Miskolci Egyetem  
Gépészmérnöki és Informatikai Kar

Beszámoló dolgozat a nyári gyakorlatról

Készítette: Kamarás Erik  
Neptun kód: EN4B5K  
Dátum: 2025. 09. 11.

# Bevezetés

A nevem Kamarás Erik, a Miskolci Egyetem Mérnökinformatikus alapképzési szakának hallgatója vagyok. A kötelező 8 hetes nyári szakmai gyakorlatomat a Fux Zrt.-nél töltöttem el.

A gyakorlatom ideje alatt szerettem volna betekintést nyerni a cég minőségellenőrzési folyamataiba, melyet a helyi laboratóriumban zajló vizsgálatokon keresztül szerettem volna megvalósítani. Célom volt továbbá megismerkedni a különféle ellenőrzési és mérési módszerekkel.

A gyakorlatom idejében lehetőségem nyílt aktívan részt venni egy konkrét projektben is - melyet a dolgozatomban részletesen kifejtek - amely a vezetékek nedvesíthetőségének vizsgálatára fókuszált.

*„A FUX Zrt. a modern és fenntartható elektromos vezetékgyártás elkötelezett hazai szereplője.”*

**FUX ZRT. TEVÉKENYSÉGI KÖRE**

A **FUX Zrt.** (FUX Fémszerelvény- és Kábelgyár Zártkörűen Működő Részvénytársaság) több évtizedes múltra tekint vissza, - 1991-ben alakult és ekkor még Kft-ként működött - elsősorban **villamosenergia-ipari termékek,** például **szigetelt és csupasz vezetékek,** sodronyok, valamint egyéb kábelrendszerek gyártásával és forgalmazásával foglalkozik. Termékei nemzetközi piacokon is jelen vannak, több, mint 20 országba exportál. Alkalmazottai száma meghaladja a 220 főt. Folyamatos fejlődését bizonyítja, hogy **2025.** májusban megvásárolták az Omco International bezárt miskolci gyártóüzemének területét, mely lehetőséget nyújt új csarnokok, raktárak építéséhez.

A vállalat nagy hangsúlyt fektet a **minőségbiztosításra**, a **folyamatos innovációra**, valamint a **környezetvédelemre**.

A FUX Zrt. saját, két műszakban működő laboratóriummal rendelkezik, amelyet tapasztalt és jól képzett laboránsok működtetnek. A labor modern mérő- és vizsgálóberendezésekkel van felszerelve, valamint kézi mérőeszközök is rendelkezésre állnak.

A vállalat elkötelezett a minőségirányítás mellett az **ISO 9001** szabvány szerint, emellett a labor jelenleg az **MSZ EN ISO/IEC 17025:2018** akkreditáció megszerzésén dolgozik – ez a laboratórium-specifikus nemzetközi szabvány a vizsgálati és kalibrálási tevékenységekre. A vizsgálatok nemzeti és nemzetközi szabványok szerint zajlanak.

A feladatom elkészítéséhez szükséges vezetékeket a Fux Zrt. szolgáltatta tesztelés céljából.   
A vezetékek a típusai:

* Felület kezeletlen
* Festett
* Homokszórt
* Régi (használt)

Képek a vezetékekről sorrendben:



A vezetékek nedvesíthetőségének vizsgálatára az úgynevezett „vízpermet próbát” alkalmazzák, amelyet az IEC TS 62073 szabvány ír le. Ez a szabványosított vizsgálati eljárás lehetővé teszi annak értékelését, hogy a vezetékek felülete mennyire képes elvezetni vagy megtartani a nedvességet. A teszt során a vezetékeket meghatározott ideig és intenzitással vízpermetnek teszik ki, majd a kialakult vízfilm viselkedését, valamint a száradási jellemzőket elemzik. Ez a módszer fontos szerepet játszik a vezetékek hosszú távú megbízhatóságának és biztonságos működésének értékelésében, különösen nedves vagy párás környezetben való alkalmazás esetén.

Az ehhez használt folyadékok a Csapvíz, Desztillált víz és közönséges Esővíz.

# A Python programnyelv

A **Python** egy magas szintű, általános célú, **interpreteres** programozási nyelv, amelyet Guido van Rossum fejlesztett ki az 1990-es évek elején. A fő célja az volt, hogy a nyelv **könnyen olvasható és tanulható** legyen, ugyanakkor erőteljes eszköz a fejlesztők kezében. A Python név nem a kígyóra, hanem a **Monty Python** nevű brit humorcsoportra utal.

A Python szintaxisa tiszta, kevés speciális karaktert használ. Nincs szükség sorvégi pontosvesszőre (;), mint a C, Java vagy JavaScript esetében. A kódblokkokat kapcsos zárójelek helyett behúzás határozza meg.

Nem kell előre meghatározni a változók típusát sem. (Pl.: int, float)

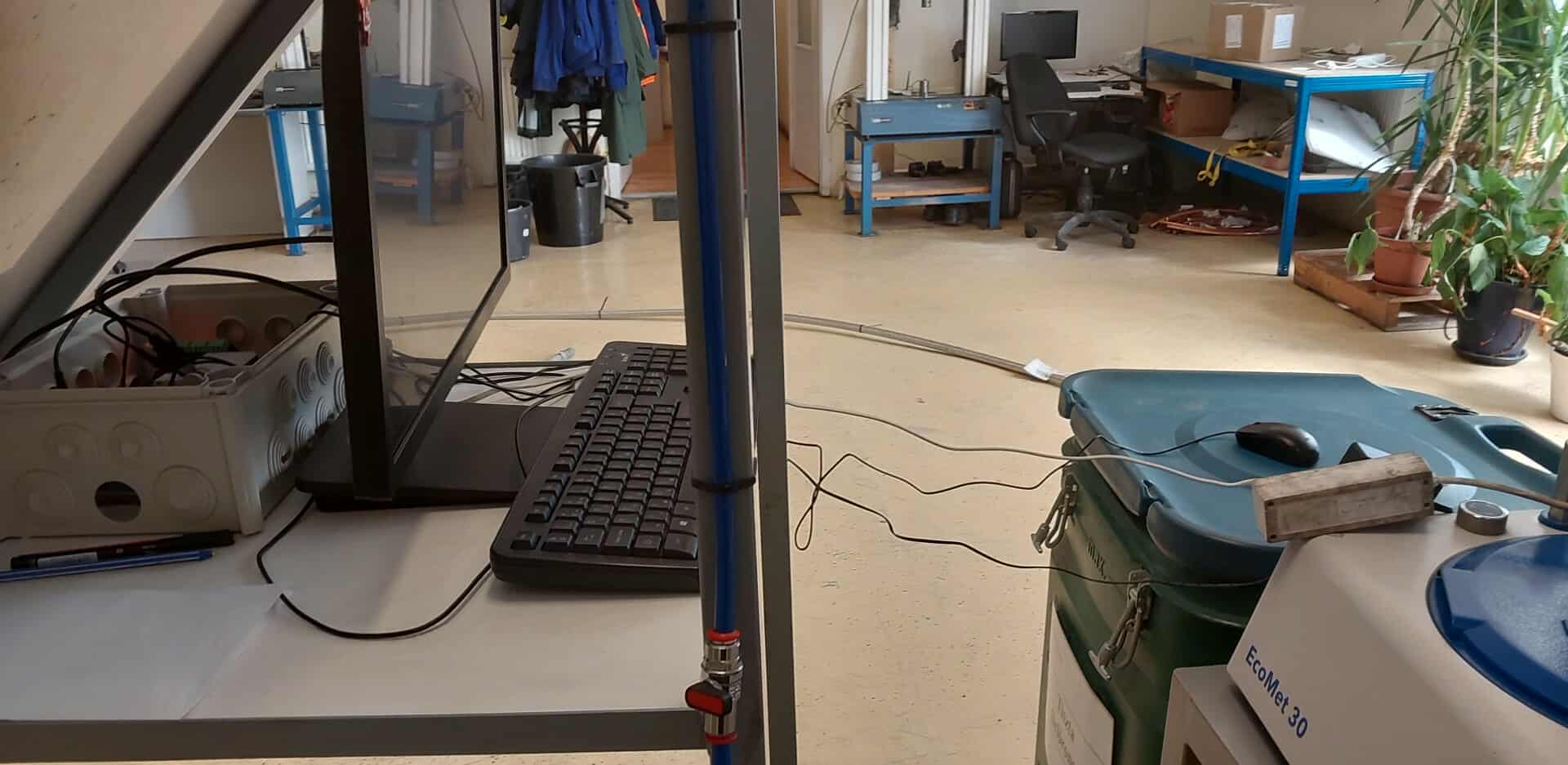
Python kód futtatható Windows, Linux, macOS rendszereken – általában változtatás nélkül, ezért volt különösen hasznos az én feladatomhoz is.

# Első feladat: Raspberry Pi

Az első feladatom a már imént említett Vízpermet próbához használt gép beüzemelése volt (4 hét).

A vezeték tartó alatt található 2 raspberry pi számítógép, 1 darab monitor, billentyűzet és egér, az állványhoz rögzített 2 db pi camera, valamint az ezeket összeköttető vezetékek álltak rendelkezésemre.





A két gépre ettől a ponttól fogva úgy fogok hivatkozni, hogy A és B gép és kamera. (Bal és Jobb oldal szemből)

Feladatom az volt, hogy a 30 másodperces áztatás után, a program futtatását követően, 10 másodpercen belül készüljenek képek mindkettő kamerával szinkronban, majd ezek a képek kerüljenek lekövethetően mentésre.

3 scriptet készítettem ennek a feladatnak a megoldásához.

Az első, a **picamera.py**, csak a kamerák felrögzítéséhez és a fókusztávolság beállításához volt szükséges. Ez a script egy előre megadott ideig kivetíti a monitorra a kamera elő nézetét.

import io

import picamera

from time import sleep

camera = picamera.PiCamera()

camera.start\_preview()

sleep(120)

camera.stop\_preview()

Tervben volt egy killswitch implementálása is, de mivel ez a script megtalálható az A és B gépen is, és egyszerre csak az egyik tudja használni az adott egeret vagy billentyűzetet, redundánsnak találtam és egyszerűbbnek, ha magától leáll.

A második script felel a képek elkészítéséért. A **pisave.py** script feladata az, hogy minél gyorsabban elkészítsen 10 db „.jpg” formátumú képet.

Azt a kérést kaptam, hogy ha a jövőben szükség lenne videó készítésére is, akkor a program már csinálja meg azt is. Mivel a kamerák nem képesek egyszerre fotók és videók készítésére, ezért a képek élveznek prioritást és habár kisebb eltolódással, de a videó is rögtön elkészül, de csak a fotók után.

A képeket és a videót a script az asztalra menti, a képek mozgatásáért és másolásáért már majd a harmadik script felel.

Az elkészült képek neve sorban image1.jpg, 10.-ig.

import io

import time

import picamera

from datetime import datetime

from PIL import Image

timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')

h264\_file = f"/home/pi/Desktop/video\_{timestamp}.h264"

mp4\_file = f"/home/pi/Desktop/video\_{timestamp}.mp4"

with picamera.PiCamera() as camera:

    # Set the camera's resolution to VGA @40fps and give it a couple

    # of seconds to measure exposure etc.

    camera.resolution = (1920, 1080)

    camera.framerate = 15

    camera.rotation = 0

    time.sleep(2)

    camera.start\_recording(h264\_file)

    # Set up 40 in-memory streams

    outputs = [io.BytesIO() for i in range(10)]

    start = time.time()

    camera.capture\_sequence(outputs, 'jpeg', use\_video\_port=True)

    finish = time.time()

    time.sleep(8)

    camera.stop\_recording()

    # How fast were we?

    print('Captured 10 images at %.2ffps' % (10 / (finish - start)))

    count = 0

    for frameData in outputs:

        rawIO = frameData

        rawIO.seek(0)

        byteImg = Image.open(rawIO)

        count += 1

        filename = "image" + str(count) + ".jpg"

        byteImg.save(filename, 'JPEG')

    camera.close()

A harmadik script (starting\_script.py) felel a pisave szinkornizált indításáért, a képek hálózaton keresztüli másolásáért, és a tároló mappák létrehozásáért.

Mivel ez a futtatandó script, ez is a leghosszabb.  
Ezért ezt csak részekre tördelve mutatom be.

import subprocess

import time

import datetime

import os

import glob

# === Config ===

REMOTE\_HOST = "pi@192.168.25.82"

REMOTE\_SCRIPT = "/home/pi/Desktop/pisave.py"

LOCAL\_SCRIPT = "/home/pi/Desktop/pisave.py"

REMOTE\_IMAGE\_PATTERN = "/home/pi/image\*.jpg"

LOCAL\_IMAGE\_PATTERN = "/home/pi/Desktop/image\*.jpg"

DESTINATION\_ROOT = "/home/pi/Desktop/"

VIDEO\_PATTERN = "/home/pi/Desktop/video\_\*.h264"

start = time.time()

A fenti kódrészben a globális változók találhatóak, amellyek többnyire a fájlok tartózkodási helyét tartalmazzák. A „REMOTE\_HOST” a B számítógép ip címe ami majd az ssh és scp protokollokhoz kell.

# === Step 1: Start local script ===

print("Starting local pisave.py...")

local\_proc = subprocess.Popen(["python3", LOCAL\_SCRIPT])

# === Step 2: SSH into remote and run remote script ===

print("Starting remote pisave.py...")

result = subprocess.run([

    "ssh", REMOTE\_HOST,

    f"python3 {REMOTE\_SCRIPT}"

    ], capture\_output=True, text=True, check=True)

# === Step 3: Wait for local process(es) to complete ===

print("Waiting for local capture to finish...")

local\_proc.wait()

end = time.time()

print("TIME:", end - start)

Ez a rész felel a kettő subprocess szinkronizált futtatásáért. A script addig vár, amíg mind a kettő gép be nem fejezte a képek készítését.

# === Step 4: Making a folder for A pictures

timestampA = datetime.datetime.now().strftime("%Y%m%d\_%H%M%S" + " A")

dir\_name = os.path.join(DESTINATION\_ROOT, timestampA)

os.makedirs(dir\_name, exist\_ok=True)

print(f"Created folder A: {dir\_name}")

# === Step 5: Create timestamped directory for B pictures ===

timestampB = datetime.datetime.now().strftime("%Y%m%d\_%H%M%S" + " B")

dest\_dir = os.path.join(DESTINATION\_ROOT, timestampB)

os.makedirs(dest\_dir, exist\_ok=True)

print(f"Created folder B: {dest\_dir}")

# === Step 6: Copy images to local folder

print("Copying images from local...")

subprocess.run([

    f"cp {LOCAL\_IMAGE\_PATTERN} '{dir\_name}'",

], shell=True, check=True)

# === Step 7: Copy images to local folder

print("Copying video from local...")

subprocess.run([

    f"cp {VIDEO\_PATTERN} '{dir\_name}'",

], shell=True, check=True)

# === Step 8: Delete pictures from desktop

print("Cleaning up local images...")

subprocess.run([

     f"rm -f {LOCAL\_IMAGE\_PATTERN}"

    ], check=True, shell=True)

print("Local images cleaned up")

# === Step 9: Delete video from desktop

print("Cleaning up local video...")

subprocess.run([

     f"rm -f {VIDEO\_PATTERN}"

    ], check=True, shell=True)

# === Step 10: Pull images from remote to local ===

print("Copying images from remote...")

subprocess.run([

    "scp",

    f"{REMOTE\_HOST}:{REMOTE\_IMAGE\_PATTERN}",

    dest\_dir

], check=True)

A 4.-10. lépésekben történik a másolása a képeknek a B gépről az A-ra. A képek egy mappába kerülnek aminek a neve a jelenlegi dátum + a származási gép (Pl.: 20250714\_112856 A).

# === Step 11: Delete images from remote

print("Cleaning up remote images...")

subprocess.run([

    "ssh", REMOTE\_HOST, "rm -f /home/pi/Desktop/image\*.jpg"

    ], check=True)

print("Copying video from remote...")

subprocess.run([

    "scp",

    f"{REMOTE\_HOST}:{VIDEO\_PATTERN}",

    dest\_dir

], check=True)

print("Cleaning up remote video...")

subprocess.run([

    "ssh", REMOTE\_HOST, "rm -f /home/pi/Desktop/video\_\*.h264"

    ], check=True)

print("All done.")

A 11.-edik lépés a törlés.

# Második feladat: OpenCV

A második feladat, amit kaptam egy képfelismerő program készítése volt a python OpenCV könyvtár használatával. Ebben a könyvtárban sok előre elkészített függvény van, amik a képek és akár videók kezelésével foglalkozik.

Mivel a feladatom a vízcseppek felismerése volt egy vezeték felületén, ezért a hasonló nevű erre a célra készült ún. „watershed” függvényt vettem használatba. Segítségemre az OpenCV hivatalos weboldala volt, ami részletesen leírja ennek a beépített függvénynek a működését.

A vízcseppek felismerése ezért nem is okozott különösebb problémát, de a vezetékek felismerése a képeken igen. Rengeteg módszerrel próbálkoztam a különböző vastagságú, színű és elhelyezkedésű vezetékek pozíciójának felismerésénél. Végül a szín szerinti szűrésnél maradtam azért, hogy a script többi részét tesztelhessem sikeresen legalább 1 típusú vezetéken (A festett vezetékek teljesen feketék).

import os

from pathlib import Path

import cv2

import numpy as np

# Input folder with images, and output folder for results

INPUT\_DIR = "/home/erik/Desktop/Fux/Fux\_OpenCV/kepek/Csapvíz/Festett/20250725\_112100 A"  #Loop maybe instead? TODO

OUTPUT\_DIR = "/home/erik/Desktop/Fux/Fux\_OpenCV/out/v4"

Ezekben a mappákban találhatóak a képek, az útvonal globális változóként van letárolva.

def detect\_coverage\_noresize(

    bgr,

    wire\_morph\_close=11,   # close gaps (helical grooves)

    wire\_morph\_open=5,     # remove specks

    wire\_erode=2,          # shrink slightly

    spec\_s\_low\_q=0.30,     # specular: low saturation threshold

    spec\_v\_high\_q=0.90,    # specular: high brightness threshold

    gradient\_quantile=0.92,# droplet edges: top quantile of gradients

    min\_component\_px=12,   # ignore tiny noise blobs

    surebg\_dilate=7,       # expand sure background for watershed

    surefg\_erode=1,        # shrink sure foreground for watershed

    final\_dilate=2         # expand droplet result to full footprint                            TODO smaller? larger? idk

):

Ez a függvény az, amit a main majd meghív futáskor. Minden változó kézi értékadása itt történik meg.

 # --- 1) Wire segmentation ---

    hsv = cv2.cvtColor(bgr, cv2.COLOR\_BGR2HSV)

    V = hsv[:, :, 2]                                                                            #TODO other wires

    V\_blur = cv2.GaussianBlur(V, (7, 7), 0)

    # Invert brightness so dark wire appears bright, then Otsu threshold

    Vin = 255 - V\_blur

    \_, th = cv2.threshold(Vin, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

    # Pick largest connected component as wire

    num, labels = cv2.connectedComponents(th, connectivity=8)

    areas = [(labels == i).sum() for i in range(1, num)]

    wire\_mask = np.zeros\_like(th, dtype=np.uint8)

    if areas:

        k = 1 + int(np.argmax(areas))

        wire\_mask[labels == k] = 255

    # Morphological smoothing/refinement

    if wire\_morph\_close > 0:

        kclose = cv2.getStructuringElement(cv2.MORPH\_RECT, (wire\_morph\_close, wire\_morph\_close // 3))

        wire\_mask = cv2.morphologyEx(wire\_mask, cv2.MORPH\_CLOSE, kclose)

    if wire\_morph\_open > 0:

        kopen = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (wire\_morph\_open, wire\_morph\_open))

        wire\_mask = cv2.morphologyEx(wire\_mask, cv2.MORPH\_OPEN, kopen)

    if wire\_erode > 0:

        ker = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (2 \* wire\_erode + 1,) \* 2)

        wire\_mask = cv2.erode(wire\_mask, ker)

    wire\_mask = (wire\_mask > 0).astype(np.uint8)

    if wire\_mask.sum() == 0:

        raise RuntimeError("Wire mask is empty.")

Az első lépés a wiremask.png legenerálása. A végeredmény kép jelöli, hogy a program a továbbiakban hol keresi majd a vízcseppeket.

Szín szerint keresünk (RGB: ( :, :, 2) *nagyon* *sötét kék*) majd **Otsu** algoritmusával fekete fehér kép és a legnagyobb egységes sík lesz a maszk.

# --- 2) Droplet candidate seeds ---

    gray = cv2.cvtColor(bgr, cv2.COLOR\_BGR2GRAY)

    gray\_blur = cv2.GaussianBlur(gray, (5, 5), 0)

    # Normalize HSV values

    S = hsv[:, :, 1].astype(np.float32) / 255.0

    Vf = hsv[:, :, 2].astype(np.float32) / 255.0

    wp = wire\_mask == 1  # wire pixels only

    # Specular highlights = low saturation & high brightness

    s\_low = float(np.quantile(S[wp], spec\_s\_low\_q))

    v\_high = float(np.quantile(Vf[wp], spec\_v\_high\_q))

    specular = ((S <= s\_low) & (Vf >= v\_high)).astype(np.uint8)

    # Gradient-based edges

    gx = cv2.Sobel(gray\_blur, cv2.CV\_32F, 1, 0, ksize=3)

    gy = cv2.Sobel(gray\_blur, cv2.CV\_32F, 0, 1, ksize=3)

    mag = cv2.magnitude(gx, gy)

    gthr = float(np.quantile(mag[wp], gradient\_quantile))

    gradmask = (mag >= gthr).astype(np.uint8)

    # Combine specular + gradient inside wire

    candidates = ((specular | gradmask) & wire\_mask).astype(np.uint8)

    # Remove tiny blobs

    num, labels = cv2.connectedComponents(candidates, connectivity=8)

    clean = np.zeros\_like(candidates, dtype=np.uint8)

    for i in range(1, num):

        if (labels == i).sum() >= min\_component\_px:

            clean[labels == i] = 1

Vízcseppek detektálása a vezetéken:

1. Keresés fényes, alacsony telítettségű (speculáris visszaverődés) részekre. (s\_low, v\_high)
2. Keresés erős gradiensű (élű) részekre, ahol a cseppek körvonalai látszanak.
3. Ezek kombinálása → kezdeti „csepp magok” (seeds).

# --- 3) Watershed segmentation ---

    # Build sure foreground/background from candidates

    kernel\_fg = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (2 \* surefg\_erode + 1,) \* 2)

    kernel\_bg = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (2 \* surebg\_dilate + 1,) \* 2)

    sure\_fg = cv2.erode(clean, kernel\_fg)

    sure\_bg = cv2.dilate((wire\_mask & (~clean)).astype(np.uint8), kernel\_bg)

    # Label markers: 1 = background, 2+ = droplets

    markers = np.zeros\_like(gray, dtype=np.int32)

    \_, lbl\_fg = cv2.connectedComponents(sure\_fg.astype(np.uint8))

    markers[sure\_bg == 1] = 1

    markers[sure\_fg == 1] = (lbl\_fg[sure\_fg == 1] + 1)

    # Run watershed on gradient image

    grad\_u8 = cv2.normalize(mag, None, 0, 255, cv2.NORM\_MINMAX).astype(np.uint8)

    grad\_rgb = cv2.cvtColor(grad\_u8, cv2.COLOR\_GRAY2BGR)

    cv2.watershed(grad\_rgb, markers)

    droplet\_ws = ((markers > 1) & (wire\_mask == 1)).astype(np.uint8)

cv2.watershed előtt erosion és dilation ahogy az a OpenCV weboldalán szereplő tutorial-ban is szerepel, így biztosra mehetünk abba, hogy ami maradt az vízcsepp!

# Expand result slightly (approximate droplet footprint)

    if final\_dilate > 0:

        kd = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (2 \* final\_dilate + 1,) \* 2)

        droplet\_ws = cv2.dilate(droplet\_ws, kd)

        droplet\_ws = (droplet\_ws & wire\_mask).astype(np.uint8)

    # --- 4) Coverage calculation ---

    wire\_area = int(wire\_mask.sum())

    wet\_area = int(droplet\_ws.sum())

    coverage = 100.0 \* wet\_area / max(1, wire\_area)

Mivel a víz látható nyomot hagy maga után ezért érdemesnek gondoltam azt is figyelembe venni.

Ezután fedettség kiszámítása (coverage = 100.0 \* wet\_area / max(1, wire\_area))

# --- 5) Visualization ---

    overlay = bgr.copy()

    # Darken the wire region for contrast

    overlay[wire\_mask == 1] = (

        0.6 \* overlay[wire\_mask == 1] + 0.4 \* np.array([40, 40, 40])

    ).astype(np.uint8)

    # Draw droplet contours in red

    contours, \_ = cv2.findContours(droplet\_ws, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

    cv2.drawContours(overlay, contours, -1, (0, 0, 255), thickness=2)

Vizualizáció. Ez lesz az overlay.jpg a piros markerekkel.

def categorize\_wc(coverage\_pct: float) -> str:

    """

    Convert numeric coverage (%) into WC category.

    WC4: < 10%

    WC5: 10–50%

    WC6: 90–99%

    WC7: 99–100%

    Anything else (50–90%) = WCx (currently undefined).

    """

    c = max(0.0, min(100.0, coverage\_pct))  # clamp into [0,100]

    if c < 10.0:

        return "WC4"

    if 10.0 <= c <= 50.0:

        return "WC5"

    if 90.0 <= c < 99.0:

        return "WC6"

    if c >= 99.0:

        return "WC7"

    return "WCx"  # placeholder for undefined band TODO

# Add text labels (top-left)

    vis = overlay.copy()

    font = cv2.FONT\_HERSHEY\_SIMPLEX

    scale, thick, color = 1.0, 2, (255, 255, 255)

    wc\_tag = categorize\_wc(coverage)

    cv2.putText(vis, f"Coverage: {coverage:.1f}%", (20, 40), font, scale, color, thick, cv2.LINE\_AA)

    cv2.putText(vis, wc\_tag, (20, 80), font, scale, color, thick, cv2.LINE\_AA)

Kategorizálás a wettability class (WC) szerint. Ez, a script tetején külön függvényben szerepel.

def main():

    in\_dir = Path(INPUT\_DIR)

    out\_dir = Path(OUTPUT\_DIR)

    out\_dir.mkdir(parents=True, exist\_ok=True)

    for fname in sorted(in\_dir.glob("image\*.jpg")):

        print(f"Processing {fname.name}...")

        bgr = cv2.imread(str(fname))

        if bgr is None:

            print("  Skipped (could not read).")

            continue

        try:

            coverage, wc\_tag, wire\_mask, droplets, vis = detect\_coverage\_noresize(bgr)

        except Exception as e:

            print(f"  Failed: {e}")

            continue

        # Save outputs

        base = out\_dir / fname.stem

        cv2.imwrite(f"{base}\_overlay.jpg", vis)

        cv2.imwrite(f"{base}\_wire\_mask.png", (wire\_mask \* 255).astype(np.uint8))

        cv2.imwrite(f"{base}\_droplets.png", (droplets \* 255).astype(np.uint8))

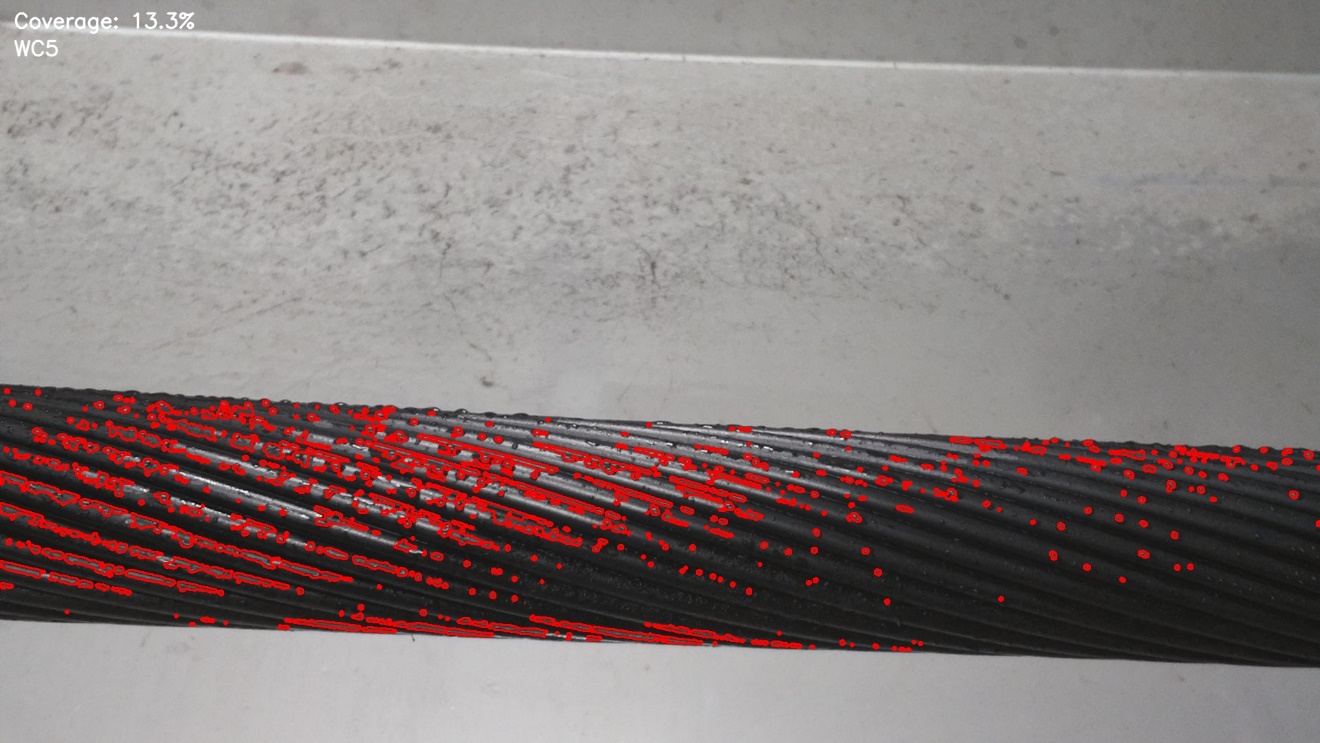
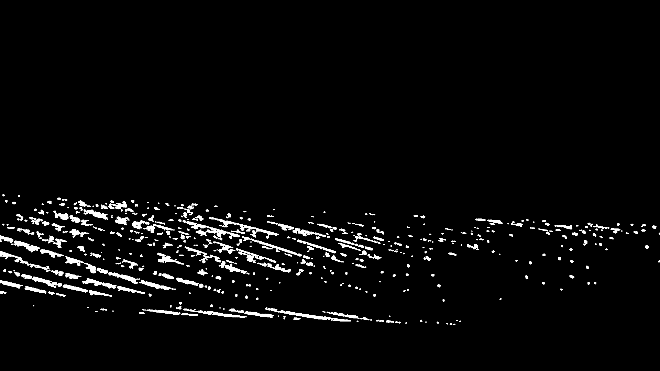
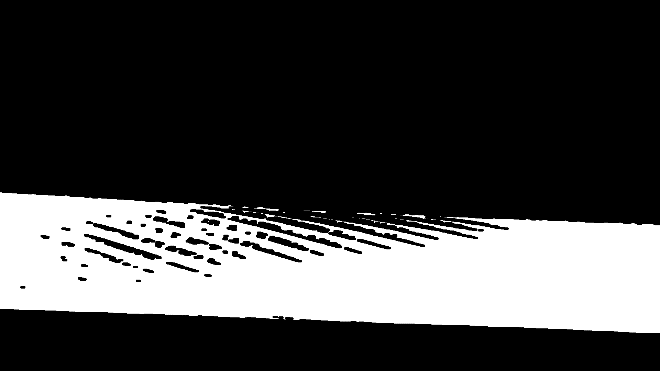
        print(f"  Coverage: {coverage:.2f}%  ->  {wc\_tag}  (saved {base}\_\*.jpg/png)")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Végül a main függvény, a megadott mappában minden image\*.jpg-re hasonlító fájlra lefut.

A végeredmény képek az out mappában találhatóak. Például:



# Összefoglaló

A gyakorlatom ideje alatt betekintést nyertem a cég minőségellenőrzési folyamataiba (Quality Assurance, röviden QA), különösen a helyi laboratóriumban zajló vizsgálatokon keresztül. Megismerkedhettem a különféle ellenőrzési és mérési módszerekkel, valamint a laborban alkalmazott szabványokkal és eszközökkel is. Emellett lehetőségem nyílt aktívan részt venni egy konkrét projektben is, amely a vezetékek nedvesíthetőségének vizsgálatára fókuszált.

A projekt során gyakorlati tapasztalatokat szereztem, megértettem az anyagvizsgálat és a minőségbiztosítás szerepét a gyártási folyamatokban.

# Hivatkozások

Fux Zrt. weboldala: <https://www.fux.hu/>

OpenCV tutorial: <https://docs.opencv.org/4.x/d3/db4/tutorial_py_watershed.html>

Rapsberry Pi kamera installáció és kód a vezérléshez: <https://projects.raspberrypi.org/en/projects/getting-started-with-picamera>

Github repository: <https://github.com/KamarasE/Fux_OpenCV>

Otsu algoritmusa: <https://en.wikipedia.org/wiki/Otsu%27s_method>

Sobel függvény: <https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html>

IEC TS 62073 szabvány: <https://cdn.standards.iteh.ai/samples/21232/5d1aedc6878f46aba2cff90a33e51efd/IEC-TS-62073-2016.pdf>