HEIGHT ESTIMATION USING ARUCO MARKER

MEN2020 REPORT
GARVIT MEENA (B20ME033)
Supervisor: Dr. Jayant Kumar Mohanta

Link to colab: image detection.ipynb

PART-1: HEIGHT DETECTION

Introduction:

The code provides an estimated height of a person in the image based on the assumption that two ArUco markers representing the head and foot are detected.
The visual output includes the original image with detected markers, a line connecting the head and foot markers, and the estimated height displayed on the image.
The estimated height is also printed to the console.

Explanation of code:

1. Used OpenCV library to do the image detection.

```
[1]  1 from google.colab.patches import cv2_imshow
     2 import cv2
     3 import cv2.aruco as aruco
     4 import numpy as np
```

2. Function Definition:

def calculate_distance(point1, point2): This line defines the function calculate_distance that takes two points (point1 and point2) as input.

Variable Extraction:

x1, y1 = point1[0], point1[1]: Extracts the x and y coordinates of point1.
x2, y2 = point2[0], point2[1]: Extracts the x and y coordinates of point2.

Distance Calculation:

distance = np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2): Calculates the Euclidean distance between the two points using the formula:

$\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$.

```python
def calculate_distance(point1, point2):
    x1, y1 = point1[0], point1[1]
    x2, y2 = point2[0], point2[1]
    distance = np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
    return distance
```

3. Convert Image to Grayscale:

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY): Converts the input color image (image) to grayscale using OpenCV's cvtColor function.

Define ArUco Dictionary and Parameters:

aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_4X4_100): Creates an instance of the ArUco dictionary with a 4x4 grid and 100 different marker codes.
parameters = aruco.DetectorParameters(): Creates an instance of the DetectorParameters class, which provides parameters for the ArUco marker detection algorithm.

Detect ArUco Markers:

corners, ids, _ = aruco.detectMarkers(gray, aruco_dict, parameters=parameters): Detects ArUco markers in the grayscale image using the specified dictionary and parameters. Returns the corner coordinates (corners), marker IDs (ids), and rejected candidates (not used here).

Convert Corner Coordinates to Integers:

int_corners = np.int0(corners): Converts the floating-point corner coordinates to integers using NumPy's np.int0 function. This is often done to facilitate drawing the markers on the image.

Calculate Perimeter:

perimeter = cv2.arcLength(corners[0], True): Calculates the perimeter (arc length) of the contour formed by the corner coordinates of the first detected marker. This can be useful for further calculations or visualizations.

Return Detected Information:

return corners, ids, perimeter: Returns the detected corner coordinates, marker IDs, and perimeter from the function.

```
[3]  1 def detect_markers(image):
     2     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
     3     aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_4X4_100)
     4     parameters = aruco.DetectorParameters()
     5     corners, ids, _ = aruco.detectMarkers(
     6         gray, aruco_dict, parameters=parameters)
     7
     8     int_corners = np.int0(corners)
     9     perimeter = cv2.arcLength(corners[0], True)
    10
    11     return corners, ids, perimeter
```

4. In this part we have loaded our image and taken three parameters to detect markers which are corners, ids, perimeter.

```
[4]  1 # Load the image
     2 image = cv2.imread("/content/Garvit.jpg")

     1 # Detect ArUco markers
     2 corners, ids, perimeter = detect_markers(image)
```

5. Conversion Factor Calculation:

conversion = 25 / perimeter: Calculates a conversion factor based on a known distance (25, for example) and the detected marker perimeter.

Check for Two Markers:

if len(corners) == 2:: Checks if exactly two markers are detected.

Calculate Mean Positions:

head_marker = np.mean(corners[0][0], axis=0).astype(int): Calculates the mean position of the corner coordinates of the head marker.

foot_marker = np.mean(corners[1][0], axis=0).astype(int): Calculates the mean position of the corner coordinates of the foot marker.

Draw Line on Image:

cv2.line(image, tuple(head_marker), tuple(foot_marker), (0, 255, 0), 2): Draws a line connecting the head and foot markers on the image.

Calculate Real-World Distance:

height = calculate_distance(head_marker, foot_marker) * conversion: Calculates the real-world distance (height) between the head and foot markers using the conversion factor.

Format and Display Results:

height_text = f"Estimated Height: {height:.2f} cm": Formats the height value as a string.
print("Estimated Height:", height) and print(height_text): Prints the estimated height to the console.
cv2.putText(...): Adds the estimated height text to the image.

Handle Case of Fewer Than Two Markers:

else:: Executes if the number of detected markers is not exactly two. Prints a message indicating that two markers (head and foot) were not detected.

```python
# Assuming you have detected two markers (head and foot) and want to calculate the distance between them
conversion = 25 / perimeter
if len(corners) == 2:
    head_marker = np.mean(corners[0][0], axis=0).astype(int)
    foot_marker = np.mean(corners[1][0], axis=0).astype(int)

    cv2.line(image, tuple(head_marker), tuple(foot_marker), (0, 255, 0), 2)

    # Calculate distance between head and foot markers
    height = calculate_distance(head_marker, foot_marker) * conversion

    # Format the distance value
    height_text = f"Estimated Height: {height:.2f} cm"

    print("Estimated Height:", height)
    print(height_text)

    # Display the estimated height on the image
    cv2.putText(image, height_text, (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

else:
    print("Could not detect two markers (head and foot).")
```

```
Estimated Height: 172.10776221081574
Estimated Height: 172.11 cm
```

6. Draw Detected Markers:

aruco.drawDetectedMarkers(image, corners, ids): Draws markers on the image based on the detected corners and IDs. This is part of the ArUco library and helps visualize which parts of the image were recognized as markers.

Display Image with cv2_imshow:

cv2_imshow(image): Displays the image with detected markers. Note: cv2_imshow is specific to certain environments like Colab. If you are working in a local environment, you might want to use cv2.imshow instead.

Save the Image:

cv2.imwrite("final_image_with_height.jpg", image): Writes the image with detected markers to a file named "final_image_with_height.jpg." This allows you to save the visualized result for future reference.

```python
# Display the image with detected markers and lines
aruco.drawDetectedMarkers(image, corners, ids)
cv2_imshow(image)
cv2.imwrite("final_image_with_height.jpg", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Result:

In the provided image, ArUco markers have been strategically placed on the walls. To mitigate potential errors, the ArUco marker was positioned slightly higher to account for the remaining height at the bottom. The actual height in real life is approximately **173 cm**, and the computed result is approximately **172.11 cm**. This discrepancy is attributed to the difference in height at the bottom.

Additionally, for more accurate measurements, advanced camera calibration techniques could be implemented. These techniques encompass factors such as **scaling, camera location,** and other parameters to provide a more precise estimation of the real-life distance corresponding to the height observed in the image.

In this image, Aditya’s height is coming to be 174.90 cm, which is approximately the same as Aditya’s actual height is 175cm.

In this image Udit’s height is a bit shorter, so we have to lower the upper aruco marker to get the height. The height is around 165.52cm which is approximately the same as the actual height which is 166cm.

PART-2: SCALING AND CAMERA CALIBRATION

1.

```python
from google.colab.patches import cv2_imshow
import cv2
import cv2.aruco as aruco
import numpy as np

def calculate_distance(point1, point2):
    x1, y1 = point1[0], point1[1]
    x2, y2 = point2[0], point2[1]
    distance = np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
    return distance

def detect_markers(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_4X4_100)
    parameters = aruco.DetectorParameters()
    corners, ids, _ = aruco.detectMarkers(
        gray, aruco_dict, parameters=parameters)

    int_corners = np.int0(corners)
    perimeter = cv2.arcLength(corners[0], True)

    return corners, ids, perimeter
```

1. Import Statements:
`cv2_imshow`: This is a function used for displaying images in Google Colab. It's specific to the Colab environment.
`cv2`: OpenCV library for computer vision tasks.
`cv2.aruco`: Part of OpenCV containing functions for ArUco markers, a type of augmented reality marker.
`numpy as np`: NumPy is a library for numerical operations in Python. It is commonly aliased as `np` for brevity.

2. calculate_distance Function:

- This function takes two points, `point1` and `point2`, as input.
- It extracts the x and y coordinates from each point.
- Calculates the Euclidean distance between the two points using the distance formula (`np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)`).
- Returns the calculated distance.

3. `detect_markers` Function:

- This function takes an image as input.
- Converts the input image to grayscale using `cv2.cvtColor`.
- Initializes an ArUco marker dictionary (`aruco_dict`) with a specific predefined set of markers (`aruco.DICT_4X4_100`).
- Creates a set of parameters for marker detection using `aruco.DetectorParameters()`.
- Detects markers in the grayscale image using `aruco.detectMarkers`.
- Converts the corners of the detected markers to integer values using `np.int0`.
- Computes the perimeter of the first detected marker using `cv2.arcLength`.
- Returns the detected corners, marker IDs, and the perimeter of the first marker.

2.

```
def calculate_height_and_distance(image_path, focal_length, sensor_height, head_marker_height):
    image = cv2.imread(image_path)

    corners, ids, perimeter = detect_markers(image)

    if len(corners) == 2:
        head_marker = np.mean(corners[0][0], axis=0).astype(int)
        foot_marker = np.mean(corners[1][0], axis=0).astype(int)

        pixel_height = calculate_distance(head_marker, foot_marker)

        distance = (focal_length * head_marker_height) / (pixel_height * sensor_height)

        height = (head_marker_height * 100) / (pixel_height / perimeter)

        distance_text = f"Estimated Distance: {distance:.2f} units"
        height_text = f"Estimated Height: {height:.2f} cm"

        print("Estimated Distance in pixels :", distance)
        print("Estimated Height Using ArUco markers :", height)

        aruco.drawDetectedMarkers(image, corners, ids)
        cv2_imshow(image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    else:
        print("Could not detect two markers (head and foot).")
```

1. Read Image and Detect Markers:

- Reads an image from the specified file path using `cv2.imread`.
- Calls the `detect_markers` function to identify ArUco markers in the image, obtaining their corners, IDs, and perimeter.

2. Extract Marker Information:

- Checks if exactly two markers (head and foot) were detected.
- Calculates the center coordinates of the head and foot markers using `np.mean` along the axis 0 (mean of x and y coordinates). The result is converted to integer values.

3. Calculate Distance:

- Calls the `calculate_distance` function to find the pixel height between the head and foot markers.
- Calculates the real-world distance using the formula: `distance = (focal_length * head_marker_height) / (pixel_height * sensor_height)`.

4. Calculate Height:

- Computes the real-world height using the formula: `height = (head_marker_height * 100) / (pixel_height / perimeter)`.
- Prints the estimated distance in pixels and the estimated height in centimeters.
- Constructs text strings for distance and height.

5. Visualize Results:

- Draws the detected markers on the image using `aruco.drawDetectedMarkers`.
- Displays the image with detected markers using `cv2_imshow`.
- Waits for a key event to close the image window.

6. Handle Detection Failure:
If the number of detected markers is not equal to 2, it prints a message indicating that it couldn't detect both head and foot markers.

Overall, this function takes an image, detects ArUco markers, calculates the pixel height, and then estimates the real-world distance and height based on provided parameters. It visualizes the detected markers on the image and displays the result.

3.

```
image_path = "/content/Garvit.jpg"
focal_length = 10  # Focal length of the camera (in pixels)
sensor_height = 10  # Sensor height of the camera (in millimeters)
head_marker_height = 12  # Height of the head marker in real-world units
calculate_height_and_distance(image_path, focal_length, sensor_height, head_marker_height)
```

1. `image_path`:
Specifies the file path of the image to be processed. In this case, it is set to "/content/Garvit.jpg".

2. `focal_length`:
Represents the focal length of the camera in pixels. The focal length is a critical parameter in camera calibration and is used in the calculation of real-world distances based on pixel measurements.

3. `sensor_height`:
Denotes the sensor height of the camera in millimeters. The sensor height is another parameter used in camera calibration to convert pixel measurements to real-world units.

4. `head_marker_height`:
Specifies the height of the head marker in real-world units (e.g., centimeters or inches). This value is used in conjunction with the pixel measurements to estimate the real-world height.

5. `calculate_height_and_distance(image_path, focal_length, sensor_height, head_marker_height)`:

- Calls the `calculate_height_and_distance` function, passing the provided image path, focal length, sensor height, and head marker height as arguments.
- The function processes the image, detects ArUco markers, and calculates the estimated distance and height based on the given parameters.
- The results are printed, and the image with detected markers is displayed.

This code snippet essentially sets up the parameters required for the height and distance calculation and then calls the function to perform the calculations on the specified image. The provided parameters (focal length, sensor height, and head marker height) are crucial for accurately converting pixel measurements to real-world distances and heights.

RESULT :

```
Estimated Distance in pixels : 0.004324905931822408
Estimated Height Using ArUco markers : 174.30939554749912
```

This is giving an estimated height and estimated distance between any two points in pixels. And the resulting image is giving an estimation that how distance is calculated using aruco markers.


STREET WEAR
Angeles

MATHEMATICAL FORMULATION (AN ALTERNATIVE APPROACH):

sf be the scaling factor.
$\mathbf{H_{real}}$ be the real-world height of the person.
$\mathbf{H_{image}}$ is the height of the person in the image.
$\mathbf{D_{real}}$ is the distance from the camera to the ArUco marker (known value).
$\mathbf{D_{image}}$ is the distance from the camera to the ArUco marker in the image.
f be the focal length of the camera.

The scaling factor (sf) can be calculated as:
$\mathbf{sf = H_{real} / H_{image}}$

Now, let's find a relation $\mathbf{H_{image}}$ and $\mathbf{D_{image}}$ between using similar triangles. The relation is given by:

$$\mathbf{H_{image} / D_{image} = H_{real} / D_{real}}$$

Solving for $\mathbf{H_{image}}$ we get:

$$\mathbf{H_{image} = (f \times H_{real}) / D_{real}}$$

Now, substitute $\mathbf{H_{image}}$ in the scaling factor formula:

$$\mathbf{sf = H_{real} / ((f \times H_{real}) / D_{real})}$$

Simplify to find **sf**:

$$\mathbf{sf = D_{real} / f}$$

So, the scaling factor (sf) is inversely proportional to the focal length (f) and directly proportional to the known distance ($\mathbf{D_{real}}$).

Finally, to find the real-world height of the person **($\mathbf{H_{real}}$)**, you can use the formula:

$$\mathbf{H_{real} = H_{image} \times sf}$$

This formula allows you to calculate the real-world height of the person based on the image height and the scaling factor obtained from ArUco marker pose estimation.
