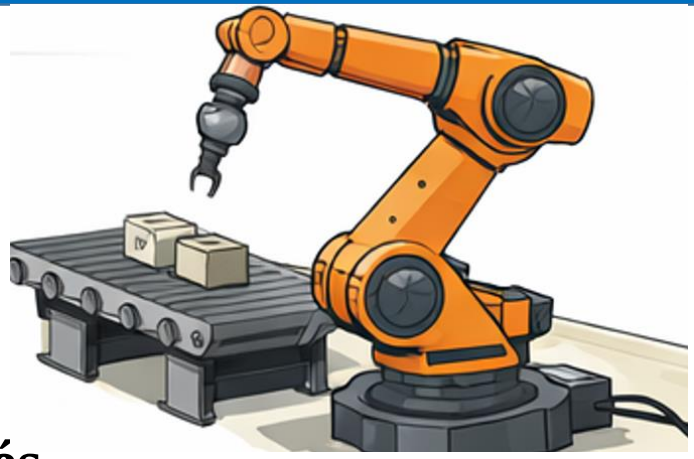
Robotique



Master en ingénierie des systèmes embarqués (MUS ISE)

Chapitre 5: Computer Vision Application to a robot-soccer

Année universitaire 2025/2026

Dr. Ing. LASSIOUI Abdellah

1. Introduction to Computer Vision in Robotics

1.1 Definition of Computer Vision

Computer Vision (CV) is a field of artificial intelligence and signal processing that enables machines to acquire, process, analyze, and interpret visual information from the real world in order to make decisions or perform actions.

In robotics, computer vision acts as the **visual perception system**, allowing robots to:

- ❑ Perceive their environment
- ❑ Understand objects, scenes, and motion
- ❑ Interact intelligently with the physical world

1.2 Role of Computer Vision in Robotics

Computer vision enables robots to:

- ❑ Localize themselves in an environment
- ❑ Detect and recognize objects
- ❑ Track moving targets
- ❑ Navigate autonomously
- ❑ Manipulate objects precisely
- ❑ Interact safely with humans

Without vision, robots rely only on predefined trajectories or limited sensors.

Vision provides **adaptability, autonomy, and intelligence**.

1.3 Human Vision vs Robotic Vision

Aspect	Human Vision	Robotic Vision
Sensor	Eyes (retina)	Cameras (CCD, CMOS, depth sensors)
Processing	Brain (biological neural network)	Algorithms and artificial neural networks
Adaptability	High	Depends on models and training
Precision	Moderate	Can be very high (pixel-level accuracy)

2. Image Formation and Cameras
2.1 Image Definition

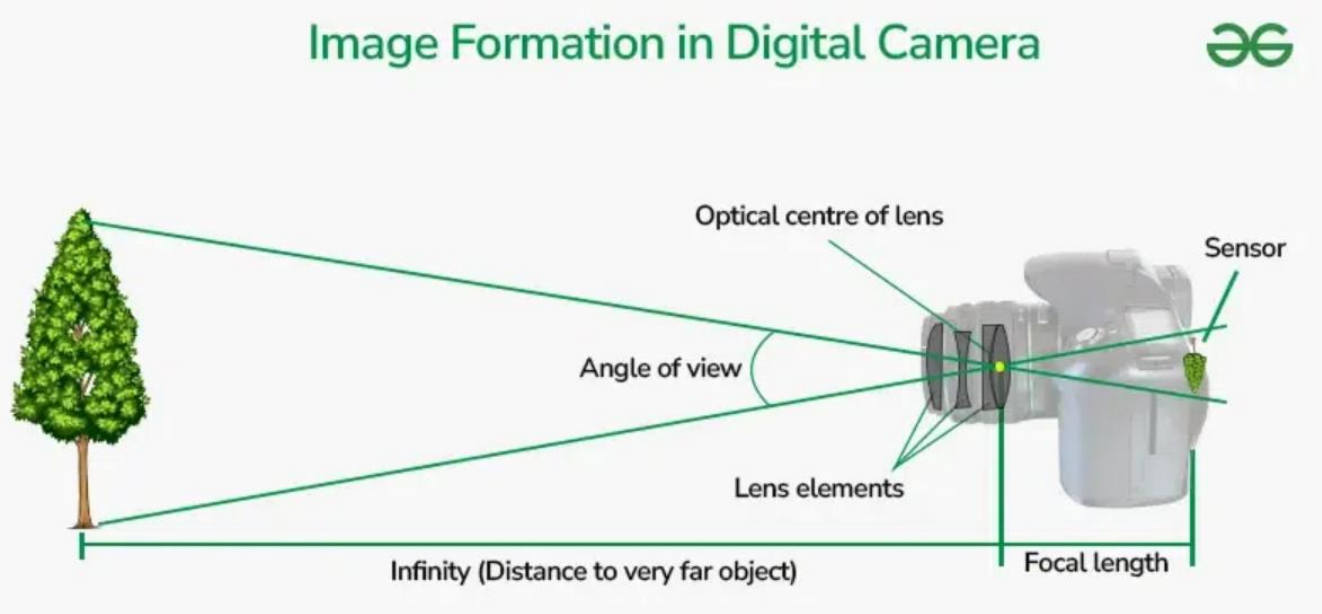
- An **image** is a 2D representation of a 3D scene formed by projecting light reflected from objects onto a sensor.

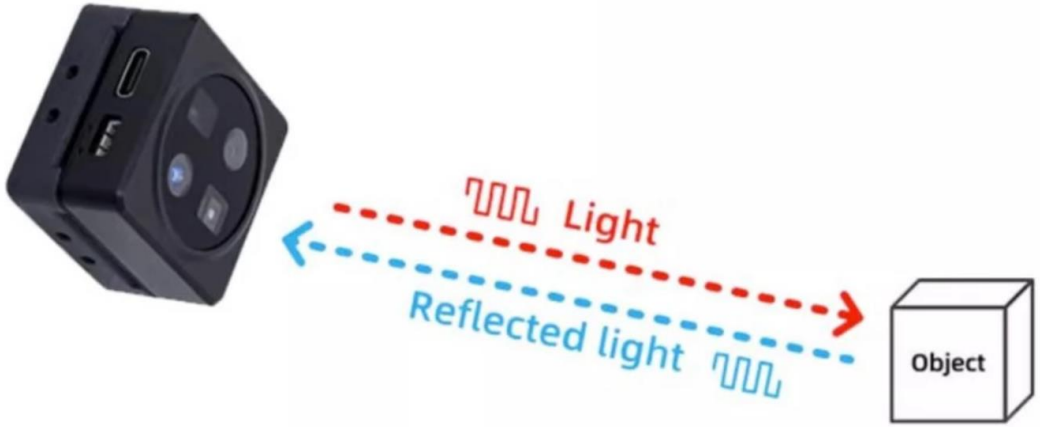
- Mathematically, a digital image is a discrete function:

I(x,y) where x and y are pixel coordinates and I represents intensity or color.

2.2 Image Formation and Cameras

Image formation in camera may be termed as the process through which camera creates a visual representation of image which it captures. Light enters the camera through small opening called as aperture, which is directed by lens to focus on a sensitive surface such as film. The film then records this image allowing camera to capture and save the photograph of the scene. This process involves the precise control of light to produce accurate images.





2.2 Camera Types Used in Robotics

2.2.1 Monocular Camera

- ❑ Single camera
- ❑ Provides 2D information

2.2.2 Stereo Camera

- ❑ Two cameras separated by a baseline

2.2.3 RGB-D Camera

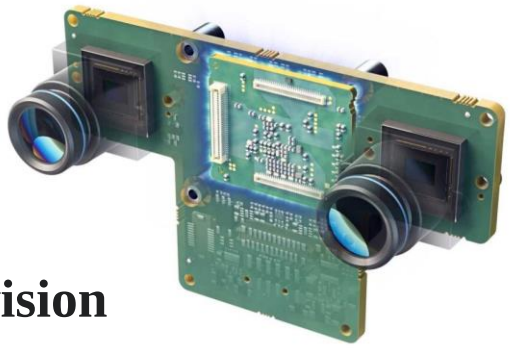
- ❑ Provides color (RGB) and depth (D)
- ❑ Examples: Kinect, Intel RealSense

2.2.4 Event-Based Camera

- ❑ Asynchronous sensing of brightness changes
- ❑ High temporal resolution
- ❑ Used in fast robotics applications

RGB-D 3D Imaging and Real-Time Rendering Sensor Camera for SLAM and Robotic Navigation - DFRobot

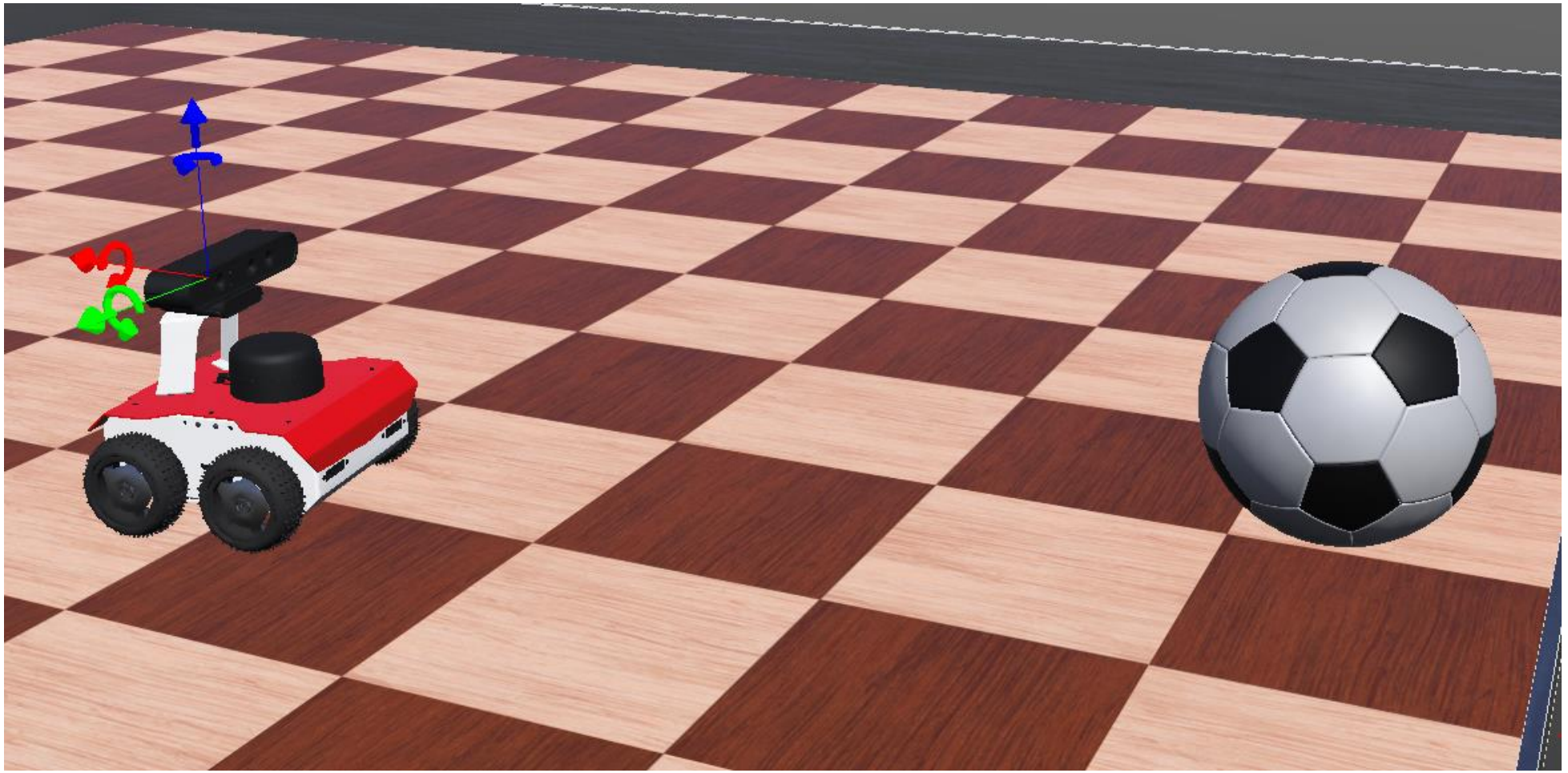
Single camera



Stereo Camera

An **event camera**, also known as a **neuromorphic camera**, **silicon retina**, or **dynamic vision sensor**, is an imaging sensor that responds to local changes in brightness.
Event cameras do not capture images as conventional (frame) cameras do. Instead, each pixel inside an event camera operates independently and asynchronously, reporting changes in brightness as they occur, and staying silent otherwise.

Application of computer vision in robotics: robot-soccer



ROSBOT mobile robot as an use case

ROSbot versions

ROSbot is available in three options which, next to features mentioned before, also include:

ROSbot 3 / ROSbot 3 PRO | **ROSbot 2R** | **ROSbot 2 PRO** | **ROSbot 2**

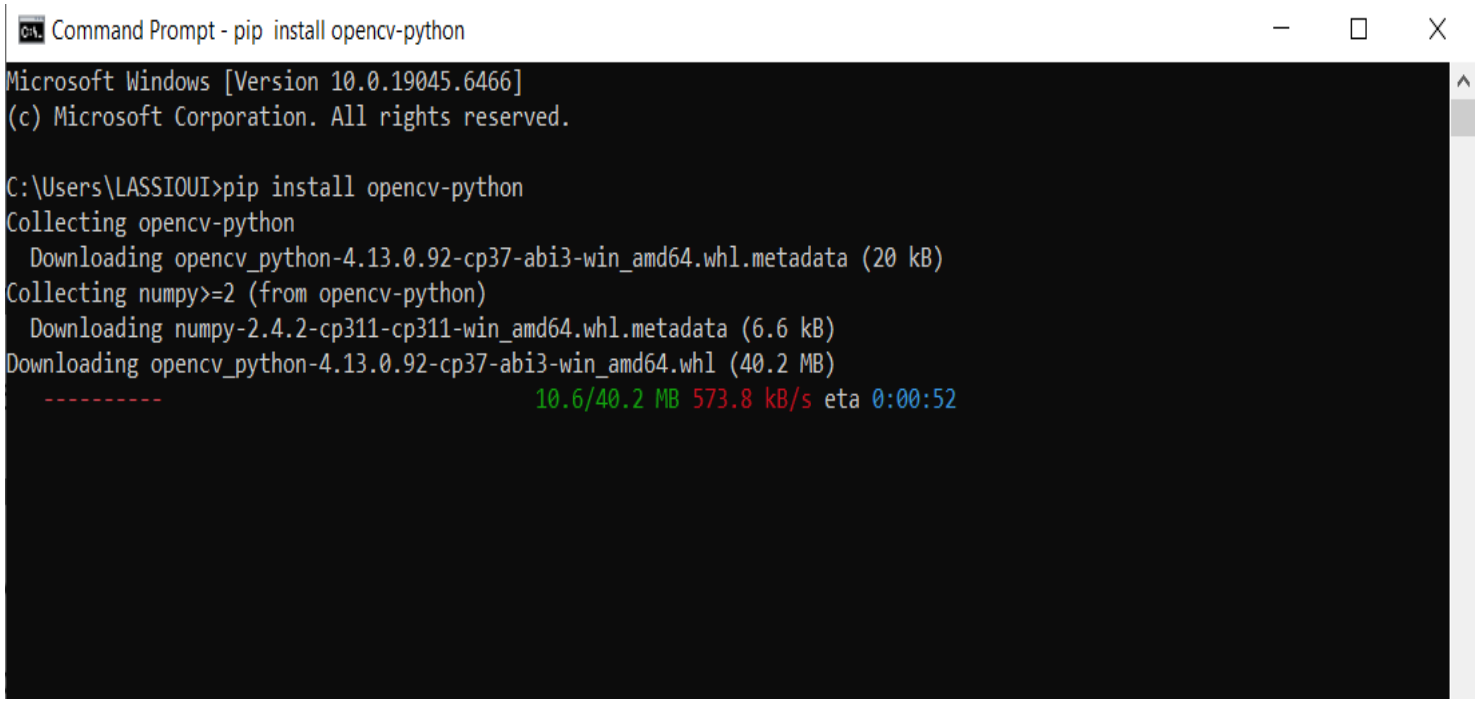
- Raspberry Pi 5 (ARM64 architecture) quad-core ARM-8 Cortex A76 @ 2.4GHz, 8GB RAM and 64 GB MicroSD flash memory.
- 3D camera: Luxonis OAK-D Lite / Pro
- LIDAR: RPLIDAR C1 / S2

Need of of opencv python library

OpenCV (Python) = a computer vision library used in Python to process images and videos.
More precisely:
•**OpenCV** = *Open Source Computer Vision Library*
•It provides ready-made functions for:

- Reading images and video
- Image filtering
- Edge detection
- Object detection
- Face detection
- Motion tracking
- Image transformations

Install opencv-python
To do that open the command prompt and tape the instruction :
Pip install opencv-python
You will see somrthing like the following picture:



The code used for the application (part1)

controller_3.py | controller_1.py | controller_2.py | controller_3.py

```python
from controller import Robot
import cv2
import numpy as np

robot = Robot()
timestep = int(robot.getBasicTimeStep())

# Hardware Setup
cam = robot.getDevice("camera rgb")
cam.enable(timestep)
width = cam.getWidth()
height = cam.getHeight()

motor_names = ['fl_wheel_joint', 'fr_wheel_joint', 'rl_wheel_joint', 'rr_wheel_joint']
motors = [robot.getDevice(name) for name in motor_names]
for m in motors:
    m.setPosition(float('inf'))
    m.setVelocity(0.0)

def set_speed(left, right):
    motors[0].setVelocity(left)
    motors[2].setVelocity(left)
    motors[1].setVelocity(right)
    motors[3].setVelocity(right)

print("Searching and Approaching Ball...")
```

The code used for the application (part2)

```
while robot.step(timestep) != -1:
    raw_image = cam.getImage()
    if not raw_image: continue

    frame = np.frombuffer(raw_image, np.uint8).reshape((height, width, 4))
    frame = cv2.cvtColor(frame, cv2.COLOR_BGRA2BGR)

    # 1. Image Processing
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    ball_data = None # Store (x, y, area)

    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area > 150: # Ignore noise
            M = cv2.moments(cnt)
            if M["m00"] != 0:
                cX = int(M["m10"] / M["m00"])
                cY = int(M["m01"] / M["m00"])
                ball_data = (cX, cY, area)
                break
```

The code used for the application (part3)

```
52      # 2. Driving Logic
53      if ball_data:
54          cX, cY, area = ball_data
55
56          # Is the ball close enough to stop? (Adjust 8000 based on your ball size)
57          if area > 8000:
58              print("Target Reached! Stopping.")
59              set_speed(0.0, 0.0)
60              # break # Uncomment to end the simulation here
61          else:
62              # P-Controller for Steering
63              # center_error > 0 means ball is to the right
64              center_error = cX - (width / 2)
65
66              base_forward_speed = 4.0
67              steering_sensitivity = 0.02
68
69              left_speed = base_forward_speed + (steering_sensitivity * center_error)
70              right_speed = base_forward_speed - (steering_sensitivity * center_error)
71
72              set_speed(left_speed, right_speed)
73      else:
74          # If ball is lost, spin to find it
75          set_speed(2.0, -2.0)
```

Code explanation

1. The Initialization (The Hardware Link)

Before the robot can move, we have to tell Python how to talk to the Webots simulation.

Before the robot can move, we have to tell Python how to talk to the Webots simulation.

- `Robot()` **and** `timestep`: This creates the software object representing the robot. The `timestep` is the "heartbeat" of the simulation (usually 32ms); every time this beat happens, the robot gets new data.
- **The Motors**:

Python

```
m.setPosition(float('inf'))
m.setVelocity(0.0)
```

In Webots, motors are "Servos" by default (they move to a specific angle). By setting the position to **infinity**, we tell the robot: "Don't go to an angle; just spin at a constant speed." This is called **Velocity Control**.

- **The Camera**: We enable the camera using `cam.enable(timestep)`. Without this, the camera stays "off" to save your computer's CPU.

Code explanation

2. Image Processing (The Vision System)

The camera returns a long string of raw numbers. We use **OpenCV** to turn those numbers into a shape the robot can understand.

Step A: Conversion

Python

```
frame = np.frombuffer(raw_image, np.uint8).reshape((height, width, 4))
frame = cv2.cvtColor(frame, cv2.COLOR_BGRA2BGR)
```

Webots provides an image in **BGRA** (Blue, Green, Red, Alpha). OpenCV doesn't use "Alpha" (transparency), so we convert it to **BGR** format so we can process the colors.

Code explanation

Step B: Thresholding (The "Filter")

Since your ball is white and black, we use a **Threshold**.

Python

```
_, thresh = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)
```

This tells the robot: "If a pixel is brighter than 200 (white-ish), turn it pure white. If it's darker, turn it pure black." This creates a "mask" where the ball appears as a bright white blob and the rest of the world is invisible.

Step C: Finding the Center (Moments)

We find the **Contours** (the outline of the white blob). To find the exact center of that blob, we calculate **Moments**.

- `cX` : The horizontal center of the ball.
- `area` : The total number of white pixels. This is our "distance sensor." Because of perspective, an object that is closer takes up more pixels (a larger area).

Code explanation

3. The Logic (The "Brain")

Now that the robot knows **where** the ball is (cX) and **how big** it is (area), it makes a decision.

The Search State

If `ball_data` is empty, the robot spins: `set_speed(2.0, -2.0)`. This is the "Search" mode.

The P-Controller (Proportional Steering)

This is the most "intelligent" part of the code. We want the ball to stay in the **center** of the camera (which is `width / 2`).

1. **Calculate Error**: `center_error = cX - (width / 2)`.
    - If the ball is on the right, the error is a positive number.
    - If the ball is on the left, the error is a negative number.
2. **Adjust Wheels**:
    - `left_speed = base + (0.02 * error)`
    - `right_speed = base - (0.02 * error)`

      If the ball moves right, the left wheel speeds up and the right wheel slows down, causing the robot to "curve" toward the ball.

Code explanation

The Stop Condition

Python

```python
if area > 8000:
    set_speed(0.0, 0.0)
```

We don't use a ruler to measure distance. We simply say: "When the white blob is larger than 8000 pixels, it means we are close enough to touch it." At that moment, we kill the motor velocity.

Variable	What it represents	Why we need it
cX	Horizontal center of ball	To know if we should turn left or right.
area	Size of the white blob	To know if we are close enough to stop.
center_error	Distance from image center	To calculate how sharply we need to turn.
base_speed	Forward speed	To keep the robot moving toward the target.

Implémentation in webot