Nama : Rhonni Irama Noorhuda

NIM : 1103210176

Kelas : TK – 45 – G09

ANALISIS HASIL SIMULASI

1. **Analisis Tugas 1**

Pemanfaatan teknologi pemrosesan citra (image processing) semakin berkembang pesat dalam berbagai bidang seperti pengenalan wajah, pengindeksan gambar, serta robotika. Dalam simulasi ini, Google Colab dipilih sebagai platform untuk menjalankan proses pemrograman menggunakan pustaka OpenCV. Google Colab menawarkan keunggulan berupa lingkungan berbasis cloud yang mendukung eksekusi kode Python dengan daya komputasi yang cukup tinggi tanpa memerlukan perangkat keras lokal yang canggih.

Simulasi yang dilakukan di Google Colab menunjukkan bahwa penggunaan OpenCV dapat diimplementasikan dengan efisien dalam analisis citra, didukung oleh kemudahan berbagi serta integrasi dengan layanan cloud. Hasil simulasi ini juga membuktikan bahwa kolaborasi antara image processing, feature detection, dan feature description dapat menghasilkan sistem analisis citra yang andal dan adaptif untuk berbagai aplikasi teknologi.

- !pip install opencv-python opencv-contrib-python (Untuk install OpenCV terlebih dahulu di google collab)
- ```
  from google.colab import files
  import cv2
  import numpy as np
  import matplotlib.pyplot as plt

  # Upload gambar
  uploaded = files.upload()
  image_path = list(uploaded.keys())[0]

  # Load gambar
  image = cv2.imread(image_path, cv2.IMREAD_COLOR)
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert
  BGR to RGB for display

  # Tampilkan gambar asli
  plt.imshow(image)
  plt.title("Uploaded Image")
  plt.axis('off')
  plt.show()
  ```
- ```
  # Membuat filter moving average
  kernel = np.ones((5, 5), np.float32) / 25
  smoothed_image = cv2.filter2D(image, -1, kernel)

  # Tampilkan hasil
  plt.imshow(smoothed_image)
  ```

```
plt.title("Moving Average Filter")
plt.axis('off')
plt.show()
# Konversi ke grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# Inisialisasi SIFT
sift = cv2.SIFT_create()

# Deteksi fitur
keypoints, descriptors = sift.detectAndCompute(gray_image,
None)

# Gambar keypoints
sift_image = cv2.drawKeypoints(image, keypoints, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

# Tampilkan hasil
plt.imshow(sift_image)
plt.title("SIFT Feature Detection")
plt.axis('off')
plt.show()
# Hitung histogram untuk masing-masing channel (RGB)
colors = ('r', 'g', 'b')
plt.figure(figsize=(10, 5))

for i, color in enumerate(colors):
    hist = cv2.calcHist([image], [i], None, [256], [0, 256])
    plt.plot(hist, color=color)
    plt.xlim([0, 256])

plt.title("Histogram Gambar")
plt.xlabel("Intensity Value")
plt.ylabel("Pixel Count")
plt.show()
# Aplikasi Gaussian Blur
gaussian_blurred = cv2.GaussianBlur(image, (5, 5), 0)

# Tampilkan hasil
plt.imshow(gaussian_blurred)
plt.title("Gaussian Smoothing")
plt.axis('off')
plt.show()
# Konversi ke grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# Sobel filter
```

```
sobelx = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0, ksize=3)  # Gradien x
sobely = cv2.Sobel(gray_image, cv2.CV_64F, 0, 1, ksize=3)  # Gradien y

# Kombinasi gradien
sobel_combined = cv2.magnitude(sobelx, sobely)

# Tampilkan hasil
plt.imshow(sobel_combined, cmap='gray')
plt.title("Sobel Edge Detection")
plt.axis('off')
plt.show()
```

- ```
  from skimage.feature import hog
  from skimage import exposure

  # Konversi ke grayscale
  gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

  # Ekstraksi fitur HOG
  hog_features, hog_image = hog(
      gray_image,
      orientations=9,
      pixels_per_cell=(8, 8),
      cells_per_block=(2, 2),
      block_norm='L2-Hys',
      visualize=True,
      feature_vector=True
  )

  # Tampilkan hasil
  plt.imshow(hog_image, cmap='gray')
  plt.title("HOG Feature Representation")
  plt.axis('off')
  plt.show()
  ```

2. Analisis Tugas 2

- Visual Tracking

  Visual tracking merupakan salah satu elemen penting dalam robotika modern, khususnya dalam aplikasi navigasi dan interaksi robot dengan lingkungan. Dalam konteks simulasi menggunakan Webots, visual tracking memungkinkan robot untuk mendeteksi, melacak, dan merespons objek tertentu berdasarkan input visual dari kamera virtual yang disediakan. Teknologi ini menjadi dasar untuk mengimplementasikan sistem robotik yang lebih cerdas, seperti robot pengikut objek atau kendaraan otonom.

  Keunggulan simulasi visual tracking di Webots adalah kemampuannya untuk menyediakan lingkungan uji yang realistis tanpa memerlukan perangkat keras fisik. Dengan alat ini, pengembang dapat mengevaluasi performa
  ```

algoritma visual tracking dalam berbagai skenario, seperti variasi kondisi pencahayaan atau perubahan kecepatan objek. Hal ini memberikan fleksibilitas untuk mengoptimalkan desain sistem sebelum diterapkan pada robot dunia nyata.

Secara keseluruhan, penerapan visual tracking pada Webots menjadi komponen esensial dalam pengembangan robot cerdas. Dengan dukungan fitur yang komprehensif dari Webots, simulasi ini tidak hanya mempercepat proses penelitian dan pengembangan tetapi juga meminimalkan risiko kesalahan pada tahap implementasi di dunia nyata.

- """This controller moves the red ball around the robot."""
  # Komentar ini menjelaskan bahwa program ini bertujuan untuk menggerakkan bola merah di sekitar robot.
  # Bola digerakkan dalam lintasan melingkar di bidang simulasi Webots.

  from math import sin, cos, pi
  # Mengimpor fungsi trigonometri `sin` dan `cos` dari modul `math` untuk menghitung posisi bola.
  # `pi` juga diimpor sebagai konstanta untuk perhitungan sudut dalam radian.

  from controller import Supervisor
  # Mengimpor kelas `Supervisor` dari pustaka `controller`.
  # Supervisor memungkinkan kontrol penuh atas elemen-elemen dalam simulasi Webots, termasuk posisi objek.

  # Inisialisasi Supervisor
  supervisor = Supervisor()
  # Membuat objek `supervisor` untuk mengontrol simulasi Webots.

  timestep = int(supervisor.getBasicTimeStep())
  # Mendapatkan nilai langkah waktu dasar simulasi dalam milidetik.
  # Langkah waktu ini digunakan untuk memastikan simulasi berjalan secara konsisten.

  # Mendapatkan referensi ke objek "BALL"
  ball = supervisor.getFromDef('BALL')
  # `getFromDef` digunakan untuk mendapatkan referensi ke objek bola (ball) berdasarkan DEF node-nya di file .wbt.
  # Nama "BALL" harus sesuai dengan nama DEF di file dunia Webots.

  ball_translation = ball.getField('translation')
  # Mengakses properti `translation` dari objek bola.
  # Properti ini digunakan untuk mengatur posisi bola di dunia simulasi.

  # Inisialisasi variabel sudut
  angle = 0

`angle` adalah variabel yang digunakan untuk menghitung posisi bola berdasarkan fungsi trigonometri.

# Loop utama simulasi
while supervisor.step(timestep) != -1:
    # `supervisor.step(timestep)` menjaga simulasi tetap berjalan dengan interval waktu tertentu.
    # Jika nilai yang dikembalikan tidak sama dengan -1, simulasi akan terus berjalan.

    x = cos(angle)
    y = sin(angle)
    # Menghitung koordinat x dan y dari bola berdasarkan sudut `angle`.
    # Fungsi `cos` digunakan untuk menghitung posisi x, sedangkan `sin` untuk posisi y.
    # Bola bergerak dalam lintasan melingkar di bidang horizontal.

    ball_translation.setSFVec3f([x, y, 0.2])
    # Memperbarui posisi bola di dunia simulasi.
    # `setSFVec3f` menetapkan posisi bola dalam koordinat 3D.
    # Koordinat z tetap 0.2 agar bola berada pada ketinggian tertentu dari dasar bidang simulasi.

    angle += 0.01
    # Menambah nilai sudut secara bertahap untuk menghasilkan gerakan melingkar.
    # Kenaikan kecil pada sudut menciptakan lintasan yang mulus.

- """This controller makes the robot following the red ball."""
Komentar ini menjelaskan bahwa program ini bertujuan untuk membuat robot mengikuti bola merah.

import cv2
import numpy as np
from controller import Robot
# Mengimpor pustaka OpenCV (`cv2`) untuk pengolahan citra, numpy (`np`) untuk manipulasi array,
# dan kelas `Robot` dari Webots untuk mengontrol robot dalam simulasi.

# P value for P controller
# High P value makes the robot more aggressive
# Low P value makes the robot more sluggish
P_COEFFICIENT = 0.1
# Konstanta `P_COEFFICIENT` menentukan tingkat sensitivitas robot terhadap error dalam sistem kontrol proporsional (P controller).
# Nilai lebih tinggi membuat robot bereaksi lebih agresif, sedangkan nilai lebih rendah membuatnya lebih lambat.

```
# Initialize the robot
robot = Robot()
# Membuat objek `robot` untuk mengontrol robot di simulasi Webots.

timestep = int(robot.getBasicTimeStep())
# Mengambil waktu langkah dasar simulasi dalam milidetik untuk sinkronisasi
robot.

# Initialize camera
camera = robot.getDevice('camera')
camera.enable(timestep)
# Menginisialisasi kamera robot dan mengaktifkannya dengan langkah waktu
simulasi (`timestep`).

# Initialize motors
motor_left = robot.getDevice('left wheel motor')
motor_right = robot.getDevice('right wheel motor')
# Mendapatkan referensi ke motor kiri dan kanan robot.

motor_left.setPosition(float('inf'))
motor_right.setPosition(float('inf'))
# Menetapkan posisi motor sebagai `inf` untuk mode kecepatan kontinu.
# Robot tidak memiliki batas rotasi tertentu untuk motor roda.

motor_left.setVelocity(0)
motor_right.setVelocity(0)
# Menetapkan kecepatan awal roda kiri dan kanan ke 0.

# Main control loop
while robot.step(timestep) != -1:
    # Loop utama simulasi, berjalan selama simulasi masih aktif.
    # `robot.step(timestep)` memastikan simulasi berjalan sesuai waktu langkah
yang ditentukan.

    img = np.frombuffer(camera.getImage(),
dtype=np.uint8).reshape((camera.getHeight(), camera.getWidth(), 4))
    # Mengambil gambar dari kamera sebagai array buffer dalam format
RGBA.
    # Array ini diubah menjadi array numpy 3D dengan dimensi (tinggi, lebar,
4) untuk memproses piksel gambar.

    # Segment the image by color in HSV color space
    img = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
    # Mengonversi gambar dari ruang warna RGB ke HSV (Hue, Saturation,
Value).
```

```
    # HSV lebih baik untuk segmentasi warna dibandingkan RGB.

    mask = cv2.inRange(img, np.array([50, 150, 0]), np.array([200, 230, 255]))
    # Membuat mask biner untuk mendeteksi warna tertentu (bola merah).
    # Warna yang berada dalam rentang [50, 150, 0] hingga [200, 230, 255]
akan disegmentasi.

    # Find the largest segmented contour (red ball) and its center
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
    # Menemukan kontur pada mask biner.
    # `cv2.RETR_EXTERNAL` digunakan untuk mengambil kontur eksternal
saja.
    # `cv2.CHAIN_APPROX_NONE` menyimpan semua titik kontur tanpa
pengurangan.

    largest_contour = max(contours, key=cv2.contourArea)
    # Memilih kontur dengan area terbesar, yang diasumsikan sebagai bola
merah.

    largest_contour_center = cv2.moments(largest_contour)
    # Menghitung momen geometrik dari kontur terbesar untuk menemukan
pusatnya.

    center_x = int(largest_contour_center['m10'] /
largest_contour_center['m00'])
    # Menentukan koordinat horizontal (x) dari pusat kontur berdasarkan
momen geometrik.

    # Find error (ball distance from image center)
    error = camera.getWidth() / 2 - center_x
    # Menghitung error sebagai jarak horizontal antara pusat gambar kamera
dan pusat bola.
    # Jika bola berada di tengah gambar, error akan bernilai nol.

    # Use simple proportional controller to follow the ball
    motor_left.setVelocity(- error * P_COEFFICIENT)
    motor_right.setVelocity(error * P_COEFFICIENT)
    # Mengatur kecepatan roda kiri dan kanan menggunakan kontrol
proporsional (P controller).
    # Kecepatan diatur berdasarkan nilai error:
    # - Jika bola terlalu ke kiri, roda kanan berputar lebih cepat untuk
mengarahkan robot ke kanan.
    # - Jika bola terlalu ke kanan, roda kiri berputar lebih cepat untuk
mengarahkan robot ke kiri.
```

- Document Scanner Simulation

  Simulasi Document Scanner di aplikasi Webots merupakan implementasi sistem robotik yang dirancang untuk memindai dokumen secara otomatis dalam lingkungan simulasi. Proses ini melibatkan pengambilan gambar dokumen menggunakan sensor kamera yang terpasang pada robot, pengolahan citra untuk mendeteksi tepi dokumen, serta pengubahan gambar menjadi format yang sesuai untuk aplikasi pemindai. Teknologi ini memanfaatkan kemampuan simulasi robotik Webots untuk menguji sistem tanpa memerlukan perangkat keras fisik.

  Keunggulan simulasi Document Scanner di Webots adalah fleksibilitas dalam pengujian algoritma pada berbagai kondisi, seperti pencahayaan yang berbeda atau keberadaan noise di sekitar dokumen. Selain itu, simulasi memungkinkan pengembang untuk merancang mekanisme pemindai yang lebih kompleks, misalnya robot yang dapat memindai beberapa halaman dokumen secara otomatis.

  Secara keseluruhan, simulasi ini memberikan pendekatan yang efisien untuk mengembangkan dan menguji sistem document scanner sebelum diterapkan pada perangkat keras nyata. Hal ini mempercepat proses pengembangan dan meminimalkan biaya serta risiko kesalahan pada tahap implementasi.

- # Standard Library imports
  from itertools import count
  # Mengimpor pustaka standar `itertools.count` yang digunakan untuk membuat penghitung otomatis.

  # External imports
  import cv2
  import numpy as np
  from controller import Robot, Display
  # Mengimpor pustaka eksternal:
  # `cv2` untuk pemrosesan citra menggunakan OpenCV.
  # `numpy` untuk manipulasi array numerik.
  # `Robot` dan `Display` dari Webots untuk mengontrol robot dan tampilan.

  # Local imports
  from scanner import get_warped_document, resize_and_letter_box, segment_by_color
  # Mengimpor fungsi-fungsi lokal dari modul `scanner`:
  # - `get_warped_document`: Meluruskan dokumen berdasarkan masker.
  # - `resize_and_letter_box`: Mengubah ukuran gambar agar sesuai dengan tampilan.
  # - `segment_by_color`: Melakukan segmentasi berdasarkan warna.

  # Konstanta
  TIME_STEP = 100 # Langkah waktu untuk simulasi Webots.

```
HSV_LOW_RANGE = np.array([27, 0, 66])  # Batas bawah rentang warna HSV untuk segmentasi.
HSV_UP_RANGE = np.array([180, 38, 255])  # Batas atas rentang warna HSV untuk segmentasi.
SAVE_TO_DISK = False  # Mengontrol apakah gambar akan disimpan ke disk.

def counter(_count=count(1)):
    """Fungsi penghitung otomatis berbasis itertools.count."""
    return next(_count)

def save_image(image):
    """Menyimpan gambar ke disk dengan nama berurutan."""
    cv2.imwrite(f"image_{counter()}.jpg", image)

def initialize():
    """Inisialisasi robot, kamera, dan tampilan."""
    robot = Robot()  # Membuat objek robot untuk mengontrol robot di Webots.

    camera = robot.getDevice("camera")  # Mengakses perangkat kamera robot.
    camera.enable(100)  # Mengaktifkan kamera dengan interval pengambilan gambar 100 ms.

    display = robot.getDevice("image display")  # Mengakses perangkat tampilan (display).

    return robot, camera, display  # Mengembalikan objek yang diinisialisasi.

def webots_image_to_numpy(im, h, w):
    """Mengubah gambar Webots ke format Numpy."""
    return np.frombuffer(im, dtype=np.uint8).reshape((h, w, 4))
    # Mengonversi gambar dari Webots menjadi array Numpy dengan dimensi (tinggi, lebar, 4).

def display_numpy_image(im, display, display_width, display_height):
    """Menampilkan gambar Numpy ke perangkat Display di Webots."""
    display_image = display.imageNew(
        im.tobytes(), Display.BGRA, display_width, display_height
    )
    # Membuat gambar baru untuk ditampilkan di display menggunakan format BGRA.
    display.imagePaste(display_image, 0, 0, blend=False)
    # Menempelkan gambar ke display di posisi (0, 0) tanpa pencampuran (blend).
    display.imageDelete(display_image)
    # Menghapus gambar dari memori setelah ditampilkan untuk efisiensi.
```

```
if __name__ == "__main__":
    # Program utama
    robot, camera, display = initialize()
    # Memanggil fungsi `initialize` untuk menyiapkan robot, kamera, dan tampilan.

    # Mendapatkan dimensi kamera dan tampilan
    camera_width = camera.getWidth()
    camera_height = camera.getHeight()
    display_width = display.getWidth()
    display_height = display.getHeight()

    # Menampilkan dimensi kamera dan tampilan di konsol
    print(f"Camera HxW: {camera_height}x{camera_width}")
    print(f"Display HxW: {display_height}x{display_width}")

    while robot.step(TIME_STEP) != -1:
        # Loop utama untuk menjalankan simulasi hingga dihentikan.

        # Mengambil gambar dari kamera dalam format Webots
        webots_im = camera.getImage()
        # Mengubah gambar menjadi format Numpy
        numpy_im = webots_image_to_numpy(webots_im, camera_height, camera_width)

        if SAVE_TO_DISK:
            # Jika `SAVE_TO_DISK` diaktifkan, simpan gambar ke disk.
            save_image(cv2.cvtColor(numpy_im, cv2.COLOR_RGB2BGR))

        # Segmentasi warna pada gambar untuk mendeteksi objek tertentu
        mask = segment_by_color(numpy_im, HSV_LOW_RANGE, HSV_UP_RANGE)

        try:
            # Meluruskan dokumen berdasarkan segmentasi warna
            document = get_warped_document(numpy_im, mask, debug=False)
            # Mengubah ukuran dan menambahkan padding agar sesuai dengan tampilan
            document = resize_and_letter_box(document, display_height, display_width)
        except ValueError as e:
            # Jika gagal mendeteksi dokumen, buat gambar kosong dengan latar belakang putih.
            document = np.zeros((display_width, display_height, 4), dtype=np.uint8)
```

```
                document[:, :, 0] = 255  # Mengatur warna latar belakang ke putih.
                print(e)  # Menampilkan pesan kesalahan.

            # Menampilkan gambar dokumen di perangkat tampilan Webots
            display_numpy_image(document, display, display_width, display_height)

    robot.cleanup()
    # Membersihkan sumber daya robot setelah simulasi selesai.
```

- from controller import Supervisor
  # Mengimpor kelas Supervisor dari Webots untuk mengontrol simulasi.

  from math import radians as rad
  # Mengimpor fungsi `radians` dari modul matematika untuk mengonversi derajat ke radian.
  # Fungsi ini diubah namanya menjadi `rad` untuk penggunaan yang lebih ringkas.

  import random
  # Mengimpor modul `random` untuk menghasilkan angka atau pilihan secara acak.

  TIME_STEP = 10000
  # Menentukan langkah waktu simulasi. Nilai ini menunjukkan interval waktu (dalam milidetik) antara setiap langkah simulasi.

  def initialize():
  """Inisialisasi Supervisor dan motor belt conveyor."""
  robot = Supervisor()
  # Membuat objek Supervisor untuk mengontrol seluruh elemen dalam simulasi.

  belt_motor = robot.getDevice("belt_motor")
  # Mengakses perangkat motor dengan nama "belt_motor" dari simulasi.

  belt_motor.setPosition(float("inf"))
  # Menyetel motor ke mode kecepatan tak terbatas (infinite position) agar dapat berputar terus-menerus.

  belt_motor.setVelocity(0.15)
  # Menetapkan kecepatan rotasi belt conveyor sebesar 0.15 unit.

  return robot
  # Mengembalikan objek Supervisor untuk digunakan dalam loop utama.

  def add_box(supervisor, model="1"):
  """

```
    Fungsi untuk menambahkan kotak kardus baru ke dalam simulasi.
    - supervisor: Objek Supervisor yang mengontrol simulasi.
    - model: Jenis model kotak kardus yang akan ditambahkan (default "1").
    """
    root_node = supervisor.getRoot()
    # Mendapatkan node root dari pohon adegan (scene tree), yang merupakan elemen teratas dalam hierarki simulasi.

    root_children_field = root_node.getField("children")
    # Mengakses kolom "children" dari node root, yang berisi semua objek di dalam simulasi.

    box_models = ["1", "2"]
    # Daftar model kotak kardus yang tersedia.

    # Menambahkan node baru berupa kotak kardus ke dalam kolom "children".
    root_children_field.importMFNodeFromString(
        -1,  # Menambahkan node di akhir daftar anak (index -1).
        """
        CardboardBox1 {{
        name "CardboardBox1 {rot}"
        translation -0.16 8.2 0.94
        rotation 0 0 1 {rot}
        }}
        """.format(
            rot=rad(random.randint(0, 360)),  # Menentukan rotasi acak dalam radian.
            model=random.choice(box_models)  # Memilih model kotak secara acak dari daftar `box_models`.
        ),
    )

if __name__ == "__main__":
    # Bagian utama program

    supervisor = initialize()
    # Memanggil fungsi `initialize` untuk menginisialisasi Supervisor dan motor.

    while supervisor.step(TIME_STEP) != -1:
        # Loop utama simulasi. Akan terus berjalan hingga simulasi dihentikan.

        add_box(supervisor)
        # Menambahkan kotak kardus baru ke dalam simulasi pada setiap langkah waktu.
```

```
    supervisor.cleanup()
    # Membersihkan sumber daya setelah simulasi selesai.
```

- ```
  # Impor pustaka standar
  import argparse
  # Modul argparse digunakan untuk menangani argumen baris perintah.

  # Impor pustaka eksternal
  import numpy as np
  import cv2
  from imutils import resize
  # NumPy digunakan untuk operasi array.
  # OpenCV digunakan untuk pemrosesan gambar.
  # imutils menyediakan fungsi bantu seperti resize untuk manipulasi gambar.

  def get_box_width(top_left, top_right, bottom_right, bottom_left):
      """Menghitung lebar kotak berdasarkan sudut-sudutnya."""
      x1, y1 = top_left
      x2, y2 = top_right
      # Menghitung lebar atas menggunakan jarak Euclidean
      width1 = np.hypot(x2 - x1, y2 - y1)

      x1, y1 = bottom_left
      x2, y2 = bottom_right
      # Menghitung lebar bawah menggunakan jarak Euclidean
      width2 = np.hypot(x2 - x1, y2 - y1)

      # Mengambil nilai maksimum antara lebar atas dan bawah
      width = int(max(width1, width2))

      return width

  def get_box_height(top_left, top_right, bottom_right, bottom_left):
      """Menghitung tinggi kotak berdasarkan sudut-sudutnya."""
      x1, y1 = top_left
      x2, y2 = bottom_left
      # Menghitung tinggi kiri menggunakan jarak Euclidean
      height1 = np.hypot(x2 - x1, y2 - y1)

      x1, y1 = top_right
      x2, y2 = bottom_right
      # Menghitung tinggi kanan menggunakan jarak Euclidean
      height2 = np.hypot(x2 - x1, y2 - y1)

      # Mengambil nilai maksimum antara tinggi kiri dan kanan
      height = int(max(height1, height2))
  ```
  ```

```
    return height

def identify_corners(approx_contour):
    """Mengidentifikasi sudut kotak dari kontur yang diaproksimasi."""
    # Mengurutkan titik berdasarkan jumlah koordinat x + y (menentukan posisi relatif)
    src_points = sorted(approx_contour, key=lambda p: p[0][0] + p[0][1])
    src_points = [p[0] for p in src_points]  # Mengambil koordinat dari setiap titik

    # Menentukan top-left dan bottom-right
    top_left, up1, up2, bottom_right = src_points

    # Menentukan top-right dan bottom-left berdasarkan koordinat y
    if up1[1] > top_left[1] and up1[0] < bottom_right[0]:
        bottom_left = up1
        top_right = up2
    else:
        bottom_left = up2
        top_right = up1

    return top_left, top_right, bottom_right, bottom_left

def resize_and_letter_box(image, rows, cols, channels=4):
    """
    Mengubah ukuran gambar sesuai proporsi tertentu dengan menambahkan letterbox.
    """
    image_rows, image_cols = image.shape[:2]
    # Rasio ukuran baru
    row_ratio = rows / float(image_rows)
    col_ratio = cols / float(image_cols)
    ratio = min(row_ratio, col_ratio)  # Mengambil rasio terkecil untuk menjaga proporsi

    # Mengubah ukuran gambar
    image_resized = cv2.resize(image, dsize=(0, 0), fx=ratio, fy=ratio)

    # Membuat bingkai hitam (letterbox) sesuai ukuran yang diinginkan
    letter_box = np.zeros((int(rows), int(cols), int(channels)), dtype=np.uint8)
    row_start = int((letter_box.shape[0] - image_resized.shape[0]) / 2)
    col_start = int((letter_box.shape[1] - image_resized.shape[1]) / 2)

    # Memasukkan gambar ke dalam bingkai
    letter_box[
        row_start : row_start + image_resized.shape[0],
```

```
        col_start : col_start + image_resized.shape[1],
    ] = image_resized
    return letter_box

def validate_image_shape(width, height):
    """
    Memvalidasi ukuran gambar untuk memastikan kotak terdeteksi dengan
benar.
    """
    if width > height:
        ratio = width / height
    else:
        ratio = height / width

    if height < 250 or width < 250:
        raise ValueError(f"Kotak terlalu kecil: H{height}xW{width}")
    elif ratio > 3:
        raise ValueError("Kotak terlalu tipis")

def segment_by_color(image, low_range, up_range):
    """
    Memisahkan gambar berdasarkan warna menggunakan ruang warna HSV.
    """
    if image.shape[2] == 4:
        image = cv2.cvtColor(image, cv2.COLOR_BGRA2BGR)  # Mengubah
gambar BGRA ke BGR
    im_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)  # Mengubah
gambar ke HSV
    mask = cv2.inRange(im_hsv, low_range, up_range)  # Membuat masker
berdasarkan rentang warna
    return mask

def get_warped_document(image, mask, debug=False):
    """
    Mendeteksi dokumen dari masker dan melakukan transformasi perspektif.
    """
    # Mendapatkan kontur dari masker
    contours, hierarchy = cv2.findContours(mask, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)

    # Mengambil kontur terbesar
    contours = sorted(contours, key=cv2.contourArea, reverse=True)[:3]

    for i in range(len(contours)):
        contour = contours[i]
        contour_length = cv2.arcLength(contour, closed=True)
```

```
        approx_contour = cv2.approxPolyDP(
            contour, epsilon=contour_length * 0.01, closed=True
        )

        # Berhenti jika kontur memiliki 4 titik (kotak)
        if len(approx_contour) == 4:
            break

    if debug:
        # Menampilkan kontur jika mode debug diaktifkan
        cv2.drawContours(image, [approx_contour], 0, color=(0, 0, 255),
thickness=10)
        cv2.imshow("Contours", resize(image, 540))
        cv2.waitKey(1)

    # Menentukan sudut dokumen
    top_left, top_right, bottom_right, bottom_left =
identify_corners(approx_contour)

    # Menghitung lebar dan tinggi dokumen
    width = get_box_width(top_left, top_right, bottom_right, bottom_left)
    height = get_box_height(top_left, top_right, bottom_right, bottom_left)

    validate_image_shape(width, height)

    # Menyesuaikan orientasi untuk transformasi perspektif
    if height > width:
        src_points = [top_left, top_right, bottom_right, bottom_left]
    else:
        src_points = [top_right, bottom_right, bottom_left, top_left]
        width, height = height, width

    src_points = np.array(src_points, dtype=np.float32)
    dst_points = np.array(
        [[0, 0], [width - 1, 0], [width - 1, height - 1], [0, height - 1]],
        dtype="float32",
    )

    # Menghitung matriks transformasi perspektif
    M = cv2.getPerspectiveTransform(src_points, dst_points)

    # Menerapkan transformasi
    warped = cv2.warpPerspective(
        image, M, (width, height), borderMode=cv2.BORDER_CONSTANT
    )
```

```
    return warped

if __name__ == "__main__":
    # Bagian utama program

    ap = argparse.ArgumentParser()
    # Membuat parser untuk argumen baris perintah
    ap.add_argument(
        "-i", "--image", required=True, type=str, help="Nama file input gambar"
    )
    args = vars(ap.parse_args())

    image_path = args["image"]
    # Membaca path gambar dari argumen

    image = cv2.imread(image_path)
    # Membaca gambar menggunakan OpenCV

    mask = segment_by_color(image, np.array([27, 0, 66]), np.array([180, 38,
255]))
    # Memisahkan area berdasarkan warna tertentu

    warped = get_warped_document(image, mask)
    # Melakukan transformasi perspektif pada dokumen

    cv2.imshow("image", resize(image, 540))
    cv2.waitKey(0)

    cv2.imshow("warped", resize(warped, 540))
    cv2.waitKey(0)
```

- Fruits Detection Robot with OpenCV and Webots

  Simulasi Fruits Detection Robot di aplikasi Webots menggunakan OpenCV merupakan salah satu implementasi robotik yang dirancang untuk mengenali berbagai jenis buah berdasarkan citra visual. Sistem ini menggabungkan kemampuan pemrosesan citra dengan algoritma deteksi dan klasifikasi objek, memungkinkan robot untuk mengidentifikasi buah-buahan secara otomatis. Simulasi ini memberikan lingkungan uji yang ideal untuk mengembangkan teknologi pengenalan objek tanpa perlu perangkat keras fisik.

  Simulasi di Webots memberikan keuntungan karena memungkinkan pengujian di berbagai skenario, seperti kondisi pencahayaan yang berbeda, variasi jarak, atau gangguan visual. Hal ini membantu mengembangkan robot yang lebih adaptif terhadap tantangan dunia nyata. Selain itu, simulasi dapat mencakup pengujian tugas lanjutan, seperti pengambilan buah menggunakan lengan robotik setelah deteksi berhasil dilakukan.

Secara keseluruhan, Fruits Detection Robot dengan OpenCV di Webots adalah inovasi yang signifikan dalam teknologi robotika berbasis visi. Simulasi ini tidak hanya mempercepat pengembangan sistem, tetapi juga membuka jalan bagi aplikasi di dunia nyata, seperti robot pemanen buah di sektor pertanian atau robot seleksi buah dalam industri makanan.

- ```
  /*
   * File:         fruit_ctrl.c
   */

  #include <webots/robot.h>        // Header untuk kontrol robot
  #include <webots/supervisor.h>    // Header untuk supervisor (kontrol atas dunia simulasi)
  #include <webots/camera.h>       // Header untuk kontrol kamera
  #include <stdio.h>               // Header untuk fungsi input/output standar
  #include <stdlib.h>              // Header untuk fungsi standar lainnya (misalnya, fungsi rand())

  #define TIME_STEP 64             // Interval waktu untuk setiap langkah simulasi (dalam milidetik)

  int main(int argc, char **argv) {
    int i = 0;                  // Variabel untuk perhitungan langkah simulasi
    int j = 8;                  // Variabel untuk menghitung jumlah apel
    int k = 8;                   // Variabel untuk menghitung jumlah jeruk
    int fr = 1;                  // Variabel untuk memilih jenis buah (1 = apel, 2 = jeruk)
    int max = 50;               // Batas maksimal jumlah buah yang dapat muncul
    char name[20];               // Array untuk menyimpan nama buah (apel atau jeruk)

    wb_robot_init();             // Inisialisasi robot Webots (memulai simulasi)

    // Mendeklarasikan dan mengaktifkan kamera
    WbDeviceTag camera = wb_robot_get_device("camera");  // Mendapatkan perangkat kamera dengan nama "camera"
    wb_camera_enable(camera, TIME_STEP);                 // Mengaktifkan kamera dengan interval TIME_STEP

    WbNodeRef fruit;             // Referensi node untuk buah
    WbFieldRef fruit_trans_field; // Referensi field untuk translasi posisi buah
    double fruit_initial_translation[3] = {0.570002, 2.85005, 0.349962};  // Posisi awal buah

    // Loop utama robot (akan terus berjalan selama simulasi)
    while (wb_robot_step(TIME_STEP) != -1) {
  ```
  ```

```
   // Setelah waktu 7.5 detik berlalu, program mulai menambahkan buah
   if (wb_robot_get_time() > 7.5){
     if (i == 0){
       if (fr > 0){
         // Memilih jenis buah secara acak (1 atau 2) antara apel dan jeruk
         fr = 1 + (rand() % 2);  // fr akan bernilai 1 atau 2 (apel atau jeruk)

         // Mengatur buah yang akan ditambahkan sesuai dengan jumlah yang
sudah ada
         if (j == max) fr = 2;  // Jika jumlah apel mencapai batas, pilih jeruk
         if (k == max) fr = 1;  // Jika jumlah jeruk mencapai batas, pilih apel

         // Menambahkan apel jika fr == 1 dan jumlah apel masih kurang dari
max
         if ((fr == 1) && (j < max)) {
           sprintf(name, "apple%d", j);  // Menyusun nama apel
           j += 1;  // Menambah jumlah apel
         }

         // Menambahkan jeruk jika fr == 2 dan jumlah jeruk masih kurang dari
max
         if ((fr == 2) && (k < max)) {
           sprintf(name, "orange%d", k);  // Menyusun nama jeruk
           k += 1;  // Menambah jumlah jeruk
         }

         // Mendapatkan node buah yang baru berdasarkan nama yang telah
dibuat
         fruit = wb_supervisor_node_get_from_def(name);
         // Mendapatkan field translasi untuk mengubah posisi buah
         fruit_trans_field = wb_supervisor_node_get_field(fruit, "translation");
         // Mengatur posisi buah ke posisi awal yang telah ditentukan
         wb_supervisor_field_set_sf_vec3f(fruit_trans_field,
fruit_initial_translation);

         // Jika jumlah apel dan jeruk mencapai batas maksimal, hentikan
penambahan buah
         if ((j == max) && (k == max)) fr = 0;
       }
     }

     i += 1; // Meningkatkan penghitung langkah simulasi
     if (i == 120) i = 0; // Reset penghitung jika mencapai 120
   }

  };
```

```
  wb_robot_cleanup(); // Membersihkan dan menutup simulasi robot

  return 0; // Mengembalikan 0, menandakan program selesai
}
```

- ```
  /*
   * File:         fruit_sorting_controller_V1.c
   */

  #include <webots/robot.h>              // Header untuk kontrol robot
  #include <webots/distance_sensor.h>    // Header untuk sensor jarak
  #include <webots/position_sensor.h>    // Header untuk sensor posisi
  #include <webots/camera.h>             // Header untuk kontrol kamera
  #include <webots/camera_recognition_object.h> // Header untuk objek
  pengenalan kamera
  #include <webots/motor.h>              // Header untuk kontrol motor
  #include <webots/supervisor.h>         // Header untuk supervisor yang
  mengontrol dunia simulasi

  #include <stdio.h>                     // Header untuk fungsi input/output standar

  #define TIME_STEP 32                   // Interval waktu untuk setiap langkah
  simulasi (dalam milidetik)

  enum State {WAITING, PICKING, ROTATING, DROPPING,
  ROTATE_BACK}; // Enum untuk berbagai state pada lengan robot

  // Fungsi utama program
  int main(int argc, char **argv) {
    wb_robot_init();  // Inisialisasi robot dan memulai simulasi

    int counter = 0, i = 0;  // Variabel penghitung dan loop
    int state = WAITING;     // Variabel untuk menyimpan state lengan robot
    const double target_positions[] = {-1.570796, -1.87972, -2.139774, -
  2.363176, -1.50971};  // Posisi target untuk lengan robot

    double speed = 2.0;      // Kecepatan motor robot
    int model = 0;           // Variabel untuk menentukan jenis buah (0 = jeruk, 1 =
  apel)
    char fruit[20];          // Nama buah yang dikenali
    int apple = 0;           // Hitung jumlah apel
    int orange = 0;          // Hitung jumlah jeruk

    char strP[100];          // Variabel untuk menampilkan string pada supervisor

    // Membaca parameter kecepatan dari argumen input (jika ada)
  ```
  ```

```
  if (argc == 2)
    sscanf(argv[1], "%lf", &speed);  // Membaca kecepatan jika diberikan
sebagai argumen

  // Mendapatkan dan mendeklarasikan motor jari tangan untuk gripper
(penjepit)
  WbDeviceTag hand_motors[3];
  hand_motors[0] = wb_robot_get_device("finger_1_joint_1");
  hand_motors[1] = wb_robot_get_device("finger_2_joint_1");
  hand_motors[2] = wb_robot_get_device("finger_middle_joint_1");

  // Mendapatkan dan mendeklarasikan motor untuk lengan robot
  WbDeviceTag ur_motors[5];
  ur_motors[0] = wb_robot_get_device("shoulder_pan_joint");
  ur_motors[1] = wb_robot_get_device("shoulder_lift_joint");
  ur_motors[2] = wb_robot_get_device("elbow_joint");
  ur_motors[3] = wb_robot_get_device("wrist_1_joint");
  ur_motors[4] = wb_robot_get_device("wrist_2_joint");

  // Mengaktifkan kamera untuk pengenalan objek
  WbDeviceTag camera = wb_robot_get_device("camera");
  wb_camera_enable(camera, 2 * TIME_STEP);  // Mengaktifkan kamera
dengan pengambilan gambar setiap 2 * TIME_STEP
  wb_camera_recognition_enable(camera, 2 * TIME_STEP);  // Mengaktifkan
pengenalan objek kamera

  // Mengatur kecepatan motor lengan robot
  for (i = 0; i < 5; ++i)
    wb_motor_set_velocity(ur_motors[i], speed);

  // Mendapatkan dan mengaktifkan sensor jarak
  WbDeviceTag distance_sensor = wb_robot_get_device("distance sensor");
  wb_distance_sensor_enable(distance_sensor, TIME_STEP);

  // Mendapatkan dan mengaktifkan sensor posisi untuk pergelangan tangan
  WbDeviceTag position_sensor =
wb_robot_get_device("wrist_1_joint_sensor");
  wb_position_sensor_enable(position_sensor, TIME_STEP);

  // Loop utama simulasi
  while (wb_robot_step(TIME_STEP) != -1) {

    // Mendapatkan detail pengenalan objek dari kamera
    int number_of_objects =
wb_camera_recognition_get_number_of_objects(camera);
```

```
  // Mendapatkan informasi objek yang terdeteksi
  const WbCameraRecognitionObject *objects =
wb_camera_recognition_get_objects(camera);

  // Jika ada objek yang terdeteksi, tentukan jenis buah
  if (number_of_objects > 0) {
    sprintf(fruit, "%s", objects[0].model);  // Menyimpan model buah yang
terdeteksi
    if (fruit[0] == 97) model = 1;  // Jika buahnya apel (kode ASCII untuk 'a')
    else model = 0;  // Jika buahnya jeruk
  }

  // Proses kontrol berdasarkan state lengan robot
  if (counter <= 0) {
    switch (state) {
      case WAITING:  // Menunggu objek terdeteksi
        if (wb_distance_sensor_get_value(distance_sensor) < 500) {  // Jika
jarak kurang dari 500 (ada objek di depan)
          state = PICKING;  // Pindah ke state PICKING (menjepit objek)
          if (model == 1) apple += 1;  // Jika buah adalah apel, tambahkan ke
hitungan apel
          else orange += 1;  // Jika buah adalah jeruk, tambahkan ke hitungan
jeruk
          counter = 8;  // Set penghitung untuk menunggu sebelum berpindah ke
state berikutnya
          for (i = 0; i < 3; ++i)
            wb_motor_set_position(hand_motors[i], 0.52);  // Posisi tangan untuk
mengambil buah
        }
        break;
      case PICKING:  // Menjepit objek
        for (i = model; i < 5; ++i)
          wb_motor_set_position(ur_motors[i], target_positions[i]);  // Mengatur
posisi lengan robot untuk mengambil buah
        state = ROTATING;  // Pindah ke state ROTATING (memutar lengan
untuk menempatkan buah)
        break;
      case ROTATING:  // Memutar lengan robot
        if (wb_position_sensor_get_value(position_sensor) < -2.3) {  // Jika
posisi pergelangan tangan cukup rendah
          counter = 8;  // Set penghitung untuk menunggu sebelum melanjutkan
ke state DROPPING
          state = DROPPING;  // Pindah ke state DROPPING (menjatuhkan
buah)
          for (i = 0; i < 3; ++i)
```

```
            wb_motor_set_position(hand_motors[i],
wb_motor_get_min_position(hand_motors[i]));  // Buka tangan untuk
menjatuhkan buah
          }
          break;
        case DROPPING:  // Menjatuhkan buah
          for (int i = model; i < 5; ++i)
            wb_motor_set_position(ur_motors[i], 0.0);  // Mengembalikan posisi
lengan ke semula
          state = ROTATE_BACK;  // Pindah ke state ROTATE_BACK (kembali
ke posisi semula)
          break;
        case ROTATE_BACK:  // Kembali ke posisi semula
          if (wb_position_sensor_get_value(position_sensor) > -0.1) {
            state = WAITING;  // Jika posisi sudah kembali ke semula, pindah ke
state WAITING
          }
          break;
      }
    }
    counter--;  // Mengurangi penghitung setiap langkah simulasi

    // Menampilkan jumlah apel dan jeruk di supervisor
    sprintf(strP, "Oranges: %d", orange);
    wb_supervisor_set_label(0, strP, 0.45, 0.96, 0.06, 0x5555ff, 0, "Lucida
Console");

    sprintf(strP, "Apples : %d", apple);
    wb_supervisor_set_label(1, strP, 0.3, 0.96, 0.06, 0x5555ff, 0, "Lucida
Console");

  }
  wb_robot_cleanup();  // Membersihkan dan menutup simulasi robot

  return 0;  // Mengembalikan 0, menandakan program selesai
}
```

- ```
  # Import library yang diperlukan
  import cv2  # Library untuk pengolahan citra dan visi komputer (OpenCV)
  import numpy as np  # Library untuk operasi numerik dan array
  from controller import Supervisor, DistanceSensor, PositionSensor, Camera,
  Motor  # API Webots untuk mengontrol robot

  # Set waktu simulasi per langkah
  TIME_STEP = 32

  # Definisikan status yang dapat diambil oleh robot arm
  ```
  ```

```
WAITING, PICKING, ROTATING, DROPPING, ROTATE_BACK =
range(5)

# Fungsi untuk memproses gambar menggunakan OpenCV
def process_image_with_opencv(image_data, width, height):
    # Konversi gambar dari format Webots (BGRA) ke format OpenCV (BGR)
    img = np.frombuffer(image_data, np.uint8).reshape((height, width, 4))
    img = cv2.cvtColor(img, cv2.COLOR_BGRA2BGR)

    # Konversi gambar dari format BGR ke format HSV (Hue, Saturation,
Value)
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Tentukan rentang warna HSV untuk mendeteksi buah hijau (apel)
    lower_green = np.array([35, 50, 50])  # Rentang nilai bawah untuk warna
hijau
    upper_green = np.array([85, 255, 255])  # Rentang nilai atas untuk warna
hijau
    green_mask = cv2.inRange(hsv, lower_green, upper_green)  # Membuat
mask untuk warna hijau

    # Tentukan rentang warna HSV untuk mendeteksi buah orange (jeruk)
    lower_orange = np.array([10, 100, 100])  # Rentang nilai bawah untuk
warna oranye
    upper_orange = np.array([25, 255, 255])  # Rentang nilai atas untuk warna
oranye
    orange_mask = cv2.inRange(hsv, lower_orange, upper_orange)  # Membuat
mask untuk warna oranye

    # Gabungkan kedua mask untuk mendeteksi buah hijau dan oranye
    combined_mask = cv2.bitwise_or(green_mask, orange_mask)

    # Temukan kontur-kontur yang terdeteksi dalam mask
    contours, _ = cv2.findContours(combined_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    detected_fruit = None  # Variabel untuk menyimpan jenis buah yang
terdeteksi

    # Iterasi melalui setiap kontur yang ditemukan
    for contour in contours:
        # Hitung panjang keliling kontur
        perimeter = cv2.arcLength(contour, True)
        # Perkirakan kontur menjadi bentuk polygon
        approx = cv2.approxPolyDP(contour, 0.04 * perimeter, True)
```

```
        # Hitung luas kontur dan abaikan kontur yang terlalu kecil
        area = cv2.contourArea(contour)
        if area < 500:
            continue

        # Cek apakah bentuknya kira-kira bulat (untuk jeruk)
        if len(approx) > 8:
            detected_fruit = "orange"  # Jika bentuknya bulat, deteksi sebagai jeruk
            color_mask = orange_mask  # Gunakan mask oranye
        else:
            detected_fruit = "apple"  # Jika bentuknya tidak bulat, deteksi sebagai
apel
            color_mask = green_mask  # Gunakan mask hijau

        # Opsional: gambar kontur pada gambar untuk debugging
        cv2.drawContours(img, [contour], -1, (0, 255, 0), 2)

    # Tampilkan gambar yang telah diproses (untuk debugging)
    cv2.imshow("Detected Shapes", img)
    cv2.waitKey(1)  # Tampilkan gambar selama 1ms

    return detected_fruit  # Kembalikan jenis buah yang terdeteksi

# Fungsi utama program
def main():
    supervisor = Supervisor()  # Inisialisasi objek Supervisor untuk mengontrol
robot

    counter = 0  # Variabel untuk menghitung waktu antara perubahan status
    state = WAITING  # Status awal robot arm adalah WAITING
    target_positions = [-1.570796, -1.87972, -2.139774, -2.363176, -1.50971]  #
Posisi target untuk setiap joint robot

    speed = 2.0  # Kecepatan gerakan robot
    model = 0  # Model buah yang terdeteksi (0 = jeruk, 1 = apel)
    apple = 0  # Variabel untuk menghitung jumlah apel yang terdeteksi
    orange = 0  # Variabel untuk menghitung jumlah jeruk yang terdeteksi

    # Mengambil dan mendeklarasikan motor jari-jari gripper
    hand_motors = []
    hand_motors.append(supervisor.getDevice("finger_1_joint_1"))
    hand_motors.append(supervisor.getDevice("finger_2_joint_1"))
    hand_motors.append(supervisor.getDevice("finger_middle_joint_1"))

    # Mengambil dan mendeklarasikan motor lengan robot
    ur_motors = []
```

```
    ur_motors.append(supervisor.getDevice("shoulder_pan_joint"))
    ur_motors.append(supervisor.getDevice("shoulder_lift_joint"))
    ur_motors.append(supervisor.getDevice("elbow_joint"))
    ur_motors.append(supervisor.getDevice("wrist_1_joint"))
    ur_motors.append(supervisor.getDevice("wrist_2_joint"))

    # Mengaktifkan dan mengonfigurasi kamera untuk pengenalan objek
    camera = supervisor.getDevice("camera")
    camera.enable(2 * TIME_STEP)  # Mengaktifkan kamera dengan waktu
step
    camera.recognitionEnable(2 * TIME_STEP)  # Mengaktifkan pengenalan
objek pada kamera

    # Set kecepatan motor robot
    for motor in ur_motors:
        motor.setVelocity(speed)

    # Mengaktifkan sensor jarak untuk deteksi objek
    distance_sensor = supervisor.getDevice("distance sensor")
    distance_sensor.enable(TIME_STEP)

    # Mengaktifkan sensor posisi untuk memantau posisi sendi pergelangan
tangan robot
    position_sensor = supervisor.getDevice("wrist_1_joint_sensor")
    position_sensor.enable(TIME_STEP)

    # Loop utama
    while supervisor.step(TIME_STEP) != -1:
        # Ambil data gambar dari kamera
        image_data = camera.getImage()
        width = camera.getWidth()  # Ambil lebar gambar
        height = camera.getHeight()  # Ambil tinggi gambar

        # Proses gambar untuk mendeteksi buah
        detected_fruit = process_image_with_opencv(image_data, width, height)

        # Update penghitung buah berdasarkan hasil deteksi
        if detected_fruit == "apple":
            model = 1  # Set model menjadi apel
            apple += 1  # Increment jumlah apel
        elif detected_fruit == "orange":
            model = 0  # Set model menjadi jeruk
            orange += 1  # Increment jumlah jeruk

        # Mesin status untuk mengontrol perilaku lengan robot
        if counter <= 0:
```

```
            if state == WAITING:
                # Jika jarak deteksi sensor lebih kecil dari 500, robot mulai
mengambil buah
                if distance_sensor.getValue() < 500:
                    state = PICKING
                    counter = 8
                    for motor in hand_motors:
                        motor.setPosition(0.52)  # Posisi motor tangan saat mengambil
            elif state == PICKING:
                # Gerakkan motor lengan robot ke posisi target untuk mengambil
buah
                for i in range(model, 5):
                    ur_motors[i].setPosition(target_positions[i])
                state = ROTATING
            elif state == ROTATING:
                # Jika posisi sendi wrist_1 sudah mencapai nilai tertentu, robot siap
menjatuhkan buah
                if position_sensor.getValue() < -2.3:
                    counter = 8
                    state = DROPPING
                    for motor in hand_motors:
                        motor.setPosition(motor.getMinPosition())  # Lepaskan buah
            elif state == DROPPING:
                # Set posisi motor untuk menjatuhkan buah
                for i in range(model, 5):
                    ur_motors[i].setPosition(0.0)
                state = ROTATE_BACK
            elif state == ROTATE_BACK:
                # Kembalikan posisi lengan robot
                if position_sensor.getValue() > -0.1:
                    state = WAITING

        counter -= 1  # Kurangi counter setiap langkah simulasi

        # Tampilkan jumlah jeruk dan apel pada layar supervisor
        strP = f"Oranges: {orange}"
        supervisor.setLabel(0, strP, 0.45, 0.96, 0.06, 0x5555ff, 0, "Lucida
Console")

        strP = f"Apples: {apple}"
        supervisor.setLabel(1, strP, 0.3, 0.96, 0.06, 0x5555ff, 0, "Lucida
Console")

    # Bersihkan setelah simulasi selesai
    supervisor.cleanup()
```

```
# Jalankan fungsi utama
if __name__ == "__main__":
    main()
```

- ```
  # Import library yang diperlukan
  import cv2  # Library untuk pengolahan citra dan visi komputer (OpenCV)
  import numpy as np  # Library untuk operasi numerik dan array
  from controller import Supervisor, DistanceSensor, PositionSensor, Camera,
  Motor  # API Webots untuk mengontrol robot

  # Set waktu simulasi per langkah
  TIME_STEP = 32

  # Definisikan status yang dapat diambil oleh robot arm
  WAITING, PICKING, ROTATING, DROPPING, ROTATE_BACK =
  range(5)

  # Fungsi untuk memproses gambar menggunakan OpenCV
  def process_image_with_opencv(image_data, width, height):
      # Mengonversi gambar dari format Webots (BGRA) ke format OpenCV
  (BGR)
      img = np.frombuffer(image_data, np.uint8).reshape((height, width, 4))
      img = cv2.cvtColor(img, cv2.COLOR_BGRA2BGR)

      # Contoh pemrosesan gambar: Konversi ke grayscale dan deteksi tepi
      gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Mengonversi
  gambar ke grayscale
      edges = cv2.Canny(gray, 100, 200)  # Menggunakan deteksi tepi Canny
  untuk mendeteksi tepi pada gambar

      # Menampilkan gambar yang telah diproses menggunakan OpenCV
  (opsional, untuk debugging)
      cv2.imshow("Edges", edges)  # Menampilkan hasil deteksi tepi
      cv2.waitKey(1)  # Tampilkan gambar selama 1ms untuk debugging

      # Anda bisa menambahkan logika pemrosesan gambar lainnya di sini
      # Mengembalikan hasil yang mungkin berguna untuk robot
      return edges

  # Fungsi utama program
  def main():
      supervisor = Supervisor()  # Inisialisasi objek Supervisor untuk mengontrol
  robot

      counter = 0  # Variabel untuk menghitung waktu antara perubahan status
      state = WAITING  # Status awal robot arm adalah WAITING
  ```
  ```

```
    target_positions = [-1.570796, -1.87972, -2.139774, -2.363176, -1.50971]  # Posisi target untuk setiap joint robot

    speed = 2.0  # Kecepatan gerakan robot
    model = 0  # Model buah yang terdeteksi (0 = jeruk, 1 = apel)
    fruit = "  # Variabel untuk menyimpan jenis buah yang terdeteksi
    apple = 0  # Variabel untuk menghitung jumlah apel yang terdeteksi
    orange = 0  # Variabel untuk menghitung jumlah jeruk yang terdeteksi

    # Mendeklarasikan dan mengakses motor jari-jari gripper
    hand_motors = []
    hand_motors.append(supervisor.getDevice("finger_1_joint_1"))  # Motor untuk jari pertama
    hand_motors.append(supervisor.getDevice("finger_2_joint_1"))  # Motor untuk jari kedua
    hand_motors.append(supervisor.getDevice("finger_middle_joint_1"))  # Motor untuk jari tengah

    # Mendeklarasikan dan mengakses motor lengan robot
    ur_motors = []
    ur_motors.append(supervisor.getDevice("shoulder_pan_joint"))  # Motor untuk sendi bahu
    ur_motors.append(supervisor.getDevice("shoulder_lift_joint"))  # Motor untuk sendi angkat bahu
    ur_motors.append(supervisor.getDevice("elbow_joint"))  # Motor untuk sendi siku
    ur_motors.append(supervisor.getDevice("wrist_1_joint"))  # Motor untuk sendi pergelangan tangan 1
    ur_motors.append(supervisor.getDevice("wrist_2_joint"))  # Motor untuk sendi pergelangan tangan 2

    # Mendeklarasikan dan mengaktifkan kamera untuk pengenalan objek
    camera = supervisor.getDevice("camera")  # Mengakses perangkat kamera robot
    camera.enable(2 * TIME_STEP)  # Mengaktifkan kamera dengan waktu step
    camera.recognitionEnable(2 * TIME_STEP)  # Mengaktifkan pengenalan objek pada kamera

    # Mengatur kecepatan motor lengan robot
    for motor in ur_motors:
        motor.setVelocity(speed)

    # Mengaktifkan sensor jarak untuk deteksi objek di sekitar robot
    distance_sensor = supervisor.getDevice("distance sensor")
    distance_sensor.enable(TIME_STEP)
```

```
    # Mengaktifkan sensor posisi untuk memantau posisi sendi pergelangan tangan robot
    position_sensor = supervisor.getDevice("wrist_1_joint_sensor")
    position_sensor.enable(TIME_STEP)

    # Loop utama untuk menjalankan simulasi robot
    while supervisor.step(TIME_STEP) != -1:
        # Ambil data gambar dari kamera
        image_data = camera.getImage()  # Mendapatkan data gambar dari kamera
        width = camera.getWidth()  # Mendapatkan lebar gambar
        height = camera.getHeight()  # Mendapatkan tinggi gambar

        # Proses gambar dengan OpenCV
        edges = process_image_with_opencv(image_data, width, height)  # Memanggil fungsi untuk memproses gambar

        # Mendapatkan informasi pengenalan objek dari kamera
        number_of_objects = camera.getRecognitionNumberOfObjects()  # Mengambil jumlah objek yang terdeteksi

        # Mengambil informasi objek yang terdeteksi dan menentukan jenis buah
        objects = camera.getRecognitionObjects()  # Mendapatkan objek yang terdeteksi
        if number_of_objects > 0:  # Jika ada objek yang terdeteksi
            fruit = objects[0].getModel()  # Mendapatkan model objek pertama yang terdeteksi
            if fruit[0] == 'a':  # ASCII nilai dari 'a' adalah 97, artinya objek pertama adalah apel
                model = 1  # Set model menjadi apel
            else:
                model = 0  # Set model menjadi jeruk

        # Mesin status untuk mengontrol perilaku lengan robot
        if counter <= 0:
            if state == WAITING:  # Jika status robot adalah WAITING
                if distance_sensor.getValue() < 500:  # Jika sensor jarak mendeteksi objek lebih dekat dari 500
                    state = PICKING  # Ubah status menjadi PICKING (mengambil buah)
                    if model == 1:  # Jika buah yang terdeteksi adalah apel
                        apple += 1  # Tambah jumlah apel
                    else:  # Jika buah yang terdeteksi adalah jeruk
                        orange += 1  # Tambah jumlah jeruk
                    counter = 8  # Set waktu counter untuk menunggu
```

```
                for motor in hand_motors:
                    motor.setPosition(0.52)  # Posisi motor jari tangan saat mengambil buah
            elif state == PICKING:  # Jika status robot adalah PICKING
                for i in range(model, 5):  # Gerakkan motor lengan robot ke posisi yang sesuai
                    ur_motors[i].setPosition(target_positions[i])
                state = ROTATING  # Ubah status menjadi ROTATING (memutar lengan)
            elif state == ROTATING:  # Jika status robot adalah ROTATING
                if position_sensor.getValue() < -2.3:  # Jika posisi sendi pergelangan tangan mencapai nilai tertentu
                    counter = 8  # Set waktu counter untuk menunggu
                    state = DROPPING  # Ubah status menjadi DROPPING (menjatuhkan buah)
                    for motor in hand_motors:
                        motor.setPosition(motor.getMinPosition())  # Lepaskan buah
            elif state == DROPPING:  # Jika status robot adalah DROPPING
                for i in range(model, 5):  # Kembalikan posisi motor lengan robot
                    ur_motors[i].setPosition(0.0)
                state = ROTATE_BACK  # Ubah status menjadi ROTATE_BACK (kembali ke posisi awal)
            elif state == ROTATE_BACK:  # Jika status robot adalah ROTATE_BACK
                if position_sensor.getValue() > -0.1:  # Jika posisi sendi pergelangan tangan kembali ke posisi awal
                    state = WAITING  # Ubah status kembali ke WAITING

        counter -= 1  # Kurangi counter setiap langkah simulasi

        # Tampilkan jumlah jeruk dan apel pada layar supervisor
        strP = f"Oranges: {orange}"  # Menampilkan jumlah jeruk
        supervisor.setLabel(0, strP, 0.45, 0.96, 0.06, 0x5555ff, 0, "Lucida Console")

        strP = f"Apples: {apple}"  # Menampilkan jumlah apel
        supervisor.setLabel(1, strP, 0.3, 0.96, 0.06, 0x5555ff, 0, "Lucida Console")

    # Bersihkan setelah simulasi selesai
    supervisor.cleanup()

# Jalankan fungsi utama
if __name__ == "__main__":
    main()
```

- # File: fruit_sorting_controller_V1.py

```
# Mengimpor modul-modul yang dibutuhkan dari Webots untuk mengontrol robot
from controller import Supervisor, DistanceSensor, PositionSensor, Camera, Motor

# Waktu per langkah simulasi (dalam milidetik)
TIME_STEP = 32

# Menentukan status robot (state) yang digunakan dalam mesin status
WAITING, PICKING, ROTATING, DROPPING, ROTATE_BACK = range(5)

# Fungsi utama program
def main():
    supervisor = Supervisor()  # Membuat objek Supervisor untuk mengontrol robot

    counter = 0  # Variabel untuk menghitung waktu antara perubahan status
    state = WAITING  # Status awal robot adalah WAITING
    target_positions = [-1.570796, -1.87972, -2.139774, -2.363176, -1.50971]  # Posisi target untuk setiap joint robot

    speed = 2.0  # Kecepatan motor lengan robot
    model = 0  # Menyimpan model buah yang terdeteksi (0 untuk jeruk, 1 untuk apel)
    fruit = ''  # Variabel untuk menyimpan jenis buah yang terdeteksi
    apple = 0  # Variabel untuk menghitung jumlah apel yang terdeteksi
    orange = 0  # Variabel untuk menghitung jumlah jeruk yang terdeteksi

    # Mengakses dan mendeklarasikan motor jari-jari gripper
    hand_motors = []
    hand_motors.append(supervisor.getDevice("finger_1_joint_1"))  # Motor untuk jari pertama
    hand_motors.append(supervisor.getDevice("finger_2_joint_1"))  # Motor untuk jari kedua
    hand_motors.append(supervisor.getDevice("finger_middle_joint_1"))  # Motor untuk jari tengah

    # Mengakses dan mendeklarasikan motor lengan robot
    ur_motors = []
    ur_motors.append(supervisor.getDevice("shoulder_pan_joint"))  # Motor untuk sendi bahu
    ur_motors.append(supervisor.getDevice("shoulder_lift_joint"))  # Motor untuk sendi angkat bahu
```

```
    ur_motors.append(supervisor.getDevice("elbow_joint"))  # Motor untuk sendi siku
    ur_motors.append(supervisor.getDevice("wrist_1_joint"))  # Motor untuk sendi pergelangan tangan 1
    ur_motors.append(supervisor.getDevice("wrist_2_joint"))  # Motor untuk sendi pergelangan tangan 2

    # Mengakses dan mengaktifkan kamera untuk pengenalan objek
    camera = supervisor.getDevice("camera")  # Mengakses perangkat kamera robot
    camera.enable(2 * TIME_STEP)  # Mengaktifkan kamera dengan waktu step
    camera.recognitionEnable(2 * TIME_STEP)  # Mengaktifkan pengenalan objek pada kamera

    # Mengatur kecepatan motor lengan robot
    for motor in ur_motors:
        motor.setVelocity(speed)

    # Mengaktifkan sensor jarak untuk deteksi objek di sekitar robot
    distance_sensor = supervisor.getDevice("distance sensor")
    distance_sensor.enable(TIME_STEP)

    # Mengaktifkan sensor posisi untuk memantau posisi sendi pergelangan tangan robot
    position_sensor = supervisor.getDevice("wrist_1_joint_sensor")
    position_sensor.enable(TIME_STEP)

    # Loop utama untuk menjalankan simulasi robot
    while supervisor.step(TIME_STEP) != -1:
        # Mengambil jumlah objek yang terdeteksi oleh kamera
        number_of_objects = camera.getRecognitionNumberOfObjects()

        # Mengambil informasi tentang objek yang terdeteksi
        objects = camera.getRecognitionObjects()

        # Jika ada objek yang terdeteksi
        if number_of_objects > 0:
            fruit = objects[0].getModel()  # Mendapatkan model objek pertama yang terdeteksi
            if fruit[0] == 'a':  # Jika huruf pertama model objek adalah 'a' (yang berarti apel)
                model = 1  # Set model menjadi apel
            else:
                model = 0  # Set model menjadi jeruk
```

```
    # Mesin status untuk mengontrol perilaku lengan robot
    if counter <= 0:
        if state == WAITING:  # Jika status robot adalah WAITING
            if distance_sensor.getValue() < 500:  # Jika sensor jarak mendeteksi
objek lebih dekat dari 500
                state = PICKING  # Ubah status menjadi PICKING (mengambil
buah)
                if model == 1:  # Jika buah yang terdeteksi adalah apel
                    apple += 1  # Tambah jumlah apel
                else:  # Jika buah yang terdeteksi adalah jeruk
                    orange += 1  # Tambah jumlah jeruk
                counter = 8  # Set waktu counter untuk menunggu
                for motor in hand_motors:
                    motor.setPosition(0.52)  # Posisi motor jari tangan saat
mengambil buah
        elif state == PICKING:  # Jika status robot adalah PICKING
            for i in range(model, 5):  # Gerakkan motor lengan robot ke posisi
yang sesuai
                ur_motors[i].setPosition(target_positions[i])
            state = ROTATING  # Ubah status menjadi ROTATING (memutar
lengan)
        elif state == ROTATING:  # Jika status robot adalah ROTATING
            if position_sensor.getValue() < -2.3:  # Jika posisi sendi pergelangan
tangan mencapai nilai tertentu
                counter = 8  # Set waktu counter untuk menunggu
                state = DROPPING  # Ubah status menjadi DROPPING
(menjatuhkan buah)
                for motor in hand_motors:
                    motor.setPosition(motor.getMinPosition())  # Lepaskan buah
        elif state == DROPPING:  # Jika status robot adalah DROPPING
            for i in range(model, 5):  # Kembalikan posisi motor lengan robot
                ur_motors[i].setPosition(0.0)
            state = ROTATE_BACK  # Ubah status menjadi ROTATE_BACK
(kembali ke posisi awal)
        elif state == ROTATE_BACK:  # Jika status robot adalah
ROTATE_BACK
            if position_sensor.getValue() > -0.1:  # Jika posisi sendi pergelangan
tangan kembali ke posisi awal
                state = WAITING  # Ubah status kembali ke WAITING

    counter -= 1  # Kurangi counter setiap langkah simulasi

    # Menampilkan jumlah jeruk dan apel pada layar supervisor
    strP = f"Oranges: {orange}"  # Menampilkan jumlah jeruk
    supervisor.setLabel(0, strP, 0.45, 0.96, 0.06, 0x5555ff, 0, "Lucida
Console")
```

```
        strP = f"Apples: {apple}"  # Menampilkan jumlah apel
        supervisor.setLabel(1, strP, 0.3, 0.96, 0.06, 0x5555ff, 0, "Lucida
Console")

    # Bersihkan setelah simulasi selesai
    supervisor.cleanup()

# Fungsi utama dijalankan ketika script ini dieksekusi
if __name__ == "__main__":
    main()
```