Fundusze Europejskie dla Rozwoju Społecznego | Rzeczpospolita Polska | Dofinansowane przez Unię Europejską

Politechnika Świętokrzyska
Wydział Mechatroniki i Budowy Maszyn

Kierunek studiów:
Automatyka i Robotyka

Krzysztof Borkowski

Materiały dydaktyczne do przedmiotu

Techniki Wizyjne i Przetwarzanie Obrazów

opracowane w ramach realizacji Projektu
**„Dostosowanie kształcenia
w Politechnice Świętokrzyskiej do potrzeb
współczesnej gospodarki”**
FERS.01.05-IP.08-0234/23

Kielce, 2025

Spis treści

Wstęp

Poradnik „Instalacja środowiska Python + VS Code + Jupyter (Windows 10/11)” stanowi rozszerzenie merytoryczne treści podstawowych przedmiotu Techniki wizyjne i przetwarzanie obrazów, umożliwiając studentom praktyczne przygotowanie narzędzi niezbędnych do realizacji zajęć.

Modyfikacje przedmiotu *Techniki Wizyjne i Przetwarzanie Obrazów* wprowadzono, aby lepiej przygotować studentów do pracy z nowoczesnymi narzędziami i technologiami.

Kluczowym czynnikiem determinującym wprowadzenie zmian było włączenie do programu studiów I stopnia na kierunku Automatyka i Robotyka nowego przedmiotu *Programowanie Skryptowe*, w ramach którego studenci nabywają podstawowe kompetencje w zakresie programowania w języku Python.

Podstawowa modyfikacja przedmiotu polega na przejściu z języka C++ na Python oraz wdrożeniu środowiska Jupyter Notebook i Visual Studio Code. Dzięki temu studenci uczą się przetwarzania obrazów w języku będącym standardem w branży data science, uczenia maszynowego i automatyzacji.

Wprowadzone modyfikacje obejmują:

1. Zmianę platformy programistycznej z C++ na Python, co umożliwia wykorzystanie bogatego ekosystemu bibliotek specjalistycznych (NumPy, OpenCV, Matplotlib, scikit-image).
2. Zastosowanie środowiska Jupyter Notebook w połączeniu z Visual Studio Code jako głównego narzędzia rozwoju, co zapewnia interaktywność, możliwość wizualizacji wyników oraz dokumentowania procesu analitycznego.
3. Rozszerzenie spektrum metodologicznego o nowoczesne techniki przetwarzania obrazów, które byłyby trudniejsze do implementacji w paradygmacie niskopoziomowym.
4. Integrację z kompetencjami nabytymi w ramach przedmiotu Programowanie skryptowe, w celu zapewnia spójność programu kształcenia i efektywne wykorzystanie wcześniej nabytej wiedzy

Wprowadzone modyfikacje pozwalają na efektywne wykorzystanie specjalistycznych bibliotek, a także umożliwiają szybsze prototypowanie i testowanie algorytmów oraz pracę w środowisku charakterystycznym dla współczesnej inżynierii oprogramowania.

Niniejszy materiał dydaktyczne powstały w ramach projektu FERS „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”.

Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

Szybki start

Rozdział ten przedstawia najważniejsze kroki niezbędne do uruchomienia środowiska programistycznego. Dzięki zwięzłym instrukcjom szybko rozpoczniesz pracę z narzędziami wykorzystywanymi w przedmiocie Techniki wizyjne i przetwarzanie obrazów.

1. Pobierz i zainstaluj Visual Studio Code (zaznacz „Add to PATH”).
2. Pobierz i zainstaluj Python 3.11.xx (zaznacz „Add Python to PATH”).
3. Otwórz cmd.exe i wykonaj:

```
Wiersz polecenia

C:\...>cd %HOMEPATH%
C:\Users\student>mkdir VisionLab
C:\Users\student>cd VisionLab
C:\Users\student\VisionLab>python -m venv dip_venv
C:\Users\student\VisionLab>dip_venv\Scripts\activate.bat
(dip_venv) C:\Users\student\VisionLab>python -m pip install --upgrade pip
(dip_venv) C:\Users\student\VisionLab>python -m pip install numpy opencv-python
  matplotlib jupyter ipykernel
(dip_venv) C:\Users\student\VisionLab>python -m ipykernel install --user --
  name=dip_venv --display-name="Python (Vision Lab)"
```

4. Otwórz VS Code, wybierz folder VisionLab, ustaw interpreter na:

   dip_venv\Scripts\python.exe.
5. Przeprowadź prosty test w VS Code z wykorzystaniem kodu:

```
python:
import numpy as np
import matplotlib.pyplot as plt
img = np.zeros((100,100)); img[25:75, 25:75] = 255
plt.imshow(img, cmap='gray'); plt.show()
```

Kolejne rozdziały szczegółowo opisują i wyjaśniają każdy etap konfiguracji środowiska, pozwalając zrozumieć nie tylko jak, ale również dlaczego poszczególne elementy są istotne. Zachęcamy do dalszej lektury, by świadomie i efektywnie wykorzystywać narzędzia w praktyce inżynierskiej.

Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

1. Architektura środowiska

1.1. Czym jest środowisko deweloperskie?

Środowisko programistyczne to zestaw narzędzi i komponentów umożliwiających efektywną pracę nad projektami programistycznymi. Składa się z:

- interpretera języka (Python) – silnika wykonującego kod,
- edytora/IDE (VS Code) – interfejsu do pisania, uruchamiania i debugowania kodu,
- menedżera pakietów (pip) – narzędzia do instalacji i zarządzania bibliotekami,
- środowiska wirtualnego (venv) – systemu izolacji zależności dla każdego projektu,
- kernela Jupyter – pośrednika umożliwiającego uruchamianie kodu w notatnikach Jupyter.

1.2. Dlaczego izolacja środowisk?

W pracy nad wieloma projektami często pojawia się potrzeba korzystania z różnych wersji tych samych bibliotek. Środowiska wirtualne pozwalają na tworzenie odizolowanych przestrzeni, w których każdy projekt ma własny, niezależny zestaw bibliotek i ich wersji. Dzięki temu zmiany w jednym projekcie nie wpływają na pozostałe, co zapewnia stabilność i powtarzalność pracy.

Przykład problemu:

- projekt A potrzebuje: numpy==1.20.0, opencv==4.5.0
- projekt B potrzebuje: numpy==1.24.0, opencv==4.8.0

Instalowanie bibliotek globalnie prowadzi do konfliktów wersji i niestabilności projektów. Rozwiązaniem są środowiska wirtualne, np.:

- visionLab/dip_venv/ → numpy==1.24.0, opencv==4.8.0
- machineLearning/ml_env/ → numpy==1.20.0, tensorflow==2.10.0
- webScraping/web_env/ → requests==2.28.0, beautifulsoup==4.11.0

Tworzenie osobnych środowisk wirtualnych dla każdego projektu to standard branżowy i podstawa profesjonalnej pracy programistycznej.

2. Instalacja Visual Studio Code

Visual Studio Code (VS Code) to lekki, nowoczesny edytor kodu, który dzięki modułowej architekturze i bogatemu ekosystemowi rozszerzeń stał się standardem w pracy z Pythonem i narzędziami do analizy danych. Oferuje integrację z Jupyter

Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

Notebook, zaawansowane wsparcie dla Pythona, debugowanie, obsługę Git oraz szybkie działanie nawet na słabszych komputerach.

Aby zainstalować Visual Studio Code, wykonaj następujące kroki:

- pobierz instalator ze strony https://code.visualstudio.com
- podczas instalacji zaznacz:
  - ✓ „Add to PATH” (konieczne)
  - ✓ „Add 'Open with Code' action to Windows Explorer”
  - ✓ „Register Code as editor for supported file types”
- po instalacji uruchom VS Code i zainstaluj wymagane rozszerzenia, wpisując w terminalu

```
Wiersz polecenia

code --install-e xtension ms-python.python
code --install-extension ms-toolsai.jupyter
code --install-extension ms-python.vscode-pylance
code --install-extension ms-toolsai.jupyter-keymap
```

Prawidłowe ustawienie zmiennej środowiskowej PATH jest niezbędne, aby możliwe było uruchamianie VS Code z poziomu terminala. Bez tego system nie odnajdzie polecenia code.

Opis kluczowych rozszerzeń:

- Python – wsparcie dla języka Python, podświetlanie składni, debugowanie, IntelliSense.
- Jupyter – praca z notebookami Jupyter bezpośrednio w VS Code,
- Pylance – zaawansowana analiza kodu, sprawdzanie typów, inteligentne podpowiedzi,
- Jupyter Keymap – skróty klawiszowe znane z Jupyter Lab, ułatwiające pracę z notebookami,

3. Instalacja Python

3.1. Pobieranie właściwej wersji

Pobierz instalator ze strony https://www.python.org/downloads/

Zalecana wersja: Python 3.11.3 (lub najnowsza z serii 3.11.x)

Dlaczego 3.11, a nie 3.12?

Python 3.11 ma najlepsze wsparcie bibliotek do computer vision. Nowsze wersje mogą mieć problemy z kompatybilnością niektórych pakietów.

Politechnika Świętokrzyska Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

3.2. Instalacja z krytycznymi ustawieniami

Uruchom instalator jako administrator i zaznacz:

- ✓ Add Python 3.11 to PATH (na pierwszym ekranie!)
- ✓ Install for all users (jeśli masz uprawnienia)

Po zakończeniu instalacji zaleca się ponowne uruchomienie komputera, ponieważ Windows nie zawsze odświeża zmienne PATH bez restartu.

3.3. Weryfikacja instalacji

Otwórz wiersz polecenia (Win + R, wpisz cmd) i sprawdź wersje.

Poprawne wyświetlenie wersji oznacza, że instalacja przebiegła pomyślnie.

```
Wiersz polecenia

C:\Users\student>python –version
# Oczekiwany wynik: Python 3.11.3

C:\Users\student>where python
# Oczekiwany wynik:
#C:\Users\student\AppData\Local\Programs\Python\Python311\python.exe

C:\Users\student>python -c "import sys; print(sys.executable)"
# Sprawdza, który interpreter faktycznie używa Python
```

4. Środowisko wirtualne - Izolacja projektów

Współczesne projekty programistyczne w Python-ie niemal zawsze opierają się na zewnętrznych bibliotekach, których liczba i wersje mogą się znacząco różnić w zależności od zastosowania. Instalowanie wszystkich zależności globalnie prowadzi do tzw. „piekła zależności” (ang. *dependency hell*), gdzie konflikty wersji uniemożliwiają jednoczesne utrzymanie kilku projektów na jednym systemie.

Środowisko wirtualne to mechanizm, który pozwala utworzyć odizolowaną przestrzeń dla każdego projektu, w której można niezależnie instalować i zarządzać bibliotekami oraz interpreterem Pythona. Dzięki temu każdy projekt staje się autonomiczny, a jego uruchomienie i rozwój są powtarzalne iodporne na zmiany w innych częściach systemu.

Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

4.1. Tworzenie struktury projektu

Przygotuj nowy katalog *VisionLab*, w którym będziemy przechowywać wszystkie pliki projektu.

```
Wiersz polecenia

C:\...>cd %HOMEPATH%
C:\Users\student>mkdir VisionLab
C:\Users\student>cd VisionLab
```

Dlaczego w *%HOMEPATH%?* To katalog domowy użytkownika - mamy tam pełne uprawnienia, bez problemów z UAC (ang. *User Account Control*).

4.2. Inicjalizacja środowiska wirtualnego

Moduł *venv* tworzy kopię interpretera Python wraz z niezależną strukturą katalogów dla bibliotek. Aktywacja środowiska wirtualnego modyfikuje zmienne środowiskowe (*PATH*, *PYTHONPATH*) tak, aby wskazywały na lokalne kopie.

```
Wiersz polecenia

C:\Users\student\VisionLab>python -m venv dip_venv
```

Co się dzieje pod spodem:

- python tworzy kopię interpretera w *dip_venv\Scripts*
- powstaje struktura katalogów dla bibliotek w *dip_venv\Lib\site-packages*
- zostaje wygenerowany skrypty aktywacji w *dip_venv\Scripts*

Struktura katalogu po utworzeniu nowego środowiska wirtualnego:

```
Eksplorator plików
VisionLab/
└── dip_venv/
    ├── Scripts/
    │   ├── python.exe          # Kopia interpretera
    │   ├── pip.exe             # Menedżer pakietów
    │   ├── activate.bat        # Skrypt aktywacji (Windows)
    │   └── deactivate.bat      # Skrypt dezaktywacji
    ├── Lib/
    │   └── site-packages/      # Katalog na biblioteki
    └── pyvenv.cfg              # Konfiguracja środowiska
```

4.3. Aktywacja środowiska

```
Wiersz polecenia

C:\Users\student\VisionLab>dip_venv\Scripts\activate.bat
```


Politechnika Świętokrzyska Kielce University of Technology

Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

```
(dip_venv) C:\Users\student\VisionLab>_
```

Wskaźniki poprawnej aktywacji:

- prefiks *(dip_venv)* przed promptem
- *where python* wskazuje na *dip_venv\Scripts\python.exe*
- *echo %VIRTUAL_ENV%* pokazuje ścieżkę do środowiska

Mechanizm aktywacji:

Skrypt activate.bat modyfikuje zmienne środowiskowe:

- dodaje *dip_venv\Scripts* na początek *PATH*
- ustawia *VIRTUAL_ENV* na ścieżkę środowiska
- po aktywacji prompt terminala zmienia się na `(dip_venv)`.
- *where python* powinno wskazywać na *VisionLab\dip_venv\Scripts\python.exe.*

4.4. Dezaktywacja

Dezaktywacja przywraca oryginalne zmienne środowiskowe – jeśli potrzeba!

```
Wiersz polecenia

(dip_venv) C:\Users\student\VisionLab>deactivate
C:\Users\student\VisionLab>_
```

5. Instalacja bibliotek

Pip (ang. *Python Package Installe*) to standardowy menedżer pakietów dla języka Python, który umożliwia łatwe instalowanie, aktualizowanie i zarządzanie bibliotekami oraz modułami. Dzięki pip, możesz bez problemu korzystać z tysięcy pakietów dostępnych w oficjalnym repozytorium PyPI (ang. P*ython Package Index*), co znacząco przyspiesza rozwój aplikacji. Jest to kluczowe narzędzie dla każdego, kto pracuje z Pythonem, zapewniające spójność i kontrolę nad zależnościami projektu.

5.1. Aktualizacja pip

```
Wiersz polecenia

(dip_venv) C:\Users\student\VisionLab>python -m pip install --upgrade pip
```

Dlaczego python -m pip zamiast pip?

Politechnika Świętokrzyska Kielce University of Technology

Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

Wywołanie przez python -m pip gwarantuje, że używamy pip-a powiązanego z aktywnym interpreterem. Bezpośrednie wywołanie pip może używać globalnej instalacji.

5.2. Instalacja pakietów podstawowych

Po aktywacji środowiska wirtualnego kluczowe jest zainstalowanie niezbędnych bibliotek, które stanowią fundament dla operacji przetwarzania obrazów i analizy danych:

- **numpy** - operacje na tablicach wielowymiarowych (podstawa wszystkiego)
- **opencv-python** - computer vision (filtrowanie, detekcja, transformacje)
- **matplotlib** - wizualizacja (wykresy, wyświetlanie obrazów)
- **jupyter** - interaktywne notatniki
- **ipykernel** - mostek między Jupyter a interpreterem Python

```
Wiersz polecenia

(dip_venv) C:\Users\student\VisionLab>python -m pip install numpy opencv-python
  matplotlib jupyter ipykernel
```

Opcjonalne - dla zaawansowanych operacji:

```
Wiersz polecenia

(dip_venv) C:\Users\student\VisionLab>python -m pip install scipy scikit-image
  pillow
```

5.3. Zarządzanie zależnościami

Plik requirements.txt to manifest zależności (ang. *MANIFESTDEPENDENCY*) - lista wszystkich pakietów z konkretnymi wersjami, która umożliwia reprodukcję środowiska programistycznego.

Eksport aktualnych pakietów do listy.

```
Wiersz polecenia

(dip_venv) C:\Users\student\VisionLab>pip freeze > requirements.txt
```

Przykład pliku requirements.txt:

```
Notatnik - requirements.txt

numpy==1.24.3
opencv-python==4.8.0.74
```


Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

```
matplotlib==3.7.1
jupyter==1.0.0
ipykernel==6.23.1
```

Instalacja z pliku (na innym komputerze):

```
Wiersz polecenia
(dip_venv) C:\Users\student\VisionLab>python -m pip install -r requirements.txt
```

6. Integracja Jupyter

Jupyter Notebook umożliwia interaktywną pracę z kodem oraz dokumentowanie procesu analizy i przetwarzania danych. Stanowi standardowe narzędzie w projektach związanych z analizą danych i uczeniem maszynowym. Integracja Jupyter Notebook z Visual Studio Code pozwala na korzystanie z funkcjonalności notatników w ramach jednego, rozbudowanego środowiska programistycznego. Takie rozwiązanie zwiększa efektywność pracy oraz ułatwia zarządzanie kodem i wynikami analiz.

6.1. Rejestracja kernela

Aby środowisko wirtualne było dostępne w Jupyter Notebook oraz Visual Studio Code, konieczne jest zarejestrowanie jego kernela w systemie Jupyter.

```
Wiersz polecenia
(dip_venv) C:\Users\student\VisionLab>python -m ipykernel install --user --
  name=dip_venv --display-name="Python (Vision Lab)"
```

Mechanizm działania:

- Jupyter tworzy wpis konfiguracyjny w katalogu \~/.local/share/jupyter/kernels/dip_venv/.
- plik kernel.json w tym katalogu zawiera ścieżkę do interpretera Pythona oraz niezbędne parametry uruchomieniowe.
- kernel staje się dostępny do wyboru we wszystkich instancjach Jupyter Notebook oraz JupyterLab.

Politechnika Świętokrzyska Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

6.2. Weryfikacja kerneli

Weryfikacja zarejestrowanych kerneli pozwala upewnić się, że środowisko wirtualne zostało poprawnie dodane i jest dostępne do użycia w Jupyter Notebook oraz Visual Studio Code.

```
Wiersz polecenia

(dip_venv) C:\Users\student\VisionLab>jupyter kernelspec list

#Przykładowy wynik
# Available kernels:
#  (dip_venv) C:\Users\Student\AppData\Roaming\jupyter\kernels\dip_venv
#   python3   C:\Users\Student\AppData\Roaming\jupyter\kernels\python3
```

6.3. Usuwanie kernela (jeśli potrzeba)

W przypadku konieczności usunięcia nieużywanego lub błędnie zarejestrowanego kernela, Jupyter umożliwia jego bezpieczne i trwałe usunięcie z systemu.

```
Wiersz polecenia

(dip_venv) C:\Users\student\VisionLab>jupyter kernelspec remove dip_venv
```

7. Konfiguracja Visual Studio Code

Visual Studio Code stanowi główne środowisko programistyczne do pracy z kodem Python oraz notatnikami Jupyter. Odpowiednia konfiguracja edytora oraz instalacja niezbędnych rozszerzeń znacząco wpływa na efektywność pracy i komfort programowania. Integracja VS Code z wcześniej skonfigurowanym środowiskiem wirtualnym oraz kernelami Jupyter umożliwia płynne przełączanie między różnymi projektami. Prawidłowe ustawienia zapewniają dostęp do wszystkich funkcjonalności niezbędnych w procesie przetwarzania obrazów.

7.1. Otwieranie projektu

Aby rozpocząć pracę z projektem w Visual Studio Code, wykonaj następujące kroki:

- uruchom aplikację Visual Studio Code.
- w menu wybierz File → Open Folder... i wskaż katalog projektu (np. VisionLab).
- VS Code automatycznie utworzy plik konfiguracyjny VisionLab.code-workspace oraz katalog .vscode/ zawierający ustawienia specyficzne dla tego obszaru roboczego.

Politechnika Świętokrzyska Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

7.2. Wybór interpretera

Aby zapewnić, że Visual Studio Code korzysta z właściwego środowiska wirtualnego, należy wybrać odpowiedni interpreter Pythona. Można to zrobić na dwa sposoby:

Metoda 1: Command Palette

- naciśnij Ctrl+Shift+P, aby otworzyć Command Palette.
- wpisz Python: Select Interpreter i wybierz tę opcję.
- z listy dostępnych interpreterów wskaż ścieżkę do interpretera w Twoim środowisku wirtualnym, np. .\dip_venv\Scripts\python.exe.

Metoda 2: Pasek statusu

- kliknij na aktualnie wybrany interpreter Pythona, znajdujący się w lewym dolnym rogu paska statusu Visual Studio Code.
- z wyświetlonej listy wybierz właściwy interpreter odpowiadający Twojemu środowisku wirtualnemu.

7.3. Automatyczna konfiguracja terminala

Visual Studio Code automatycznie aktywuje środowisko wirtualne w zintegrowanym terminalu, jeśli spełnione są następujące warunki:

- w katalogu projektu znajduje się plik pyvenv.cfg.
- interpreter Pythona jest ustawiony na środowisko wirtualne.

Weryfikacja:

- otwórz terminal w VS Code (Ctrl + `).
- upewnij się, że w promptcie pojawia się prefiks środowiska, np. (dip_venv).

7.4. Konfiguracja workspace (.vscode/settings.json)

Visual Studio Code może automatycznie utworzyć plik konfiguracyjny .vscode/settings.json zawierający ustawienia specyficzne dla projektu. Poniżej przedstawiono przykładowy plik konfiguracyjny.

Notatnik - .vscode/settings.json

```
{   "python.pythonPath": "./dip_venv/Scripts/python.exe",
    "python.terminal.activateEnvironment": true, "python.linting.enabled": true,
    "python.linting.pylintEnabled": true,
    "jupyter.jupyterServerType": "local"
}
```


Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

8. Testowanie i walidacja środowiska

8.1. Test funkcjonalności interpretera - Python script

Aby zweryfikować poprawność konfiguracji środowiska, utwórz plik test_environment.py w katalogu projektu i zapisz w nim następujący kod:

```
python:
import sys
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt

def test_environment():
    """Comprehensive environment test"""
    print("=== ENVIRONMENT TEST ===")
    print(f"Python version: {sys.version}")
    print(f"Python executable: {sys.executable}")
    print(f"Virtual environment: {os.environ.get('VIRTUAL_ENV', 'Not
activated')}")

    print("\n=== LIBRARY VERSIONS ===")
    print(f"NumPy: {np.__version__}")
    print(f"OpenCV: {cv2.__version__}")
    print(f"Matplotlib: {plt.matplotlib.__version__}")

    print("\n=== FUNCTIONALITY TEST ===")
    # Test NumPy
    arr = np.random.rand(3, 3)
    print(f"NumPy array creation: {'✓' if arr.shape == (3, 3) else 'X'}")

    # Test OpenCV
    img = np.zeros((100, 100, 3), dtype=np.uint8)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    print(f"OpenCV color conversion: {'✓' if gray.shape == (100, 100) else 'X'}")

    # Test Matplotlib
    fig, ax = plt.subplots()
    ax.plot([1, 2, 3], [1, 4, 2])
    print(f"Matplotlib plotting: ✓")
    plt.close(fig)

    print("\n=== TEST COMPLETED ===")

if __name__ == "__main__":
    test_environment()
```


Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

Uruchom skrypt w aktywnym środowisku wirtualnym:

```
Wiersz polecenia
(dip_venv) C:\Users\student\VisionLab>python test_environment.py
```

8.2. Test Jupyter Notebook

W Visual Studio Code utwórz nowy notatnik Jupyter o nazwie test_notebook.ipynb i dodaj dwie komórki typu Code.

Komórka 1 - Environment check:

```python
import sys
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt

print(f"Kernel: {sys.executable}")
print(f"Virtual env: {os.environ.get('VIRTUAL_ENV', 'Not detected')}")
print(f"Working dir: {os.getcwd()}")
```

Komórka 2 - Image processing demo:

```python
# Create test image with patterns
def create_test_pattern(size=256):
    """Create test image with various patterns"""
    img = np.zeros((size, size), dtype=np.uint8)

    # Gradient background
    for i in range(size):
        img[i, :] = int(255 * i / size)

    # Geometric shapes
    cv2.rectangle(img, (50, 50), (150, 150), 255, -1)
    cv2.circle(img, (200, 200), 30, 128, -1)

    # Add some noise
    noise = np.random.randint(0, 50, (size, size), dtype=np.uint8)
    img = cv2.add(img, noise)

    return img

# Generate and process image
original = create_test_pattern()
blurred = cv2.GaussianBlur(original, (15, 15), 0)
```


Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

```
edges = cv2.Canny(original, 50, 150)

# Visualization
fig, axes = plt.subplots(2, 2, figsize=(10, 10))

axes[0, 0].imshow(original, cmap='gray')
axes[0, 0].set_title('Original Pattern')
axes[0, 0].axis('off')

axes[0, 1].imshow(blurred, cmap='gray')
axes[0, 1].set_title('Gaussian Blur')
axes[0, 1].axis('off')

axes[1, 0].imshow(edges, cmap='gray')
axes[1, 0].set_title('Canny Edge Detection')
axes[1, 0].axis('off')

# Histogram
axes[1, 1].hist(original.ravel(), bins=256, range=[0, 256], alpha=0.7)
axes[1, 1].set_title('Intensity Histogram')
axes[1, 1].set_xlabel('Pixel Intensity')
axes[1, 1].set_ylabel('Frequency')

plt.tight_layout()
plt.show()

print("✓ Jupyter Notebook integration working correctly!")
```

9. Diagnostyka

Prawidłowa diagnostyka środowiska programistycznego pozwala szybko zidentyfikować i rozwiązać najczęstsze problemy pojawiające się podczas pracy z Pythonem, środowiskami wirtualnymi oraz narzędziami deweloperskimi.

Najczęstsze problemy i sposoby ich rozwiązywania:

- Python lub pip nie są rozpoznawane w terminalu:
  Sprawdź, czy Python został dodany do zmiennej środowiskowej PATH. Wpisz w terminalu python --version oraz pip --version. Jeśli polecenia nie działają, uruchom ponownie komputer lub sprawdź ustawienia PATH.
- środowisko wirtualne nie aktywuje się:
  Upewnij się, że używasz poprawnej komendy aktywacji (.\nazwa_środowiska\Scripts\activate w cmd). Sprawdź, czy środowisko zostało utworzone w odpowiednim katalogu.
- problemy z instalacją pakietów:

Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

Sprawdź, czy środowisko jest aktywne przed instalacją. W razie błędów spróbuj zaktualizować pip (python -m pip install --upgrade pip) lub sprawdź komunikaty o błędach.

- kernel nie pojawia się w Jupyter Notebook:
  Upewnij się, że kernel został poprawnie zarejestrowany (python -m ipykernel install --user --name=NAZWA). Sprawdź katalog \~/.local/share/jupyter/kernels/ lub %APPDATA%\jupyter\kernels\ (Windows).
- nieprawidłowe wersje bibliotek:
  Sprawdź zainstalowane wersje poleceniem pip list. W razie potrzeby odinstaluj i zainstaluj właściwą wersję (pip install numpy==1.24.0).
- VS Code nie widzi środowiska lub interpretera:
  Wybierz interpreter ręcznie przez Command Palette (Ctrl+Shift+P → Python: Select Interpreter). Upewnij się, że ścieżka wskazuje na właściwe środowisko.

Dodatkowe narzędzia diagnostyczne:

- where python – sprawdza lokalizację aktywnego interpretera.
- pip freeze – wyświetla listę zainstalowanych pakietów i ich wersji.
- logi Jupyter i VS Code – w przypadku trudnych do zdiagnozowania problemów sprawdź komunikaty w terminalu lub wbudowanych konsolach.

10. Dobre praktyki i optymalizacja przepływu pracy

Stosowanie dobrych praktyk w organizacji środowiska pracy jest kluczowe dla zapewnienia stabilności, powtarzalności i efektywności realizowanych projektów:

- dla każdego projektu twórz osobne środowisko wirtualne
- przed instalacją pakietów zawsze aktywuj odpowiednie środowisko
- eksportuj zależności do pliku requirements.txt i aktualizuj go po każdej zmianie
- nie dodawaj katalogu środowiska wirtualnego do repozytorium (np. dip_venv/)
- automatyzuj konfigurację środowiska za pomocą skryptów .bat lub .sh

10.1. Struktura projektu

Zaleca się utrzymywanie przejrzystej struktury katalogów, np. oddzielając kod źródłowy, dane, notatniki i pliki konfiguracyjne. Ułatwia to zarządzanie projektem i współpracę w zespole

Eksplorator plików

```
VisionLab/
├── dip_venv/                    # Środowisko wirtualne (nie commitować)
```


Politechnika Świętokrzyska Kielce University of Technology

Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

```
├── .vscode/                    # Konfiguracja Visual Studio Code
│   ├── settings.json           # Ustawienia Workspace
│   ├── launch.json             # Konfiguracja Debugera
│   └── tasks.json              # Zadania buildowania
├── src/                        # Kod źródłowy projektu
│   ├── __init__.py
│   ├── image_processing/       # Moduły do przetwarzania obrazów
│   │   ├── __init__.py
│   │   ├── filters.py          # Filtry obrazu
│   │   └── transformations.py # Transformacje obrazu
│   └── utils/                  # Funkcje pomocnicze
│       ├── __init__.py
│       └── helpers.py
├── notebooks/                  # Notatniki Jupyter
│   ├── 01_introduction.ipynb
│   ├── 02_basic_operations.ipynb
│   └── experiments/            # Eksperymenty i prototypy
├── data/                       # Zbiory danych (nie commitować dużych plików)
│   ├── raw/                    # Dane surowe
│   ├── processed/              # Dane przetworzone
│   └── samples/                # Przykładowe dane
├── tests/                      # Testy jednostkowe
│   ├── __init__.py
│   ├── test_filters.py
│   └── test_transformations.py
├── docs/                       # Dokumentacja projektu
├── requirements.txt            # Lista zależności projektu
├── .gitignore                  # Plik z regułami ignorowania w Git
├── README.md                   # Dokumentacja główna projektu
└── setup.py                    # Konfiguracja pakietu (opcjonalnie)
```

10.2. Automatyzacja workflow

Konfigurację środowiska można zautomatyzować, tworząc skrypt, np. *setup_environment.bat*, który przyspiesza i standaryzuje proces przygotowania projektu

Notatnik - setup_environment.bat

```
@echo off
echo Setting up Vision Lab environment...

python -m venv dip_venv
call dip_venv\Scripts\activate.bat

python -m pip install --upgrade pip
python -m pip install -r requirements.txt
```


Politechnika Świętokrzyska Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

```
python -m ipykernel install --user --name=dip_venv --display-name="Python
   (Vision Lab)"

echo Environment setup completed!
pause
```

10.3. Git - zarządzanie wersjami i bezpieczeństwo projektu

Współczesne projekty programistyczne wymagają nie tylko pisania kodu, ale także skutecznego zarządzania jego wersjami oraz ochrony przed utratą danych. Git (ang. *Global Information Tracker*) to rozproszony system kontroli wersji, który stał się standardem w zarządzaniu kodem źródłowym. Umożliwia śledzenie zmian w projekcie, pracę zespołową, szybkie przywracanie wcześniejszych wersji oraz skuteczny backup – zarówno lokalnie, jak i w chmurze (np. na GitHubie czy GitLabie). Stosowanie Gita nie tylko zabezpiecza projekt przed utratą danych, ale także pozwala na eksperymentowanie z kodem bez ryzyka – w każdej chwili można wrócić do stabilnej wersji lub prześledzić historię rozwoju projektu.

Typowy workflow z Gitem obejmuje:

- inicjalizację repozytorium (git init) w katalogu projektu.
- dodawanie plików do śledzenia (*git add nazwa_pliku* lub *git add .*).
- tworzenie snapshotów zmian poprzez commity (*git commit -m "Opis zmian"*).
- wysyłanie zmian do zdalnego repozytorium (git push), co pełni rolę backupu i umożliwia współpracę z innymi.
- pobieranie najnowszych zmian od zespołu (git pull).

Do obsługi Gita można wykorzystywać różne narzędzia:

- Git Bash – terminal z poleceniami Gita, umożliwiający pracę w stylu linuksowym na systemie Windows.
- TortoiseGit – graficzny interfejs do Gita, zintegrowany z Eksploratorem Windows, ułatwiający zarządzanie repozytoriami bez użycia terminala.
- Visual Studio Code – posiada wbudowaną obsługę Gita (w panelu Source Control), umożliwiającą wykonywanie najważniejszych operacji (commit, push, pull, przeglądanie historii) bezpośrednio z poziomu edytora.

Warto pamiętać, że do repozytorium nie powinno się dodawać plików tymczasowych, środowisk wirtualnych czy dużych zbiorów danych – do ich wykluczania służy plik .gitignore. Dobrą praktyką jest częste commitowanie zmian z czytelnymi opisami oraz regularna synchronizacja z repozytorium zdalnym. Aby zapewnić porządek w repozytorium i uniknąć przypadkowego dodania niepożądanych plików, warto skonfigurować plik .gitignore, który określa, które pliki i katalogi mają być pomijane przez system kontroli wersji.

Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

Przykładowy plik .gitignore dla projektów Python:

Notatnik - .gitignore

```
# Virtual environment
dip_venv/
venv/
env/

# IDE
.vscode/settings.json
.idea/

# Python
__pycache__/
*.pyc
*.pyo
*.pyd
.Python
*.so

# Jupyter
.ipynb_checkpoints/

# Data files
data/
*.csv
*.jpg
*.png
*.bmp
```

11. Dodatkowe zasoby

W celu pogłębienia wiedzy oraz samodzielnego rozwiązywania problemów, warto korzystać z oficjalnych materiałów i dokumentacji:

- Python Virtual Environments https://docs.python.org/3/tutorial/venv.html
- VS Code Python Tutorial https://code.visualstudio.com/docs/python/python-tutorial
- Jupyter Documentationhttps://jupyter.org/documentation
- NumPy Documentation https://numpy.org/doc/
- OpenCV Documentation https://docs.opencv.org/
- Matplotlib Documentation https://matplotlib.org/stable/contents.html
- scikit-image Documentation https://scikit-image.org/docs/stable/

Politechnika Świętokrzyska
Kielce University of Technology
Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

12. Literatura

1. Johansson, R. (2020). Matematyczny Python. Obliczenia naukowe i analiza danych z użyciem NumPy, SciPy i Matplotlib. Helion.
2. McKinney, W. (2022). Python w analizie danych. Przetwarzanie danych za pomocą pakietów pandas i NumPy oraz środowiska Jupyter (3. wydanie). Helion.
3. Rother, K. (2021). Python dla profesjonalistów. Debugowanie, testowanie i utrzymywanie kodu. Helion.
4. Smith, J. (2021). Python Virtual Environments: A Practical Guide to venv, pip, and Dependency Management. O'Reilly Media.
5. Visual Studio Code Docs. (2024). Jupyter Notebooks in VS Code. https://code.visualstudio.com/docs/datascience/jupyter-notebooks (dostęp: 15.07.2025)
6. Zimoch, M. (2021). OpenCV. Kurs video. Przetwarzanie obrazów w języku Python. Helion.
7. Obliczenia naukowe w języku Python – wprowadzenie do biblioteki Matplotlib. (2023). Nearshore IT. https://www.nearshore-it.eu/pl/artykuly/biblioteka-matplotlib/ (dostęp: 15.07.2025)