Campus Irapuato-Salamanca | División de Ingenierías

XXIX Verano de la Ciencia Universidad de Guanajuato

Determinación de presencia y ubicación de obstáculos mediante análisis estadístico de imágenes secuenciales de profundidad.

Verano de la Ciencia 2024.

Autores

Aguilar Pérez María Luisa

Alcacio Rodríguez Ariel Imanol

Mosqueda Gutiérrez María Fernanda

Ramírez García Juan Alejandro

Encargado Del Proyecto

Dr. José Luis Contreras Hernández

Descripción general

En este proyecto se desarrollará un programa para determinar la presencia y ubicación de obstáculos mediante el análisis de valores estadísticos, como la media y la varianza de pixeles, en imágenes secuenciales de profundidad. Se utilizará el video adquirido previamente con una cámara de profundidad Intel en ambientes de interior para extraer la secuencia de imágenes presentando obstáculos típicos. El programa desarrollado indicará la presencia y ubicación de obstáculos mediante una matriz que representa el lado derecho, izquierdo, arriba, abajo y las esquinas de la imagen. Además, se utilizará la secuencia de imágenes para la eliminación de posible ruido y errores en la adquisición con la cámara de profundidad.

Justificación del equipo

El desarrollo de un algoritmo que determine la presencia y ubicación de obstáculos mediante análisis estadístico de imágenes secuenciales de profundidad en ambientes interiores aportará para la disminución en el ruido y mejorar la identificación de obstáculos para el desarrollo futuro de proyectos enfocados en la adaptación en distintos medios de personas con problemas de visión.

Objetivos

Crear un manual de uso del software desarrollado y que explique su algoritmo de procesamiento con el fin de facilitar su uso en proyectos de investigación futuros.

Resultados esperados

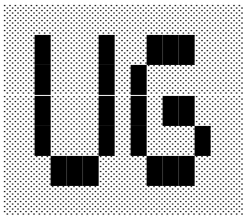
Redacción de un manual tutorial con la descripción del uso del software y del algoritmo de procesamiento realizado.

Requisitos

- Matlab R2020a o superior.
- Base de datos con imágenes a color y profundidad.
- Migrar la base de datos y el archivo deteccion.m a la carpeta raíz de MATLAB.
- Iniciar Matlab R2020a o superior.
- Abrir el archivo deteccion.m

Código

Encabezado y Descripción

```
%  
%
%
%
%
%
%
%
% DETERMINACIÓN DE PROXIMIDAD MEDIANTE EL ANÁLISIS ESTADÍSTICO DE
% IMÁGENES DE PROFUNDIDAD.
%
% En este script se realiza la detección de imágenes en base al análisis
% por frames dando así una mayor predicción para la detección de objetos.
% Haciendo uso de frames anteriores para promediar y dar un resultado
% de detección de objetos, esto con la finalidad de mitigar el error por
% iluminación y reflejos.
%
clc; close all;
```

Verano de la ciencia - Universidad de Guanajuato

| Campus irapauto - salamanca
| DICIS
| Verano 2024

Este encabezado proporciona contexto sobre el proyecto y la finalidad del script. Es un proyecto realizado durante el "Verano de la Ciencia" en la Universidad de Guanajuato. El objetivo es detectar la proximidad mediante el análisis estadístico de imágenes de profundidad.

Ajustes para la Detección

```
%% Ajustes para la detección. ----------------------------------------

% Modificar en base al nombre de la carpeta donde se encuentren las
pruebas
carpetaData = 'output6';

% Modificar en base a la cantidad total de imágenes que contiene la
prueba
nImagenes = 1428;

% Modificar la cantidad de FPS que se analizan en la prueba
fps = 30;

% Ajuste de umbral de luz a captar en las pruebas
```

```
umbral = 15000;
% ------------------------------------------------------------------------
```

Estas son las variables que se pueden cambiar según las necesidades del usuario:

- carpetaData: Nombre de la carpeta que contiene las imágenes.
- nImagenes: Número total de imágenes en el conjunto de pruebas.
- fps: Cantidad de fotogramas por segundo a analizar.
- umbral: Umbral de luz para la detección.

Código de Procesamiento

```
%% Codigo de procesamiento
frames_cache = [];
for k=0:1:nImagenes
    % Foto en el presente
    ori = imread([carpetaData '/color/' 'prueba_' num2str(k)
'_color.png']);
    img = imread([carpetaData '/depth/' 'prueba_' num2str(k)
'_color.png']); % Foto de profundidad

    img2 = 65535 - img; % Inversión de la imagen de profundidad

    % Selección del umbral
    u = img2 >= 63500; % Selecciona los valores que pasan el umbral
    conv = uint16(u);  % Convierte la matriz de umbral al tipo de matriz
original
    img3 = img2 .* conv; % Filtra los datos que no pasan el umbral

    % Segmentación de la imagen
    mat1 = img3(1:160, 1:213);
    mat2 = img3(1:160, 213:426);
    mat3 = img3(1:160, 426:640);
    mat4 = img3(160:320, 1:213);
    mat5 = img3(160:320, 213:426);
    mat6 = img3(160:320, 426:640);
    mat7 = img3(320:480, 1:213);
    mat8 = img3(320:480, 213:426);
    mat9 = img3(320:480, 426:640);

    % Cálculo del promedio de los segmentos
    promedios = [mean(mat1, 'all'), mean(mat2, 'all'), mean(mat3, 'all');
                 mean(mat4, 'all'), mean(mat5, 'all'), mean(mat6, 'all');
                 mean(mat7, 'all'), mean(mat8, 'all'), mean(mat9,
'all')];

    % Agregar el promedio actual a frames_cache
    frames_cache = cat(3, frames_cache, promedios);

    % Verificar si frames_cache tiene más de 30 frames
    if size(frames_cache, 3) > fps
        % Eliminar el frame más antiguo
        frames_cache = frames_cache(:, :, 2:end);
    end
```

```
    % Sumar todas las matrices en frames_cache a lo largo de la tercera
dimensión
    total_sum = sum(frames_cache, 3);

    % Cantidad de frames en frames_cache
    nFrames = size(frames_cache, 3);

    % Calcular el promedio dividiendo por la cantidad de frames
    promedios_corr = total_sum / nFrames;

    % Detección del objeto mediante el umbral
    PU = promedios_corr >= umbral;

    % Dimensiones de la imagen original
    [height, width, ~] = size(img);
    pixel_height = height / size(PU, 1);
    pixel_width = width / size(PU, 2);

    % Mostrar la imagen original con los objetos detectados
    imshow(ori);
    hold on;
    for row = 1:size(PU, 1)
        for col = 1:size(PU, 2)
            if PU(row, col) == 1
                rectangle('Position', [(col-1)*pixel_width, (row-
1)*pixel_height, pixel_width, pixel_height], 'EdgeColor', 'g',
'LineWidth', 2);
            end
        end
    end
    pause(0.0001);
end
```

Explicación del Código de Procesamiento

1. **Inicialización**:
   - `frames_cache` se inicializa como una matriz vacía para almacenar los promedios de los frames.
2. **Bucle principal**:
   - El bucle itera sobre cada imagen (`nImagenes`).
   - `ori` lee la imagen a color actual.
   - `img` lee la imagen de profundidad actual.
3. **Procesamiento de la imagen de profundidad**:
   - `img2` invierte los valores de la imagen de profundidad.
   - `u` crea una máscara binaria basada en el umbral.
   - `conv` convierte la máscara binaria a una matriz del mismo tipo que la imagen original.
   - `img3` aplica la máscara a la imagen invertida.
4. **Segmentación y Promedio**:
   - La imagen procesada se divide en 9 segmentos.
   - Se calcula el promedio de cada segmento y se almacena en `promedios`.

5. **Cache de Frames**:
    - `promedios` se agrega a `frames_cache`.
    - Si `frames_cache` contiene más de `fps` frames, elimina el frame más antiguo.
    - Se suma todas las matrices en `frames_cache` y se calcula el promedio.
6. **Detección del Objeto**:
    - `PU` determina las ubicaciones donde el promedio corregido excede el umbral.
    - Se obtienen las dimensiones de la imagen original y se calculan las dimensiones de los píxeles.
    - Se muestra la imagen original con rectángulos verdes alrededor de los objetos detectados.

Ajustes

Instrucciones de Ajuste del Código en detección.m

Dentro del código del archivo deteccion.m, encontraremos la sección de **Ajustes**, la cual contiene variables que podrán ser modificadas dependiendo de la prueba en cuestión. Algunas pruebas pueden variar en intensidad de luminosidad o cantidad de fotogramas.
Además, la luz captada en cada toma puede variar debido al entorno en el cual se realizó, lo que puede afectar el rendimiento en la detección de objetos cercanos. Por esta razón, será necesario ajustar el valor del umbral y la cantidad de fotogramas por segundo (FPS) para la prueba correspondiente.

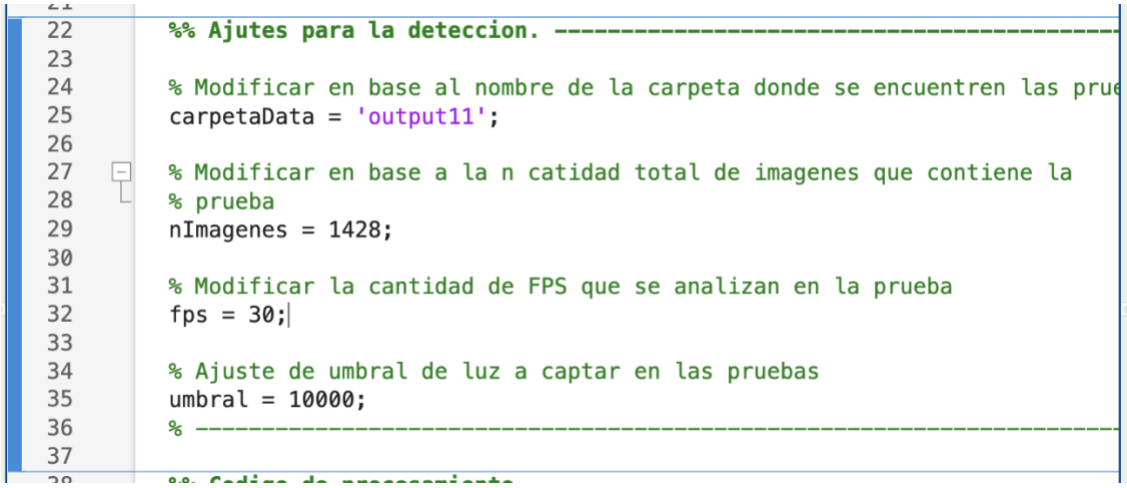


Figura 1. Sección de ajustes dentro del código

1. **Selección de prueba**
   Modificar la variable “carpetaData” con el nombre de la prueba que se desea testear

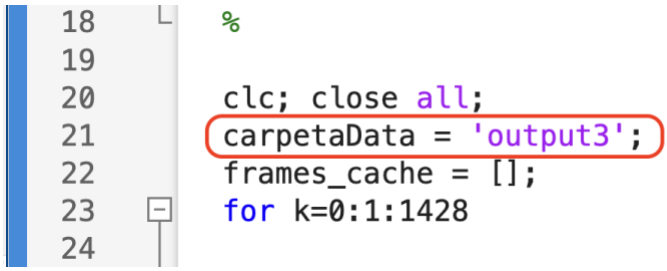


Figura 2. Línea de código en donde se encuentra la variable carpetaData

Advertencia
El nombre que se le asigne a ‘carpetaData’ deberá coincidir con el nombre de la carpeta donde están contenidas las imágenes a color y depth. De lo contrario Matlab resaltará el siguiente error en la ventana de comandos.

Figura 3. Error arrojado por MATLAB al no encontrar la capeta indicada

2. **Número de Imágenes**
   Para el ajuste de la variable “nImagenes” deberemos tomar en cuenta la n cantidad de fotogramas que contenga la prueba, debido a que cada prueba contiene distinta cantidad de fotogramas, en caso de no ajustar y tener una cantidad mayor de fotogramas a la que contiene la prueba podremos presentar un error relacionado a la cantidad de n iteraciones que realiza el programa o en caso de ser menor la cantidad de fotogramas establecidos a los que contiene la prueba, esta se podrá ver afectada limitando su detección en posibles frames restantes.

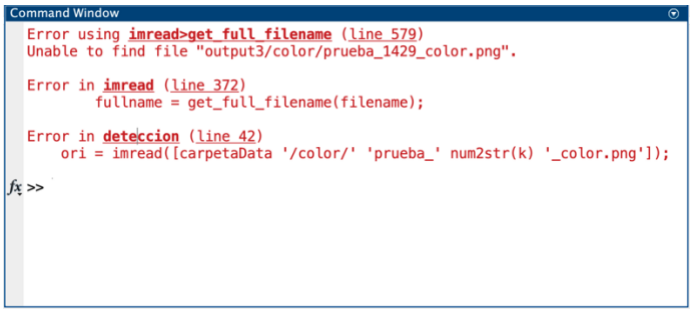


Figura 4. Error arrojado por MATLAB al no detectar imágenes/frames existentes

3. **Frames por segundo (FPS)**
   El ajuste de la variable "FPS" dependerá de la cantidad de frames por segundo que se desee acumular para llevar el análisis entre la cantidad total de frames y descartar posibles detecciones fantasma por causa de iluminación excesiva.

   La recomendación para el ajuste es de un rango de 30 a 90 FPS debido a la configuración con las que se realizaron la tomas. En caso de estar fuera de este rango podrían pasar los siguientes 2 escenarios.

   - <30 fps: Detección de objetos fantasma a causa de luz intensa o reflejos de luz en la toma.

   - >90 fps: poca detección de objetos cercanos, además de mayor esfuerzo computacional lo cual puede hacer lenta la visualización de datos.

4. **Umbral**
   El ajuste del umbral permite poner un límite al umbral de luz para las tomas evaluadas si el umbral el mayor a la cantidad establecida este permitirá la detección de dichos objetos en alguna de las 9 secciones en la que está dividida la toma.

Una vez concluidos los ajustes se realiza la ejecución del código en Matlab.

Resultados

A partir de la base de datos junto con el código desarrollado se obtuvieron 4 videos en donde se presenta la ejecución del código aplicado a diferentes escenarios con diferentes características de luminosidad, brillo y profundidad.

Figura 5. Detección de objetos en el escenario de la rampa

En la figura 5 se muestra la detección de objetos del escenario de la rampa, en la cual se aprecia el buen desempeño del código, detectando únicamente los espacios donde se encuentran los objetos más próximos, en este caso los barandales ubicados a los costados de la rampa.
Haciendo referencia al video realizado con todos los frames disponibles se observa que el código no detecta las luces del fondo, sin embargo, el cambio de luz que percibe la cámara al dar la vuelta es percibido momentáneamente infiriendo que hay un objeto y mostrando un recuadro aproximadamente por un segundo. Esto se minímizó lo más posible con la implementación del promedio entre frames.

Figura 6. Detección de objetos dentro del aula.

La figura 6 muestra la ejecución del programa aplicado al escenario casi perfecto dentro del salón de clase. Se le describe como “casi perfecto” debido a la buena iluminación y a la baja cantidad de brillos que se presentan. En el respectivo video se aprecia el buen funcionamiento del código detectando casi únicamente los objetos más cercanos, en este caso, la pared y las butacas.

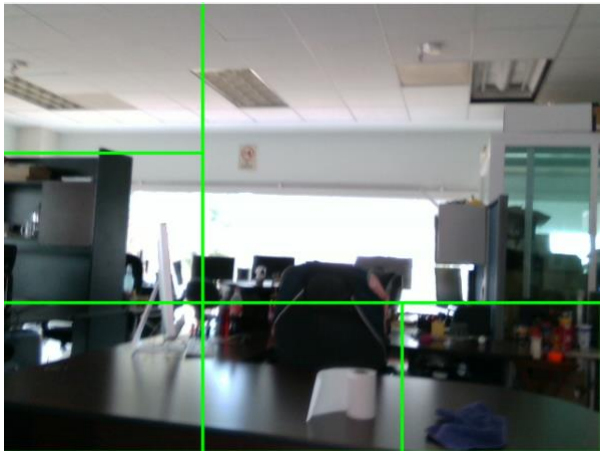


Figura 7. Detección de objetos dentro del laboratorio.

La figura 7 muestra la ejecución del código en un ambiente más hostil para el programa, debido a la gran cantidad de brillos que se presentan y al ser un escenario con una gran cantidad de objetos a diferentes distancias. En su respectivo video se aprecian muchos recuadros donde aparentemente no deberían presentarse, estos debidos a los brillos mencionados anteriormente.

Figura 8. Detección de objetos frente a biblioteca.

La figura 8 muestra el peor caso donde se aplicó la detección de objetos mediante el programa realizado, dicho escenario expuesto, con alta cantidad de luminosidad junto con la mala ejecución del video, el cual enfoca repetitivamente el techo, hace que el programa tenga muchos errores en su ejecución. Esos errores se minimizaron usando los parámetros llamados ajustes para la detección.

Conclusión

El proyecto logró desarrollar un programa capaz de detectar y ubicar obstáculos en secuencias de imágenes de profundidad con una precisión aceptable en condiciones de iluminación favorables. En pruebas realizadas en ambientes como la rampa y el aula, el algoritmo mostró un buen desempeño, destacando su capacidad para ignorar fuentes de luz no deseadas y enfocarse en los obstáculos relevantes. En escenarios con alta luminosidad y numerosos reflejos, como el laboratorio y la biblioteca, el programa enfrentó mayores desafíos, generando detecciones erróneas que se intentaron minimizar mediante ajustes en los parámetros del programa. Los resultados obtenidos en escenarios como el aula y la rampa muestran una alta precisión en la identificación de obstáculos, minimizando las detecciones fantasmas causadas por reflejos y cambios abruptos de luminosidad. Sin embargo, en ambientes con alta cantidad de brillos y numerosos objetos a diferentes distancias, la precisión del programa disminuye, evidenciando la necesidad de ajustar los parámetros de umbral y FPS para mejorar la detección. Los ajustes implementados permitieron reducir los errores en estos escenarios más hostiles, pero se sugiere continuar optimizando el algoritmo para su aplicación en ambientes variados.