PROCESAMIENTO DE IMÁGENES Y ALGORITMOS PARA LA VISIÓN DE UN VEHÍCULO AUTÓNOMO A ESCALA, MEDIANTE UNA CÁMARA ESTEREOSCÓPICA

CHRISTIAN CAMILO TORRES CASTILLO

Proyecto de grado presentado como requisito
para aspirar al título de Ingeniero Electrónico

Director
Enrique Estupiñan Escalante

UNIVERSIDAD ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO.
DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
BOGOTÁ
2023

Índice

1. Introducción

El programa de ingeniería electrónica viene desarrollando el proyecto del vehículo de conducción autónoma VAE, el cual consta de un vehículo (rover), cuya trayectoria y velocidad están controladas mediante una Raspberry pi 3B+; Las trayectorias del rover actualmente son ingresadas mediante código y se llevan al MIL-Model-in-the-Loop, para evaluar los algoritmos de control, mientras que un control de velocidad ya fue implementado en el rover. Con este proyecto se busca generar las trayectorias a seguir por el VAE en un carril exclusivo para el VAE, involucrando trayectorias rectas, curvas y obstáculos; Para cumplir con este propósito se analizarán las imágenes obtenidas por una cámara estéreo, y con el uso de un sensor ToF aportar información del entorno, esto como un complemento a posibles oclusiones de la cámara y determinar si es necesario detener el vehículo.

2. Resumen

Este proyecto presenta el desarrollo de diferentes algoritmos para la visión de un vehículo autónomo a escala, considerando la obtención de profundidad con una cámara estéreo, identificación del carril a seguir por el vehículo y segmentación de imágenes para reconocer objetos en el mapa de profundidad, además se hace el paso paso para la instalación del software OpenCV para Python en una Raspberry Pi 3 B+ de 1GB de RAM, y la recuperación de imágenes de una cámara estéreo Aptina LI-USB30-V024STEREO, las cuales serán material para el desarrollo de los algoritmos anteriormente mencionados.

3. Objetivos

3.1. Objetivo general

Implementar en la plataforma móvil del laboratorio una herramienta de hardware/software que permita generar una trayectoria basada en la detección del carril en un escenario convencional de vehículo autónomo.

3.2. Objetivos específicos

- Generar un método para almacenar y procesar imágenes provenientes de una cámara estéreo USB, con el propósito de usar las imágenes para posicionar un vehículo en el carril.
- Generar una línea a partir de puntos discretos que representen las líneas de carril de una imagen de la cámara USB.
- Determinar en una imagen de carril convencional la información de la profundidad.
- Establecer una trayectoria media en un plano bidimensional en una imagen de un carril.
- Generar un método para asociar la información obtenida de la cámara y el sensor ToF.

4. Metodología de trabajo

Dado que el objetivo de este proyecto es desarrollar algoritmos para el calculo de la profundidad, la segmentación de imágenes para reconocer objetos cercanos, identificar el carril y tomar datos de proximidad con un sensor ToF, se plantea el siguiente diagrama que presenta las diferentes partes del proyecto

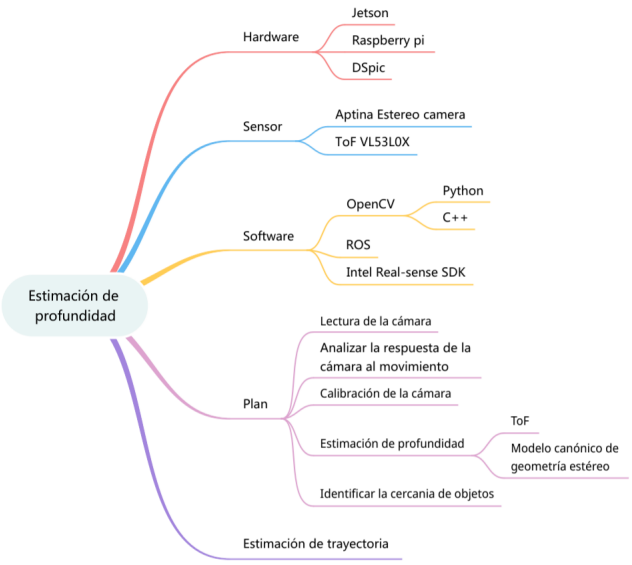


Figura 1: Elementos de la estimación de profundidad

Como se puede observar en la Fig. 1, algunas de las opciones que este presenta no se realizan ya sea por el alcance del proyecto como por ser posibles opciones de desarrollo.

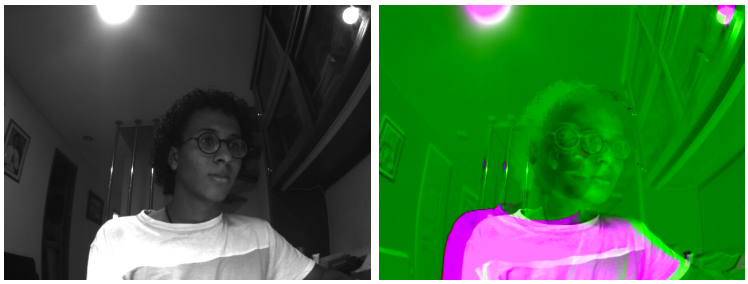
Por comodidad de trabajo y capacidad de cómputo, aunque el proyecto se lleve a cabo sobre una Raspberry Pi, los algoritmos se diseñan, ensayan y configuran, desde un computador con Linux en el que se instaló OpenCV, cuya instalación corresponde de igual manera a la indicada para la Raspberry como se mostrará adelante 5.1. Esto se hace debido a que la configuración de las variables de ciertos algoritmos, demanda demasiada capacidad de computo, por lo que una Raspberry es incapaz de realizar dicha tarea, pero es capaz de correr el algoritmo ya ajustado.

5. Ensayando la Aptina V024STEREO & OpenCV

Para poner a prueba el funcionamiento de la Aptina V024STEREO se descarga el software ofrecido por el fabricante LEOPARD IMAGING INC [11], el cual es exclusivo para Windows, por lo que no funciona en Raspbian (sistema operativo Linux de la raspberry), entonces se prueba emular con wine,(un emulador de programas de Windows en Linux y Mac) en un computador con Manjaro(derivado de Arch Linux) sin éxito, por lo que se averigua en la red sobre diferentes programas para visualización de cámaras USB en Linux, consiguiendo y poniendo a prueba del programa guvcview en Manjaro, el cual funciona sin inconvenientes, que luego se instala en la raspberry mediante el comando

```
sudo apt-get install guvcview
```

Se conecta la cámara, se ejecuta guvcview desde el terminal o la opción ejecutar de la raspberry y se observa lo siguiente



(a) (b)

Figura 2: Imágenes entregadas en bruto por la cámara Aptina. (a) Se observa la imagen solo en blanco y negro, con el ajuste 'mono' en guvcview. (b) Se observa la salida original de la cámara con un contraste de verde y rosado, aparentemente en base a la intensidad de luz recibida.

En la cámara se encuentran dos lentes ajustables, que, después de enfocar lo mejor posible la imagen, se fijan con un trozo de cinta de enmascarar para evitar que se muevan de lugar.

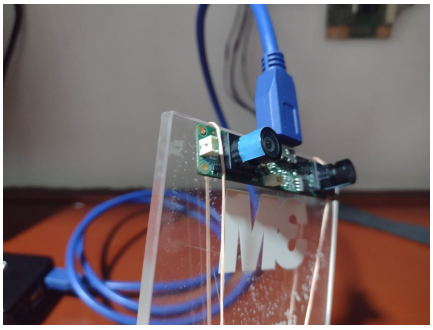


Figura 3: lentes Aptina.

Desde guvcview se puede observar el valor instantáneo de los cuadros por segundo (fps-por sus siglas en ingles), y resulta importante el comportamiento de este valor bajo las siguientes condiciones:

- Al recibir la señal de la cámara sin aplicar filtros en guvcview Fig. 2 (b), bajo las condiciones de movimiento o disminución de la intensidad de luz, el valor de los fps puede caer hasta 26, donde cualquiera de las dos condiciones son igual de significativas.
- Si se aplica el filtro mono Fig. 2 (a), el valor de los fps puede caer hasta 18 bajo las mismas condiciones anteriores, sin embargo, la principal causa y la más significativa es la reducción de la intensidad de luz.

Además de lo anterior, algo de mayor importancia es que durante el uso solo de la cámara mediante guvcview, la raspberry usa alrededor del 70 % de su capacidad de procesamiento.

Ahora que la cámara funciona en la raspberry, es necesario buscar un candidato para el procesamiento de imágenes, tomando por facilidad OpenCV.

5.1. Instalación de OpenCV en la raspberry

Para la instalación de OpenCV en la raspberry se hace revisión de un tutorial de LearnOpenCV [15], acerca de la instalación de OpenCV-4 para C++ y las versiones 2.7 y 3.5 de Python, dirigido a la instalación en Raspbian sobre una raspberry pi modelo 2 o 3.

El proceso de instalación es enteramente desde terminal, y consta con pasos para la instalación de

librerías de Raspbian y Python, como de dependencias necesarias para el funcionamiento de OpenCV. Dentro del tutorial se ofrece descargar un bash script[1] desde GitHub para ejecutar con un solo comando todos los pasos a seguir. La instalación propiamente de `opencv_contrib` depende de dos paquetes, `opencv` y `opencv_contrib`; Al ejecutar el script de instalación, todo corre bien hasta el paso de instalación de las librerías de `opencv` mediande un Makefile, donde la capacidad de la raspberry llega a su límite debido a que este paso presume realizar la compilación del paquete OpenCV directamente en la Raspberry, lo cual excede la capacidad del hardware. Este proceso se intentó en tres ocasiones, alrededor de unas 6 horas por intento, dentro de las cuales unas 3 horas la raspberry estaba totalmente bloqueada y se decide apagar por la fuerza.

En busca de una segunda opción para la instalación de OpenCV, se sigue el procedimiento encontrado en TowardsDataScience [17], el cual consta de un menor número de pasos respecto del tutorial de LearnOpenCV, aunque solo para su uso desde Python, y únicamente el paquete `opencv`.
Igual que en el caso anterior, se instalan librerías y dependencias de OpenCV, de las cuales, las librerías `libqtgui4` y `libqt4-test` presentan el siguiente error al intento de instalarse

```
E: Package libqtgui4 has no installation candidate
E: Package libqt4-test has no installation candidate
```

Investigando el error, se encuentra que para sistemas operativos derivados de Debían, las librerías de QT4 ya no tienen soporte(en estado *oldoldstable*), y fueron reemplazadas por QT5. Dicho esto, y revisando la función de las librerías QT [2] se concluye que el paquete `libqtgui4` actualmente no es una dependencia para el paquete `opencv`, así que se prosigue con la instalación de `opencv` para Python con el comando

```
pip3 install opencv-python
```

En este punto solo hace falta instalar `opencv_contrib`, y se consigue una discusión dentro del foro de programadores TouSu [18], un procedimiento para la instalación de este paquete, con lo que una vez más se realiza la instalación de dependencias y por último el comando para la instalación de interés

```
pip install opencv-contrib-python==4.1.0.25
```

el cual entrega el siguiente error:

```
Defaulting to user installation because normal site-packages is not writeable Looking in
indexes: https://pypi.org/simple, https://www.piwheels.org/simple ERROR: Could not find a
version that satisfies the requirement opencv-contrib-python ==4.1.0.25 (from versions:
3.4.11.45, 3.4.13.47, 3.4.14.51,3.4.15.55, 3.4.16.59, 3.4.17.61, 4.4.0.46, 4.5.1.48,
4.5.2.52, 4.5.3.56, 4.5.4.58, 4.5.4.60, 4.5.5.62) ERROR: No matching distribution
found for opencv-contrib-python==4.1.0.25
```

El mensaje de error indica que la versión que se intenta instalar actualmente no existe, y se listan las versiones disponibles. Del mensaje de error se observa que pip3 busca los paquetes a instalar desde piwheels, que nos permite bajar un paquete precompilado para cada arquitectura a trabajar, reduciendo el proceso antes mencionado de varias horas sin éxito a unos pocos minutos. La instalación de piwheels se realiza con el siguiente comando:

```
pip3 install wheel
```

[1] Un archivo de comandos interpretables por el shell "terminal".
[2] https://packages.debian.org/search?arch=s390x&keywords=libqtgui4

Ahora con la herramienta necesaria se visita la página pythonwheels [12] donde se indica la versión precompilada con éxito de `opencv_contrib` para cada versión de Python y Raspbian correspondientemente

Releases

Version	Released	Stretch (Python 3.5)	Buster (Python 3.7)	Bullseye (Python 3.9)	Files
4.5.5.62	2021-12-29	✗	✗	✗	
4.5.4.60	2021-11-22	✗	✗	✗	
4.5.4.58	2021-10-21	✗	✗	✗	
4.5.3.56	2021-07-11	✗	✓	✓	+
4.5.2.54	2021-06-07	⊖	⊖	⊖	

Figura 4: Tabla de lanzamientos de opencv-contrib-python [12].

con esta información se realiza la instalación con el anterior comando y la versión corregida:

```
pip3 install opencv-contrib-python==4.5.3.56
```

Hecho esto, tenemos por resultado la instalación de OpenCV para Python.

6. Recuperación de imágenes estéreo

La recuperación de imágenes por parte de la librería de OpenCV para la gran mayoría de cámaras genéricas utilizan un índice para especificar a qué lente de la cámara se piden los datos retornados, como se muestra en el siguiente código

```
.
.
.
CamL_id = 2 # Camera ID for left camera
CamR_id = 0 # Camera ID for right camera

CamL= cv2.VideoCapture(CamL_id)
CamR= cv2.VideoCapture(CamR_id)
while(True):

  # Capture the video frame
  # by frame
    ret, frame = vid.read()
  # Display the resulting frame, for filter cv2.convertScaleAbs(frame, alpha=0.8, beta
    =1)

    cv2.imshow('camera', frame)
.
.
.
```

Listing 1: Indices de lentes en cámara estéreo

La ejecución del código anterior retorna un error de indexación, indicando textualmente que el ID número 2 asignado a `cv2.VideoCapture()` no existe o no puede encontrarse, y haciendo prueba con otros valores como 1 o 3 tampoco se obtiene el resultado deseado, por otra parte, comentando la línea

donde se solicita el índice número 2, y solo solicitando el número 1, se obtiene una imagen como en Fig. 2 [b].

Revisando en internet al respecto, mas algunas pruebas entre la Raspberry y el computador, se encuentra el funcionamiento de la indexación para las cámaras, el cual funciona de la siguiente forma: Si el dispositivo posee una cámara principal, el índice asociado a esta es el 0, y si se conecta una cámara adicional, el índice es 2. En el caso de la Raspberry que no posee cámara incorporada, el índice para una cámara externa por USB es 0. Para otras cámaras estéreo, la imagen del primer lente se recupera con el índice anteriormente mencionado, y el segundo lente con el índice de la cámara +1, para la cámara Aptina esto no funciona, por lo que será necesario encontrar la manera de obtener ambas imágenes.
Haciendo las correcciones del código 1, usando solo el primer índice, como se observa en el código 2

```
.
.
.
# define a video capture object
vid = cv2.VideoCapture(2)## 0 with rasp, 2 with pc
while(True):

  # Capture the video frame
  # by frame
    ret, frame = vid.read()
  # Display the resulting frame, for filter cv2.convertScaleAbs(frame[:, :, 1], alpha
    =0.8, beta=1)

    cv2.imshow('camera', frame)
.
.
.
```

Listing 2: Código 1

Se revisa que información retorna la función `read()` sobre el objeto `vid`, `vid.read()` en la variable `frame`, siendo esta una matriz de mínimo 2 dimensiones, que contiene los píxeles de la imagen, y una posible tercer dimensión para los píxeles de la imagen derecha; esta suposición resulta correcta, y se aplica de la siguiente forma

```
# define a video capture object
vid = cv2.VideoCapture(2)## 0 with rasp, 2 with pc
while(True):

  # Capture the video frame
  # by frame
    ret, frame = vid.read()

  # Display the resulting frame, for filter cv2.convertScaleAbs(frame[:, :, 1], alpha
    =0.8, beta=1)
    r_frame=frame[:, :, 1]
    l_frame=frame[:, :, 0]
    cv2.imshow('left', l_frame)
    cv2.imshow('rigth',r_frame)
```

Listing 3: Código 2

Obteniendo el siguiente resultado

(a) Frame 0 (b) Frame 1 (c) Imagen en bruto

Figura 5: Recuperación errónea de imágenes

En la Fig. 5 (c), se observa la imagen entregada directamente por la cámara (RAW), y en las figuras 5 (a) y (b) se muestran las imágenes recuperadas de cada lente; En estas se ve superpuesta la información del otro lente, con una tonalidad invertida, comparando las 3 imágenes se evidencia que el lente izquierdo requiere una mayor exposición a la luz para poder resaltar sus detalles, observando que las regiones claras de la imagen son las regiones rosadas de la imagen directa, mientras que el derecho capta con facilidad las demás intensidades de luz, correspondiendo a las regiones verdes de la imagen original retornada originalmente por la cámara.

Para obtener las imágenes puras de cada lente, se busca indicar a la cámara que no realice dicha comparación de iluminación, o que los valores retornados no contengan dicha información. Para esto se revisa el datasheet de la cámara, el cual indica que su formato de salida es RAW(como se indicó anteriormente ambas imágenes van en una única matriz), pero que permite conversión de formato RAW a RGB, por lo cual se buscan los formatos para entrada y salida de vídeo en la documentación de OpenCV[3], consiguiendo el flag para el método `set()` del objeto de cámara.

```
# define a video capture object
vid = cv2.VideoCapture(2)## 0 with rasp, 2 with pc
vid.set(cv2.CAP_PROP_CONVERT_RGB, 0)## conversion a RGB
while(True):

  # Capture the video frame
  # by frame
    ret, frame = vid.read()

  # Display the resulting frame, for filter cv2.convertScaleAbs(frame[:, :, 1], alpha
    =0.8, beta=1)
    r_frame=frame[:, :, 1]
    l_frame=frame[:, :, 0]
    cv2.imshow('left', l_frame)
    cv2.imshow('rigth',r_frame)
```

Listing 4: Código 3

Con el cual se obtiene el siguiente resultado

[3]https://docs.opencv.org/3.4/d4/d15/group__videoio__flags__base.html

(a) Lente izquierdo (b) Lente derecho

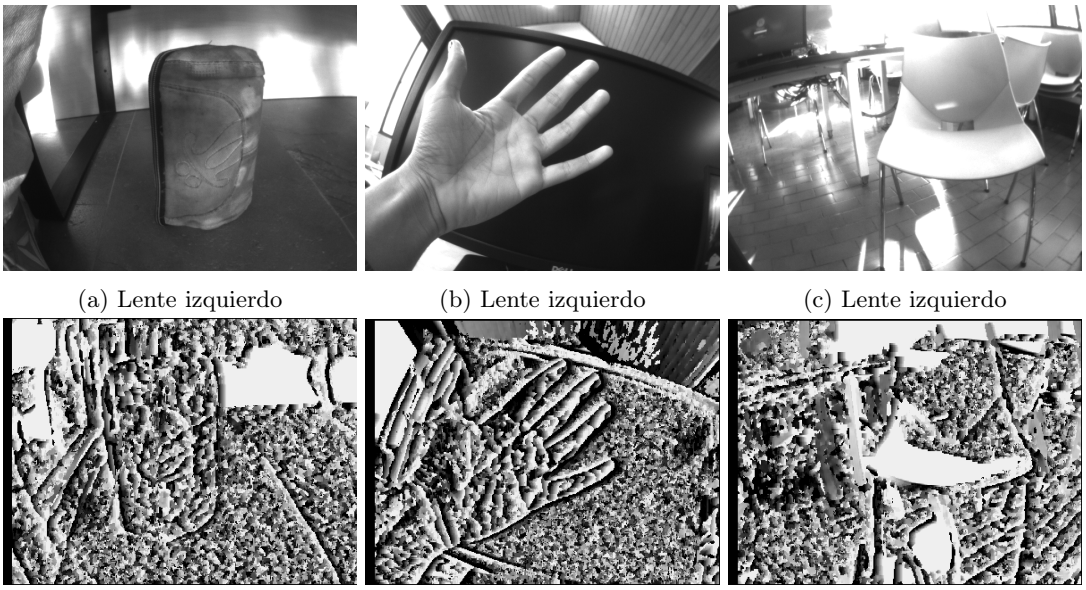
Figura 6: Recuperación correcta de imágenes de cada lente

7. Calibración y Rectificación

El problema de recuperar las imágenes de cada lente es bastante claro, pero, ya con las imágenes, ¿por qué no ir directo a calcular el mapa de disparidad?, esta fue una decisión inicial y en esta subsección se describen los problemas asociados a esto, evidenciando la necesidad de los procesos de calibración y rectificación.

7.1. Mapa de disparidad con imágenes sin rectificar

Un mapa de disparidad es una forma de representar la información de profundidad, donde se define una escala de color para representar la profundidad, asignando a cada valor de la escala de color, un valor de profundidad, como por ejemplo con la escala de grises, lo más blanco representa lo mas cercano, y lo más oscuro lo mas lejano. Obtener un mapa de disparidad es indispensable para recuperar la información de profundidad, y por lo tanto es necesario obtener dicho mapa.
Partiendo del código de LearnOpenCV [8] para obtener un mapa de disparidad con parámetros ajustables, bajo las adaptaciones de la sección anterior para poder recuperar los datos de ambos lentes, y comentando la sección del código asociada a la rectificación de las imágenes, se observan los siguientes resultados



(a) Lente izquierdo (b) Lente izquierdo (c) Lente izquierdo

(d) Mapa de disparidad (e) Mapa de disparidad (f) Mapa de disparidad

Figura 7: Mapas de disparidad sin rectificación.

En las imágenes de la Figura 7, se observa la imagen del lente izquierdo, y el mapa de disparidad asociado; De estas imágenes se pueden identificar dos grandes problemas, si se observa la Figura 7 (e), la pantalla del computador es curva, es decir que la imagen está distorsionada por el lente de la cámara, y en las imágenes de la derecha, los mapas de disparidad no parecen tener mucho sentido, dado que en un mapa de disparidad se espera tener una tonalidad única para cada profundidad, sin embargo de dichos mapas solo pueden rescatarse los contornos de los objetos, donde sin estos, la imagen es completamente irreconocible.

Entonces surge la pregunta, ¿por qué es necesario calibrar y rectificar la cámara?
Al momento de obtener un mapa de disparidad es necesario tener dos imágenes, izquierda y derecha, que según el algoritmo aplicado(en este documento, el Stereo Semi-Global Block Matching [14]) se comparan conjuntos de píxeles de ambas imágenes para hacer una triangulación de bloque a bloque de píxeles y así determinar la profundidad del entorno capturado por la cámara; Para esto son necesarias dos condiciones, haber eliminado la distorsión de las imágenes causada por el lente e imperfecciones entre el lente y el sensor, y que ambas imágenes contengan la misma información pero desde una perspectiva diferente.

Para poder hacer estas correcciones es necesario calibrar la cámara, que consiste en obtener el conjunto de parámetros y matrices intrínsecas de la cámara; Estos resultados luego serán utilizados para hacer las correcciones necesarias en las imágenes capturadas, dicho proceso es llamado rectificación, donde se elimina la distorsión por el lente y se aplican transformaciones sobre cada imagen para por último recortar la zonas de cada imagen que no contiene información común en ambas, como se puede ver el la Fig. 8

(a) Imagenes previas a la rectificación (b) imagenes con rectificación

Figura 8: Efecto de rectificación sobre imágenes estéreo [2]

Para comprender los parámetros se basa la calibración, es necesario entender tres conceptos, la geometría de un lente, la geometría del conjunto de 2 lentes(geometría epipolar) y la distorsión debida al lente.

7.2. Geometría de imagen mono, pinhole

El primer modelo de una cámara es el modelo pinhole(agujero infinitesimal), en este modelo se considera la cámara como una caja en la que el único lugar por el que puede ingresar la luz es por un agujero hecho por una aguja, muy pequeño, justo en frente de una de las paredes de la caja, donde se plasma invertida la imagen capturada por la luz que entra en el agujero. En este modelo no se considera ninguna distorsión de la imagen ya que no hay ningún elemento que perturbe los haces de luz, sin embargo, este modelo es solo para la representación teórica, porque aunque no perturbe la imagen, a mayor enfoque, mas pequeño es el agujero y menor cantidad de luz ingresa a la caja.

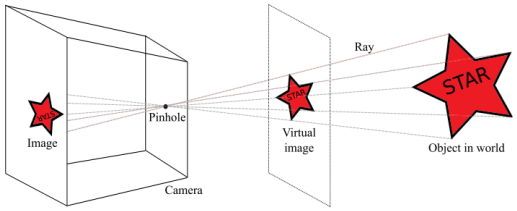


Figura 9: Modelo de cámara tipo pinhole [13]

De la imagen anterior nos interesan tres elementos, las coordenadas del objeto en el mundo real 3D, las coordenadas del objeto sobre la imagen virtual 2D y la transformación del sistema de coordenadas del pinhole con el sistema de coordenadas del mundo real.

Para la conversión de los puntos del mundo 3D al plano 2D se normalizan los valores respecto de la distancia focal

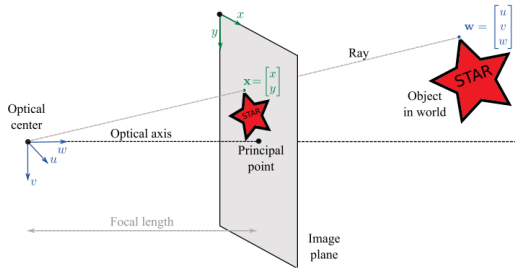


Figura 10: Paso de coordenadas 3D a 2D, se considera el centro óptico centrado con la imagen a la distancia focal) [13]

También nos es de interés las dimensiones de nuestra imagen, ya que la cámara debe discretizar los valores obtenidos en el plano de la distancia focal a un número finito de píxeles, dividiendo las unidades del mundo físico en metros/píxel para el eje X y Y de la imagen. La matriz que se forma con los datos de la distancia focal, las dimensiones de la imagen y el escalado metros/píxel de X y Y se llama matriz intrínseca de la cámara.

Por último debemos tener en cuenta donde y como se sitúa el sistema de coordenadas de la cámara en el sistema del mundo físico, esto se consigue con una matriz formada por los vectores de rotación (**R**) y traslación (**t**) que indican a donde se traslada el origen del sistema de la cámara y hacia adonde apunta o rota su orientación. Lo anterior se puede observar con la siguiente imagen, donde se observa fácilmente la necesidad de estos vectores.

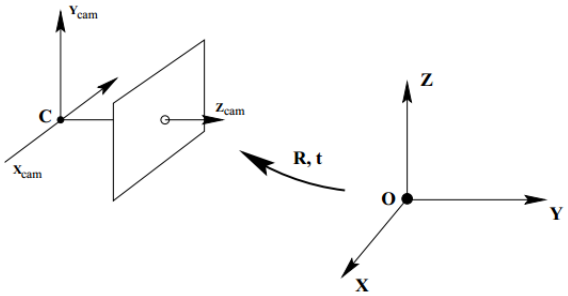
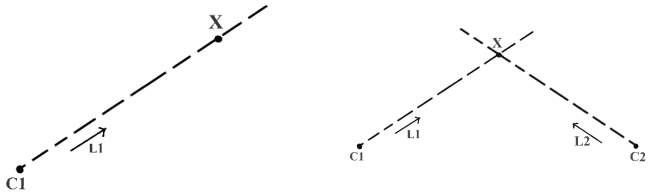


Figura 11: Translación y rotación del sistema de coordenadas de la cámara respecto al mundo físico [6]

7.3. Geometría de imagen estéreo, geometría epipolar

El uso de las cámaras estéreo vs las cámaras mono para la percepción de profundidad es muy simple, un punto proyectado del espacio en la imagen 2D de una cámara mono no contiene información acerca de la profundidad porque estando cerca o lejos de la cámara puede estar contenido en un mismo vector, ver Fig. 12, pero al usar dos cámaras se tienen 2 vectores distintos con un punto común, del cual se puede hacer triangulación.



(a) Mono (b) Estéreo

Figura 12: Geometría de un punto con cámara mono vs. estéreo [14]

La geometría epipolar nos ayuda a describir los puntos del espacio percibidos por una cámara estéreo.

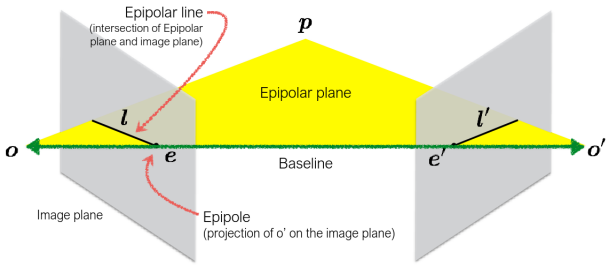


Figura 13: Geometría epipolar [2]

La imagen anterior, Fig. 13, expone algunos de los componentes principales de la geometría epipolar; para obtener la profundidad (distancia entre el Baseline y el punto p) se puede identificar que en plano epipolar, que es infinito, solo dos vectores desde los puntos o y o' dan con el punto p, entonces se puede armar un sistema de dos ecuaciones y dos incógnitas con la lineas epipolares de cada imagen y los vectores que apuntan al punto p; este proceso es aplicable a cualquier otro punto p' en el mismo plano epipolar, y para cualquier planos epipolar paralelo al plano epipolar original siempre que corte con los planos de imagen.

La calibración estéreo se encarga de mapear los puntos de los planos de imagen ideales de la geometría epipolar a los planos de cada lente de la cámara estéreo, para lo cual se necesitan las matrices intrínsecas de cada lente, y una vez mapeados dichos puntos se sitúan las líneas de fuga de ambas imágenes a una misma altura, para que después del mapeo el punto p siga estando en el mismo plano epipolar de ambas imágenes, que fue mapeado como una recta (Baseline), para así volver a tener un sistema de dos ecuaciones y dos incógnitas [3].

7.4. Distorsión debida al lente

La distorsión debida al lente, es aquella que deforma la imagen, haciendo un efecto de barril o también de fondo de botella [9], como se puede ver en la siguiente imagen

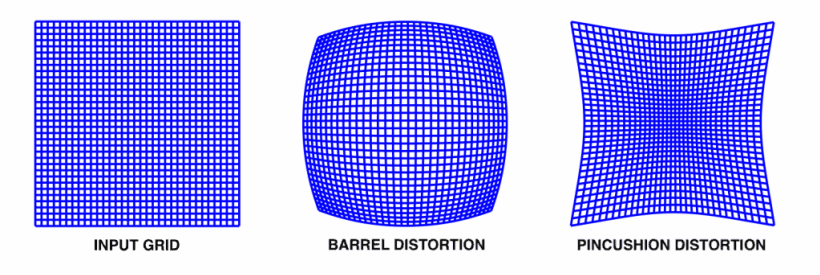


Figura 14: Tipos de distorsión radial, distorsión de barril(medio) y distorsión de fondo de botella(derecha) [9]

Para parametrizar o definir que tan distorsionada está una imagen, se tienen los coeficientes de distorsión radial y tangencial, donde la distorsión radial indica que tanto se corre un píxel a un ángulo fijo pero desplazado a un radio mayor del real, y la distorsión tangencial es cuando un píxel a se desplaza por la circunferencia descrita por un radio fijo.

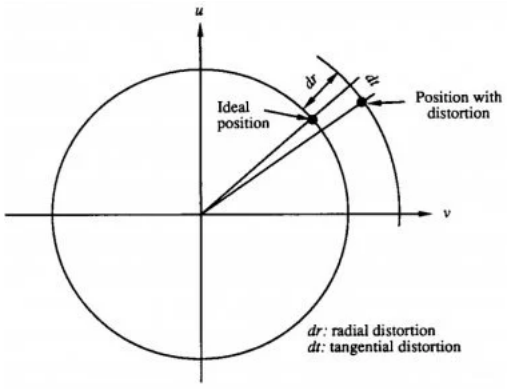


Figura 15: Visualización radial y tangencial [9]

Los coeficientes de distorsión radial pueden ser 2 o 3, y los tangenciales 2.

7.5. Calibración y Rectificación con Python y MATLAB

El proceso de calibración consiste en obtener la matriz intrínseca de la cámara y los coeficientes de distorsión para hacer el proceso de rectificación, el cual recupera la imagen original que capta la cámara, eliminando los efectos de distorsión del lente y seleccionando solo la región de la imagen que conserva la información sin perturbaciones.

La calibración de una cámara mono y una estéreo son idénticos, solo que en el caso de una cámara estéreo es necesario obtener adicional una matriz que relacione ambas imágenes.

Para que un algoritmo haga la identificación de los coeficientes de distorsión, obtenga la distancia focal y la matriz intrínseca necesita recuperar puntos o coordenadas del mundo físico y hacerlas pasar a través de los lentes de la cámara para así poder plantear las ecuaciones necesarias. Para recuperar puntos de la imagen de la cámara que correspondan con puntos del mundo real el método más sencillo es usar puntos del mundo real que se puedan reconocer y reconstruir después de pasar por la cámara sin perder la información de cómo deben ser, es decir, si entregamos un conjunto de puntos desconocidos a la cámara, solo conoceremos como se verán exactamente después de pasar por la cámara, dejándonos con una ecuación y dos incógnitas, pero si hacemos pasar puntos que conocemos exactamente como son

antes de pasar por la cámara, después de pasar por la cámara sabremos como han sido perturbados por el lente y como deberían verse, entonces son una ecuación y una incógnita. Por ejemplo, si se tomasen puntos rojos formando una linea recta con la cámara, el resultado serán puntos formando una curva, entonces ya tenemos la información necesaria, porque conocemos los puntos después de pasar por la cámara, y al obtener la transformación que genere la linea recta de puntos otra vez, tendremos el resultado deseado.

Esto es en general el proceso que se utiliza en casi todo software disponible para obtener la calibración de cámaras. El conjunto de puntos reconocibles suele ser un tablero de ajedrez, para identificar los vértices de los cuadrados de las casillas, como se puede observar en la siguiente imagen.

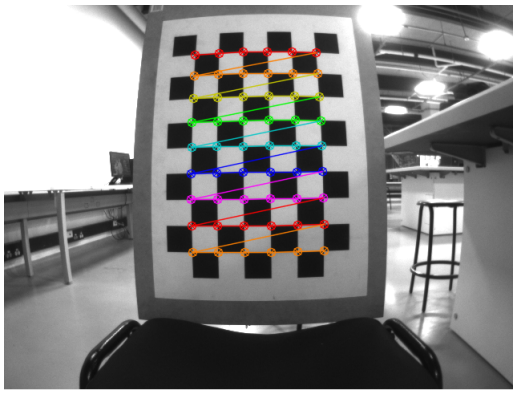


Figura 16: Ejemplo de calibración con tablero de ajedrez

7.6. Procedimiento para la toma de fotos

En la Figura 16, se observa el tablero a utilizar, este es de 7x10 casillas, con casillas de 5cm de lado, sobre el cual se reconocieron los vértices de las casillas internas.

La toma de imágenes para calibración debe cumplir las siguientes condiciones:

- El conjunto de fotos debe tener por lo menos 50 elementos por cada lente, un buen valor son entre 100 y 150 fotos.
- Las fotos deben variar la distancia al tablero y el ángulo desde el cual es observado, así mismo, el tablero no debe permanecer en el centro de la imagen sino desplazarse a los extremos, que es donde mas se ve el efecto de distorsión
- Dado que se trabaja una cámara estéreo, ambos lentes deben estar a la misma altura del suelo, y en el caso de querer realizar tomas de fotos inclinadas(un lente a diferente altura de otro), el ángulo de inclinación no debe superar los 45 grados
- Tener en cuenta la distancia de enfoque de la cámara, para no tener dificultad en reconocer los puntos del tablero

7.7. Calibración con Python

Se utiliza un código de OpenCV para Python, el cual lee el conjunto de imágenes, identifica los puntos del tablero, genera la matriz intrínseca, un vector de coeficientes de distorsión(3 de distorsión radial y 2 tangenciales), la matriz de rotación y el vector de traslación, con estos elementos genera un mapa de rectificación para cada lente, se almacena el mapa de rectificación en un archivo .xml que luego es leído y aplicado sobre las imágenes.

Se tomaron 5 conjuntos de fotos, de los cuales algunos no cumplían todas las condiciones mencionadas anteriormente, entregando resultados de la imagen rectificada como el siguiente

Figura 17: Rectificación lente izquierdo, primeros 5 conjuntos de fotos

La rectificación de la imagen anterior claramente es incorrecta, por lo que es necesario probar otro entorno para realizar un nuevo conjunto de fotos, entonces se busca un sitio con menor luminosidad que los anteriores, y es en esta toma de fotos donde se descubren algunas de las condiciones mencionadas, por lo que el resultado de la rectificación mono es exitoso para cada lente

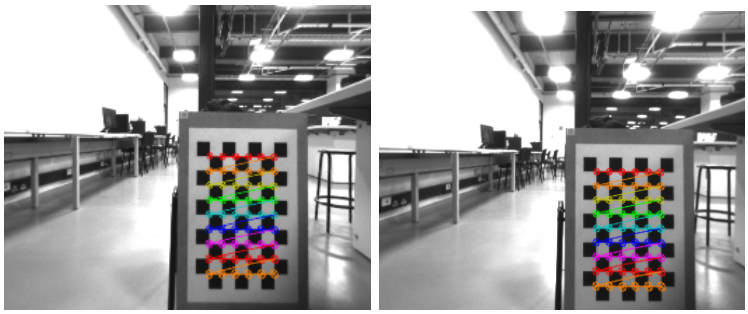


Figura 18: Rectificación individual de cada lente, nuevo set de fotos

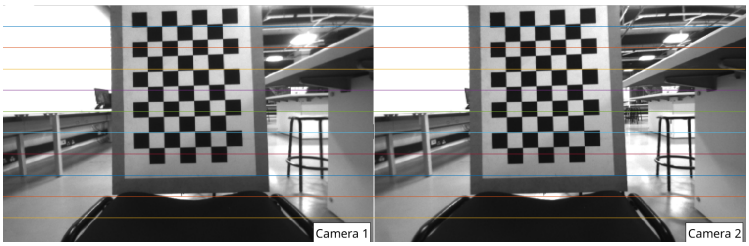
De la Figura 18 se observa una rectificación con resultados mucho mejores a los anteriores, sin embargo estos resultados fueron obtenidos de forma individual, por lo que se puede observar que ambas imágenes no tienen la misma altura, es decir que las lineas epipolares de cada imagen no pertenecen al mismo plano epipolar, osea que el resultado no es funcional para la obtención de la disparidad, además de que con este método de rectificación la imagen reduce su tamaño considerablemente, pasando de una imagen de tamaño 640x480 a 340x277.

Al probar la rectificación estéreo el resultado es sorpresivamente una imagen totalmente en negro, que luego de hacer algunos análisis dentro del código se decide probar a reducir gradualmente el número de coeficientes de distorsión que se utilizan, probando los casos de 1, 2 y 3 radiales, luego 3 radiales y uno tangencial, por último 3 radiales y 2 tangenciales, que entregando resultados mas cercanos a los obtenidos con la rectificación mono, aún quedan lejos de ser una rectificación útil, los resultados se pueden observar en la sección 14.1 en Anexos

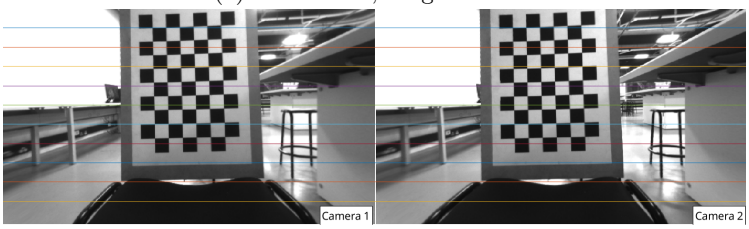
En este nuevo caso, el resultado es mucho mejor, sin embargo el resultado todavía no es el deseado. Ante 6 intentos de calibración y rectificación con Python fallidos, se revisa la herramienta de MATLAB para calibración estéreo,

7.8. Calibración con MATLAB

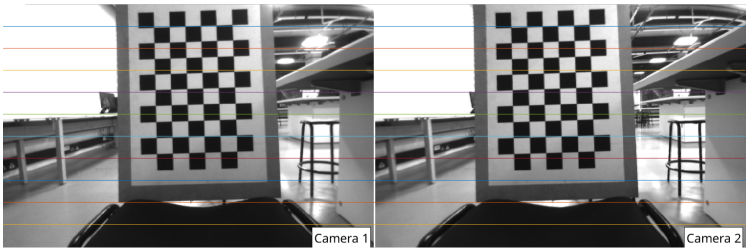
La herramienta de MATLAB se llama Stereo Camera Calibrator, que maneja la misma estructura de funcionamiento de Python, pero muy seguramente con un algoritmo de obtención de los parámetros diferente. Pero cabe mencionar que la cantidad de coeficientes de distorsión a identificar es ajustable por el usuario. Empleando el último conjunto de fotos usado en Python, se obtiene el siguiente resultado



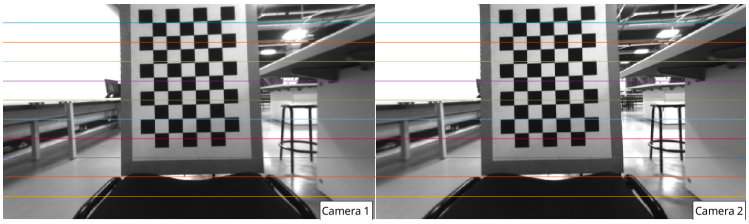
(a) Radiales: 2, tangenciales:0



(b) Radiales: 3, tangenciales:0



(c) Radiales: 2, tangenciales:2



(d) Radiales: 3, tangenciales:0

Figura 19: Rectificación y calibración mediante MATLAB, se usan diferentes cantidades de coeficientes de distorsión, para cada caso con 0.23 píxeles de error medio

Dado que en todos los casos se mantiene el mismo error medio(seguramente por ser el mismo conjunto de imágenes), se revisan todas las imágenes y se elige cualitativamente el caso donde se observen menos imágenes que todavía presenten distorsión, eligiendo usar solo dos coeficientes de distorsión radial

7.9. Uso de parámetros de MATLAB en código de Python

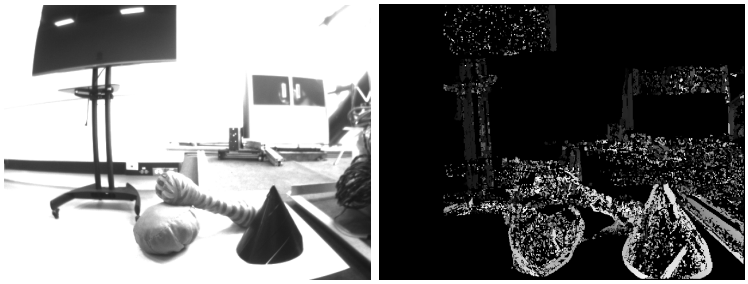
Esta etapa consiste únicamente en exportar los parámetros de cada cámara en MATLAB al entorno de trabajo, y copiar manualmente las matrices y vectores en el código de Python, para generar el mapa de rectificación con estos valores. Se aclara que los resultados finales varían levemente en cuanto a la región total de la imagen, pues el único vector que no se genera en MATLAB pero si en python es uno llamado ROI(Region Of Interest), pero igualmente las variaciones son muy sutiles.

8. Obtención de profundidad

Teniendo ya la rectificación de imágenes es posible generar mapas de disparidad adecuados, para lo cual se toman en cuenta los algoritmos BM(Block Matching) y SGBM(Semi Global Block Matching) que hacen parte de la librería de OpenCV.

8.1. Métodos

Inicialmente se realizan ensayos con el algoritmo de BM, con opción de configurar los distintos valores del algoritmo, sin embargo este nunca entrego los resultados deseados, ya que como se observa en las Figuras 7 que aun después de la rectificación de las imágenes, no es posible obtener de manera funcional la superficie de los objetos(una disparidad única y uniforme en la superficie del objeto) sino por el contrario una disparidad únicamente en los bordes de los objetos, mientras que a lo largo de su superficie se obtiene algo mas parecido al ruido de sal y pimienta. Pese a los errores de este algoritmo presenta dos grandes ventajas, la primera es que es inmune a grandes variaciones del nivel de luminosidad, no solo por no cambiar su comportamiento o valores de disparidad con grandes incrementos luz, sino que no se forman regiones çercanas.[a]l observar directamente una lampara con la cámara; La segunda ventaja es que ante regiones del espacio que reconoce la cámara que son muy lejanas, este simplemente no entrega ruido o algún valor indeseado, sino que asume directamente el valor de cero para estas regiones, ejemplo, en un salón muy grande solo se perciben los objetos notorios como mesas, lamparas, botellas, personas o sillas, pero las paredes no son reconocidas en el mapa de disparidad. En la siguiente imagen se puede observar un ejemplo de su funcionamiento



(a) Imagen izquierda (b) Disparidad

Figura 20: Ejemplo de funcionamiento del algoritmo BM

Debido a los resultados del algoritmo BM se hace prueba del algoritmo SGBM, el cual exige una mayor capacidad de procesamiento que el anterior pero ofrece mejores resultados, mencionando que de este si se pueden apreciar las superficies con un valor de disparidad, además de ser bastante ruidoso, sufre dos grandes problemas, los cuales son su alta sensibilidad a los niveles de iluminación del ambiente, y su respuesta ante grandes superficies como paredes o regiones muy amplias o lejanas, este entrega un resultado completamente erróneo, dando valores de disparidad como una especie de ruido

de lineas verticales y horizontales de distintos valores de disparidad.

Para obtener mejores resultados de este algoritmo se trabaja un código para el ajuste de sus distintos parámetros, sin embargo su comportamiento ruidoso hace que la disparidad obtenida de ciertos ambientes sea completamente inútil, ya que el ruido afecta a todo el mapa de disparidad haciendo la información irreconocible, por lo que se buscan maneras de filtrar el resultado de este algoritmo, encontrando el filtro WLS exclusivamente para uso con el SGBM, este filtro también maneja 4 parámetros ajustables, de los cuales en su documentación solo suelen configurar dos de estos, entregando muy buenos resultados como se puede observar en la Fig 23, pero con cambios sutiles de la iluminación el mapa de disparidad pasa a ser una imagen totalmente en blanco, cosa que ocurre prácticamente todo el tiempo mientras se utiliza el código, e inclusive con casi todas las imágenes de los conjuntos de fotos realizados anteriormente para la calibración, incluyendo el set de fotos final para la calibración.

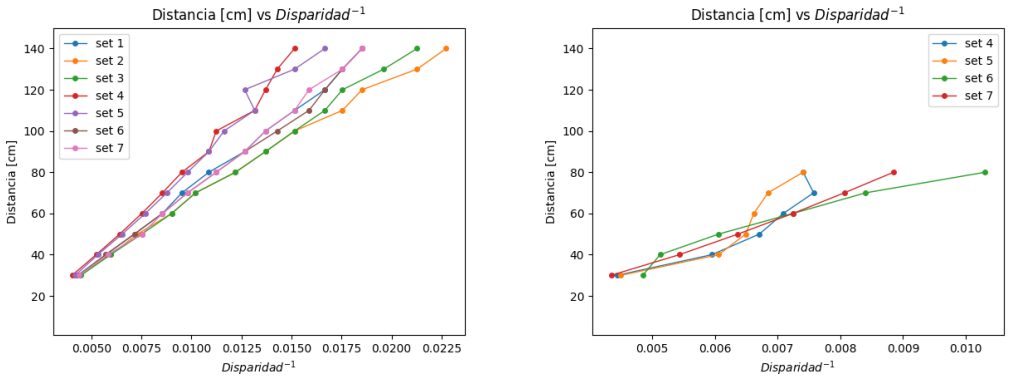
Ante lo anterior se modifica el código para tunnear los parámetros del algoritmo y del filtro, probando con 8 nuevos conjuntos de fotos en los que se prueban diferentes entornos y niveles de profundidad, obteniendo en la mayoría de los casos resultados negativos, hasta obtener algunos valores relativamente genéricos ante las variaciones de luminosidad del entorno.

De los 8 conjuntos de fotos se usan solo la mitad de estos en los casos que se aplica el filtro WLS, esto porque el uso del filtro genera algunos cambios del valor de la disparidad dependiendo de si el objeto enfocado es blanco o negro, y en los conjuntos de fotos descartados el objeto del cual se toma el dato de la disparidad es un papel blanco con la distancia a la que se encuentra escrita en color negro, esto genera errores de los valores de disparidad solo cuando se utiliza el filtro sobre la disparidad.

El montaje para realizar estos conjuntos de fotos se puede observar en la sección 15 de los anexos. En este se sitúa la cámara frente al objeto y se separa desde 30cm hasta 140cm cada 10cm, para luego comparar el valor inverso de la disparidad contra la distancia a la cual se encontraba en ese momento.

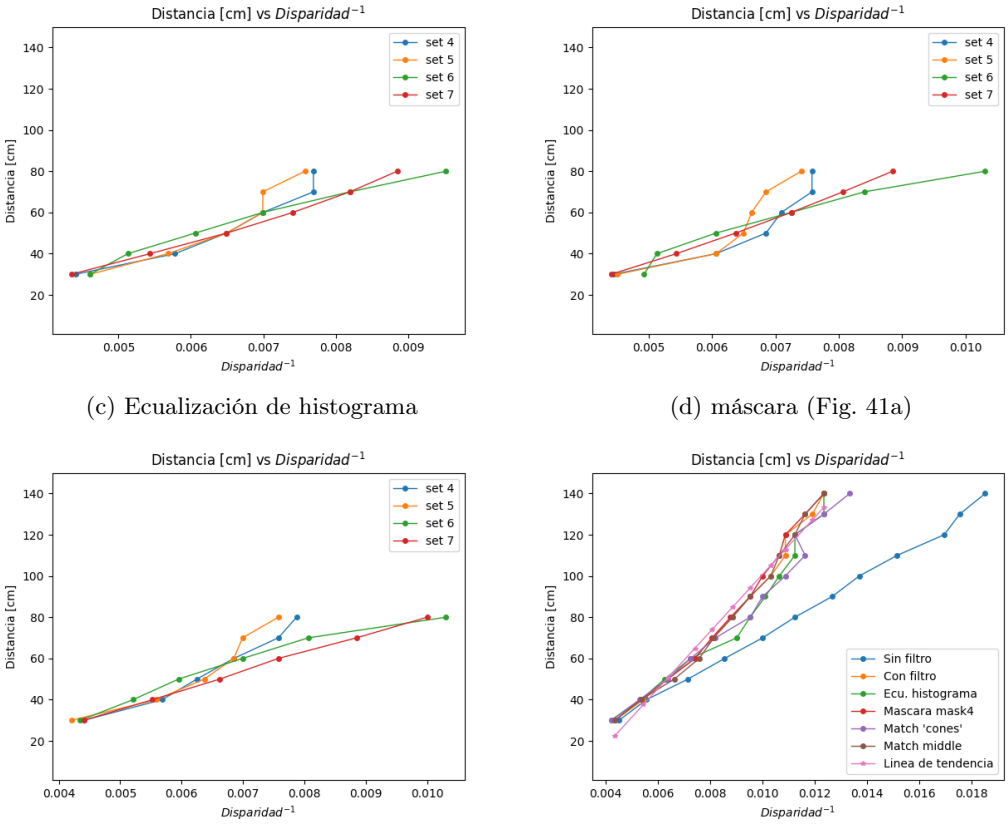
Ante los errores debidos a luminosidad en el calculo de la disparidad, se prueba aplicar ecualización de histograma, histograma de referencia, y máscara sobre las imágenes de la cámara antes del proceso de rectificación, para observar si hay alguna mejora en el resultado de la disparidad. El proceso de máscara consiste en aplicar una imagen sobre las imágenes originales para oscurecer ciertas regiones de la imagen original.

8.2. Resultados del SGBM



(a) Sin filtro WLS

(b) Con filtro WLS



(c) Ecualización de histograma

(d) máscara (Fig. 41a)

(e) Histograma de referencia (Fig. 41b)

(f) Curva en tiempo real

Figura 21: Curvas de distancia vs. disparidad^{-1}. En la figura (a) se observan las curvas de todos los conjuntos de fotos realizados sin filtrado del mapa de disparidad, para todas las siguientes figuras se aplica el filtro WLS sobre el mapa de disparidad. Se aplican algunas herramientas de histogramas para intentar corregir el comportamiento del filtro WLS sobre la disparidad. En la figura (f) se aplican todos los métodos anteriores pero no sobre un set de fotos sino de una toma de datos en vivo, además de una linea de tendencia para los datos donde solo se aplica el filtro WLS con la siguiente ecuación $z = 13850{,}42/disp - 37{,}69$.

9. Algoritmos de segmentación

El propósito de realizar segmentación en este proyecto es identificar objetos o cuerpos a partir del mapa de disparidad. Con esta información es posible determinar a que distancia se encuentran los objetos para saber si el vehículo puede o no colisionar con estos y tomar la decisión de si detenerse o no.

La pregunta correspondiente es, ¿qué método de segmentación utilizar?. Para saber esto hay que considerar cuales son los métodos de segmentación disponibles en las librerías de OpenCV, o si se desea utilizar alguna otra librería que contenga otros métodos. Se consiguen los siguientes métodos de segmentación para Python

- OpenCV: Watershed
- scikit-image: Felzenszwalb, Quickshift, SLIC Superpixels, Watershed entre otros [4]

Es claro que OpenCV queda bastante limitado en cuanto a métodos de segmentación se trata, pero

se decide qué librería usar con base en el tiempo de procesamiento, así que se prueban ejemplos de cada método mencionado (medianamente ajustado) y se verifican los tiempos de ejecución de estos con la librería `time` de Python, obteniendo una respuesta bastante clara para la elección de la librería; Los resultados obtenidos de cada método se puede observar en la sección de Segmentación en Anexos 15.2

Cuadro 1: Tiempos de ejecución en PC

Librería	**Método de segmentación**	**Tiempo de ejecución[s]**
scikit-image	felzenszwalb	0.5
	slic	0.26
	quickshift	9.3
	watershed	0.16
OpenCV	watershed	0.015

De la anterior tabla se decide usar el algoritmo de Watershed de la librería OpenCV

9.1. Watershed

Según explica la pagina web de Watershed [7], el funcionamiento del algoritmo es el siguiente

- Una imagen en escala de grises se puede considerar como una superficie topográfica.
- El algoritmo se encarga de inundar cualquier cuenca existente en el mapa topográfico, y una vez la cuenca llega a tope, evita que su inundación se mezcle o junte con otra.
- El funcionamiento del item anterior sufre sobresegmentación, como puede ser el caso con una imagen ruidosa, entonces se entrega al algoritmo unos marcadores que le sirve como semilla para iniciar la inundación.

9.2. Generación de marcadores para Watershed

La función de OpenCV, watershed, recibe una imagen de un solo canal y un marcador como una matriz de dos dimensiones del mismo tamaño que la imagen. El marcador consiste en una matriz donde los ceros representan regiones que no son conocidas y que watershed deberá reconocer de la imagen original por cuenta propia, mientras que cualquier otro valor positivo representa una región ya identificada para que watershed termine de inundar esa región de la imagen.

El marcador consiste en una matriz de números naturales que forman regiones continuas, donde cada región cuyo valor es diferente de cero es una semilla para que el algoritmo de watershed inunde dicha región hasta toparse con un borde de valor cero, el cual es una región aun desconocida, cuando termina la inundación de una región se denomina un segmento, el cual se separa de otros segmentos por un contorno de valor -1.

Para formar el marcador es necesaria una matriz de valores 0 o 255, donde los valores de 0 una vez mas son regiones desconocidas y los valores de 255 son regiones que deben formar parte de un segmento, es decir, esa es una cuenca en la imagen que ya se empezó a inundar. Esta matriz se denomina 'máscara' y cuando una región de valor 255 se identifica en su totalidad hasta el limite donde se encuentra con valores de 0, esta región sera un valor del marcador, es decir, es la semilla para un segmento.

Para reconocer los objetos dentro del mapa de disparidad, se considera que un objeto está en casi su totalidad a una misma distancia de la cámara, por lo que cualquier región del mapa de disparidad que tenga en promedio el mismo valor deberá ser un objeto, entonces lo que delimita el contorno del objeto

es el cambio de profundidad, dicho esto es bastante intuitivo obtener la derivada o gradiente de la imagen, donde los valores mas altos obtenidos describirían los bordes de los objetos, pero otros valores aunque no tan altos pero si significativos, como las variaciones de la superficie del objeto, podrían ser confundidos e interpretados como parte del contorno, afectando el resultado.

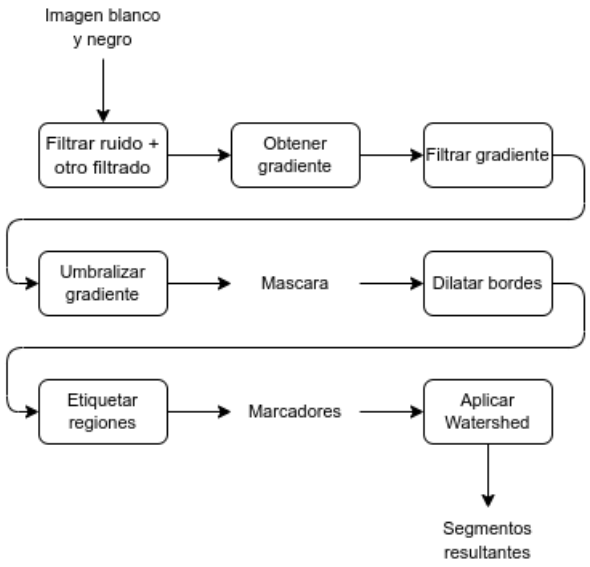


Figura 22: Pasos para generar el marcador

Como se mencionó en el párrafo anterior, las pequeñas variaciones sobre el mapa de disparidad debidas a errores de la misma disparidad o cambios en la superficie de los objetos puede resultar en segmentos no deseados, para lo cual es necesario procesar un poco la imagen, en otras palabras aplicar un filtro blur sobre la imagen, después de esto se calcula el gradiente de la imagen, pero ya que una imagen es una información discreta, hay diferentes métodos para calcular el gradiente, de modo que se probaran varios de estos métodos para comparar los resultados. Dependiendo del método utilizado sera necesario filtrar la imagen sin ruido antes y después del gradiente de una u otra forma para mejorar los resultados. Una vez el gradiente es calculado se compara con un nivel umbral para obtener valores de 0 o 1 traducido en 0 o 255 para negro o blanco, donde los bordes de las regiones en la imagen resultante se toman con el valor de 0 y se dilatan (engruesan), esto con el fin de que si el contorno de una región por alguna razón no llegase a cerrarse, este se cierre, ya que de lo contrario se mezclarían dos posibles segmentos en uno solo, perdiendo información. Por último, cada región blanca delimitada de otras regiones blancas por un borde negro se le asigna un numero, para definir a cada una de estas como una semilla para un segmento, por último se aplica la función de watershed y se obtienen los segmentos de la imagen.

Con lo anterior dicho, se decide probar diferentes formas de obtener el gradiente de la imagen, además de probar métodos basados en frecuencia, y comparar los resultados. En las siguientes figuras se muestran la respectiva detección de contornos por los diferentes métodos y la forma de la máscara resultante, donde se observan mejores resultados por lo métodos de Sobel, Scharr y Canny, sin embargo dada la semejanza entre los diferentes resultados, una vez mas se tomará preferencia al método que implique el menor tiempo de ejecución, siendo Canny el método de preferencia; En la tabla 1 se observan los diferentes tiempos de ejecución para cada algoritmo realizado en este proyecto.

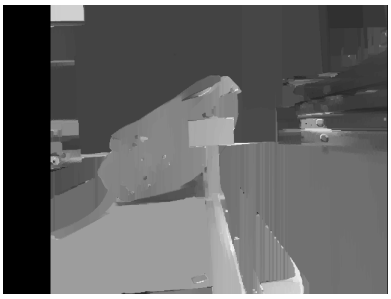
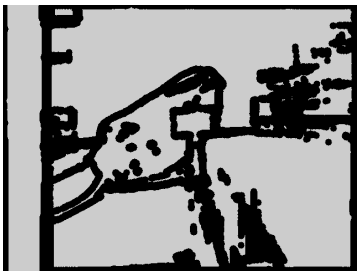


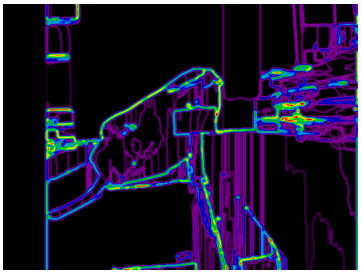
Figura 23: Mapa de disparidad original

La imagen anterior corresponde al mapa de disparidad de un riel sobre el cual está un bolso y frente a el una figura de dos piezas rectangulares. Los objetos de la derecha son mas lejanos y las lineas verticales en la parte inferior derecha son errores causados por el filtro del algoritmo SGBM.

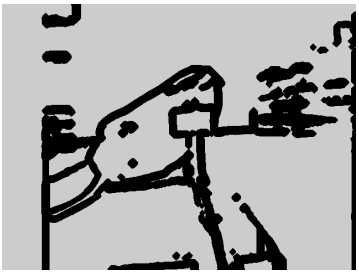
(a) Filtro pasa altos sobre la FFT



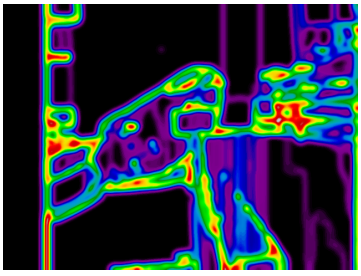
(b) Marcadores obtenidos



(c) Gradiente obtenido con Sobel

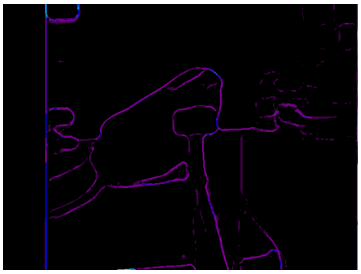


(d) Marcadores obtenidos



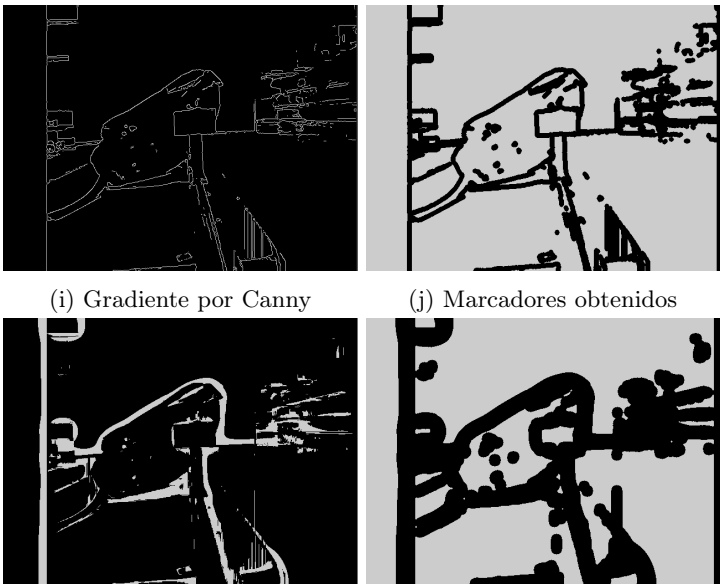
(e) Gradiente por Scharr

(f) Marcadores obtenidos



(g) Gradiente por Laplace

(h) Marcadores obtenidos

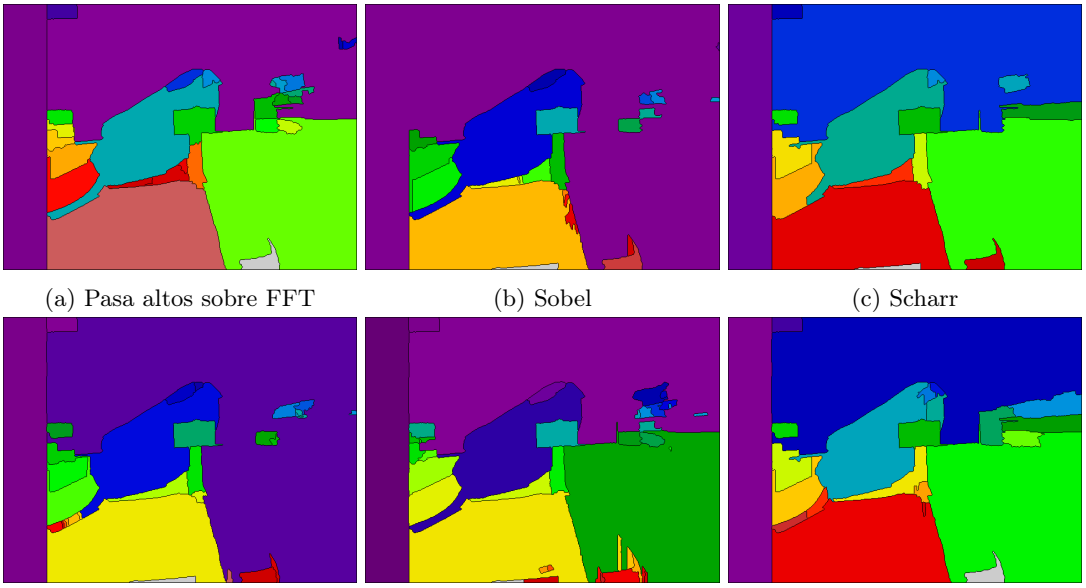


(i) Gradiente por Canny (j) Marcadores obtenidos

(k) Filtro pasa altos (l) Marcadores obtenidos

Figura 24: Generación de mascaras por distintos métodos

9.3. Resultados de segmentación



(a) Pasa altos sobre FFT (b) Sobel (c) Scharr

(d) Laplace (e) Canny (f) Filtro pasa altos

Figura 25: Resultado de segmentación por Watershed con diferente generación de mascaras

10. Identificación de carril

Como se mencionó al inicio del documento en la Figura 1, el vehículo debe ser capaz de identificar el carril por el que va a conducir, para esto debe identificar únicamente las lineas que delimitan el

carril, para poder asociar ecuaciones o un conjunto de puntos a las lineas obtenidas. En la siguiente Figura 26 se observa un diagrama con los pasos a seguir para realizar esto

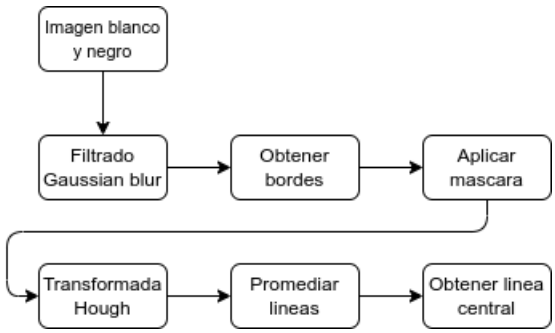


Figura 26: Procesos para obtención del carril

Lo primero es entregar una imagen en blanco y negro del carril, la imagen debe estar previamente rectificada, para este caso se utilizara la imagen recuperada por el lente izquierdo.

Figura 27: Imagen original

Sobre la imagen original se aplica un filtro tipo blur gausiano, esto con el fin de eliminar ruido y desenfocar un poco la imagen para que al identificar los bordes con el algoritmo de Canny, estos sean debido a los contrastes entre los colores más significativos en la imagen, como el contraste entre negro y blanco y se eliminen efectos nos deseados como bordes obtenidos por resplandor o reflejos sobre otras superficies, e inclusive, como es el caso de la imagen superior, Fig. 27, se forman en el suelo manchas blancas debido a la ecualización de histograma aplicada sobre la imagen al momento de rectificarla, dichas manchas generan bordes no deseados en la imagen que se verán en las imágenes posteriores.

El motivo por el cual es conveniente usar imágenes con ecualización de histograma es realzar los colores blanco y negro sobre la imagen, facilitando la identificación de los bordes del carril, esto es muy conveniente en condiciones donde hay alta luminosidad y se opacan las lineas del carril, sin embargo esto también trae consigo problemas como lo mencionado anteriormente, la ecualización realza los colores blancos y negros producidos por sombras y reflejos, dando bordes no deseados.

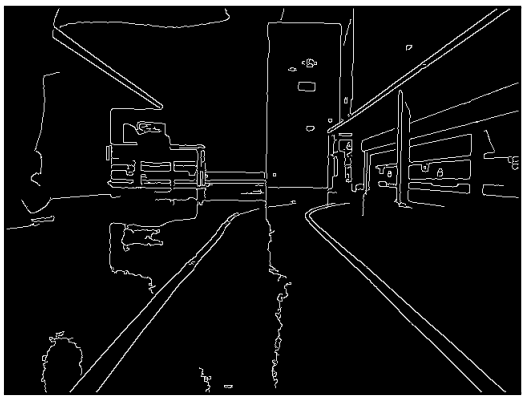


Figura 28: Imagen original

La imagen anterior corresponde con los bordes obtenidos de la imagen, y como se puede observar hay muchos bordes que podrían interpretarse como lineas de carril, desde las mesas hasta los bordes de los reflejos en el suelo, esto se corrige de la siguiente manera, considerando una región de interés de la imagen, de modo que solo se tengan en cuenta los bordes en esta región.

La siguiente imagen muestra la región de interés

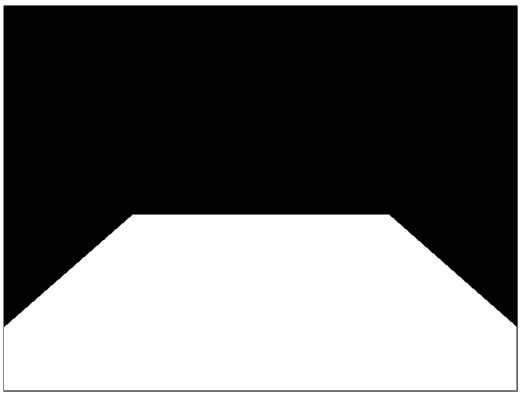


Figura 29: Región de interés (ROI)

La imagen de la Figura 29 se usa como máscara sobre la imagen que contiene los bordes del carril, eliminando todo lo que se encuentre en la región negra. La forma de la región blanca se hace considerando que el carril se hace angosto en el centro de la imagen, y tiene cierta altura en la parte inferior de los lados como precaución si una linea de carril no empieza desde el borde inferior de la imagen sino desde un costado de la imagen.

El resultado de aplicar la máscara sobre los bordes es el siguiente

Figura 30: Bordes bajo máscara

Para poder identificar las lineas del carril se toma este nuevo conjunto más reducido de bordes, para reconocer todas las lineas presentes en la imagen, y de estas recuperar solo aquellas que corresponden con los carriles, para esto se utiliza la transformada de Hough, que se explicara brevemente.
La transformada de Hough parte del hecho de que una linea recta en su representación polar $xcos(\theta) + ysin(\theta) = \rho$, puede recibir como parámetros no a θ y ρ, sino a x y y, de manera que cada punto (x, y) de la recta tiene una representación senoidal en el plano $\theta - \rho$, de modo que los N puntos (x, y) de una linea generan N representaciones en el plano $\theta - \rho$ todas con intersección en un único punto [16]. Así, para un conjunto de más de dos puntos que tienden a formar una recta, sus representaciones de la transformada Hough no tendrán un intercepto, pero sí un conjunto de puntos igual de grande que se acercan mucho, y entre más tiendan a ser colineales los puntos originales, más tienden a ser el mismo punto los interceptos de sus representaciones, con dicha dispersión se determina si estos forman o no una linea recta.

Con esto en mente, la función de la transformada de Hough en Python requiere definir un mínimo de puntos detectados y una dispersión(en píxeles) entre estos para que puedan ser considerados una linea, así lo más probable es que solo los bordes debido al carril sean aquellos que retornen lineas identificadas, y las demás lineas no deseadas no cumplan dichas características. El resultado de la función de Hough se muestra en la siguiente imagen.

Figura 31: Lineas obtenidas con la transformada de Hough

De la imagen anterior se aprecia que no se forman lineas completamente continuas, sino interrumpidas, e inclusive las lineas de mayor extensión están en realidad formadas por la unión de dos o más lineas con puntos iniciales y finales que coinciden, pero cuyas pendientes varia ligeramente aunque no

sea algo apreciable a simple vista. para cada conjunto de lineas, las de izquierda y derecha, es necesario promediarlas para obtener una linea central de cada lado, sin embargo la función de Hough en Python retorna un arreglo de vectores de cuatro posiciones que contienen las coordenadas de píxeles de cada linea, entonces para agrupar las lineas de cada lado del carril es necesario obtener la pendiente de cada linea, y dependiendo de su signo, agrupar todas las lineas en un vector para la izquierda y otro para la derecha, de modo que el promedio se haga sobre estos nuevos vectores y se plasmen una vez mas sobre la imagen original. Este método tiene un problema importante, y es que en el caso de que ambas lineas del carril tengan pendientes del mismo signo, no sera posible hacer un promedio para cada lado del carril, y por lo tanto, solo se identificará una linea del carril en vez de dos.

Figura 32: Lineas promedio

Una vez con el carril identificado, se necesita que el vehículo se dirija por el centro del carril, para esto se promedian una vez más ambas lineas del carril en una linea central, cuyas coordenadas se entregan por una función luego de mostrar las lineas del carril en la imagen, además de entregar el ángulo que se forma entre el eje vertical y la linea central.

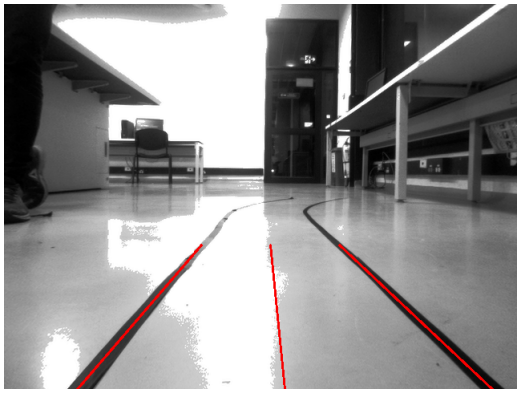


Figura 33: Lineas promedio mas linea central

Hasta la imagen anterior el problema de la identificación del carril parece solucionado, sin embargo es necesario considerar un caso extremo, en el que solo una linea o ninguna es identificada, ya sea por error de la recuperación de los bordes, la forma de la máscara o porque la cámara no capto el carril completo durante un momento, lo cual resultaría en más de un error porque al intentar generar la linea central el vector de la linea no identificada se encontraría vacío, por lo tanto, para esta circunstancia, se verifica si el vector final de cada linea se encuentra o no vacío, y en caso de estarlo, se genera una linea por defecto a 5 píxeles del borde correspondiente, cosa que aunque brinda una aproximación muy

adecuada del problema, permite que el vehículo no deje de recibir información en ninguna situación, cabe aclarar que para estos casos sera necesario comparar las coordenadas del carril reconocido con el carril por defecto en caso de error, para que el vehículo sepa si está siguiendo una trayectoria identificada (de confianza) o una por defecto (no es de confianza), y este tome alguna decisión al respecto.

Una última aclaración de la generación de las trayectorias promedio, es que la trayectoria completa no se toma hasta donde pudo identificar el carril como una linea recta, sino que es un valor fijo para todas las posibles trayectorias.

Figura 34: Caso extremo, reconocimiento de una sola linea

11. Sensor VL53L0X

Este sensor es una añadidura al funcionamiento de la cámara por dos principales motivos, 1. aunque tiene una mayor precisión que el cálculo de profundidad con la cámara, solo toma el dato del sitio al que apunta y 2. al generar el mapa de profundidad hay ciertas regiones del espacio que se pierden dentro del mapa, por lo que para la cámara, el obstáculo no se encuentra ahí.

La librería para utilizar este sensor se obtiene del siguiente repositorio de GitHub de Adafruit [5], en el cual también indican como instalar la librería con el comando `pip3` desde terminal y se mencionan sus dos dependencias,

- `Adafruit CircuitPython`
- `Bus Device`

Para poder instalar la librería primero se deben satisfacer sus dependencias. Los comandos para instalar `Adafruit CircuitPython` se recuperan del siguiente tutorial de Adafruit [10]

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3-pip
sudo pip3 install --upgrade setuptools
sudo pip3 install --upgrade adafruit-python-shell
wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/raspi-
linka.py
sudo python3 raspi-blinka.py
```

Para instalar el paquete `Bus Device` se usa el siguiente comando

```
sudo pip3 install adafruit-circuitpython-busdevice
```

y por último se instala la librería del sensor VL53L0X

```
sudo pip3 install adafruit-circuitpython-vl53l0x
```

Ahora, para hacer uso del sensor ToF(VL53L0X) basta con importar las siguientes librerías al código en ejecución y tomar los datos con el metodo `range` sobre el objeto `tof`

```
import board
import busio
import adafruit_vl53l0x
i2c = busio.I2C(board.SCL, board.SDA)
tof = adafruit_vl53l0x.VL53L0X(i2c)
distancia_mm=tof.range #distancia en mm
```

12. Tiempos de ejecución en la Raspberry Pi

Cuadro 2: Tiempos de ejecución en la Raspberry

Código	**Tiempo de ejecución [s]**
Disparidad	2.52
Detección de línea	0.038
Segmentación por	
CANNY	0.134
LAPLACE	0.308
SOBEL	0.196
SCHARR	0.202
HPF	0.541
FFT	0.419

13. Conclusiones

Gracias a que se pudo detectar la profundidad del carril y además, de manera satisfactoria, se pudo leer y representar en una imagen el carril identificado, se pudo implementar un software que permitiría a un vehículo móvil desplazarse en un carril definido, y detectar el riesgo de un obstáculo. Sin embargo, estas dos funciones no se implementan de forma simultanea sino que funcionan de forma independiente debido a la insuficiencia de procesamiento y memoria del hardware utilizado.

13.1. Recuperación de imágenes

La recuperación de imágenes aunque correcta es conveniente indagar más dentro de las imágenes recuperadas sin conversión al formato RGB como una posibilidad de recuperar imágenes que aprovechen la corrección de los niveles de luz que la cámara realiza de manera independiente. Los fps a los que trabaja la cámara no son susceptibles a mejora, ya que dependen principalmente del hardware sobre el cual se toman las imágenes con la cámara, siendo esto un primer inconveniente de la Raspberry Pi para esta tarea.

13.2. Calibración y Rectificación

El uso de Python para la calibración de cámaras estéreo no es de suficiente fiabilidad, para esto es mejor hacer uso de herramientas como MATLAB como se pudo evidenciar, y aunque la calibración realizada en este trabajo está hecha en MATLAB y los valores intrínsecos copiados para utilizarlos en la rectificación mediante Python, es ideal reconocer las matrices o vectores que almacenan los puntos identificados del tablero de ajedrez para agregarlos a Python y con esto hacer coincidir aun más el resultado de la calibración forzada en Python con la calibración por MATLAB, ya que como se mencionó, la calibración final en Python no coincidía totalmente con la de MATLAB.

13.3. Algoritmo SGBM

El rango de medición que se obtiene cuando sobre el mapa de disparidad no se aplica el filtro WLS es el mayor posible(desde 30cm hasta distancias superiores a los 140cm), sin embargo el resultado de esto puede impedir el correcto funcionamiento de la segmentación posterior, dado que la información obtenida es más difícil de interpretar, por lo que el uso de esto no es conveniente.

El uso del filtro WLS sobre el mapa de disparidad aumenta la pendiente de las curvas de distancia contra disparidad$^{-}1$, y hace que el resultado sea más susceptible a los niveles de intensidad de luz, además que puede generar ciertas variaciones en los valores de disparidad cuando la superficie de un objeto es blanca o negra, que pese a no tener suficientes datos para generar una gráfica de la cual se pueda apreciar dicha variación, durante las mediciones en los diferentes montajes fue algo a tener en cuenta, sobre todo cuando estas superficies de color blanco y negro se encuentran más cerca de la cámara. También es necesario mencionar que el uso del filtro limita el rango de medición posible, limitándolo de 30cm hasta máximo 80cm, para distancias superiores los valores de disparidad empiezan a repetirse u oscilar alrededor del valor de disparidad para poco mas de los 80cm.

Dada la sensibilidad de los mapas de disparidad a la iluminación, a menos que se consiga un adecuado histograma de referencia en base al histograma de las imágenes recuperadas en cada escenario nuevo que se presente, hay que considerar esta variación para cada valor de disparidad obtenido.

Los métodos de ecualización de histograma, máscara sobre las imágenes e histograma de referencia aunque pueden llegar a mejorar en ciertos valores a las curvas antes mencionadas, incrementan las oscilaciones o ruido en el mapa de disparidad en comparación cuando solo se hace uso del filtro WLS, así que no es necesariamente lo más conveniente para el mejoramiento de estas curvas.

Por último, la ejecución del código del tiempo real contra a ejecución de este sobre una pareja de imágenes no entrega el mismo resultado, donde el ejecutarlo en tiempo real puede llevar a extender el rango de medición una vez mas hasta los 140cm. Se asume que esto es debido a que el algoritmo tarda algunos ciclos de ejecución para estabilizar su funcionamiento.

El uso de la Raspberry Pi 3 B+ es inviable al considerar la velocidad del vehículo dado el excesivo tiempo de ejecución para el cálculo de disparidad, como se presenta en la tabla 2, por lo que es necesario emplear otro hardware para el funcionamiento de la visión del vehículo autónomo.

13.4. Segmentación

Los diferentes métodos para generar la máscara para el algoritmo de watershed, aunque todos casi igual de funcionales, los más eficaces por calidad de resultados y tiempos de ejecución fueron Canny, Sobel y Scharr.

Dos detalles importantes a tener en cuenta de las imágenes de disparidad sobre las cuales se aplica la segmentación son:

- Las superficies brillantes como baldosas, pantallas, vidrios, etc., generan oscilaciones en el mapa de disparidad, por lo que esto confunde al algoritmo de segmentación, generando errores como

la figura de pico en la parte inferior de cada uno de los resultados de segmentación, sección 9.3.

- El algoritmo SGBM genera pequeñas lineas verticales en cambios de superficies, generando ruidos indeseados para la segmentación, lo que puede terminar en nuevos segmentos falsos o partiendo un segmento en varios segmentos.

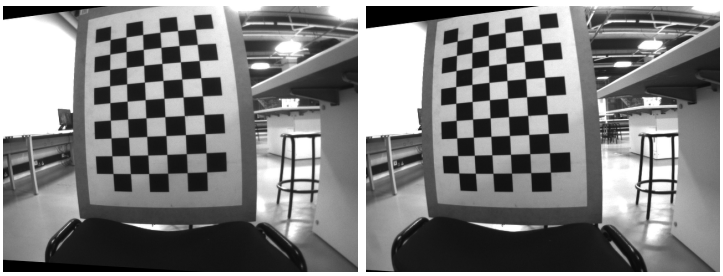
13.5. Identificación de carril

Ya que algunos datos usados para la identificación del carril son forzados y no adaptables a las circunstancias, como son la máscara para la región de interés, la separación de cala lado del carril por su pendiente y la distancia hasta la cual se generan las lineas identificadas, el carril obtenido se vuelve muy susceptible a fallar en curvas pronunciadas, por lo que es necesario hacer correcciones objetivas como estimar la distancia máxima de confianza a la que se reconoce un carril, un agrupamiento de las lineas por su cercanía geométrica y no por el signo de su pendiente y por último una forma de recuperar solo las lineas del carril que no sea mediante una máscara fija, sino de preferencia un método de inundación en la región intermedia entre ambas lineas del carril.

También es conveniente modificar a futuro que la identificación del carril no corresponda con lineas rectas sino con curvas polinomiales, siendo así la obtención del carril mas acertada a la realidad, permitiendo así también captar una mayor longitud del carril mismo.

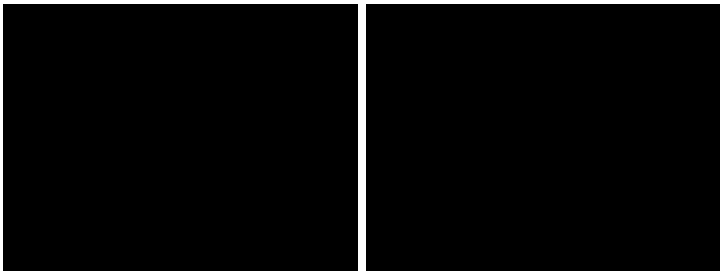
14. Anexos

14.1. Calibración Estéreo, coeficientes de distorsión



(a) Lente izquierdo (b) Lente derecho

Figura 35: Rectificación sin coeficientes de distorsión



(a) Lente izquierdo (b) Lente derecho

Figura 36: Rectificación con 1 coeficiente de distorsión radial, mismo resultado que con 3 coeficientes de distorsión radial y 2 tangencial

(a) Lente izquierdo (b) Lente derecho

Figura 37: Rectificación con 2 coeficientes de distorsión radial

(a) Lente izquierdo (b) Lente derecho

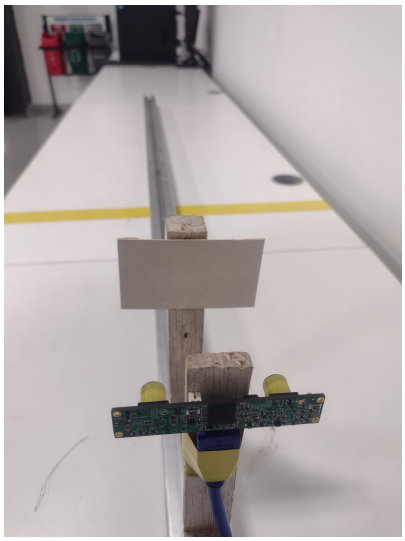
Figura 38: Rectificación con 3 coeficientes de distorsión radial

(a) Lente izquierdo (b) Lente derecho

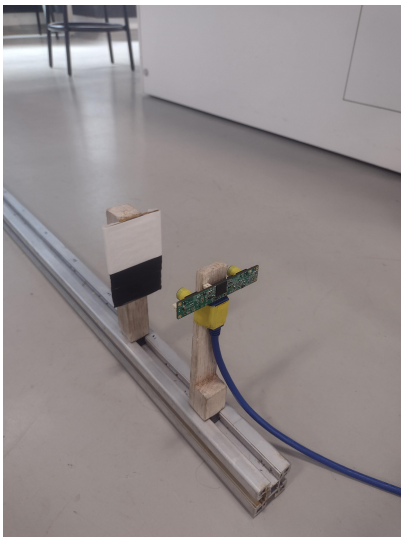
Figura 39: Rectificación con 3 coeficientes de distorsión radial y 1 tangencial

15. Montaje

Los siguientes son algunos de los montajes realizado para la obtención de curvas de disparidad y profundidad.



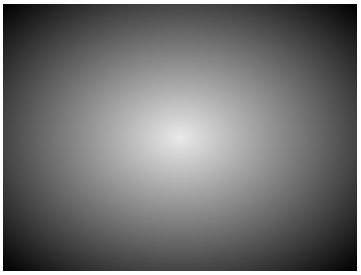
(a) Lente izquierdo



(b) Lente derecho

Figura 40: Montaje realizado para toma de datos de disparidad vs distancia

15.1. Disparidad con SGBM

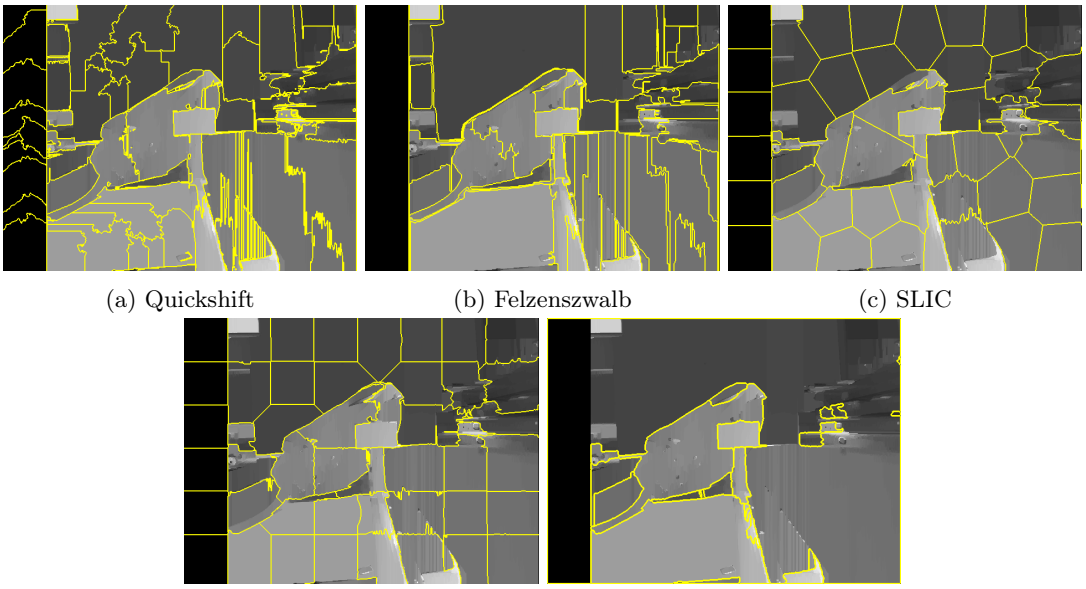


(a) máscara para imágenes

(b) Histograma de referencia

Figura 41: En la figura (a) se observa la máscara empleada en la Figura 21 (d), y en la figura (b) se presenta la imagen 'Cones' del set de middlebury [1], usado como histograma de referencia para la Figura 21 (e)

15.2. Segmentación



(a) Quickshift (b) Felzenszwalb (c) SLIC

(d) Watershed (e) Watershed de OpenCV

Figura 42: Resultados de los diferentes métodos de segmentación, de la figura (a) a la (d) son métodos de la librería scikit-image.

Referencias

[1] URL: https://vision.middlebury.edu/stereo/data/scenes2003/.

[2] *16-385 Computer Vision, Spring 2020*. Lecture no.10 [Online; accessed 3. Oct. 2022]. Oct. de 2022. URL: https://www.cs.cmu.edu/~16385.

[3] *16-385 Computer Vision, Spring 2020*. Lecture no.13 [Online; accessed 3. Oct. 2022]. Oct. de 2022. URL: https://www.cs.cmu.edu/~16385.

[4] *Comparison of segmentation and superpixel algorithms — skimage v0.19.2 docs*. [Online; accessed 13. Dec. 2022]. Dic. de 2022. URL: https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_segmentations.html.

[5] Alec Delaney. *Adafruit_CircuitPython_VL53L0X*. URL: https://github.com/adafruit/Adafruit_CircuitPython_VL53L0X.

[6] R. I. Hartley y A. Zisserman. Second. 2004. Cap. 9.

[7] *IMAGE SEGMENTATION AND MATHEMATICAL MORPHOLOGY*. URL: https://people.cmm.minesparis.psl.eu/users/beucher/wtshed.html.

[8] Sadekar Kaustubh. *Depth Estimation using Stereo Camera and OpenCV (Python/C++)*. [Online; accessed 1. Oct. 2022]. Jul. de 2022. URL: https://learnopencv.com/depth-perception-using-stereo-camera-python-c.

[9] Sadekar Kaustubh. *Understanding Lens Distortion | LearnOpenCV #*. [Online; accessed 1. Oct. 2022]. Mayo de 2021. URL: https://learnopencv.com/understanding-lens-distortion.

[10] M. LeBlanc-Williams. «CircuitPython on Linux and Raspberry Pi». En: *Adafruit Learning System* (jun. de 2018). URL: https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/installing-circuitpython-on-raspberry-pi.

[11] *Li-USB30-V024STEREO*. Abr. de 2021. URL: https://www.leopardimaging.com/product/3d-stereo-cameras/li-usb30-v024stereo/.

[12] *Piwheels*. URL: https://piwheels.org/project/opencv-contrib-python/.

[13] Simon J. D. Prince. *Computer vision: models, learning, and inference*. New York: Cambridge University Press, 2012. Cap. 14. ISBN: 9781107011793.

[14] Mallick Satya. *Introduction to Epipolar Geometry and Stereo Vision | LearnOpenCV #*. [Online; accessed 1. Oct. 2022]. Mayo de 2021. URL: https://learnopencv.com/introduction-to-epipolar-geometry-and-stereo-vision.

[15] Vishwesh Shrimali. *Install opencv 4 on raspberry pi*. Abr. de 2021. URL: https://learnopencv.com/install-opencv-4-on-raspberry-pi/.

[16] Anne Solverg. *INF 4300 – Hough transform*. URL: https://www.uio.no/studier/emner/matnat/ifi/INF4300/h09/undervisningsmateriale/hough09.pdf.

[17] David Tian. *DeepPiCar-part 3 make picar see and think*. Mar. de 2020. URL: https://towardsdatascience.com/deeppicar-part-3-d648b76fc0be.

[18] 深蓝. *Linux - raspberry pi和opencv无法安装libhdf5-100(raspberry pi and opencv cant install libhdf5-100)*. Mar. de 2021. URL: https://tousu.in/?qa=153250%5C%2F#a153251.

ANEXO 1
AUTORIZACIÓN DE PUBLICACIÓN DE DOCUMENTOS EN EL REPOSITORIO COLECCIONES DIGITALES DE LA ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

Fecha 12 | 1 | 2023

1. Datos de la publicación (trabajo de grado, artículo, video, conferencia, libro, imagen, fotografía, audio, presentación u otros) y del (los) autor(es)

Documento de Identidad		Apellidos	Nombres	Correo electrónico
Tipo	Número			
CC	1007855511	TORRES CASTILL	CHRISTIAN CAMI	christian.torres-c@mail.escue

Título del Documento	Procesamiento de imágenes y algoritmos para la visión de un vehículo autónomo a escala, mediante una cámara estereoscópica
Nombre del evento origen (si aplica)	
Fecha del evento (si aplica)	
Palabras claves	visión estéreo, mapa de disparidad, segmentación, Watershed, Hough, detección de carril, Python

Acuerdos de confidencialidad: **No Tiene Acuerdo(s)** ☑ **Tiene Acuerdo(s)** ☐
(Si tiene acuerdos de confidencialidad, por favor diligencie el siguiente cuadro)

Persona jurídica o natural	Desde			Hasta		
	DD	MM	AAAA	DD	MM	AAAA

ANEXO 1
AUTORIZACIÓN DE PUBLICACIÓN DE DOCUMENTOS EN EL REPOSITORIO COLECCIONES DIGITALES DE LA ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

2. Autorización de publicación de documentos en el Repositorio Institucional

Autorizo a la Escuela Colombiana de Ingeniería Julio Garavito para publicar el trabajo de grado, artículo, video, conferencia, libro, imagen, fotografía, audio, presentación u otro (en adelante documento) que en la fecha entrego en formato digital, y le permito de forma indefinida que lo publique en el repositorio institucional, en los términos establecidos en la Ley 23 de 1982, la Ley 44 de 1993, y demás leyes y jurisprudencia vigente al respecto, para fines educativos y no lucrativos. Esta autorización es válida para las facultades y derechos de uso sobre la obra en formato digital, electrónico, virtual; y para usos en redes, internet, extranet, y cualquier formato o medio conocido o por conocer.

En mi calidad de autor, expreso que el documento objeto de la presente autorización es original y lo elaboré sin quebrantar ni suplantar los derechos de autor de terceros. Por lo tanto, es de mi exclusiva autoría y, en consecuencia, tengo la titularidad sobre él. En caso de queja o acción por parte de un tercero referente a los derechos de autor sobre el documento en cuestión, asumiré la responsabilidad total y saldré en defensa de los derechos aquí autorizados. Esto significa que, para todos los efectos, la Escuela actúa como un tercero de buena fe.

Toda persona que consulte el Repositorio Institucional de la Escuela, el Catálogo en línea u otro medio electrónico, podrá copiar apartes del texto, con el compromiso de citar siempre la fuente, la cual incluye el título del trabajo y el autor. Esta autorización no implica renuncia a la facultad que tengo de publicar total o parcialmente la obra en otros medios.

Esta autorización está respaldada por las firmas del (los) autor(es) del documento.

Sí autorizo (amos)

3. Firmas de autor(es)

Firma autor 1 Christian C. Torres Castillo Documento de identidad N.°: 1007855511	Firma autor 2 _____ Documento de identidad N.°:
Firma autor 3 _____ Documento de identidad N.°:	Firma autor 4 _____ Documento de identidad N.°:

4. Firmas de aprobación

Director del Trabajo de Grado _____ Documento de identidad N.°: 79'447.648	Director del Programa (Si aplica) _____ Documento de identidad N.°: