Powered by IEEE

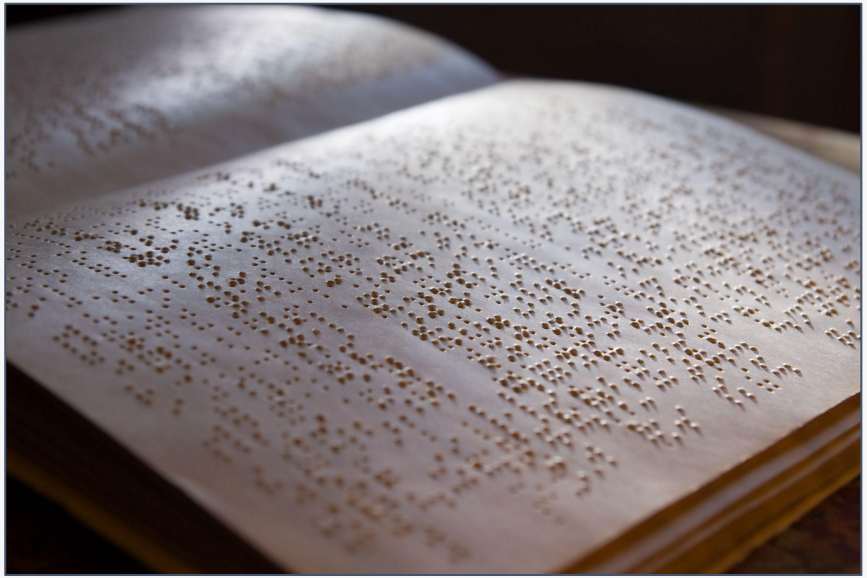
TRY Engineering

Bright Beats Challenge

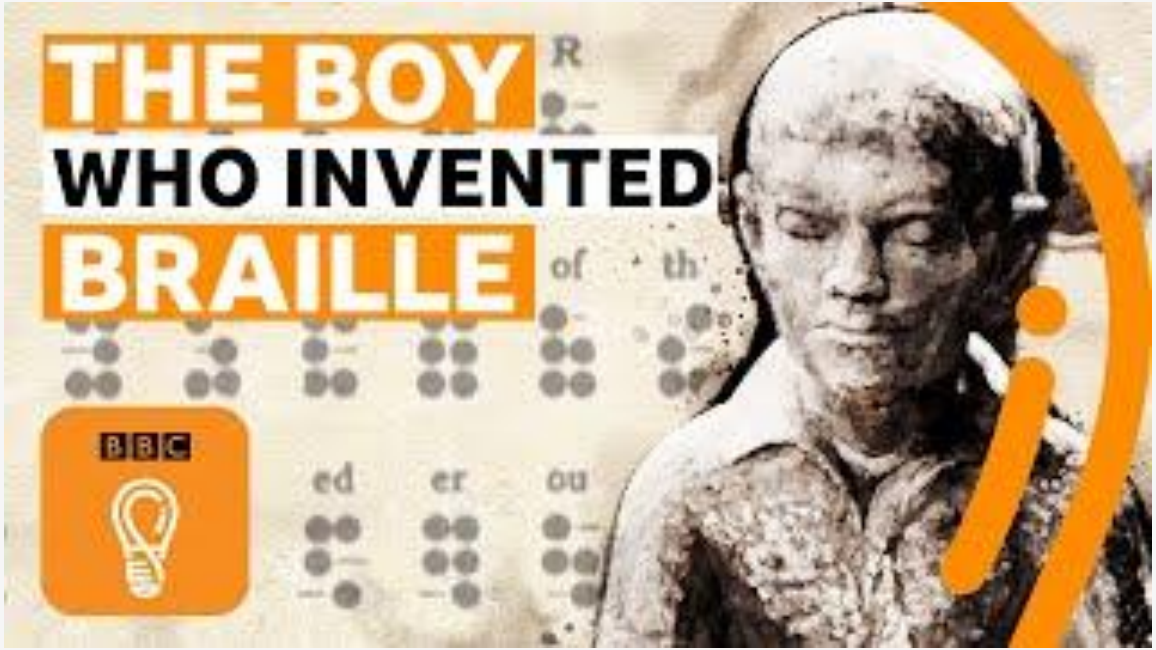
A MathWorks Lesson

IEEE

Do you know what this is?



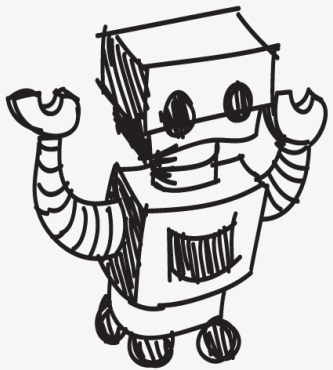
Watch: The Boy Who Invented Braille



Braille to Bright Beats Challenge

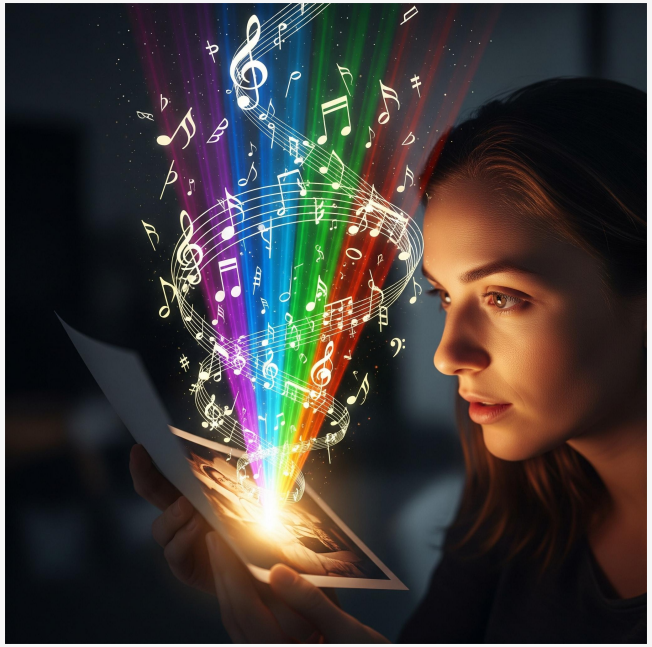
Similar to Braille, where visual information is translated into a tactile representation, in the Bright Beats Challenge, we will translate visual information into an audio representation.

The Design Challenge



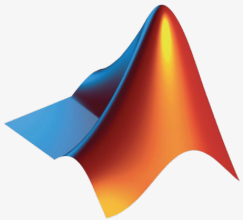
Bright Beats Challenge

You will create a **light-sensitive musical synthesizer** using a **micro:bit** to transform light levels into corresponding musical notes. By translating visual signals into audio signals, this device can allow users to experiences photos in a new and unique way.



MATLAB Simulation

In the accompanying **MATLAB** activity, you will turn pixels from digital images into piano notes. In the process, you'll uncover the power of computers and programs like MATLAB for transforming visual and audio signals through **signal processing** and learn how images are represented by computers. Building from the micro:bit activity, you'll be able to customize sounds further and translate more complex images into sounds.



MathWorks®

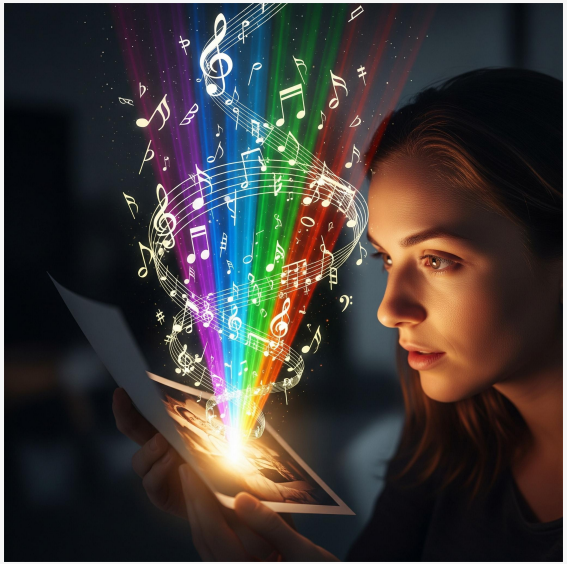
Criteria & Constraints

Criteria

- Light Detection: Use the micro:bit's light sensor to measure ambient brightness.
- Sound Mapping: Program musical notes that correspond to specific light levels.

Constraints:

- Must use only onboard micro:bit components (no external sensors or speakers).
- The system must operate continuously without manual resets.



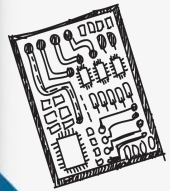
Materials

- Computer (1 per team of 2 kids)
- 1 micro:bit v2 (per team)
- MATLAB: matlab.mathworks.com
- MATLAB Activity Files and Instructor Guide
- Printed black and white image
- White paper, black marker, and colored markers
- Mini flashlight, head lamp, or cell phone light
- Cardboard or Cardstock
- Tape/Glue
- Rubber Bands
- Scissors
- Other fun decorative items for the device (pipe cleaners, etc)
- TryE Signal Processing Video
- Wave Wonders Ebook
- TryE Signal Pro

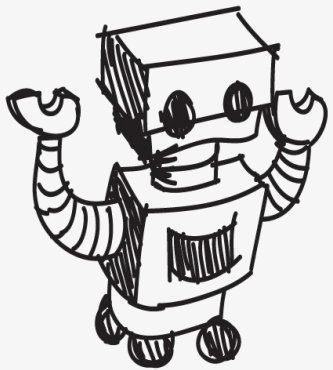
Failure is part of the process

If your code does not work, don't worry, we expect that. Engineers fail and they try again and again. So troubleshoot and then redesign until you get your best solution. Failure is part of the process! Have fun!

Ready, Set, Let's Engineer!



Prepare Your Image

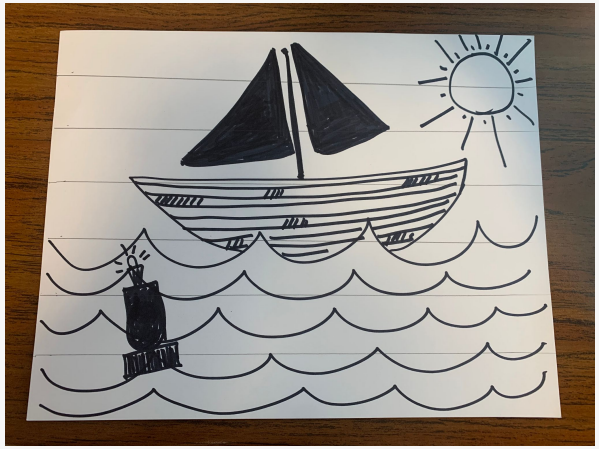


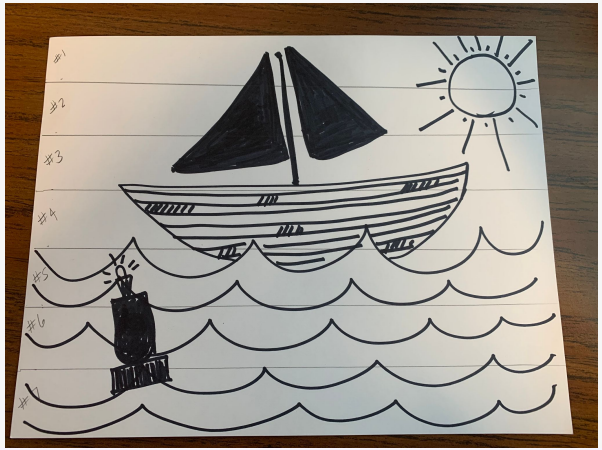
Prepare Your Image

Draw a bold, high-contrast image (black on white) or use a printed black-and-white image to cut into strips. The strips will be pulled across the micro:bit's light sensor to trigger different musical tones based on light intensity.

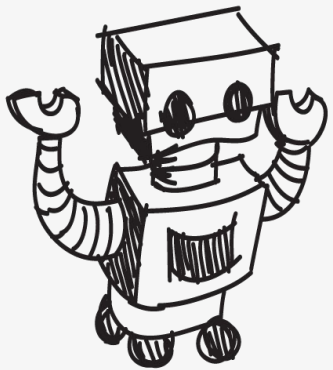
- **Choose a Black-and-White Image & Print OR Draw Your Image**
- **Cut Strips (vertical or horizontal) & Number them**

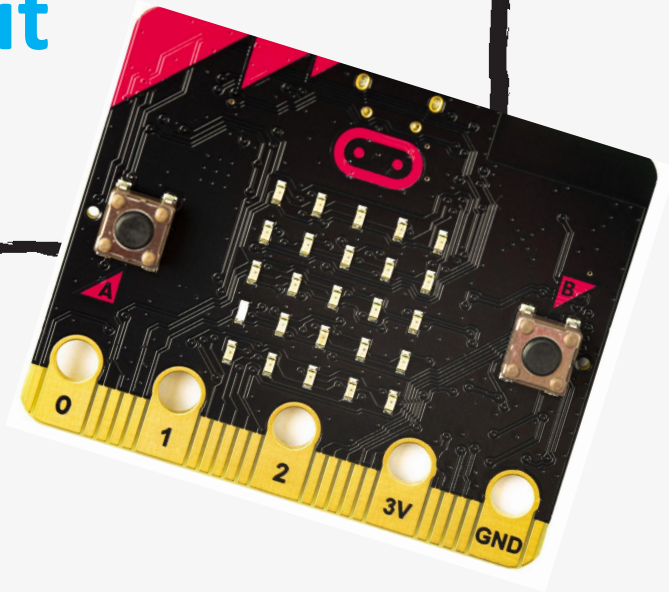
  Make sure it's sized to fit over the micro:bit's LED grid. Number them so you can "read" the image in the correct order.





micro:bit





micro:bit Pre-Activity

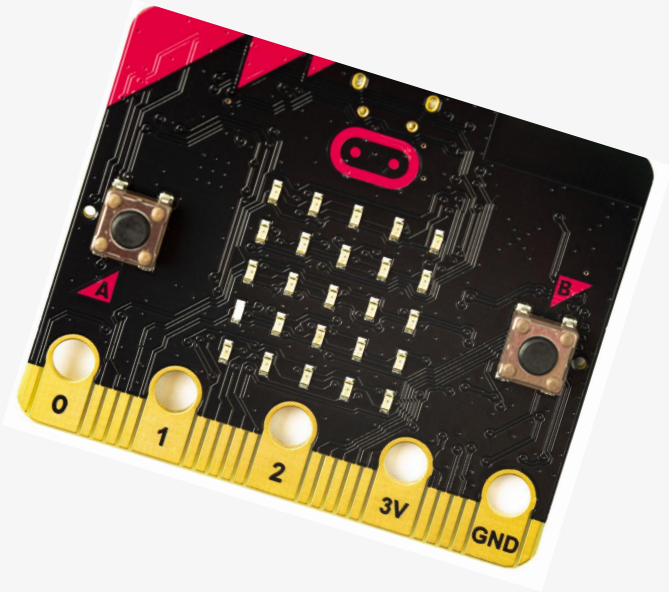
To prepare you for the design challenge with micro:bit, try this pre-activity in Microsoft MakeCode

- Write a program that shows the light level on the screen.
- Shine a flashlight on the micro:bit — watch the number go up!
- Cover it with your hand — the number goes down!
- Open the serial monitor to view the data.
- Observe how the light level changes in real time.

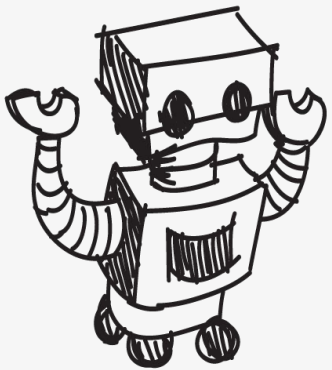
Program micro:bit to Solve the Challenge

Program the micro:bit in MakeCode to solve the design challenge:

- Initialize Variables
- Log Light Level to Serial Monitor
- Map Light Level to Sound Frequency
- Play the Sound



Need More Support?
See Sample Code



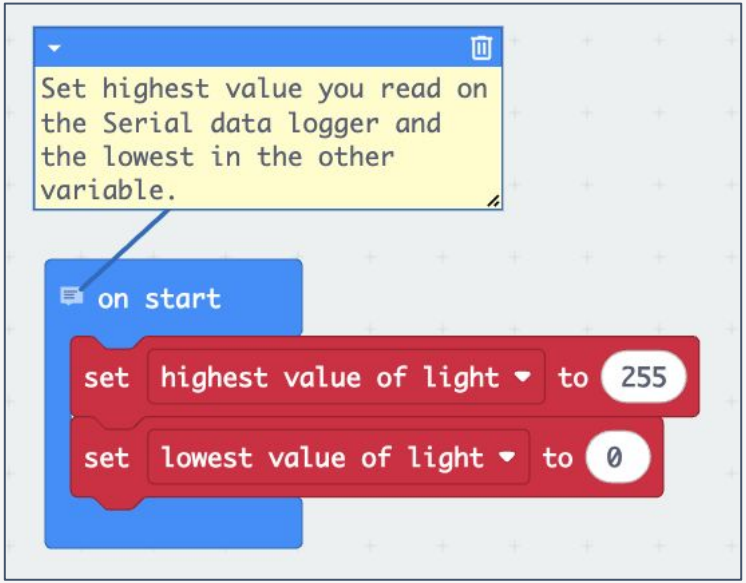
Sample Code

Initialize Variables

Go to the Variables category and:

- Create a variable called note
- Create a variable called brightness
- Create a variable called highest_value_of_light and set it to 255
- Create a variable called lowest_value_of_light and set it to 0

Use the “set [variable] to [value]” block to assign initial values.



Log Light Level to Serial Monitor

Go to Loops and drag a forever block.

Inside it:

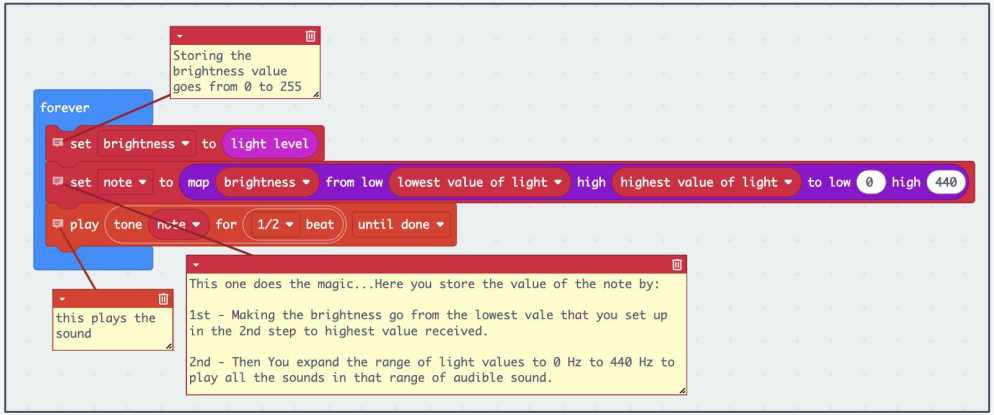
- Go to Input and drag light level block
- Go to Serial and use write number block to send the light level
- Add a serial write line block with an empty string to separate entries

This will continuously log the light level to the serial monitor.

Map Light Level to Sound Frequency

Add another forever block. Inside it:

- Set brightness to light level (from Input)
- Use the map block from Math to convert brightness to frequency:
  - Map brightness from lowest_value_of_light to highest_value_of_light
  - To range 0 to 440
  - Store result in note



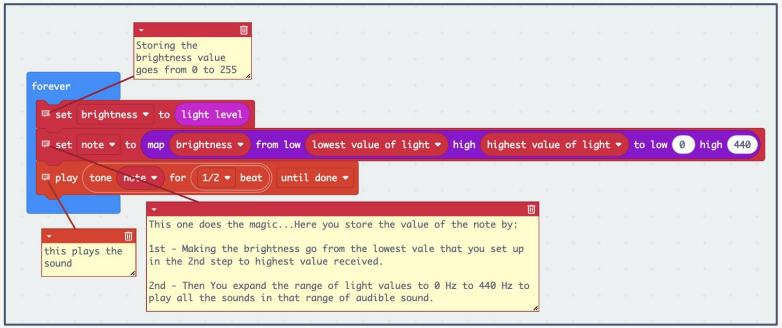
Powered by IEEE TRYEngineering | IEEE

Play Sound

Still inside the second forever block:

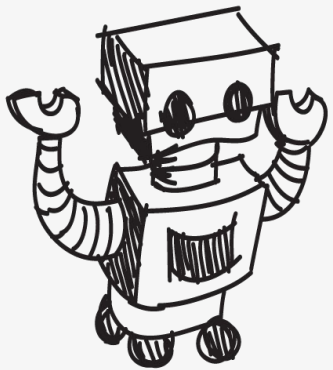
- Go to Music
- Use play tone block
- Replace the frequency with note
- Set duration to half beat

- Make sure both forever loops are running simultaneously
- Connect your micro:bit to your computer and open the Serial Monitor to view light levels
- Shine light or cover the sensor to hear pitch changes



Powered by IEEE TRYEngineering | IEEE

Build & Test the Light-Sensitive Musical Synthesizer

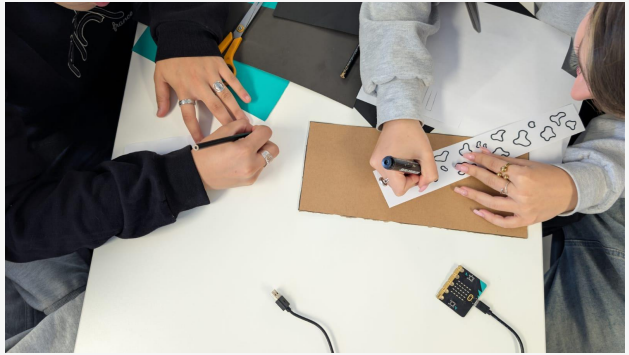


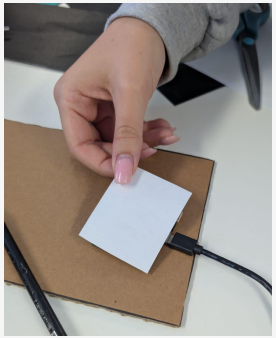
Build the Housing

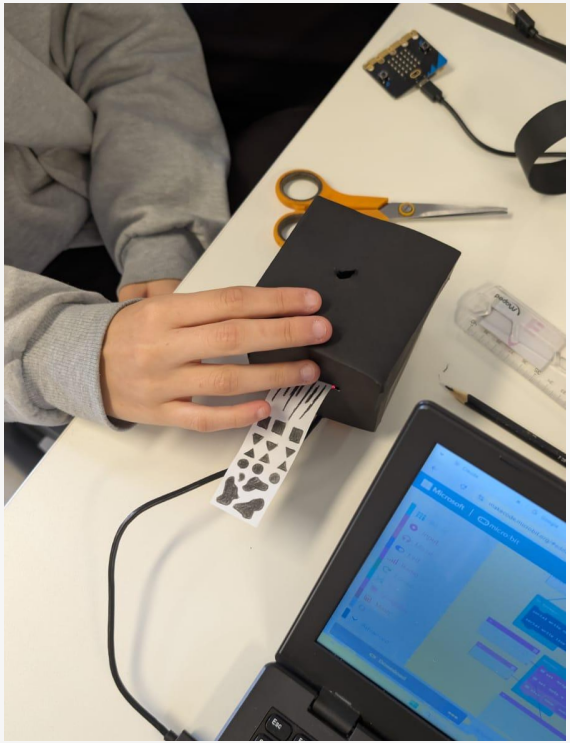
- **Create Box/Stand:** Using cardboard (or card stock), create a box or stand that can securely hold your micro:bit flat.
- **Mount Light:** Design a way to mount a mini-flashlight (head lamp or cell phone) directly above the micro:bit, pointing down. This ensures consistent lighting as the strip moves across the sensor.
- **Create Slit to Pull Strips:** Create a slit/opening in the box/stand that allows you to pull the image strips across the micro:bit's LED grid.
- **Mount the Micro:bit:** Place the micro:bit flat inside your prepared box or on the surface of your stand, ensuring its light sensor or photoresistor faces upwards towards where the image strip will pass.
- **Add a Light Source:** Mount the flashlight directly above the micro:bit, pointing down, to ensure consistent lighting.

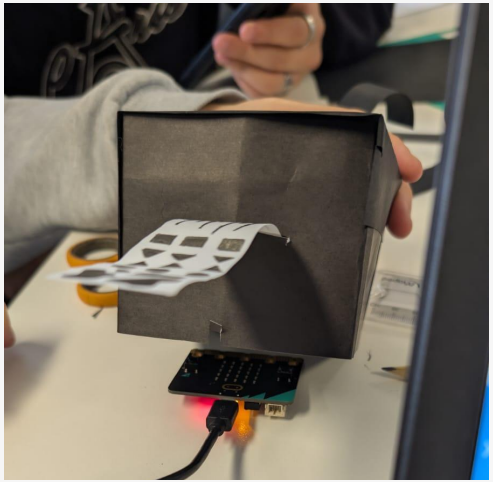
Test Device

- **Pull the Strips Across:** Slowly pull your image strip #1 across the micro:bit's LED grid, observing how the light and dark areas passing over the sensor trigger different musical tones. Continue to pull all the strip in the correct order to "read the image"
- **Listen to the Sound Changes:** The micro:bit will map the light levels to musical notes. Darker areas will produce lower tones, and lighter areas will produce higher tones — creating a unique sound pattern based on your image.
- **Experiment:** Try different images, patterns, or even draw your own with black markers. You can also reverse the strip or change the speed of movement to alter the sound.

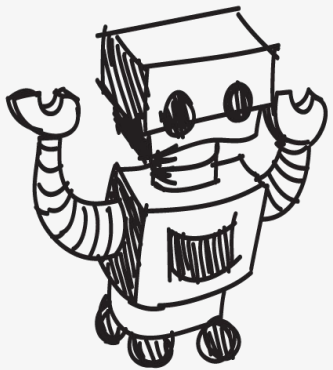








MATLAB Simulation



MATLAB Simulation

In the **MATLAB** activity, you'll turn pixels from digital images into sounds. Using the power of MATLAB, you'll learn how computers and programs like MATLAB represent and transform visual and audio signals through **signal processing**.

With the micro:bit device, you were limited to simple black and white images. With MATLAB, you'll be able to translate complex digital images into a wider variety of sounds.

- MATLAB Activity Files - click ' Open in MATLAB Online' and follow the step-by-step instructions in 'BrightBeatsChallenge_Student.mlx'
- PDF of Instructor Guide

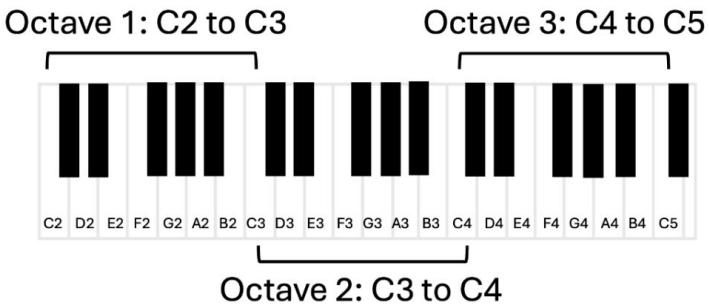
Powered by IEEE TRYEngineering | IEEE

Get Started

First, please choose a range of piano notes to use for this activity. Choose an octave from the drop down menu *or* use the default notes. Click 'Use default notes' to use them.

Choose an octave: Octave 2

Use default notes



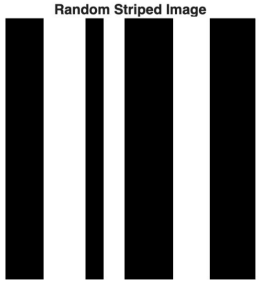
Play notes

Part 1: Translate stripes to sounds

Your micro:bit device 'translated' a black and white image to sounds, with different sounds representing black or white parts of the image. You can do the same thing in MATLAB.

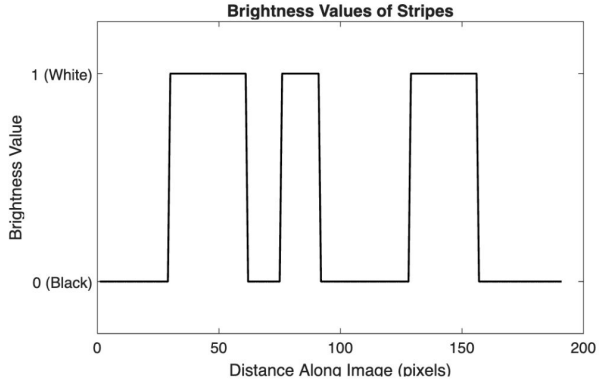
First, let's generate a random image of black and white stripes.

Generate random striped image



Next, let's use MATLAB to measure which parts of the image are black or white by measuring the brightness of the stripes going from left to right. For a black and white image, the brightness will either be 0 (black) or 1 (white).

Plot image brightness

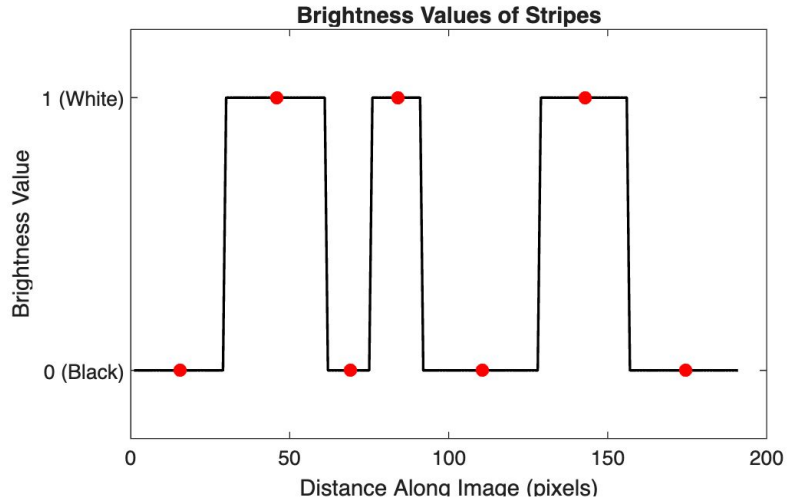


Try. Do you notice how the plot of brightness values looks like a bit like the image itself? The brightness values vary from 0 (black stripe) to 1 (white stripe) and follow the pattern of the stripes! Try generating a few more different stripe images and their corresponding brightness value plots.

Try: Generate few random striped images, and make note of how the brightness value plots change corresponding to the stripe patterns.

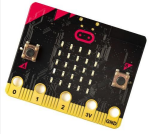
Now, we're ready to translate these brightness values into sounds. We'll use MATLAB to play piano notes that correspond to the brightness of the stripes. We want the black stripes to be played as the lowest piano note in the octave, and the white stripes to be played as the highest note. The length that each sound is played corresponds to the width of the stripe.

Play Sounds



Challenge: Close your eyes and listen to the sounds again. Can you visualize the striped image just by listening to the sounds?

- How many stripes are there in total?
- How many black stripes are there? How many white?
- Can you tell which stripes are wide and which are thin?



Challenge: Grab a blank strip of paper and draw the striped image we generated with MATLAB. Copy it as closely as you can. Pull this image strip across the micro:bit device you built earlier. What are some differences you notice between how the micro:bit translates the stripes into sounds compared to MATLAB? Is one translation more accurate than the other?

Part 2: Translate a grayscale image to sounds

The micro:bit device you built earlier can only read black and white images. That's because the light sensor in the micro:bit can only detect big changes in ambient light (like going between black and white). But MATLAB is more sensitive than the micro:bit, and can 'read' the image in more detail.

When you open a digital picture on a computer, each individual point on an image is called a ***pixel***. Every pixel has its own brightness value. A digital picture is made up of rows of pixels. Just like you can cut a physical image into strips, you can divide a digital image into rows using MATLAB. And MATLAB can 'read' the brightness value of each individual pixel.

Let's use MATLAB to 'read' the brightness of the pixels along any row of an image. First, let's get an image to work with. You can either upload an image from your computer, load one of the example images, or take a picture with your webcam.

Use this button to upload an image:

Upload a picture from your computer

Or, load one of the example images:

Choose an example image cat1

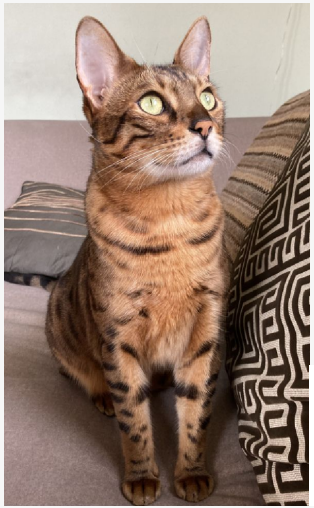
Load image

Or, take a picture with your webcam. A camera preview will first pop up and it will stay up for 5 seconds while you prepare your best pose!

Take a picture with your webcam

For this activity, you can:

1. Upload a picture *or*
2. Load an example image *or*
3. Take a picture with your webcam



One of the example images: ‘cat1’

Let's convert your image to grayscale. Click the button below:

Convert image to grayscale

Your image has 741 rows.

Next, pick a row to get the brightness values of pixels in that row. You must pick a row number less than the total number of rows in the image.

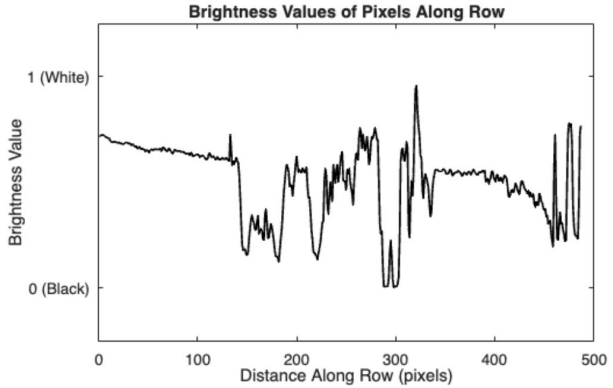
Imagine that picking a row is like cutting out a specific strip of the image. A red line will appear on the image to show which row you selected.

Pick a row: 191

Reflect: Try to match the brightness values to the image row you selected. Does the plot make sense? How well do the brightness values track the dark and bright pixels along the image row?

Now, let's the plot the brightness values of the pixels along the row you selected.

Plot brightness values



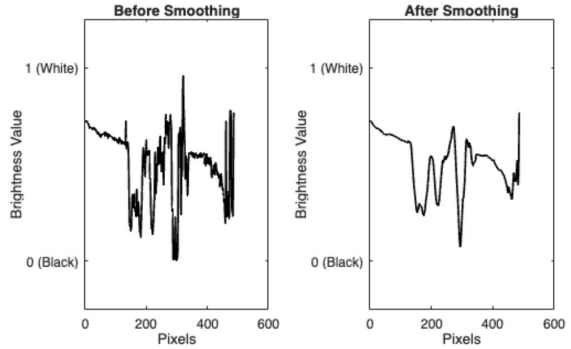
You might have noticed that the plot of the brightness values is bumpy, or jagged. Just like you can apply a filter to an image to blur sharp edges, we can also smooth out the brightness signal. ***Smoothing*** is a common step when scientists and engineers are trying to understand different signals (like brightness or sound signals). It helps to:

- **Make patterns clearer**: If you're trying to see the overall trend, like whether the brightness is generally increasing or decreasing, smoothing can help you ignore tiny random changes.
- **Reduces noise**: Smoothing removes random little wiggles that aren't important (usually called *noise*), so you can focus on the real signal.
- **Easier to analyze**: A smooth line is simpler to work with if you're using the signal to try and make predictions or decisions.

Think of it like brushing messy hair. The hair is still there, but now it's neat and easier to see the shape! An engineer designing a heart-rate monitor might smooth the raw sensor data so that sudden spikes caused by movement don't confuse the system when detecting the actual heartbeat pattern.

Try it out here:

Choose how much smoothing to apply 20



Smoothing: A common step when scientists and engineers are trying to understand different signals (like brightness or sound signals). It helps to make patterns clearer, reduce noise, and make signals easier to analyze.

Try applying different amounts of smoothing. Do you think the smoothed signal provides a more useful representation of the brightness changes in the image? Is there such a thing as too much smoothing?

Let's use MATLAB to translate the smoothed brightness signal into sounds. You can choose to have one note played for each pixel. Or, you can choose to sample a smaller number of pixels and have a note played for every other, every 5th, every 10th, every 50th, every 100th pixel, and so on. This is called the ***sampling period***.

When scientists and engineers analyze signals, it's important to choose the right sampling period. Sampling is like taking snapshots of the signal. If the sampling period is small, we take snapshots of the signal very often. If the sampling period is large, we wait longer between snapshots.

Why does sampling period matter?

- Short period (lots of snapshots) is great for accuracy and detail, but can take up a lot of space on the computer
- Long period (fewer snapshots) can save space and power, but might miss out on details

Sometimes less is more! Imagine you're filming a turtle walking.

- If you take a picture every second (short period), you'll have way too many pictures that look almost the same
- If you take a picture every five seconds (long period), you'll still see the turtle moving without wasting computer space

First, let's ask MATLAB to count how many pixels are in the image row.

Count pixels

```
487 pixels
```

Sampling Period: When scientists and engineers analyze signals, it's important to choose the right sampling period. Sampling is like taking snapshots of the signal.

If the sampling period is small, we take snapshots of the signal very often. If the sampling period is large, we wait longer between snapshots.

Choosing the right sampling period can be tricky: a shorter period (lots of snapshots) is great for detail, but can take up a lot of space while a longer period (fewer snapshots) can save space, but miss out on details.

If we played every single pixel as a sound lasting half a second, how long would it take to play every pixel in the row? Check your answer in MATLAB!

IEEE

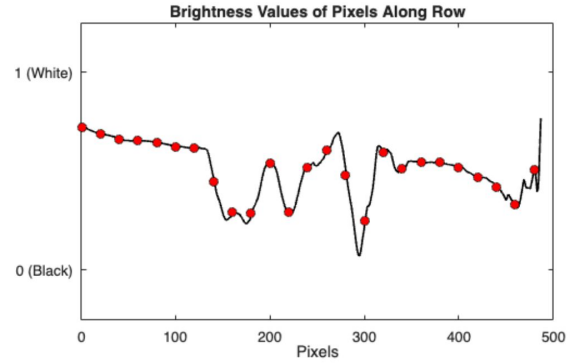
Try. Now, try using different sampling periods or use the default sampling period. Click 'Use default sampling period' to use the default. To stop the sounds before they're done playing or to change the sampling period, first hit the 'Stop' button.

Choose a sampling period: Every 20th pixel

Submitted! Thank you for selecting a sampling period. Scroll down to continue.

Use default sampling period

Play sounds



Reflect. Is there an 'ideal' sampling period that saves time while still accurately representing the changing brightness values in the image?

Challenge: Close your eyes and listen carefully to the sounds. Can you visualize the changes in brightness in the image? How does choosing different sampling periods affect how you visualize the brightness changes in the image?

Reflect: Is there an ‘ideal’ sampling period that saves time while still accurately representing the changing brightness values in the image?

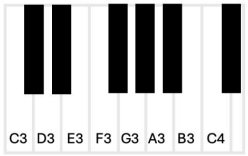
Challenge: Close your eyes and listen to the sounds again. Can you visualize the changes in brightness in the image? How does choosing different sampling periods affect how you visualize the brightness changes in the image?

Part 3: Translate colors to sounds

The world around us isn't just in black and white or shades of gray. For a visually impaired person or someone who is colorblind, think about how difficult it could be to choose between a red or green apple or choosing what color clothes to wear.

The micro:bit light sensor can't detect different colors. It can only detect big changes in light (like going from bright white to dark black). But MATLAB can 'read' the colors of pixels in an image. Instead of translating pixel brightness into sounds, let's translate pixel colors into sounds.

First, choose which piano notes you want to represent different colors of the rainbow. Either use the drop down menus to pick notes to represent each color *or* use the default notes. Click 'Use default notes' to use them. Remember to pick **different** notes to represent each color.



Use default notes

Choose a note for red C3 Play sound for red

Choose a note for orange D3 Play sound for orange

Choose a note for yellow E3 Play sound for yellow

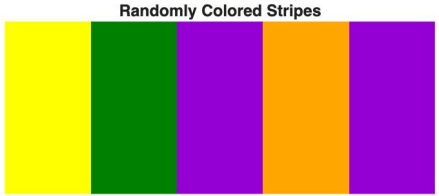
Choose a note for green F3 Play sound for green

Choose a note for blue G3 Play sound for blue

Choose a note for purple A3 Play sound for purple

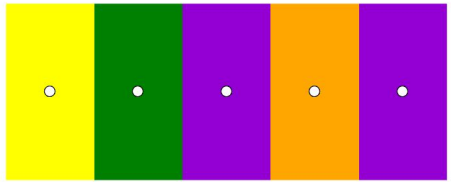
Next, let's generate a random image of colored stripes.

Generate random color stripes



Now, play the colors as sounds!

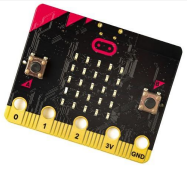
Play sounds



Try generating a few more colored stripe images and playing the sounds. Practice memorizing which sound represents each color.

Challenge: Close your eyes and listen to the sounds again. Can you visualize the colors of each stripe?

- Pair up and have your partner generate a new colored stripe image and play the sounds. Guess what color each stripe is based on the sounds. Have your partner check your answers. Compete to see who can get the most correct.



Challenge: Grab a blank strip of paper and colored markers or pencils. Draw the colored striped image generated with MATLAB. Copy it as closely as you can. Pull the strip across the micro:bit device you built earlier. Can the micro:bit light sensor detect the different colors and play different sounds for each color?

- Close your eyes and listen to the sounds as you pull the strip through your micro:bit device. Can you guess the color of the stripes based on the sounds?

Real-World Application

Think about how a device that plays different sounds based on an object's color could help someone who is visually impaired or colorblind. For example, what if someone who was visually impaired wanted to know if they had any blue colored pencils? Or, pick out the orange peppers from a display at the grocery store? Load one of the real-world examples to simulate how such a device would function.

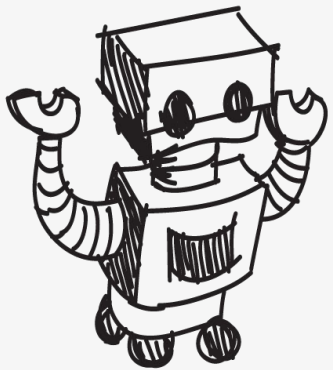
Choose a real-world image: Pencils

Load image

Play sounds

In MATLAB, we can simulate and test how our device functions using digital images. We can refine the sounds and make sure the device is functioning correctly. Then, when we are satisfied with the testing, we could download the code we created to a smartphone or a smart watch, and use our device in the real world. The beauty of simulating the device's function in MATLAB first is that we can troubleshoot it before we launch it on a device. Simulations like these can save scientists and engineers precious resources like time and supplies.

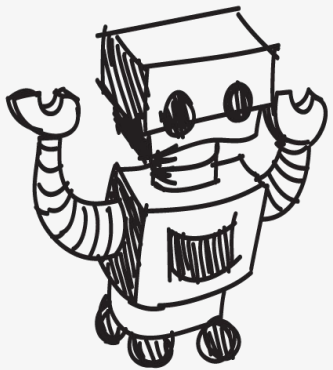
Connection to Signal Processing



Signal Processing Connection

- **Signal Processing:** The challenge uses signal processing by converting light signals from an image into audio signals (musical notes).
- **Image Processing:** "Reading" the black-and-white image with the micro:bit is similar to how computers process images, by interpreting light and dark areas as data. The MATLAB simulation further explores this by showing how computers represent images as grids of pixels.
- **Translating Data:** Both the design challenge and MATLAB simulation involve translating physical signals (light) into numerical data that can be understood and manipulated by a device or computer.

Discuss, Reflect & Debrief



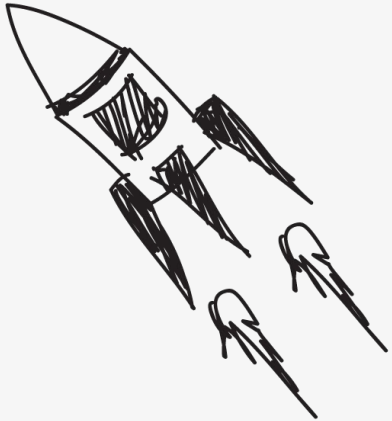
Signal Processing Connection

- How did the process of converting light levels to musical notes demonstrate signal processing?
- In what ways did the "reading" of the black-and-white image relate to how computers process images?
- What challenges did you encounter in translating a visual signal (light from the image) into an audio signal (musical notes)?
- How might understanding signal processing and image processing be useful in other real-world applications?
- If you were to enhance the "Bright Beats Challenge," what aspects of signal processing or image processing would you try to incorporate or improve?

Reflection

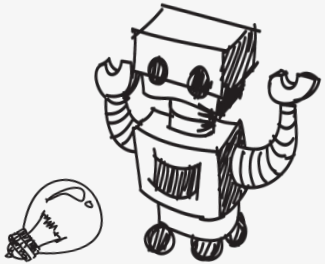
- What was the most challenging part of building your light-sensitive musical synthesizer?
- How did the micro:bit translate light levels into musical notes?
- What did you learn about how computers "see" and interpret images?
- In what ways do computers and programs like MATLAB represent images differently from the micro:bit device?
- What are key differences between how the micro:bit device functions and how MATLAB represents and transforms images?
- How might this project help visually impaired individuals interact with images in new ways?
- If you were to do this challenge again, what would you do differently?

Powered by IEEE
TRY Engineering



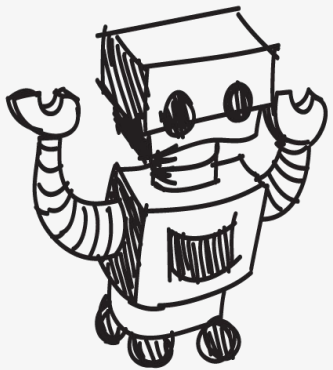
MORE RESOURCES:

Vocabulary, Background Knowledge, Dig Deeper, EDP, Engineering Fields



IEEE

Vocabulary



Vocabulary

- **Amplitude**: How tall the wave is from the middle line to the crest or trough. Bigger amplitude means more energy, like louder sounds or brighter light. Smaller amplitude means softer or dimmer signals.
- **Frequency (Hz)**: How fast something vibrates each second. Higher frequency means higher pitch (like a whistle); lower frequency means lower pitch (like a drum). For images, different frequencies correspond to different colors.
- **Image Processing**: How computers transform signals in an image to, for example, make the image clearer or smaller, or transform the image by adding filters, finding faces, or removing blur.
- **Intensity**: How bright a light is shining as measured by a light sensor or represented numerically by computer programs.
- **LED (Light-Emitting Diode):** An LED is a small light that converts electricity into light. It's super efficient, doesn't get hot like old bulbs, and comes in lots of colors. LEDs are used in things like flashlights, screens, and even the micro:bit's 5×5 grid.
- **Light Sensor**: A small tool that measures how bright the light is. Found in phones, cameras, and automatic lights.

Vocabulary

- **Light Wave**: A wave made of light energy that moves super fast and helps us see colors, shapes, and brightness.
- **MATLAB**: MATLAB is a programming platform with the power to perform complex calculations. It's used by millions of engineers and scientists to analyze data, develop algorithms, and create models.
- **Pixel**: A pixel is a tiny dot that makes up a picture on a screen. When you look at a photo or a game on a computer, tablet, or TV, the picture is made of thousands (or even millions!) of these little dots. Each pixel can be a different color and have different light intensity, and when they all work together, they create the image you see.
- **Phototransistor:** A tiny electronic part that reacts to light. When light shines on it, the phototransistor lets more electricity flow through a circuit. The brighter the light, the more electricity it allows. It's like a light-sensitive switch that helps devices measure how much light is around. Phototransistors are used in devices such as remote controls and in the micro:bit's 5x5 grid.

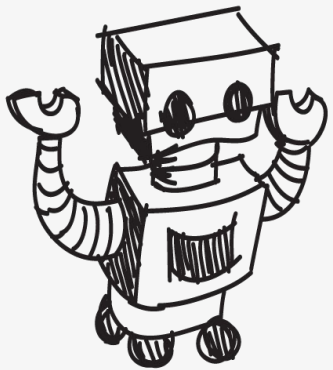
Vocabulary

- **Propagation**: The direction the wave travels. Even though the wave moves up and down, the energy moves forward, like a stadium wave passing through the crowd.
- **Sampling Period:** When scientists and engineers analyze signals, it's important to choose the right sampling period. Sampling is like taking snapshots of the signal. If the sampling period is small, we take snapshots of the signal very often. If the sampling period is large, we wait longer between snapshots. Choosing the right sampling period can be tricky: a shorter period (lots of snapshots) is great for accuracy and detail, but can take up a lot of space on the computer while a longer period (fewer snapshots) can save space and power, but might miss out on details.
- **Signal Processing**: How machines or computers clean up, change, or understand signals like sound, light, or movement.
- **Smoothing:** A common step when scientists and engineers are trying to understand different signals (like brightness or sound signals). It helps to make patterns clearer, reduce noise, and make signals easier to analyze. Think of it like brushing messy hair. The hair is still there, but now it's neat and easier to see the shape!

Vocabulary

- **Sound Wave**: A wave made of vibrating air that travels to our ears so we can hear music, voices, and noises.
- **Synthesizer**: An electronic device or app that creates sound by changing wave patterns. Used in music to make beats and effects.
- **Velocity:** The velocity of a wave is how fast it travels through a material
- **Wave**: A repeating pattern that moves energy from one place to another. Waves can carry sound, light, or motion.
- **Wavelength**: The distance between two matching points on a wave, like crest to crest. Short wavelengths are close together (like a whistle), and long wavelengths are spread out (like a drumbeat).

Background Knowledge



Signal Processing

Have you ever taken a photo or recorded a sound on your phone? That's signal processing in action! A signal is just a way to carry information, like light from a sunset or sound from your voice. These signals come from the world around us, and they usually travel in waves.

Light and sound are both types of waves.

- **Light waves** move super fast (about 300 million meters per second) and help us see colors, shapes, and brightness.
- **Sound waves** travel much slower through the air (around 340 meters per second) and let us hear music, speech, and noise.

Signal Processing

Each wave has a frequency, which means how fast it oscillates or vibrates.

High frequency means fast vibrations (like a whistle), and low frequency means slow vibrations (like a drumbeat).

The velocity of a wave is how fast it travels through a material, and it depends on the type of wave and what it's moving through. Light has a much higher velocity than sound, which is why we see lightning before we hear thunder!

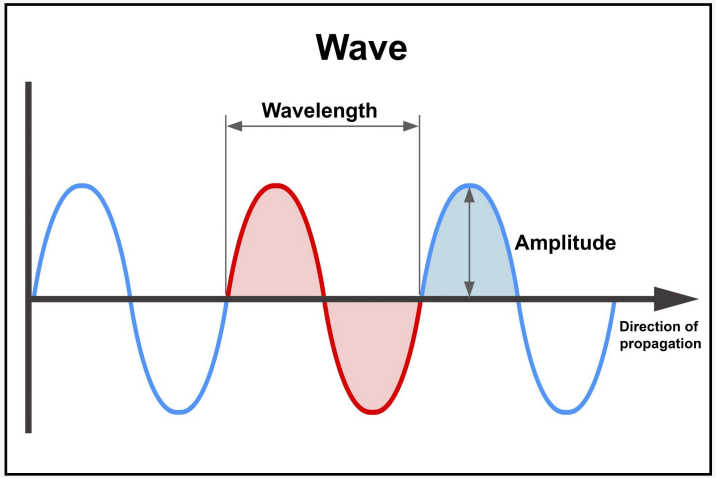


Image Processing

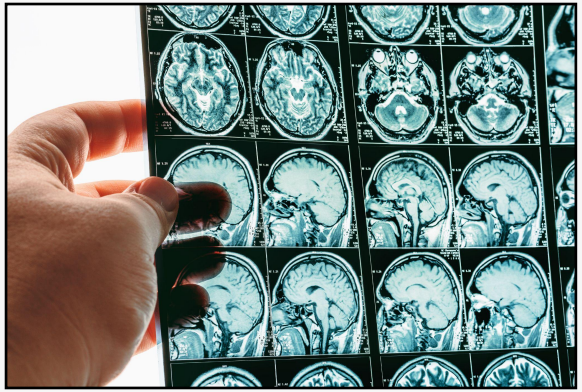
Image processing is a type of **signal processing**. It's how computers and devices understand, improve, and transform pictures. Just like sound is a wave that carries information to our ears, light is a wave that carries visual information to our eyes (and to cameras!). When you snap a photo, your phone captures light waves bouncing off objects and turns them into digital signals, patterns of numbers that represent colors, brightness, and shapes.

Once the image is captured, it can be digitally transformed using image processing techniques. This can include:

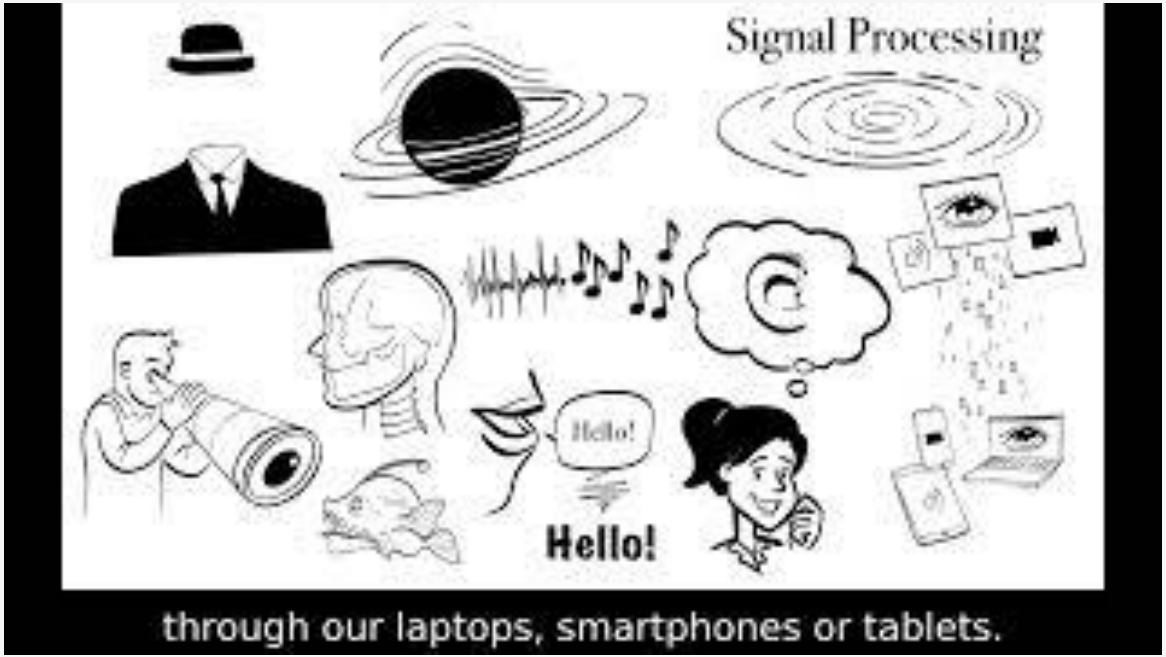
- **Sharpening** a blurry photo so edges look clearer
- **Smoothing** a photo to make it blurry
- **Adjusting brightness or contrast** to make details pop
- **Removing noise** (such as graininess) to make the subject of the image more clear
- **Detecting edges** to find outlines of objects
- **Recognizing faces** or objects using AI
- **Compressing** the image to make it smaller for storage or sharing

Real World Image Processing

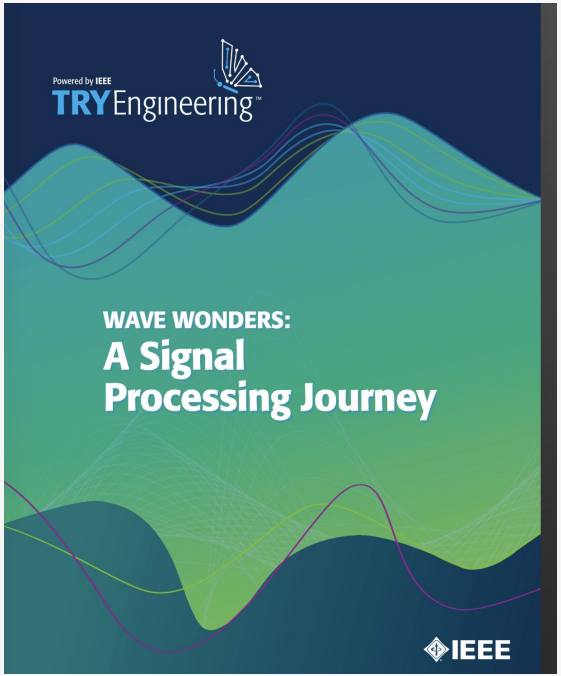
- **Photo Filters**: When you add a filter on Instagram or Snapchat, image processing changes the color balance, contrast, and texture to create a mood or style.
- **Medical Imaging**: MRI and X-ray machines use image processing to highlight tissues, detect tumors, and help doctors make diagnoses.
- **Self-Driving Cars**: Cameras on autonomous vehicles use image processing to detect lanes, signs, pedestrians, and other cars.
- **Satellite Images**: Scientists use image processing to study weather patterns, track wildfires, or monitor forests from space.
- **Microscope Analysis**: In biology labs, image processing helps researchers count cells, measure growth, or detect tiny structures.



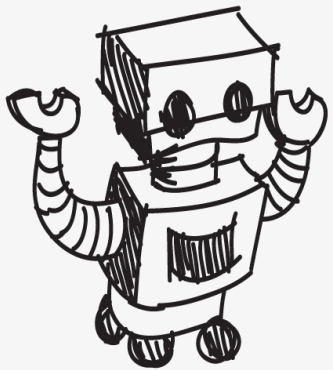
Watch: What is Signal Processing?



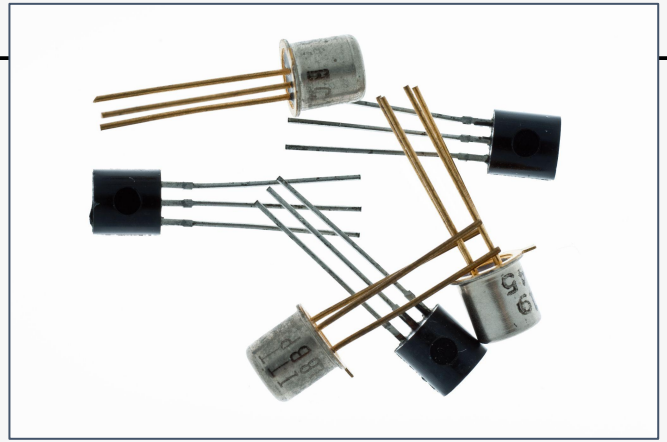
Explore Wave Wonders EBook



Dig Deeper



Light Sensors & micro:bit



Light sensors, like the one built into the micro:bit, detect the intensity of ambient light using a component called a **phototransistor.** A phototransistor is a light-sensitive semiconductor that changes its electrical resistance based on how much light it receives.

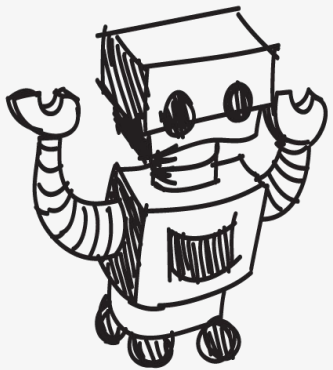
In the micro:bit, the LED display doubles as a light sensor by measuring how much light is reflected back into the LEDs. This allows the device to quantify brightness on a scale from 0 (dark) to 255 (very bright), enabling it to respond to environmental lighting conditions.

Can LEDs See Light?

It sounds strange, but yes — the same little lights (LEDs) on the micro:bit that shine can also help sense light! The micro:bit doesn't have a separate light sensor. Instead, it cleverly uses the LED display to measure how bright the room is.

How It Works: The micro:bit has 25 tiny LED lights arranged in a 5×5 grid, and they're not just for glowing, they can also sense light. When you ask the micro:bit to check the light level, it briefly turns off the LEDs and measures how much light is bouncing back. Based on what it detects, it gives you a number between 0 and 255, where 0 means super dark and 255 means very bright.

Engineering Design Process



The Engineering Design Process

Learn about the engineering design process (EDP). The process engineers use to solve problems.

(Video 1:47)

Source: TeachEngineering YouTube Channel http://www.youtube.com/watch?v=b0ISWaNoz-c

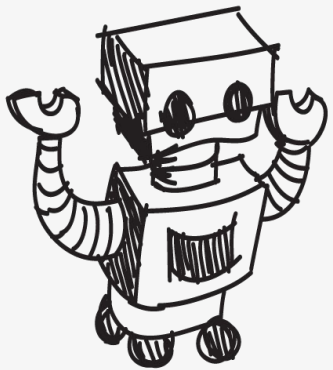
Engineering Design Process

- Divide into teams of two (or up to 4 max)
- Review the challenge and criteria & constraints
- Brainstorm possible solutions (sketch while you brainstorm!)
- Choose best solution and build a prototype
- Test then redesign until solution is optimized
- Reflect as a team and debrief as a class

Productive Failure

- The engineering design process involves productive failure: test, fail, redesign. Iterate again and again until you have the best possible solution.
- It is important to document iterations to keep track of each redesign. Use the engineering notebook to sketch ideas, document iterations and any measurement and/or calculations.
- It’s also important to showcase the fact that there can be multiple solutions to the same problem. There’s no one “right” solution.

Engineering Fields



What is Engineering?

Learn about engineering and how engineers are creative problem solvers and innovators who work to make the world a better place.
(Video 3:43)

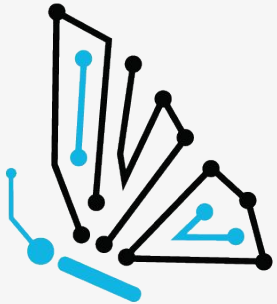
Source: TeachEngineering YouTube Channel - http://www.youtube.com/watch?v=H9VDkvqGmVo

Related Engineering Fields

- There are several types of engineering fields that are involved circuits. Here are just some of the related engineering fields.
  - Computer Engineering
  - Software Engineering
  - Electrical Engineering

- Download the Engineering Fields Infographic
  How will **YOU** change the world?
  Engineers are the inventors and problem solvers of the world! More than twenty five major specialties are recognized in the field of engineering. **Engineers make the world a better place!**

For more engineering lesson plans and resources like games, engineering careers, and STEM opportunities visit IEEE's TryEngineering.org



Powered by IEEE

TRY Engineering