543653
Juan C. Chávez Villarreal

Homework #8: 3D measurements of flat surface objects

Visión Computacional 14:30 hrs

Universidad de Monterrey

1 Planteamiento del Problema

Objetivos:

1. Implementar el modelo matemático que mapee píxeles seleccionados de una imagen 2D en puntos del mundo 3D, medidos con respecto a un marco de coordenadas de un sensor de visión.
2. Utilizar esos puntos del mundo 3D para llevar a cabo mediciones 3D de los objetos de superficie plana seleccionados.

2 Materiales

- Computadora personal
- Raton de computadora.
- Camara monocular.
- Github
- Código de Python proporcionado por el profesor para calibración de camara.
- Parametros de calibración de camara webcam.
- Python 3.10 o superior junto con las siguientes librerias: OpenCV, NumPy, ArgParse, OpenCV.
- Visual Studio Code

3 Metodología

El código de 'get-measurements.py' utiliza visión computacional para obtener mediciones 3D precisas de objetos en imágenes capturadas por una cámara monocular. Al principio, la función 'initialize_camera' configura la cámara y carga los parámetros de calibración necesarios para las mediciones precisas. Durante la ejecución, los usuarios interactúan con la imagen a través de eventos de mouse gestionados por 'mouse_callback', seleccionando puntos o áreas de interés. La función 'compute_line_segments' calcula los segmentos de línea entre los puntos seleccionados, lo que es fundamental para determinar las dimensiones de los objetos en la imagen. 'compute_perimeter' toma estos segmentos de línea y calcula el perímetro de la forma delineada, lo que puede ser útil para estimaciones de área o para más análisis geométricos. Finalmente, la función 'pipeline' actúa como el núcleo del proceso, invocando las funciones anteriores en secuencia y manejando el flujo de datos entre ellas para llevar a cabo el proceso completo de medición, desde la captura de imagen hasta la presentación de resultados. A continuación veremos version colapsada del codigo y posteriormente veremos a mayor detalle que hace cada función.

get-measurements.py

```python
import numpy as np
import cv2
import glob
import os
import argparse
import sys
import textwrap
import json
import platform
from numpy.typing import NDArray
from typing import List, Tuple

points = [] #Tupla que almacena los puntos del dibujo
drawing = False #True si ya se complet  el dibujo

def user_arguments():
def load_calibration_parameters_from_json_file():
def initialize_camera(args):
def mouse_callback(event,x,y,flags,param):
def undistort_images():
def compute_line_segments(points: List[Tuple[int,int]]):
def compute_perimeter():
def pipeline():

if __name__ == '__main__':
    pipeline()
```

3.1 user_arguments

La función 'user_arguments' está diseñada para analizar y organizar los argumentos suministrados por el usuario al momento de ejecutar el script, haciendo uso de la biblioteca 'argparse' de Python. En la función definimos tres argumentos clave que los usuarios deben proporcionar: el índice de la cámara ('–camera_index' o '-c'), que determina cuál cámara se usará para las mediciones y requiere un número entero; la distancia entre la cámara y el objeto ('–z'), un dato crítico para las mediciones que espera un valor flotante; y la ruta al archivo JSON que contiene los parámetros de calibración de la cámara ('–input_calibration_parameters' o '-j'), esencial para ajustar la precisión de las mediciones. Todos estos argumentos son obligatorios y específicos en su tipo, asegurando que el usuario proporcione toda la información necesaria para que el proceso de medición se ejecute correctamente. Finalmente, la función analiza estos argumentos y devuelve un objeto que los contiene, facilitando el acceso a estos datos a lo largo del script y permitiendo que el proceso de medición se ajuste según las configuraciones definidas por el usuario.

get-measurements.py

```
def user_arguments() -> argparse.ArgumentParser:

    parser = argparse.ArgumentParser(prog='HW8 - 3D camera measurement',
                                     description='Calculate dimensions of
-----------------------------------user provided geometries.',
                                     epilog='JCCV - 2024')
    parser.add_argument('--camera_index',
                        '-c',
                        type=int,
                        required=True,
                        help="Index for desired camera")
    parser.add_argument('--z',
                        type=float,
                        required=True,
                        help="Distance between camera and object")
    parser.add_argument('--input_calibration_parameters',
                        '-j',
                        type=str,
                        required=True,
                        help='JSON file with calibration parameters')
    args = parser.parse_args()
    return args
```

3.2 load_calibration_parameters_from_json_file

La función load_calibration_parameters_from_json_file se encarga de cargar los parámetros de calibración de una cámara desde un archivo JSON, utilizando los argumentos proporcionados por el usuario a través de la línea de comandos. La entrada de esta función es un objeto que contiene los argumentos analizados por el script, específicamente interesado en la ruta al archivo JSON donde se almacenan los parámetros de calibración.

get-measurements.py

```
def load_calibration_parameters_from_json_file(
        args:argparse.ArgumentParser):

    # Check if JSON file exists
    json_filename = args.input_calibration_parameters
    check_file = os.path.isfile(json_filename)

    # If JSON file exists, load the calibration parameters
    if check_file:
        f = open(json_filename)
        json_data = json.load(f)
        f.close()

        camera_matrix = np.array(json_data['camera_matrix'])
        distortion_coefficients = np.array(json_data['distortion_coefficients'])
        return camera_matrix, distortion_coefficients

    # Otherwise, the program finishes
    else:
        print(f"The file {json_filename} does not exist!")
        sys.exit(-1)
```

3.3 initialize_camera(args)

La función initialize_camera tiene como propósito inicializar la cámara basándose en los argumentos recibidos desde la línea de comandos. Recibe como entrada un objeto que contiene los argumentos analizados por el script, de donde extrae el índice de la cámara especificado por el usuario.

get-measurements.py

```
def initialize_camera(args):
    cap = cv2.VideoCapture(args.camera_index)
    if not cap.isOpened():
        print("Error al inicializar la c mara")
        return None
    return capv
```

3.4 mouse_callback()

La función mouse_callback es una función de callback diseñada para manejar eventos de mouse en una ventana de imagen. Estos eventos incluyen clics de botón derecho, izquierdo y medio, así como la interacción con teclas específicas en combinación con clics del mouse. La función opera con dos variables globales, points y drawing, que respectivamente almacenan puntos seleccionados por el usuario y un indicador de estado que señala si se ha completado la selección de puntos.

Cuando el usuario hace clic con el botón derecho del mouse (EVENT_RBUTTONDOWN), la función verifica si se presionó simultáneamente la tecla Alt (EVENT_FLAG_ALTKEY). Si es así y ya existen puntos en la lista points, esta se limpia completamente, eliminando todos los puntos almacenados. Si no se presiona la tecla Alt pero hay puntos en la lista, el último punto añadido se elimina, permitiendo al usuario corregir errores de selección.

Al hacer clic con el botón izquierdo del mouse (EVENT_LBUTTONDOWN), la función añade la posición actual del cursor (dada por las coordenadas x e y) a la lista points, lo que permite al usuario marcar puntos de interés sobre la imagen.

Un clic con el botón medio del mouse (EVENT_MBUTTONDOWN) añade el primer punto de la lista points al final de la misma y cambia el estado de la variable drawing a True, lo cual puede utilizarse para indicar que la selección de puntos está completa y posiblemente iniciar un proceso de dibujo o medición basado en los puntos seleccionados.

get-measurements.py

```
def mouse_callback(event,x,y,flags,param):
    global points
    global drawing

    if event == cv2.EVENT_RBUTTONDOWN:
        if flags & cv2.EVENT_FLAG_ALTKEY:
            if points:
                points.clear()
        else:
            if points:
                points.pop()
    elif event == cv2.EVENT_LBUTTONDOWN:
        points.append((x,y))
    elif event == cv2.EVENT_MBUTTONDOWN:
        points.append(points[0])
        drawing = True
```

3.5 undistort_images

La función undistort_images se utiliza para corregir las distorsiones en las imágenes capturadas por una cámara, utilizando los parámetros de calibración de la misma. Estos parámetros incluyen la matriz de la cámara (mtx)

y los coeficientes de distorsión (dist). La corrección de la distorsión es un paso crucial para mejorar la precisión de las mediciones y análisis realizados con las imágenes.

El proceso comienza con la toma de un fotograma (frame) distorsionado como entrada. Primero, se determina el tamaño de este fotograma extrayendo las dimensiones de altura y anchura de la imagen.

Utilizando la matriz de la cámara y los coeficientes de distorsión, la función calcula una "nueva matriz de cámara óptima" con la función cv2.getOptimalNewCameraMatrix. Esta nueva matriz está optimizada para reducir la distorsión al mínimo posible sin perder partes significativas de la imagen. Durante este cálculo, se puede especificar el área de interés (ROI, por sus siglas en inglés) de la imagen que se desea mantener después de la corrección.

A continuación, la imagen se desdistorsiona utilizando la función cv2.undistort, que aplica tanto la matriz original de la cámara como la nueva matriz de cámara óptima calculada, resultando en una imagen corregida sin las distorsiones originales.

Finalmente, la función opcionalmente recorta la imagen desdistorsionada según el área de interés definida previamente, asegurando que la imagen resultante contenga solo las partes relevantes y esté libre de bordes que podrían haberse distorsionado o quedado vacíos tras la corrección.

get-measurements.py

```
def undistort_images(
        frame,
        mtx:NDArray,
        dist:NDArray,
        )->NDArray:
    # Get size
    h,  w = frame.shape[:2]

    # Get optimal new camera
    newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w, h), 0, (w, h))

    # Undistort image
    dst = cv2.undistort(frame, mtx, dist, None, newcameramtx)

    # Crop image
    x, y, w, h = roi
    dst = dst[y:y+h, x:x+w]
```

3.6 compute_line_segments

La función 'compute_line_segments' está diseñada para calcular las longitudes de los segmentos de línea entre una serie de puntos especificados. Esta función toma como entrada una lista de puntos, donde cada punto es una tupla que contiene coordenadas 'x' e 'y' (enteros), representando posiciones en un espacio bidimensional.

El proceso comienza inicializando una lista vacía llamada 'line_length', donde se almacenarán las longitudes calculadas de los segmentos de línea entre puntos consecutivos en la lista de entrada.

La función luego itera a través de la lista de puntos, comenzando desde el segundo punto hasta el último. Para cada punto en esta iteración (excepto el primero), se toma el punto actual y el punto anterior para formar un par de puntos que definen un segmento de línea. Utilizando las coordenadas de estos dos puntos ('x1', 'y1' para el punto anterior y 'x2', 'y2' para el punto actual), la función calcula la distancia euclidiana entre ellos. Esta distancia se calcula utilizando la fórmula de la raíz cuadrada de la suma de los cuadrados de las diferencias entre las coordenadas 'x' e 'y' de los dos puntos, lo que resulta en la longitud del segmento de línea entre ellos.

Cada longitud de segmento calculada se añade a la lista 'line_length'. Después de iterar a través de todos los puntos y calcular las longitudes de todos los segmentos de línea consecutivos, la función devuelve la lista 'line_length'. Esta lista contiene las longitudes de todos los segmentos de línea formados por pares consecutivos de puntos en la lista de entrada, proporcionando una medida de la distancia entre cada par de puntos.

get-measurements.py

```
def compute_line_segments(points: List[Tuple[int,int]]):
    line_length = [] #matriz donde se guardar n las distancias medidas
    for i in range(1, len(points)):
        x1, y1 = points[i-1] #punto punto anterior
        x2, y2 = points[i] # punto nuevo
        length = np.sqrt((x2 - x1)**2 + (y2 - y1)**2) #distancia entre dos puntos
        line_length.append(length) #agrega dato a matriz
    return line_length
```

3.7 compute_perimeter

La función 'compute_perimeter' está diseñada para calcular el perímetro de una forma geométrica y las distancias entre sus puntos consecutivos, usando parámetros de calibración de cámara para convertir medidas de píxeles a distancias reales en el mundo. Recibe una lista de puntos en 2D, la distancia 'z' desde la cámara al objeto, la matriz de calibración 'mtx', y las dimensiones de la imagen capturada.

Primero, ajusta los valores centrales y los factores de escala de la matriz de calibración a las dimensiones actuales de la imagen. Luego, para cada par de puntos, convierte sus coordenadas de píxeles a coordenadas reales usando 'z' y los factores de escala corregidos. Calcula la distancia euclidiana entre cada par de puntos en el espacio real y suma estas distancias para obtener el perímetro de la forma.

Finalmente, devuelve una lista con las distancias entre puntos consecutivos y el valor total del perímetro, proporcionando una medida precisa del tamaño de la forma en el mundo real basada en los datos de calibración de la cámara.

get-measurements.py

```
def compute_perimeter(points: List[Tuple[int, int]], z: float, mtx: np.ndarray, heigh
    distance = []
    perimeter = 0.0
    Cx = mtx[0,2]*width/1280
    Cy = mtx[1,2]*height/720
    fx = mtx[0,0]*width/1280
    fy = mtx[1,1]*height/720
    for i in range(1, len(points)):
        x1, y1 = points[i-1]
        x2, y2 = points[i]
        # Convertir de p xeles a coordenadas
        X1 = (x1 - Cx) * z / fx
        Y1 = (y1 - Cy) * z / fy
        X2 = (x2 - Cx) * z / fx
        Y2 = (y2 - Cy) * z / fy

        # Calcular distancia entre puntos
        dist = np.sqrt((X2 - X1)**2 + (Y2 - Y1)**2 )
        distance.append(dist)
        perimeter += dist
    return distance, perimeter
```

3.8 pipeline

get-measurements.py

```
def pipeline():
    global drawing
    #datos de usuario
    args = user_arguments()
    #iniciar camara
    cam = initialize_camera(args)
    #Abrir pantalla y activar mouse
    cv2.namedWindow('Live Camera View')
    cv2.setMouseCallback('Live Camera View',mouse_callback)
    #cargar datos de .json
    mtx, dist = load_calibration_parameters_from_json_file(args)

    while True:
        ret, frame = cam.read() #ret es una variablo booleana para saber si hay video
        if not ret:
            print("Error no hay se al")
            break

        h,w = frame.shape[:2]
        if drawing:
            #calcular distancias y perimetros
            distance,perimeter = compute_perimeter(points, args.z,mtx,h,w)

            # distancias a dos puntos decimales en orden de registro
            text = "Distancias (puntos seleccionados):\n"
            for i, dist in enumerate(distance, start=1):
                if i < len(points):
                    text += f"Punto {i}-{i+1}: {dist:.2f} cm\n"
                else:
                    text += f"Punto {i}-{1}: {dist:.2f} cm\n"

            text += f"\nPer metro total: {perimeter:.2f} cm"

            text += "\nMedidas ordenadas de mayor a menor:\n"
            sorted_distance = sorted(distance, reverse=True)
            for i, dist in enumerate(sorted_distance, start=1):
                index = distance.index(dist)
                if index == len(points) - 1:
                    text += f"Punto {len(points)}-{1}: {dist:.2f} cm\n"
                else:
                    text += f"Punto {index+1}-{index+2}: {dist:.2f} cm\n"
            print(text)
            drawing = False  #reiniciar estado del dibujo
        # Dibujar las l neas entre los puntos seleccionados
        for i in range(1, len(points)):
            cv2.line(frame, points[i-1], points[i], (0, 255, 0), 1)
        # Dibujar los puntos seleccionados
        for point in points:
            cv2.circle(frame, point, 3, (0, 255, 0), -1)
        cv2.imshow('Live Camera View', frame)
        k = cv2.waitKey(1) & 0xFF
        if k == 27:
            break
    cam.release()
    cv2.destroyAllWindows()
```

La función 'pipeline' es el núcleo de un script diseñado para realizar mediciones en imágenes capturadas por una cámara, utilizando interacción del usuario a través del mouse y parámetros de calibración de la cámara cargados de un archivo JSON.

Comienza recolectando los argumentos proporcionados por el usuario mediante la función 'user_arguments', que incluyen el índice de la cámara, la distancia entre la cámara y el objeto a medir, y la ruta al archivo JSON con los parámetros de calibración.

Luego, inicializa la cámara con 'initialize_camera' y abre una ventana titulada 'Live Camera View' para mostrar la vista en vivo de la cámara. Se establece un callback para eventos de mouse en esta ventana, permitiendo al usuario seleccionar puntos sobre la imagen capturada en tiempo real.

Los parámetros de calibración de la cámara se cargan desde el archivo JSON especificado utilizando la función 'load_calibration_parameters_from_json_file', lo que permite corregir distorsiones en las imágenes capturadas y realizar mediciones precisas.

Dentro de un bucle infinito, la función lee continuamente fotogramas de la cámara y verifica si la captura es exitosa. Si no hay señal de video, el bucle se interrumpe con un mensaje de error.

Si la variable 'drawing' está activada (indicando que el usuario ha completado una selección de puntos), la función calcula las distancias entre los puntos seleccionados y el perímetro de la forma formada por ellos con 'compute_perimeter', ajustando las mediciones a escala real utilizando la distancia 'z' y los parámetros de calibración. Luego, se generan y muestran mensajes que detallan las distancias entre los puntos y el perímetro total, tanto en el orden de selección como ordenadas de mayor a menor.

A continuación, dibuja líneas entre los puntos seleccionados y marca cada punto con un círculo en el fotograma capturado, mostrando visualmente la forma que el usuario ha definido.

El bucle continúa hasta que el usuario presiona la tecla ESC ('27'), momento en el cual se liberan los recursos de la cámara y se cierran todas las ventanas de OpenCV. Este flujo permite al usuario interactuar dinámicamente con la imagen capturada, seleccionando áreas de interés y obteniendo mediciones precisas de estas, todo visualizado en tiempo real.

4 Conclucion

En conclusión, el trabajo presentado en "Homework 8: 3D measurements of flat surface objects" demuestra un enfoque innovador y práctico para la obtención de mediciones 3D precisas a través de la visión computacional, utilizando una cámara monocular. El desarrollo del script 'get-measurements.py', acompañado de una detallada metodología, ilustra cómo se pueden implementar técnicas de calibración de cámaras y procesamiento de imágenes para convertir píxeles seleccionados en mediciones del mundo real.

Este proyecto resalta la importancia de la calibración precisa de la cámara y la interacción del usuario para seleccionar puntos de interés, permitiendo mediciones detalladas y precisas en superficies planas. Las funciones desarrolladas, desde 'user$_arguments'hasta'pipeline', trabajanenconjuntoparafacilitaresteproceso, demostrandolacapacia

La capacidad de traducir puntos de una imagen 2D a coordenadas 3D y realizar mediciones precisas tiene aplicaciones significativas en muchos campos, incluyendo la robótica, la arquitectura y la ingeniería. Este trabajo no solo proporciona una herramienta útil para tales mediciones sino que también ofrece una base sobre la cual se pueden construir proyectos más complejos, abriendo la puerta a futuras investigaciones y desarrollos en el campo de la visión por computadora.

Finalmente, este reporte, junto con el código y las explicaciones proporcionadas, sirve como un recurso educativo valioso para aquellos interesados en explorar la intersección entre la visión computacional y las mediciones físicas, promoviendo la innovación y la aplicación práctica de conceptos teóricos en el mundo real.