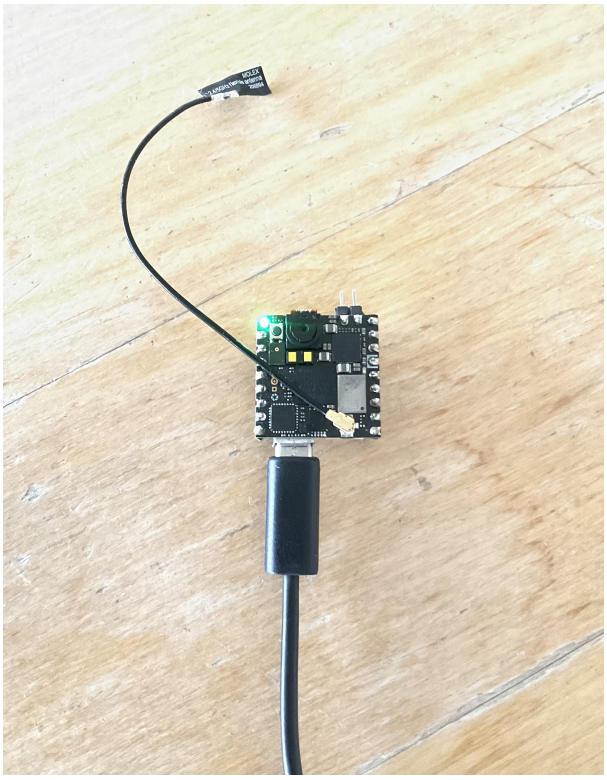
Object Detection with an Arduino Nicla Vision Camera

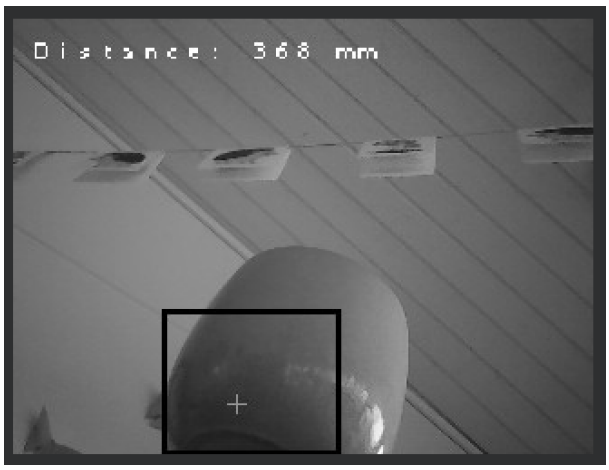
Course: TTK8
Year: 2025
Student: Rebekka Alve

1. Summary

This project implements a real-time obstacle detection system on an Arduino Nicla Vision using OpenMV firmware. It detects dark objects using blob detection and reads the distance from the integrated ToF sensor. The information is transmitted as camera frames over 2.4 GHz Wi-Fi, with an overlaid bounding box and distance on the frame, marking the closest obstacle.

The algorithm worked reliably for single dark objects in many conditions. Support for bright-object detection was removed because of too many false positives (walls, windows, etc.). For multiple dark objects, the selection was unstable but functional in some cases. Distance measurements were accurate when the target was centered, and less accurate when off-center. The stream experienced some lag; trade-offs (grayscale, QVGA, JPEG quality) were applied to improve streaming responsiveness at the cost of image and detection accuracy.





Note: Use the system only with proper consent and respect for privacy.

2. Table of Contents

3. List of Abbreviations

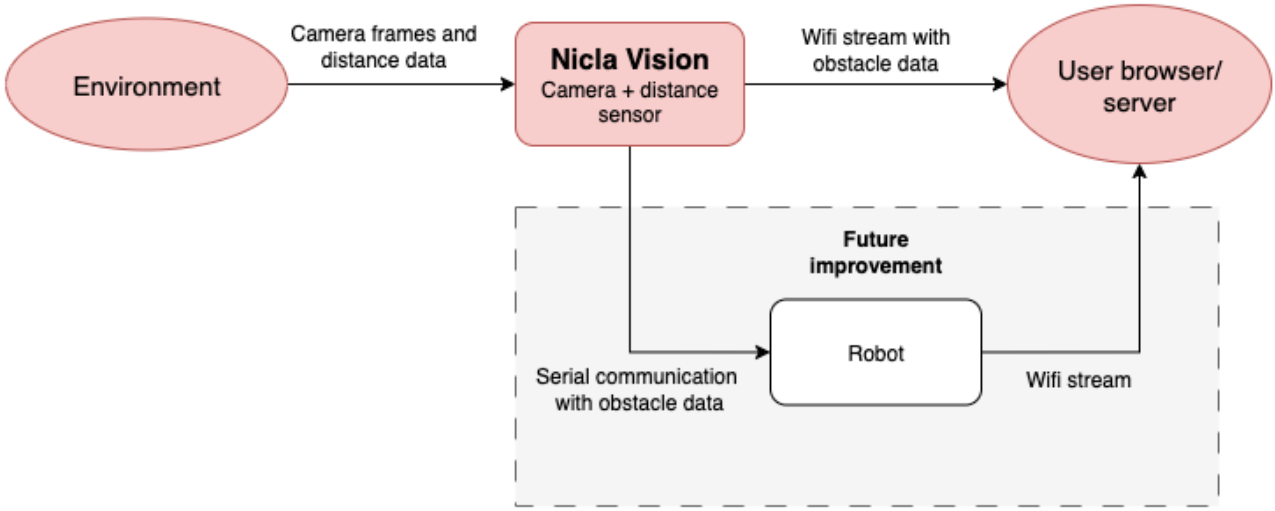
- ToF — Time-of-Flight (distance sensor, e.g., VL53L1X)
- QVGA — Quarter VGA (320 × 240 resolution)
- FPS — Frames Per Second
- Wi-Fi — Wireless Fidelity (2.4 GHz network used for streaming)

- OpenMV — OpenMV firmware / platform (used on Nicla Vision)
- IDE — Integrated Development Environment (OpenMV IDE)
- Nicla — Arduino Nicla Vision (camera module)
- USB — Universal Serial Bus (programming / power connection)
- HTTP — HyperText Transfer Protocol (used for browser streaming)
- JPEG — Joint Photographic Experts Group (image compression format used for streaming)
- WPA2 — Wi-Fi Protected Access II (wireless network security)
- IP — Internet Protocol (address used to access the stream)
- DHCP — Dynamic Host Configuration Protocol (network IP assignment)
- mm — millimetre(s) (distance unit)
- VL53L1X — STMicroelectronics ToF distance sensor model

4. Future Improvements

The Nicla Vision camera is intended to be mounted on a mobile robot in a larger project: detected obstacles and distance readings will be sent to the robot's control system to enable obstacle avoidance behaviors (stop, steer around, or re-route).

For future development, sending only obstacle info (distance and size/position) over serial communication is recommended, as streaming can lag and is vulnerable to unstable Wi-Fi.



5. Success criteria

1. Deliverables

    1.1. Working Nicla Vision script -> yes

    1.2. Step-by-step setup instructions (firmware update, upload, network setup) -> yes

    1.3. Test results with examples and analysis -> yes

2. Success criteria

    2.1. Reliable detection of the nearest dark object -> sometimes

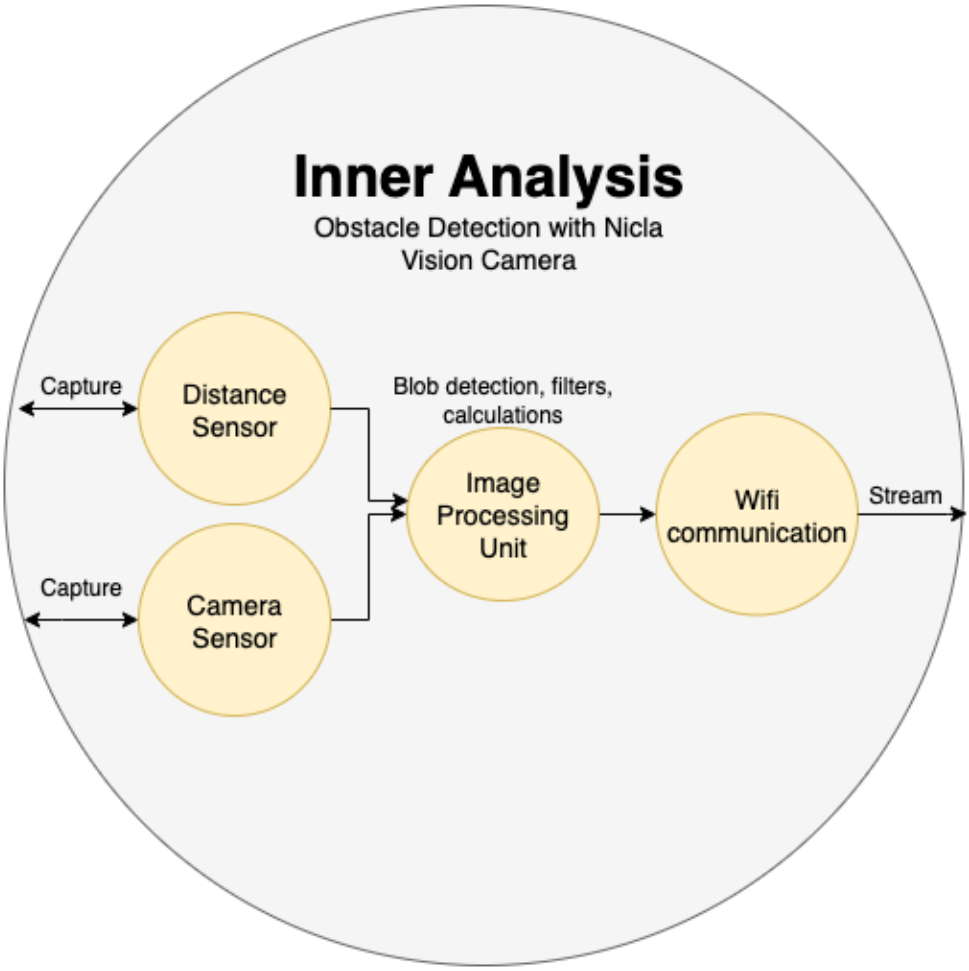
2.2. Reliable distance readings within the specified accuracy range -> yes

2.3. Continuous Wi-Fi streaming -> mostly

6. System Architecture

6.1 Overview

The system consists of an integrated distance and camera sensor on the Nicla Vision, which executes the object detection algorithm locally. Results are sent over the connected Wi-Fi to a specified port and IP address.



Detection system architecture

6.2 Algorithm

An algorithm had to be chosen for detecting obstacles and sending the information to a user or server. Blob detection was chosen because it is lightweight, fast and easy to tune for high-contrast (dark) objects. It runs efficiently on the OpenMV / Nicla platform compared to heavier ML methods, which helps maintain real-time FPS and reduces power use. Blob detection works by defining "dark" as the average background brightness plus an offset and grouping regions of similar luminance together into blobs. To handle changing lighting, the background brightness is recalculated every frame, improving reliability without complex preprocessing. The code for the blob detection algorithm is based on the Arduino Nicla Vision blob detection tutorial.

When several blobs/obstacles are present, one has to be selected as the nearest, but there is no per-pixel depth available from the ToF sensor. The design solution is to select the blob closest to the image center, where the ToF distance sensor has its line-of-sight, and hence where the distance reading is most relevant.

Having chosen blob detection further motivated using grayscale, because it reduces data size and preserves luminance contrast, which the blob detector uses. Processing grayscale frames is faster and requires less memory/CPU than color, improving real-time performance on the Nicla Vision. QVGA (320×240) resolution was chosen as a compromise between spatial detail and throughput: it gives enough image resolution for obstacle detection while keeping processing and network transmission load low to meet FPS targets.

6.3 Communication

Wi-Fi streaming was chosen primarily for convenience and visualization and is based on the Arduino Nicla Vision Wi-Fi streaming tutorial. The measured distance and a bounding box are drawn onto each Wi-Fi frame, marking the nearest object and its proximity. This is the easiest way to visualize and validate obstacle detection in a browser during development and demonstrations. However, it adds latency and can be unstable. The bandwidth is limited to 2.4 GHz, which affects the JPEG quality that can be transmitted. For robot integration, a simple serial transmission of obstacle data (distance + object size/position) is recommended.

7. Results and System Performance

This section is divided into three parts: object detection performance, distance accuracy and Wi-Fi streaming performance, and evaluates the success criteria in relation to each part.

7.1 Object detection Performance

7.1.1 Single Dark Object Detection

Current Status: Working

Selects the object closest to the center of the image, where the distance sensor is the most accurate.

Dark object detection with distance measurement

7.1.2 Bright Object Detection

Current Status: Not supported.

Issue: The system focuses on dark objects. In the example image below, light-coloured glasses are placed in front of a dark chair, but the algorithm identifies the chair as the closest obstacle because:

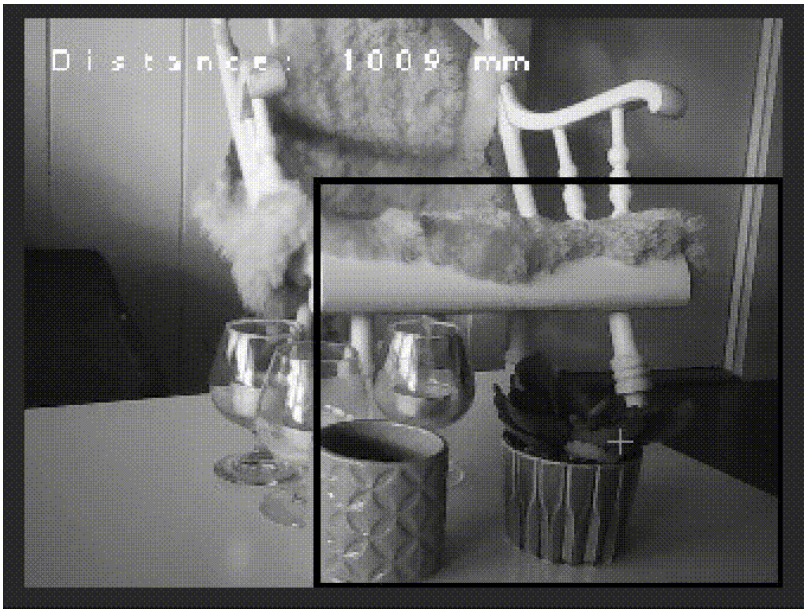
- It prioritizes dark regions
- The chair is closer to the image center (where the distance sensor is most accurate) than the dark plant.

Design Decision: Earlier code versions included a toggle between dark and bright object detection. However, bright-object detection frequently misidentified walls, windows and background elements as targets. To improve reliability, the feature was removed and the system now focuses on dark object detection.

Bright object detection example

7.1.3 Multiple Objects

Current Status: Partial



Live video demonstration of multiple object detection

Issue: The system struggles with stability when multiple objects are present. It frequently switches between different objects because the algorithm cannot determine which object is actually closest — it only identifies which object is closest to the image center and dark relative to the background. When the camera moves or lighting conditions change slightly, the detected "closest" object switches unpredictably between available targets, causing unstable detection.

7.2 Distance measurement accuracy

Limitation: The camera has a single ToF distance sensor that measures distance at the center point only. This creates the impression that the entire image has the same depth as the center point. While the distance reading is accurate when the closest object lies in the center, it might not precisely reflect the distance when the closest object is positioned elsewhere.

7.2.1 Close Range Performance

- **Minimum distance**: 40 mm (4 cm) as specified in the VL53L1X datasheet
- **Issue**: Objects placed closer than 40 mm give unreliable readings.

7.2.2 Long Range Performance

- **Maximum range**: The sensor supports up to ~4 m theoretically; practical implementation range here is limited to ~2 m.
- **Hardware limitation**: Performance degrades in low-light conditions (see the VL53L1X datasheet).
- **Algorithm limitation**: At longer distances, the blob detection algorithm struggles to identify which detected object corresponds to the center-point ToF measurement.

7.3 Streaming Performance

Current Status: The Wi-Fi stream experiences intermittent lag that can affect real-time monitoring.

Lag Sources:

- **Network limitation**: Use of the 2.4 GHz Wi-Fi band limits bandwidth.
- **Processing overhead**: Real-time image calculations (blob detection, distance measurement, visualization) consume processing time.
- **Image transmission**: Converting and streaming frames over HTTP adds latency.

Optimization Strategy:

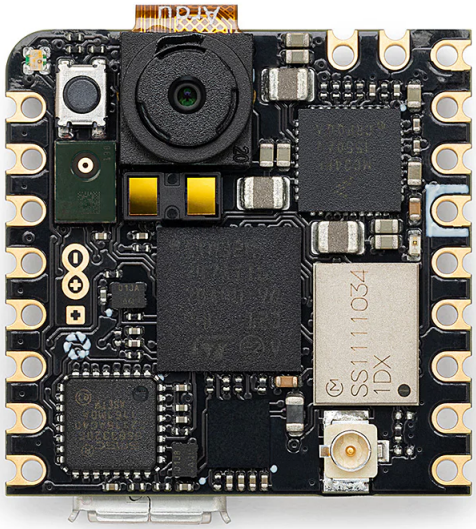
- **Grayscale format**: Reduces data size compared to color images.
- **QVGA resolution**: Lowers processing and transmission load.
- **Reduced JPEG quality**: Balances stream responsiveness with visual clarity.

Trade-off: There is a balance between smooth streaming performance and preserving enough image quality for accurate object detection.

8. Prerequisites

8.1 Hardware Requirements

- Arduino Pro Nicla Vision
- USB micro-B cable for programming and power
- Wi-Fi network with 2.4 GHz band for streaming
- Computer (Windows / Linux / Mac)

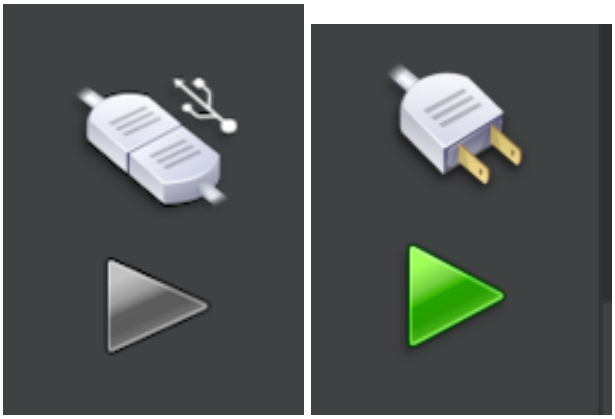


Arduino Pro Nicla Vision

8.2 Install OpenMV IDE

Download & install OpenMV IDE (Windows / Linux / Mac).

1. Connect your Nicla Vision via USB to your computer.
2. Open the OpenMV IDE and connect the camera by clicking the plug icon (Ctrl+E). The play button turns green when connected.



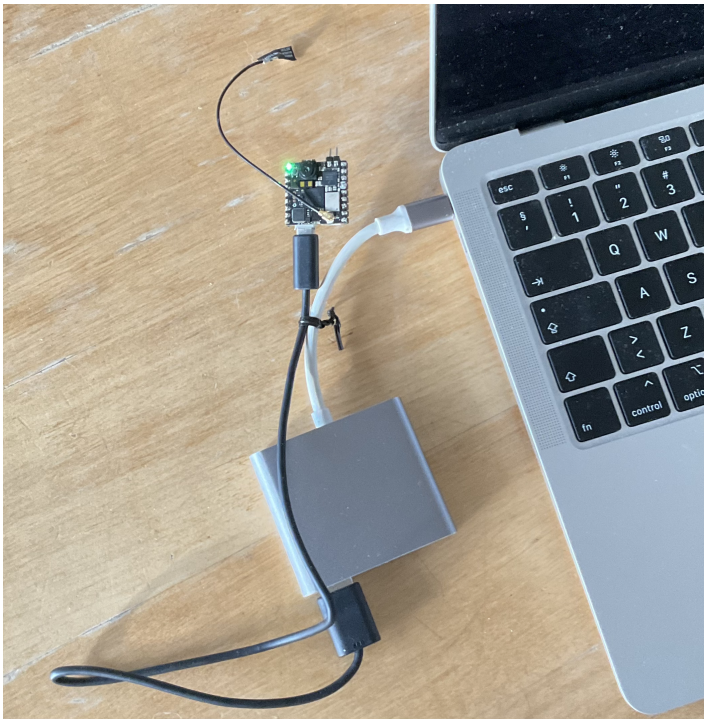
3. If prompted, update the firmware → install OpenMV firmware on the Nicla Vision.
4. Test by running an example: File → Examples → OpenMV → HelloWorld

9. How to Run

9.1 Without Wi-Fi Streaming

1. Open `ttk8.py` in OpenMV IDE and set `ENABLE_WIFI_STREAMING = False`.

2. Connect your Nicla Vision to the computer with a USB cable and hit play — it will start object detection and the LED turns green. The stream is shown in OpenMV.



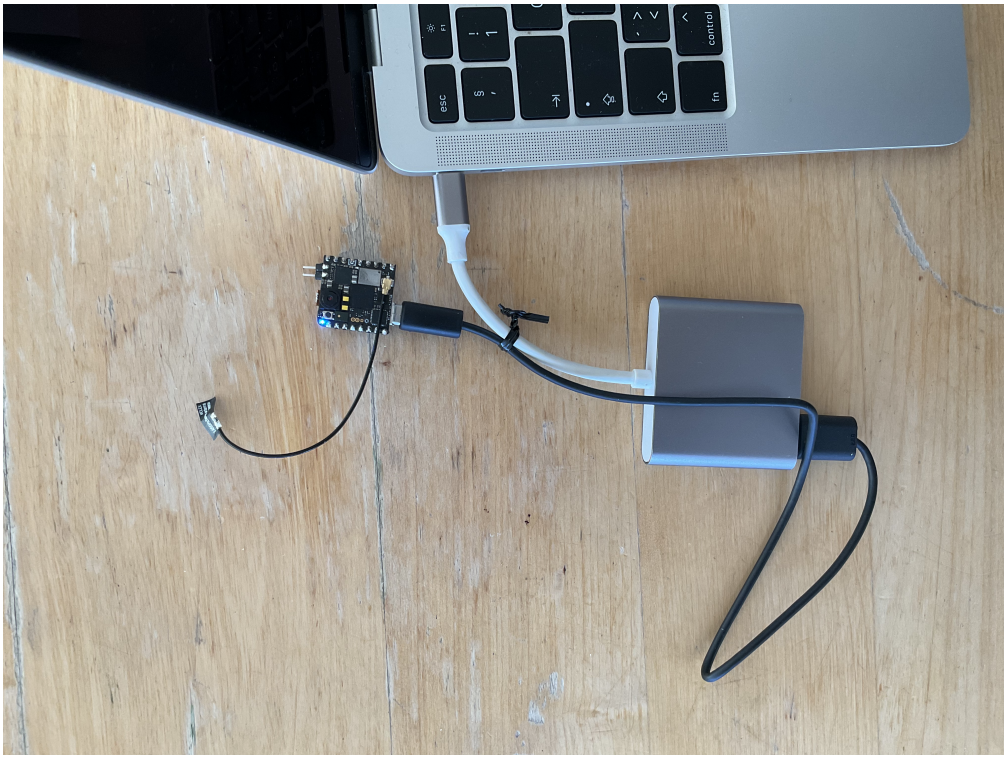
9.2 With Wi-Fi streaming

1. Open `ttk8.py` in OpenMV IDE and modify these parameters. Ensure your Wi-Fi supports the 2.4 GHz band:

```
ENABLE_WIFI_STREAMING = True  # Or False to disable wifi streaming
WIFI_NAME = "your_network_name"
WIFI_KEY = "your_password"
```

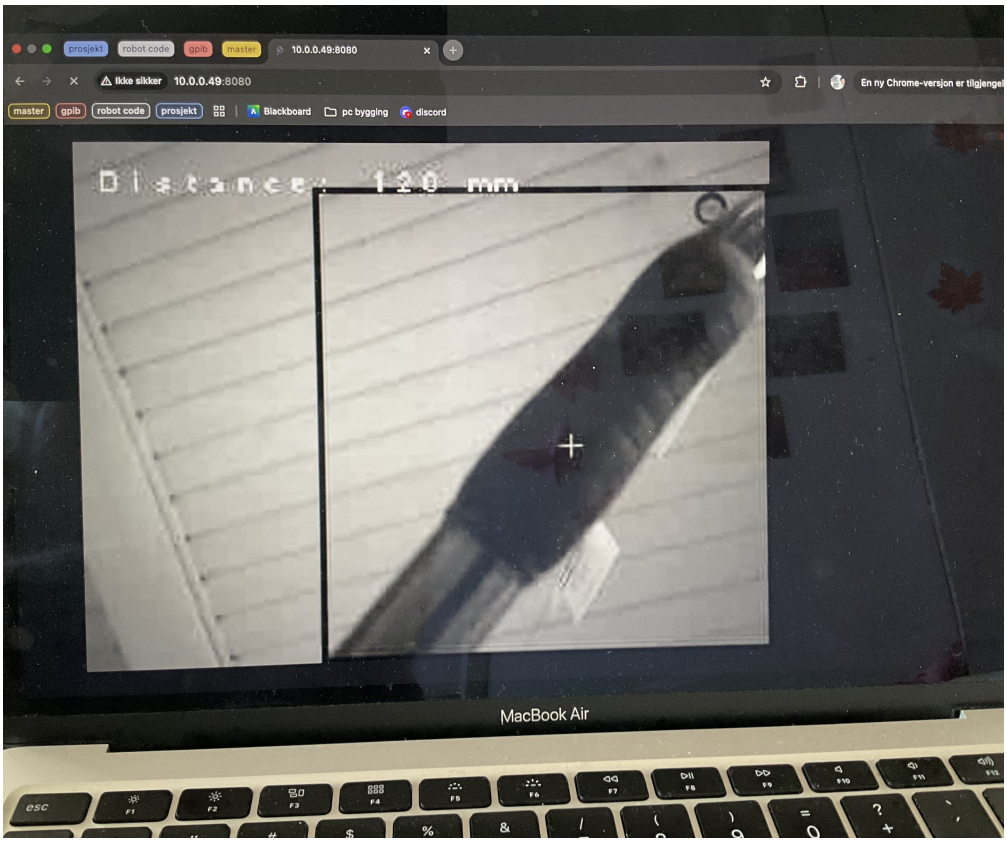
This is where most issues occur. See 9.3 Troubleshooting Wi-Fi Connection for solutions.

2. Connect your Nicla Vision to the computer with a USB cable. In the OpenMV IDE click connect (plug icon) and hit play. The LED turns blue during network setup.

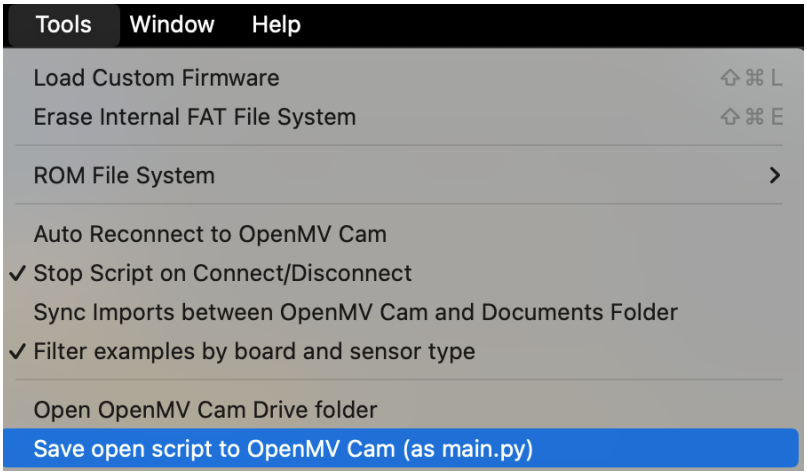


3. The terminal output will tell you to open a browser and access the stream at a certain IP and port. When this is done, the stream starts and the LED turns green.

```
1. Activate a mobile hotspot or use wifi router
2. If not activated: Set the band to be 2.4 GHz
3. Change the wifi name and passkey parameters to match the wifi

Trying to connect to network "Volvevegen_2G"
with passkey "Volvevegen"

Status: 1
Status: 1
Status: 3
WiFi Connected

Open a browser and enter http://10.0.0.49:8080/

Waiting for user connection...
```



4. When the stream works, upload the code to the Nicla Vision: Tools → Save open script to OpenMV Cam.



5. Disconnect the camera from the computer and connect it to a power source (outlet or battery).

- The camera will automatically connect to the specified network and the LED will be blue.
- Open a browser and navigate to the stream. The stream starts when a browser connects and the LED turns green.
- View the live stream with obstacle detection.

9.3 Troubleshooting Wi-Fi Connection

1. Test Wi-Fi streaming while the camera is connected to the computer so you can see terminal output in OpenMV for debugging.
2. Check that your network supports the 2.4 GHz band.
   - On macOS: Option + click the Wi-Fi icon and check the network channel.
   - On Windows: Settings → Network & Internet → Wi-Fi → click current network and check the band.
3. If your network does not support 2.4 GHz, enable 2.4 GHz on the router or use a mobile hotspot configured for 2.4 GHz.
4. If your browser blocks the HTTP stream with a "Not Secure" warning, click "Advanced" and select "Proceed to [IP address] (unsafe)" or use a different browser.

10. Tips & Tricks

If you divide the code into multiple files in OpenMV, include errors may occur.

To avoid this, copy the Python files you want to include directly onto the Nicla Vision camera's internal storage (when connected via USB it appears as a removable drive). Copy the files there so the camera can import them locally.

11. Appendix

11.1 References

- Arduino Nicla Vision datasheet
- Arduino Nicla Vision blob detection tutorial
- Arduino Nicla Vision Wi-Fi streaming tutorial
- VL53L1X Datasheet
- OpenMV Documentation

11.2 Code

```
import sensor, time, pyb
from machine import I2C
from vl53l1x import VL53L1X

# Enable wifi
WIFI_STREAMING = True # Set to False to disable wifi streaming
WIFI_NAME = "Volvevegen_2G"
WIFI_KEY = "Volvevegen"

# Parameters BLOB DETECTION
OFFSET = 30  # threshold offset around mean background brightness
min_area = 300       # ignore tiny blobs
min_pixels = 300     # ignore tiny blobs
max_fraction = 0.95   # ignore blobs covering more than 90% of image (not
working?)

# Camera setup
sensor.reset()
sensor.set_pixformat(sensor.GRAYSCALE)  # grayscale for detection
sensor.set_framesize(sensor.QVGA) # smaller frame size for speed
sensor.skip_frames(time=500)

# Distance sensor setup (ToF = time of flight)
i2c = I2C(2)
tof = VL53L1X(i2c)
MIN_VALID_DISTANCE = 40  # mm
MAX_VALID_DISTANCE = 2000 # mm

# LEDs
red = pyb.LED(1)
green = pyb.LED(2) # streaming video
blue = pyb.LED(3)  # setting up stream

def leds_off():
    red.off()
    green.off()
    blue.off()

clock = time.clock()
```

```
def wifi_setup(name, key): # returns wifi client
    import network, socket

    # User instructions
    print('\n1. Activate a mobile hotspot or use wifi router')
    print('2. If not activated: Set the band to be 2.4 GHz')
    print('3. Change the wifi name and passkey parameters to match the wifi\n')

    # WiFi connection parameters
    HOST = ""
    PORT = 8080

    # Init wlan module and connect to network
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(name, key)
    # Set statisk IP: (IP, netmask, gateway, DNS)
    #wlan.ifconfig(('192.168.2.30', '255.255.255.0', '192.168.2.1', '8.8.8.8'))

    print('Trying to connect to network "{:s}"'.format(name))
    print('with passkey "{:s}"\n'.format(key))

    while not wlan.isconnected():
        time.sleep(1)
        print("Status:", wlan.status())

    print("WiFi Connected\n")
    print("Open a browser and enter http://{:s}:{:d}/ \n".format(wlan.ifconfig()
[0], PORT))

    # Create server socket
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    s.bind([HOST, PORT])
    s.listen(5)
    s.setblocking(True)

    print("Waiting for user connection...")
    client, addr = s.accept()
    client.settimeout(5.0)
    print("Connected to " + addr[0] + ":" + str(addr[1]))

    # Read request from client
    data = client.recv(1024)

    # Send multipart header
    client.sendall(
        "HTTP/1.1 200 OK\r\n"
        "Server: OpenMV\r\n"
        "Content-Type: multipart/x-mixed-replace;boundary=openmv\r\n"
        "Cache-Control: no-cache\r\n"
        "Pragma: no-cache\r\n\r\n"
    )
    return client
```

```
def wifi_stream_frame(client, img):
    # Convert to JPEG for streaming
    cframe = img.to_jpeg(quality=35, copy=True)
    header = (
        "\r\n--openmv\r\n"
        "Content-Type: image/jpeg\r\n"
        "Content-Length:" + str(cframe.size()) + "\r\n\r\n"
    )
    client.sendall(header)
    client.sendall(cframe)

def detect_obstacles(wifi_client=None):
    clock.tick()
    img = sensor.snapshot()

    ## DISTANCE SENSOR
    dist = tof.read()  # Read distance in mm

    # Dynamic background brightness
    stats = img.get_statistics()
    mean_val = stats.l_mean()
    limit = max(0, mean_val - OFFSET)
    color_thresholds = [(0, limit)]

    # Find dark blobs
    blobs = img.find_blobs(color_thresholds, pixels_threshold=min_pixels,
area_threshold=min_area)

    if blobs and dist > MIN_VALID_DISTANCE and dist < MAX_VALID_DISTANCE:
        # Center of the image (where ToF points)
        center_x = img.width() // 2
        center_y = img.height() // 2

        # Filter blobs to ignore very large ones
        max_pixels = int(img.width() * img.height() * max_fraction)
        valid_blobs = [b for b in blobs if b.pixels() < max_pixels]

        # Find the blob covering the distance sensor's line-of-sight (center)
        target = None
        for b in valid_blobs:
            if b.pixels() < (img.width() * img.height() * max_fraction):
                if b.x() <= center_x <= b.x()+b.w() and b.y() <= center_y <=
b.y()+b.h():
                    target = b
                    break
        if not target:
            # If no blob covers the center, find the nearest blob
            nearest = None
            nearest_dist = float('inf')
            for b in blobs:
                # Calculate distance from blob center to image center
                dx = b.cx() - center_x
                dy = b.cy() - center_y
```

```
                distance = (dx*dx + dy*dy)**0.5
                if distance < nearest_dist:
                    nearest = b
                    nearest_dist = distance
            target = nearest

        if target:
            # Draw target square and distance
            img.draw_cross(target.cx(), target.cy(), color=(0, 255, 0))
            img.draw_rectangle(target.rect(), color=(0, 0, 0), thickness=3)
            img.draw_string(10, 10, f"Distance: {dist} mm", color=(255, 255,
255),scale=1.5)

    ## WIFI STREAMING
    if WIFI_STREAMING and wifi_client:
        wifi_stream_frame(wifi_client, img)

try:
    blue.on()
    wifi_client = (wifi_setup(WIFI_NAME, WIFI_KEY) if WIFI_STREAMING else None)
    blue.off()
    green.on()
    print("Offset mm, Object width mm, Distance mm")
    while True:
        detect_obstacles(wifi_client)
except:
    leds_off()
    print("\nProgram finished\n")
```