Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencias de la Computación
R-521 — Robótica Móvil

Trabajo Práctico

Visión por Computadora: Reconstrucción 3D y Estimación de Pose

1. Introducción

El objetivo del trabajo práctico es la realización de los pasos básicos para poder triangular y proyectar puntos con una cámara estéreo. En este trabajo se debe utilizar las librerías OpenCV[1] y software de calibración ampliamente utilizado en el campo de visión por computadora y robótica.

2. Entrega

- Se debe proveer un repositorio git que contenga el código desarrollado y un archivo `README.md` con las instrucciones de compilación y ejecución. Se recomienda hacer una imagen Docker para facilitar la reproducción de los resuldatos.
- Se debe entregar un informe en Lyx o LaTeX explicando el trabajo realizado y analizando los resultados obtenidos.

3. Datos

Para trabajar se utilizarán los datos provistos por la cátedra. Caso contrario puede hacer uso del dataset EuRoC[2]. Debera convertir la secuencia de calibración a formato ROS2. El resto de las secuencias ya se encuentran en formato ROS2 y pueden ser descargadas de `https://docs.openvins.com/gs-datasets.html`

4. Calibración

Antes de comenzar a trabajar con un dataset se debe realizar una calibración de los sensores. En este trabajo solo deben calibrar una cámara estéreo.

Para la calibración se puede utilizar cualquiera de las aplicaciones que se detallan a continuación:

- ROS2 camera_calibration: `https://navigation.ros.org/tutorials/docs/camera_calibration.html`
- OpenCV tutorial_camera_calibration: `https://docs.opencv.org/4.x/d4/d94/tutorial_camera_calibration.html`
- Kalibr: `https://github.com/ethz-asl/kalibr/wiki/ROS2-Calibration-Using-Kalibr`
- Camera Calibration Toolbox for Matlab: `https://www.cs.toronto.edu/pub/psala/VM/cameraCalibrationExample.html`

Se pide desarrollar un programa que lea un par de imágenes estéreo cualquiera y realice los siguientes pasos:

[1] `https://opencv.org/`

[2] `https://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets`

5. Sincronizar imágenes

Debido a que en el rosbag las imágenes izquierda y derecha se almacenaron en un instante (timestamp en el rosbag) distinto al que fueron capturadas (timestamp en el header de las imágenes). Vamos a tener que sincronizar las imágenes utilizando el timestamp que figura en el header, esto se hace mediante el paquete `message_filters`[3]. Para esto hay que

```
# Importamos del paquete
import message_filters
```

```
# Creamos subscribers a las cámaras izq y der
self.left_rect = message_filters.Subscriber(self, Image, '/left/image_rect')
self.right_rect = message_filters.Subscriber(self, Image, '/right/image_rect')
```

```
# Creamos el objeto ts del tipo TimeSynchronizer encargado de sincronizar los mensajes
    recibidos.
ts = message_filters.TimeSynchronizer([self.left_rect, self.right_rect], 10)
# Registramos un callback para procesar los mensajes sincronizados.
ts.registerCallback(self.callback)
La función callback es un método del nodo con esta asignatura:
def callback(self, left_msg, right_msg):
```

6. Rectificar imágenes

Con los parámetros íntrinsecos y extrínsecos de la cámara estéreo, rectificar las imágenes haciendo:

- Utilizando ROS2: por medio del paquete `stereo_image_proc`

  ```
  ros2 launch stereo_image_proc stereo_image_proc.launch.py
  ```

  Para reproducir el rosbag y remapear los tópicos a los que el paquete `stereo_image_proc` requiere puede hacer:

  ```
  ros2 bag play CARPETA_ROSBAG --remap /stereo/left/image_raw:=/left/image_raw \
  /stereo/left/camera_info:=/left/camera_info \
  /stereo/right/image_raw:=/right/image_raw \
  /stereo/right/camera_info:=/right/camera_info
  ```

- o bien, utilizando la librería OpenCV. Por medio de las funciones: `cv::stereoRectify()`, `cv::initUndistortRectifyMap()` y `remap()`.

7. Extraer Feaures Visuales: Keypoints y descriptores

Selecionar un detector de keypoints (FAST, ORB, SIFT, SURF, GFTT, BRISK, etc.) y un descriptor (BRIEF, ORB, BRISK, etc.), y extraer features en ambas imágenes.

8. Buscar correspondencias visuales

Realizar la búsqueda de correspondencias entre los feature de ambas imágenes (*matching*). Para esto se debe utilizar la función `cv::BFMatcher::BFMatcher()`.

[3] `http://wiki.ros.org/message_filters`

9. Triangular Puntos 3D

Dadas las correspondencias visuales (*matches*) obtenidas en el paso anterior, realizar la triangulación de los features detectados utilizando la función `cv::sfm::triangulatePoints()`. Para la visualización de la nube de puntos 3D se puede publicar un mensaje de tipo `sensor_msgs/PointCloud2`[4] y hacer uso de RViz.

10. Filtrar de Correspondencias Espúreas

Aplicar RANSAC (*Random sample consensus*) para filtrar los matches espúreos y computar la Matríz Fundamental. Para esto puede utilizar la función `cv::findHomography()`. Para verificar el impacto del filtrado, visualizar los matches entre las imágenes nuevamente como en la nube de puntos 3D generada.

11. Computar el Mapa de Disparidad

Computar el mapa de disparidad con las librerías utilizando la función `cv::StereoMatcher::compute()`. Opcionalmente para tener mejores resultados puede utilizar la librería LIBELAS[5]. Visualizar el mapa de disparidad.

12. Reconstruir la Escena 3D de manera Densa

Utilizando el mapa de disparidad obtenido en el paso anterior realizar una reconstrucción densa de la escena observada utilizando la función `cv::reprojectImageTo3D()`. Para esto debe utilizar la matríz de reproyección Q retornada por la función `cv::stereoRectify()`. Visualizar la nube de puntos 3D.

13. Estimación de pose Monocular

Utilizando `cv::recoverPose()` estimar la transformación entre la cámara izquierda y la cámara derecha. Para esto deberá calcular la matríz esencial utilizando la función `cv::findEssentialMat()`. Notar que `cv::recoverPose()` retorna el vector unitario de traslación, por lo tanto deberá multiplicarlo por el factor de escala (en este caso el baseline entre las cámaras) para obtener la traslación estimada. Una vez hecho esto se pide:

- Visualizar la pose estimada de ambas cámaras.
- Estimar y visualizar la trayectoría realizada por la cámara izquierda a un factor de escala cualquiera.

[4] `http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/PointCloud2.html`
[5] `http://www.cvlibs.net/software/libelas/`