can be very hot and should not be touched directly! Since these modules also work analogously with 5V, you also need a MCP3008 as well as a 3.3V-5V TTL to read the signal.

You should take care that these sensors can be very hot and they should not be touched directly. Since these modules also work analogically with 5V, you need also a MCP3008 as well as a 3.3V-5V TTL to read the sign



Humidity sensor:
This analogue humidity sensor is ideal for automatic irrigation systems. It is placed in the ground and measures the humidity by current flowing between the strands. The more humid the earth in between, the higher the (analogue) signal. To read the value with the Raspberry Pi, the MCP3008 is needed (Arduinos recognise analogue signals directly).



A problem with analogue moisture sensors is that they erode over time and are not always very precise. Capacitive sensors prevent these problems. The relative humidity is calculated by means of the frequency. However, a frequency divider is also suitable for use with the Raspberry Pi.

3. Motion sensors

The PIR motion sensor has some advantages over other similar products: apart from the low price, it only sends a signal if something moves. This allows you to wait for signal flanks using the GPIOs. In addition, resistance can be varied, so that a signal is only sent when the movement is close or when changes that are already far away are perceived.

In addition to outdoor projects, the PIR can also be used in buildings – whether to activate the lighting or, as I use it, to turn on my touchscreen for home automation as soon as someone approaches it.

The HC-SR04 sensor is not a distance / motion detector, but an ultrasonic sensor. Using a small trick, it is nevertheless possible to measure distances. You can derive the distance by measuring the time elapsed between transmitting and receiving an ultrasound signal, as the sound velocity in the air is known. I explain the details in the tutorial. The wide opening angle is an aspect that must, however, be considered: since the ultrasound propagates not only in a straight line, but at an angle of about 15°, the signal is first reflected from the nearest point in this area – which can be also an external point.

As a rough estimate, or for moving robots, it is nevertheless useful, also because of the low price.

You can check for binary states by means of magnetic sensors / reed relays. The magnetic relay is opened as soon as a magnet is in the vicinity. Otherwise, the access is closed. If voltage is then passed through, you can check the condition.

These magnetic sensors are suitable for inspecting windows and doors by mounting them on the frame and checking whether the door / window is open or closed.

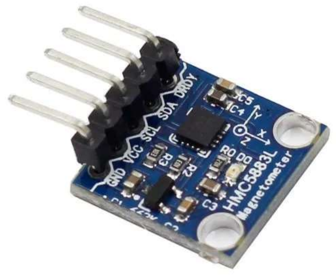
With the GP2Y0A02YK infrared distance meter, much more accurate measurements can be performed, as with e.g. the HC-SR04. The module is limited to a range of 20-150cm. Alternatively, the similar sensor GP2Y0A710K0F can be used, which has a range of 100 to 500cm

4. Navigation modules

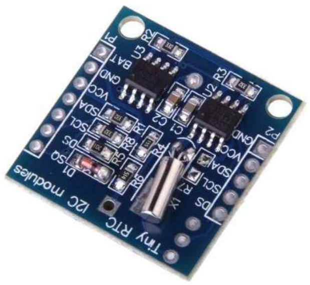
The most common and best-known GPS receiver is the NEO-6M module. All GPS position data can be determined with the help of orbiting satellites. In addition, it is compatible with the minicom and gpsd Raspberry Pi packages, which makes reading the coordinates very easy.

As an alternative to GPS modules connected via the GPIOs, USB GPS receivers can also be used. They have the advantage that (almost) all are compatible with Windows, Linux and Mac and no additional connection is necessary. On the other hand, these modules are usually more expensive, but they are not inferior in terms of accuracy. As such, the type of receiver is an individual preference.

A gyroscope (circular instrument) is used to detect the rotation along the three axes. The MPU 6050 sensor also contains an acceleration sensor. This module can be used in robot arms to determine the angle of rotation.

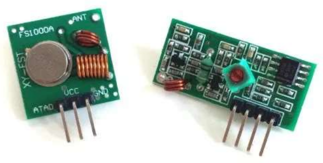


As with analogue compasses, the directional display can also be read digitally. The HMC5883L sensor, which is read out via I2C, which returns an angle in radians, is suitable for this purpose. As with a normal compass, the value can be confounded by metal objects nearby.



If the Raspberry Pi is connected to the Internet, it can request the exact time. This could be a problem in applications where no (permanent) Internet connection is given but the date and the exact time are nevertheless important (car PC, weather station, etc.). A so-called Realtime Clock (RTC) module can help: Once initialised, it saves the current time using a small battery, i.e. even without power supply. Such modules are installed on computer mainboards, which is why you do not have to re-adjust the time of the computer every time you restart. The Raspberry Pi / Arduino does not carry an RTC module as a default feature, but it can be retrofitted.

5. Raspberry Pi Sensors – Wireless / Infrared (IR) / Bluetooth



One of the simplest methods to transmit signals via radio is by means of a 433 MHz transmitter and receiver. These sets are very cheap and they are used in many projects. This way, you can let several Raspberry Pis communicate with each other, for example. Many other devices also work with 433 MHz radio signals, such as garage doors or radio-controlled sockets, and these codes can be recorded and sent for specific tasks

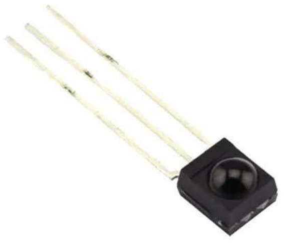
A more advanced method for wireless communication is the use of the 2.4 GHz frequency. The advantages compared to the 433 MHz transmission rate are mainly that a larger amount of data can be transferred at once. Thus, whole sentences and commands can be sent with a signal / data package. A second Raspberry Pi or an Arduino can also be equipped with a 2.4 GHz receiver / transmitter, receive commands from a "base station" and send back data.

In the field of home automation, wireless sockets are almost a standard. The vast majority of these devices work with 433 MHz radio signals. By reading the codes of the remote control with a receiver on the Raspberry Pi, these radio sockets can be switched individually.
There are different models, usually pure switchable radio sockets, but dimmable lamp sockets are also offered.
You should pay attention to one criterion: There are models with a "generic" code, which means the code is randomly generated and changes. These frequencies are hard to read out. Sockets in which the code is freely selectable, on the other hand, are very easy to control.

The Si470x module offers the option to upgrade the Pi to a radio receiver, which can be very interesting in Car PCs or Raspberry Pi Jukeboxes. As with conventional radios, the frequency and certain options can be adjusted via software. If that were not enough, you can also use your Pi as a radio station.

The Raspberry Pi has not always had an integrated Bluetooth module. Before the model 3 was published, neither Bluetooth nor Wi-Fi modules were onboard. The inexpensive Zero model also comes without a Bluetooth adapter. Since almost every mobile phone supports this communication method as standard, it is easy to exchange pictures and other files between a smartphone and Raspberry Pi. Other projects such as controlling the Pi via Bluetooth commands are also possible.

The Raspberry Pi is used in many outdoors projects, e.g. as a weather station or for monitoring. However, many functions may be restricted if no (or only a weak) Wi-Fi signal is available. If you still want to have access to the Pi and also receive the data of such an outside project, an Internet connection is necessary. Mobile surfsticks that are often available as gifts for data rate contracts can be useful here. With such a stick and a SIM card with data volume, the Pi can be permanently online. In addition, it is also possible to use the stick to send and receive text messages, for example to remotely control the Raspberry Pi using a mobile phone.



Most remote controls use infrared LEDs to transmit signals. These codes can be read and stored easily with an infrared receiver. With the LIRC package, these codes can be sent with an IR transmitter diode. For example, a TV can be controlled with the Raspberry Pi.

In addition, there are also IR LEDs, which can be used as a light barrier.

Although standard laser modules do not have great functionality (can be switched on and off), they are used in various interesting projects. There are, for example, projects with distance measuring devices that use a camera and a laser module. The laser is switched on and off very quickly and pictures are recorded. The distance can then be calculated by means of the beam set.

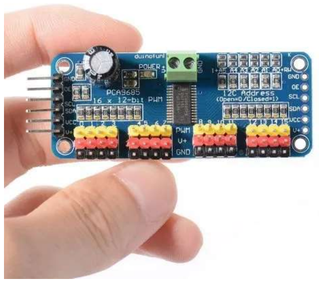
Due to the exchangeable mirrors at the head of the laser modules, different patterns such as grids are possible.

6. Motors

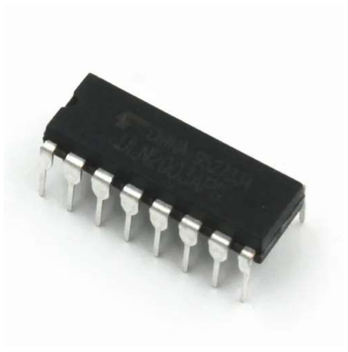
Servomotor: unlike ordinary motors, servomotors can be individually controlled. Only the indication of the angle of rotation for moving the motor is necessary. PWM (pulse width modulation) signals are sent to the motor. The Raspberry Pi can use this method of transmission. Using the Python GPIO library or WiringPi is particularly easy.

Stepper motors: stepper motors are motors that divide a full rotation into a number of equal steps. Two built-in electromagnets move the axis through different poles. What the polarity looks like is specified in the motor's data sheet.

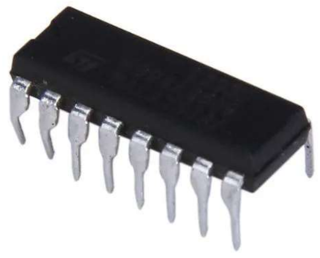
One of the most popular stepper motors (because it has a lot of steps and is nevertheless cheap) is the 28BYJ-48 model. This motor has 512 steps, each consisting of 8 sequences. This means that a full rotation consists of 4,096 steps (or one step is made per 0.087°).



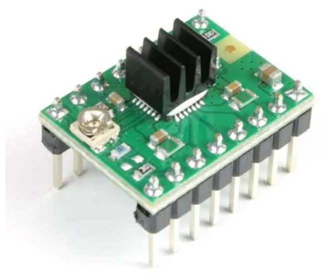
Servo board: using PWM, servos can be controlled directly from the Raspberry Pi. However, as soon as you want to control several servomotors, you either need more GPIOs or more power. The PCA9685 servo driver board is ideally suited for this purpose because you can control up to 16 motors per board via I2C. And if this is not enough, it is even possible to connect several boards one after the other. It is also possible to connect an external power supply. This is the best board if you want to use a robot arm, for example.



ULN 2003: 28BYJ-48 stepper motors are often supplied with a driver board. The board usually has a ULN2003 IC, which holds the voltage for the 5V motor, but can be controlled with 3.3V. This is important because the GPIOs are protected and no transistor or relay is needed.



L293D: an alternative driver IC is the L293D. The advantage of this module, compared to the ULN2003, is that it can also be used with voltages higher than 5V. Because many alternative stepper motors (e.g., fewer steps for faster rotation or higher pulling force) require more than 5V, they must be powered by an external current source. The L293 IC is ideal for controlling these motors. It is, incidentally, even possible to control two motors simultaneously (individually).



A4988: this IC is another way to control step motors. It is especially designed for motors in 3D printers and can withstand voltages of 8V to 35V with a current of 1 amp. Since it can get hot very quickly, a cooling sink is included on the chip of the breakout board.

7. Analogous Raspberry Pi Sensors

MCP3008 Analogue-to-digital converter: Unlike the Arduino, the Raspberry Pi does not have its own analogue IO pins. This means that you cannot simply read out analogue modules. The MCP3008 module helps you: It enables the use of analogue modules with the Raspberry Pi and as such this digital converter is required for all analogue modules on the Raspberry Pi.

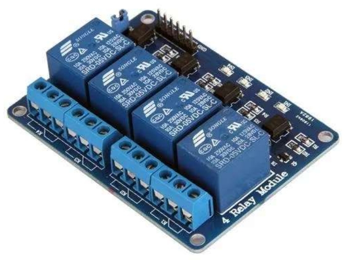
Joystick: One of these analogue modules is a 2-axis joystick. Two potentiometers (see below) for X and Y axes are installed, which allow more or less voltage to pass through the movement. Conversion of the analogue value into a digital value results in numbers between 0 (no voltage) and 1,023 (full voltage). In the centre, a digital value of approx. 512 is returned on both axes

Potentiometer/Rotary switch: Potentiometers are basically rotatable resistors. The resistor value can be changed easily by rotating the control knob. Each module has a maximum resistance (minimum is zero). In addition to joysticks, potentiometers are also used, e.g. in brightness or volume controllers.

Raindrop sensor: a rainwater sensor can be used to determine whether it is raining or how much precipitation is falling. It works analogously and can be read with the MCP3008. Depending on the amount of water, the capacitance is increased and a stronger analogue signal is read out.

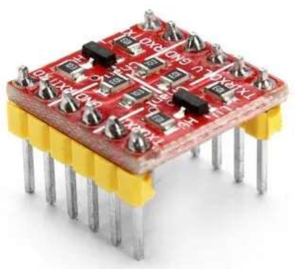
Heartbeat/Pulse sensor: With a pulse sensor, the heart rate can be read out on the Raspberry Pi. The analogously detected value changes depending on the pulse. This is again converted with an ADC and the pulse is determined on the basis of the last measured values.

8. Power/Current Supply



Relay: The GPIOs of the Raspberry Pi work with 3.3V, although it also has a 5V pin. However, many devices require a higher voltage. In order not to combine the circuits, you can use relays, which are basically switches. This has the advantage that you can also switch circuits with higher voltages with the Raspberry Pi, without risking anything

Buck Converter/Step Down Module: With the LM2596 (and similar) modules, higher voltages can be regulated downwards. For example, you can regulate the current of (rechargeable) batteries to the required 5V USB input voltage. However, alternating current (AC) is not allowed, but only direct current (DC) as supplied by batteries..



3.3V – 5V TTL I2C Logic Level Converter:
Some modules and sensors for the Arduino produce 5V signals, but this would destroy the GPIOs, since those work with 3.3V. Here, a level converter can be used to further control the signals.

It is important to ensure that bi-directional level converters are purchased so that you can both send and receive signals.

9. Displays

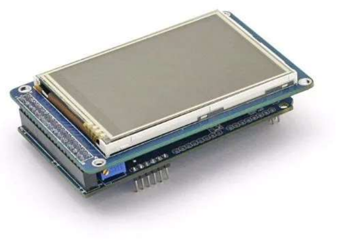
Official 7" Touchscreen:

In September 2015, the Raspberry Pi Foundation finally introduced the official touchscreen display. It measures 7" and has a resolution of 800×480 pixels. The 10-point capacitive touchscreen is connected through the DSI port and does not occupy any USB ports or GPIOs. The initial startup is very easy and you do not need any additional software (only a current version of Raspbian or NOOBS).

Other 7" Touchscreen:

Before the Raspberry Pi Foundation introduced its touchscreen module, many other companies developed touch displays for the Pi. The advantages are mostly the better resolution and sometimes even a bigger size (10" or more). Most of them do not use the DSI port, which means that the HDMI and USB port (for touch) and / or several GPIO pins will be used. Moreover a separate driver is usually needed

3.2" Touchscreens:



Not everyone needs displays of 7" or larger; sometimes a smaller touchscreen is also enough, but the choice is relatively large. Sizes between 2.4 and 4.3 inches are very common, but these modules have almost exclusively resistive touch. You can connect them, depending on the model, via the GPIOs or (if available) directly via HDMI.

In addition to touchscreens, there are also pure character displays. The most common are 16×02 and 20×04 displays, which specifies the number of characters per line and the number of rows. Almost all of these displays have an HD44780 controller, which can be easily accessed with the Raspberry Pi.

7-Segment Display:

7-segment displays are often used to display numbers and, as the name implies, have seven luminous segments, which can be addressed individually. In order not to occupy too many GPIOs, generally a controller like the MAX7219 is used.

In addition to the usual 7-segment displays, there are also models that contain 15 controllable segments and can also display letters (even if it does not look great).

10. Other Modules, Components and Raspberry Pi Sensors

Led Matrix:
The square 8×8 LED matrices are available in red and green. It is possible to control each individual LED with the help of the MAX7219 IC. In addition, many of these modules can be plugged together, resulting in a large dot display. The signal is sent via SPI. I have written a library, which let's you easily control these matrices.

Optical Fingerprint Sensor:
Fingerprint Sensor can be used to implement safety-relevant applications. For example, the fingerprints of different persons are stored and authorization rights are given to them. Electronic saves or door locks can be built. A password can also be requested in conjunction with a numpad. The interrogation of the sensor is surprisingly accurate and takes place by means of features. After reading or storing the imprint, it is even possible to export the imprint as an image.

Arduino Uno:
The Raspberry Pi can also be used as a micro-controller, but it has a lot more functions because it runs an operating system. A true micro-controller is the Arduino. It can also read analog sensors. The Arduino can also be operated very easily on and with the Raspberry Pi, via USB or 433 MHz or 2.4 GHz radio. Since Arduinos are cheaper than normal Raspberry Pis, they can either serve as extensions for the GPIOs or as an outdoor station for certain sensors whose data is transmitted wirelessly. As there are more projects for the Arduino than for the Raspberry Pi, you can also implement and run those projects on the Raspberry Pi (via the Arduino detour).

ESP8266 NodeMCU:
The ESP8266 NodeMCU is a microcontroller that has a built-in Wifi module. Because of this and by its very low price, it is clearly more attractive than an Arduino. The programming takes place via the serial port can be done either via the Arduino IDE or other programs (LUA). If one has distributed a weather station or other IoT devices that are on the same WLAN network, you can send their data to a Raspberry Pi "mother station" via wifi. The ESP8266 is available in different versions, although the most favorable variant (ESP-01) has only two GPIO pins. Other models like the ESP-12 offer a lot more pins for a small extra charge.

Keypad/Numpad:
A numeric input field is required for vault or code lock projects. For this there are own modules, which look like a numpad on the PC keyboard. These modules are available in different sizes (3×4, 4×4, etc.) and can be read directly at the Raspberry Pi. By entering certain numerical codes / combinations, you can execute secret actions.

Magnet Valve:

A solenoid valve is suitable for interrupting the flow of liquids or gases. A kind of "opener" can be built between two pipes or hoses. Ideally, magnetic valves are used, which are operated at 12 volts. All you need is an external power supply and a relay on the Raspberry Pi, which switches the solenoid.

You can use those valves e.g. in the outdoor area (keyword: automatic irrigation) or in smaller projects such as intelligent coffee makers, etc.

Water Flow Meter:

With the aid of water flow meters (Hall effect sensors), the amount of water flowing through the tube per minute / second can be determined at the Raspberry Pi. There are different sensors with a higher accuracy or a higher flow rate and maximum water pressure. These measuring aids are particularly interesting in the outdoor and garden areas. For example, they can measure rainfall during a thunderstorm (drainage channel) or check the irrigation of plants.

Port Expander:

The MCP 23017 device is an IO port expander. Because the Pi has only a limited number of GPIOs, these can easily run out for larger projects or multiple connected modules. A port expander is controlled by I2C and extends the number of IO pins. You have an additional 16 pins per port expander, which you can declare as input or output. You can also connect and control multiple port expanders at the same time.

Weighing Sensor + Load Cell:

Using the HX711 sensor, the Raspberry Pi can also weigh items. This requires a LoadCell (US* | UK*), which must be connected and calibrated once. The accuracy and maximum weight to be measured will vary depending on the model.

Ethernet Module:

If you have a Raspberry Pi Zero, you probably know the problem: Because there is only one (micro) USB port, an Internet connection is only possible by means of Wi-Fi, because – unlike the Raspberry Pi 3 – it does not have an integrated Wi-Fi adapter. If you would like to use another USB device, you need a USB hub. This is where the ENC28J60 module comes in: it is connected to the GPIOs and allows a wired Ethernet connection. This means you do not need an external Wi-Fi stick or USB hub.

Raspberry Pi Camera Module:

Cameras are used in many Raspberry Pi projects. In this case, common-or-garden USB webcams can be used, but their quality is often not very good and they occupy a USB port. A better alternative is the official camera module of the Raspberry Pi Foundation, which can be directly connected to the CSI port. The module is available in two versions: With (green) and without (black) infrared filter. The lack of an infrared filter allows a higher light sensitivity, which results in better images at dusk / night.

Robot Arm:

Multiple servomotors allow you to control a multi-axis robot arm with the Raspberry Pi. There are different versions, of which the most familiar is the one with 6 engines. Each individual servo can be individually controlled, resulting in a high degree of accuracy. Besides the servomotors, a driver board like the PCA9685 is very useful.

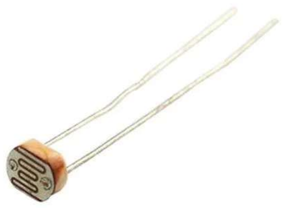


Photo Resistors:

In addition to conventional resistors and potentiometers, there are also photoresistors. These have a light-sensitive surface and a different resistance value, depending on the light intensity. They can be used, for example, to detect day / night or to build light barriers.

WS2801 LED Strips:

WS2801 LED strips contain many controllable RGB LEDs, which can be addressed individually. Depending on the model, there are variants with 30/60/144 LEDs per metre. These LED strips are ideal for the implementation of Ambilight projects. In contrast to the cheaper WS2812B models (which have only one data line), the WS2801B RGB LED strips can be addressed directly from the Raspberry Pi, which means that no additional Arduino is required for intermediate storage.

WS 2812 RGB LED Strip

The WS2812 LED strips offer the advantage of a better price / performance ratio. However, either an Arduino must be used as an intermediate segment, or the onboard sound is deactivated, whereby this strip is also directly accessible. However, since the Raspberry Pi does not send real-time signals and the frequency is not so high, this strip is not suitable for all projects. The WS2812 nevertheless offers some advantages

Exercises

Site:	[DTAM Online Training Platform]()	Printed by:	Ioanna Matouli
Course:	Advanced sensors	Date:	Friday, 8 December 2023, 4:36 PM
Book:	Exercises		

Description

Table of contents

1. Exercises
1.1. Reading Out RFID RC522 Tags (NFC)
1.2. Control a HD44780 LCD display via I2C
1.3. Using a distance sensor (ultrasonic sensor HC-SR04)sensor

1. Exercises

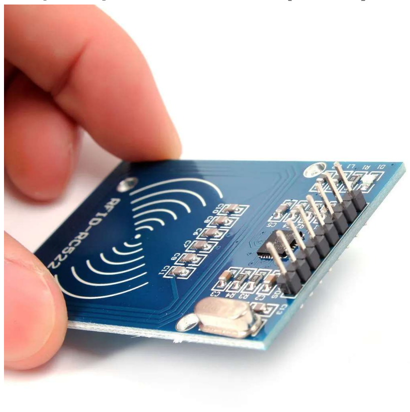
In the following chapters you can practice your skills, with three exercises about:

- Reading out RFID RC522 Tags (NFC);
- Control a HD44780 LCD display via I2C;
- Using a distance sensor (ultrasonic sensor HC-SR04)

1.1. Reading Out RFID RC522 Tags (NFC)

In the module "introduction", the basics of the Python programming language are discussed. If you are not confident with the basic syntax, please review the Python chapter in the Introduction again. This exercise uses Python and a Raspberry PI and a RFID module (RC522).

RFID is a technology whereby data is transmitted without touch, which is used in chip cards. Access cards can be read out with a Raspberry Pi RFID module (RC522) and thus e.g. access to doors or locks can be given. Current smartphones have similar ones.



In this tutorial you will learn how to read RFID tags with the RC522 and the Raspberry Pi and also write chip cards. The code presented can also be used for other projects (door opener, access control). Near Field Communications (NFC) is a related technology, the differences of which can be found here. Both RFID and NFC transmit on a frequency of 13.56 MHz, which is why the modules are compatible with each other.

Hardware Parts to be used::

- **Raspberry Pi 4* (also works with all previous versions)**
- **Mifare RC522 RFID Module* (incl. KeyCard)**
- **Female – Female Jumper Cables**
- **Soldering Utensils***

If you want to use the card reader as an entrance control, etc., it makes sense to give every user a card. You can also buy these chip cards in smaller and larger quantities* for small money and then individually write RC522 in on each card (instructions below)

Wiring

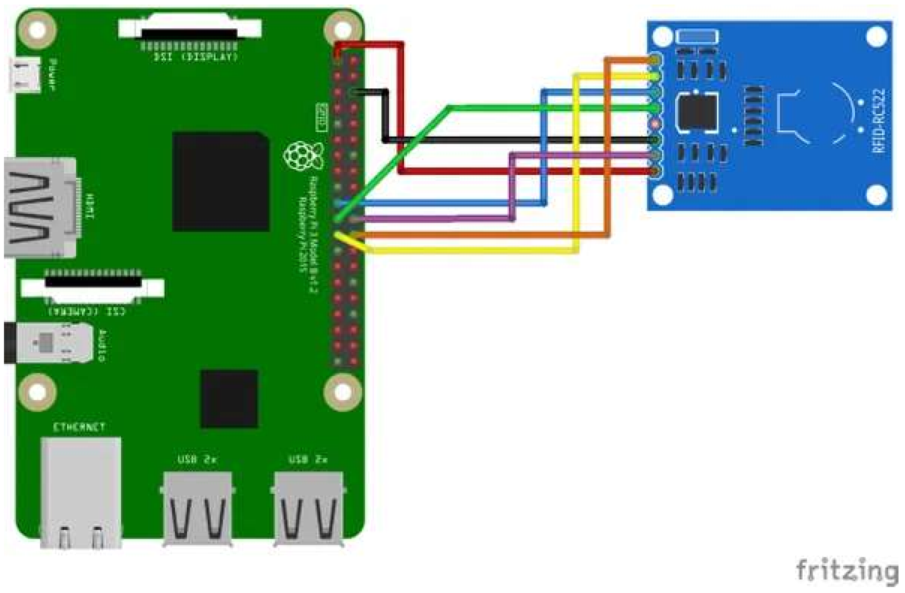
Depending on the hardware that you are using, the pin strip of the RFID module must be soldered first.

The wiring between the RFID module and the Raspberry is as follows:

RF522 Module	Raspberry Pi
SDA	Pin 24 / GPIO8 (CE0)
SCK	Pin 23 / GPIO11 (SCKL)

MOSI	Pin 19 / GPIO10 (MOSI)
MISO	Pin 21 / GPIO9 (MISO)
IRQ	—
GND	Pin6 (GND)
RST	Pin22 / GPIO25
3.3V	Pin 1 (3V3)

Schematics:



Activating SPI and Software Installation

In order to use the RFID RC522 Shield, we need the SPI bus. So that the Kernel is loaded at startup, we edit the config file:

```
sudo nano /boot/config.txt
```

The following content is added to the end of the file:

```
device_tree_param=spi=on

dtoverlay=spi-bcm2708
```

You save and exit with CTRL + O, CTRL + X. Then we activate SPI:

```
sudo raspi-config
```

Activate under "Advanced Options"> "SPI" and confirm the restart (alternatively usesudo reboot now).

Then you can use dmesg | grep spi to checked whether the module has been loaded. The output should look like this:

```
pi@raspberrypi:~ $ dmesg | grep spi

[   10.784368] bcm2708_spi 20204000.spi: master is unqueued, this is deprecated

[   10.813403] bcm2708_spi 20204000.spi: SPI Controller at 0x20204000 (irq 80)
```

Now the packages have to be installed so that we can access the SPI bus and load a corresponding library from GitHub.

```
sudo apt-get install git python-dev --yes
```

Firstly we install the Python SPI module

```
git clone https://github.com/lthiery/SPI-Py.git

cd SPI-Py

sudo python setup.py install

cd ..
```

and then the Raspberry Pi RFID RC522 library:

```
git clone https://github.com/mxgxw/MFRC522-python.git && cd MFRC522-python
```

Testing the Raspberry Pi RFID Reader/Writer

In addition to the RC522 module, a white card and an NFC-compatible key fob are usually supplied. These parts can be used as authentication because they are writable and readable. An NFC-enabled (Android/iOS) smartphone could also be used (which most newer cell phones are).

To run the first test of the card/key fob, we run the script:

```
sudo python Read.py
```

As soon as the chip card is held to it and recognized, you will see an output like this:

```
pi@raspberrypi:~/MFRC522-python $ sudo python Read.py

Welcome to the MFRC522 data read example

Press Ctrl-C to stop.

Card detected

Card read UID: 69,245,238,117

Size: 8

Sector 8 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

In order to change the stored data (numbers) on the chip, we edit the "Write.py" file (sudo nano Write.py). To do this, edit the code from line 55 as follows (you can freely choose the 16 numbers from data between 0 and 255. I represented a word with ASCII characters)

```
# Variable for the data to write
data = [114, 97, 115, 112, 98, 101, 114, 114, 121, 45, 116, 117, 116, 111, 114, 0]

# Fill the data with 0xFF
#for x in range(0,16):
#    data.append(0xFF)

print "Sector 8 looked like this:"
# Read block 8
MIFAREReader.MFRC522_Read(8)
print "\n"

#print "Sector 8 will now be filled with 0xFF:"
# Write the data
MIFAREReader.MFRC522_Write(8, data)
#print "\n"

print "It now looks like this:"
# Check to see if it was written
MIFAREReader.MFRC522_Read(8)
print "\n"

"""
data = []
# Fill the data with 0x00
for x in range(0,16):
    data.append(0x00)

print "Now we fill it with 0x00:"
MIFAREReader.MFRC522_Write(8, data)
print "\n"

print "It is now empty:"
# Check to see if it was written
```

```
MIFAREReader.MFRC522_Read(8)

print "\n"

"""
```

Using NFC/RFID reader in Raspberry Pi projects (door lock, etc.)

The two Python files “Read.py” and “Write.py” contain some sample code for reading and writing a chip which can be used in other projects. The most important file is “MFRC522.py”, which can be copied into another project.

The following excerpt can be used in other projects, e.g. as a check of a code lock or door lock. I use one authentication level (you could also set several) with a certain initial code. The last digits provide information about the holder of the card (if that data is stored somewhere). You could only identify the user by the UID, but I assume that several cards can belong to one user. If you don’t like this solution, you can, of course, change it.

You have to make a small change in the “MFRC522.py” file so that the MIFAREReader.MFRC522_Read function has a return value:

```
  def MFRC522_Read(self, blockAddr):

    recvData = []

    recvData.append(self.PICC_READ)

    recvData.append(blockAddr)

    pOut = self.CalulateCRC(recvData)

    recvData.append(pOut[0])

    recvData.append(pOut[1])

    (status, backData, backLen) = self.MFRC522_ToCard(self.PCD_TRANSCEIVE,
recvData)

    if not(status == self.MI_OK):

      print "Error while reading!"

    i = 0

    #if len(backData) == 16:

    #  print "Sector "+str(blockAddr)+" "+str(backData)

    return backData
```

The sample code then looked like this (the previous changes are important, otherwise no comparison can take place):

```
#!/usr/bin/env python

# -*- coding: utf8 -*-


import RPi.GPIO as GPIO

import MFRC522


def sample_func(sample_var):

    # Beispiel Funktion

    # Skript starten, Daten loggen, etc.

    print("Test Funktion wurde aufgerufen")


# ...


MIFAREReader = MFRC522.MFRC522()

authcode = [114, 97, 115, 112, 98, 101, 114, 114, 121] # die ersten 9 Ziffern sind der
Authentifizierungscode


try:

    while True:

        # Scan for cards

        (status,TagType) = MIFAREReader.MFRC522_Request(MIFAREReader.PICC_REQIDL)


        # If a card is found

        if status == MIFAREReader.MI_OK:

            # Get the UID of the card

            (status,uid) = MIFAREReader.MFRC522_Anticoll()

            # This is the default key for authentication

            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]

            # Select the scanned tag

            MIFAREReader.MFRC522_SelectTag(uid)

            # Authenticate

            status = MIFAREReader.MFRC522_Auth(MIFAREReader.PICC_AUTHENT1A, 8, key,
uid)

            # Check if authenticated

            if status == MIFAREReader.MI_OK:
```

```
            # Read block 8

            data = MIFAREReader.MFRC522_Read(8)

            if data[:9] == authcode:

                sample_func(data)

            #elif ...


except KeyboardInterrupt:

    print("Abbruch")

    GPIO.cleanup()
```

1.2. Control a HD44780 LCD display via I2C

LCD character displays are a simple and a cost-effective way to display a text. Thanks to the HD44780 controller, the control of the modules has become very simple. However, one must occupy many GPIOs for it. An alternative is the I2C data bus, which means that only two GPIOs are used.

In this tutorial a 20×04 HD44780 character display is controlled using a I2C display adapter. A logic converter is used to adjusting the voltage level for the module without damaging GPIOs.

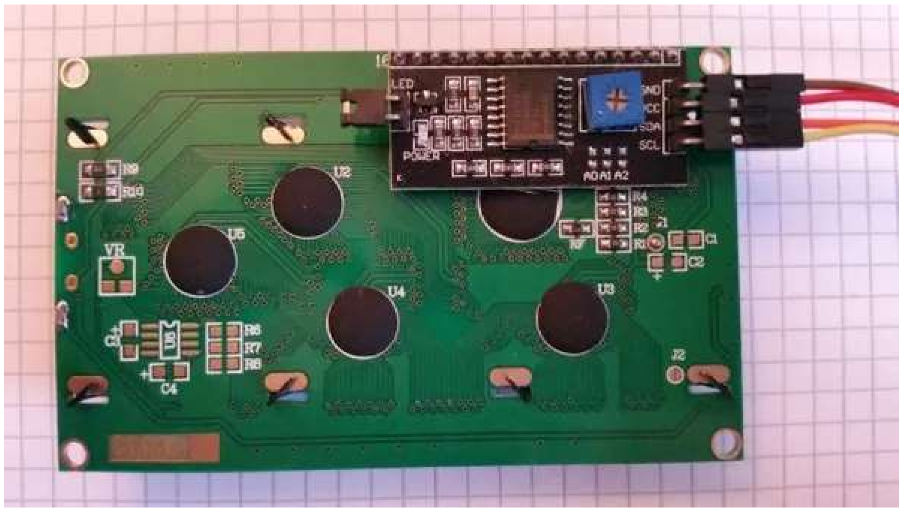
The original version of this exercise (in German) can be found at https://tutorials-raspberrypi.de/hd44780-lcd-display-per-i2c-mit-dem-raspberry-pi-ansteuern/

Source codes are available at: https://github.com/CaptainStouf/raspberry_lcd4x20_I2C.

Accessories

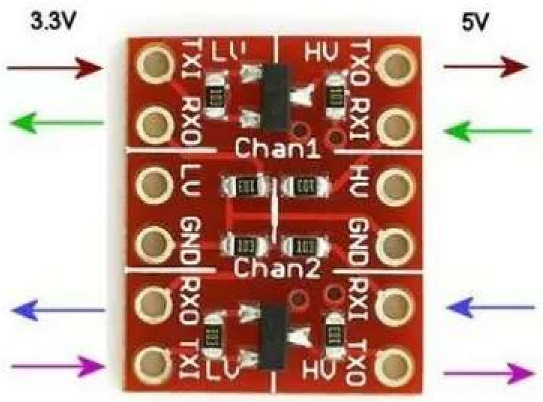
In order to access an HD47780 display via I²C, I have used the following accessories:

- **20×04 or 16×02 Character HD44780 Display**
- **I2C Display Adapter**
- **I2C Logic Level Converter**
- **Breadboard**
- **Jumper wire**



Setup

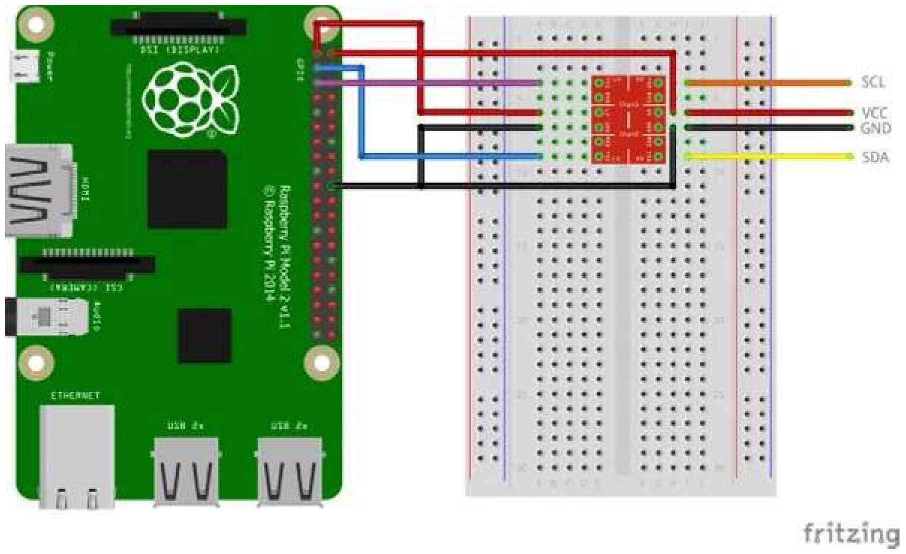
Raspberry Pi GPIOs can not get more than 3.3V voltage, but there are some modules (like this display), which send and want to receive 5V signals. For this, a Logic Level Converter can be used, which has 2 sides. On one side those connections that are running on 3.3V are connected and on the other those with 5V. You can recognize this at different characteristics (LV – HV), as you see in the following picture:



The pins are then connected as follows:

Raspberry Pi	3.3V Level Converter	5V Level Converter	I2C LCD Adapter
3.3V (Pin 1)	LV	—	—
5V (Pin 2)	—	HV	VCC
GND (Pin 6)	GND	GND	GND
GPIO2 / SDA (Pin 3)	TX1 (below)	—	—
GPIO3 / SCL (Pin 5)	TX1 (above)	—	—
—	—	TX0 (below)	SDA
—	—	TX0 (above)	SCL

Here is a schematic drawing:



Any ground pin can be taken. For the sake of clarity, I chose pin 20 instead of pin 6 on the schematic diagram.

This configuration is also usable with other modules which require signals with a higher voltage than 3.3V (in this case 5V) (real time clock, etc.).

SoftwareBefore we can start, two I²C tools are needed, which we install:
sudo apt-get install python-smbus i2c-toolsThen we will release I2C (if you have already released it from previous tutorials, you can skip it):
sudo raspi-config
Under “Interfacing Options”> “I2C” we activate it. Now add the corresponding entries to the modules file:
sudo nano /etc/modules
These two lines are added to the end:

i2c-bcm2708

i2c-dev

Afterwards it has to be restarted, so that all changes take effect.

sudo reboot

If you have already connected the display, you can now test whether it has been detected (if you have one of the first Raspberry Pi’s [Rev.1], you have to pass 0 instead of 1):

sudo i2cdetect -y 1

The output should look like this:

```
pi@raspberrypi ~ $ sudo i2cdetect -y 1

 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f

00:       -- -- -- -- -- -- -- -- -- -- -- -- --

10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

20: -- -- -- -- -- -- -- 27 -- -- -- -- -- -- -- --

30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

70: -- -- -- -- -- -- -- --
```

If you see a number other than 27, you must change this in the lcddriver.py file (ADDRESS = 0x27).

Let's start with the code:

```
mkdir hd44780 && cd hd44780

wget http://tutorials-raspberrypi.de/wp-content/uploads/scripts/hd44780_i2c.zip

unzip hd44780_i2c.zip
```

With the help of the two included scripts, the display can now be addressed. To do this, we open the Python console and enter the following code:

```
sudo python
```

```
import lcddriver

from time import *


lcd = lcddriver.lcd()

lcd.lcd_clear()


lcd.lcd_display_string("Tutorials-", 1)

lcd.lcd_display_string("  RaspberryPi.de", 2)

lcd.lcd_display_string("", 3)

lcd.lcd_display_string("HD44780 I2C Tutorial", 4)
```

The first parameter of the lcd_display_string is for the text and the second for the row. You do not have to change all lines at once, but you can not easily replace individual characters. For this, the entire text (with the character changed at the desired position) would have to be retransmitted.

The contrast of my I2C adapter was at the beginning very low. If nothing should be shown, test the wheel on the back and look obliquely on the display.

1.3. Using a distance sensor (ultrasonic sensor HC-SR04)sensor

For many (outdoor) projects a distance measurement is necessary or advantageous. These small modules are available starting at 1-2 bucks and can measure the distance up to 4-5 meters by ultrasound and are suprisingly accurate. This exercise shows the

connection and control.

Hardware

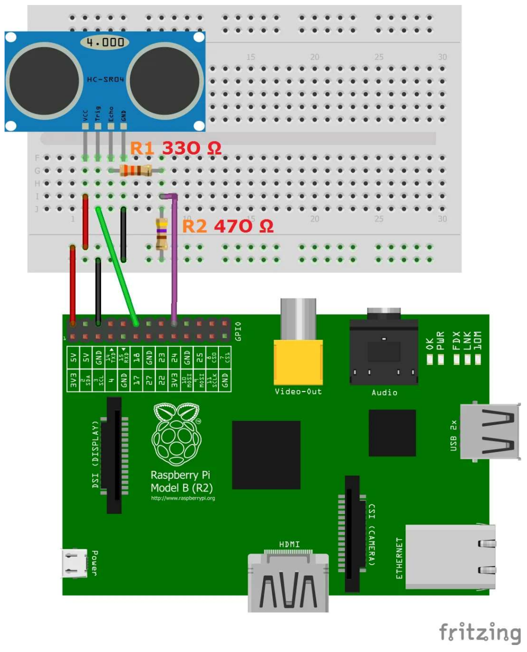
- HC-SR04 Module (US* / UK*)
- Resistors: 330Ω and 470Ω (US* / UK*)
- Jumper wire (US* / UK*)
- Wiring

There are four pins on the ultrasound module that are connected to the Raspberry:

- VCC to Pin 2 (VCC)
- GND to Pin 6 (GND)
- TRIG to Pin 12 (GPIO18)
- connect the 330Ω resistor to ECHO. On its end you connect it to Pin 18 (GPIO24) and through a 470Ω resistor you connect it also to Pin6 (GND).

We do this because the GPIO pins only tolerate maximal 3.3V. The connection to GND is to have a obvious signal on GPIO24. If no pulse is sent, the signal is 0 (through the connection with GND), else it is 1. If there would be no connection to GND, the input would be undefined if no signal is sent (randomly 0 or 1), so ambiguous.

Here is the structure as a circuit diagram:



Script for controlling

First of all, the Python GPIO library should be installed

To use the module, we create a new script

```
sudo nano ultrasonic_distance.py
```

with the following content:

```
#Libraries
import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
GPIO_TRIGGER = 18
GPIO_ECHO = 24

#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()
    # time difference between start and arrival

    TimeElapsed = StopTime - StartTime

    # multiply with the sonic speed (34300 cm/s)

    # and divide by 2, because there and back

    distance = (TimeElapsed * 34300) / 2


    return distance


if __name__ == '__main__':

    try:

        while True:

            dist = distance()

            print ("Measured Distance = %.1f cm" % dist)

            time.sleep(1)


        # Reset by pressing CTRL + C

    except KeyboardInterrupt:

        print("Measurement stopped by User")

        GPIO.cleanup()
```

After that we run:

sudo python ultrasonic_distance.py

So every second, the distance will be measured until the script is cancelled by pressing CTRL + C.

That‘s it. You can use it many fields, but who still want to measure larger distances would have to rely on laser measuring devices, which, however, are much more expensive.

Using a device to work with sensors

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Advanced sensors	Date:	Friday, 8 December 2023, 4:36 PM
Book:	Using a device to work with sensors		

Table of contents

1. Which device is suitable for prototyping?

2. Arduino Uno R3

3. Raspberry PI 4 and Raspberry 3B+

1. Which device is suitable for prototyping?

Many different devices can be used to measure something (reading sensors), to send commands (actuators) and to make decisions on what to do (for example: if the temperature in this room drops below 20 degrees, start the heater for 15 minutes).

When new solutions must be created, often a prototype is created first. Prototype devices should be easy to use, easy to extend and change, and have good connectivity.

Common examples of prototyping devices are: Raspberry Pi and Arduino. Besides being an ideal prototyping device, they are used because they are cheap, easy to acquire and portable.

When a solution is mature enough to use in production, other demands are important: device operation must be stable for many years, the device should be predictable and shielded from moisture, falling damage and touching by people. Although Raspberry Pi devices can be used in production, often PLC devices are used.

Reading tip: comparative report for Arduino vs PLC in an industrial solution by someone using both for the first time: https://www.controldesign.com/control/plcs-pacs/article/11316877/plc-vs-arduino-for-industrial-control

Raspberry Pi:

- costs about 50 euros;
- has built-in connection options such as network communication, storage, display, audio, USB;
- reads and writes digital signals, but cannot read and write analogue signals;
- uses more energy than an Arduino;
- many programming languages can be used.

Arduino:

- costs about 10 euros;
- does not have built-in connection options (network communication, storage, audio), although some can be added by using optional modules;
- reads and writes both digital and analogue signals;
- uses C as programming language;
- uses less energy than a Raspberry Pi.

PLC:

- reliable and can be used in "hostile" environments (fluids, gas);
- costs about 100-1,000 euros;
- 2 textual programming languages: Structured Text (ST; similar to Pascal) and Instruction List (IL); as well as 3 graphical languages: Ladder Diagram, Function Block Diagram (FBD) and Sequential Function Chart (SFC). Instruction List (IL) was deprecated in the third edition of the standard.

2. Arduino Uno R3

Arduino Uno Rev3 (see: http://arduino.cc for more details)

Figure: Arduino Uno

The Arduino is a microcontroller platform, available in many different form factors and sizes, mounted on a PCB that plugs easily into most computers' USB ports. It allows the user to program the onboard Atmega chip to do various things via a C-like programming language in programs called sketches.

The Arduino is not as powerful as the Pi when it comes to computing power, but it is also completely different, as it is a microcontroller, not a computer. The two machines do complement each other well. In this module we will focus on the Raspberry Pi as a computer to develop different kind of solutions.

The big difference between the two is that the Raspberry Pi has an operating system, like a Windows or Apple computer, and can be used like that: you can add a mouse, a keyboard and a monitor and even a printer to it. It can do a lot of things at the same time.

An Arduino is much simpler. You can light up LEDs, read values from all kinds of sensors and you can program it to do all kinds of actions with these sensors, but it has no operating system. An Arduino can only do one thing after another. You must tell (program) it what to do next.

3. Raspberry PI 4 and Raspberry 3B+

Afbeelding met tekst, elektronica, circuit Automatisch gegenereerde beschrijving

Figure: Raspberry PI 4 (left) and PI 3B (right)

The Raspberry Pi Foundation released an updated version of the Pi (version 4) as the successor to the 3 B+. This newer version offers a few upgrades to the original 3B+ version, including dual-band Wi-Fi AC, a slightly faster CPU (4x1.5GHz), USB 3.0 and power-over-Ethernet (PoE) capabilities. As this version is still very new, its form factor is almost identical to the original version 3. The size of the Pi has not changed over the years; the Pi 3 has the same dimensions as the Pi 1: 85.6mm x 56mm x 21mm. The Pi Zero and Zero W (smaller editions of the Raspberry Pi) are a bit smaller: 30mm x 65mm x 3.5mm (not having USB and Ethernet ports makes a huge difference in thickness). The newest Pi is a bit heavier—46 grams versus the original's 31 grams.

GPIO

As you can see in Figure 1, there is a lot packed onto the board's small space. Running along the top is one of the biggest improvements from the Pi's early version to the current models: the increase from 26 to 40 GPIO (General Purpose Input/Output) pins. These pins allow you to connect the Pi to any number of physical extensions, from LEDs and servomotors to motor controllers and extension boards (often referred to as "hats"). With a normal desktop or laptop, interfacing with physical devices like those is virtually impossible, as the serial port has all but disappeared on newer devices, and not everybody can write low-level device drivers for the USB port. The Pi, however, comes with pre-installed libraries that allow you to access the pins using Python, C, or C++, and there are additional libraries (e.g., PiGPIO and ServoBlaster) available if you do not want to use the pre-installed versions. In this module we will focus on Python as a programming language for the Raspberry Pi and PiGPIO as a module to connect sensors to the board.

Afbeelding met tekst, elektronica, schermafbeelding Automatisch gegenereerde beschrijving

Figure: GPIO Pin layout Raspberry Pi 4

USB & Ethernet

The next thing we come to along the outside edge is the two pairs of USB ports and the Ethernet port. These are both connected to the hip just to the left of the USB port, which supplies USB3.0 on the Pi 4 and USB2.0 on the 3B+,10/100/1000 Ethernet connectivity on the Pi 4 and 10/100 Ethernet connectivity on the Pi 3B+. As with all other Pis, the chip acts as a USB-to-Ethernet adapter, which is what allows the onboard Ethernet to work.

Audio Jack

The 3.5mm audio jack on the board can be used with any standard pair of headphones. HDMI sound is delivered, if available, via the HDMI connectors.

HDMI

Next to the camera board connector is the Pi's HDMI (High-Definition Multimedia Interface) port. Many Pi aficionados argue that this is where the Pi distinguishes itself from the early stages, as it has always been able to display high-definition graphics. The newest version of the Pi has a 500MHz Broadcom VideoCore VI GPU on board, enabling it to produce 4K HD video at up to 60fps. It can support Blu-ray quality playback and supports OpenGL and OpenVG libraries on the chip, and while it does not have H.265 decoding hardware, the GPU runs fast enough to be able to decode H.265 in software. The Pi 4 supports up to two displays via two micro-HDMI connectors.

Power

Continuing clockwise, we come to the USB-C power input port. Like previous versions of the Pi, you can probably use a standard mobile phone charger to power your Pi, but make sure it can source at least 2A. The Pi 3 may not use that much current on its own, but it definitely can if four devices are plugged into the four USB ports.

System on a Chip

The most important piece on the whole Pi is the large black chip in the middle, also referred to as an SoC, or System on a Chip. The Pi's chip is a Broadcom BCM2711, with a 1.5GHz ARM Cortex A72 quad-core cluster.

SD Card

Finally, on the bottom of the board, is the microSD card slot. One of the Pi's greatest space-saving features is its lack of a real hard drive. The SD card acts like a solid-state drive (SSD). This form factor has varied over the course of the Pi's versions; the current version takes microSD cards only and is not spring-loaded. You will need to use at least a 4GB card to get a minimum install of Raspbian (the Pi's preferred OS) to work on the Pi, and 8GB or higher is recommended.

Programming IoT with Python on a Raspberry Pi

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Advanced sensors	Date:	Friday, 8 December 2023, 4:38 PM
Book:	Programming IoT with Python on a Raspberry Pi		

Table of contents

1. Using Python as IoT programming language

2. Arguments for using Python for IoT

3. Book: "The Coder's Apprentice" by Pieter Spronck

4. Project: physical computing by Raspberry Pi Foundation

5. Exercise: operating a "traffic light" from Python code

6. Exercise: operating a "traffic light" from Python code (answer)

1. Using Python as IoT programming language

The DTAM "Introduction" module includes a chapter on "Python & Database fundamentals".   In it, Python is introduced as a useful, multipurpose programming language, suitable for working with big data sets, machine learning, web development, and many other purposes. We assume you have learned the programming basics in Python.

In this module, we will extend your knowledge and teach you the skills needed to develop IoT solutions with Python running on a Raspberry Pi. It is very easy to develop Internet of Things (IoT) applications with Python. This chapter will explain why Python is a good choice for IoT and will introduce you to using Python as an IoT programming language.

As explained in the previous chapter, the Raspberry is an easy-to-use device, capable of running Python and connecting to a wide range of sensors and to many other devices over the Internet.

2. Arguments for using Python for IoT

Why use Python on the Internet of Things? A Python programmers tutorial states: "For many developers, Python is considered as the language of preference in the market. It is simple to learn, has clean syntax, and has a large online community supporting it. Python becomes a great choice when it comes to IoT." (https://www.javatpoint.com/internet-of-things-with-python).

In this 4-minute video: "

" (4:06, by "Programming with Mosh", 23 Oct. 2018), several other arguments are given:
- fastest growing programming language;
- easy to learn, also for people who are not programmers;
- lots of libraries (very useful because new hardware sensors are developed quickly)

In the world of scripting languages, Python is a relative newcomer to the scene, though it is not as new as many people believe. It was developed in the late 1980s, perhaps 15 years after the conception of Unix. It was implemented in December 1989 by its principal author, Guido Van Rossum. He has remained active in Python's development and progress, and his contributions to the language have been rewarded by the Python community, which gifted him the title Benevolent Dictator for Life (BDFL). Python's philosophy has always been to make code readable and accessible. That philosophy is summed up in Python's "PEP 20 (The Zen Of Python)" document (found here: https://peps.python.org/pep-0020/), which reads as follows:

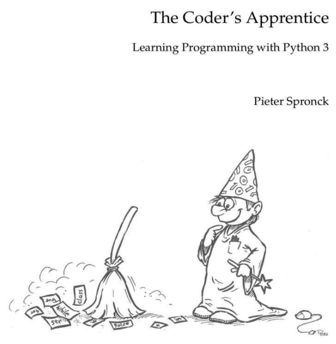
- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one -- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.

- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

In addition to these commandments, Python has a "batteries included" mindset, which means that whatever strange task you need to do in Python, chances are that a module already exists to do just that, so you don't have to reinvent the wheel.

3. Book: "The Coder's Apprentice" by Pieter Spronck

In this course we will reference parts of the freely available book "The Coder's Apprentice" by Pieter Spronck. "The Coder's Apprentice" is a course book, written by Pieter Spronck, aimed at teaching Python 3 to students who are completely new to programming. Contrary to many of the other books that teach Python programming, this book assumes no previous knowledge of programming on the part of the students and contains numerous exercises that allow students to train their programming skills



(source: https://www.spronck.net/pythonbook/index.xhtml).
The English and Dutch translations of this book can be downloaded from https://www.spronck.net/pythonbook/index.xhtml.

The last available version is 1.0.16 (2017), so the book is a number of years old.
Yet the contents are still valid and very useful.

4. Project: physical computing by Raspberry Pi Foundation

As a way of helping you to get started using a Raspberry PI with Python for physical computing, the Raspberry Pi Foundation has released a number of projects to discover how to use Python code to read sensor data from GPIO pins and write commands to actuators, displays and LEDs. In this module we use some parts from this project. The full project is available in English free of charge and can be found at: https://projects.raspberrypi.org/en/projects/physical-computing. You can use the project using this link and follow the steps. Or you can use the same steps in this chapter.

The Raspberry comes pre-installed with Python 3, many Python modules and an editor. Depending on the version of Raspbian, the editor is either "mu" or "Thonny".

It is possible, of course, to write code on another device (like a laptop) and then transfer your code file to the Raspberry for execution.

As well as a Raspberry Pi with an SD card and the usual peripherals, you will also need:

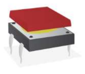
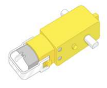
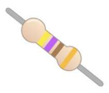
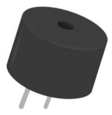
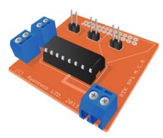
1x Solderless breadboard	**Male-to-female jumper leads**	**Female-to-female jumper leads**	**Male-to-male jumper leads**
1x Tactile button	**3x LEDs**	**Ultrasonic distance sensor**	**Passive infrared motion sensor**
			
Light Dependent Resistor	**5V Motor**	**3x 330Ω Resistor**	**470Ω Resistor**
			
1x 1µF Capacitor	**Buzzer**	**Motor Controller**	
			
Battery Pack	**1x MCP3008 ADC**	**Potentiometer**	

Figure 9: List of required components

Python code for switching an LED on and off

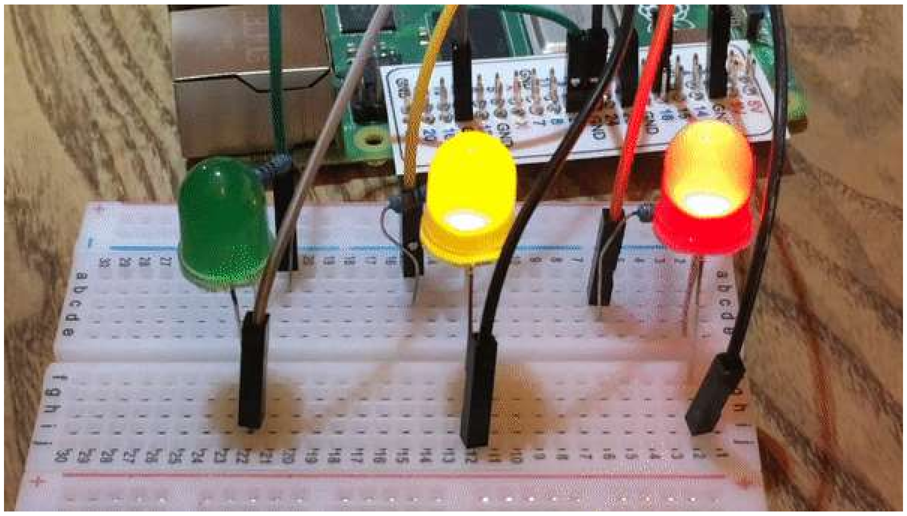


Figure 10: Raspberry Pi, GPIO breakout board, breadboard, LEDs

One powerful feature of the Raspberry Pi is the row of GPIO pins along the top edge of the board. GPIO stands for General-Purpose Input/Output. These pins are physical interfaces between the Raspberry Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off (input) or that the Pi can turn on or off (output).

The GPIO pins allow the Raspberry Pi to control and monitor the outside world by being connected to electronic circuits. The Pi is able to control LEDs, turning them on or off, run motors, and many other things. It can also detect whether a switch has been pressed, as well as the temperature and light. We refer to this as physical computing.

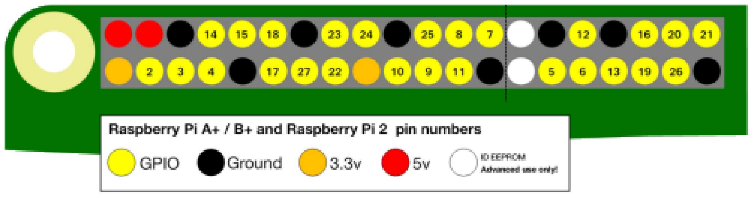


Figure 11: Raspberry PI GPIO pin layout

There are 40 pins on the Raspberry Pi (26 pins on early models), which provide various functions. LEDs are delicate little things. If you put too much current through them, they will pop (sometimes quite spectacularly). To limit the current going through the LED, you should always use a resistor in series with it.

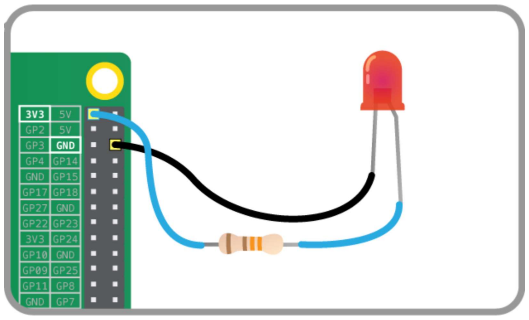
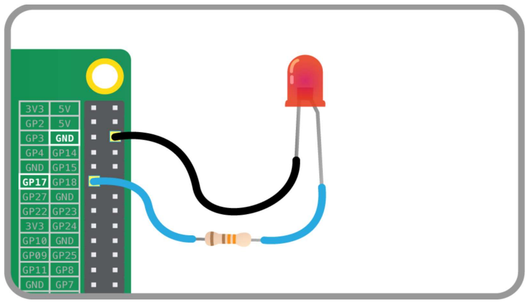


Figure 12: LED, resistor, wires

- Try connecting the long leg of an LED to the Pi's 3V3 and the short leg to a GND pin. The resistor can be anything over about 50Ω.

The LED should light up. It will always be on, because it is connected to a 3V3 pin, which is itself always on.

- Now try moving it from 3V3 to GPIO pin 17:



The LED should turn off, but now it is on a GPIO pin and can therefore be controlled by code.

GPIO Zero is a new Python library (module) that provides a simple interface to everyday GPIO components. It comes installed by default in Raspbian. See "The Coder's Apprentice", chapter 5.3 "modules" to read more about importing modules.

- Connect your Raspberry to a power supply, monitor, keyboard and mouse
- Wait until the Raspberry is fully booted (you will see a desktop, with the Raspberry icon at the top left)
- Open the Raspberry menu at the top left, choose "Programming" and then "Thonny Python IDE"
- The Thonny window will open with a new untitled file
- Type the following lines into Thonny:

```
from gpiozero import LED
led = LED(17)
led.on()
```

- Save the file as “led.py”
- Click the "Run" button (green arrow) and watch the LED light up.

Python code for flashing an LED

You can make the LED flash with the help of the time library and a little loop.

- Create a new file by clicking New.
- Save the new file by clicking Save. Save the file as gpio_led.py.
- Enter the following code to get started (note the TAB character before lines 5-8):

```
from gpiozero import LED

from time import sleep

led = LED(17)

while True:

    led.on()

    sleep(1)

    led.off()

    sleep(1)
```

- Save the file and run the code with by clicking Run.

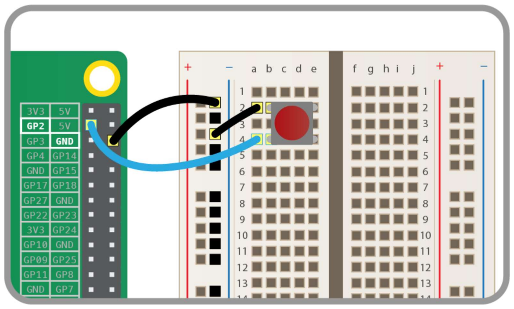
The LED should be flashing on and off. Click Stop to exit the program.

- Now try to connect your wire to another GPIO pin and adjust the code to use the new PIN

Python code for using buttons to get input

Now that you are able to control an output component (an LED), let's connect and control an input component: a button.

Connect a button to another GND pin and GPIO pin 2, like this:



- Create a new file by clicking New.
- Save the new file by clicking Save. Save the file as gpio_button.py.

This time you will need the Button class and tell it that the button is on pin 2. Write the following code in your new file:

```
from gpiozero import Button

button = Button(2)
```

Now you can get your program to do something when the button is pushed. Add these lines:

```
button.wait_for_press()

print('You pushed me')
```

- Save (gpio_button.py) and run the code.
- Press the button and your text will appear.

If your code does not work, double-check if you connected the button wires to the correct GPIO pins. Also, check if Python generated an error message.

Python code for making a light switch

With a switch, a single press and release on the button would turn the LED on, and another press and release would turn it off again.

- Modify your code so that it looks like this:

```
from gpiozero import LED, Button
from time import sleep

led = LED(17)
button = Button(2)

while True:
    button.wait_for_press()
    led.toggle()
    sleep(0.5)
```

led.toggle() switches the state of the LED from on to off, or off to on. Since this happens in a loop, the LED will turn on and off each time the button is pressed.

It would be great if you could make it so that the LED switch switches on only when the button is being held down. With GPIO Zero, that is easy. There are two methods of the Button class, called when_pressed and when_released. These do not block the flow of the program, so if they are placed in a loop, the program will continue to cycle indefinitely.

- Modify your code to look like this:

```
from gpiozero import LED, Button
from signal import pause

led = LED(17)
button = Button(2)

button.when_pressed = led.on
button.when_released = led.off

pause()
```

- Save and run the program. Now when the button is pressed, the LED will light up. It will turn off again when the button is released.

Python code for using a buzzer

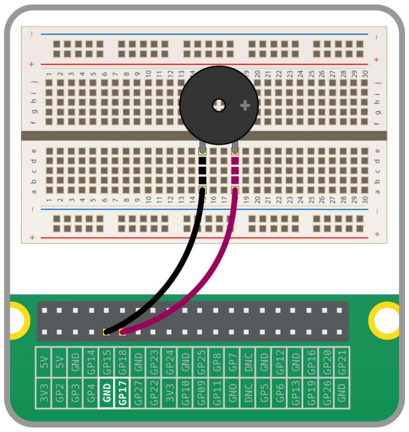
There are two main types of buzzer: active and passive.

A passive buzzer emits a tone when a voltage is applied across it. It also requires a specific signal to generate a variety of tones. As active buzzers are a lot simpler to use, these are covered here.

Connecting a buzzer

An active buzzer can be connected just like an LED, but as they are a little more robust, you do not need a resistor to protect them.

Set up the circuit as shown below:



Add Buzzer to the from gpiozero import... line:

```
from gpiozero import Buzzer

from time import sleep
```

Add a line below your creation of button and lights to add a Buzzer object

```
buzzer = Buzzer(17)
```

In GPIO Zero, a Buzzer works exactly like an LED, so try adding a buzzer.on() and buzzer.off() into your loop:

```
while True:

    buzzer.on()

    sleep(1)

    buzzer.off()

    sleep(1)
```

A Buzzer has a beep() method that works like an LED's blink. Try it:

```
while True:

    buzzer.beep()
```

Python code for using a light-dependent resistor

Analogue inputs

In the world of electrical engineering, there are two types of input and output (I/O): analogue and digital. Digital I/O is fairly easy to understand; it is either on or off, 1 or 0.

When talking about voltages and the Raspberry Pi, any input that is approximately below 1.8V is considered off and anything above 1.8V is considered on. For output, 0V is off and 3.3V is on.

Analogue I/O is a little trickier. With an analogue input, we can have a range of voltages from 0V up to 3.3V, and the Raspberry Pi is unable to detect exactly what that voltage is.

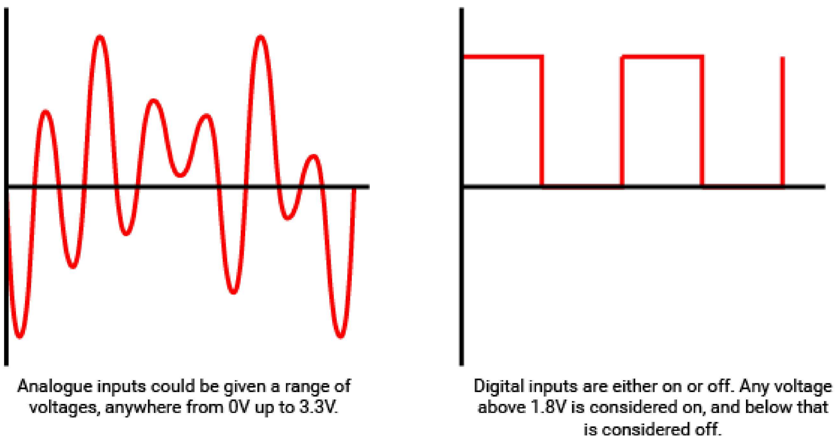
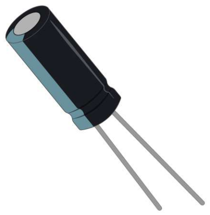


Figure 13: Analogue (left) and digital (right) signals

How, then, can we use a Raspberry Pi to determine the value of an analogue input, if it can only tell when the voltage to a GPIO pin goes above 1.8V?

Using a capacitor for analogue inputs

Capacitors are electrical components that store charge.



When current is fed into a capacitor, it will begin to store charge. The voltage across the capacitor will start off low, and increase as the charge builds up.

By putting a resistor in series with the capacitor, you can slow the speed at which it charges. With a high resistance, the capacitor will charge slowly, whereas a low resistance will let it charge quickly.

If you time how long it takes the capacitor's voltage to get over 1.8V (or be on), you can work out the resistance of the component in series with it.

Many, but not all, capacitors are polarised, which means they have a positive and a negative leg. In this case, the negative leg is shorter and should be marked with a '-' symbol.

Light-dependent resistors

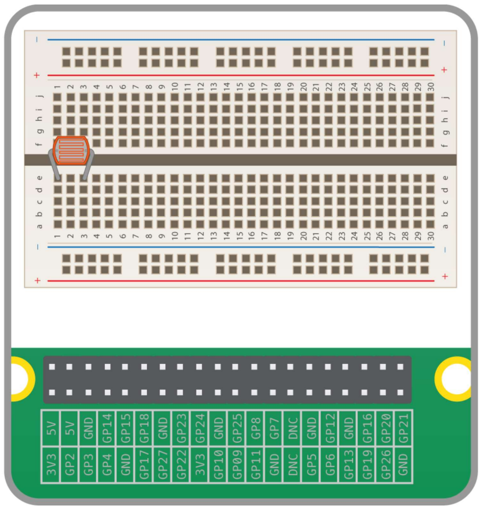
An LDR (sometimes called a photocell) is a special type of resistor.

When light hits the LDR, its resistance is very low, but when it is in the dark its resistance is very high.

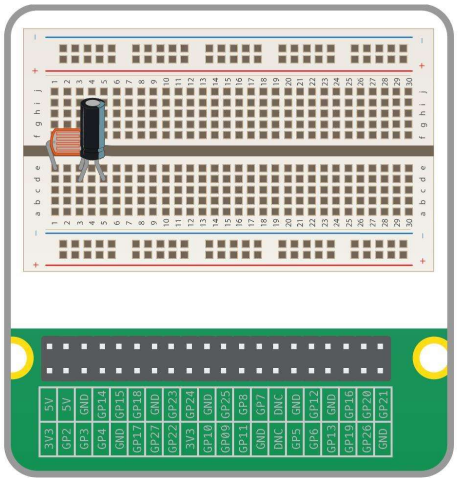
By placing a capacitor in series with an LDR, the capacitor will charge at different speeds, depending on whether it is light or dark.

Creating a light-sensing circuit

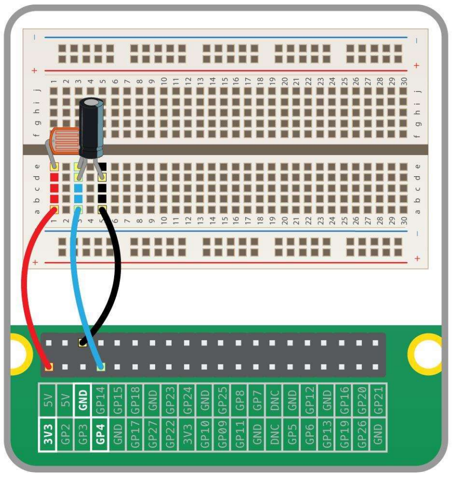
- Place an LDR into your breadboard, as shown below:


5V 5V GND GP14 GP15 GP18 GND GP23 GP24 GND GP25 GP8 GP7 DNC GND GP12 GND GP16 GP20 GP21
3V3 GP2 GP3 GP4 GND GP17 GP27 GP22 3V3 GP10 GP09 GP11 GND DNC GP5 GP6 GP13 GP19 GP26 GND

- Now place a capacitor in series with the LDR. As the capacitor is a polar component, you must make sure the positive, long leg is on the same track as the LDR leg.


5V 5V GND GP14 GP15 GP18 GND GP23 GP24 GND GP25 GP8 GP7 DNC GND GP12 GND GP16 GP20 GP21
3V3 GP2 GP3 GP4 GND GP17 GP27 GP22 3V3 GP10 GP09 GP11 GND DNC GP5 GP6 GP13 GP19 GP26 GND

- Finally, add jumper leads to connect the two components to your Raspberry Pi.


5V 5V GND GP14 GP15 GP18 GND GP23 GP24 GND GP25 GP8 GP7 DNC GND GP12 GND GP16 GP20 GP21
3V3 GP2 GP3 GP4 GND GP17 GP27 GP22 3V3 GP10 GP09 GP11 GND DNC GP5 GP6 GP13 GP19 GP26 GND

Coding a light sensor

Luckily, most of the complicated code you would have to write to detect the light levels received by the LDR has been abstracted away by the gpiozero library. This library will handle the timing of the capacitor's charging and discharging for you.

Use the following code to set up the light sensor:

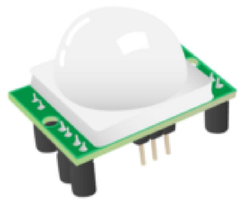
```
from gpiozero import LightSensor, Buzzer


ldr = LightSensor(4)  # alter if using a different pin

while True:

    print(ldr.value)
```

Run this code, then cover the LDR with your hand and watch the value change. Try shining a strong light onto the LDR.

Python code for using a movement (PIR) sensor

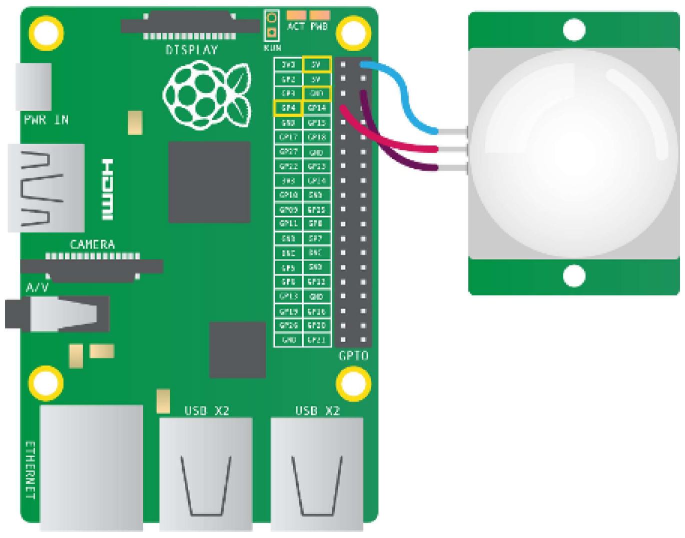
Humans and other animals emit radiation all the time. This is nothing to be concerned about, as the type of radiation we emit is infrared radiation (IR), which is pretty harmless at the levels at which it is emitted by humans. In fact, all objects at temperatures above absolute zero (-273.15°C) emit infrared radiation.

A PIR sensor detects changes in the amount of infrared radiation it receives. When there is a significant change in the amount of infrared radiation it detects, a pulse is triggered. This means that a PIR sensor can detect when a human (or any animal) moves in front of it.



Wiring a PIR sensor

The pulse emitted when a PIR detects motion needs to be amplified, and so it needs to be powered. There are three pins on the PIR; they should be labelled Vcc, Gnd and Out. These labels are sometimes concealed beneath the Fresnel lens (the white cap), which you can temporarily remove to see the pin labels.

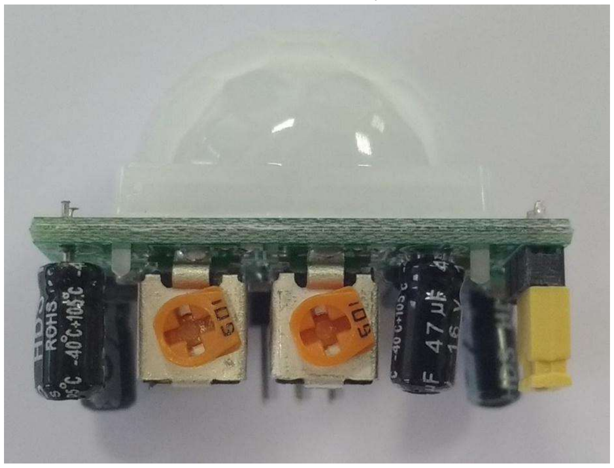


1. As shown above, the Vcc pin needs to be attached to a 5V pin on the Raspberry Pi.
2. The Gnd pin on the PIR sensor can be attached to any ground pin on the Raspberry Pi.
3. Lastly, the Out pin needs to be connected to any of the GPIO pins.

Tuning a PIR

Most PIR sensors have two potentiometers on them. These can control the sensitivity of the sensors, as well as the period of time for which the PIR will signal when motion is detected.

1. In the image above, the potentiometer on the right controls the sensitivity, and the potentiometer on the left controls the timeout. Here, both are turned fully anti-clockwise, meaning that the sensitivity and timeout are at their lowest.



When the timeout is turned fully anti-clockwise, the PIR will produce a signal for about 2.5 seconds whenever motion is detected. If the potentiometer is turned fully clockwise, the output signal will last for around 250 seconds. When tuning the sensitivity, it is best to have the timeout set as low as possible.

You can detect motion with the PIR using the code below:

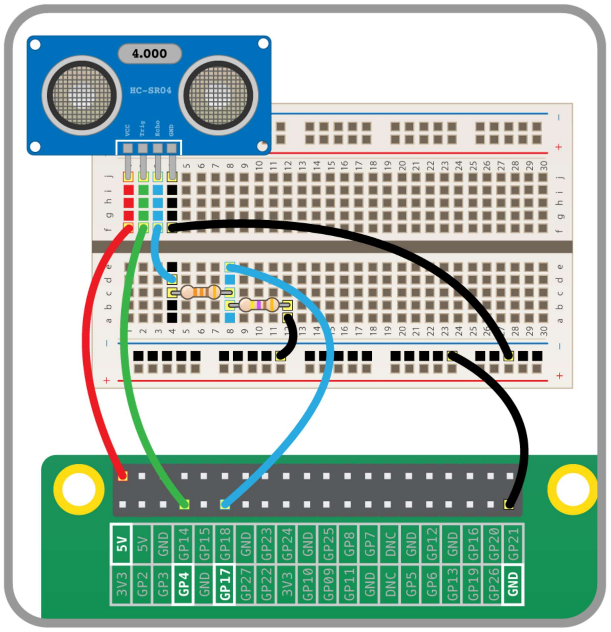
```
from gpiozero import MotionSensor

pir = MotionSensor(4)

while True:
pir.wait_for_motion()
print("You moved")
pir.wait_for_no_motion()
```

Python code for using an ultrasonic distance sensor

In air, sound travels at a speed of 343 metres per second. An ultrasonic distance sensor sends out pulses of ultrasound that are inaudible to humans, and detects the echo that is sent back when the sound bounces off a nearby object. It then uses the speed of sound to calculate the distance from the object.

Wiring

The circuit connects to two GPIO pins (one for echo, one for trigger), the ground pin and a 5V pin. You need to use a pair of resistors (330Ω and 470Ω) as a potential divider:


4.000
HC-SR04
VCC
Trig
Echo
GND
5V 5V GND GP14 GP15 GP18 GND GP23 GP24 GND GP25 GP8 GP7 DNC GND GP12 GND GP16 GP20 GP21
3V3 GP2 GP3 GP4 GND GP17 GP27 GP22 3V3 GP10 GP09 GP11 GND DNC GP5 GP6 GP13 GP19 GP26 GND

Code

To use the ultrasonic distance sensor in Python, you need to know to which GPIO pins the echo and trigger are connected.

Use the following code to read the DistanceSensor.

```
from gpiozero import DistanceSensor

ultrasonic = DistanceSensor(echo=17, trigger=4)

while True:

    print(ultrasonic.distance)
```

The value should get smaller the closer your hand is to the sensor. Press Stop to exit the loop.

Ranges

As well as being able to see the distance value, you can also get the sensor to do things when the object is in or out of a certain range.

Use a loop to print different messages when the sensor is in range or out of range:

```
while True:

   ultrasonic.wait_for_in_range()

   print("In range")

   ultrasonic.wait_for_out_of_range()

     print("Out of range")
```

Now wave your hand in front of the sensor; it should switch between showing the message "In range" and "Out of range" as your hand gets closer and further away from the sensor. See if you can work out the point at which it changes.

- The default range threshold is 0.3m. This can be configured when the sensor is initiated:
  ultrasonic = DistanceSensor(echo=17, trigger=4, threshold_distance=0.5)
  Alternatively, this can be changed after the sensor is created, by setting the threshold_distance property:
  ultrasonic.threshold_distance = 0.5
- Try the previous loop again and observe the new range threshold.

The wait_for functions are blocking, which means they halt the program until they are triggered. Another way of doing something when the sensor goes in and out of range is to use when properties, which can be used to trigger actions in the background while other things are happening in the code.
First, you need to create a function for what you want to happen when the sensor is in range:
def hello():

  print("Hello")
Then set ultrasonic.when_in_range to the name of this function:
ultrasonic.when_in_range = hello

Add another function for when the sensor goes out of range:
def bye():

  print("Bye")

  ultrasonic.when_out_of_range = bye
Now these triggers are set up, you will see "hello" printed when your hand is in range, and "bye" when it is out of range.

- You may have noticed that the sensor distance stopped at 1 metre. This is the default maximum and can also be configured on setup:
  ultrasonic = DistanceSensor(echo=17, trigger=4, max_distance=2)
  Or after setup:
  ultrasonic.max_distance = 2
- Try different values of max_distance and threshold_distance.

Python code for analogue inputs

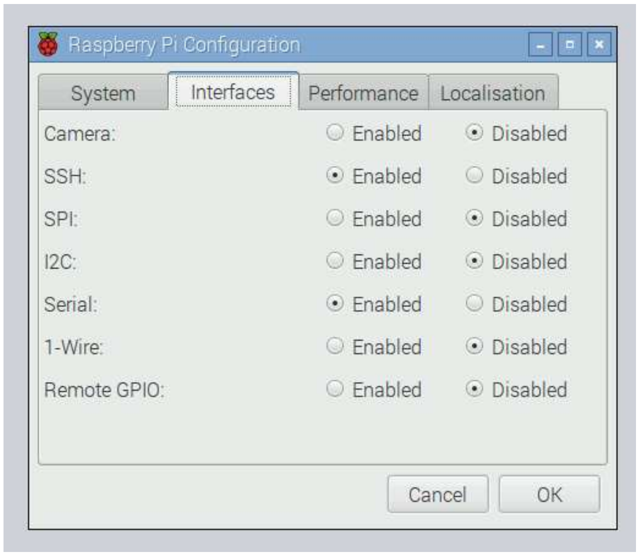
The Raspberry Pi's GPIO pins are digital pins, so you can only set outputs to high or low, or read inputs as high or low. However, using an ADC chip (Analogue-to-Digital converter), you can read the value of analogue input devices such as potentiometers.

SPI

The analogue values are communicated to the Pi using the SPI protocol. While this will work in GPIO Zero out of the box, you may get better results if you enable full SPI support.

- Open a terminal window and install the spidev package:
  sudo apt-get install python3-spidev python-spidev

- Open the Raspberry Pi Configuration dialogue from the main menu and enable SPI in the Interfaces tab:



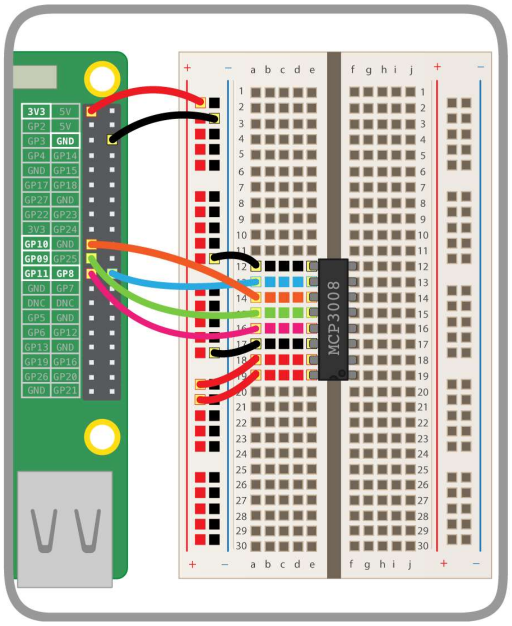
Camera:	Enabled	[✓ Disabled]
SSH:	[✓ Enabled]	Disabled
SPI:	Enabled	[✓ Disabled]
I2C:	Enabled	[✓ Disabled]
Serial:	[✓ Enabled]	Disabled
1-Wire:	Enabled	[✓ Disabled]
Remote GPIO:	Enabled	[✓ Disabled]

- Click OK and reboot the Pi.

Wiring the ADC (MCP3008)

The MCP3008 is an ADC providing eight input channels. The eight connectors on one side are connected to the Pi's GPIO pins, and the other eight are available to connect analogue input devices to read their values.

Place the MCP3008 chip on a breadboard and carefully wire it up as shown in the following diagram. You should see a small notch, or dot, in one end of the chip. In the diagram, this end of the chip is aligned with column 19 on the breadboard.



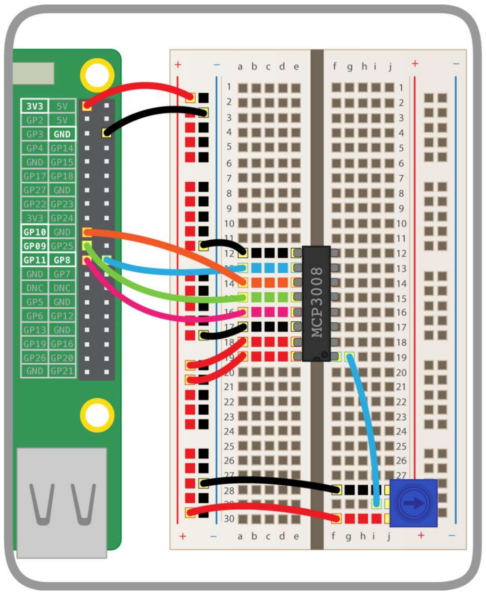
Alternatively, you could use the Analog Zero board, which provides the MCP3008 chip on a handy add-on board to save you from having to do the complicated wiring.

Add a potentiometer

Now that the ADC is connected to the Pi, you can wire devices up to the input channels. A potentiometer is a good example of an analogue input device: It is simply a variable resistor, and the Pi reads the voltage (from 0V to 3.3V).

A potentiometer's pins are ground, data and 3V3. This means you connect it to ground and a supply of 3V3, and read the actual voltage from the middle pin.

- Place a potentiometer on the breadboard and wire one side to the ground rail, the other to the 3V3 rail and the middle pin to the first input channel as shown:



Code

Now your potentiometer is connected and its value can be read from Python!

- Open Mu from the main menu.

- Start by importing the MCP3008 class from the GPIO Zero library:
  from gpiozero import MCP3008

- Create an object representing your analogue device:
  pot = MCP3008(0)
  Note that 0 represents the ADC's channel 0. There are 8 channels (0 to 7) and you are using the first one.

- Try to read its value:
  print(pot.value)

- Run your code. You should see a number between 0 and 1. This represents how far the dial is turned.

Now read the value in a loop:

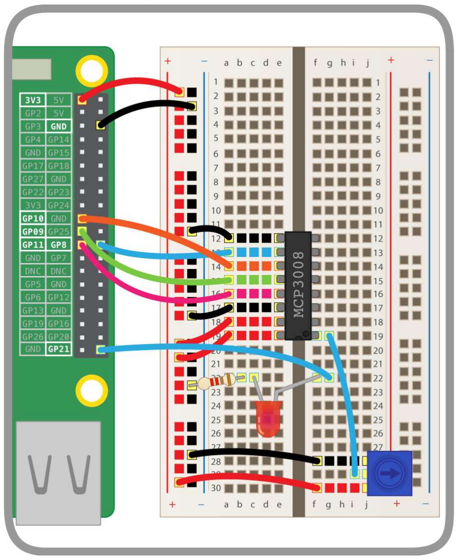
```
while True:

    print(pot.value)
```

Try twisting the dial around to see the value change.

PWMLED

Now that you have tested that you can read values from the potentiometer, you should connect it to another GPIO device.

- Add an LED to your breadboard and wire it to the Pi, connecting it to GPIO pin 21:



- In your Python code, start by importing the PWMLED class:
  from gpiozero import PWMLED
  The PWMLED class lets you control the brightness of an LED using PWM, or pulse-width modulation.

- Create a PWMLED object on pin 21:
  led = PWMLED(21)

Test whether you can control the LED manually:
led.on() # the led should be lit

led.off() # the led should go off

led.value = 0.5 # the led should be lit at half brightness

- Now connect the LED to the potentiometer:
  led.source = pot.values

- Turn the dial to change the LED brightness!

Source and values

GPIO Zero has a powerful feature: source and values. Every device has a valueproperty (the current value) and a values property (a stream of the device's values at all times). Every output device has a source property that can be used to set what the device's value should be.

- pot.value gives the potentiometer's current value (it is read only, as it is an input device)

- led.value is the LED's current value (it is read/write: you can see what it is, and you can change it)

- pot.values is a generator constantly yielding the potentiometer's current value

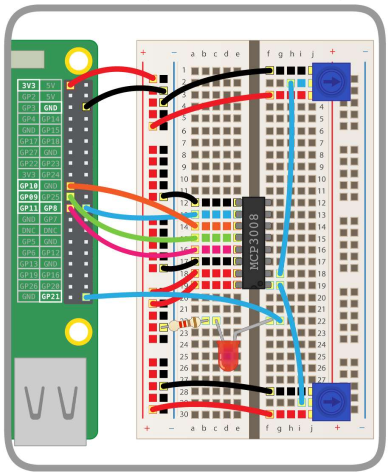
- led.source is a way of setting where the LED gets its values from

Rather than continuously setting the value of the LED to the value of the potentiometer in a loop, you can just pair the two devices. Therefore the line led.source = pot.values is equivalent to the following loop:

```
while True:

    led.value = pot.value
```

Multiple potentiometers

- Add a second potentiometer to your breadboard and connect it to the ADC's channel 1:



- Now create a second MCP3008 object on channel 1:
  pot2 = MCP3008(1)

- Make the LED blink:
  led.blink()
  The LED will blink continuously, one second on and one second off.

Change the on_time and off_time parameters to make it blink faster or slower:
led.blink(on_time=2, off_time=2)

led.blink(on_time=0.5, off_time=0.1)

Now use a loop to change the blink times according to the potentiometer values:

```
while True:

    print(pot.value, pot2.value)

    led.blink(on_time=pot.value, off_time=pot2.value, n=1, background=False)
```

Note that you have to make it blink once in the foreground, so that each iteration gets time to finish before it updates the blink times.

- Rotate the dials to make it blink at different speeds!

Also try changing blink to pulse and change on_time and off_time to fade_in_time and fade_out_time so that it fades in and out at different speeds, rather than just blinking on and off:

```
while True:

    print(pot.value, pot2.value)

    led.pulse(fade_in_time=pot.value, fade_out_time=pot2.value, n=1, background=False)
```

- Rotate the dials to change the effect.

Python code for using motors

Motors are great for physical computing: they allow you to turn a wheel forwards and backwards, or make something spin around.

A motor cannot be controlled directly from the Raspberry Pi's GPIO pins, because it needs a variable supply of 5 volts. This means you need to power it separately. Motor controller add-on boards can be used to provide this functionality.

In this guide, you will control two motors from your Raspberry Pi using Python on the desktop. First, it is best just to learn how to control the motor. Then, once you have it working, you could easily use your code to drive a Raspberry Pi-powered robot by detaching the monitor, mouse, and keyboard and building a robot around a chassis.

H bridge

A motor can be driven forwards or backwards depending on which way around current flows through it. However, it would be awkward to have to rewire a motor every time you want to change the direction it spins. To overcome this issue, motor controller boards include an H bridge. An H bridge uses 4 transistors to allow digital control of which way current flows through the motor. Most H bridges also contain flyback diodes. A flyback diode prevents the voltage spike that is generated by the motor when it is no longer powered (but still spinning) from damaging delicate electronics.

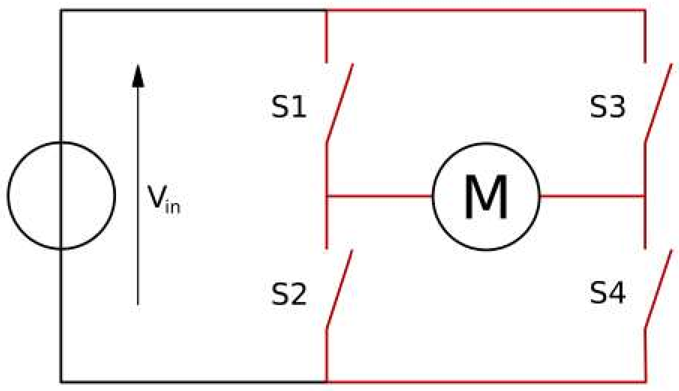
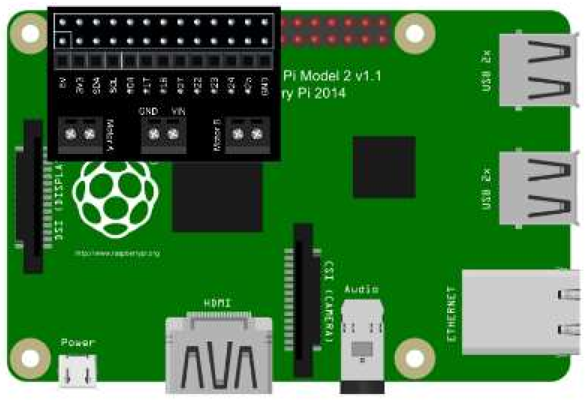


Image credit: Wikipedia, CC BY-SA

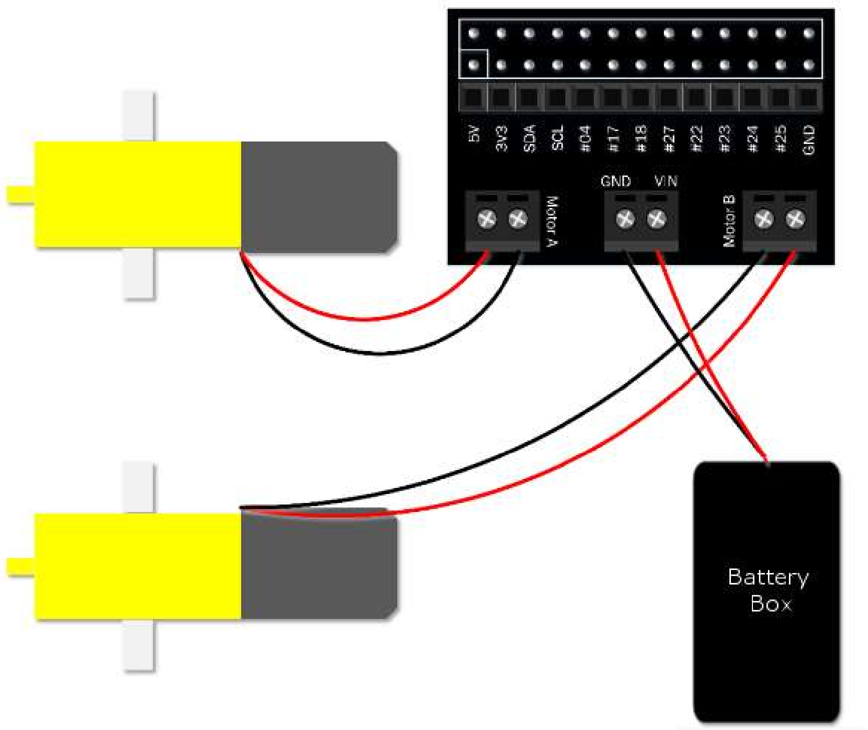
Wiring

You will need to wire up two motors and your battery pack using the motor controller.

- With your Pi switched off, mount your motor controller board on the GPIO pins:



- Connect a battery pack to the power ports of the motor controller, connecting the positive (red) battery wire to the positive (+) power terminal on the motor controller, and the negative (black) battery wire to the negative (-) power terminal on the motor controller, and connect two motors:



- You will need to know which GPIO pins your motor controller uses. Refer to the board's documentation. This will usually be described as Motor A and Motor B, or MA1, MA2, MB1 and MB2. Make a note of these pin numbers. If you are not sure which is which, you can investigate this next.

Motor class

You can use the built-in Motor class to control motors.

- *Import the Motor class:*
  from gpiozero import Motor

Now create a Motor instance using the pin numbers for each motor:
motor1 = Motor(4, 14)

motor2 = Motor(17, 27)
Note: to make it easier to see which pin is which, you can use Motor(forward=4, backward=14) for future reference.

- Now drive one of the motors forwards using the following code:
  motor1.forward()

- And the other backwards:
  motor2.backward()

Or try half speed:
motor1.forward(0.5)

motor2.backward(0.5)

The Motor class also allows you to reverse the motor's direction. Try using this loop:
motor1.forward()

motor2.backward()

while True:

  sleep(5)

  motor1.reverse()

motor2.reverse()

This will make the motors spin in opposite directions, then switch every five seconds. Press Ctrl + C to exit the loop.

Now stop the motors:
motor1.stop()

motor2.stop()

Robot class

If you had a robot with two wheels you would want to control the two motors together, rather than separately, just like you did for the two pins of each motor. Luckily, there is also a Robot class in GPIO Zero.

- Import the Robot class:
  from gpiozero import Robot

- Now create a Robot instance using the pin numbers for each motor:
  robot = Robot((4, 14), (17, 27))
  Note: to make it easier to see which pin is which, you can use Robot(left=(4, 14), right=(17, 27)) for future reference.

- Now drive one of the motors forwards using the following code:
  robot.forward()
  Both motors should now be driving forwards.

- And backwards:
  robot.backward()
  Both motors should now be driving backwards.

Try reverse a few times:
robot.reverse()

robot.reverse()

robot.reverse()

- Or try half speed:
  robot.forward(0.5)

- And that's not all! What would happen if the left wheel went forwards and the right wheel went backwards? The robot would turn right. Try it:
  robot.right()

- Then try this:
  robot.left()

- Now stop the robot:
  robot.stop()

5. Exercise: operating a "traffic light" from Python code

Using the explanation in the previous paragraph "Python code for flashing an LED", create a traffic light with a red, a green and an amber LED.

For this exercise you will need a breadboard, three LEDs, a button, a buzzer and the necessary jumper leads and resistors.

Create a Python script that will operate the traffic light as follows

- red (5 seconds)
- yellow (2 seconds)
- green (5 seconds)
- yellow (2 seconds)
- red (5 seconds)

6. Exercise: operating a "traffic light" from Python code (answer)

Wiring

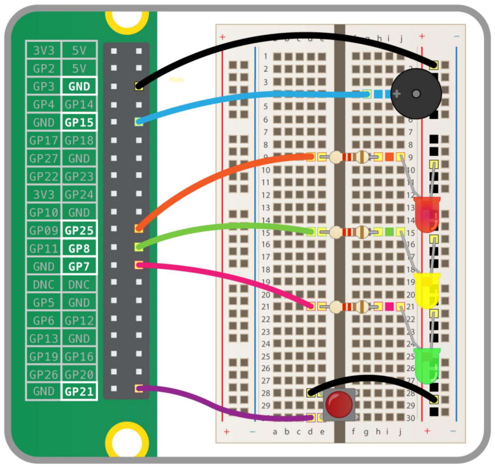
To get started, you will need to place all the components on the breadboard and connect them to the appropriate GPIO pins on the Raspberry Pi.

First, you need to understand how each component is connected:

- A push button requires 1 ground pin and 1 GPIO pin
- An LED requires 1 ground pin and 1 GPIO pin, with a current limiting resistor
- A buzzer requires 1 ground pin and 1 GPIO pin

Each component requires its own individual GPIO pin, but components can share a ground pin. We will use the breadboard to enable this.

Place the components on the breadboard and connect them to the Raspberry Pi GPIO pins, according to the following diagram:



Note that the row along the long side of the breadboard is connected to a ground pin on the Raspberry Pi, so all the components in that row (which is used as a ground rail) are connected to ground.

Observe the following table, showing to which GPIO pin each component is connected:

Component	**GPIO pin**
Button	**21**
Red LED	**25**
Amber LED	8
Green LED	7
Buzzer	15

Dive into Python

- Create a new file by clicking New.
- Save the new file straight away by clicking Save; name the file trafficlights.py.

- Enter the following code:

```
from gpiozero import Button

button = Button(21)

while True:

    print(button.is_pressed)
```

In GPIO Zero, create an object for each component used. Each component interface must be imported from the gpiozero module, and an instance must be created on the GPIO pin number to which it is connected.

- Save and run the code.
- In the shell it will be constantly printing False. When you press the button, this will switch to True, and when you release it, it will return to False.
  button.is_pressed is a property of the button object, which provides the state of the button (pressed or not) at any given time.
- Now return to the code window and modify your while loop to show the following:

```
while True:

    if button.is_pressed:

        print("Hello")

    else:

        print("Goodbye")
```

- Run the code again and you will see "Hello" printed when the button is pressed, and "Goodbye" when the button is not pressed.

Modify the loop again:

```
while True:

    button.wait_for_press()

    print("Pressed")

    button.wait_for_release()

    print("Released")
```

- When you run the code this time, nothing will happen until you press the button: you will then see "Pressed"; when you let go you will see "Released". This will occur each time the button is pressed, but rather than continuously printing one or the other, it only does it once per press.

Add an LED

Now you can add an LED into the code and use GPIO Zero to allow the button to determine when the LED is lit.

- In your code, add to the from gpiozero import... line at the top to also bring in LED:
  from gpiozero import Button, LED
- Add a line below button = Button(21) to create an instance of an LED object:
  led = LED(25)

Now modify your while loop to turn the LED on when the button is pressed:

```
while True:

    button.wait_for_press()
    led.on()

    button.wait_for_release()

    led.off()
```

- Run your code and the LED will come on when you press the button. Hold the button down to keep the LED lit.

Now swap the on and off lines to reverse the logic:

```
while True:

    led.on()

    button.wait_for_press()

    led.off()

    button.wait_for_release()
```

- Run the code and you will see that the LED stays on until the button is pressed.

Now replace led.on() with led.blink():

```
while True:

    led.blink()

    button.wait_for_press()

    led.off()

    button.wait_for_release()
```

- Run the code and you will see the LED blink on and off until the button is pressed, at which point it will turn off completely. When the button is released, it will start blinking again.
- Try adding some parameters to blink to make it blink faster or slower:
  - *led.blink(2, 2) - 2 seconds on, 2 seconds off*
  - *led.blink(0.5, 0.5) - half a second on, half a second off*
  - *led.blink(0.1, 0.2) - one tenth of a second on, one fifth of a second off*
- blink's first two (optional) parameters are on_time and off_time: they both default to 1 second.

Traffic lights

You have three LEDs: red, amber and green. Perfect for traffic lights! There is even a built-in interface for traffic lights in GPIO Zero.

- Amend the from gpiozero import... line to replace LED with TrafficLights:
  from gpiozero import Button, TrafficLights
- Replace your led = LED(25) line with the following:
  lights = TrafficLights(25, 8, 7)
  The TrafficLights interface takes three GPIO pin numbers, one for each pin: red, amber and green (in that order).

Now amend your while loop to control the TrafficLights object:

```
while True:

    button.wait_for_press()

    lights.on()

    button.wait_for_release()
    lights.off()
```

The TrafficLights interface is very similar to that of an individual LED: you can use on, off and blink, all of which control all three lights at once.

Try the blink example:

```
while True:

    lights.blink()

    button.wait_for_press()

    lights.off()

    button.wait_for_release()
```

Add a buzzer

Now you can add your buzzer to make some noise.

- Add Buzzer to the from gpiozero import... line:
  from gpiozero import Button, TrafficLights, Buzzer
- Add a line below your creation of button and lights to add a Buzzer object:
  buzzer = Buzzer(15)

Buzzer works exactly like LED, so try adding a buzzer.on() and buzzer.off() into your loop:

```
while True:

    lights.on()

    buzzer.off()

    button.wait_for_press()

    lights.off()

    buzzer.on()

    button.wait_for_release()
```

Buzzer has a beep() method which works like LED's blink. Try it out:

```
while True:

    lights.blink()

    buzzer.beep()

    button.wait_for_press()

    lights.off()

    buzzer.off()

    button.wait_for_release()
```

Traffic light sequence

As well as controlling the whole set of lights together, you can also control each LED individually. With traffic light LEDs, a button and a buzzer, you can create your own traffic light sequence, complete with pedestrian crossing!

- At the top of your file, below from gpiozero import..., add a line to import the sleep function:
  from time import sleep

Modify your loop to perform an automated sequence of LEDs being lit:

```
while True:

    lights.green.on()

    sleep(1)

    lights.amber.on()

    sleep(1)

    lights.red.on()

    sleep(1)

    lights.off()
```

Add a wait_for_press so that pressing the button initiates the sequence:

```
while True:

    button.wait_for_press()

    lights.green.on()

    sleep(1)

    lights.amber.on()

    sleep(1)

    lights.red.on()

    sleep(1)

    lights.off()
```

- **T**ry some more sequences of your own.
- Now try creating the full traffic lights sequence:
    - Green on
    - Amber on
    - Red on
    - Red and amber on
    - Green on
- Be sure to turn the correct lights on and off at the right time, and make sure you use sleep to time the sequence perfectly.
- Try adding the button for a pedestrian crossing. The button should move the lights to red (not immediately), and give the pedestrians time to cross before moving back to green until the button is pressed again.
- Now try adding a buzzer to beep quickly to indicate that it is safe to cross, for the benefit of visually impaired pedestrians.

Wired communication

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Advanced sensors	Date:	Friday, 8 December 2023, 4:39 PM
Book:	Wired communication		

Description

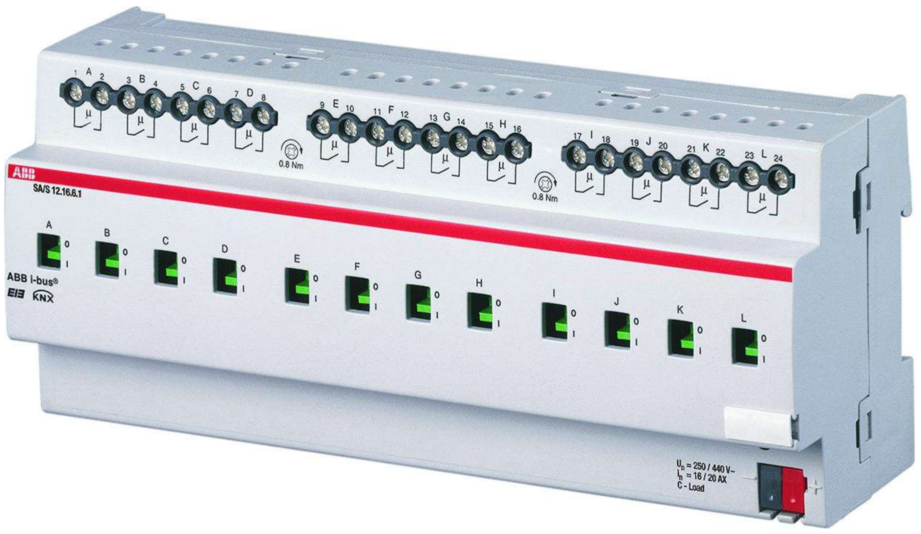
Before we compare concrete systems, it makes sense to look at the distinction between wireless and wired home automation systems. The choice between wireless and wired is an obvious starting point when choosing a suitable home automation system or industrial solution. If it is not possible to lay cables, because you do not want to mess up your house for milling pipes, most wired solutions are already eliminated in advance. In industry, it is common to use wired solutions, such as KNX and PLC.

Table of contents

1. KNX

1. KNX

KNX is the most widely used standard for building automation in commercial environments. The standard provides protocols for networks with twisted pair (knx-tp), power line (knx-pl), ethernet (knxnet/ip) and wireless (knx-rf) as communication medium. The KNX standard is a combination of the three former EHS, Batibus and EIB/Instabus standards. The standard is being developed by the KNX Association, which has more than three hundred members. Before a product is allowed to bear the KNX label, it is subjected to a conformity test by an independent test lab to ensure interoperability of the products of the different KNX manufacturers.



KNX is based on a hierarchical network topology, which is suitable for automating large buildings, such as office buildings and hotels. The highest level in the hierarchy is a domain of up to fifteen areas connected to a backbone line via backbone area couplers. Each area consists of up to fifteen lines that are connected to the backbone area coupler via a trunk line. A maximum of 256 devices can be connected to a line. A domain, including the couplers, can therefore contain a maximum of 65,536 devices.

In wired installations, twisted pair is most often used as cabling, while knxnet/ip is mainly used for the backbone connections. Twisted pair cabling can be laid in tree, line or star topology. In a home installation, most components are usually mounted centrally in the meter cupboard. KNX has a data rate of 9600bit/s. In practice this is sufficient, because the KNX telegrams are short and contain almost only control and measurement information. The bus signal is robust and can span distances of hundreds of metres.

1. PLC

A programmable logic controller (PLC) is a device that processes information from inputs and controls outputs according to a set program. A classic PLC works cyclically and first reads all inputs, then runs the program and writes the output values to a table. When the program is finished, the output values are adjusted. Depending on the size of the program, a cycle lasts a few milliseconds and then starts again. PLCs are widely used in industry to control machines, but are also well suited for building automation or other applications where input-to-output is processed. Examples of PLC systems are Wago I/O System 750, Siemens S7 and Beckhoff TwinCat.

A PLC consists of a controller that can be expanded with I/O and communication modules as desired. Manufacturer-specific software is often required to program the PLC. The initial costs of the controller and the programming software are high, but afterwards a PLC can be equipped with extensions relatively affordably. Affordability applies in particular to digital inputs and outputs, which are used, for example, to read the position of a wall switch and to control a relay. For the Wago 750 system, an extension module with four digital outputs costs just 30 euros. Analogue I/O modules are significantly more expensive than digital inputs and outputs. Expansion modules are available for various analogue signal transmission standards, such as 0-10V, 4-20mA and PT100.

Data visualizing platforms

Site:	[DTAM Online Training Platform]	Printed by:	Ioanna Matouli
Course:	Advanced sensors	Date:	Friday, 8 December 2023, 4:39 PM
Book:	Data visualizing platforms		

Table of contents

1. Data visualizing with Grafana

2. Open platform for interacting sensors (Home Assistant)

3. Processing data with Node-RED

1. Data visualizing with Grafana

Grafana is a multi-platform open-source analytics and interactive visualisation web application. It provides charts, graphs, and alerts for the web when connected to supported data sources. A licensed Grafana Enterprise version with additional capabilities is also available as a self-hosted installation or an account on the Grafana Labs cloud service.

It is expandable through a plug-in system. End users can create complex monitoring dashboards using interactive query builders. Grafana is divided into a front end and back end, written in TypeScript and Go, respectively.

As a visualisation tool, Grafana is a popular component in monitoring stacks. Learning to work with Grafana: information can be found at Grafana.com. You can play around in a sandbox.

2. Open platform for interacting sensors (Home Assistant)

Home Assistant is free and open-source software for home automation designed to be a central control system for smart home devices with a focus on local control and privacy. It can be accessed through a web-based user interface by using companion apps for Android and iOS, or by voice commands via a supported virtual assistant such as Google Assistant or Amazon Alexa.

After the Home Assistant software application is installed as a computer appliance, it will act as a central control system for home automation, commonly referred to as a smart home hub, that has the purpose of controlling IoT connectivity technology devices, software, applications and services supported by modular integration components, including native integration components for wireless communication protocols such as Bluetooth, Zigbee and Z-Wave (used to create local personal area networks with small low-power digital radios), as well as having support for controlling both open and proprietary ecosystems if they provide public access via, for example, an Open API or MQTT for third-party integrations over the Local Area Network or the Internet.

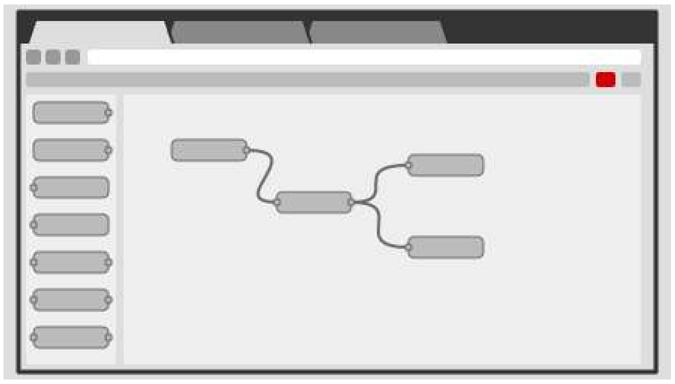
Information from all devices and their attributes (entities) that the Home Assistant software application sees can be used and controlled from within scripts trigger automations using scheduling and "blueprint" subroutines, e.g. for controlling lighting, climate, entertainment systems and home appliances. You will find lots of information on the official website: http://home-assistant.io

3. Processing data with Node-RED

As the website https://nodered.org/ explains, Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways. Node-RED is a powerful open-source tool that provides a browser-based flow editor to create event-driven applications that can run on multiple platforms such as Raspberry Pi, Arduino, or even in the cloud. It is built on top of Node.js, which is a server-side JavaScript runtime environment. Node-RED is designed to simplify the process of wiring together different devices, APIs, and services using a drag-and-drop interface, without requiring any programming skills.

It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.

Browser-based flow editing

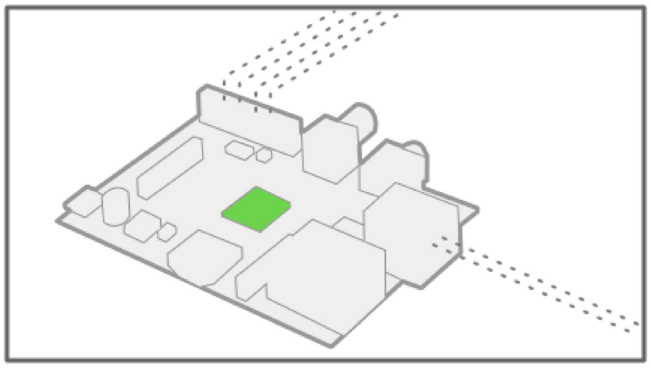


Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range of nodes in the palette. Flows can be then deployed to the runtime in a single-click.

JavaScript functions can be created within the editor using a rich text editor.

A built-in library allows you to save useful functions, templates or flows for re-use.

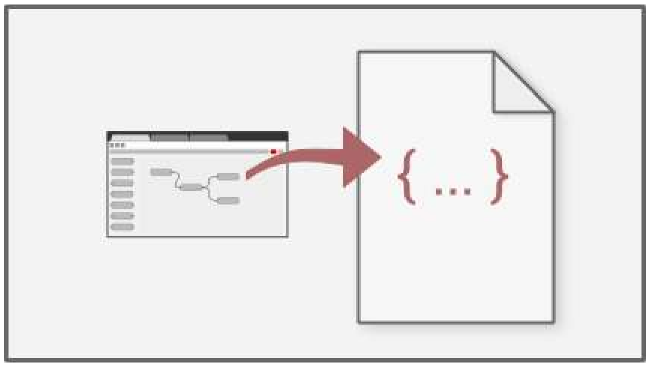
Built on Node.js



The light-weight runtime is built on Node.js, taking full advantage of its event-driven, non-blocking model. This makes it ideal to run at the edge of the network on low-cost hardware such as the Raspberry Pi as well as in the cloud.

With over 225,000 modules in Node's package repository, it is easy to extend the range of palette nodes to add new capabilities.

Social Development



The flows created in Node-RED are stored using JSON which can be easily imported and exported for sharing with others.

An online flow library allows you to share your best flows with the world.

Exercise: Connecting Siemens PLC to Raspberry Pi & Node-RED

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Advanced sensors	Date:	Friday, 8 December 2023, 4:39 PM
Book:	Exercise: Connecting Siemens PLC to Raspberry Pi & Node-RED		

Description

This exercise consists of three steps:

- Data collection from PLC;
- Analogue signal management in PLC;
- Reading PLC variables from Node-RED;

Table of contents

1. Data collection from PLC

2. Hardware and software requirements

3. Analogue signal management in PLC

4. From the input card to a numeric value

5. How to convert the numerical value to a real value

6. VALUE CONVERSION

7. PLC program for analogue signal scaling

8. SCALING USING MATH OPERATOR IN LADDER LANGUAGE

9. SCALING USING MATH OPERATOR IN SCL LANGUAGE

10. SCALING USING BUILT-IN FUNCTIONS "SCALE_X" AND "NORM_X"

11. Read PLC variables from Node-RED

12. Reading the value from PLC

1. Data collection from PLC

The DTAM project focuses on **data collection** and elaboration. The data is sourced from several kinds of sensor connected to a Raspberry Pi via a **GPIO** interface or **GROVE** boards.

To give a real experience of an Industry 4.0 scenario in the DTAM course, it is necessary to also evaluate the origin of the data from **industrial sensors passing through a PLC system or** to generally **read data from PLCs.**

A lot of PLCs have built-in functionality for data sharing, e.g. in a MMTQ or OPCUA server, but for the DTAM IoT lab structure we propose a different solution: connection of the PLC to the Raspberry Pi and data collection with **Node-RED**.

This guide will explain how to this can be done using a Siemens PLC connected to a Raspberry Pi via ProfiNet (Ethernet Protocol used by Siemens).

2. Hardware and software requirements

In addition to the IoT lab configuration, the following components and software are needed to collect data from a PLC.

Siemens PLC Simatic S7-1200 or S7-1500. Most of the 1200 family PLCs have analogue inputs available on the main board; if they are not present, an analogue input card is also required.

Ethernet cable to connect PLC and Raspberry Pi

One or more **analogue senso**rs. The type of sensor is irrelevant: the signal is always managed in the same way in the PLC.

Siemens Tia Portal programming system

Figure 1.1 - Simatic S7-1214 PLC (www.siemens.com)

3. Analogue signal management in PLC

Reading the analogue value from a sensor

The industrial standard of the analogue signals provides methods to convert the measured physical value to an electric signal readable from the PLC. The most commonly used of these are:

Voltage from 0V to 10V

Voltage from -10V to 10V

Current from 4mA to 20mA

Current from 0mA to 20mA

Other specific values for some temperature sensors

It is compulsory to then match the conversion system used by the sensor with the corresponding type of analogue input PLC card (or with the right configuration in case of a multi-standard card).

Example: Connection of a 4-20mA sensor (https://instrumentationtools.com/plc-analog-input-scaling)

In this example, a flow meter could measure a flow from 0 to 700 GPM (gallons per minutes), converting the measurement into an electric current signal. 0 GPM is converted to 4mA, 700 GPM is converted to 20mA, and all the intermediate GMP values are converted linearly in a variable current between 4 and 20 mA.

4. From the input card to a numeric value

The purpose of an analogue input PLC card is to **convert** the **electric signal** generated by sensors to a **numerical value**that can be used in the PLC program.

The format of this number depends on the method of conversion used by the PLC manufacturer, which could propose different resolution conversions and consequentially different resolutions of the final value. Refer to the manufacturer's manual for specifications.

Siemens S7-1200, used in this guide, has a conversion system with an overflow management, with the electric signal input being converted to an integer between **0 and 27684** (±27648 if the signal allows negativity). This value will be provided in the PLC input memory area, for example in the Input Word number 64 (IW64). The Word address is configurable in the input card properties.

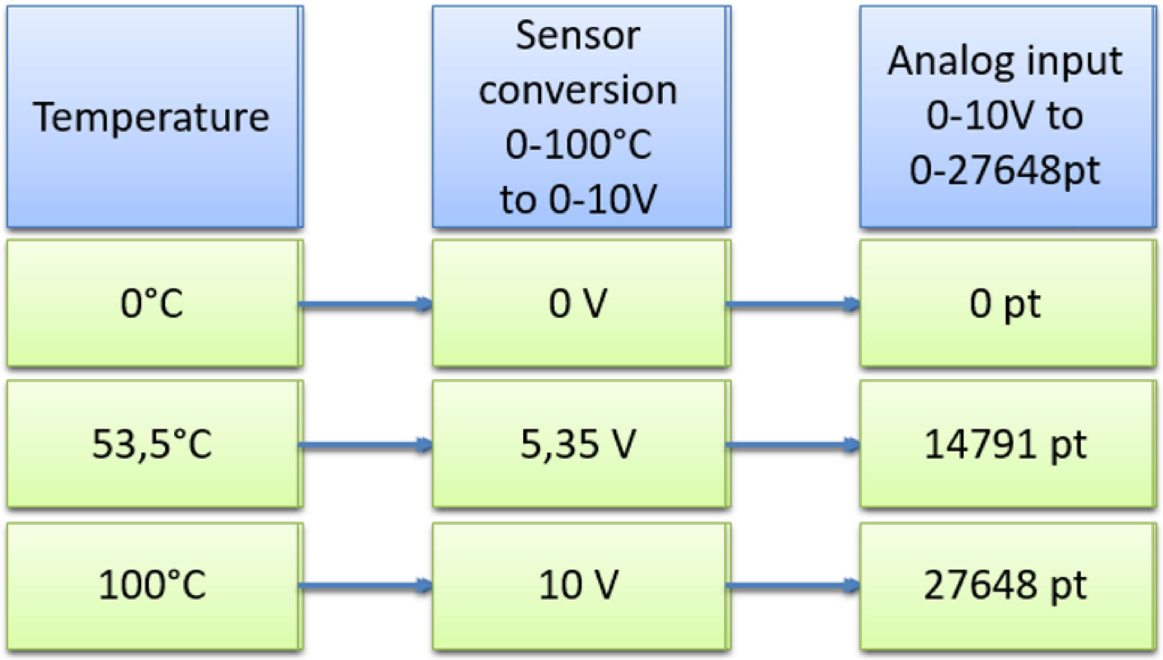

Temperature
Sensor conversion 0-100°C to 0-10V
Analog input 0-10V to 0-27648pt
0°C
0 V
0 pt
53,5°C
5,35 V
14791 pt
100°C
10 V
27648 pt

Figure 2: Example of an analogue signal conversion

5. How to convert the numerical value to a real value

The value given in the IW is relative to the sensor measurement but is not easy to interpret, so further **conversion** is required to obtain a real value representing the starting measurement.

In order to use a signal read on an analogue input, it must be converted back to a value that can be used in the system. A series of operations must be provided to obtain the value of the physical quantity again.

It is therefore necessary to know:

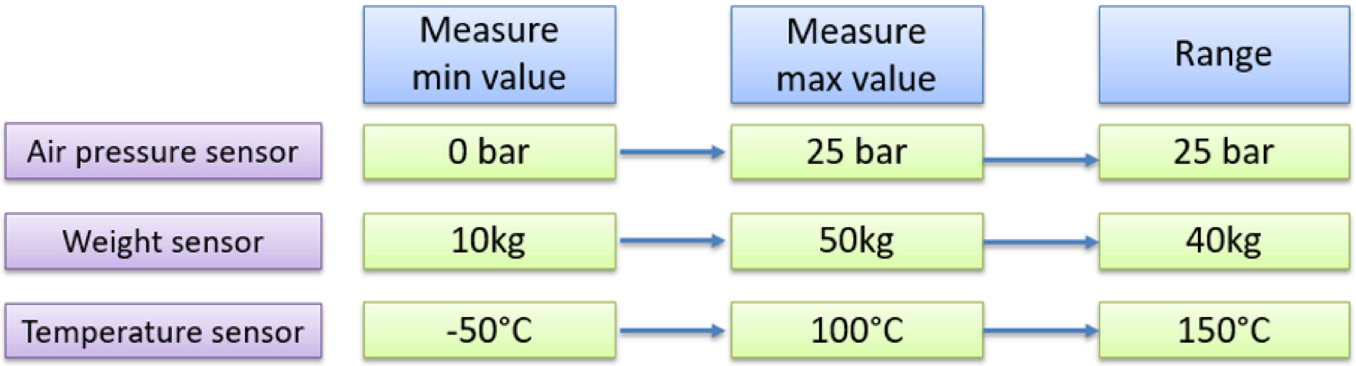
The minimum value of the signal read (from sensor characteristics)

The maximum value of the signal read (from sensor characteristics)

The number of points with which the signal is read by the controller (see manufacturers' manuals – for the S7-1200 this is 27648).

MEASUREMENT RANGE

The first step is to know the **working range** of the **measured signal**, or rather the difference between the minimum and maximum values of the signal generated by the sensor.



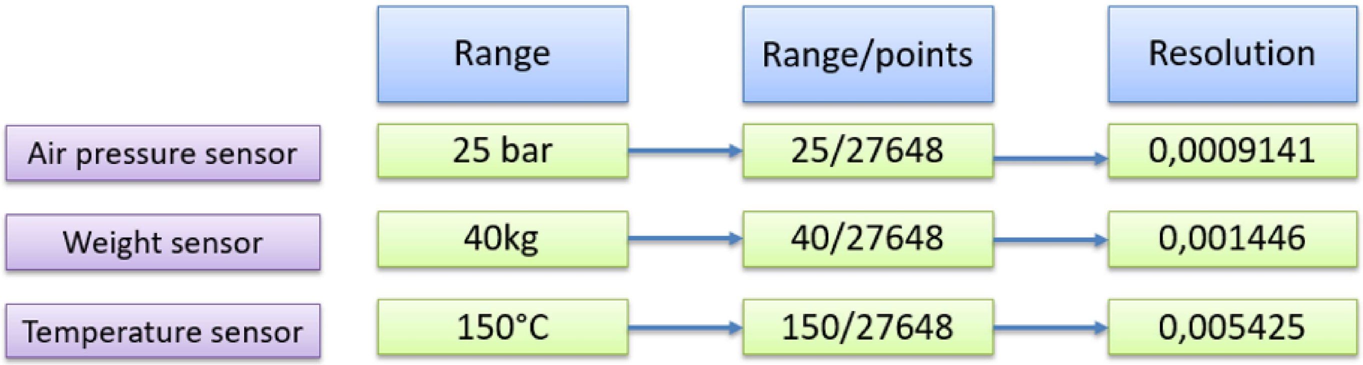
	Measure min value	Measure max value	Range
Air pressure sensor	0 bar	25 bar	25 bar
Weight sensor	10kg	50kg	40kg
Temperature sensor	-50°C	100°C	150°C

Figure 2.3- Range calculation

RESOLUTION CALCULATION

We then proceed by calculating the **resolution**, which is the value that will correspond to the **change of 1 point** on the input signal. The resolution will then be the minimum appreciable change from the signal reading, and is calculated by **dividing the range by the number of points** used by the controller to convert the signal (e.g. 27648 for S7-1200)

Note that the resolution is a decimal value. It needs to be stored as a **real value** (also known as float value in other PLCs)



	Range	Range/points	Resolution
Air pressure sensor	25 bar	25/27648	0,0009141
Weight sensor	40kg	40/27648	0,001446
Temperature sensor	150°C	150/27648	0,005425

Figure 2.4 - Resolution calculation

6. VALUE CONVERSION

After calculating the resolution it will be sufficient to multiply it by the value read on the analogue input, **obtaining the value of the physical quantity**.

However, this value still does not take into account any **OFFSET** from 0, so it will still be necessary to add the minimum value of the signal read to the result

Value = (resolution * input value) + minimum value

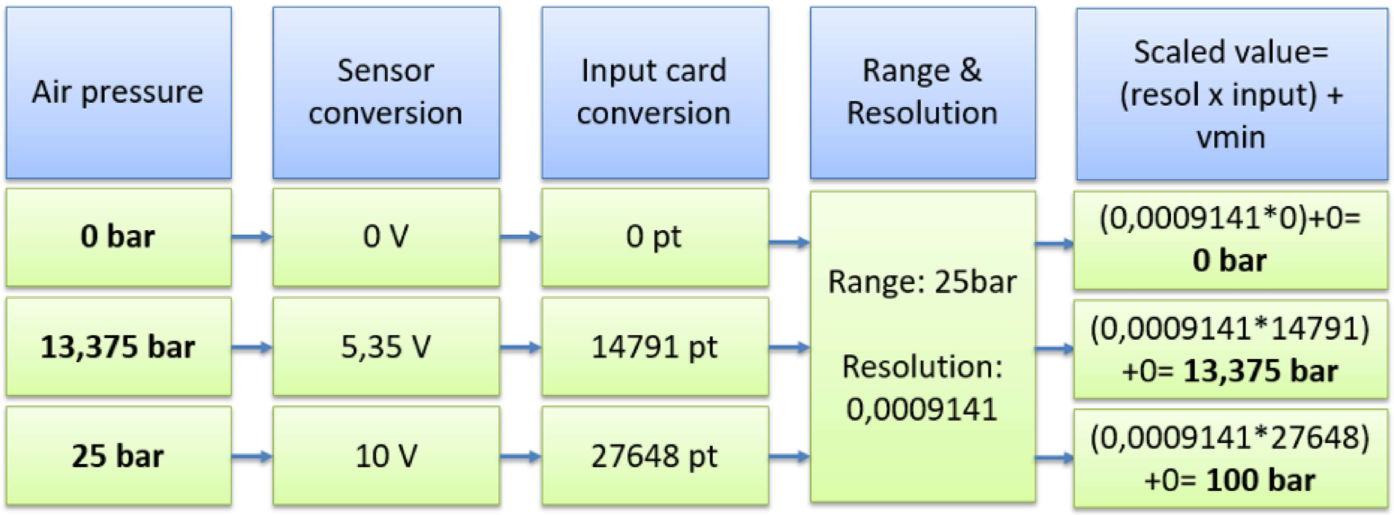


Figure 2.5 - Air pressure value scaling procedure example

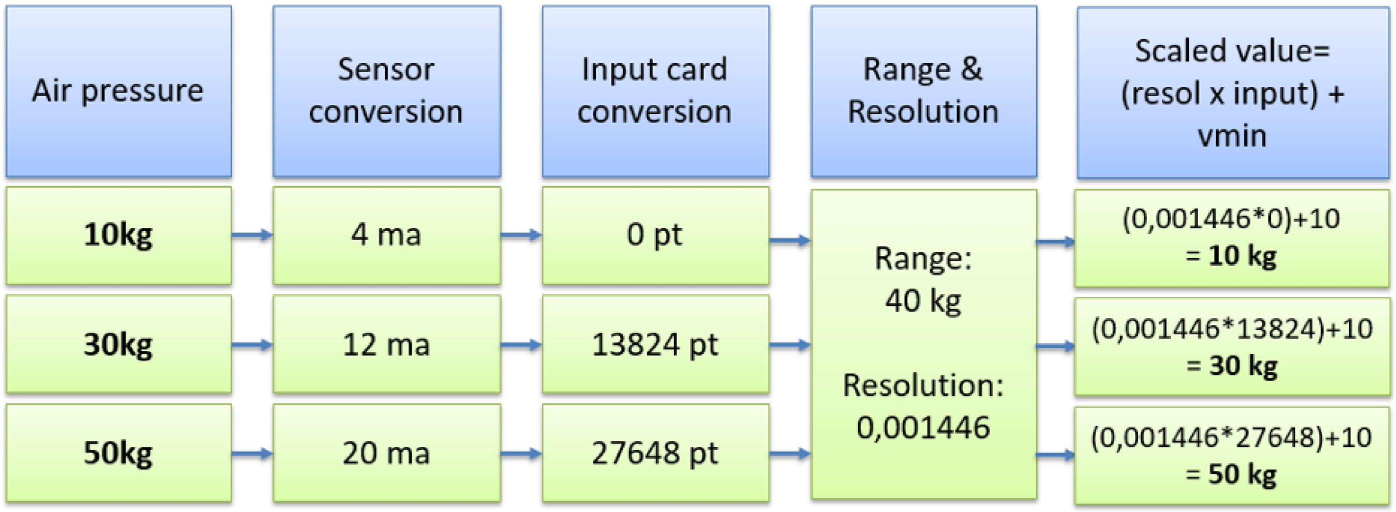


Figure 2.6 - Weight value scaling procedure example

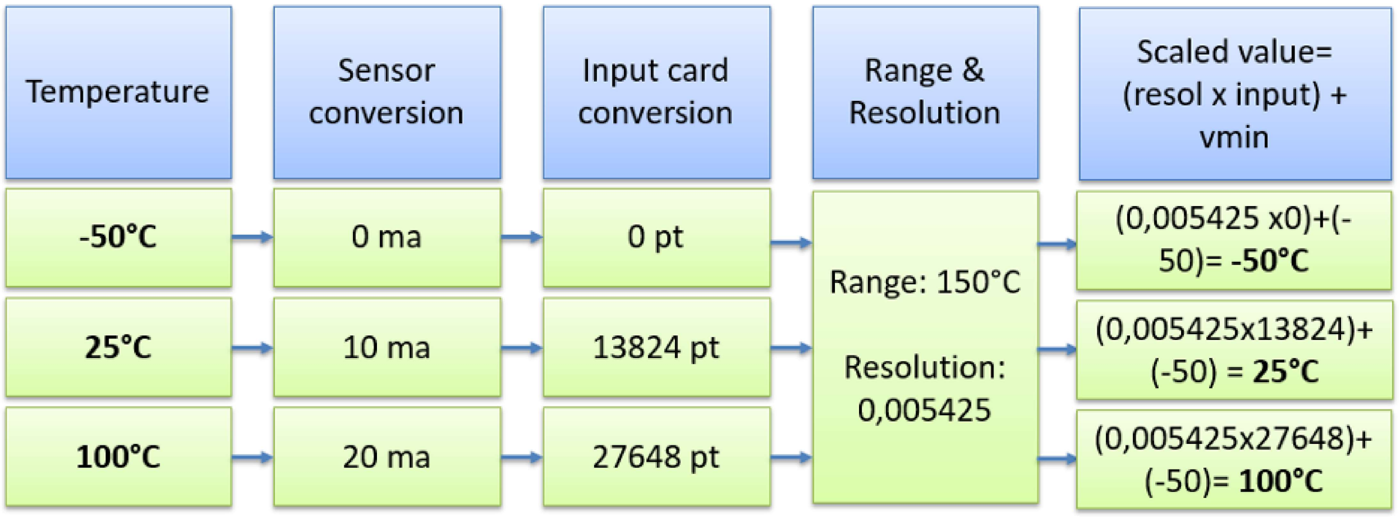


Figure 2.7 - Temperature value scaling procedure example

7. PLC program for analogue signal scaling

There could be several possible conversion methods in a PLC program. We suggest three different ways considering **different languages** and the use of **built-in functions** instead of elementary **math operations.**

In each solution we take the same system into account:

Measurement of temperature with a 0-20mA sensor, reading range between -50°C and 100°C

Temperature sensor connected to analogue input IW64 "Temperature_Sensor" (Int type), 27648 points

Scaled data stored in the PLC Data Block DB1 (Temp_Data, variable "ACT_Temperature" (Real type). The DB has to be configured as "non optimised" in their properties to allow communication with the Raspberry Pi and Node-RED

Other variables, used to store any intermediate results, are defined as Temp variables in the program block

8. SCALING USING MATH OPERATOR IN LADDER LANGUAGE

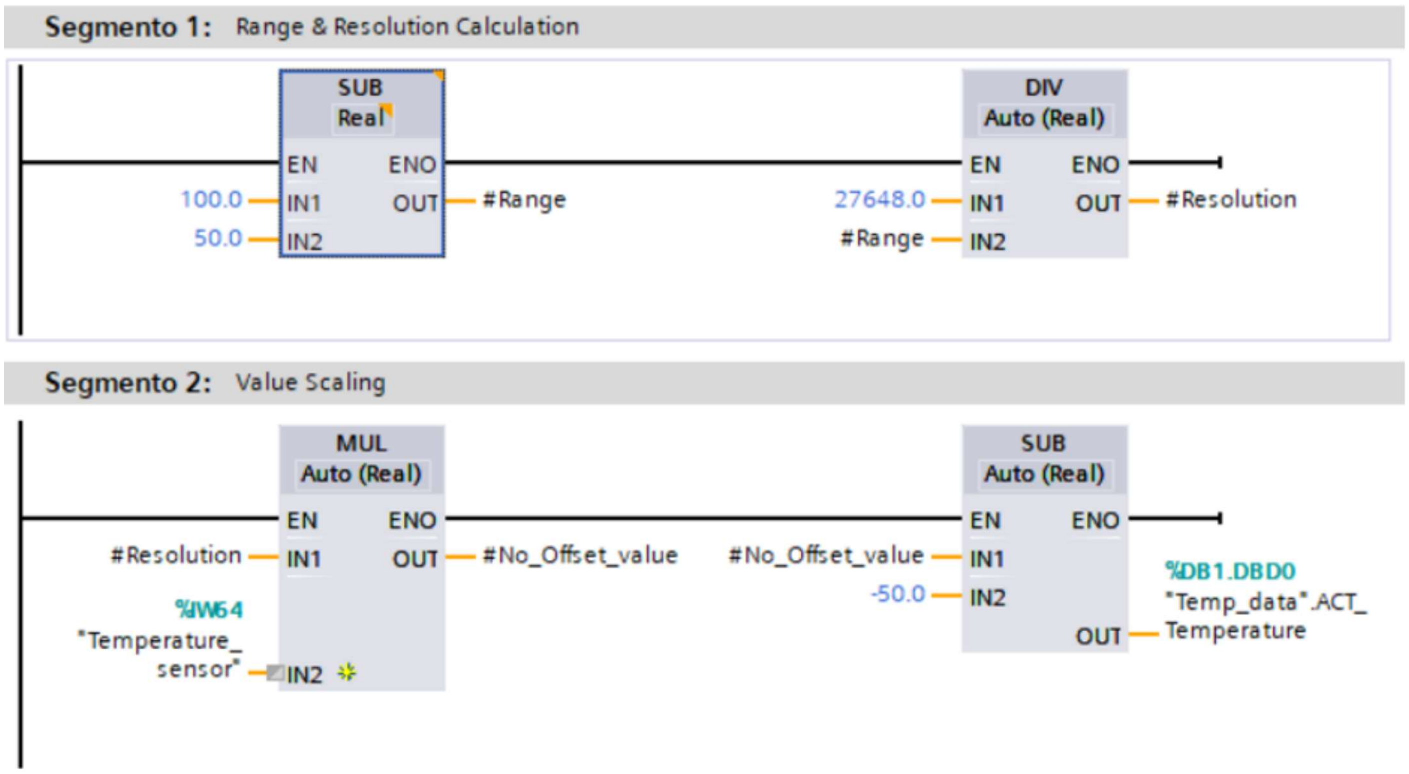
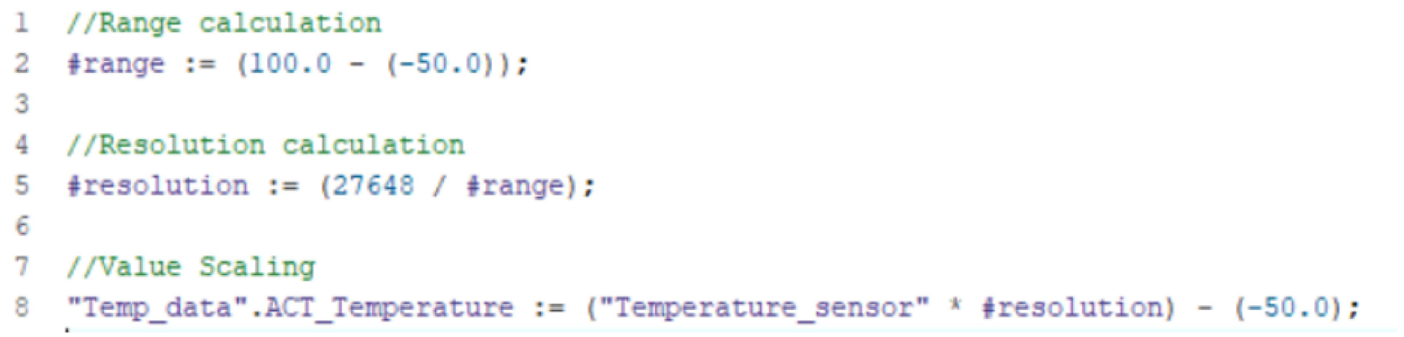


Figure 2.8 - Scaling using math operator in LADDER

9. SCALING USING MATH OPERATOR IN SCL LANGUAGE



```
//Range calculation
#range := (100.0 - (-50.0));

//Resolution calculation
#resolution := (27648 / #range);

//Value Scaling
"Temp_data".ACT_Temperature := ("Temperature_sensor" * #resolution) - (-50.0);
```

Figure 2.9 - Scaling using math operator in SCL

10. SCALING USING BUILT-IN FUNCTIONS "SCALE_X" AND "NORM_X"

This solution uses two different Siemens Built-in functions:

NORM_X: converts a value between a minimum and a maximum to a floating point value between 0.0 and 1.0. If the value entered corresponds to the minimum value, the output will be 0.0. If the value entered corresponds to the maximum value, the output will be 1.0. This value could be interpreted as the percentage value of the input referred to its minimum and maximum, 0 and 27648

SCALE_X: converts a floating-point value between 0.0 and 1.0 to a value within the range stated between minimum value and maximum value. If the value entered corresponds to 0.0, the output will be the minimum value. If the value entered corresponds to 1.0, the output will be the maximum value

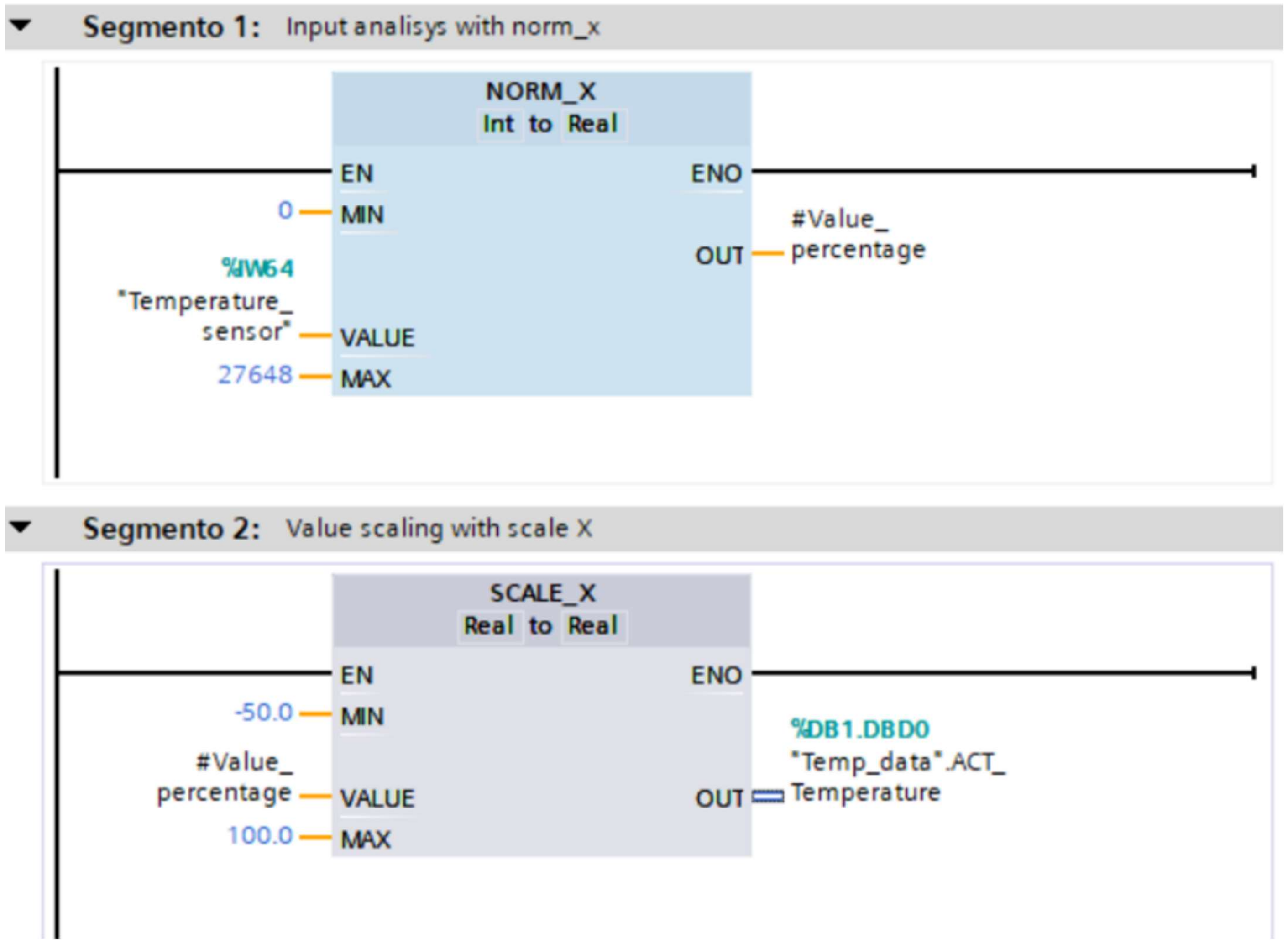


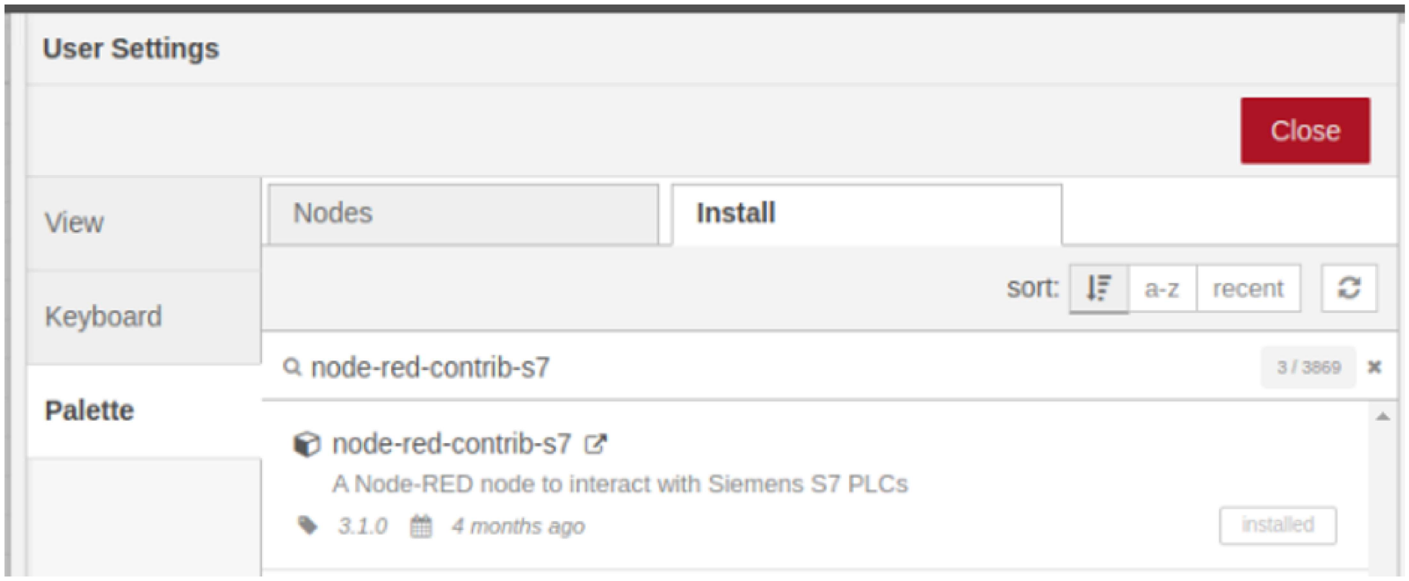
Figure 2.10 - Scale_x e Norm_x

11. Read PLC variables from Node-RED

Node-RED configuration

Once Node-RED is launched on your Raspberry Pi, you need to add a **dedicated node type** in order to manage the communication between PLC and Node-RED.

In the Node-RED palette, several optional nodes are available dedicated to **PLC communications** for a lot of PLC manufacturers. The one shown in this guide allows communication with Siemens S7 PLC and is called "**node-red-contrib-s7**".



node-red-contrib-s7

node-red-contrib-s7

A Node-RED node to interact with Siemens S7 PLCs

3.1.0 4 months ago

Figure 3.1 - Node to install for PLC communication

After "node-red-contrib-s7" installation, three new nodes will appear in the node menu. The complete reference for these nodes is available here: https://flows.nodered.org/node/node-red-contrib-s7

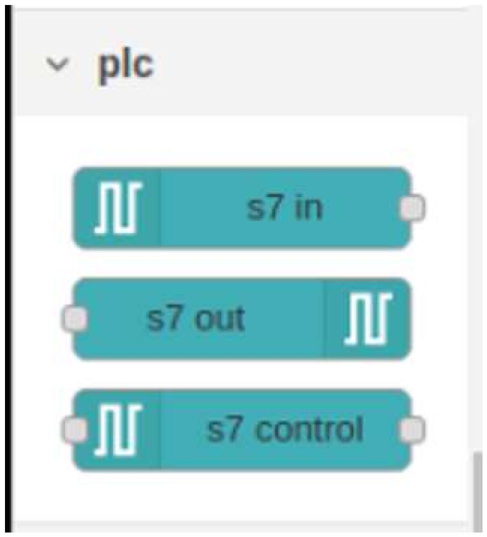


Figure 3.2 PLC communication nodes

12. Reading the value from PLC

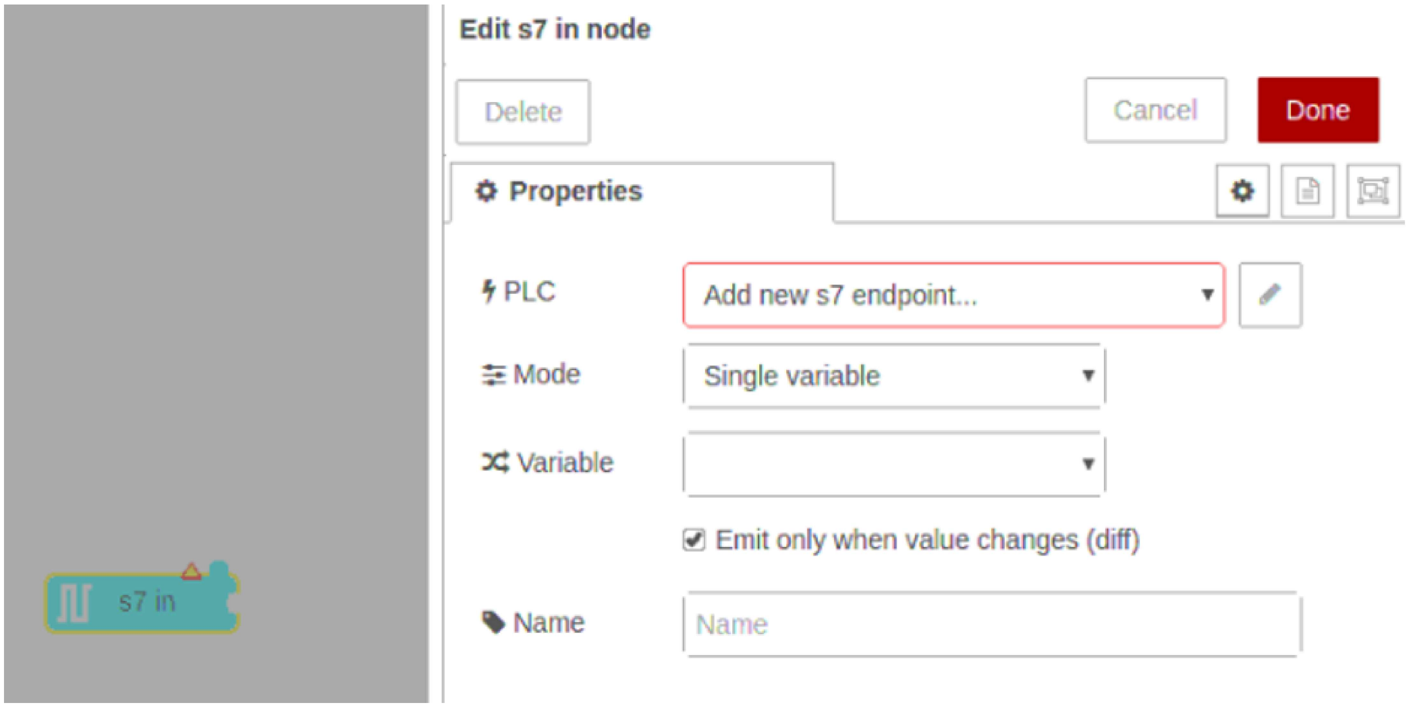
Before establishing communication, it is compulsory to verify these two conditions in the PLC Simatic S7:

"Optimized block access" must be disabled for the DBs we want to access (image)

In the "Protection" section of the CPU Properties, enable the **"Permit access with PUT/GET"**checkbox

In Node-RED you have to use the node "**S7 in**" in order to **read a PLC variable** and configure it following these steps:

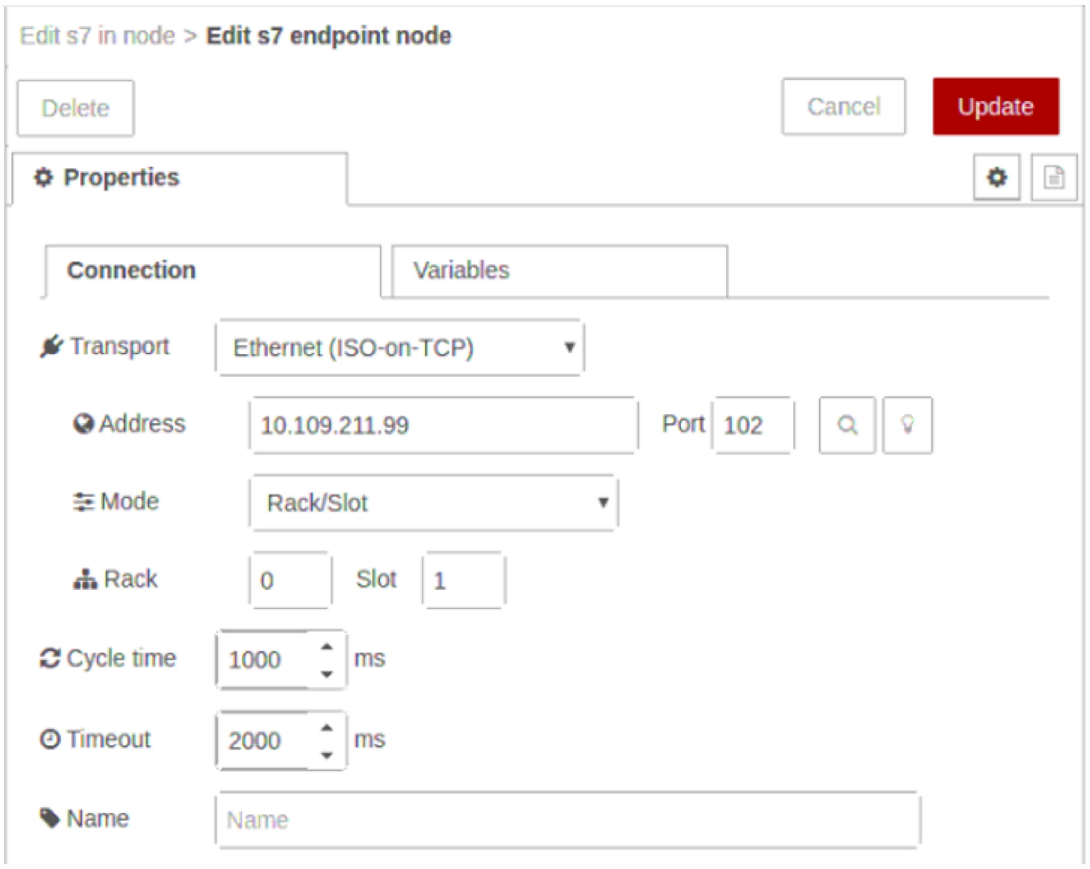
In the node properties, you have to define the **PLC connection** by clicking on "Add new S7 endpoint". Once the connection is defined, it remains available for use in other nodes.



PLC: Add new s7 endpoint...
Mode: Single variable
Variable:
☑ Emit only when value changes (diff)
Name: Name

Figure 3.3 - Adding S7 communication in node-RED

*The PLC is defined by its **communication type** (Ethernet or MPI), its IP Address, and the position of the CPU in the Simatic configuration (Rack and Slot)*


Edit s7 in node > **Edit s7 endpoint node**

Transport: Ethernet (ISO-on-TCP)

Address: 10.109.211.99 Port: 102

Mode: Rack/Slot

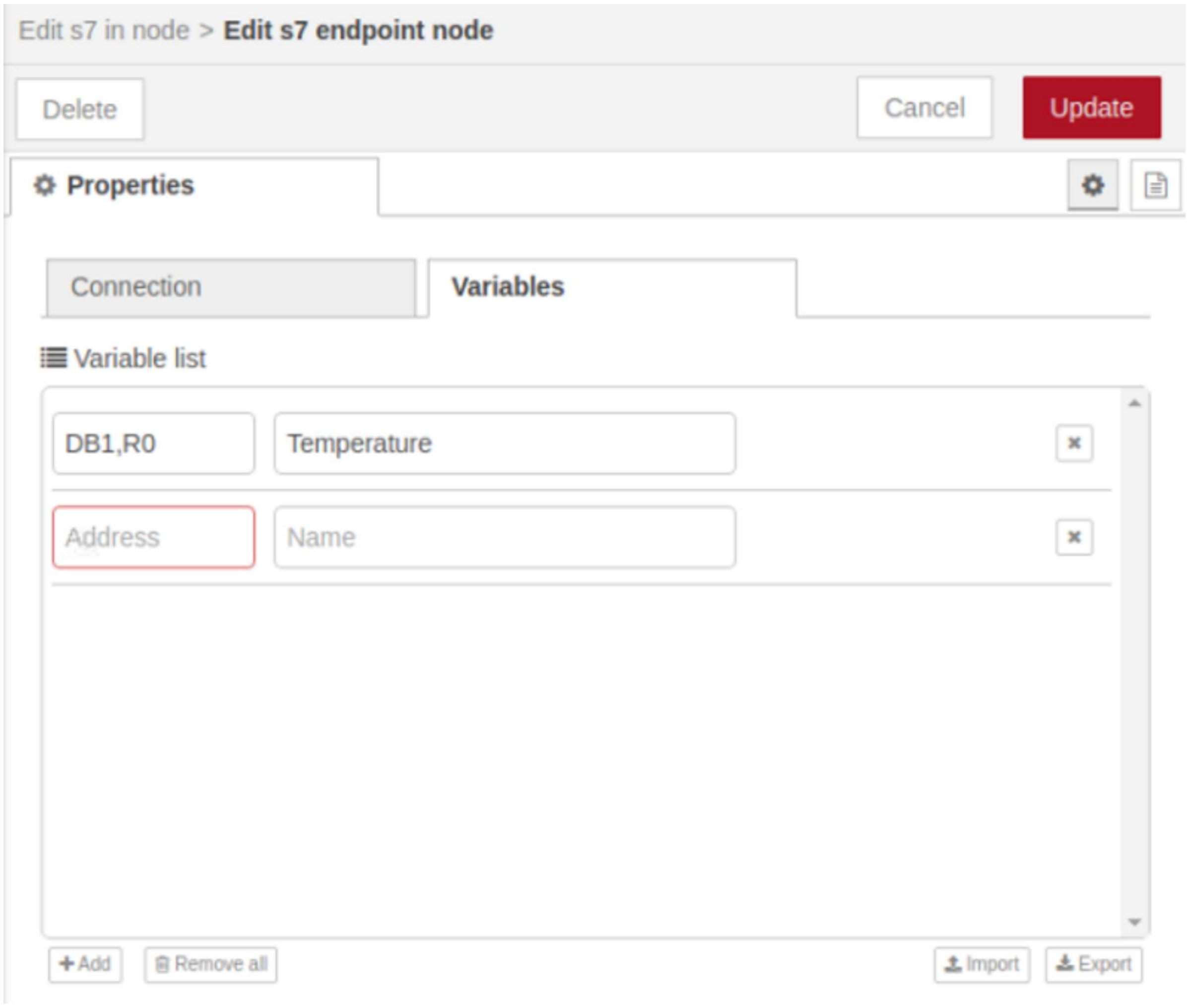
Rack: 0 Slot: 1

Cycle time: 1000 ms

Timeout: 2000 ms

Name: Name

Once the PLC connection has been established, you must ***add the variable*** *or variables in the node. The "address" field must contain the area/block you want to read (DB1 in this example) and the data (R0 means Real data in the 0 DB address, or rather the DBD0).*



Address	Name
DB1,R0	Temperature

Figure 3.5 - Variable definition in the node

Here are some examples that may help you address your variables: https://flows.nodered.org/node/node-red-contrib-s7

Node S7 is now configured and the PLC variable value is now available on the node exit point for all your purposes

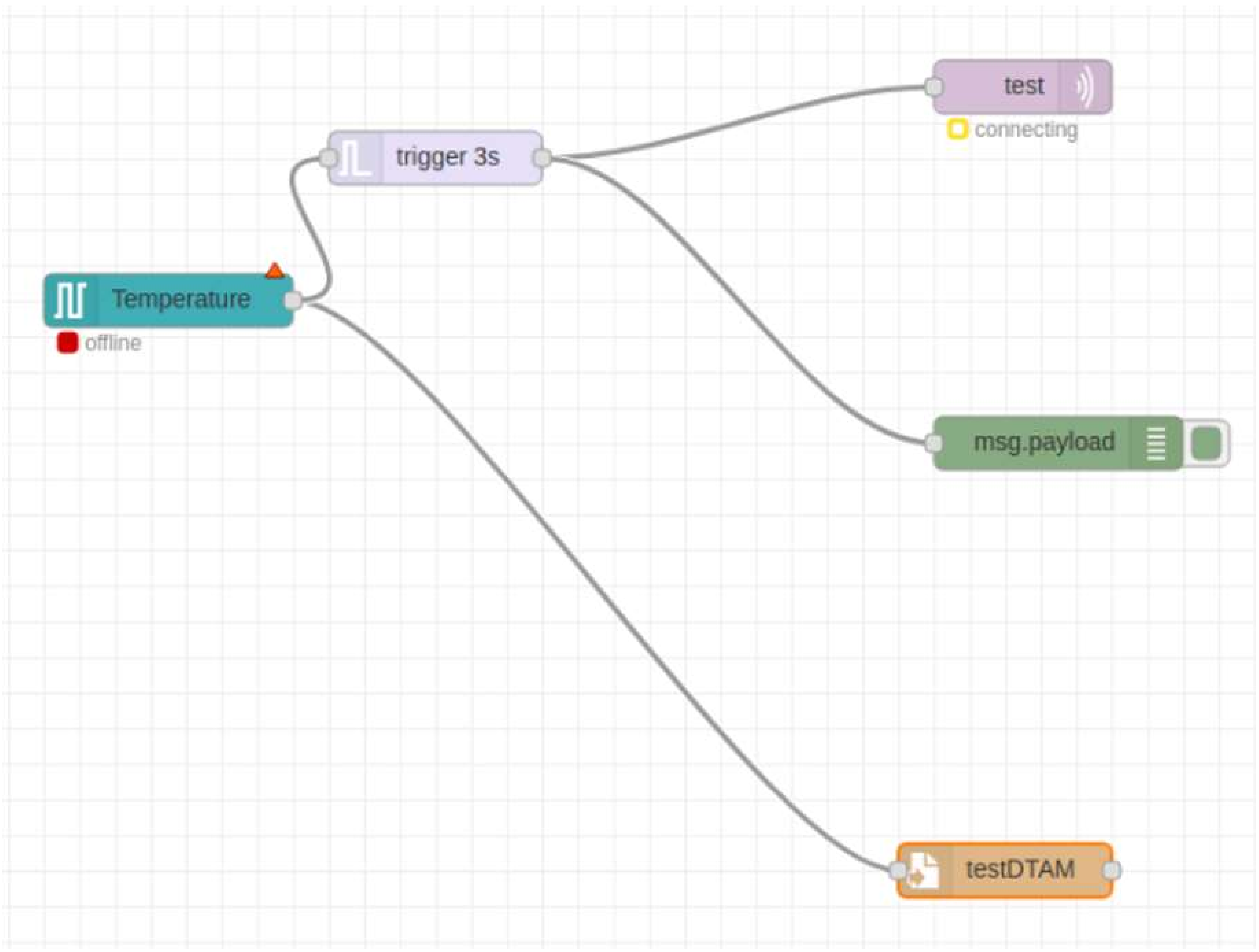


Figure 3.6 Use of S7_In functio

Sensors challenge

Site:	DTAM Online Training Platform	Printed by:	Ioanna Matouli
Course:	Advanced sensors	Date:	Friday, 8 December 2023, 4:40 PM
Book:	Sensors challenge		

Description

This challenge has to be done in a group of students and is used to proove your skills with the sensors module, as well as "soft skills" like working together.

Table of contents

1. The advanced sensorica challenge

2. Group challenge

3. Goals to reach with the challenge

4. Time necessary to complete the challenge

5. Assessment (rubrics)

1. The advanced sensorica challenge

A factory builds motors and generators. The quality control department is looking for a way to predict hardware failures before they actually happen.

The department wants to be able to inform customers and schedule a maintenance appointment before a motor breaks down.

Sensors should be created that record data to a central database, and a process should monitor the data and send warnings if a machine is about to break down.

Make sure the solution includes at least 4 of the following items:

1. A sensoring device that records motor temperature
2. A sensoring device that records motor vibrations (movements)
3. A sensoring device that records electrical current the motor is using
4. Sensor data is send to a database or message queue
5. A dashboard is available to look at the gathered data, and shows graphs for specific time periods (minute/hour/day/week/month/year)
6. A prediction algorithm is running and generates a warning when a motor is about to break down (measure data from normal operation and measure data from just after a breakdown and compare values)

The device that's being used to read the sensors is not mandatory, could be a raspberry, an arduino, a PLC or something else.

The protocol for sending measurements to storage is not mandatory, could be MQTT, a python script, a PHP script, C or anything else; students will give arguments for their choice.

See the evaluation rubric to see details.

2. Group challenge

To do this challenge, you will need to form a group with 2-4 other students.

You will be instructed by a teacher on how to perform the challenge.

When your group is finished, the product and the process will be evaluated by a teacher using the evaluation rubric (see below).

3. Goals to reach with the challenge

- Choose appropriate sensors for a given problem, and build sensoring devices (raspberry, arduino)
- Choose a data collection service (database or IOT hub or message queue)
- Have sensoring device report measurements to database and messaging bus
- Have dashboard subscribe to message bus and display trends over time
- Have students decide on the best communication protocol between devices, given the problem domain (devices with low power requirements / devices to be offline for extended periods of time / devices in environments with lots of data corruption, like factories)
- Program sensoring devices using Python for measurement correction / quality, and to provide different aggregation levels (distinct measurements, as well as averages per minute, hour, day, week, month and year)

4. Time necessary to complete the challenge

The challenge could be completed in 40 hours. That could be 5 full working days (1 full week), or 10 week x 4 hours, or another schedule.

5. Assessment (rubrics)

...