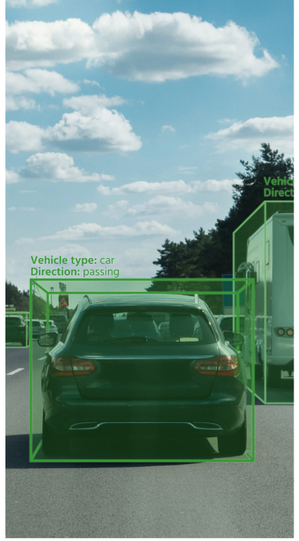
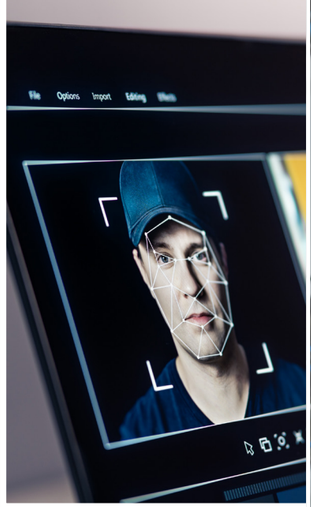
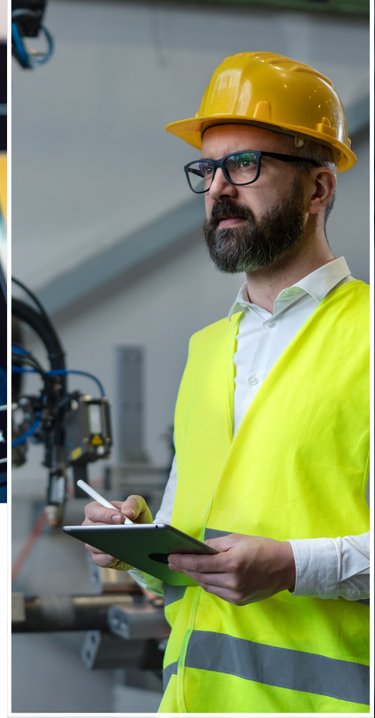
# We get technical



DigiKey





# contents



## Editor’s note

### Welcome to the DigiKey eMagazine Volume 18 – Edge AI.

As technology continues to evolve at a rapid pace, we’re constantly exploring innovative solutions that shape the future of industries from IoT to AI and machine vision. In this issue, we’ve curated a selection of articles that offer valuable insights into these game-changing technologies and how they’re transforming the way we approach engineering challenges.

In our first feature, we delve into predictive maintenance through AI-powered data acquisition, highlighting how current sensors can play a pivotal role in optimizing efficiency and minimizing downtime. Staying on the topic of AI, we also explore tinyML at the Edge – examining three unique use cases that demonstrate how machine learning can be deployed directly within resource-constrained devices for smarter, more efficient systems.

For those venturing into the world of multicore microcontrollers, we break down why they’re essential for IoT devices at the Edge and provide practical advice on getting started with these powerful, parallel-processing units. We also take a deep dive into the crucial, yet often overlooked, aspect of data preparation in machine learning – offering clarity on why clean, structured data is the foundation of successful ML projects.

On the hardware front, we explore how to design and deploy smart machine vision systems rapidly, empowering you with the tools needed to integrate visual intelligence into applications across industries. And lastly, we turn our focus to GMSL cameras, which have been road-tested and are driving innovation into new markets, presenting opportunities that are redefining how we capture and process visual data.

This issue is packed with cutting-edge information and practical tips to keep you ahead of the curve in the world of technology. We hope these articles inspire fresh ideas and new possibilities as you navigate the exciting developments in your field.

# Use a current sensor to efficiently acquire data for predictive maintenance with AI

Written by Clive 'Max' Maxfield

The Internet of Things (IoT) has brought about tremendous interest in using artificial intelligence (AI) and machine learning (ML) technologies to monitor the health of machines including motors, generators, and pumps, and to alert maintenance engineers as to any looming problems. One difficulty for the designers of AI/ML systems looking to implement this type of predictive maintenance is selecting the best sensor for the application. Another issue is that relatively few designers have any experience creating AI/ML applications.

To obtain the data for the AI/ML system to act upon, designers often opt for sophisticated sensors like three-axis accelerometers coupled with high-powered microcontroller development platforms. In many cases, however, it's possible to achieve the desired goal using a simple current sensor in conjunction with a more modest and less costly microcontroller development platform.

This article introduces the idea of using a current sense transformer to obtain the data required to simply and cost-effectively implement AI/ML applications. Using a low-cost Arduino IoT microcontroller development platform and a current sense transformer from CR Magnetics, the article also presents a simple circuit that employs the current sensor to monitor the health of a vacuum pump with an integrated filter, alerting the user when the filter has become clogged. Finally, the article presents an overview of the process of creating the associated AI/ML application.

## Simple sensors for AI/ML

In order to acquire the data for an AI/ML application to act upon, designers often opt for sophisticated sensors like three-axis accelerometers; but this type of sensor can generate vast amounts of data that are difficult to manipulate and understand. To avoid this complexity, it's worth remembering that everything is interrelated. Just as an injury to one part of a person's body can cause referred pain that is perceived elsewhere in the body, a failing bearing in a motor can modify the current being used to drive that motor. Similarly, in addition to causing overheating, a blocked air intake can also modify the current being used to drive the motor.

Consequently, monitoring one aspect of a machine's operation may cast light on other facets of its workings. As a result, it's possible to achieve the desired monitoring and sensing goal by observing a related parameter using a substantially simpler sensor, such as the low-cost, small-size, CR3111-3000 split-core current sense transformer from CR Magnetics (Figure 1).

The CR3111-3000 can be used to detect current up to 100 amperes (A) (other members of the CR31xx family can be employed for lessor or greater current values). All members of the family support a frequency range of 20 hertz (Hz) to 1 kilohertz (kHz), covering the majority of industrial applications. Also, all CR31xx devices employ a hinge and locking snap that allows them to be attached without interrupting the current carrying wire.

## The Arduino Nano 33 IoT

One example of a low-cost microcontroller development platform suitable for prototyping simple AI/ML applications is the ABX00032 Arduino Nano 33 IoT from Arduino (Figure 2). Featuring an Arm Cortex-M0+ 32-bit ATSAMD21G18A processor running at 48 megahertz (MHz) with 256 kilobytes (Kbytes) of flash memory and 32 Kbytes of SRAM, the Arduino Nano 33 IoT also comes equipped with both Wi-Fi and Bluetooth connectivity.

*Figure 1: The CR3111-3000 split-core current sense transformer provides a low-cost, easy-to-use current detector that can be employed as the primary sensor in an AI/ML predictive maintenance application. Image source: CR Magnetics*

## Data capture circuit

The circuit used for the purpose of this discussion is shown below in Figure 3. The CR3111-3000 transforms the measured current driving the machine into a much smaller one using a 1000:1 ratio.

Resistor R3, which is connected across the CR3111-3000's secondary (output) coil, acts as a burden resistor, producing an output voltage proportional to the resistor value, based on the amount of current flowing through it.

Resistors R1 and R2 act as a voltage divider, forming a 'virtual ground' with a value of 1.65 volts. This allows the values from the CR111-3000 to swing positive and negative and still not hit a rail, since the microcontroller cannot accept negative voltages. Capacitor C1 forms part of an RC noise filter that reduces noise from the 3.3 volt supply and nearby stray fields from getting into the measurements, thereby helping the voltage divider act as a better ground.

*Figure 2: The Arduino ABX00032 Nano 33 IoT provides a low-cost platform upon which to build AI/ML applications to enhance existing devices (and create new ones) to be part of the IoT. Image source: Arduino*

A vacuum pump with an integrated filter was used to provide a demonstration test bench. For the purposes of this prototype, Tripp Lite's P006-001 1 foot (ft.) extension power cord was inserted between the power supply and the vacuum pump (Figure 4).

The prototype circuit was

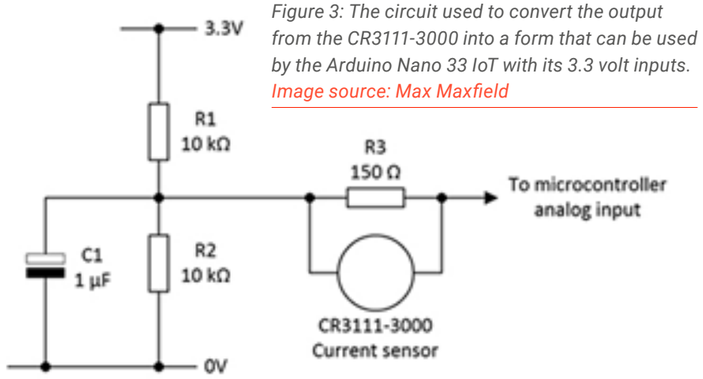



*Figure 3: The circuit used to convert the output from the CR3111-3000 into a form that can be used by the Arduino Nano 33 IoT with its 3.3 volt inputs. Image source: Max Maxfield*

implemented using components from the author's treasure chest of spare parts (Figure 5). Readily available equivalents would be as follows:

- Adafruit 64 breadboard
- Twin Industries TW-E012-000 pre-formed wire kit for use with breadboards
- Stackpole Electronics RNMF14FTC150R 150 ohm (Ω) ±1% 0.25 watt (W) through-hole resistor
- Stackpole Electronics' RNF14FTD10K0 10 kiloohm (kΩ) ±1% 0.25 W through-hole resistor
- KEMET ESK106M063AC3FA 10 microfarad (µF) 63 volt aluminum electrolytic capacitor

With regard to the leads from the current sensor, 1931 22-28 AWG crimp pins from Pololu Corp. were crimped on the ends. These pins were subsequently inserted into a 1904 5 x 1 black rectangular housing with a 0.1 inch (in.) (2.54 millimeter (mm)) pitch, also from Pololu.

## Creating the AI/ML application

In order to create the AI/ML application, a free trial version of NanoEdge AI Studio was accessed from Cartesium's website (see also, 'Easily Bring Artificial Intelligence to Any Industrial System').

When NanoEdge AI Studio is launched, the user is invited to create and name a new project. The user is then queried as to the

*Figure 4: The 1-foot extension power cord that was modified to accept the current sensor. Image source: Max Maxfield*

processor being used (an Arm Cortex-M0+ in the case of the Arduino Nano 33 IoT development board), the type(s) of sensor being used (a current sensor in this case), and the maximum amount of memory that is to be devoted to this AI/ML model (6 Kbytes was selected for this demonstration).

In order to create the AI/ML model, it is first necessary to capture representative samples of good and bad data (Figure 6). A simple Arduino sketch (program) was created to read values from the current sensor. This data can be directly loaded into NanoEdge AI Studio 'on-the-fly' from the microcontroller's USB port. Alternatively, the data can be captured into a text file, edited (to remove spurious samples at the beginning and end of the run), and then loaded into NanoEdge AI Studio.

The good data was collected with the vacuum pump running in its normal mode. In order to gather the bad data, the pump's air filter was obstructed with a disk of paper.

Using the good and bad data, NanoEdge AI Studio generates the best AI/ML library solution out of 500 million possible combinations. Its ongoing progress is displayed in a variety of different ways, including a scatter chart showing how well the normal signals (blue) are being distinguished from the abnormal signals (red) with regard to a threshold value, which was set to 90% in this example (Figure 7).

The early models typically find it difficult to distinguish between the normal and abnormal data,

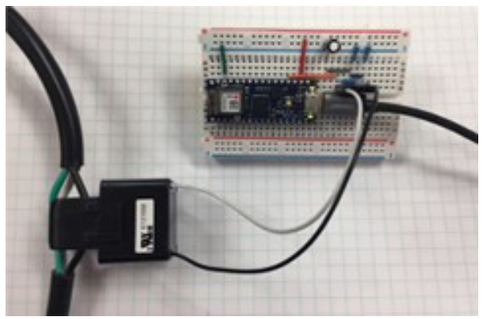


*Figure 5: The prototype circuit was implemented using a small breadboard and components from the author's treasure chest of spare parts. Image source: Max Maxfield*

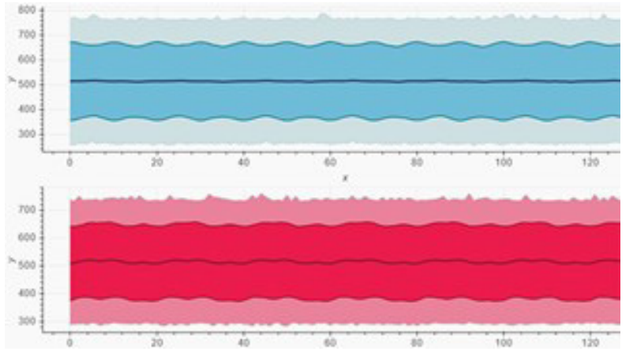


*Figure 6: Comparison of good/normal data (top) and bad/abnormal data (bottom). Apart from the differences in color, these don't seem terribly different to the human eye, but an appropriate AI/ML model can distinguish between them. Image source: Max Maxfield*

but the system evaluates different combinations of algorithmic elements, iterating on increasingly accurate solutions. In this case, the process was halted after 58,252 libraries had been evaluated. The resulting library (model) was only 2 Kbytes in size.

It's important to note that, at this stage, the model is in its untrained form. Many different factors may affect the ways in which the machines run. For example, two seemingly identical vacuum pumps could be mounted in different locations – for example, one on a concrete slab and the other on a suspended floor. Or one of the machines could be located in a hot, humid environment, while the other may be in a cold, dry setting. Furthermore, one could be connected to long lengths of metal pipe, while the other could be attached to short lengths of plastic pipe.

Thus, the next step is to incorporate the library into the applications running on the microcontrollers and sensors that are attached to machines that are deployed in the real world. The AI/ML models on the different machines will then train themselves using good data from these real-world installations. Following this self-training period, the AI/ML models can be left to monitor the health of the machines, looking for anomalies and trends, and reporting their findings and predictions to human supervisors.

## Conclusion

Predictive maintenance using AI/ML allows engineers to address problems before failures actually occur. However, the hardware used to implement the predictive maintenance system needs to be as simple and cost-effective as possible; also, designers need ready access to the required software to perform the analysis.

As shown, instead of opting for a complex multi-axis accelerometer and associated hardware, a simple, low-cost, small-size, CR3111-3000 split-core current transformer connected to a low-cost microcontroller platform can perform the required sensing and data gathering. Coupled with advances in AI/ML tools and algorithms, it's now possible for non-AI/ML experts to create sophisticated AI/ML models that can be deployed in a wide range of simple and complex sensing applications.

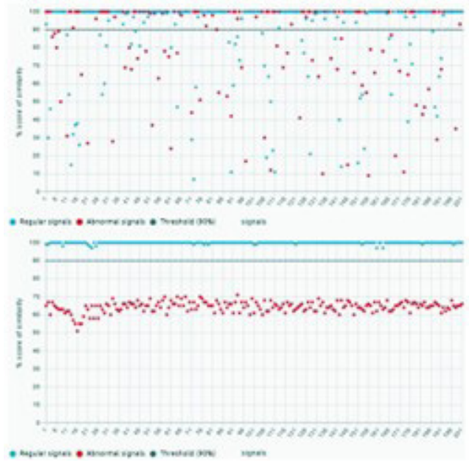


*Figure 7: NanoEdge AI Studio evaluates up to 500 million different AI/ML models to determine the optimal configuration for the normal and abnormal data. The initial models are rarely successful (top), but the tool automatically iterates on better and better solutions until the developer decides to call a halt (bottom). Image source: Max Maxfield*

# 3 uses for tinyML at the Edge

Written by Jacob Beningo

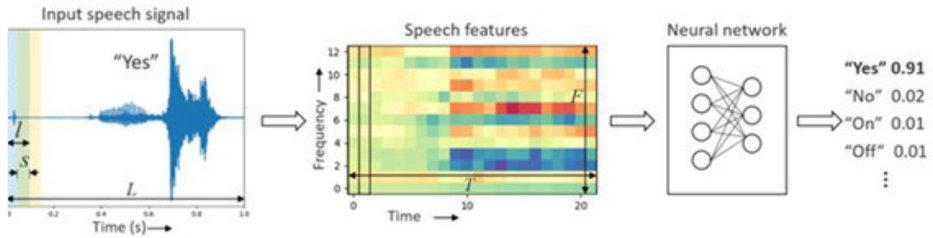



*Figure 1: An input speech signal is digitally processed to create a spectrograph used to train an NN to detect keywords. Image source: Arm*

Machine learning (ML) has found its way into many areas of the Cloud and has been finding its way to the Edge on relatively powerful processors running Linux. The problem with traditional ML running on these systems is that their power profiles are too large for them to 'disconnect' and perform work as battery-operated Edge devices. The trend, and the future of ML at the Edge, is to use tinyML. TinyML aims to bring ML algorithms to resource-constrained devices, such as microcontrollers based on Arm Cortex-M processors.

In this blog, we will explore the most popular use cases for leveraging tinyML on microcontroller-based devices for use at the Edge.

## Use case #1: keyword spotting

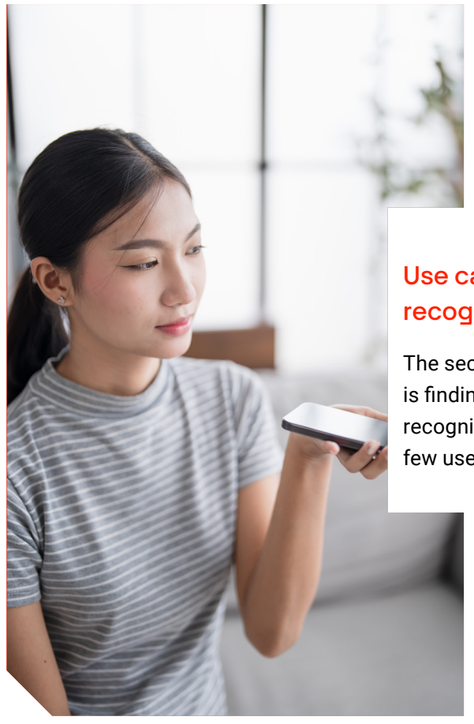
The first use case that tinyML is becoming popular for is keyword spotting. Keyword spotting is the ability of a device to recognize a keyword like 'Hey Siri', 'Alexa', 'Hello', and so forth. Keyword spotting has many uses for edge devices. For example, one might want to use a low-power processor to watch for a keyword that will wake up a more powerful one. Another use case might be to control an embedded system or a robot. I've seen examples where a microcontroller was used to decode keywords like 'forward', 'backward', 'stop', 'right', and 'left' to control a robot's movement.

Keyword spotting with tinyML

I've seen examples where a microcontroller was used to decode keywords like 'forward', 'backward', 'stop', 'right', and 'left' to control a robot's movement.

is typically done by using a microphone to capture an input speech signal. The speech signal is recorded as a voltage over time and then converted into a spectrograph using digital signal processing. The spectrograph is a time series that is plotted against the frequency of the input signal. The spectrograph can be fed into a neural network (NN) to train the tinyML algorithm to recognize specific words. The process is shown in Figure 1.

A typical implementation would feed fixed windows of speech into the NN. The network would then evaluate the probability of one of the desired keywords having been spoken. For example, if someone said, 'Yes', the NN may report that it was 91% sure it was 'Yes', with a 2% chance it's 'No', and a 1% chance it's 'On'.



The ability to use speech to control machines is a use case that many device manufacturers are carefully reviewing and hoping to enhance their devices within the coming years.

## Use case #2: image recognition

The second use case that tinyML is finding its way into is image recognition. There are quite a few use cases for Edge devices that can perform image recognition. One use case that you might already be familiar with is the ability to detect whether there is a person, package, or nothing at your door. There are certainly plenty of other applications that range from monitoring old analog meters, detecting lawn health, or even bird counting.

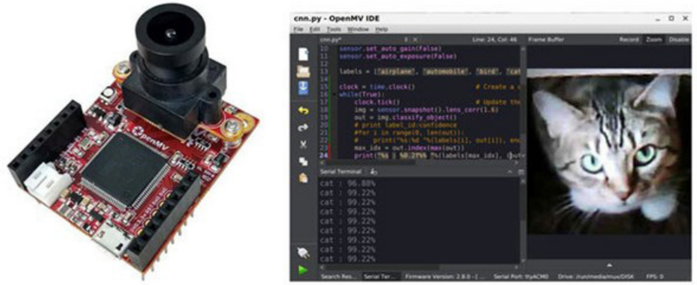



*Figure 2: The OpenMV camera module can be used for image recognition, and development can be done with a simple IDE using Python. Image source: Beningo Embedded Group*

Image recognition can seem like a complex field in which to get involved. However, there are several low-cost platforms available that can help developers get up and running. One of my favorites, and one that I use to get things done quickly, is the OpenMV.

OpenMV is an open machine vision platform that includes an integrated development environment (IDE), a library framework written in Python, and a camera module from Seeed Technology that helps developers create their machine vision applications (Figure 2).

The camera module is based on an STMicroelectronics STM32H7 Cortex-M7 processor. The hardware can be expanded through its onboard expansion headers. It can run off a battery and can even have the camera module swapped out. A good getting-started example that you may find interesting is how to use the CIFAR-10 dataset with the Arm CMSIS-NN library for image recognition. The example can be found on YouTube.

## Use case #3: predictive maintenance

The last use case that we will discuss for tinyML is predictive maintenance. Predictive maintenance uses tools such as statistical analysis and ML to predict equipment state based on:

- Abnormality detection
- Classification algorithms
- Predictive models

For example, a factory might have a series of motors, fans, and robotic equipment that are used to produce a product. A company would want to minimize downtime to maximize the number of products that it can produce. If the equipment has sensors that can be interpreted using ML and the other techniques mentioned above, they can detect when the equipment is close to failure. Such a setup might look something like that shown in Figure 3.

Connecting a smart sensor to a low-power microcontroller leveraging tinyML can result in a wide variety of useful applications. For example, HVAC units could be monitored, air filters checked, and irregular motor vibration could be detected, among many others. Preventive maintenance can become more organized, hopefully saving a company from costly reactive measures, ensuring a more optimized maintenance schedule.

## Conclusion

TinyML has so many potential applications and use cases at the Edge. We've explored what's popular now, but the use cases are nearly unlimited. TinyML can be used for gesture detection, guidance and control, and so much more. As Edge devices start to leverage the capabilities of tinyML, the question really becomes, what are you using tinyML for at the Edge?

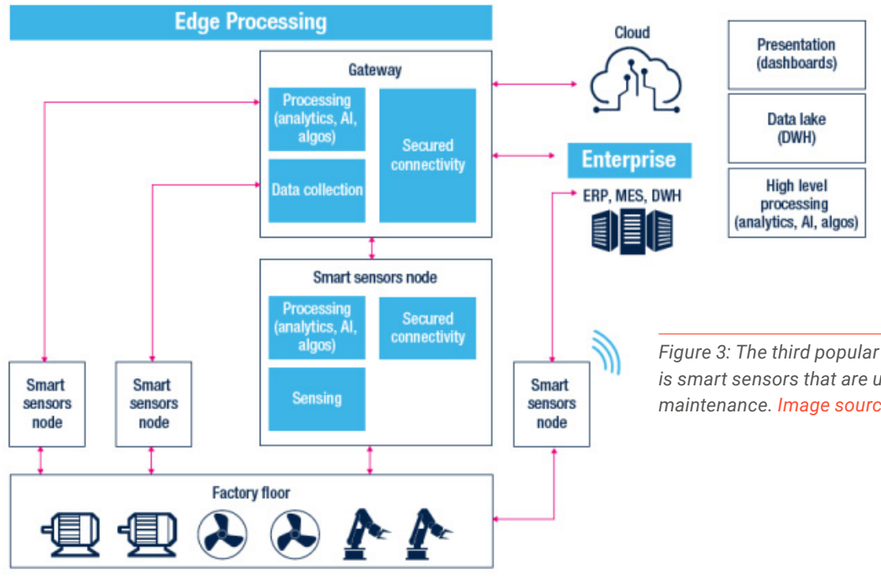



*Figure 3: The third popular use case for tinyML is smart sensors that are used for predictive maintenance. Image source: STMicroelectronics*

# Why and how to get started with multicore microcontrollers for IoT devices at the Edge

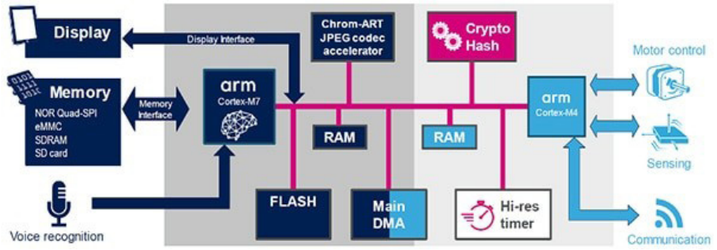
Written by Jacob Beningo

Developers of Internet of Things (IoT) devices at the Edge are being asked to incorporate an increasingly diverse and processing-intensive range of functions, from communications and sampling sensors to executing machine learning (ML) inferences. At the same time, developers are being asked to maintain or reduce power consumption. What's needed is a more flexible architectural approach to a core element of their design – the microcontroller – that will allow developers to add features while achieving the optimal balance of performance, functionality, and power consumption.

This architectural approach comes in the form of multicore microcontrollers. These have, as their name suggests, multiple processing cores built into a single package. However, just throwing more cores at the problem won't solve the issues. Developers need to understand the differences between symmetric and asymmetric multicore processors, how to approach functional partitioning, and how to program them effectively.

This article will introduce the concept of multicore microcontrollers before discussing how developers can leverage multicore microcontrollers to balance performance and

*Figure 1: One paradigm for application design with multicore microcontrollers is to place the feature rich application components in one core and the real-time components in the second core. Image source: STMicroelectronics*




energy constraints. Several multicore microcontrollers from STMicroelectronics' STM32H7 line will be introduced by way of example. The article will also examine several use cases where developers can leverage multicore processing and split the workload between multiple cores.

## Introduction to multicore microcontrollers

As mentioned, multicore microcontrollers have more than one processing core. There are two types of configurations which are often used, symmetric and asymmetric processing. Symmetric core configurations contain two or more of the exact same processing cores. For example, they might both be Arm Cortex-M4 processors. Asymmetric cores on the other hand may contain an Arm Cortex-M7 processor and an Arm Cortex-M4 processor. They could also contain an Arm Cortex-M4 and an Arm Cortex-M0+ processor. The combinations are many and depend upon application and design requirements.

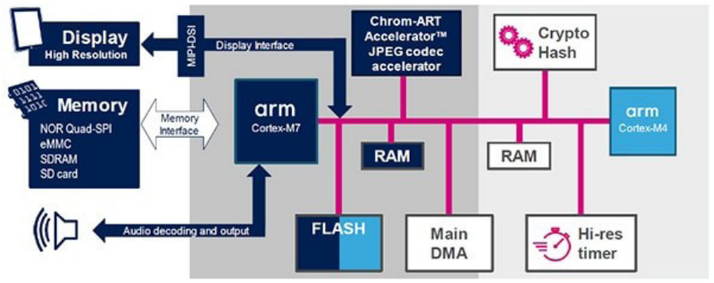



*Figure 2: Another paradigm for application design with multicore microcontrollers is to place the real-time application components in one core and all the security components in a second core. Image source: STMicroelectronics*

IoT developers are interested in multicore microcontrollers because they allow them to separate their application into multiple execution domains. Separate execution domains allow precise control of the application's performance, features, and power needs. For example, one core may be used to interact with a user through a high-resolution display and touch panel, while the second core is used manage the real-time requirements of the system such as controlling a motor, relays and sampling sensors.

There are many ways that a developer can partition their application, but the two biggest paradigms are to separate the application into:

- Feature rich/real-time
- Real-time/secure

In the first paradigm, feature rich/real-time, the system is exactly like the one described in the paragraph above. Feature rich application components, such as the display, ML inferences, audio playback, and memory storage, among others, are all handled by one core. The second core then handles real-time functions such as motor control, sensing, and communication stacks (Figure 1).

The second paradigm separates the application into real-time and secure functionality. In the first core, the application may handle things like the display, memory access, and real-time audio playback. The second core, on the other hand, may do nothing more than act as a security processor. As such, the second core would handle storage of critical data like device and network keys, handle encryption, secure bootloader, and any other features deemed to fall within the secure software category (Figure 2).

*Figure 3: The STM32H745I-DISCO board integrates a wide range of on-board sensors and memory capabilities that allow developers to test out the dual core microcontrollers running at 480 MHz and 240 MHz. Image source: STMicroelectronics*

There are other potential ways to parse up a multicore microcontrollers' application space, but these two paradigms seem to be the most popular among IoT developers.

## Selecting a multicore microcontroller development board

While multicore microcontrollers are becoming very popular, they are still not quite mainstream and selecting one can be tricky. For a developer looking to work with multicore microcontrollers, it's best to select a development board that has the following characteristics:

- Includes an LCD for feature rich application exploration
- Expansion I/O
- Is low cost
- Has a well proven ecosystem behind it including example code, community forums, and access to knowledgeable FAEs

Let's look at several examples from STMicroelectronics, starting with the STM32H745I-DISCO (Figure 3). This board is based on the STM32H745ZIT6 dual core microcontroller that comprises an Arm Cortex-M7 core running at 480 megahertz (MHz) and a second Arm Cortex-M4 processor running at 240 MHz. The part includes a double-precision floating point unit and an L1 cache with 16 kilobytes (Kbytes) of data and 16 Kbytes of instruction cache. The discovery board is particularly interesting because it includes additional capabilities such as:

- An SAI audio codec
- A microelectromechanical systems (MEMS) microphone
- On-board QUAD SPI flash
- 4 gigabyte (Gbyte) eMMC
- Daughterboard expansion
- Ethernet
- Headers for audio and headphones

The development board has a lot of built-in capabilities that make it extremely easy to start experimenting with multicore microcontrollers and really scale up an application.

For developers who are looking for a development board that has additional capabilities and far more expansion I/O, the STM32H757I-EVAL may be a better fit (Figure 4). The STM32H757I-EVAL includes additional capabilities over the evaluation board such as:

- 8 M x 32-bit SRAM
- 1 Gbit twin quad SPI NOR flash
- Embedded trace macrocell (ETM) for instruction tracing
- Potentiometer
- LEDs
- Buttons (tamper, joystick, wake-up)

These extra capabilities, especially the I/O expansion, can be extremely useful to developers looking to get started.

Having looked at several development boards, the next step is to outline some recommendations for getting started with a multicore microcontroller application.

*Figure 4: The STM32H757I-EVAL board provides developers with lots of expansion space, easy access to peripherals, and an LCD screen to get started with multicore applications. Image source: STMicroelectronics*

## How to start that first multicore application

No matter which of the two STM32H7 development boards is selected, there are two main tools that are needed to get started. The first is STMicroelectronics' STM32CubeIDE, a free integrated development environment (IDE) that lets developers compile their application code and deploy it to the development board. STM32CubeIDE also provides the resources necessary to step through and debug an application, and is available for major operating systems including Windows, Linux and MacOS.

The second tool is STMicroelectronics' STM32H7 firmware package. This includes examples for the STM32H7 development boards for:

- Multicore processing
- Using FreeRTOS
- Peripheral drivers
- FatFS (file system)

Developers will want to download the firmware application package and become familiar with the examples that are supported by the chosen development board.

**For developers of IoT systems at the network Edge, multicore microcontrollers provide the ability to better match and balance functionality, performance, and power consumption.**

Specifically, there are two folders that developers will want to pay attention to. The first is the applications folder which has two examples that show how to use OpenAMP (Figure 5). These examples show how to transmit data back and forth between the microcontroller cores where one core sends data to the other core, which then retransmits it back to the first core. Both examples perform this in a different way. One is baremetal, without an operating system, while the other is with FreeRTOS.

The second set of examples demonstrates how to configure both cores with and without an RTOS (Figure 6). One example shows how to run FreeRTOS on each core, while the other shows how to use an RTOS on one core and run the second core baremetal. There are several other examples throughout the firmware package that demonstrate other capabilities, but these are good choices to get started.

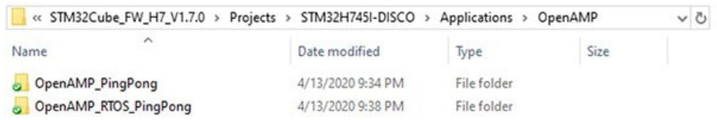



*Figure 5: The STM32Cube_FW_H7 provides several examples that demonstrate how to get started with multicore processing using OpenAMP. Image source: Beningo Embedded Group*

Loading an example project will result in a developer seeing a project layout similar to that shown in Figure 7. As illustrated, the project is broken up into application code for each core. The build configuration can also be setup such that a developer is working with only one core at a time. This can be seen in Figure 7, through the grayed-out files.

A full description of the example code is beyond the scope of this article, but the reader can examine the readme.txt file that is associated with any of the examples to get a detailed description of how it works, and then examine the source code to see how the inter-processor communication (IPC) is actually performed.

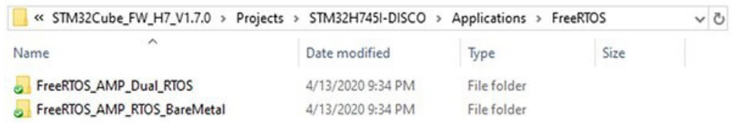



*Figure 6: The STM32Cube_FW_H7 provides several examples that demonstrate how to configure an operating system with multicore processors. Image source: Beningo Embedded Group*

## Tips and tricks for working with multicore microcontrollers

Getting started with multicore microcontrollers is not difficult, but it does require that developers start to think about their application's design a bit differently. Here are a few 'tips and tricks' for getting started with multicore microcontrollers:

- Carefully evaluate the application to determine which application domain separation makes the most sense. It is possible to mix domains on a single processor, but performance can be affected if not done carefully
- Take the time to explore the capabilities that are built into the OpenAMP framework and how those capabilities can be leveraged by the application
- Download the application examples for the STM32H7 processors and run the multicore application examples for the selected development board. The H747 includes two: one for FreeRTOS and one for OpenAMP
- When debugging an application, don't forget that there are now two cores running! Make sure to select the correct thread within the debug environment to examine its call history
- Leverage internal hardware resources, such as a hardware semaphore, to synchronize application execution on the cores
- Developers that start with a well-supported development board and then follow these 'tips and tricks' will find that they save quite a bit of time and grief when working with multicore microcontrollers for the first time.

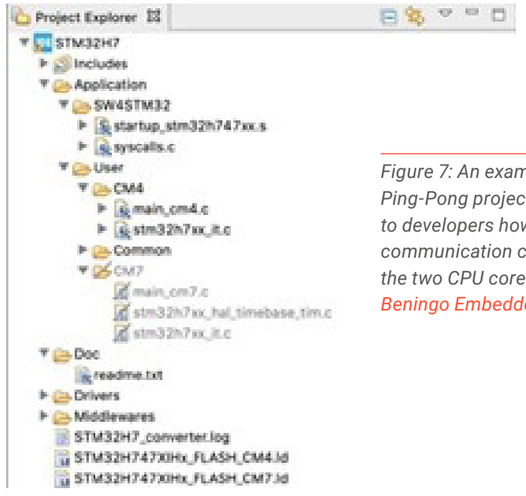



*Figure 7: An example OpenAMP Ping-Pong project demonstrates to developers how to create a communication channel between the two CPU cores. Image source: Beningo Embedded Group*

## Conclusion

For developers of IoT systems at the network Edge, multicore microcontrollers provide the ability to better match and balance functionality, performance, and power consumption per the application's requirements. Such microcontrollers allow a developer to partition their application into domains such as feature rich/real-time or real-time/secure processing. This ability to separate an application into different domains allows a developer to disable a core to conserve energy when the processing domain is no longer needed or turn it on in order to enhance application performance.

As shown, there are several different development boards that can be used to start exploring multicore microcontroller application design and take full control over its performance and energy profile.

Shawn Hymel Presents


DigiKey

# How to build an AI-powered toaster

Written by Shawn Hymel.


We can treat the toasting process like a predictive maintenance problem: how do we stop the toasting before the bread in question becomes irrevocably damaged (i.e. burnt)? We'll use a variety of gas sensors and machine learning to accomplish this task.

## Required hardware

You will need the following components:

- Wio Terminal
- Grove Multichannel Gas Sensor v2
- Grove SPG VOC and eCO2 gas sensor
- Grove BME680 temperature, pressure, and humidity sensor
- Grove I2C Hub (6 port)
- Grove cable (100 cm)
- Ammonia gas sensor
- Pololu Carrier for MQ Gas Sensors
- Fan (40 mm, 5V)
- 2x 10kΩ resistors
- N-channel MOSFET
- Mounting plate (e.g. a small piece of aluminum)
- Various wire, screws, nuts, standoffs

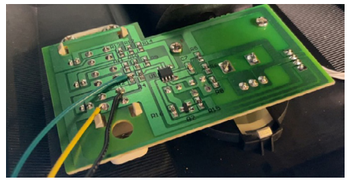
## Hardware connections

First, we need to hack the toaster. Open the toaster and find the circuit board that controls the toasting process. Use a multimeter to identify the following 3 nodes:

- Ground (GND)
- Node that becomes 3.3 or 5 V during the toasting process (for example, an LED that turns on when you press the lever down)
- Node that connects to GND when the 'cancel' button is pressed

Tack-solder 3 wires to each of these



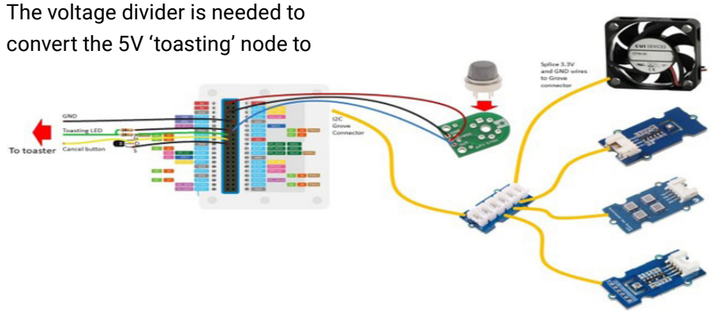


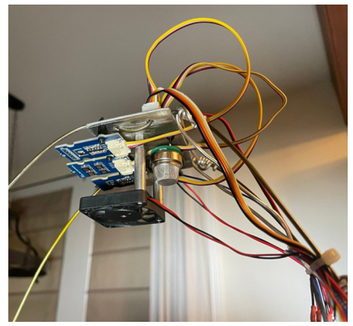
nodes. In the photo below, the black wire goes to ground, the green wire goes to the high side of the limiting resistor for the 'toasting' LED (i.e. 5V during 'toasting' and 0V otherwise), and the yellow wire goes to the 'cancel' button node opposite GND.

Attach all of the sensors and fan to the mounting plate. You'll want to position the fan to steadily blow air over the sensors. Use the I2C hub to connect all of the sensors together, and use the long Grove cable to connect the hub to the Wio Terminal. You'll also want long wires to run from the Wio Terminal to the ammonia sensor (as it is an analog sensor, not I2C).

You need the MOSFET in open-drain configuration to successfully control the 'cancel' button. The Wio Terminal might support open-drain GPIO, but I was too lazy to dig through the SAMD51 datasheet to figure out how to do this in code. The voltage divider is needed to convert the 5V 'toasting' node to 2.5 V (still considered logic HIGH for 3.3V pins).

IMPORTANT: the SAMD51 GPIO pins are NOT 5V tolerant! Make sure you use a divider, diode, etc. to drop the voltage if you're trying to sense something from 5V logic.



Screw/bolt everything to the aluminum plate (or some other mounting device).

## Mechanical build

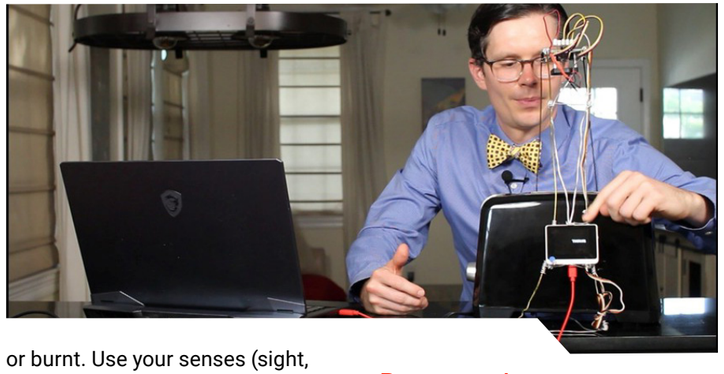
Construct a cage or arm that suspends the collection of sensors above the toaster. The microcontroller (Wio Terminal) should not be placed with the sensors to avoid letting it get too hot.

## Data collection

The model I created worked in my environment. It may or may not work for you, which means you'll likely need to collect data in your environment. Head to github.com/ShawnHymel/perfect-toast-machine to view all of the code for this project. Upload toast-odor-data-collection to the Wio Terminal. Read the comments in the code to determine which libraries you need to install prior to running the code.

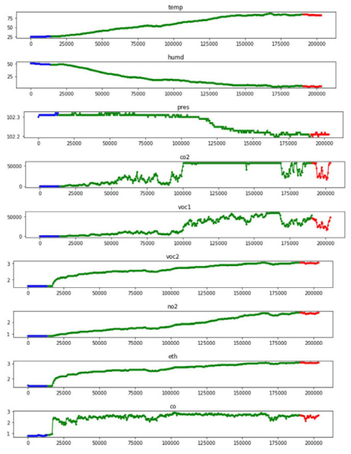
Make sure the Wio Terminal is plugged into a computer for the data collection process. I recommend waiting 15-30 minutes to let the gas sensors warm up. Use Python (v3+) to run serial-data-collect-csv.py to have it listen for serial data from the Wio Terminal. This will log each toasting instance to a CSV file on your computer. See this readme to learn how to use serial-data-collect-csv.py.

Start the toasting process with a piece of bread. Press button C (on the top of the Wio Terminal) to tag the data in one of three states: background (not toasting), toasting,



or burnt. Use your senses (sight, smell) to determine when you think the toast is 'burnt'.

See the CSV files in the datasets/ directory to see how I labeled the raw data. You're welcome to use that data as a starting point (I just can't promise that it will give you a model that works in your environment). Each folder is the brand, and a description of the bread used. The prefix of each CSV file is where I pulled the bread from (e.g. room temperature, refrigerator, freezer).




## Data curation

Run this notebook in Google Colab, following all of the directions: github.com/ShawnHymel/perfect-toast-machine/blob/main/ptm_dataset_curation.ipynb

Note that the script will automatically download the dataset from the GitHub directory. You can skip those cells and manually upload your data to the dataset/ folder in Colab if you wish to use your own data.

At the end of Step 2, you can view plots of your captured raw gas samples (one at a time). Blue is 'background', green is 'toasting', and red is 'burnt'.

At the end of Step 3, you should see means, standard deviations, minimums, and ranges printed out. Copy those down – you'll need the means and standard deviations for your inference code.

Step 4 will produce an out.zip file containing your curated dataset. Download this file and unzip it.

## Machine learning model training

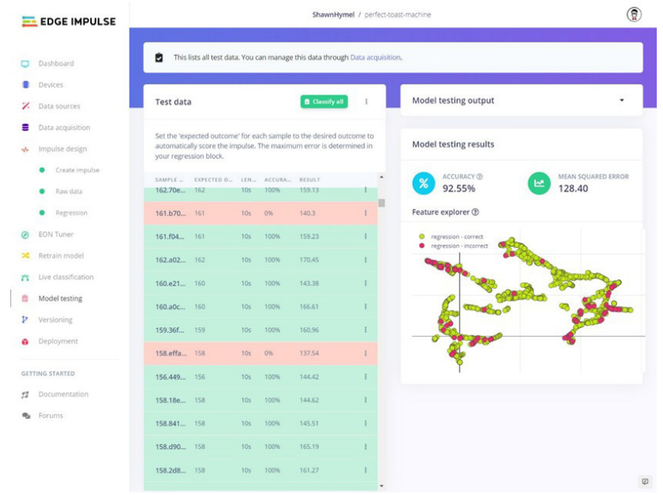
Clone this Edge Impulse project as a starting point: studio.edgeimpulse.com/public/129477/latest

If you wish to use your own data, delete all of the data in the project and upload the data from the unzipped out.zip file. Note that you should upload files in the training/ directory to training in Edge Impulse and upload files in testing/ to testing.

Go to Raw data. Click Save parameters and then Generate features.

When that's done, go to Regression. Feel free to modify the model, if you'd like. Click Start training and wait for training to finish.

Go to Model testing and click Classify all. When that's done, you should see some estimates. Expected outcome is the ground-truth label that expresses the number of seconds until the toast is burned, which is calculated based on when the state label transitioned from 'toasting' to 'burnt' (0 being the moment we believe the toast went from 'toast' to 'burnt toast'). The result is the output of our model, given the test input data. For the most part, the model is capable of predicting 'time till burnt' within a few seconds. Interestingly, the model seems to be more accurate the closer to 0.




## Deployment

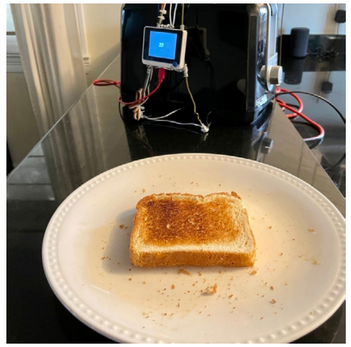
Go to the Deployment page, select Arduino library, and click Build. When the library has been downloaded (do not unzip it!) open the Arduino IDE, click Sketch > Include Library > Add .ZIP Library … select the Arduino library that you just downloaded from Edge Impulse. Note the name of the library! If it is different than ei-perfect-toast-machine-arduino-x.x.x.zip, you will need to change the .h include file name in your inference code.

Copy the perfect-toast-machine Arduino code found here to a new Arduino project. Read the comments at the beginning to see which libraries you need to install. Rename perfect-toast-machine_inferencing.h if you used a different project name in Edge Impulse. Upload the code to your Wio Terminal.

Copy the means and standard deviations from the Colab script to the means and std_devs arrays, respectively.

The first time you try the project, pay attention to the number displayed on the Wio Terminal. This shows the number of predicted seconds until the toast is burned. If your toast comes out to light, lower the CANCEL_THRESHOLD in the code (i.e. wait until it is closer to being burned before popping the toast up). If the toast is too dark, increase the CANCEL_THRESHOLD (i.e. stop the toasting process sooner).

With a little tweaking, you should be able to make perfect toast! Try different types of bread, different thicknesses, different starting temperatures, etc. It should also work for two slices of bread and even bagels!



## Recommended reading

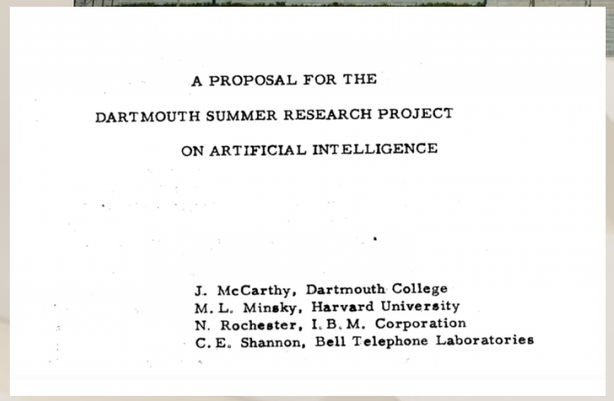
All code in this project can be found in this GitHub repository. The Edge Impulse project used for training the machine learning model is found here.

Interestingly enough, creating the perfect toast is an exercise in predictive maintenance. Instead of 'toast', imagine we're talking about an expensive piece of machinery. Is it possible to create a system that predicts when the machinery will fail (analogous to the toast burning) and notify us before it does fail (e.g. stopping the toasting process just before burning)?

Performing routine maintenance might help limit or prevent machine failure, but it is often done more than necessary, which increases equipment downtime. Predictive maintenance systems help limit this downtime by notifying us before a machine breaks so we can repair it at the appropriate times. You can read more about predictive maintenance in this blog post.

# Programming a calculator to form concepts: the organizers of the Dartmouth Summer Research Project

Written by David Ray,
Cyber City Circuits



A PROPOSAL FOR THE

DARTMOUTH SUMMER RESEARCH PROJECT

ON ARTIFICIAL INTELLIGENCE

J. McCarthy, Dartmouth College
M. L. Minsky, Harvard University
N. Rochester, I. B. M. Corporation
C. E. Shannon, Bell Telephone Laboratories

Message to the reader: this article complements a previous article about the proposal for the Dartmouth summer research project on artificial intelligence. If you would like to learn more, please read '**Programming a Calculator to Form Concepts: The Birth of Artificial Intelligence**'

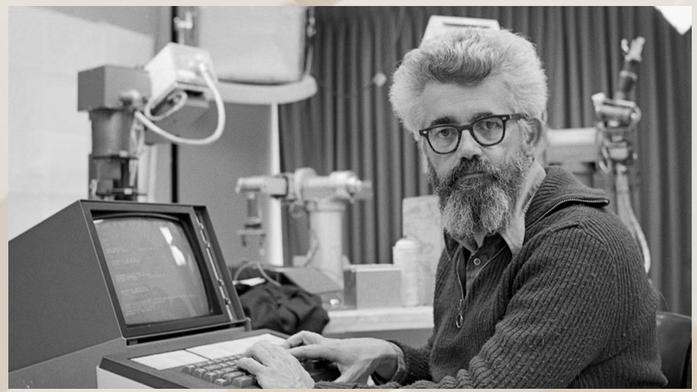
## A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence

In the summer of 1956, a groundbreaking proposal was made for what would become a milestone event in technological history: the Dartmouth Summer Research Project on Artificial Intelligence. This initiative, conceived by a group of visionary scientists, aimed to explore the nascent field of AI, which at the time was more a concept of science fiction than a tangible reality.

The proposal was simple yet ambitious: to assemble a group of mathematicians, logicians, and computer scientists for two months to delve into creating machines capable of simulating human intelligence. The goal was not just to mimic human thought but to surpass it, to automate processes that until then had been the exclusive domain of the human mind.

This project laid the groundwork for what we now recognize as AI/ML, influencing everything from the development of expert systems to the neural networks that power today's AI applications. The Dartmouth conference became a beacon of innovation, igniting a revolution that would reshape technology, business, and everyday life in ways its creators could hardly envision.

This is the story of those creators.



## John McCarthy, Dartmouth College

John McCarthy (1927-2011) is most famous for coining the term Artificial Intelligence. After completing his undergraduate degree at the California Institute of Technology (Caltech) in 1948, McCarthy pursued a PhD in mathematics from Princeton University. At the time, computers were just beginning to emerge as powerful tools for scientific and engineering tasks, and McCarthy saw their potential to model the human thought process.

As part of his PhD program, he spent at least one summer working at Bell Labs. This is where he met Claude Shannon and Marvin Minsky. Finishing his doctorate, Dr. McCarthy worked as a junior professor at Princeton before joining Dartmouth College's faculty in the summer of 1955. While at Dartmouth College, McCarthy introduced the term 'Artificial Intelligence' to describe the scope of topics outlined in the 1956 Summer Research Project Proposal.

**Retro Electro fun fact:** while enrolled at Cal Tech, McCarthy was suspended for not attending any Physical Education classes, and he enlisted in the US Army in 1945. Joining shortly before the war ended.

One of McCarthy's more significant works was the development of the

"Every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it."

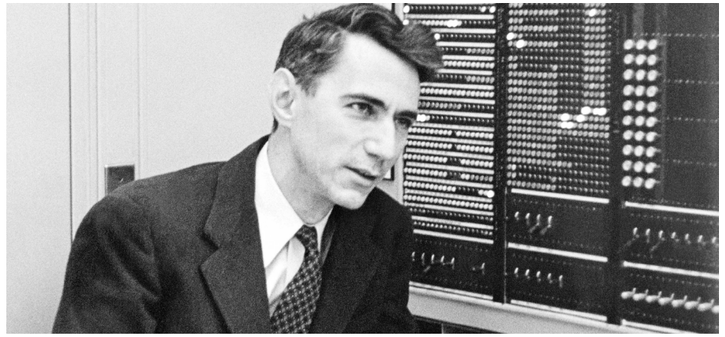
– J. McCarthy

LISP programming language. Due to its flexible memory management and ability to process symbolic expressions quickly, LISP became the language of choice for AI research and development. It introduced several pioneering concepts, including tree data structures, automatic storage management, and a self-hosting compiler.

In 1959 while working at Stanford University, where he stayed until retiring at the beginning of 2001, he published a paper titled 'Programs with Common Sense,' where he worked with Marvin Minsky and explained the need to find a way to teach common sense and natural law, aiming to equip AI systems with the everyday knowledge that humans take for granted.

McCarthy was an early pioneer of time-share computing, which allowed multiple users to interact with a single computer simultaneously. This idea was instrumental in the development of the modern internet and cloud computing.

For more on time-share computing, read the Retro Electro article on 'The Aloha System: Task II'



## Claude E. Shannon, Bell Labs

Claude Elwood Shannon (1916-2001) was an electrical engineer and unicycle enthusiast. His father was an attorney and judge, while his mother was the principal of the local high school. As a child, he was a hobbyist mechanic who built model planes and a radio-controlled boat. He even built a small telegraph between his house and his childhood friend's house. As a young man, he earned money by repairing radios at the local store.

Being an overachiever, he graduated from the University of Michigan in 1936 with bachelor's degrees in mathematics and electrical engineering. Afterward, he found a position as a research assistant running MIT's 'Differential Analyzer', which allowed him to fund his master's degree in electrical engineering.

The 'Differential Analyzer' could solve differential equations to the sixth degree. Research scientists presented him with equations each day and Shannon configured the machine to solve them.

The machine was made up of around a hundred relays to control the operations, and much of his time was spent returning it to working order and repairing malfunctions. He said he would think of new ways to design each circuit as he worked on it. He found that the symbolic logic he learned at the University of Michigan could be used to describe what happens in a switching relay circuit.

His master's thesis, 'A Symbolic Analysis of Relay and Switching Circuits,' is one of the most important foundational works in Computer Science. In it, he shows how logic operators, like 'and,' 'or,' etc., can be used to solve and simplify problems with relays used in telephone switching systems,

"That's the story of my life, the interplay between Mathematics and electrical engineering."

– C.E. Shannon

"(The) main reason the 1956 Dartmouth workshop did not live up to my expectations is that AI is harder than we thought."

- Marvin Minsky

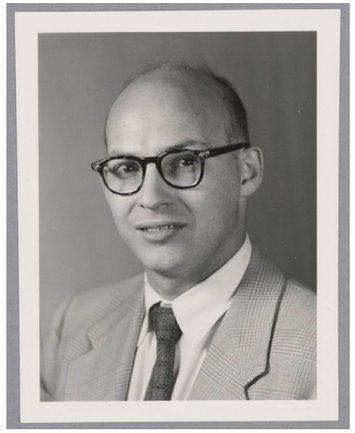
laying the groundwork for the future of digital design. For formally bringing Boolean logic to electrical engineering, he was awarded the Alfred Noble Prize (not to be confused with the Nobel Prize) in 1939. If there were a proper beginning of the 'digital age,' this document would likely be it.

Immediately following his master's program, he started a PhD program in mathematics at MIT, where he worked on problems describing genetics using algebra and Boolean operators.

After school, Dr. Shannon took a position at Bell Laboratories, where he solved problems ranging from 'color coding' to encryption. This was during the beginning of the United States' active involvement in World War II. While at Bell Labs, Shannon seems to have compulsively solved highly complex problems that others couldn't. He was described as 'finding answers to important questions nobody else was asking'. He was not 'cleared' to work in the area of encryption, but that did not stop him. In his spare time, he worked on the problems surrounding encryption and then explained it to the engineers in that department while having lunch in the cafeteria. It was later discovered that his work was instrumental in the encryption of communications used in the Manhattan Project and between Winston Churchill and Roosevelt.

In 1952, Shannon built 'Theseus'. An electromechanical 'mouse in a maze' that could solve itself automatically and in a very short time. It was made up of a couple of motors, several dozen relays, and a bar magnet dressed up like a mouse. It could navigate a customizable maze to a goal and after it initially solved the maze, it could be lifted and placed anywhere it would move straight to the goal, without any false moves.

"The real significance of this mouse and maze, lies in the four rather unusual operations it is able to perform. It has the ability to solve a problem by trial and error means, remember a solution and apply it when necessary at a later date, add new information to the solution already remembered, and forget one solution and learn a new one when the problem is changed." – C.E. Shannon



## Marvin Minsky, Harvard University

If McCarthy was the 'Father of AI,' then Minsky was the Architect. He grew up in New York City and attended the Bronx High School of Science and was a Navy veteran. In an interview, Minsky explained that when he graduated from grade school in 1944, the military draft was still active in support of World War II. To avoid being drafted into the Army, he enlisted in the Navy, where he was trained in electronics, radio, RADAR, etc. Concerning his time in the Navy, he recounts that he was in boot camp when Japan surrendered, which was a relief.

**Retro Electro fun fact:** legend has it that during his tenure at Bell Labs, Minsky invented the 'Useless Machine' novelty toy, which is now on office desks worldwide.

*The Bronx High School of Science taught many of the world's visionaries, including Carl Sagan,*

*Richard Feynman, Neil deGrasse Tyson, and many Pulitzer Prize and Nobel Prize winners.*

After two years in the Navy, Minsky enrolled at Harvard University, where he graduated with a degree in mathematics. He then attended Princeton University as a graduate student, earning his PhD. While attending Princeton, he worked at nearby Bell Laboratories. During this time, Minsky also began to simulate human intelligence with a project named SNARC.

The 'Stochastic Neural Analog Reinforcement Calculator' (SNARC) was a project at Bell Labs with Marvin Minsky while he was a graduate student at Princeton. It is credited as being the very first 'neural network' computer ever developed. Shannon would later use the SNARC in his previously mentioned 'Theseus' project.

After graduation, Dr. Minsky worked briefly as a Junior Fellow at Harvard before joining the faculty at MIT in 1958, where he continued until his death in 2016.

In 1963, Minsky and McCarthy founded MIT's Artificial Intelligence Lab. J.C.R Licklider and ARPA funded the lab under the name 'Project MAC'. Many things came out of what is now known as the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), including the first Time-Share computing system, which was improved upon by 'Project GENIE' at Berkeley. CSAIL is credited with many of the technological innovations of the late twentieth century.

## Nathaniel Rochester, IBM

Nathaniel Rochester (1919-2001) was a pivotal figure in the early development of computing. He attended MIT and graduated with a bachelor's degree in electrical engineering in 1941. His career began working at MIT's Research Labs to develop radar systems for the U.S. Navy and later at Sylvania, a bulb and vacuum tube manufacturer, where he continued to advance radar technology critical to the war effort.

Following the war, in 1948, Rochester joined IBM, where he was one of the two key designers of the IBM 701, the company's first mass-produced scientific computer. Released in 1951, the IBM 701 marked a significant leap in computing power, enabling complex calculations that were previously impractical. This machine also formally marks the beginning of IBM's move away from conventional punchcard time clocks and mechanical typewriters to focus on electric computers.

Early in 1955, IBM tasked Rochester with leading a new research group at IBM focused on the new fields of information theory and automatic pattern recognition. He programmed the first neural network simulations on the IBM 704 in this effort.

In the 1960s, Rochester became more involved in the broader computing community. He contributed to the design and standardization of programming languages, and his work influenced the development of FORTRAN at IBM, one of the earliest and most widely used programming languages of its day.

## Pioneering Artificial Intelligence

The proposal for the Dartmouth Summer Research Project on AI, led by John McCarthy, marked a historic moment in computer science. This initiative established the foundational framework for what would grow into the field of AI, creating an environment where ideas could thrive and machines could start to mimic human cognition. The completion of the project marked not just an end, but a beginning, igniting a decades-long pursuit to create intelligent systems capable of learning, reasoning, and adapting.

*The group that attended the Dartmouth Summer Research Project on Artificial Intelligence.*

Today, we see the fruits of this endeavor in the advanced AI technologies that permeate our lives, a testament to the visionary foresight of the early pioneers at Dartmouth.

## Suggested reading


1. A Proposal For the Dartmouth Summer Research Project on Artificial Intelligence
2. Marvin Minsky Memorial (MIT News)
3. Ray Solomonoff's Personal 'Dartmouth Archives'
4. 'AI: The Tumultuous History Of The Search For Artificial Intelligence' by D. Crevier
5. 'Newell And Simon's Logic Theorist: Historical Background and Impact On Cognitive Modeling'
6. 'Society of the Mind' by Marvin Minsky
7. 'The Meeting of the Minds That Launched AI' by Grace Solomonoff
8. The Turbulent Past and Uncertain Future of Artificial Intelligence by Eliza Strickland
9. 'Oral History of Nathaniel Rochester' Interview by A. Goldstein (June 1991)
10. 'Programs With Common Sense' by J. McCarthy
11. 'A Symbolic Analysis of Relay and Switching Circuits' by C.E. Shannon
12. 'Claude E. Shannon: A Retrospective on His Life, Work, and Impact' by R.G. Gallager
13. (Video) 'Claude Shannon – Father of the Information Age' from the University of California
14. 'Claude E. Shannon: A Goliath Amongst Giants' Presented by Nokia Bell Labs
15. 'Mouse With a Memory' by Bell Labs
16. 'SNARC' from HistoryOf.AI
17. (Video) Claude Shannon demonstrates "Theseus" Machine Learning @ Bell Labs
18. (Video) Marvin Minsky Interview (Recorded in 2002)
19. (Video) Marvin Minsky Interview (Recorded in 1990 for WGBH)
20. (Video) Marvin Minsky Interview Series (Life Stories of Remarkable People)

# What is data-preparation in ML, and why is it crucial for success?

This article explores how preprocessing prepares data for machine learning, tackles challenges like missing values and outliers, and helps ensure fair and accurate model results. Further, it explores vital techniques such as scaling, rebalancing, and variable transformation.

## How preprocessing prepares data for ML projects

Data-preparation, or preprocessing, is one of the most crucial steps that nearly every machine learning project must undergo to succeed. The process prepares raw data for analysis by addressing potential issues found in the samples by handling missing values, addressing outliers and inconsistencies, and encoding features into a suitable format for the algorithm. Besides ensuring data quality, preprocessing also reduces noise and often helps prevent or address issues such as overfitting. Finally, preprocessing is an important task when optimizing the performance of an algorithm, as the data-preparation step often helps reduce the dimensionality of the data by identifying the most relevant features.

## Scaling, normalization, and standardization

Some algorithms, such as kNN (K-Nearest Neighbors) or SVMs (Support Vector Machines), rely on the distance between data samples and are thus sensitive to the magnitude of features. If the distances in the data are off, some instances may dominate the learning process, leading to poor model performance. Scaling features ensure that they contribute equally to the model's learning process, leading to fair and accurate results.

There are various methods engineers can employ to ensure fair scaling across the variables. However, min-max scaling is a popular and easy-to-understand method that transforms features to a range between zero and one, ensuring that the minimum value becomes zero, the maximum value becomes one, and all other values are proportionally scaled in between, depending on their original value.

In another popular approach, Z-score standardization, each variable is converted to have a mean of zero and a standard deviation of one. The method achieves this by subtracting the global mean of the variable from each value in the dataset and then dividing it by the variable's standard deviation.

## How oversampling and undersampling help maintain fairness

Datasets with significant disparities in class distribution (e.g., 90% positive instances, 10% negative) can cause certain classifiers, such as kNN, to ignore the minority class or overestimate the importance of positive samples. In such cases, the training set can be rebalanced using one of two methods. In undersampling, entries from the majority class are omitted, while in oversampling, samples from the minority class are duplicated to achieve a balanced representation of the classes.



## Transforming variables: one-hot encoding and label-encoding

Some algorithms can only process numerical values. Therefore, some datasets require representing textual labels with numbers. This task can be achieved, for example, by introducing n new attributes – each representing one of the n old textual values. The new features have a value of 0 everywhere except for the exact position that represents the old label of the data point (one-hot encoding.)

Alternatively, the original labels can be assigned different numerical values. All data points with that label then receive the new numerical index of the label (label-encoding.)

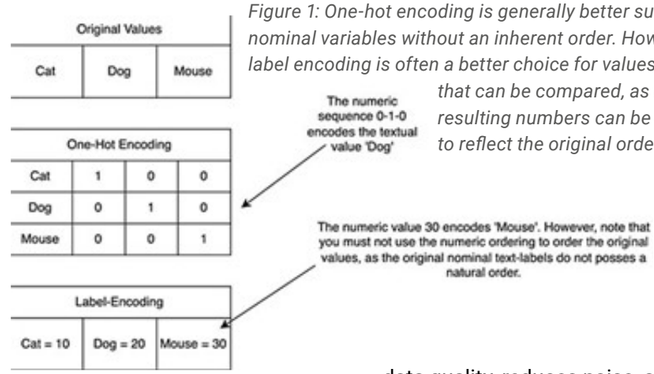



*Figure 1: One-hot encoding is generally better suited for nominal variables without an inherent order. However, label encoding is often a better choice for values that can be compared, as the resulting numbers can be chosen to reflect the original ordering.*

## Addressing missing values in ML preprocessing

The simplest solution is to delete samples with missing values from the training dataset. However, this is not an option if the model later needs to handle missing values (e.g., during testing.)

Alternatively, one can introduce a custom N/A or null value, select a random value from another entry, or compute the average or nearest integer value. These imputation methods are only meaningful when the number of missing values is small and when the entries with missing data are too critical to be deleted.

## How to apply preprocessing to prevent data leakage

Data leakage describes a problem where information from the training set leaks into the test set, which must never happen. Therefore, the original dataset must always be divided first (e.g., using holdout or cross-validation) before applying any preprocessing steps. Finally, these preprocessing steps should only be applied to the test dataset rather than the training data. The test sample must always remain unaltered to simulate realistic conditions when evaluating the model's performance later.

## Summary

Preprocessing is a critical step in machine learning projects that prepares data for analysis and ensures the project's success. It involves addressing issues like missing values, outliers, and inconsistencies and encoding features in a suitable format for the algorithm. Preprocessing enhances data quality, reduces noise, and helps prevent overfitting.

Scaling techniques like min-max scaling ensure that features contribute equally to the learning process by distributing all values between zero and one. At the same time, Z-score standardization converts variables to have a mean of zero and a standard deviation of one.

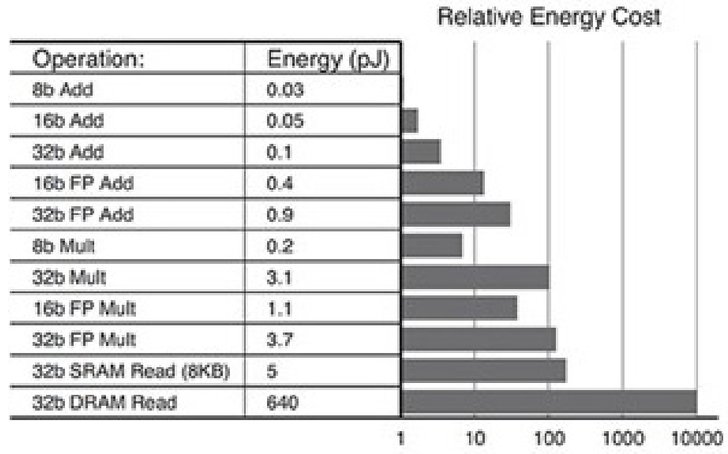
Rebalancing techniques like oversampling and undersampling can address class imbalances in the data. Transforming variables through one-hot encoding and label encoding enables the processing of textual data in algorithms that can only handle numbers. Handling missing values can involve deletion, imputation, or assigning custom values.

Finally, dividing the original dataset before preprocessing is crucial to prevent data leakage, where information from the training set influences the test set. The test set should remain unchanged for a realistic evaluation of the model's performance.

# How to rapidly design and deploy smart machine vision systems

Written by Jeff Shepard

*Figure 1: An order of magnitude less energy is needed for INT8 (8b Add) operations compared with FP32 operations (32b Add). Image source: AMD Xilinx*




| Operation: | Energy (pJ) |
|---|---|
| 8b Add | 0.03 |
| 16b Add | 0.05 |
| 32b Add | 0.1 |
| 16b FP Add | 0.4 |
| 32b FP Add | 0.9 |
| 8b Mult | 0.2 |
| 32b Mult | 3.1 |
| 16b FP Mult | 1.1 |
| 32b FP Mult | 3.7 |
| 32b SRAM Read (8KB) | 5 |
| 32b DRAM Read | 640 |

The need for machine vision is growing across a range of applications, including security, traffic and city cameras, retail analytics, automated inspection, process control, and vision-guided robotics. Machine vision is complex to implement and requires the integration of diverse technologies and sub-systems, including high-performance hardware and advanced artificial intelligence/machine learning (AI/ML) software. It begins with optimizing the video capture technology and vision I/O to meet the application needs and extends to multiple image processing pipelines for efficient connectivity. It is ultimately dependent on enabling the embedded-vision system to perform vision-based analytics in real time using high-performance hardware such as field programmable gate arrays (FPGAs), systems on modules (SOMs), systems on chips (SoCs), and even multi-processor systems on chips (MPSoCs) to run the needed AI/ML image processing and recognition software. This can be a complex, costly, and time-consuming process that is exposed to numerous opportunities for cost overruns and schedule delays.

Instead of starting from scratch, designers can turn to a well-curated, high-performance development platform that speeds time to market, controls costs, and reduces development risks while supporting high degrees of application flexibility and performance. A SOM-based development platform can provide an integrated hardware and software environment, enabling developers to focus on application customization and save up to nine months of development time. In addition to the development environment, the same SOM architecture is available in production-optimized configurations for commercial and industrial environments, enhancing application reliability and quality, further reducing risks, and speeding up time to market.

Instead of starting from scratch, designers can turn to a well-curated, high-performance development platform that speeds time, controls costs, and reduces development risks while supporting high degrees of application flexibility and performance.

This article begins by reviewing the challenges associated with the development of high-performance machine vision systems, then presents the comprehensive development environment offered by the Kria KV260 vision AI starter kit from AMD Xilinx, and closes with examples of production-

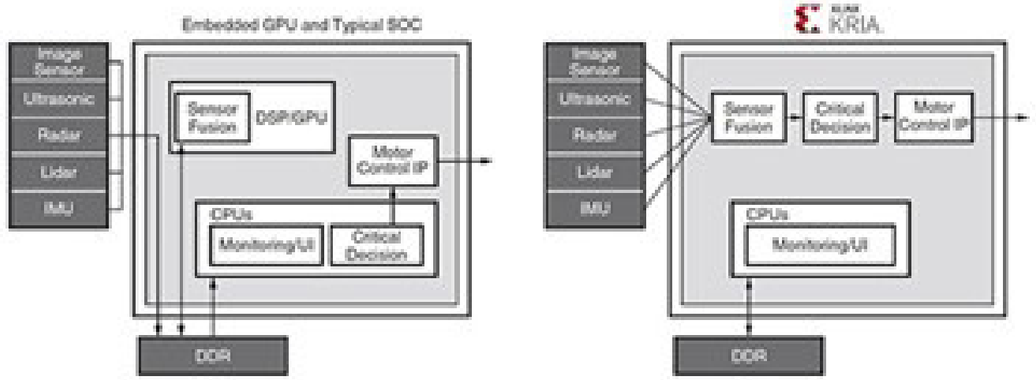



*Figure 2: In this typical automotive application, the GPU requires multiple accesses to DDR for communication between the various modules (left), while the pipeline architecture of the Zynq MPSoC (right) avoids most DDR accesses. Image source: AMD Xilinx*

ready SOMs based on the Kira 26 platform designed to be plugged into a carrier card with solution-specific peripherals.

## It begins with data type optimization

The needs of deep learning algorithms are evolving. Not every application needs high-precision calculations. Lower precision data types such as INT8, or custom data formats, are being used. GPU-based systems can be challenged with trying to modify architectures optimized for high-precision data to accommodate lower-precision data formats efficiently. The Kria K26 SOM is reconfigurable, enabling it to support a wide range of data types from FP32 to INT8 and others. Reconfigurability also results in lower overall energy consumption. For example, operations optimized for INT8 consume an order of magnitude less energy compared with an FP32 operation (Figure 1).

## Optimal architecture for minimal power consumption

Designs implemented based on a multicore GPU or CPU architecture can be power-hungry based on typical power usage patterns:

- 30% for the cores
- 30% for the internal memory (L1, L2, L3)
- 40% for the external memory (such as DDR)

Frequent accesses to inefficient DDR memory are required by GPUs to support programmability and can be a bottleneck to high bandwidth computing demands. The Zynq MPSoC architecture used in the Kria K26 SOM supports the development of applications with little or no access to external memory. For example, in a typical automotive application, communication between the GPU and various modules requires multiple accesses to external DDR memory, while the Zynq MPSoC-based solution incorporates a pipeline designed to avoid most DDR accesses (Figure 2).

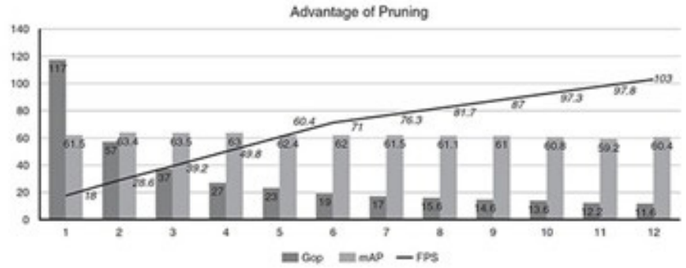



*Figure 3: After a relatively few iterations, pruning can reduce model complexity (Gop) by 10X and improve performance (FPS) by 5X, with only a 1% reduction in accuracy (mAP). Image source: AMD Xilinx*

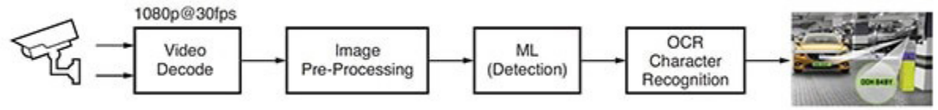



*Figure 4: Typical image processing flow for an AI-based ANPR application. Image source: AMD Xilinx*

## Pruning leverages the advantages

The performance of neural networks on the K26 SOM can be enhanced using an AI optimization tool that enables data optimization and pruning. It's very common for neural networks to be over-parameterized, leading to high levels of redundancy that can be optimized using data pruning and model compression. Using Xilinx's AI Optimizer can result in a 50x reduction in model complexity, with a nominal impact on model accuracy. For example, a single-shot detector (SSD) plus a VGG convolution neural net (CNN) architecture with 117 Giga Operations (Gops) was refined over 11 iterations of pruning using the AI Optimizer. Before optimization, the model ran 18 frames per second (FPS) on a Zynq UltraScale+ MPSoC. After 11 iterations – the 12th run of the model – the complexity was reduced from 117 Gops to 11.6 Gops (10X), performance increased from 18 to 103 FPS (5X), and accuracy dropped from 61.55 mean average precision (mAP) for object detection to 60.4 mAP (only 1% lower) (Figure 3).

## Real-world application example

A machine learning application for automobile license plate detection and recognition, also called auto number plate recognition (ANPR), was developed based on vision analytics software from Uncanny Vision. ANPR is used in automated toll systems, highway monitoring, secure gate and parking access, and other applications. This ANPR application includes an AI-based pipeline that decodes the video and preprocesses the image, followed by ML detection and OCR character recognition (Figure 4).


*Figure 5: The Kria KV260 vision AI starter kit is a comprehensive development environment for machine vision applications. Image source: AMD Xilinx*

Implementing ANPR requires one or more H.264 or H.265 encoded real-time streaming protocol (RTSP) feeds that are decoded or uncompressed. The decoded video frames are scaled, cropped, color space converted, and normalized (pre-processed), then sent to the ML detection algorithm. High-performance ANPR implementations require a multi-stage AI pipeline. The first stage detects and localizes the vehicle in the image, creating the region of interest (ROI). At the same time, other algorithms optimize the image quality for subsequent use by the OCR character recognition algorithm and track the vehicle's motion across multiple frames. The vehicle ROI is further cropped to generate the number plate ROI processed by the OCR algorithm to determine the characters in the number plate. Compared with other commercial SOMs based on GPUs or CPUs, Uncanny Vision's ANPR application ran 2-3X faster on the Kira KV260 SOM, costing less than $100 per RTSP feed.

*Figure 6: The KV260 vision AI starter kit includes: (top row, left to right) power supply, Ethernet cable, microSD card, and (bottom row, left to right) USB cable, HDMI cable, camera module. Image: AMD Xilinx*

## Smart vision development environment

Designers of smart vision applications like traffic and city cameras, retail analytics, security, industrial automation, and robotics can turn to the Kria K26 SOM AI Starter development environment. This environment is built using the Zynq UltraScale+ MPSoC architecture and has a growing library of curated application software packages (Figure 5). The AI Starter SOM includes a quad-core Arm Cortex-A53 processor, over 250 thousand logic cells, and an H.264/265 video codec. The SOM also has 4 GB of DDR4 memory, 245 IOs, and 1.4 tera-ops of AI compute to support the creation of high-performance vision AI applications offering more than 3X higher performance at lower latency and power compared with other hardware approaches. The pre-built applications enable initial designs to run in less than an hour.

To help jump-start the development process using the Kria K26 SOM, AMD Xilinx offers the KV260 vision AI starter kit that includes a power adapter, Ethernet cable, microSD card, USB cable, HDMI cable, and a camera module (Figure 6). If the entire starter kit is not required, developers can simply purchase the optional power adapter to start using the Kira K26 SOM.

Another factor that speeds development is the comprehensive array of features, including abundant 1.8 V, 3.3 V single-ended, and differential I/Os with four 6 Gb/s transceivers and four 12.5 Gb/s transceivers. These features enable the development of applications with higher numbers of image sensors per SOM and many variations of sensor interfaces such as MIPI, LVDS, SLVS, and SLVS-EC, which are not always supported by application-specific standard products (ASSPs) or GPUs. Developers can also implement DisplayPort, HDMI, PCIe, USB2.0/3.0, and user-defined standards with the embedded programmable logic.

Finally, the development of AI applications has been simplified and made more accessible by coupling the extensive hardware capabilities and software environment of the K26 SOM with production-ready vision applications. These vision applications can be implemented with no FPGA hardware design required and enable software developers to quickly integrate custom AI models and application code and even modify the vision pipeline. The Vitis unified software development platform and libraries from Xilinx support common design environments, such as TensorFlow, Pytorch, and Café frameworks, as well as multiple programming languages including C, C++, OpenCL, and Python. There is also an embedded app store for edge applications using Kria SOMs from Xilinx and its ecosystem partners. Xilinx offerings are free and open source and include smart camera tracking and face detection, natural language processing with smart vision, and more.


*Figure 7: Production-optimized Kira 26 SOMs for industrial and commercial environments are designed to be plugged into a carrier card with solution-specific peripherals. Image: DigiKey*

## Production optimized Kira 26 SOMs

Once the development process has been completed, production-ready versions of the K26 SOM designed to be plugged into a carrier card with solution-specific peripherals that can speed the transition into manufacturing (Figure 7) are available. The basic K26 SOM is a commercial-grade unit with a temperature rating of 0°C to +85°C junction temperature, as measured by the internal temperature sensor.

An industrial-grade version of the K26 SOM rated for operation from -40°C to +100°C, is also available.

The industrial market demands long operational life in harsh environments. The industrial-grade Kria SOM is designed for ten years of operation at 100°C junction and 80% relative humidity and to withstand up to 40 g of shock, and 5 g root mean square (RMS) of vibration. It also comes with a minimum production availability of ten years to support long product lifecycles.

## Summary

Designers of machine vision applications such as security, traffic, and city cameras, retail analytics, automated inspection, process control, and vision-guided robotics can turn to the Kria K26 SOM AI Starter to speed time to market, help to control costs and reduce development risks. This SOM-based development platform is an integrated hardware and software environment, enabling developers to focus on application customization and save up to nine months of development time. The same SOM architecture is available in production-optimized configurations for commercial and industrial environments, further speeding time to market. The industrial version has a minimum production availability of 10 years to support long product lifecycles.

# Road-tested GMSL cameras drive into new markets

Written by Pete Bartolik

Technologies developed for automotive applications frequently transfer to other markets due to automobile manufacturers' rigorous requirements for reliability, performance, and the need for fast data rates in an electronically hostile environment. That's why Gigabit Multimedia Serial Link (GMSL) cameras are finding ready markets for vision applications in areas such as automation and robotics, smart agriculture, digital healthcare, avionics, robotaxis, and retail and warehouse inventory management.

Initially introduced for addressing applications for high-speed video and data transmission in vehicles, Analog Devices GMSL is a widely adopted and proven technology for bringing new levels of performance to high-speed video links and enabling multi-streaming over a single cable.

Vision applications require very large data streams to ensure high-quality video. A full HD image is comprised of 1080 rows by 1920 columns. That amounts to 2 million pixels, each of which consists of a red, green, and blue element, resulting in 6 million elements. Each element represents 8 bits of data, so every frame results in nearly 50 Mbps of data. At 60 frames per second, the required data rate for one camera is over three-and-a-half Gbps.

First-generation GMSL, first available in 2008, utilized the low-voltage differential signaling (LVDS) standard to deliver parallel data downlink rates up to 3.125 Gbps. That was particularly suited for conveying data from multiple camera systems and other advanced driver assistance applications (ADAS), as well as the growing use of in-car, high-definition flat panel displays.

A second generation, GMSL2, was introduced in 2018, increasing data rates up to 6 Gbps and supporting more standard highspeed video interfaces, including HDMI and the MIPI interface standard, a popular image sensor interface for consumer and automotive cameras. These advances accommodated full high definition (FHD) displays and cameras with resolution up to 8 MP.

GMSL3, the next generation, can deliver data up to 12 Gbps over a single cable, supports multiple 4K resolution streams, the daisy-chaining of multiple displays, and aggregation of multiple cameras such as those located on the front, back, and sides of a vehicle to provide a 360° viewing capability. Today, increasing numbers of automobile manufacturers supplement rear and side-view mirrors with cameras, utilize forward and rear-facing cameras for collision avoidance, and internal cabin cameras for monitoring driver and passenger safety. GMSL3 can aggregate data from multiple video feeds as well as LiDAR and radar.

With cameras scaled down to the level of CMOS sensors, they can produce what once was considered incredible quality at low cost and with low power demands. Image sensors have millions of receptor elements, each of which converts measurements into digital values to be streamed via serial data lanes of a parallel interface, along with synchronization information.

Both GMSL2 and GMSL3 utilize MIPI interface standards that provide designers and vendors access to a wide range of image sensors for GMSL cameras.

## GMSL versus GigE

Engineers starting out on vision applications will no doubt quickly face a decision on whether to use GMSL or gigabit Ethernet (GigE) vision technology. GigE is widely used in industrial applications due largely to its reliance on Ethernet network infrastructure and standards.

GigE Vision cameras with 2.5 GigE, 5 GigE, and 10 GigE are commonplace in applications today, and 100 GigE state-of-the-art cameras can utilize up to a 100 Gbps data rate. GMSL is designed to transmit data over coaxial cable or shielded twisted pair cable at up to 15 meters, compared to 100 m for GigE, although both may be exceeded under certain conditions.

Each technology is capable of transmitting data and power

through the same cable: GMSL uses Power over Coax (PoC) so video, audio, control, data, and power can be transported on a single channel. Most GigE Vision applications rely on Power over Ethernet (PoE) for 4-pair Ethernet, or less commonly, Power over Data Line (PoDL) for Single-Pair Ethernet (SPE).

System requirements and application needs will determine which vision technology is most appropriate. GigE Vision, for example, may offer some advantages for single-camera applications, particularly where they connect directly to a PC or an embedded platform with an Ethernet port.

When using multiple cameras, GigE Vision applications will require use of a dedicated Ethernet switch, a network interface card (NIC) with multiple Ethernet ports, or an Ethernet switch IC. That switching requirement can potentially reduce the maximum total data rate and introduce unpredictable latency between the cameras and the terminal device, whereas GMSL provides a simpler, more direct architecture.

GigE Vision devices may support higher resolution and a higher frame rate – or both simultaneously – with additional buffering and compression. Frame buffering and processing are not provided by GMSL devices, so resolution and frame rate depend on what the image sensor can support within the link bandwidth. Engineers will need to determine a simple trade-off between resolution, frame rate, and pixel bit depth.

## GMSL simplifies high-speed video architecture

GigE Vision cameras typically utilize a signal chain that includes an image sensor, a processor, and an Ethernet physical layer (PHY) (Figure 1). Raw image data from the sensor is converted by the processor into Ethernet frames, often relying on compression or frame buffering to fit the data rate of the supported Ethernet bandwidth.

The GMSL camera signal chain utilizes a serializer/deserializer (SerDes) architecture that avoids the use of a processor (Figure 2). Instead, image sensor parallel data is converted by the serializer into a high-speed serial data stream. On the back end, a deserializer converts the serial data back into parallel form for processing by an electronic control unit (ECU) system-on-chip (SoC).

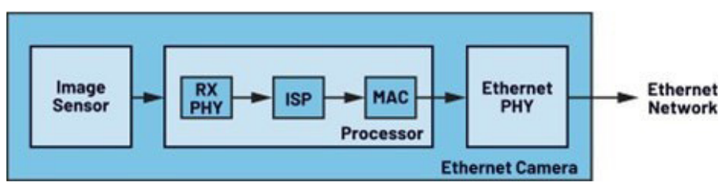



*Figure 1: Representation of key signal chain components on the sensor side of GigE Vision cameras. Image source: Analog Devices, Inc.*

The GMSL camera architecture makes it simpler to design small form factor cameras with low power consumption. Serializers can directly connect to cameras through standard MIPI CSI-2 interface and transmit packetized data through the GMSL link.

A typical host device is a customized embedded platform with one or more deserializers that transmit image data through MIPI transmitters in the same format as the image sensor MIPI output. New GMSL camera drivers are required for customized designs, but if there is an existing driver for the image sensor, it can be utilized with just a few profile registers, or register writes to enable a video stream from cameras to a control unit.

## GMSL components

ADI offers a comprehensive portfolio of serializers and deserializers to support a variety of interfaces. These feature robust PHY designs, low bit error rates (BER), and backward compatibility. Any video protocols can be bridged together – for example, HDMI to the Open LVDS Display Interface (oLDI).

Engineers will need to select the best components based on

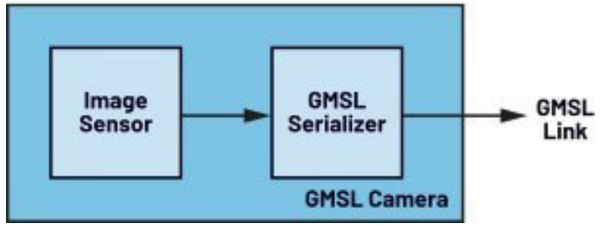



*Figure 2: GMSL cameras utilize a simpler signal chain architecture on the sensor side than GigE Vision. Image source: Analog Devices, Inc.*

application needs, such as device interfaces, data rates, bandwidth, power consumption, environmental conditions, and cable length. Other factors include EMI, error handling, and signal integrity. Some examples of ADI's GMSL components include:

- MAX96717, a CSI-2 to GMSL2 serializer (Figure 3), operates at a fixed rate of 3 Gbps or 6 Gbps in the forward direction and 187.5 Mbps in the reverse direction
- MAX96716A, which converts dual GMSL2 serial inputs to MIPI CSI-2. The GMSL2 inputs operate independently and video data from both can be aggregated for output on a single CSI-2 port or replicated on a second port for redundancy
- The MAX96724, a quad tunneling deserializer, converts four GMSL 2/1 inputs to 2 MIPI D-PHY or C-PHY outputs. Data link rates are 6/3 Gbps for GMSL2 and 3.12 Gbps for GMSL1, and reverse link rates of 187.5 Mbps for GMSL2 and 1 Mbps for GMSL1
- The MAX96714 deserializer converts a single GMSL 2/1 input to MIPI CSI-2 output, with a fixed rate of 3 Gbps or 6 Gbps in the forward direction and 187.5 Mbps in the reverse direction
- The MAX96751 is a GMSL2 serializer with HDMI 2.0 input that converts HDMI to single or dual GMSL2 serial protocol. It also enables full-duplex, single-wire transmission of video and bidirectional data
- The MAX9295D converts single- or dual-port 4-lane MIPI CSI-2 data streams to GMSL2 or GMSL1

ADI also offers several development tools, such as the MAX96724-BAK-EVK# evaluation kit for the MAX96724 devices.

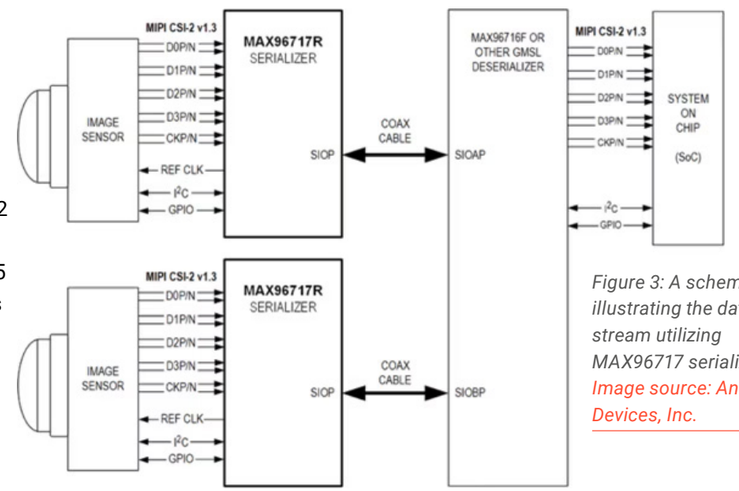



*Figure 3: A schematic illustrating the data stream utilizing MAX96717 serializers. Image source: Analog Devices, Inc.*

## Conclusion

With their reduced complexity, GMSL cameras are more compact and generally able to provide a more cost-effective solution compared to GigE Vision. GMSL provides reliable transport of high-resolution digital video with microsecond latency for a growing range of camera and display-based applications, from machine learning and autonomous operations to infotainment and safety. Millions of GMSL links are enhancing the driver experience on the road today, attesting to their reliability and performance.

# Understanding computer and machine vision

Written by Harry Fowle, Electronic Specifier

The ability for machines to interpret and analyze visual data is critical to many industries of today, enabling automated systems to extract meaningful information from images or video streams. At its core, this technology mimics how we perceive the world around us but leverages computational power, advanced algorithms, and specialized hardware to process vast amounts of visual data with high precision and speed.

The impact of machine and computer vision extends across a number of industries. In manufacturing, vision systems play a crucial role in quality control and defect detection. In automotive, vision enables autonomous vehicles to perceive and navigate their surroundings. Healthcare benefits from medical imaging applications, such as AI-assisted diagnostics, while retail and security leverage vision for facial recognition and behavioral analysis.

Modern machine vision systems rely on a combination of hardware and software components:

- **Sensors and cameras:** these capture images and video, ranging from simple 2D cameras to advanced LiDAR and hyperspectral imaging systems
- **Processing units:** GPUs, TPUs, FPGAs, and AI accelerators process visual data in real time, supporting deep learning-based decision-making
- **Algorithms:** classical techniques, such as edge detection and template matching, coexist with modern AI-driven approaches like convolutional neural networks (CNNs) and transformer-based vision models
- **Interfaces and communication protocols:** standardized interfaces (e.g., USB3 Vision, GigE Vision, MIPI) and industrial protocols (EtherCAT, Modbus) facilitate seamless integration with broader automation systems

*Figure 1: Modern machine vision system*



As the technology continues to evolve, machine and computer vision are shifting towards real-time, AI-powered, and Edge-based implementations. This article explores the history, evolution, and current landscape of machine/computer vision, diving into the technologies, topologies, and interfaces that enable them.

## The history of computer and machine vision

Whilst machine and computer vision seem like brand-new cutting-edge technology, its origins actually date back to the 1950s and 1960s, when researchers began exploring the possibility of enabling machines to interpret visual data. At these early stages, computer vision was primarily an academic pursuit, focused on recognizing simple patterns such as handwritten text and basic geometric shapes. Early efforts relied on rudimentary image processing techniques like edge detection and thresholding, laying the foundations for future advancements. Fast forward to the 1970s and feature extraction methods, including edge detection algorithms such as the Sobel operator, allowed for more sophisticated object recognition. Researchers also began developing techniques for 3D reconstruction, moving beyond simple two-dimensional image analysis.

The 1980s was a big decade for the technology, finally seeing its first practical applications, particularly in industrial automation. The emergence of digital imaging technology enabled early vision systems to be deployed for tasks such a defect detection, part inspection, and barcode reading. During this period, industries started adopting machine vision for quality control in manufacturing, leveraging these early systems to improve efficiency and accuracy across production lines. The 1990s saw further advancements, particularly in statistical image analysis and pattern recognition. The introduction of early neural networks into image processing signaled the beginning of AI-driven vision, though the technology was still in its infancy. Meanwhile, improvements in hardware, especially computational power, were making it possible to process images much faster and more efficiently. Feature-based vision techniques, such as Scale-Invariant Feature Transform (SIFT) and Speeded-Up Robust Features (SURF), began cropping up in the 2000s, massively improving object recognition. This period also saw the rapid expansion of 3D vision technologies, including stereo cameras and structured light systems, which allowed for more precise depth perception in automated vision systems.

However, the 2010s were the real turning point thanks to the rise of deep learning. The introduction of convolutional neural networks (CNNs) revolutionized computer vision by surpassing traditional algorithmic approaches. In 2012, the success of AlexNet in the ImageNet competition demonstrated that AI-powered models could achieve human-level accuracy in image classification, sparking widespread interest

*Figure 2: Camera performing 360° inspection of products on a conveyor*

in deep learning for vision applications. This breakthrough accelerated the development of more sophisticated AI-based vision systems, which soon found applications in autonomous vehicles, medical diagnostics, robotics, and facial recognition. Deep learning models such as ResNet, YOLO, and Vision Transformers further pushed the boundaries of what was possible in computer vision, enabling real-time object detection and recognition with unprecedented accuracy.

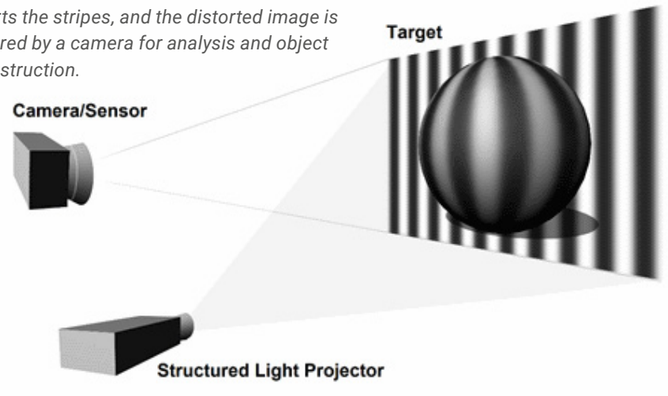
As vision systems have become increasingly reliant on AI the focus has shifted towards real-time processing and Edge computing. Edge AI has allowed machine vision applications to function in environments where latency was critical, such as robotics, industrial automation, and security systems. Today, machines and computers are embedded across a wide range of industries, from smart factories to AI-powered medical imaging. Advances in sensor technology, AI-driven models, and high-speed processors continue to refine and expand the capabilities of modern vision systems.

## Topologies, types, and technologies in computer and machine vision

Computer and machine vision systems are built on various topologies, sensor types, and processing technologies, each suited for different applications. Depth perception technologies, including structured light, stereo vision, time-of-flight (ToF), and LiDAR, provide different approaches to 3D vision, enabling machines to understand spatial relationship and movement with greater accuracy.

The topologies for depth sensing:

*Figure 3: A regular striped pattern is projected onto the ball. The rounded surface of the ball distorts the stripes, and the distorted image is captured by a camera for analysis and object reconstruction.*




### Structured light

Structured light depth sensing operates by projecting a predefined pattern – often a grid, stripe, or dot matrix – onto a scene and capturing its deformation with a camera. By analyzing how the projected pattern distorts upon contact with surfaces, the system reconstructs depth information with high precision. This technique is commonly implemented using infrared (IR) sources to avoid visible light interference.

As demonstrated in Figure 3, the depth is computed using triangulation principles. The projector and camera form a stereo vision system, where the displacement of structured patterns provides disparity information that translates into depth.

Depth, Z, is determined using the formula:

$$Z = \frac{B \cdot f}{d}$$

Here, B is the baseline distance between the projector and camera, f is the focal length, and d is the disparity.

Different projection encoding methods can be used to optimize accuracy, speed, or robustness. Phase-shift encoding uses a sinusoidal (wave-like) pattern projected with multiple phase shifts to calculate phase differences at subpixel precision. Gray code/binary-coded patterns make use of sequential binary patterns uniquely encoded onto each pixel, making it far more robust against occlusions and object discontinuities. Speckle projection is a pseudo-random dot matrix which enables dense depth estimation with a single frame, for example with Apple's Face ID.

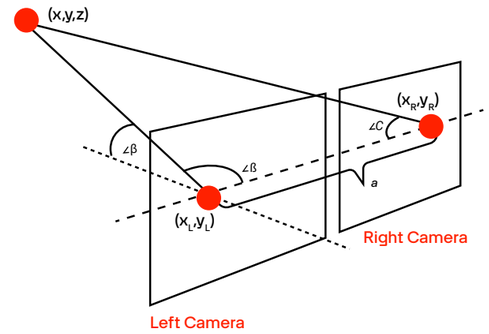



*Figure 4: A standard stereo vision system example.*

Technical advantages:

- High spatial resolution and depth accuracy at short ranges (typically under 5m)
- Effective for applications requiring fine-grained depth maps, such as 3D scanning, biometric authentication (e.g., Face ID), and industrial metrology
- Real-time operation with structured illumination strategies like phase-shifting or binary-coded patterns

Challenges:

- Susceptible to ambient light variations, making outdoor performance unreliable
- Requires a high-framerate camera and precise calibration for accurate reconstruction
- Performance degrades with highly reflective or transparent surfaces

### Stereo vision

Stereo vision replicates human binocular perception by capturing two images from slightly offset cameras and computing depth through disparity analysis.

The correspondence problem – matching pixels between left and right images – is addressed through algorithms such as block matching, semi-global matching (SGM), or deep learning-based methods. It is important to consider with disparity map computation that:

- Local methods, such as block matching, are faster but more sensitive to noise and texture
- Global methods, such as SGM, optimize for disparity consistency across images
- Deep learning methods, like CNNs and transformer-based models, improve robustness in low-texture or occluded regions

The resulting disparity map then provides relative depth estimation where the baseline (distance between cameras) and focal length determine the depth accuracy. A larger baseline improves depth precision but increases occlusion.

Technical advantages:

- Passive system that does not require active light projection, enabling outdoor operation
- Well-suited for robotic navigation, SLAM (Simultaneous Localization and Mapping), and industrial automation
- Scalable and cost-effective when using high-resolution CMOS or CCD sensors

Challenges:

- Performance is highly dependent on image texture; uniform or repetitive surfaces lead to poor disparity matching
- Requires significant computational resources for

*Figure 5: An illustration of dToF*

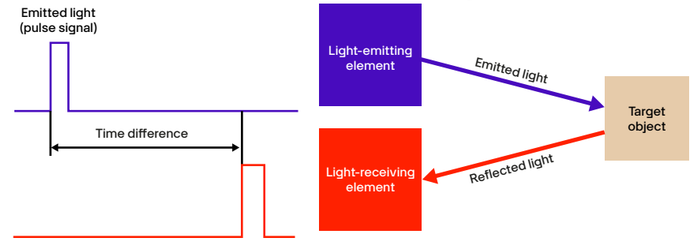
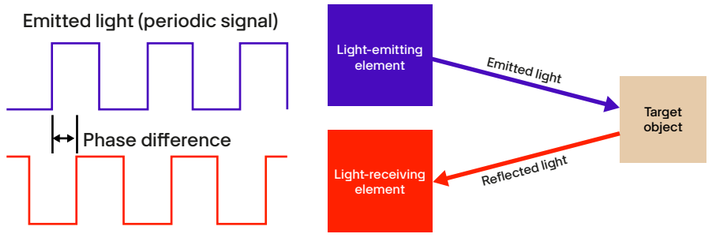


*Figure 6: An illustration of iToF*




real-time depth estimation, particularly in high-resolution applications
- Occlusion and edge bleeding effects introduce depth inaccuracies

## Time-of-Flight (ToF) sensors

Time-of-Flight (ToF) depth sensing is based on measuring the time delay or phase shift of emitted infrared (IR) light as it reflects off a target and returns to the sensor. This enables accurate distance measurements, generating real-time depth maps with minimal computational complexity. ToF systems are implemented in two main architectures:

Direct Time-of-Flight (dToF) measures the absolute travel time of individual photons using Single-Photon Avalanche Diodes (SPADs) or Silicon Photomultipliers (SiPMs). This method is well-suited for applications requiring long-range, high-precision depth measurements, such as automotive LiDAR and industrial metrology.

Indirect Time-of-Flight (iToF) emits modulated infrared signals and measures the phase shift between emitted and received light using CMOS-based image sensors with demodulation pixels. iToF is commonly used in consumer electronics and AR/VR applications due to its compact design and lower power consumption.

ToF sensor performance can be influenced through several system-level design factors to improve precision, coverage, or varying lighting conditions. This can be achieved by altering modulation frequencies/wavelengths, emitter technology, optical designs, or controlling calibration and exposure.

**The choice of technology depends on environmental constraints, computational resources, and the precision required for machine vision applications.**

Technical advantages:
- Provides dense, real-time depth maps with minimal computational overhead
- Effective in low-light and textureless environments
- Well-suited for gesture recognition, AR/VR applications, and industrial process monitoring

Challenges:
- Depth precision decreases with distance due to multi-path interference
- Susceptible to errors in highly reflective or absorptive surfaces
- Requires precise calibration of illumination and sensor exposure times to avoid artifacts

### LiDAR (Light Detection and Ranging)

LiDAR uses pulsed laser beams to generate high-resolution 3D point clouds of an environment. By measuring the time delay between pulse emission and reception, LiDAR systems construct depth maps with sub-centimeter accuracy. LiDAR can operate in 1D (single-line scanning), 2D (rotating plane scanning), or 3D (solid-state or MEMS-based scanning).

LiDAR calculates distance, d, via the following formula:

$$d = \frac{c \cdot t}{2}$$

Here, c is the speed of light, t is the total time for the laser pulse to travel to the object and back, and the division by 2 accounts for the round trip.

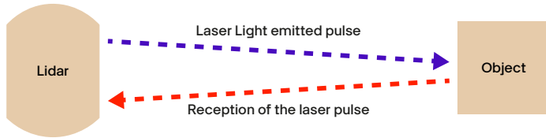



*Figure 7: Basic LiDAR principle visualized*

LiDAR systems commonly operate at 905nm or 1,550nm. The 905nm wavelength is cost-effective but has a shorter range and is subject to eye safety limitations. In contrast, 1,550nm LiDAR is safer for higher power emissions and can achieve longer ranges but requires more expensive components such as InGaAs detectors.

Various beam steering methods can be used to direct the laser beam for different outcomes or use cases. These include:
- Mechanical scanning: uses rotating mirrors to direct the laser beam over the field of view, common in traditional automotive and mapping LiDAR
- MEMS-based scanning: employs micro-electromechanical mirrors to reduce size and cost while maintaining scanning capability
- Solid-state (Flash LiDAR): illuminates the entire scene at once, eliminating the need for moving parts and enhancing robustness
- Optical Phased Arrays (OPAs): utilizes interference patterns to steer beams electronically, enabling ultra-compact, solid-state implementations without mechanical components

LiDAR systems also rely on advanced receiver technologies to capture returning laser pulses accurately and generate high-fidelity depth maps. For instance, avalanche photodiodes (APDs) provide high sensitivity for detecting weak return signals but introduce additional noise, requiring sophisticated filtering and amplification techniques. Single-photon avalanche diodes (SPADs) on the other hand enabled time-correlated single-photon counting, offering extreme sensitivity and superior performance in low-light conditions.

The effectiveness of these receivers directly impacts point cloud density and resolution, which define the granularity of the captured 3D environment. The angular resolution, which is typically between 0.1 and 1 degree, determines the level of detail in the scan and affects object detection accuracy. Scan rates, ranging from 10 to 100Hz, influence real-time application viability, with higher rates improving responsiveness in dynamic environments. High-end LiDAR systems can generate millions of points per second, allowing for precise environmental

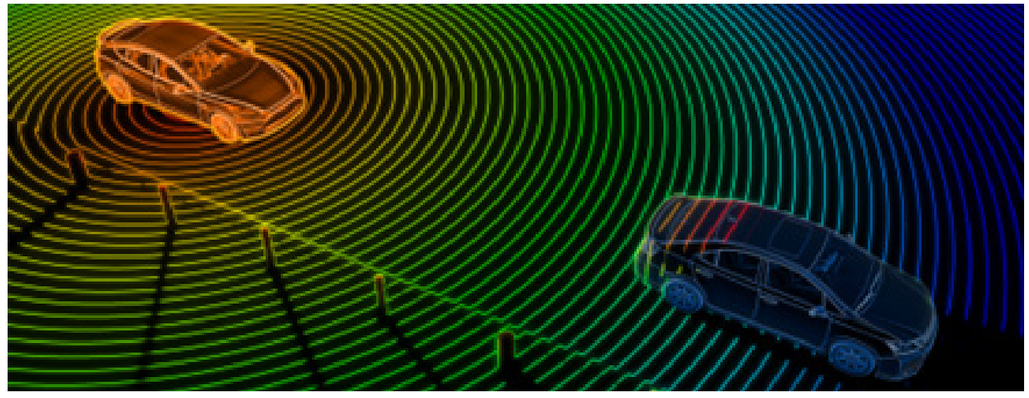


*Figure 8: Visualization of how LiDAR works on a vehicle for object detection and recognition.*

reconstruction which is crucial for applications such as autonomous navigation, mapping, and industrial automation.

Technical advantages:

- Superior range (tens to hundreds of meters) and depth accuracy
- Robust performance in diverse lighting conditions, including direct sunlight
- Essential for applications like autonomous vehicle perception, aerial mapping, and infrastructure inspection

Challenges:

- High cost and power consumption compared to other depth sensing methods
- Requires substantial data processing for real-time operation, often necessitating hardware acceleration (e.g., FPGAs, GPUs)
- Performance can be affected by rain, fog, or highly specular surfaces

Each depth sensing topology offers distinct advantages and limitations depending on the application. Structured light excels in short-range precision, stereo vision provides passive depth estimation, ToF balances real-time performance with moderate range, and LiDAR leads in long-range accuracy. The choice of technology depends on environmental constraints, computational resources, and the precision required for machine vision applications.

**Structured light excels in short-range precision, stereo vision provides passive depth estimation, ToF balances real-time performance with moderate range, and LiDAR leads in long-range accuracy.**

## Orbbec cameras implementing mainstream 3D technologies

- Monocular Structured Light: Astra Mini Pro, Astra 2
- Stereo vision structured light camera: Gemini 330 series, Gemini 2 series (Gemini 2, Gemini 2 L, Gemini 2 XL)
- TOF: Femto series (Femto Bolt, Femto Mega, Femto Mega I)

Orbbec Gemini 330 series stereo vision 3D cameras integrate two infrared imaging modules, a laser diode module (LDM) for infrared speckle pattern projection, an RGB imaging module, a depth engine processor (MX6800), an image signal processor (ISP), and an inertial measurement unit (IMU). The LDM emits infrared speckle patterns onto the target scene, while the dual infrared imaging modules capture synchronized images from distinct viewpoints. The depth engine processes these images using advanced depth reconstruction algorithms to generate a high-precision depth map of the scene.

Typical applications:

- Preferred cameras for robotic arm applications
  1. Stereo vision cameras
     - Objects in motion, require good accuracy at short distances, Varied/un-controlled lighting conditions, Multiple cameras with shared FoV
     - Example: bin-picking
  2. ToF cameras
     - Stationary objects, require high-fidelity edge definition, Require good accuracy at medium-to-long distances, Controlled lighting conditions
     - Example: palletization/de-palletization
  3. Structured light cameras
     - Stationary objects, require highest accuracy at short-to-medium distances, controlled lighting conditions
     - Example: defect detection
- Preferred depth cameras for AMR applications
  1. Stereo vision cameras
     - Require stable depth maps while in motion, operate in varied lighting conditions, often require multiple cameras for 360° view, multi-camera interference
     - Example: pallet/tote-moving, forklifts, cleaning, delivery
- Preferred depth cameras for humanoids
  1. Stereo vision cameras
     - Stable depth perception, adaptability to varied lighting conditions, multi-camera coordination and 360° view, low interference and high compatibility
  2. ToF cameras
     - High precision depth sensing, real-time performance, adaptability to complex lighting, multi-target detection and tracking
- Others
  1. Structured light cameras:
     - Facial recognition (e.g. facial payment kiosks)
     - 3D scanning (e.g. body part scanning, object scanning)

## Intel RealSense depth and tracking cameras

Intel RealSense cameras have become a go-to product within the machine and computer vision space, offering engineers high-precision depth perception and positional tracking. These solutions are widely adopted in robotics, autonomous navigation, industrial automation, medical imaging, and augmented/virtual reality (AR/VR) due to their compact form factor, real-time processing capabilities, and extensive software support.

*Figure 9: Intel RealSense D435f depth camera Credit: Intel Corporation*

Intel RealSense Depth Cameras leverage a combination of stereo vision, active infrared (IR) projection, and structured light to compute high-fidelity depth maps in real time. The stereo IR cameras capture left and right image pairs, and an onboard Intel Depth Sensing ASIC computes depth using disparity matching algorithms. The active IR pattern projector improves accuracy in low-texture environments by adding depth cues where natural visual features are sparse.

Intel's D400 series are a popular choice, for example:

- D415 – features a rolling shutter with a narrow field of view, suited for applications requiring precise object scanning
- D421 - module brings advanced depth-sensing technology to a wider audience at an affordable price point
- D435/D435i – utilizes a global shutter, making it ideal for fast-moving objects in robotics and automation
- D455 – offers an extended baseline (95mm), improving depth accuracy for mid-range applications.

These cameras see use anywhere from autonomous mobile robots (AMRs) and drones, 3D scanning and volumetric measurements, medical imaging, or even biometric security measures.

## Interfaces in computer and machine vision

The efficiency and scalability of machine vision systems depend on the interfaces used to transfer images data between cameras, processing units, and control systems. Different applications require specific connectivity solutions, balancing bandwidth, latency, power efficiency, and environmental robustness. The most widely used interfaces in machine/computer vision are GMSL/FAKRA, USB/MIPI CSI, Ethernet, and Power over Ethernet (PoE).

### GMSL/FAKRA

Gigabit Multimedia Serial Link (GMSL) is a high-speed serial interface designed for automotive and industrial vision applications. It supports long-distance, high-bandwidth video transmission with low latency, making it ideal for autonomous vehicles, ADAS (Advanced Driver Assistance Systems), and robotic vision. GMSL operates over FAKRA connectors, which provide rugged, shielded connections suitable for harsh environments.

GMSL can achieve data rates of up to 6Gbps per link while maintaining low latency, ensuring

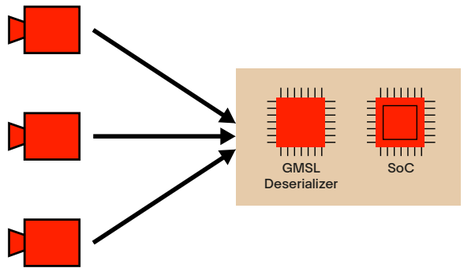
minimal delays in time-sensitive applications such as autonomous navigation. Forward error correction (FEC) mechanisms help mitigate signal degradation over long distances, making it highly reliable in electrically noisy environments. Unlike Ethernet-based solutions, GMSL is optimized for uncompressed, real-time data streaming, reducing processing overhead on receiving hardware.

### USB (USB3 Vision and MIPI CSI)

USB remains a dominant interface for machine vision, particularly in research, laboratory automation, and consumer applications. USB3 Vision, based on USB 3.0, provides high-speed data transfer up to 5Gbps (with USB 3.1 supporting up to 10Gbps), making it well-suited for high-resolution cameras requiring minimal latency. The plug-and-play nature of USB simplifies deployment in industrial inspection, microscopy, and AI-driven vision systems. However, USB's cable length limitation (typically under five meters) can be a constraint in larger-scale deployments.

For embedded vision applications, MIPI CSI (Camera Serial Interface) is the preferred standard, enabling direct connection between cameras and system-on-chip (SoC) processors. MIPI CSI supports scalable data rates, typically up to 2.5Gbps per lane, with multiple lanes available for increased bandwidth. It is common in smartphones, drones, and edge AI devices due to its low power consumption and efficient data transfer. Unlike USB, MIPI CSI is optimized for continuous, high-speed image capture without requiring a host controller.

*Figure 10: Typical GMSL cameras to host connection*




### Ethernet and Power over Ethernet (PoE)

Ethernet is a key interface for networked and industrial vision systems, offering long-distance, high-bandwidth connectivity. GigE Vision (Gigabit Ethernet Vision) is an industry standard that enables cameras to transmit uncompressed images over Ethernet networks at speeds of up to 1Gbps, with 10GigE Vision extending this to 10Gbps. Unlike USB, Ethernet allows for cable lengths of up to 100 meters, making it ideal for factory automation, security, and remote monitoring applications.

Power over Ethernet (PoE) further enhances Ethernet-based vision systems by delivering both power and data over a single cable, reducing cabling complexity. Standard PoE (IEEE 802.3af) provides up to 15.4W, while PoE+ (IEEE 802.3at) supports up to 25.5W, allowing for more powerful image sensors and onboard processing. However, power constraints can limit camera selection, particularly in applications requiring intensive onboard computing.

## Bandwidth and latency considerations

Each interface has trade-offs in terms of bandwidth, latency, and real-time performance:

- GMSL: up to 6Gbps per link, ultra-low latency, ideal for real-time applications
- USB3 Vision: 5Gbps (USB 3.0), 10Gbps (USB 3.1), moderate latency due to host processing
- MIPI CSI: 2.5Gbps per lane, very low power, efficient for embedded systems
- GigE Vision: 1Gbps (standard), 10Gbps (10GigE Vision), higher latency but long-distance support
- PoE: offers flexibility, but power constraints impact camera capability

## Synchronization and error handling

In industrial automation and robotics, precise synchronization of multiple cameras is critical. Ethernet-based vision systems support Precision Time Protocol (PTP) to enable hardware-level synchronization. GMSL and MIPI CSI provide deterministic data transfer, ensuring consistent frame timing. Error correction techniques, such as checksums in Ethernet and FEC in GMSL, enhance data integrity across long transmission distances.

The choice of interface depends on the application's requirements:

- Automotive: GMSL remains dominant, but automotive Ethernet is emerging for sensor fusion
- Factory automation: GigE Vision and PoE cameras are widely used for scalability and ease of deployment
- Embedded AI: MIPI CSI is preferred in Edge devices where power efficiency and direct SoC integration are priorities
- High-speed imaging: USB3 Vision and 10GigE Vision are favored for high-resolution, high-frame-rate cameras

Selecting the right machine vision interface involves balancing factors such as bandwidth, latency, cable length, power requirements, and environmental conditions. GMSL excels in real-time, high-speed applications, while Ethernet-based solutions provide flexibility and scalability. USB3 Vision offers simplicity and high bandwidth for close-range applications, whereas MIPI CSI is ideal for embedded systems. Understanding these trade-offs allows engineers to optimize vision systems for specific industrial and research applications.

**Machine and computer vision have evolved from early rule-based image processing to AI-powered, real-time decision-making systems that now underpin automation across industries.**

## Conclusion: the future of computer and machine vision

Machine and computer vision have evolved from early rule-based image processing to AI-powered, real-time decision-making systems that now underpin automation across industries. The integration of deep learning, Edge computing, and advanced sensing technologies has pushed machine vision beyond traditional applications, enabling more sophisticated perception and analysis at unprecedented speeds and accuracy.

Looking ahead, the next phase of computer and machine vision will focus on optimizing efficiency, adaptability, and real-time processing. Engineers will play a critical role in advancing neuromorphic vision systems, enhancing federated learning for distributed AI models, and developing low-power, high-performance vision architectures. Improving interoperability between vision systems and automation platforms will also be crucial, as industries demand more seamless integration between AI-driven vision and broader industrial ecosystems.

As the technology continues to mature, the engineering challenges will shift from feasibility to refinement – reducing computational overhead, improving adaptability in dynamic environments, and ensuring robust performance across diverse conditions. The future of machine vision is not just about seeing but understanding, adapting, and making intelligent decisions in real-time, paving the way for a new generation of autonomous and AI-driven systems.

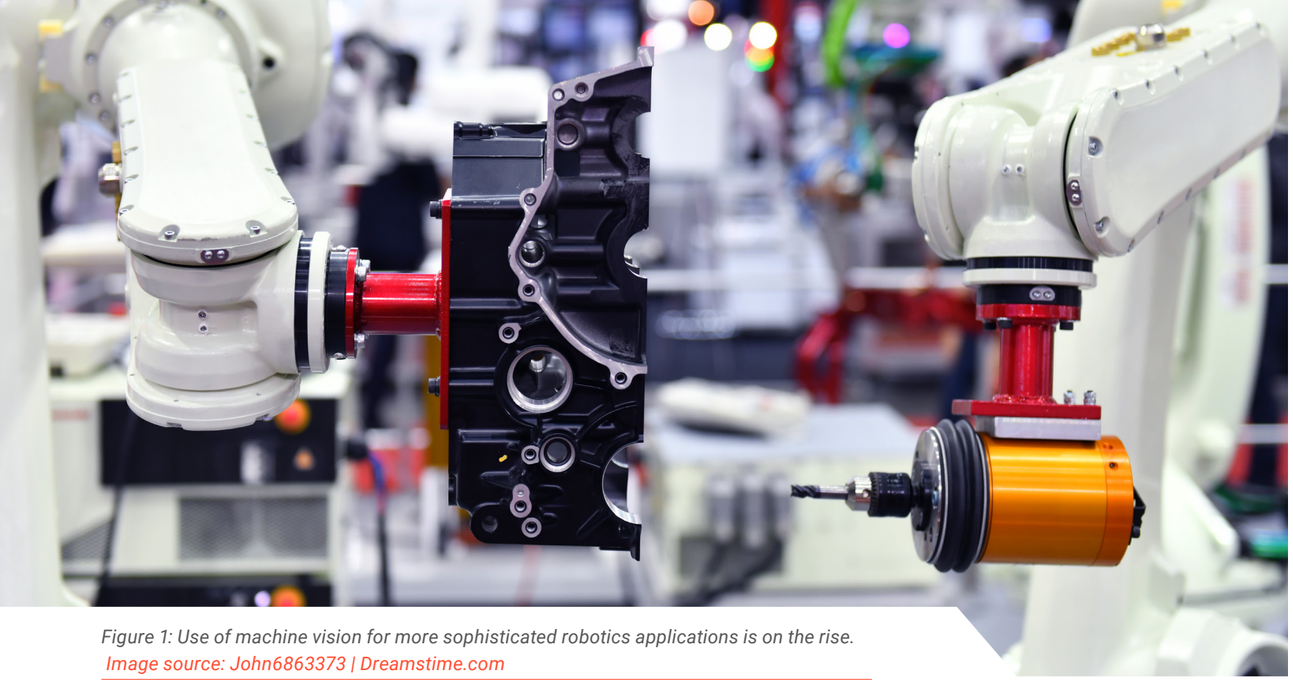


*Figure 1: Use of machine vision for more sophisticated robotics applications is on the rise. Image source: John6863373 | Dreamstime.com*

# How machine vision is advancing automation now

Written by Jody Muelaner

Machine vision is a collection of technologies that give automated equipment (industrial or otherwise) high-level understanding of the immediate environment from images. Without machine-vision software, digital images would be nothing more than simple unconnected pixel collections having various color values and tone intensities to such equipment. Machine vision lets computers (typically connected to machine controls) detect edges and shapes within such images to in turn let higher-level processing routines identify predefined objects of interest. Images in this sense aren't necessarily limited to photographic images in the visible spectrum; they can also include images obtained using infrared, laser, X-ray, and ultrasound signals.

One fairly common machine-vision application in industrial settings is to identify a specific part in a bin containing a randomly arranged (jumbled) mix of parts. Here, machine vision can help pick-and-place robots automatically pick up the right part. Of course, recognizing such parts with imaging feedback would be relatively straightforward if they were all neatly arranged and oriented the same way on a tray. However, robust machine vision algorithms can recognize objects at different distances from the camera (and therefore appearing as different sizes at the imaging sensor) as well as in different orientations.

The most sophisticated machine vision systems have enabled new and emerging designs far more sophisticated than bin picking – perhaps no more recognizable than in autonomous vehicles, for example.

## Technologies related to machine vision

The term machine vision is sometimes reserved to reference



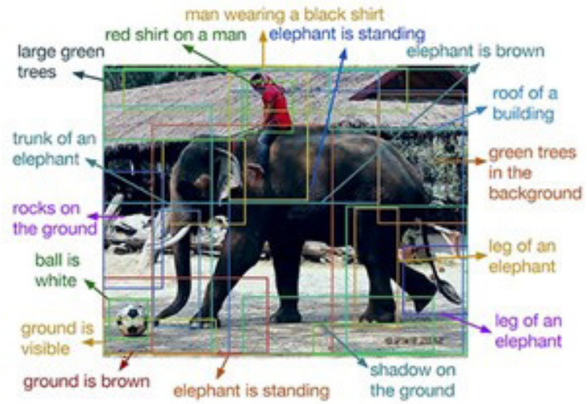



*Figure 2: Machine vision gives systems (industrial or otherwise) high-level understanding of an environment setting from images. Image source: Wikimedia*

more established and efficient mathematical methods of extracting information from images. In contrast, the term computer vision typically describes more modern and computationally demanding systems – including black-box approaches using machine learning or artificial intelligence (AI). However, machine vision can also serve as a catch-all term encompassing all methods of high-level information extraction from images; in this context, computer vision describes its underlying theories of operation.

Technologies to extract high-level meaning from images abound. Within the research community, such technologies are often considered as distinct from machine vision. However, in a practical sense, all are different ways of achieving machine vision … and in many cases, they overlap.

Digital image processing is a form of digital-signal processing involving image enhancement, restoration, encoding, and compression. Advantages over analog image processing include minimized noise and distortion as well as the availability of far more algorithms. One early image-enhancement use was correction of the first close-range images of the lunar surface. This used photogrammetric mapping as well as noise filters and corrections for geometric distortions arising from the imaging camera's alignment with the lunar surface.

Digital image enhancement often involves increasing contrast and may also make geometric corrections for viewing angle and lens distortion. Compression is typically achieved by approximating a complex signal to a combination of cosine functions – a type of Fourier transform known as a discrete cosine transform or DCT. The JPEG file format is the most popular application of DCT. Image restoration may also use Fourier transforms to remove noise and blurring.

Photogrammetry employs some kind of feature identification to extract measurements from images. These measurements can include 3D information when multiple images of the same scene have been obtained from different positions. The simplest photogrammetry systems measure the distance between two points in an image employing a scale. Including a known scale reference in the image is normally required for this purpose.

**Feature detection** lets computers identify edges and corners or points in an image. This is a required first step for photogrammetry as well as the identification of objects and motion. Blob detection can identify regions with edges that

*Figure 3: The DLPC350 integrated circuit (IC) controller provides input and output trigger signals for synchronizing displayed patterns with a camera. It works with digital micromirror devices (DMDs) designed to impart 3D machine vision to industrial, medical, and security equipment. In fact, applications include 3D scanning as well as metrology systems. Image source: Texas Instruments*

are too smooth for edge or corner detection.

**Pattern recognition** is used to identify specific objects. At its simplest, this might mean looking for a specific well-defined mechanical part on a conveyor.

**3D reconstruction** determines the 3D form of objects from 2D images. It can be achieved by photogrammetric methods in which the height of common features (identified in images from different observation points) are determined by triangulation. 3D reconstruction is also possible using a single 2D image; here, software interprets (among other things) the geometric relationships between edges or regions of shading.

A human can mentally reconstruct a cube from a simple line-art representation with ease – and a sphere from a shaded circle. Shading gives indication of the surfaces' slopes. However, the process of such deduction is more complicated than it seems because shading is a one-dimensional parameter while slope occurs in two dimensions. This can lead to ambiguities – a fact demonstrated by art depicting physically impossible objects.

## How machine-vision tasks are ordered

Many machine-vision systems progressively combine the above techniques by starting with low-level operations and then advancing one by one to higher-level operations. At the lowest level, all of an image's pixels are held as high-bandwidth data. Then each operation in the sequence identifies image features and represents information of interest with relatively small amounts of data.

The low-level operations of image enhancement and restoration come first, followed by feature detection. Where multiple sensors are used, low-level operations may therefore be carried out by distributed processes dedicated to individual sensors. Once features in individual images are detected, higher-level photogrammetric measurements can occur – as can any object identification or other tasks relying on the combined data from multiple images and sensors.


*Figure 4: 3D scanners capture 2D images of an object to create a 3D model of it. In some cases, the digital models are then employed to 3D print copies. Image source: Shenzhen Creality 3D Technology Co.*

## Direct computations and learning algorithms

A direct computation in the context of machine vision is a set of mathematical functions that are manually defined by a human programmer. These accept inputs such as image pixel values to yield outputs such as an object's edges' coordinates. In contrast, learning algorithms aren't directly written by humans but are instead trained via example datasets associating inputs with desired outputs. They, therefore, function as black boxes. Most all such machine learning now employs deep learning based on artificial neural networks to make its calculations.

Simple machine learning for industrial applications is often more reliable and less computationally demanding if based on direct computation. Of course, there are limits to what can be achieved with direct computation. For example, it could never hope to execute the advanced pattern recognition required to identify individuals by their faces, especially not from a video feed of a crowded public space. In contrast, machine learning deftly handles such applications. No wonder then that machine learning is increasingly being deployed for lower-level machine-vision operations including image enhancement, restoration, and feature detection.

*Figure 5: Computerized determination of a workpiece's 3D form from a 2D image is fraught with challenges.*

## Improving teaching approaches (not algorithms)

The maturing of deep-learning technology has made apparent that it's not learning algorithms themselves needing improvement but the way they're trained. One such improved training routine is called data-centric computer vision. Here, the deep-learning system accepts very large training sets made of thousands, millions, or even billions of images – and then stores resultant information its algorithms extract from each image. The algorithms effectively learn by practicing worked examples and then referring to an 'answer book' to verify whether they arrived at the right values.

An old story about the early days of digital pattern recognition serves as a cautionary tale. The U.S. military intended to use machine vision for target recognition, and defense-contractor demonstrations reliably identified U.S.-made and Russian-made tanks. Various tanks were all correctly differentiated from the supplier's aerial photographs, one after the other. But when tested again with the Pentagon's own library of pictures, the system kept giving wrong answers. The problem was that the defense contractor's images all depicted U.S. tanks in deserts and Russian tanks in green fields. Far from recognizing different tanks, the system was instead recognizing different-colored backgrounds. The moral? Learning algorithms need to be presented with carefully curated training data to be useful.

## Conclusion: vision for robotic workcell safety

Machine vision is no longer a niche technology. It's seeing the most increased deployment in industrial applications. Here, the most dramatic development is how machine vision now complements industrial-plant safety systems that sound alarms or issue audio announcements when plant personnel enter a working zone without a hard hat, mask, or other correct protective equipment. Machine vision can also complete systems that announce when mobile machinery such as forklifts get too close to people.

These and similar machine-vision systems can sometimes replace hard guarding around industrial robots to enable more efficient operations. They can also replace or enhance safety systems based on light guards that simply stop machinery if a plant worker enters a workcell. When machine vision monitors the factory floor surrounding the workcell, it is possible for robots in such cells to gradually slow down as people approach.

As the designs of industrial settings evolve to accommodate collaborative robots and other workcell equipment that are safe for plant personnel to move around (even while that equipment operates) these and other systems based on machine vision will become a much more common part of factory processes.

*Figure 6: Image sensors from the iVu series can identify workpieces by type, size, location, orientation, and coloring. The machine-vision components can accept configuration and monitoring an integrated screen, remote HMI, or PC. Camera, controller, lens, and light are all pre-integrated. Image source: Banner Engineering Corp.*

# What does passion, curiosity, and creativity have in common?

Answer: **you**

Whatever you call yourself (Maker, Student, Tinkerer, Hobbyist, Tech-Wizard...) you embody the spirit of invention–and that spirit is what creates a better world for us all.

**If you can dream it up, we'll help you build it at digikey.com/europe**

DigiKey

**we get technical**