1:30 HR

Robot Dozer

Simulate an automated dozer using the time-of-flight sensor.

Introduction

Our mission is to transform Alvik into an automated work machine on a simulated job site. Your task is to create a small dozer that can clear items from the work area by pushing them off your desk.

But, be careful! Drive too far and your robot may have a nasty accident.

So, how do we achieve automation? In other words, how do we provide our machine with basic intelligence so that it can respond to its environment without our help?

To achieve our goals, we'll learn about the Time-of-Flight (ToF) sensor and how to program conditional statements. The ToF sensor is a cool device that can detect the exact distance of objects in front of your robot. Conditional statements in our code will give your robot decision-making power so that it can react to environmental changes. As you learn more, you'll develop the power to provide machines with intelligence and build smart systems.

Alright, gloves on. The job site won't clear itself... at least not yet!

Learning Objectives

- Understand how the Time-of-Flight (ToF) sensor works.
- Write code to read and print ToF sensor data.
- Make decisions with conditional statements `if` , `elif` , and `else` .
- Use incoming sensor data to control motor movements.
- Develop automated systems to clear items from the simulated site.

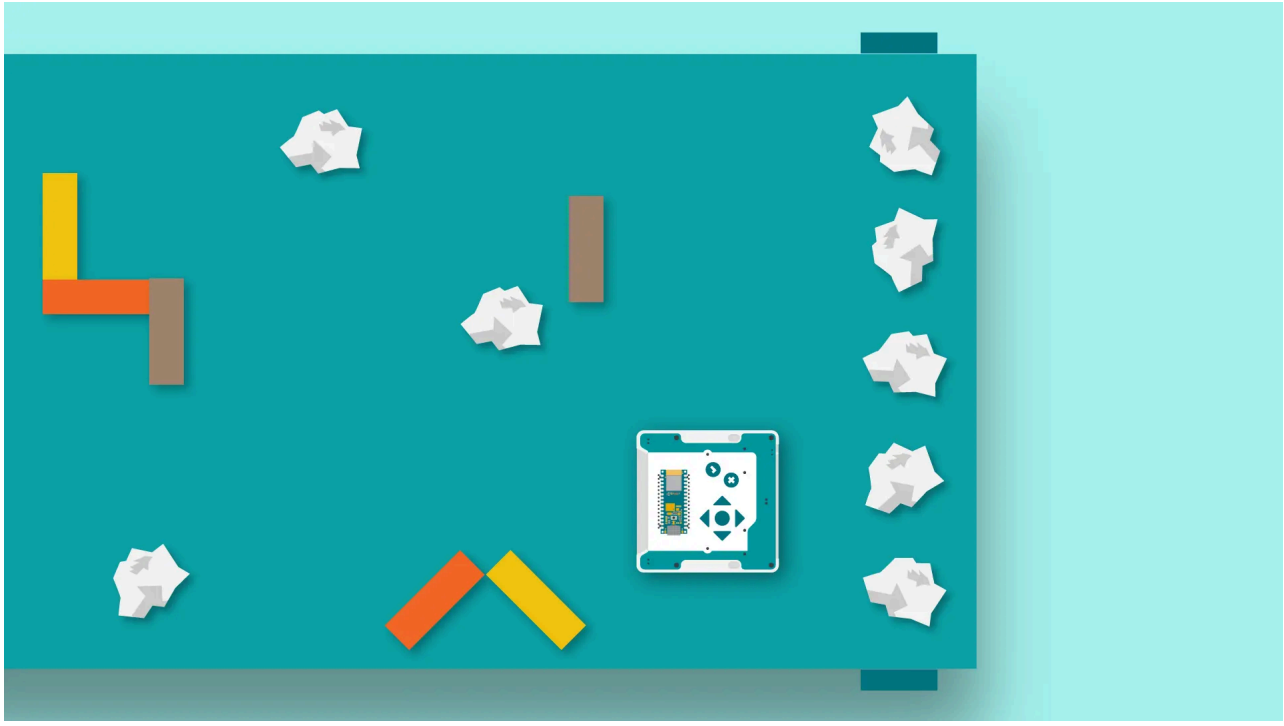
Setting Up

> **`WARNING!`** Be careful - do not allow your robot to fall on the floor - it could seriously damage the device! Some safety tips include:
>
> - Test movements at slow speeds.
> - Conduct the first code experiments on the floor.
> - Always be ready to catch your robot just in case.

Gather the materials needed for this project. Keep in mind that objects should be lightweight or your robot won't be able to push them.

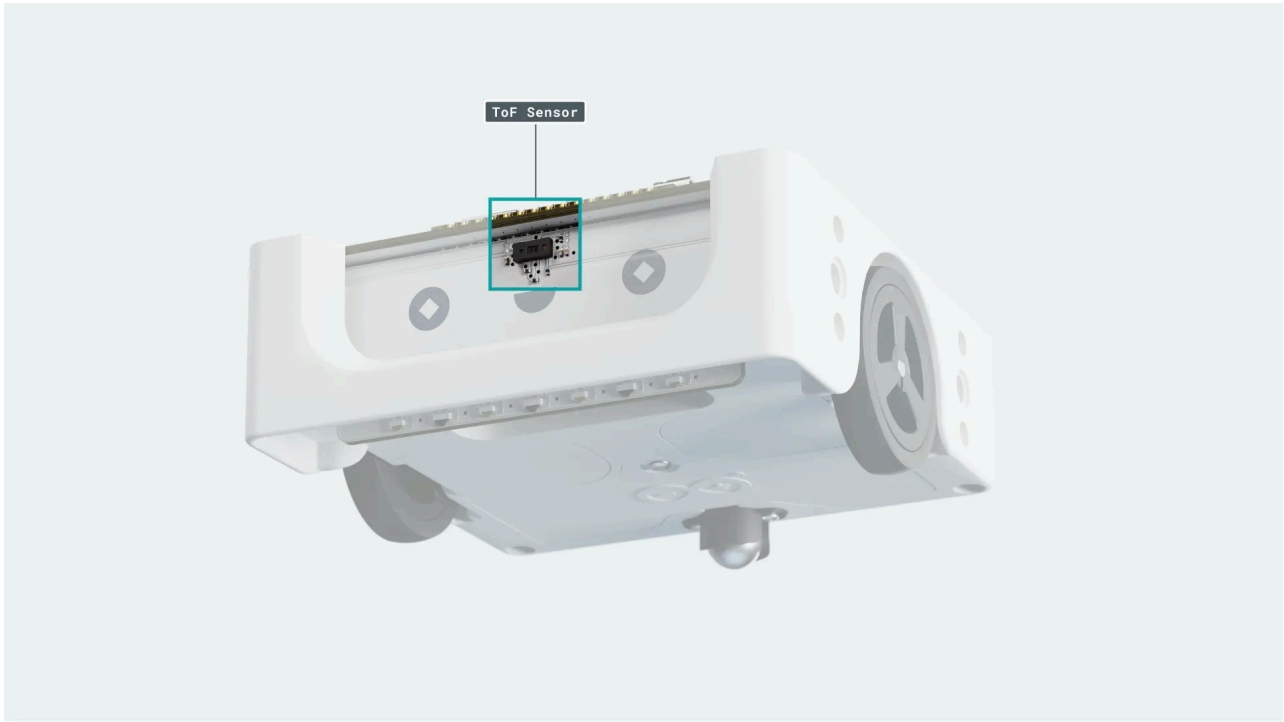
- **Objects** (small empty boxes, paper balls, etc.)
- **Tape Measure or Ruler**
- **Notebook** (recommended)

Here we have an example of the "job site" using paper balls. Feel free to position items differently on your own simulated site.

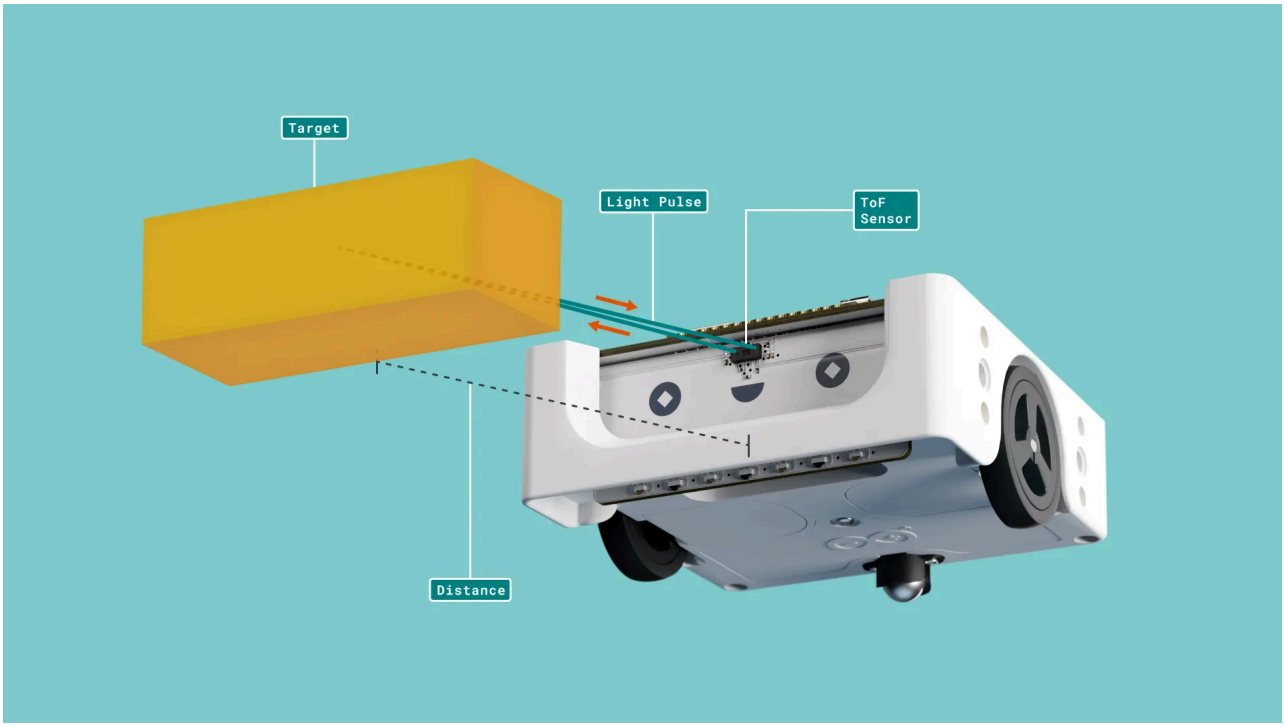


ToF Sensor

Have you ever played "echo" in a big room? You shout and then wait to hear your voice bounce back to you. The Time-of-Flight (ToF) sensor on the Alvik robot works a bit like that, but instead of using sound, it uses light.



The ToF sensor sends out a very quick flash of light that we can't see because it's invisible to our eyes. This light travels out, hits an object like a chair or a wall, and then bounces back to the sensor. The really clever part is that the sensor can measure exactly how long the light took to hit the object and come back.



Why is this important? It helps the robot figure out how far away things are. If the light comes back super quickly, the object is close. But if it takes longer, the object is further away. Knowing this, the robot can decide what to do, like stopping before it runs into things or moving towards something it needs to reach.

By using the ToF sensor, our robot gets a special way to "see" the world around it using light, and we can program it to use this information to react accordingly.

> **WARNING!** The ToF sensor might still have a protective sticker, remove it before using the sensor to ensure accurate measurements.

Sensor Testing

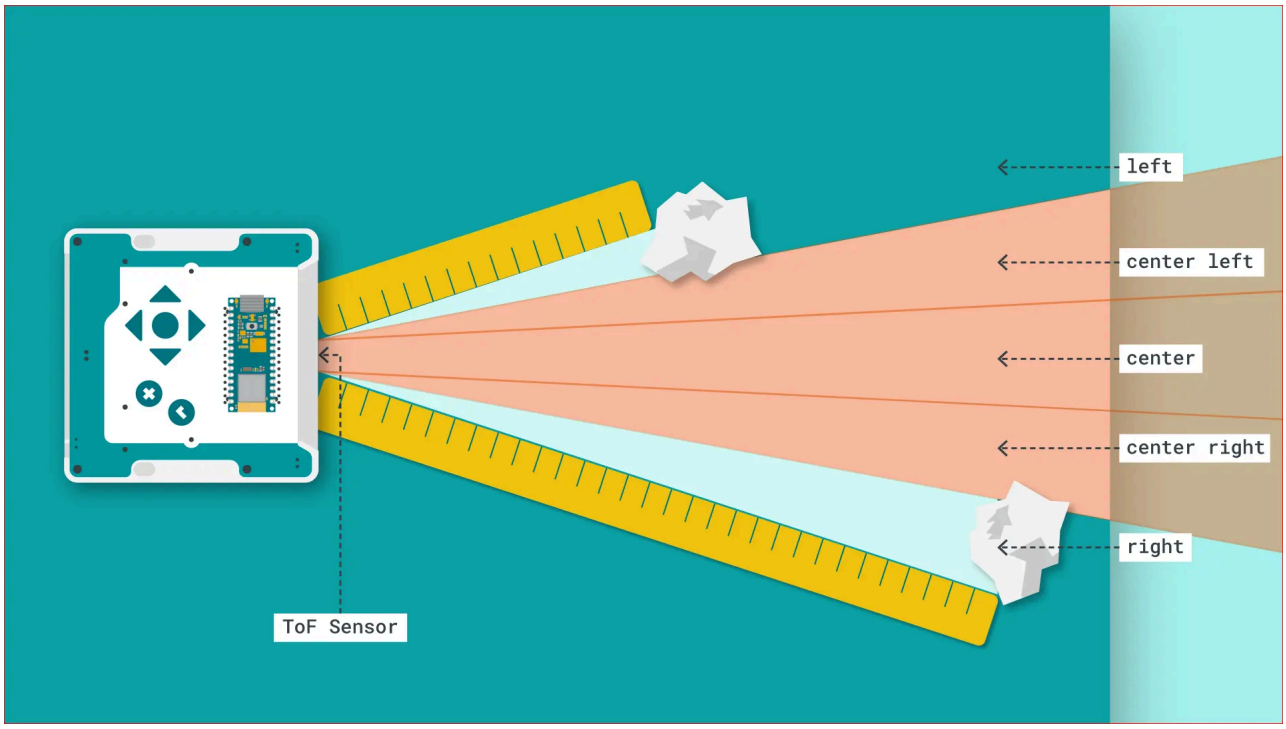
Alvik is equipped with a powerful ToF sensor that can detect multiple objects in different zones, all thanks to the sensor's wide field of view. During testing, we will explore these zones to better understand their practical use in upcoming projects.

Begin by creating a new file called ***tof_test.py*** in Arduino Lab and copy the code below into the editor.

```
from arduino import *
from arduino_alvik import ArduinoAlvik

alvik = ArduinoAlvik()

def setup():
  alvik.begin()
  delay(1000)

def loop():
  all_zones = alvik.get_distance()
  print(all_zones)
  delay(1000)

def cleanup():
  alvik.stop()

start(setup, loop, cleanup)
```

In this example, we are working with a new method called `get_distance()` to gather distance data from the ToF sensor. Run the code and place your hand in front of Alvik, moving it from side to side as you observe the data printed to the terminal.

As you likely noticed, there are a total of **five distance zones measuring in millimeters** in the following order - left, center left, center, center right, and right. These zones are capable of detecting objects nearly three meters away, but be aware that the reliability of readings decreases at greater distances.



The next script provides greater control of individual ToF zones thanks to **multiple variable assignment**. Previously, we created a single variable called `all_zones` to store a list of the parameters returned by the `get_distance()` method, displayed as (left, center left, center, center right, right). By creating many variables in a single line, separated by commas, each variable assigns itself to a corresponding index of the parameters.

Observe the updated `def loop()` below to see multiple variable assignment in action.

```
def loop():
  left_tof, cleft_tof, center_tof, cright_tof, right_tof = alvik.get_distan
  print("Center:", center_tof)
  delay(1000)
```

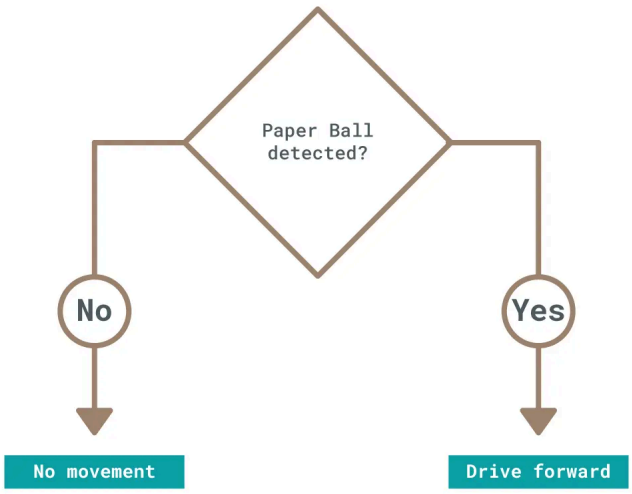
- **Challenge:** Our last script has multiple variables assigned, but we only printed the Center Zone. Can you create more print statements to display another zone in a similar format? Remember to test the sensor while observing the terminal to ensure values are appearing as expected.

Basic Dozer

The goal is to program your robot to behave like a small dozer, clearing items from the desk as soon as they are detected. For testing, we recommend using paper balls, as they are lightweight and won't be damaged when they fall to the floor. As soon as a paper ball is placed in front of Alvik, it should push it over the edge of the desk. However, if nothing is detected within range, Alvik should remain stationary.

> **`WARNING!` Be prepared to catch Alvik during testing!** Your robot will come dangerously close the the edge of the table, and although it should stop before reaching the edge, be ready just in case.

So, let's take a moment to think how all this could work. Using logic, you may have already realized that we are dealing with a simple condition in which we ask a yes/no question - **Is an object detected in the robot's field of view?** If the answer is *yes*, the statement is `True`, so Alvik should drive forward. Otherwise, the answer is *no*, so the statement is `False` and Alvik should not move.

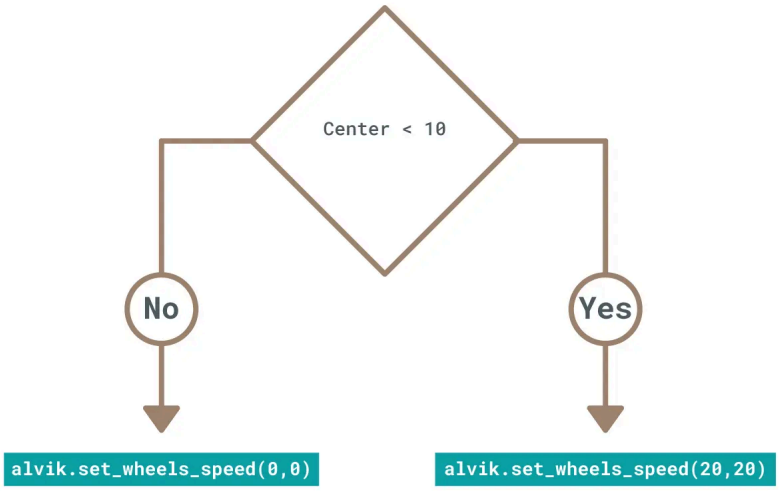


Let's translate all this into code. Make sure your last practice script is saved and **create a new file called *robot_dozer.py*** for our next program.

```
from arduino import *
from arduino_alvik import ArduinoAlvik

alvik = ArduinoAlvik()

def setup():
  alvik.begin()
  delay(1000)

def loop():
  left, cleft, center, cright, right = alvik.get_distance()
  print(center, "cm")
  delay(100)

  if center < 10:
      alvik.set_wheels_speed(20, 20)
  else:
      alvik.set_wheels_speed(0, 0)

def cleanup():
```

This script focuses on the Center Zone of the ToF sensor, which means our simulated dozer will only respond to items placed directly in front of it. If a paper ball is placed less than 10 centimeters in front of the sensor, Alvik drives forward at a speed of 20 RPM. However, if this condition is not true, the RPM values remain at 0.

Let's take a look at the flowchart example once again with updated pseudocode descriptions.



Once you have tested your code with paper balls, move on to the next section where we create more advanced conditions to improve Alvik's intelligence.

Object Tracking

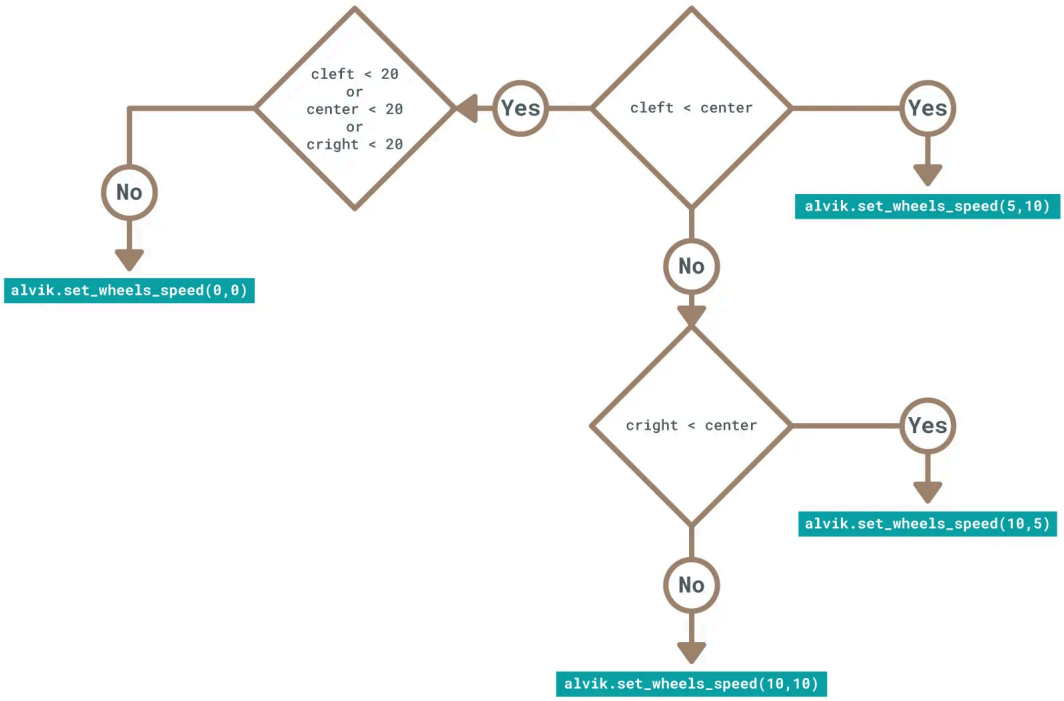
Alvik can now sense objects placed directly in its path. That's awesome but we can build on what we have learned to do even better! In the real world, things won't always be easy to find, so it's our job to help machines track objects. Imagine, for example, we place a paper ball slightly to the left or right of your robot. How could we locate its position and move toward it?

To get the job done, Alvik needs more complex decision making abilities, which means our code needs more conditional statements and even a condition inside of a condition! The good news is that multiple conditions are easy - after your first `if` statement, simply use `elif` for each additional conditional statement you would like to check. As always, you may include `else` as your final statement in the case that all previous statements return `False`.

- **Challenge:** Can you write a new program to help our dozer move towards objects outside its center zone? Use the flowchart below for guidance and keep

in mind that we are using the following sensor zones: center-left, center, and center-right.

Follow the logic below to create multiple conditions and even embedded conditions.



If your code worked, you're a true champ. If things didn't go your way, no stress, you'll get there soon enough. In any case, compare your strategy to the example below, and remember to save your work!

```
from arduino import *
from arduino_alvik import ArduinoAlvik

alvik = ArduinoAlvik()

def setup():
  alvik.begin()
  delay(1000)

def loop():
  left, cleft, center, cright, right = alvik.get_distance()
  print(cleft, "|", center, "|", cright )
  delay(100)

  # If any zone is True, an item must be in range
  if cleft < 20 or center < 20 or cright < 20:
      if cleft < center:
          alvik.set_wheels_speed(5, 10)  # Item detected left, so turn le
```

Similar to examples earlier in this lesson, we start by gathering data from ToF sensor zones and printing them to the terminal to observe their values. Later, we use these values in our conditional statements to instruct Alvik to move forward, turn right, turn left, or not move at all.

The first condition `if cleft < 20 or center < 20 or cright < 20:` checks if there is a paper ball detected within 20 cm of any of the three zones. If the value is less than 20 in any of these zones, the condition is `True` and the lines of code indented within it are executed. Otherwise, if `False`, there is no paper ball within range, so the `else:` statement idles the motors with `alvik.set_wheels_speed(0, 0)`.

When the first condition is true, meaning a ball is detected, the script checks three statements embedded within it. If `cleft < center`, a ball must be to the left of the robot, so the RPM speeds adjust to turn slightly to the left. If `cright < center`, a ball is detected to the right and Alvik turns in that direction. And at last, if not right nor left, we can conclude that the ball is in the center field of view and we must drive straight ahead.

Placing conditions inside other conditions is known as **nesting**. If the outer condition is true, the inner conditions "nested" inside are checked. We will see more nested conditions in future lessons.

Expanded Vision

Consider how we could expand the vision of your robot using what we have learned in this lesson. By improving Alvik's overall field of view, we could develop a smarter system capable of scanning a larger area. Try out the challenges below to put your skills to the test.

Challenge 1

In our last script, we successfully used three zones of the ToF sensor to detect and track down items on your desk. For this challenge, provide Alvik with a wider field of view by using all five zones in your script.

Tip: You will have several conditional statements, some embedded inside of others. **Operators** like `and`, `or`, and `not` will help you develop your conditional statements.

Challenge 2

Think about the use of movement to improve the behavior of your robot. How could we use the Spin Turn to scan a larger area of your desk?

Tip: Investigate how radar scanning works in the real world to detect objects such as airplanes or ships. This may provide the inspiration needed for a solution.