MULTIPLES DISPAROS SINCRONIZADOS

Investigacion UHPC

Lukas Wolff Casanova

1 de mayo de 2024

---

1. INTRODUCCION

Posterior a sincronizar las camaras, se determino que la siguiente tarea era lograr sacar 5000 por camara, lo que da un rango de operacion de 25 segundos a 200 fps. Para poder lograrlo se postularon varias ideas como listas concatenadas, determinar el tamaño de la lista o usar el buffer de las camaras.

2. METODOLOGIA

Para comenzar se investigo sobre las listas concatenadas o listas con punteros. La diferencia que tienen con una lista normal es que en vez de estar almacenadas de manera conjunta en la memoria RAM, estas se almacenan los datos de forma particular, junto a un puntero que indica donde esta el siguiente indice, de esta manera se logra que el proceso de almacenar datos sea mucho mas eficiente en terminos de velocidad y memoria, pero tiene como contra que recorrer una lista es mas lento. Tomando a favor la estructura del codigo, donde se recorre la lista una vez que las fotos ya fueron tomadas, se determino que era una buena herraienta a utilizar.

Ademas, se activo el buffer de la camara, el cual de igual manera envia unaq foto por ciclo al computador, pero ayuda a que la camara no se sature.

En primer lugar se crea la clase node, la cual permite crear y recorrer una lista concatenada, la celda mas importante del codigo es la siguiente:

```
# Método para recorrer la lista de nodos
def traverse(self):
    current_node = self.head

    while current_node:

        img = current_node.data.GetArray()
        img = cv2.equalizeHist(img)
        camara = current_node.data.GetCameraContext()
        tiempo = current_node.data.GetTimeStamp()

        # Carpeta donde se almacenarán las imágenes
        folder_path = f'FOTOS/{RPM}/{camara}'
        print(folder_path)

        # Crear la carpeta si no existe
        if not os.path.exists(folder_path):
            os.makedirs(folder_path)
```

```
        # Nombre del archivo de la imagen
        file_name = f'{tiempo}.png'

        # Ruta completa del archivo de la imagen
        file_path = os.path.join(folder_path, file_name)

        # Crear una imagen PIL a partir de la matriz de la imagen
        image = Image.fromarray(img)

        # Guardar la imagen como PNG
        image.save(file_path)

        print(f"Imagen guardada: {file_path}")
        current_node = current_node.next
```

Como se puede ver, todo el proceso que se realizaba anteriormente al recorrer la lista imagenes, ahora se realiza dentro de la classe node, la cual es llamada al final del codigo:

Posteriormente, se setean los ajustes tanto de camaras como fotos:

```
#----------------------
# DETERMINO EL NOMBRE DE CARPETA
RPM = 12055

# DETERMINO CANTIDAD DE IMAGENES A SACAR
countOfImagesToGrab = 5000
#----------------------
```

Donde el directorio final de las fotos sera el siguiente:

FOTOS/RPM/numero-camara/tiempo-foto.jpg

Posteriormente, se configuran las camaras:

```
# Create and attach all Pylon Devices.
for i, cam in enumerate(cameras):

  cam.Attach(tlFactory.CreateDevice(devices[i]))

  # Open the camera to set parameters
  cam.Open()

  # Buffer Size Adjustment
  cam.MaxNumBuffer = buffer  # Adjust buffer size as needed

  # Configure camera settings
  cam.AcquisitionFrameRateEnable.SetValue(True)
  cam.AcquisitionFrameRate.SetValue(200)
  cam.ExposureTime.SetValue(100)

  # Close the camera after setting parameters
  cam.Close()
```

Comienza el trigger:

```
#----------------------
# Disparo las camaras
```

```
cameras.StartGrabbing()
#----------------------
```

Y se almacenan las fotos:

```
for i in range(countOfImagesToGrab):
  if not cameras.IsGrabbing():
      break

  #recibo la foto y la guardo como array y en la lista concatenada
  R = cameras.RetrieveResult(40000, pylon.TimeoutHandling_ThrowException)
  s.add_at_front(R)
```

Finalmente, se recorre la lista:

```
s.traverse()
print("Fotos Guardadas")
```

Enlace a GitHub con toda la información: GIT-HUB-INVESTIGACION-UHPC.

Enlace a GitHub con codigo base: : GIT-HUB-CODIGO-BASE.