imageServer

1.0

Generado por Doxygen 1.8.8

Domingo, 22 de Enero de 2017 13:34:28

Índice general

Capítulo 1

Paquete Computacional "imageServer"

A través de éste paquete se implementa un sistema capaz de capturar, adecuar, enmascarar y transmitir imágenes en tiempo-real de manera simultanea a multiples usuarios.

El paquete se desarrollo con miras de que funciones como una aplicación util para labores de investigación y docencia, y esta enfocádo principalmente a su uso en actividades de teleoperación de robots moviles que realizan tareas en areas delimitadas.

El sistema captura las imágenes, corrige posibles distorsiones lineales provocadas por la orientación de la cámara con respecto al area de exploración y añade información virtual a las imagenes, para simular problemas a resolver.

Capítulo 2

imageServer

An openCV image server, to distribute captured image to multiple clients.

Capítulo 3

Indice de namespaces

3.1. Lista de 'namespaces'

Lista de los 'namespaces', con una breve descripción:

Capítulo 4

Indice jerárquico

4.1. Jerarquía de la clase

Esta lista de herencias esta ordenada aproximadamente por orden alfabético:

Capítulo 5

Índice de clases

5.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

Capítulo 6

Indice de archivos

6.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

Capítulo 7

Documentación de namespaces

7.1. Referencia del Namespace remoteFrame

Clases

- class remoteFrame

7.2. Referencia del Namespace testClient

Variables

- string address = '127.0.0.1'
- tuple argc = len(sys.argv)
- tuple img = rF.getFrame()
- Mask = None
- int port = 8888
- tuple rF = remoteFrame(address, port, Mask)

7.2.1. Documentación de las variables

7.2.1.1. list address = '127.0.0.1'

Definición en la línea 10 del archivo testClient.py.

7.2.1.2. tuple argc = len(sys.argv)

Definición en la línea 13 del archivo testClient.py.

7.2.1.3. tuple img = rF.getFrame()

Definición en la línea 26 del archivo testClient.py.

7.2.1.4. tuple Mask = None

Definición en la línea 11 del archivo testClient.py.

7.2.1.5. tuple port = 8888

Definición en la línea 9 del archivo testClient.py.

7.2.1.6. tuple rF = remoteFrame(address, port, Mask)

Definición en la línea 21 del archivo testClient.py.

Capítulo 8

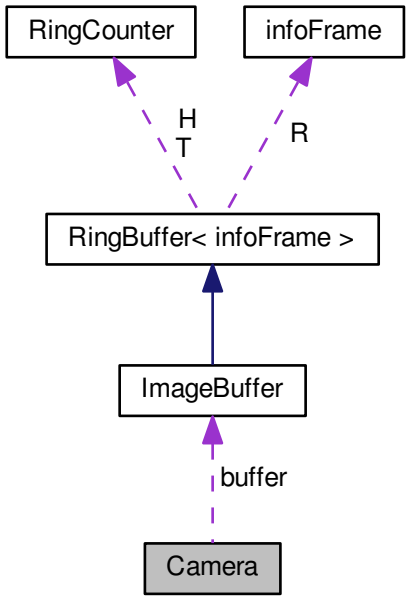
Documentación de las clases

8.1. Referencia de la Clase Camera

Esta clase permite capturar imágenes de una cámara, utilizando los métodos provisto por la biblioteca highgui. Las imagenes capturadas son almacenadas en una cola. Se proveen mecanismos de sincronización para el acceso concurrente a dicha cola.

`#include <Camera.h>`

Diagrama de colaboración para Camera:



Métodos públicos

- Camera (int captureDevice, unsigned long capPer=33333, bool autoCapt=true)

  Constructor de la clase.
- Camera ()

  Constructor de la clase. Abre la cámara 0 por defecto con un peridod de captura d 33 ms e inicial la captura automáticamente.
- ∼Camera ()

  Destructor de la clase.
- int getLastFrame (infoFrame &iF)

  Esta función regresa el ultimo cuadro que se haya insertado al buffer. Esto es, es el cuadro mas reciente, el que se encuentra al final de la cola, no al inicio.
- void startCapture ()

  Inicia la captura de imágenes y su almacenamiento en el buffer.
- void stopCapture ()

  Detiene la captura de imágenes y su almacenamiento en el buffer.

Métodos privados estáticos

- static void ∗ readFramesThread (void ∗ptr)

  Esta es la función principal que es ejecutada como un hebra. Captura los cuadros, del dispositivo de captura y los inserta en un buffer.

Atributos privados

- ImageBuffer buffer

  Cola de imagenes en donde se almacenarán los cuadros capturados.
- VideoCapture cap

  Objeto asociado al dispositivo de captura.
- bool capture

  Bandera que controla cuando se capturan imágenes.
- unsigned long capturePeriod

  Periodo de captura (en microsegundos).
- pthread_t captureThread

  Identificador de la hebra de captura.
- int deviceID

  Identificador del dispositivo de captura.

8.1.1. Descripción detallada

Esta clase permite capturar imágenes de una cámara, utilizando los métodos provisto por la biblioteca highgui. Las imagenes capturadas son almacenadas en una cola. Se proveen mecanismos de sincronización para el acceso concurrente a dicha cola.

Definición en la línea 36 del archivo Camera.h.

8.1.2. Documentación del constructor y destructor

8.1.2.1. Camera (int *captureDevice,* unsigned long *capPer* = 33333, bool *autoCapt* = true)

Constructor de la clase.

Camera (int captureDevice, unsigned long capPer = 33333, bool autoCapt = true);

Parámetros

int	captureDevice Número que indica la cámara que se va a usar para la captura.
unsigned	long capPer Periodo de captura de imagenes: por default el valoe es igual 33 ms (lo que equivale a una frecuencia de 30 fps).
bool	autoCapt Bandera que se utiliza para indicar si se deben inicial la captura de manera automñatica: true => si, false => no.

Definición en la línea 13 del archivo Camera.cpp.

```
14 {
15          // Set device ID
16          deviceID = captureDevice;
17
18          // Init buffer
19          buffer.setBufferSize (5);
20
21          capture = autoCapt;
22
23          capturePeriod = capPer;
24
25          // init capture device
26          cap.open (deviceID);
27          if (!cap.isOpened ())
28          {
29                          perror ("Camera::Failed to initialize video capture!");
30                          return;
31          }
32
33          //Launch capture thread.
34          if (pthread_create (&captureThread, NULL,
      Camera::readFramesThread, this))
35          {
36             perror ("Camera::Couldn't start capture thread.");
37             return;
38          }
39
40          // Start capture
41          capture = autoCapt;
42
43 }
```

8.1.2.2. Camera ( )

Constructor de la clase. Abre la cámara 0 por defecto con un peridod de captura d 33 ms e inicial la captura automáticamente.

Camera ()

Definición en la línea 8 del archivo Camera.cpp.

```
9 {
10          Camera (0);
11 }
```

8.1.2.3. ~Camera ( )

Destructor de la clase.

~Camera()

Definición en la línea 45 del archivo Camera.cpp.

```
46 {
47      void *ret;
48
49      for (int cont = 0; pthread_cancel(captureThread)!=0 && cont < 10; cont++)
50      {
51         perror ("Camera::~Camera: Couldn't finish capture thread on exit");
52         usleep(100000);
53      }
54      if (pthread_join(captureThread, &ret) != 0)
55         perror ("Camera::~Camera: Couldn't joint capture thread");
56          cap.release ();
57 }
```

8.1.3. Documentación de las funciones miembro

8.1.3.1. int getLastFrame (infoFrame & *iF*)

Esta función regresa el ultimo cuadro que se haya insertado al buffer. Esto es, es el cuadro mas reciente, el que se encuentra al final de la cola, no al inicio.

int getLastFrame (infoFrame &iF);

Parámetros

infoFrame	&if objeto de tipo infoFrame en donde se copiara el cuadro al final del buffer, junto con el tiempo de captura.

Devuelve

la función regresa 0 en case de exito y -1 en caso de que el Frame este vacio.

Definición en la línea 78 del archivo Camera.cpp.

```
79 {
80          buffer.getLast (iF);
81
82          if (iF.frame.empty ())
83             return -1;
84          else
85             return 0;
86 }
```

8.1.3.2. void * readFramesThread (void * *ptr*) [static], [private]

Esta es la función principal que es ejecutada como un hebra. Captura los cuadros, del dispositivo de captura y los inserta en un buffer.

static void *readFramesThread (void *ptr)

Parámetros

De	acuerdo al estandar definido por pthread_create, recibe un apuntador a void, pero este es interpretado por la funcíon como un apuntador a

Definición en la línea 59 del archivo Camera.cpp.

```
60 {
61          Camera *ptr = (Camera *) camera_ptr;
```

```
62          infoFrame frame;
63          do
64          {
65             if (ptr->capture)
66                 {
67                         ptr->cap » frame.frame; // get a new frame from camera
68
69                         ptr->buffer.advHead ();
70                         ptr->buffer.Queue (frame);   // queue frame into buffer
71             }
72             usleep (ptr->capturePeriod);
73          }
74          while (true);
75 }
```

8.1.3.3. void startCapture ( ) [inline]

Inicia la captura de imágenes y su almacenamiento en el buffer.

void startCapture();

Definición en la línea 87 del archivo Camera.h.

```
87 {capture = true;}
```

8.1.3.4. void stopCapture ( ) [inline]

Detiene la captura de imágenes y su almacenamiento en el buffer.

void stopCapture();

Definición en la línea 93 del archivo Camera.h.

```
93 {capture = false;}
```

8.1.4. Documentación de los datos miembro

8.1.4.1. ImageBuffer buffer [private]

Cola de imagenes en donde se almacenarán los cuadros capturados.

Definición en la línea 42 del archivo Camera.h.

8.1.4.2. VideoCapture cap [private]

Objeto asociado al dispositivo de captura.

Definición en la línea 39 del archivo Camera.h.

8.1.4.3. bool capture [private]

Bandera que controla cuando se capturan imágenes.

Definición en la línea 44 del archivo Camera.h.

8.1.4.4. unsigned long capturePeriod [private]

Periodo de captura (en microsegundos).

Definición en la línea 43 del archivo Camera.h.

8.1.4.5. pthread_t captureThread [private]

Identificador de la hebra de captura.

Definición en la línea 40 del archivo Camera.h.

8.1.4.6. int deviceID [private]

Identificador del dispositivo de captura.

Definición en la línea 38 del archivo Camera.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- include/Camera.h
- src/Camera.cpp

8.2. Referencia de la Clase Client

Esta clase crea un objeto que permite conectarse via un socket con un servidor, para transmitir información.

`#include <Client.h>`

Métodos públicos

- Client (int port, const char *address, int msgDS=MAXDATASIZE)
- ~Client ()

  Destructor de la clase.
- void connectSocket ()

  Realiza la conexion del objeto cliente con el servidor. En caso de falla, se invoca exit(1).
- int getFrame (Mat &out)

  Método que recibe una imagen del servidor.

Métodos privados

- void configure ()

  metodo que configura el socket. Se crea el socket y se inicializa la estructura que contiene la dirección a conectarse.

Atributos privados

- char address [256]

  Cadena de caracteres que contiene la dirección a la cual conectarse.
- int cfd

El descriptor del conector.

- struct sockaddr_in client

  Estructura en donde se almacena la direccion a conectarse.

- unsigned char * msgData

  Apuntador al buffer de datos a enviar y/o recibir.

- long int msgDataSize

  Tamaño del bufer de datos.

- int port

  El puerto al cual el cliente va a conectarse.

8.2.1. Descripción detallada

Esta clase crea un objeto que permite conectarse via un socket con un servidor, para transmitir información.

Definición en la línea 34 del archivo Client.h.

8.2.2. Documentación del constructor y destructor

8.2.2.1. Client (int *port*, const char * *address*, int *msgDS* = MAXDATASIZE)

Definición en la línea 8 del archivo Client.cpp.

```
9 {
10          this->port = port;
11          strncpy (this->address, address, 255);
12          msgDataSize = msgDS;
13          msgData = new unsigned char[msgDataSize];
14          if (!msgData)
15          {
16                          std::cerr « "Error Client Constructor: Couln not allocate enough"
17                                  « " memory for image buffer." « std::endl;
18                          return;
19          }
20
21          configure ();
22 }
```

8.2.2.2. ~Client ( )

Destructor de la clase.

Definición en la línea 24 del archivo Client.cpp.

```
25 {
26          if (msgData)
27                          delete[]msgData;
28 }
```

8.2.3. Documentación de las funciones miembro

8.2.3.1. void configure ( ) [private]

metodo que configura el socket. Se crea el socket y se inicializa la estructura que contiene la dirección a conectarse.

Definición en la línea 30 del archivo Client.cpp.

```
31 {
32     struct hostent *hp;
33 
34      if ((cfd = socket (AF_INET, SOCK_STREAM, 0)) < 0)
35          {
36                          cerr « "Falla en el socket cliente" « errno « endl;
37                          perror ("conectar ");
38                          exit (1);
39          }
40 
41          client.sin_port = htons (port);
42          client.sin_family = AF_INET;
43 
44          fprintf (stderr, "conectando: %s:%d\n", address, ntohs (client.sin_port));
45 
46          hp = gethostbyname (address);
47          if (hp == NULL)
48          {
49                          printf ("fallo gethostbyname %d\n", errno);
50                          perror ("Socket cliente");
51                          close (cfd);
52                          exit (1);
53          }
54 
55          memcpy (&client.sin_addr.s_addr, hp->h_addr_list[0], hp->h_length);
56 }
```

8.2.3.2. void connectSocket ( )

Realiza la conexion del objeto cliente con el servidor. En caso de falla, se invoca exit(1).

Definición en la línea 58 del archivo Client.cpp.

```
59 {
60          if ((connect (cfd, (struct sockaddr *) &client, sizeof (client))) < 0)
61          {
62                          printf ("Falla del cliente %d\n", errno);
63                          perror ("cliente");
64                          close (cfd);
65                          exit (1);
66          }
67 }
```

8.2.3.3. int getFrame (Mat & *out*)

Método que recibe una imagen del servidor.

Devuelve

Regresa SEND_SUCCESS en caso de exito, y SEND_FAILURE en caso contrario.

Definición en la línea 71 del archivo Client.cpp.

```
72 {
73          unsigned char msg[MSG_LENGTH];
74          struct ImageInfo imgInfo;
75 
76          strncpy ((char *) msg, "IMG", MSG_LENGTH);
77          if (Write (cfd, MSG_LENGTH, msg) < 0)
78                          return SEND_FAILURE;
79 
80          Read (cfd, sizeof (struct ImageInfo), (unsigned char *) &imgInfo);
81 
82          if (imgInfo.size > msgDataSize)
83          {
84                          cerr « "error in Client::getFrame" « endl « endl
85                                  « "Image sent size exceeds the buffer size" « endl;
86                          return SEND_FAILURE;
```

```
87          }
88          // Read image data
89          if (Read (cfd, imgInfo.size, msgData) < 0)
90                      return SEND_FAILURE;
91
92          Mat img (imgInfo.rows, imgInfo.cols, imgInfo.type, (void *) msgData);
93          out = img.clone ();
94
95
96          return SEND_SUCCESS;
97 }
```

8.2.4. Documentación de los datos miembro

8.2.4.1. char address[256] [private]

Cadena de caracteres que contiene la dirección a la cual conectarse.

Definición en la línea 36 del archivo Client.h.

8.2.4.2. int cfd [private]

El descriptor del conector.

Definición en la línea 38 del archivo Client.h.

8.2.4.3. struct sockaddr_in client [private]

Estructura en donde se almacena la direccion a conectarse.

Definición en la línea 39 del archivo Client.h.

8.2.4.4. unsigned char* msgData [private]

Apuntador al buffer de datos a enviar y/o recibir.

Definición en la línea 40 del archivo Client.h.

8.2.4.5. long int msgDataSize [private]

Tamaño del bufer de datos.

Definición en la línea 41 del archivo Client.h.

8.2.4.6. int port [private]

El puerto al cual el cliente va a conectarse.

Definición en la línea 37 del archivo Client.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- include/Client.h
- src/Client.cpp

8.3. Referencia de la Clase ConnServer

Esta clase crea un "stream socket" y lo asocia a una direccion IP y un puerto, y provee servicios básicos para atender de manera concurrente a multiples clientes.

`#include <ConnServer.h>`

Métodos públicos

- ConnServer (int pt, const char *ipa, void *(*connHndlr)(void *), int mxConn)

  Constructor de la clase. Crea un socket y lo asocia con la direccion y el puerto que se le pasa por parametro, e incializa la estructuras necesarias para atender hasta mxConn conexiones.
- ~ConnServer ()

  Destructor de la clase.
- void acceptConnections ()

  Este método se invoca cuando el sistema está listo para recibir conexiones. Mientras accCon sea verdadero y el numero de conexiones actuales sea menor que maxConnections, se esperará solicitudes de conexion, y cuando ésto ocurra, se creará una hebra que ejecute connection_handler.

Atributos privados

- bool accCon

  Bandera que se va a utilizar para controla cuando el servidor acepta conexiones.
- int * connData

  Datos que se comparten a cada instancia que atiende al cliente.
- void *(* connection_handler)(void *)

  Funcion que se invoca para atender cada conexión.
- string ipAddress

  Cadena que contienen la direccion ip que se va a asociar al servidor.
- int maxConnections

  Número máximo de conexiones permitidas.
- int nConnections

  Número de conecciones realizadas.
- int port

  Puerto que se va usar para la comunicación.
- struct sockaddr_in server

  Estructura en donde se almacena la informacion de conexión del servidor.
- int socketDesc

  descriptor del socket.
- pthread_t * thrIds

  Apuntador a arreglo que contiene los identificadores de de cada hebra en ejecución.

8.3.1. Descripción detallada

Esta clase crea un "stream socket" y lo asocia a una direccion IP y un puerto, y provee servicios básicos para atender de manera concurrente a multiples clientes.

Definición en la línea 26 del archivo ConnServer.h.

8.3.2. Documentación del constructor y destructor

8.3.2.1. ConnServer (int *pt,* const char * *ipa,* void *(*)(void *) *connHndlr,* int *mxConn*)

Constructor de la clase. Crea un socket y lo asocia con la direccion y el puerto que se le pasa por parametro, e incializa la estructuras necesarias para atender hasta mxConn conexiones.

Parámetros

int	pt Puerto que se va a usar para el socket.
const	char *ipa Direccion IP (IPV4) que se va a asociar al socket).
void	*(*connHndlr) (void *) Función que se va a invocar para atender cada conexión.
int	mxConn Número máximo de conexiones permitidas.

Definición en la línea 9 del archivo ConnServer.cpp.

```
11 {
12         accCon = false;
13         connection_handler = connHndlr;
14         socketDesc = socket (AF_INET, SOCK_STREAM, 0);
15         maxConnections = mxConn;
16         nConnections = 0;
17         if (socketDesc == -1)
18         {
19              perror ("ConnServer Constructor: Creating socket endpoint\n");
20              return;
21         }
22         port = pt;
23         server.sin_family = AF_INET;
24         if (ipa[0] == '\0')
25         {
26              server.sin_addr.s_addr = INADDR_ANY;
27              ipAddress = string ("Anny");
28         }
29         else
30         {
31              ipAddress = string (ipa);
32
33              if (!inet_aton (ipAddress.c_str (), &server.sin_addr))
34              {
35                   cerr « "conServer Constructor: invalid IP address: " «
     ipAddress
36                        « endl;
37                   return;
38              }
39         }
40         server.sin_port = htons (port);
41
42         if (bind (socketDesc, (struct sockaddr *) &server, sizeof (
     server)) <
43                 0)
44         {
45              perror ("ConnServer Constructor: binding socket endpoint\n");
46              return;
47         }
48
49         if (listen (socketDesc, BACKLOG) == -1)
50         {
51              perror ("ConnServer Constructor: error on listen.");
52              return;
53         }
54         accCon = true;
55         connData = new int[maxConnections];
56         thrIds = new pthread_t[maxConnections];
57 }
```

8.3.2.2. ~ConnServer ( )

Destructor de la clase.

Definición en la línea 59 del archivo ConnServer.cpp.

```
60 {
61     if (connData)
62         delete[] connData;
63     if (thrIds)
64         delete[] thrIds;
65 }
```

8.3.3. Documentación de las funciones miembro

8.3.3.1. void acceptConnections ( )

Este método se invoca cuando el sistema está listo para recibir conexiones. Mientras accCon sea verdadero y el numero de conexiones actuales sea menor que maxConnections, se esperará solicitudes de conexion, y cuando ésto ocurra, se creará una hebra que ejecute connection_handler.

Definición en la línea 68 del archivo ConnServer.cpp.

```
69 {
70         int clntSocket, sockAddInSize = sizeof (struct sockaddr_in);
71         struct sockaddr_in client;
72 
73         while (accCon)
74         {
75              if (nConnections < maxConnections)
76              {
77                     clntSocket =
78                          accept (socketDesc, (struct sockaddr *) &client,
79                                                (socklen_t *) & sockAddInSize);
80                     if (clntSocket == -1)
81                     {
82                          perror ("ConnServer:acceptConnections");
83                          accCon = false;
84                     }
85                     else
86                     {
87                          connData[nConnections] = clntSocket;
88                          if (pthread_create
89                                  (&(thrIds[nConnections]), NULL,
     connection_handler,
90                                      (void *) &(connData[nConnections])))
91                          {
92                                 perror ("No se pudo crear el hilo");
93                                 return;
94 
95                          }
96                          else
97                          {
98                                 nConnections++;
99                                 cout « "Se acaba de hacer una nueva conexión: "
100                                      « nConnections « endl;
101                                 cout.flush();
102                          }
103                    }
104             }
105         }
106 }
```

8.3.4. Documentación de los datos miembro

8.3.4.1. bool accCon [private]

Bandera que se va a utilizar para controla cuando el servidor acepta conexiones.

Definición en la línea 35 del archivo ConnServer.h.

8.3.4.2. int* connData [private]

Datos que se comparten a cada instancia que atiende al cliente.

Definición en la línea 37 del archivo ConnServer.h.

8.3.4.3. void*(* connection_handler)(void *) [private]

Funcion que se invoca para atender cada conexión.

Definición en la línea 36 del archivo ConnServer.h.

8.3.4.4. string ipAddress [private]

Cadena que contienen la direccion ip que se va a asociar al servidor.

Definición en la línea 34 del archivo ConnServer.h.

8.3.4.5. int maxConnections [private]

Número máximo de conexiones permitidas.

Definición en la línea 29 del archivo ConnServer.h.

8.3.4.6. int nConnections [private]

Número de conecciones realizadas.

Definición en la línea 30 del archivo ConnServer.h.

8.3.4.7. int port [private]

Puerto que se va usar para la comunicación.

Definición en la línea 32 del archivo ConnServer.h.

8.3.4.8. struct sockaddr_in server [private]

Estructura en donde se almacena la informacion de conexión del servidor.

Definición en la línea 33 del archivo ConnServer.h.

8.3.4.9. int socketDesc [private]

descriptor del socket.

Definición en la línea 31 del archivo ConnServer.h.

8.3.4.10. pthread_t* thrIds [private]

Apuntador a arreglo que contiene los identificadores de de cada hebra en ejecución.

Definición en la línea 38 del archivo ConnServer.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- include/ConnServer.h
- src/ConnServer.cpp

8.4. Referencia de la Estructura cornerData

Esta estructura contiene información de esquinas que se utiliza durante la captura de posiciones en la pantalla usando el raton.

Métodos públicos

- cornerData ()

  Constructor del objeto.

Atributos públicos

- int cont

  Contador.
- Point2f crn [4]

  Arreglo de 4 objetos tipo Point2f en donde se almacenaran las esquinas.

8.4.1. Descripción detallada

Esta estructura contiene información de esquinas que se utiliza durante la captura de posiciones en la pantalla usando el raton.

Definición en la línea 33 del archivo imgServer.cpp.

8.4.2. Documentación del constructor y destructor

8.4.2.1. cornerData ( ) [inline]

Constructor del objeto.

Definición en la línea 42 del archivo imgServer.cpp.

```
43     {
44         cont = 0;
45     }
```

8.4.3. Documentación de los datos miembro

8.4.3.1. int cont

Contador.

Definición en la línea 36 del archivo imgServer.cpp.

8.4.3.2. Point2f crn[4]

Arreglo de 4 objetos tipo Point2f en donde se almacenaran las esquinas.

Definición en la línea 35 del archivo imgServer.cpp.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- imgServer.cpp

8.5. Referencia de la Clase ImageBuffer

Esta clase especializa la clase Ringbuffer para operar on objetos de tipo infoFrame. Aparte añade dos métodos nuevos: getLast y advHead.

`#include <imageBuffer.h>`

Diagrama de herencias de ImageBuffer

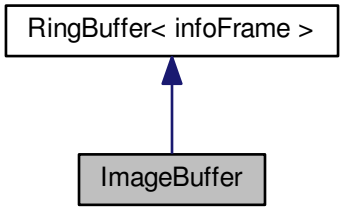
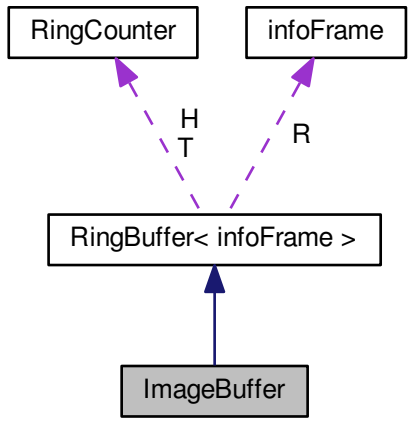


Diagrama de colaboración para ImageBuffer:



Métodos públicos

- ImageBuffer ()

  Constructor del objeto por defecto. Inicializa un una cola de objeto infoFrame con cero objetos.
- int advHead ()

  Método que avanza la cabeza de la cola.
- int Dequeue (infoFrame &e)
- int getLast (infoFrame &e)

  Método que saca un elemento del final de la cola.
- int getSize ()

  Método que regresa el tamaño del buffer.
- int Queue (const infoFrame &e)

  Método virtual que inserta un elemento en la cola.
- void setBufferSize (int N)

  Define el tamaño del buffer a N elementos.

Otros miembros heredados

8.5.1. Descripción detallada

Esta clase especializa la clase Ringbuffer para operar on objetos de tipo infoFrame. Aparte añade dos métodos nuevos: getLast y advHead.

Definición en la línea 15 del archivo imageBuffer.h.

8.5.2. Documentación del constructor y destructor

8.5.2.1. ImageBuffer ( ) [inline]

Constructor del objeto por defecto. Inicializa un una cola de objeto infoFrame con cero objetos.

Definición en la línea 23 del archivo imageBuffer.h.

```
23                          :RingBuffer < infoFrame > (0)
24          {
25          }
```

8.5.3. Documentación de las funciones miembro

8.5.3.1. int advHead ( ) [inline]

Método que avanza la cabeza de la cola.

Devuelve

Regresa 0 en caso de éxito y -1 en caso de la cola este vacia.

Definición en la línea 105 del archivo imageBuffer.h.

```
106         {
107                         pthread_mutex_lock (&this->RB_mutex);
108                         if (this->H != this->T)
109                         {
110                                 this->H++;
111                                 pthread_mutex_unlock (&this->RB_mutex);
112                                 return 0;
113                         }
114                         pthread_mutex_unlock (&this->RB_mutex);
115                         return -1;
116         }
```

8.5.3.2. int Dequeue (infoFrame & *e*) [inline], [virtual]

Reimplementado de RingBuffer< infoFrame >.

Definición en la línea 76 del archivo imageBuffer.h.

```
77          {
78                          return RingBuffer < infoFrame >::Dequeue (e);
79          }
```

8.5.3.3. int getLast (infoFrame & *e*) [inline]

Método que saca un elemento del final de la cola.

Parámetros

cont	infoFrame & e Objeto en donde se regresa una copia del objeto que se encuentra al final de la cola.

Devuelve

El método regresa 0 en caso de éxito, y 1 en caso de que la cola se encuentre vacia.

Definición en la línea 87 del archivo imageBuffer.h.

```
88          {
89                          pthread_mutex_lock (&this->RB_mutex);
90                          if (this->T != this->H)
91                          {
92                                  e = this->R[(uint32_t) (this->T - 1)];
93                                  pthread_mutex_unlock (&(this->RB_mutex));
94                                  return 0;
95                          }
96                          pthread_mutex_unlock (&this->RB_mutex);
97                          return -1;
98          }
```

8.5.3.4. int getSize ( ) [inline]

Método que regresa el tamaño del buffer.

Devuelve

Regresa el tamaño del buffer.

Definición en la línea 54 del archivo imageBuffer.h.

```
55          {
56                          return this->RBF_Size;
57          }
```

8.5.3.5. int Queue (const infoFrame & *e*) [inline],[virtual]

Método virtual que inserta un elemento en la cola.

Parámetros

cont	X & e Elemento a ser insertado en la cola.

Devuelve

El método regresa 0 en caso de éxito, y 1 en caso de que la cola se encuentre llena.

Reimplementado de RingBuffer< infoFrame >.

Definición en la línea 65 del archivo imageBuffer.h.

```
66          {
67                          return RingBuffer < infoFrame >::Queue (e);
68          }
```

8.5.3.6. void setBufferSize (int *N*) [inline]

Define el tamaño del buffer a N elementos.

Parámetros

int	N El nuevo tamaño del buffer.

Definición en la línea 32 del archivo imageBuffer.h.

```
33          {
34                          pthread_mutex_lock (&this->RB_mutex);
35                          if (this->R)
36                                  delete[]this->R;
37
38                          this->RBF_Size = N;
39                          if (this->RBF_Size > 1)
40                                  this->R = new infoFrame[this->
      RBF_Size];      //Reserva memoria dinamica para la cola
41                          else
42                                  this->R = 0;
43                          this->H.SetRingSize (this->RBF_Size);
44                          this->T.SetRingSize (this->RBF_Size);
45                          this->H = this->T = 0;
46                          pthread_mutex_unlock (&this->RB_mutex);
47          }
```

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/imageBuffer.h

8.6. Referencia de la Estructura ImageInfo

Esta clase define la estructura ImagenInfo, que define el encabezado que se utiliza para la transmisión de imágenes.

`#include <structures.h>`

Métodos públicos

- ImageInfo ()

  Constructor por defecto.

Atributos públicos

- int cols

  Número de columnas en la imagen.
- int rows

  Número de renglones en la imagen.
- int size

  El tamaño de la imagen.
- int type

  Tipo de la imagen (de acuerdo a la representación usada en openCV).

8.6.1. Descripción detallada

Esta clase define la estructura ImagenInfo, que define el encabezado que se utiliza para la transmisión de imágenes.

Definición en la línea 21 del archivo structures.h.

8.6.2. Documentación del constructor y destructor

8.6.2.1. ImageInfo ( ) [inline]

Constructor por defecto.

Definición en la línea 32 del archivo structures.h.

```
33          {
34             rows = cols = type = size = 0;
35          }
```

8.6.3. Documentación de los datos miembro

8.6.3.1. int cols

Número de columnas en la imagen.

Definición en la línea 24 del archivo structures.h.

8.6.3.2. int rows

Número de renglones en la imagen.

Definición en la línea 23 del archivo structures.h.

8.6.3.3. int size

El tamaño de la imagen.

Definición en la línea 26 del archivo structures.h.

8.6.3.4. int type

Tipo de la imagen (de acuerdo a la representación usada en openCV).

Definición en la línea 25 del archivo structures.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

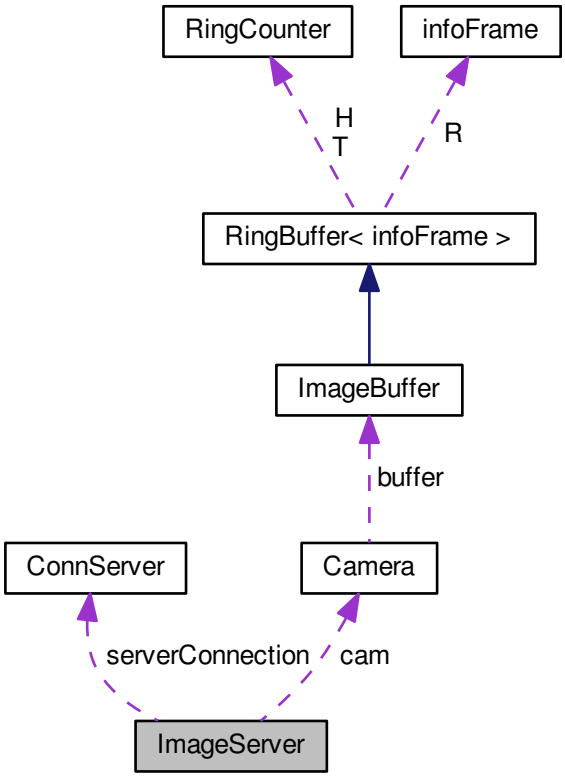
- include/structures.h

8.7. Referencia de la Clase ImageServer

Objetos instanciados de esta clase, capturan imágenes de una cámara y las almacena en un buffer, y permite la conexión via sockets de clientes a quienes provee de dichas imágenes.

`#include <ImageServer.h>`

Diagrama de colaboración para ImageServer:



Métodos públicos

- ImageServer (int cid, int port, const char ∗inetAddress, Mat ∗hrv, Mat ∗hmv, Mat ∗Mz, int mxConn=MAX_CON↩ NECTIONS)

  Constructor del objeto.

- ImageServer ()

  Constructor por defecto del objeto.

- void start ()

  Pone al servidor en un estado en donde puede aceptar conexiones de clientes.

Métodos privados estáticos

- static void ∗ connectionHandler (void ∗cd)

  Funcion que atiene a los clientes que se conectan al servicio.

Atributos privados

- int camId

  Numero que identifica la cámara que se va a utilizar.
- char * inetAddress

  Apuntador a arreglo de caracteres que contiene la dirección IP que se va a asociar al socket.
- int maxConnections

  El número máximo de conexiones permitidas.
- int port

  Puerto que se va a asociar al socket donde el servidor recibirá solicitudes.
- ConnServer * serverConnection

  Apuntador al objeto del tipo ConnServer que administra el servicios.

Atributos privados estáticos

- static Camera * cam

  Apuntador a objeto de la lcase Camera.
- static Mat * Hmv

  Apuntador a matriz de 3x3 que describe la transformación proyectiva que se aplica a la Máscara para ajustarla al tamáño de las imágenes capturadas.
- static Mat * Hrv

  Apuntador a matriz de 3x3 que describe la transformación proyectiva que se aplica a las imágenes capturadas.
- static Mat * Maze

  Apuntador a matriz que contiene la imagen de la máscara que se aplica a cada imagen capturada.

8.7.1. Descripción detallada

Objetos instanciados de esta clase, capturan imágenes de una cámara y las almacena en un buffer, y permite la conexión via sockets de clientes a quienes provee de dichas imágenes.

Definición en la línea 31 del archivo ImageServer.h.

8.7.2. Documentación del constructor y destructor

8.7.2.1. ImageServer (int *cid,* int *port,* const char * *inetAddress,* Mat * *hrv,* Mat * *hmv,* Mat * *Mz,* int *mxConn* = MAX_CONNECTIONS)

Constructor del objeto.

Parámetros

int	cid Identificador de la cámara que se va a usar.
int	port Puerto que se va a asociar al servidor.
const	char *inetAddress Apuntador a la cadena de caracteres que contiene la dirección IP a la que se va a asociar el servidor.

Mat	∗hrv Apuntador a Matriz que describe transformación proyectiva entre la región rectangular en el piso a observar y la imagen.
Mat	∗hmv Apuntador a la matriz que descrive la transformación proyectiva (usualmente una similitud), entre la imagen capturada, y la máscara.
Mat	∗Mz Apuntador a matriz que contiene la imagen que contiene el laberinto/Mascara.
int	mxConn Número máximo de conexiones permitidas.

Definición en la línea 28 del archivo ImageServer.cpp.

```
29 {
30     camId = cid;
31     Mat tmp;
32
33     maxConnections = mxConn;
34
35     // config Server
36          this->port = port;
37
38          this->inetAddress = (char *) malloc (strlen (inetAddress));
39          strcpy (this->inetAddress, inetAddress);
40
41          // init camera and assign thread to it
42
43          cam = new Camera (camId);
44      Hrv = new Mat;
45      Hmv = new Mat;
46      Maze = new Mat;
47          hrv->copyTo(*Hrv);
48          hmv->copyTo(*Hmv);
49          Mz->copyTo(*Maze);
50
51          //Se utilizó una variable temporal para evitar Bug en openCv en la version 2.3.1 que estaba
      instalada en el servidor que estamos ocupando.
52          tmp=Mat::zeros(Maze->size(), Maze->type());
53       warpPerspective(*Maze, tmp, *Hmv, Size(Maze->cols, Maze->rows), INTER_LINEAR,
     BORDER_CONSTANT);
54       tmp.copyTo(*Maze);
55          // init connection to server
56          serverConnection =
57                       new ConnServer (port, inetAddress, (
     ImageServer::connectionHandler), maxConnections);
58 }
```

8.7.2.2. ImageServer ( )

Constructor por defecto del objeto.

Definición en la línea 15 del archivo ImageServer.cpp.

```
16 {
17      maxConnections = MAX_CONNECTIONS;
18      camId = 0;
19      Hrv = new Mat;
20      Hmv = new Mat;
21      Maze = new Mat;
22      *Hrv = Mat::eye(3,3,CV_32FC1);
23      *Hmv = Mat::eye(3,3,CV_32FC1);
24      *Maze = Mat::ones(480,640,CV_32FC1);
25          ImageServer (0, 8888, "127.0.0.1", Hrv, Hmv, Maze,
     maxConnections);
26 }
```

8.7.3. Documentación de las funciones miembro

8.7.3.1. void ∗ connectionHandler (void ∗ *cd*) [static], [private]

Funcion que atiene a los clientes que se conectan al servicio.

Definición en la línea 83 del archivo ImageServer.cpp.

```
84 {
85         int sock = *((int *)cd);
86         unsigned char msg[MSG_LENGTH];
87         //Recibe mensaje del cliente
88         do
89         {
90                         // Leer comando
91                         memset (msg, 0, MSG_LENGTH);
92                         if (Read (sock, MSG_LENGTH, msg) < 0)
93                                 break;
94
95                         // Check what to send
96                         if (strncmp ((char *) msg, "IMG", 3) == 0)
97                         {
98                                 // Send an [IMG]
99
100                                 // Get frame from camera
101                                 infoFrame iF;
102                                 Mat mImg;
103                                 cam->getLastFrame (iF);
104                                 warpPerspective(iF.frame, mImg, *Hrv, Size(iF.
    frame.cols, iF.frame.rows), INTER_LINEAR, BORDER_CONSTANT);
105
106                 paintMaze(mImg, *Maze);
107
108                                 // Get info
109                                 struct ImageInfo imgInfo;
110                                 imgInfo.rows = mImg.rows;
111                                 imgInfo.cols = mImg.cols;
112                                 imgInfo.type = mImg.type ();
113                                 imgInfo.size = mImg.total () * mImg.elemSize ();
114
115                                 // send info
116                                 if (Write
117                                                 (sock, sizeof (struct ImageInfo), (unsigned char
    *) &imgInfo) < 0)
118                                                 break;
119
120                                 // send data
121                                 uchar *data;
122                                 data = mImg.data;
123                                 if (Write (sock, imgInfo.size, data) < 0 )
124                                    break;
125                         }
126                         else if (strncmp ((char *) msg, "POLL", 4) == 0)
127                         {
128                                 // Send ACK
129                                 memset (msg, 0, MSG_LENGTH);
130                                 strncpy ((char *) msg, "ACK", 3);
131                                 if (Write (sock, MSG_LENGTH, msg) < 0)
132                                                 break;
133                         }
134                         else if (strncmp ((char *) msg, "QUIT", 4) == 0)
135                         {
136                                 // Send ACK
137                 cout << "Recibimos QUIT" << endl;
138                 cout.flush();
139                                 memset (msg, 0, MSG_LENGTH);
140                                 strncpy ((char *) msg, "BYE", 3);
141                                 if (Write (sock, MSG_LENGTH, msg) < 0)
142                                                 break;
143                 cout << "Enviamos BYE" << endl;
144                 cout.flush();
145                                 break;
146                         }
147                         else
148                         {
149                                 // Send ERR
150                                 memset (msg, 0, MSG_LENGTH);
151                                 strncpy ((char *) msg, "ERR", 3);
152                                 if (Write (sock, MSG_LENGTH, msg) < 0)
153                                                 break;
154                         }
155         }
156         while (true);
157         cout << "Cliente con id: " << sock<< " Teminó"<< endl;
158     cout.flush();
159
160         return (void *) 0;
```

```
161 }
```

8.7.3.2. void start ( )

Pone al servidor en un estado en donde puede aceptar conexiones de clientes.

Definición en la línea 60 del archivo ImageServer.cpp.

```
61 {
62         serverConnection->acceptConnections ();
63 }
```

8.7.4. Documentación de los datos miembro

8.7.4.1. Camera * cam [static], [private]

Apuntador a objeto de la lcase Camera.

Definición en la línea 34 del archivo ImageServer.h.

8.7.4.2. int camId [private]

Numero que identifica la cámara que se va a utilizar.

Definición en la línea 41 del archivo ImageServer.h.

8.7.4.3. Mat * Hmv [static], [private]

Apuntador a matriz de 3x3 que describe la transformación proyectiva que se aplica a la Máscara para ajustarla al tamáño de las imágenes capturadas.

Definición en la línea 36 del archivo ImageServer.h.

8.7.4.4. Mat * Hrv [static], [private]

Apuntador a matriz de 3x3 que describe la transformación proyectiva que se aplica a las imágenes capturadas.

Definición en la línea 35 del archivo ImageServer.h.

8.7.4.5. char* inetAddress [private]

Apuntador a arreglo de caracteres que contiene la dirección IP que se va a asociar al socket.

Definición en la línea 39 del archivo ImageServer.h.

8.7.4.6. int maxConnections [private]

El número máximo de conexiones permitidas.

Definición en la línea 33 del archivo ImageServer.h.

8.7.4.7. Mat * Maze [static], [private]

Apuntador a matriz que contiene la imagen de la máscara que se aplica a cada imagen capturada.

Definición en la línea 37 del archivo ImageServer.h.

8.7.4.8. int port [private]

Puerto que se va a asociar al socket donde el servidor recibirá solicitudes.

Definición en la línea 40 del archivo ImageServer.h.

8.7.4.9. ConnServer* serverConnection [private]

Apuntador al objeto del tipo ConnServer que administra el servicios.

Definición en la línea 43 del archivo ImageServer.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- include/ImageServer.h
- src/ImageServer.cpp

8.8. Referencia de la Estructura infoFrame

Objetos instanciados a partir de esta estructura almacenan una imagen utilizando un objeto cv::Mat y el tiempo en que fue capturada (un 'timestamp').

`#include <infoFrame.h>`

Métodos públicos

- infoFrame ()

  constructor por defecto.
- infoFrame (const infoFrame &I)
- infoFrame & operator= (const infoFrame &I)

  Método que sobrecarga el operador de asignación.
- void setTime (struct timeval &_t)

  Define el timestamp del objeto con el valor _t.
- void setTime ()

  Método que asigna el timestamp del objeto con la fecha actual.

Atributos públicos

- cv::Mat frame

  Matriz en donde se almacenará la imagen capturada.
- struct timeval t

  estructura tipo timeval en donde se registra el instante en que la imagen fue capturada.

8.8.1. Descripción detallada

Objetos instanciados a partir de esta estructura almacenan una imagen utilizando un objeto cv::Mat y el tiempo en que fue capturada (un 'timestamp').

Definición en la línea 21 del archivo infoFrame.h.

8.8.2. Documentación del constructor y destructor

8.8.2.1. infoFrame ( ) [inline]

constructor por defecto.

Constructor de copia.

Parámetros

const	infoFrame &I Objeto que se va a copiar durante la inicialiación.

Devuelve

Definición en la línea 30 del archivo infoFrame.h.

```
30 { }
```

8.8.2.2. infoFrame (const infoFrame & I) [inline]

Definición en la línea 38 del archivo infoFrame.h.

```
39     {
40          I.frame.copyTo (this->frame);
41          this->t = I.t;
42     }
```

8.8.3. Documentación de las funciones miembro

8.8.3.1. infoFrame & operator= (const infoFrame & I) [inline]

Método que sobrecarga el operador de asignación.

Parámetros

const	infoFrame &I Objeto que se va a copiar.

Devuelve

Definición en la línea 50 del archivo infoFrame.h.

```
51     {
52          I.frame.copyTo (this->frame);
53          this->t = I.t;
54          return *this;
55     }
```

8.8.3.2. void setTime (struct timeval & *_t*) [inline]

Define el timestamp del objeto con el valor _t.

Parámetros

struct	timeval &_t Objeto que se va a utilizar en la incializacion.

Definición en la línea 62 del archivo infoFrame.h.

```
63      {
64           t = _t;
65      }
```

8.8.3.3. void setTime ( ) [inline]

Método que asigna el timestamp del objeto con la fecha actual.

Definición en la línea 71 del archivo infoFrame.h.

```
72      {
73           gettimeofday (&t, NULL);
74      }
```

8.8.4. Documentación de los datos miembro

8.8.4.1. cv::Mat frame

Matriz en donde se almacenará la imagen capturada.

Definición en la línea 23 del archivo infoFrame.h.

8.8.4.2. struct timeval t

estructura tipo timeval en donde se registra el instante en que la imagen fue capturada.

Definición en la línea 24 del archivo infoFrame.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- include/infoFrame.h

8.9. Referencia de la Clase remoteFrame

Métodos públicos

- def __init__
- def __del__
- def getFrame

Atributos públicos

- address
- cl
- Mask
- msgLength
- port
- where

8.9.1. Descripción detallada

Definición en la línea 8 del archivo remoteFrame.py.

8.9.2. Documentación del constructor y destructor

8.9.2.1. def __init__ (*self, address = '127.0.0.1', port = 8888, Mask = None*)

Definición en la línea 9 del archivo remoteFrame.py.

```
9
10      def __init__(self, address='127.0.0.1', port=8888, Mask=None):
11          self.address = address
12          self.port = port
13          self.cl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14          self.cl.connect((self.address, self.port))
15          self.msgLength = 10
16          if Mask != None:
17              b, g, r = cv2.split(Mask)
18              self.where = b & 255
19              self.where &= g & 255
20              self.where &= r & 255
21              self.where = 255-self.where
22              self.where = 255 - self.where
23              self.where=cv2.merge((self.where,self.where,self.
      where))
24              self.Mask = Mask & ~self.where
25          else:
             self.Mask = None
```

8.9.2.2. def __del__ (*self*)

Definición en la línea 26 del archivo remoteFrame.py.

```
26
27      def __del__ (self):
28          msg="QUIT      "
29          self.cl.sendall(msg)
30          msg = self.cl.recv(10)
31          if msg[:3] != "BYE":
32              print 'No se recibió "BYE" :-(, se recibió |'+msg+'| '
33          else:
             print 'remoteFrame: Cerrando conexión'
```

8.9.3. Documentación de las funciones miembro

8.9.3.1. def getFrame (*self*)

Definición en la línea 34 del archivo remoteFrame.py.

```
34
35      def getFrame(self):
36          msg="IMG       "
37          self.cl.sendall(msg)
38          msg = self.cl.recv(16)
39          msgb = map(ord, msg)
40          rows =(msgb[1]«8)+msgb[0]+(((msgb[3]«8)+msgb[2])«16)
41          cols =(msgb[5]«8)+msgb[4]+(((msgb[7]«8)+msgb[6])«16)
42          itype =(msgb[9]«8)+msgb[8]+(((msgb[11]«8)+msgb[10])«16)
43          size =(msgb[13]«8)+msgb[12]+(((msgb[15]«8)+msgb[14])«16)
44          recibidos = size
45          img=[]
46          while (recibidos != 0):
47              buff = self.cl.recv(recibidos)
```

```
48              recibidos -= len(buff)
49              img += buff
50          I=np.array(bytearray(img)).reshape(rows,cols,3)
51          if self.Mask != None:
52              I = (I & self.where) + self.Mask
53
54          return I
```

8.9.4. Documentación de los datos miembro

8.9.4.1. address

Definición en la línea 10 del archivo remoteFrame.py.

8.9.4.2. cl

Definición en la línea 12 del archivo remoteFrame.py.

8.9.4.3. Mask

Definición en la línea 23 del archivo remoteFrame.py.

8.9.4.4. msgLength

Definición en la línea 14 del archivo remoteFrame.py.

8.9.4.5. port

Definición en la línea 11 del archivo remoteFrame.py.

8.9.4.6. where

Definición en la línea 17 del archivo remoteFrame.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

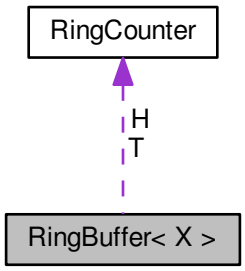
- remoteFrame.py

8.10. Referencia de la plantilla de la Clase RingBuffer< X >

Esta clase implementa un contador modular. El objeto funciona como un entero, que aritmeticamente opera usando aritmética modular.

`#include <RingBuffer.h>`

Diagrama de colaboración para RingBuffer< X >:



Métodos públicos

- RingBuffer (uint32_t N)

  Constructor de la clase.
- ~RingBuffer ()

  Destructor del objeto.
- virtual int Dequeue (X &e)
- virtual int getH ()

  Método que regresa el índice en el buffer donde se extraerá un objeto de la cola cuando se invoque método Dequeu. Esto es, la cabeza de la cola.
- virtual size_t getT ()

  Método que regresa el índice en el buffer donde se va a insertar un objeto cuando se invoque el método Queue. Esto es, el final de la cola.
- virtual void Init_Elements (const X &e)

  Inicializa el buffer con el objeto que se pasa por referencia.
- virtual int Queue (const X &e)

  Método virtual que inserta un elemento en la cola.
- virtual bool QueueIsEmpty ()

  Metodo que se utiliza para determinar si la cola esta vacia.

Atributos protegidos

- RingCounter H

  Indice circular que indica la cabeza de la cola. Se incrementa cuando se saca un objeto de la cola.
- X * R

  Apuntador al arreglo que consitituye el area de almacenamiento del buffer.
- pthread_mutex_t RB_mutex

  pthread_mutex que permite controlar el acceso cundo múltiples hebras operan sobre el objeto.
- uint32_t RBF_Size

  El tamaño del Buffer.
- RingCounter T

  Indice circular que indica el final de la cola. Se incrementa cuando se inserta un objeto de la cola.

8.10.1. Descripción detallada

template<typename X>class RingBuffer< X >

Esta clase implementa un contador modular. El objeto funciona como un entero, que aritmeticamente opera usando aritmética modular.

Definición en la línea 38 del archivo RingBuffer.h.

8.10.2. Documentación del constructor y destructor

8.10.2.1. RingBuffer (uint32_t *N*) [inline]

Constructor de la clase.

Parámetros

int	N Tamaño del arreglo que se utiliza para almacenar la cola; la cola así creada puede almacenar hasta N-1 elementos.

Definición en la línea 54 del archivo RingBuffer.h.

```
55          {
56                          pthread_mutex_init (&RB_mutex, NULL);   //initialises the mutex referenced
      by mutex with attributes specified by attr.
57                          pthread_mutex_lock (&RB_mutex);
58                          RBF_Size = N;
59                          if (RBF_Size > 1)
60                          {
61                                  R = new X[RBF_Size];          //Reserva memoria dinamica para la
      cola
62                          }
63                          else
64                                  R = 0;
65                          H.SetRingSize (RBF_Size);
66                          T.SetRingSize (RBF_Size);
67                          H = T = 0;
68                          pthread_mutex_unlock (&RB_mutex);
69          }
```

8.10.2.2. ~RingBuffer ( ) [inline]

Destructor del objeto.

Definición en la línea 75 del archivo RingBuffer.h.

```
76          {
77                          pthread_mutex_lock (&RB_mutex);
78                          if (R)
79                                  delete[]R;
80                          pthread_mutex_unlock (&RB_mutex);
81          }
```

8.10.3. Documentación de las funciones miembro

8.10.3.1. virtual int Dequeue (X & *e*) [inline], [virtual]

Reimplementado en ImageBuffer.

Definición en la línea 110 del archivo RingBuffer.h.

```
111         {
112                         pthread_mutex_lock (&RB_mutex);
113                         if (H != T)
114                         {
115                                 e = R[(uint32_t) H];
116                                 H++;
117                                 pthread_mutex_unlock (&RB_mutex);
118                                 return 0;
119                         }
120                         pthread_mutex_unlock (&RB_mutex);
121                         return -1;
122         }
```

8.10.3.2. int getH ( ) [inline], [virtual]

Método que regresa el índice en el buffer donde se extraerá un objeto de la cola cuando se invoque método Dequeu. Esto es, la cabeza de la cola.

Devuelve

El índice que referencía la cabeza de la cola.

Definición en la línea 140 del archivo RingBuffer.h.

```
141         {
142                         return H.getC ();
143         }
```

8.10.3.3. size_t getT ( ) [inline], [virtual]

Método que regresa el índice en el buffer donde se va a insertar un objeto cuando se invoque el método Queue. Esto es, el final de la cola.

Devuelve

El índice que referencia al final de la cola.

Definición en la línea 129 del archivo RingBuffer.h.

```
130         {
131                         return T.getC ();
132         }
```

8.10.3.4. void Init_Elements (const X & *e*) [inline], [virtual]

Inicializa el buffer con el objeto que se pasa por referencia.

Parámetros

const	X & e Objeto que se va a copiar a todos los elementos del buffer.

Definición en la línea 165 del archivo RingBuffer.h.

```
166         {
167                         uint32_t i;
168                         pthread_mutex_lock (&RB_mutex);
169                         H = T = 0;                                          //Asigna la
       C y RingSize de T a H
170                         for (i = 0; i < RBF_Size; ++i)
171                                 R[i] = e;
172                         pthread_mutex_unlock (&RB_mutex);
173         }
```

8.10.3.5. int Queue (const X & *e*) [inline], [virtual]

Método virtual que inserta un elemento en la cola.

Parámetros

cont	X & e Elemento a ser insertado en la cola.

Devuelve

El método regresa 0 en caso de éxito, y 1 en caso de que la cola se encuentre llena.

Reimplementado en ImageBuffer.

Definición en la línea 89 del archivo RingBuffer.h.

```
90          {
91                          pthread_mutex_lock (&RB_mutex);
92                          if (T + 1 != H)                                         //Compara T.C con
      H.C
93                          {
94                                  R[(uint32_t) T] = e;
95                                  T++;
      //T indica cuantos elementos hay en la cola
96                                  pthread_mutex_unlock (&RB_mutex);
97                                  return 0;
98                          }
99                          pthread_mutex_unlock (&RB_mutex);
100                          return -1;
101         }
```

8.10.3.6. bool QueueIsEmpty ( ) [inline], [virtual]

Metodo que se utiliza para determinar si la cola esta vacia.

Devuelve

El método regresa verdadero en caso de que la cosa este vacia, y falso en caso contrario.

Definición en la línea 151 del archivo RingBuffer.h.

```
152         {
153                          if (T == H)
154                                  return true;
155                          else
156                                  return false;
157         }
```

8.10.4. Documentación de los datos miembro

8.10.4.1. RingCounter H [protected]

Indice circular que indica la cabeza de la cola. Se incrementa cuando se saca un objeto de la cola.

Definición en la línea 43 del archivo RingBuffer.h.

8.10.4.2. X* R [protected]

Apuntador al arreglo que consitituye el area de almacenamiento del buffer.

Definición en la línea 42 del archivo RingBuffer.h.

8.10.4.3. pthread_mutex_t RB_mutex [protected]

pthread_mutex que permite controlar el acceso cundo múltiples hebras operan sobre el objeto.

Definición en la línea 46 del archivo RingBuffer.h.

8.10.4.4. uint32_t RBF_Size [protected]

El tamaño del Buffer.

Definición en la línea 41 del archivo RingBuffer.h.

8.10.4.5. RingCounter T [protected]

Indice circular que indica el final de la cola. Se incrementa cuando se inserta un objeto de la cola.

Definición en la línea 44 del archivo RingBuffer.h.

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/RingBuffer.h

8.11. Referencia de la Clase RingCounter

`#include <RingCounter.h>`

Métodos públicos

- RingCounter ()
  Constructor del objeto.
- RingCounter (size_t rz)
  Constructor del objeto.
- RingCounter (const RingCounter &R)
  Constructor de copia.
- size_t getC ()
  Método que regresa el valor almacenado en el objeto.
- operator size_t ()
  Método que sobrecarga el operador size_t.
- bool operator!= (const RingCounter &R)
  Método que sobrecarga el operador de comparacion (no igualdad).
- template<typename X >
  RingCounter operator+ (const X &n)
  Método plantilla que sobrecarga el operador +.
- RingCounter & operator++ ()
  Método que sobrecarga el operador ++.
- RingCounter & operator++ (int)
  Version postfix del método que sobrecarga el operador ++.
- template<typename X >
  RingCounter & operator+= (X &val)

- template<typename X >
  RingCounter operator- (const X &n)

  Método plantilla que sobrecarga el operador -.
- RingCounter & operator-- ()

  Método que sobrecarga el operador –.
- RingCounter & operator-- (int)

  Version postfix del método que sobrecarga el operador –.
- template<typename X >
  RingCounter & operator-= (const X &n)

  Método plantilla que sobrecarga el operador -=.
- RingCounter & operator= (const size_t &v)

  Método que sobrecarga el operador de asignación para asignar un valor al objeto.
- RingCounter & operator= (const RingCounter &R)

  Método que sobrecarga el operador de asignación para asignar un objeto de tipo RingCounter al objeto que invoca.
- bool operator== (const RingCounter &R)

  Método que sobrecarga el operador de comparacion (igualdad).
- bool operator> (const RingCounter &R)

  Método que sobrecarga el operador 'mayor que' (>).
- bool operator>= (const RingCounter &R)

  Método que sobrecarga el operador 'mayor o igual' (>=).
- void SetRingSize (size_t val)

  Define el numero de valores diferentes permitidos en el objeto, i.e. el módulo.

Atributos privados

- size_t C

  En esta variable se almacena el valor representado por el objeto.
- size_t RingSize

  El módulo; el valor máximo que puede tomar el objeto.

8.11.1. Descripción detallada

Definición en la línea 36 del archivo RingCounter.h.

8.11.2. Documentación del constructor y destructor

8.11.2.1. RingCounter ( ) [inline]

Constructor del objeto.

Definición en la línea 46 del archivo RingCounter.h.

```
47          {
48                          RingSize = 0;
49                          C = 0;
50          }
```

8.11.2.2. RingCounter (size_t *rz*) [inline]

Constructor del objeto.

Parámetros

size_t	rz El numero de valores diferentes permitidos en el objeto, i.e. el módulo.

Definición en la línea 57 del archivo RingCounter.h.

```
58          {
59
60                          RingSize = rz;
61                          C = 0;
62          }
```

8.11.2.3. RingCounter (const RingCounter & *R*) [inline]

Constructor de copia.

Parámetros

const	RingCounter & R Objeto de tipo RingCounter que se va a copiar.

Definición en la línea 69 del archivo RingCounter.h.

```
70          {
71                          C = R.C;
72                          RingSize = R.RingSize;
73          }
```

8.11.3. Documentación de las funciones miembro

8.11.3.1. size_t getC ( ) [inline]

Método que regresa el valor almacenado en el objeto.

Devuelve

Regresa el valor almacenado en el objeto.

Definición en la línea 283 del archivo RingCounter.h.

```
284          {
285                          return C;
286          }
```

8.11.3.2. operator size_t ( ) [inline]

Método que sobrecarga el operador size_t.

Definición en la línea 92 del archivo RingCounter.h.

```
93          {
94                          return C;
95          }
```

8.11.3.3. bool operator!= (const RingCounter & *R*) [inline]

Método que sobrecarga el operador de comparacion (no igualdad).

Parámetros

const	RingCounter &R El objeto que se va a comparar con el objeto que invoca.

Devuelve

El método regresa falso si ambos objetos son iguales; i.e. si el módulo (RingSize) es el mismo, y si el valor almacenado (C) es igual. En caso contrario regresa verdadero.

Definición en la línea 245 del archivo RingCounter.h.

```
246         {
247                         if (RingSize == R.RingSize && C != R.C)
248                                 return true;
249                         return false;
250         }
```

8.11.3.4. template< typename X > RingCounter operator+ (const X & *n*) [inline]

Método plantilla que sobrecarga el operador +.

Parámetros

const	X & n Objeto que se suma al objeto que invoca.

Devuelve

Regresa un nuevo objeto resultado de la suma del objeto que invoca y el parámetro recibido.

Definición en la línea 128 del archivo RingCounter.h.

```
129         {
130                         RingCounter R (RingSize);
131
132                         R.C = (C + (size_t) n) % RingSize;
133                         return R;
134         }
```

8.11.3.5. RingCounter & operator++ ( ) [inline]

Método que sobrecarga el operador ++.

Devuelve

Regresa el objeto que invoca.

Definición en la línea 188 del archivo RingCounter.h.

```
189         {
190                         C = (C + 1) % RingSize;
191                         return *this;
192         }
```

8.11.3.6. RingCounter & operator++ (int) [inline]

Version postfix del método que sobrecarga el operador ++.

Devuelve

Regresa el objeto que invoca.

Definición en la línea 210 del archivo RingCounter.h.

```
211         {
212                         C = (C + 1) % RingSize;
213                         return *this;
214         }
```

8.11.3.7. RingCounter& operator+= (X & *val*) [inline]

Definición en la línea 160 del archivo RingCounter.h.

```
161         {
162                         C = (C + (size_t) val) % RingSize;
163                         return *this;
164         }
```

8.11.3.8. template< typename X > RingCounter operator- (const X & *n*) [inline]

Método plantilla que sobrecarga el operador -.

Parámetros

const	X & n Objeto que se resta al objeto que invoca.

Devuelve

Regresa un nuevo objeto resultado de la resta del parametro recibido al objeto que invoca.

Definición en la línea 142 del archivo RingCounter.h.

```
143         {
144                         RingCounter R (RingSize);
145                         int diff = (int) C - (int) n;
146
147                         if (diff >= 0)
148                                 R.C = (C - n) % RingSize;
149                         else
150                                 R.C = RingSize - (-diff % RingSize);
151                         return R;
152         }
```

8.11.3.9. RingCounter & operator-- ( ) [inline]

Método que sobrecarga el operador –.

Devuelve

Regresa el objeto que invoca.

Definición en la línea 199 del archivo RingCounter.h.

```
200         {
201                         C = !C ? RingSize - 1 : (C - 1) % RingSize;
202                         return *this;
203         }
```

8.11.3.10. RingCounter & operator-- (int) [inline]

Version postfix del método que sobrecarga el operador –.

Devuelve

Regresa el objeto que invoca.

Definición en la línea 220 del archivo RingCounter.h.

```
221         {
222                         C = !C ? RingSize - 1 : (C - 1) % RingSize;
223                         return *this;
224         }
```

8.11.3.11. template< typename X > RingCounter & operator-= (const X & *n*) [inline]

Método plantilla que sobrecarga el operador -=.

Parámetros

const	X & val Objeto que se resta al objeto que invoca.

Devuelve

Regresa el objeto que invoca,

Definición en la línea 172 del archivo RingCounter.h.

```
173         {
174                         int diff = (int) C - (int) n;
175
176                         if (diff >= 0)
177                                 C = (C - n) % RingSize;
178                         else
179                                 C = RingSize - (-diff % RingSize);
180                         return *this;
181         }
```

8.11.3.12. RingCounter & operator= (const size_t & *v*) [inline]

Método que sobrecarga el operador de asignación para asignar un valor al objeto.

Parámetros

const	size_t & v El objeto que va a copiarse en el objeto que invoca

Devuelve

Regresa el objeto que invoca.

Definición en la línea 103 del archivo RingCounter.h.

```
104         {
105                         C = v % RingSize;
106                         return *this;
107         }
```

8.11.3.13. RingCounter & operator= (const RingCounter & *R*) [inline]

Método que sobrecarga el operador de asignación para asignar un objeto de tipo RingCounter al objeto que invoca.

Parámetros

const	RungCounter &R Objeto que va a copiarse al objeto que invoca.

Devuelve

Regresa el objeto que invoca.

Definición en la línea 115 del archivo RingCounter.h.

```
116         {
117                         C = R.C;
118                         RingSize = R.RingSize;
119                         return *this;
120         }
```

8.11.3.14. bool operator== (const RingCounter & *R*) [inline]

Método que sobrecarga el operador de comparacion (igualdad).

Parámetros

const	RingCounter &R El objeto que se va a comparar con el objeto que invoca.

Devuelve

El método regresa verdadero si ambos objetos son iguales; i.e. si el módulo (RingSize) es el mismo, y si el valor almacenado (C) es igual. En caso contrario regresa falso.

Definición en la línea 232 del archivo RingCounter.h.

```
233         {
234                         if (RingSize == R.RingSize && C == R.C)
235                                 return true;
236                         return false;
237         }
```

8.11.3.15. bool operator> (const RingCounter & *R*) [inline]

Método que sobrecarga el operador 'mayor que' (>).

Parámetros

const	RingCounter &R El objeto que se va a comparar con el objeto que invoca.

Devuelve

El método regresa verdadero si el objeto que invoca es mayor que el objeto que se pasa por parametro. En caso contrario regresa verdadero.

Definición en la línea 258 del archivo RingCounter.h.

```
259         {
260                         if (RingSize == R.RingSize && C > R.C)
261                                 return true;
262                         return false;
263         }
```

8.11.3.16. bool operator>= (const RingCounter & *R*) [inline]

Método que sobrecarga el operador 'mayor o igual' (>=).

Parámetros

const	RingCounter &R El objeto que se va a comparar con el objeto que invoca.

Devuelve

El método regresa verdadero si el objeto que invoca es mayor o igual que el objeto que se pasa por parametro. En caso contrario regresa verdadero.

Definición en la línea 271 del archivo RingCounter.h.

```
272         {
273                         if (C > R.C && RingSize == R.RingSize)
274                                 return true;
275                         return false;
276         }
```

8.11.3.17. void SetRingSize (size_t *val*) [inline]

Define el numero de valores diferentes permitidos en el objeto, i.e. el módulo.

Parámetros

size_t	val El nuevo valor a asignar

Definición en la línea 80 del archivo RingCounter.h.

```
81          {
82                          RingSize = val;
83                          if (C > val)
84                              C = C % val;
85          }
```

8.11.4. Documentación de los datos miembro

8.11.4.1. size_t C [private]

En esta variable se almacena el valor representado por el objeto.

Definición en la línea 38 del archivo RingCounter.h.

8.11.4.2. size_t RingSize [private]

El módulo; el valor máximo que puede tomar el objeto.

Definición en la línea 39 del archivo RingCounter.h.

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/RingCounter.h

8.12. Referencia de la Clase template

Esta clase define una cola finita a partir de un arreglo. Depende del objeto RingCounter para manejar indices circulares. La clase se construye como un plantilla, y se espera que los objetos quw consitituyan la cola tengan sobrecargado el operador de copia. La clase provee métodos para añadir un objeto al final de la cola, sacar del frente de la cola. La clase cuenta con pthread_mutexes para sincronizar acceso cuando se utiliza en un ambiente multihebras.

8.12.1. Descripción detallada

Esta clase define una cola finita a partir de un arreglo. Depende del objeto RingCounter para manejar indices circulares. La clase se construye como un plantilla, y se espera que los objetos quw consitituyan la cola tengan sobrecargado el operador de copia. La clase provee métodos para añadir un objeto al final de la cola, sacar del frente de la cola. La clase cuenta con pthread_mutexes para sincronizar acceso cuando se utiliza en un ambiente multihebras.

class RingBuffer

La documentación para esta clase fue generada a partir del siguiente fichero:

- include/RingBuffer.h

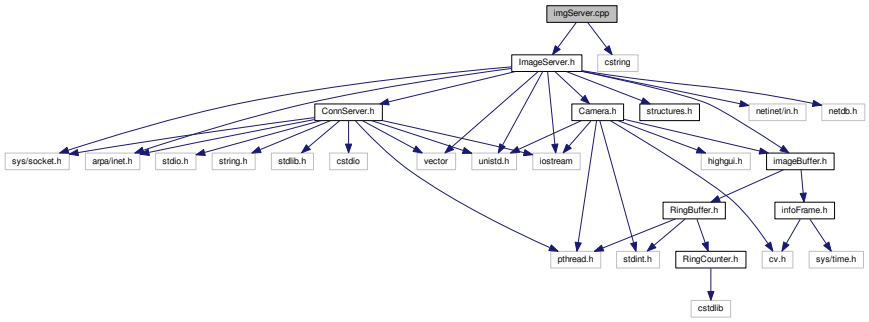
Capítulo 9

Documentación de archivos

9.1. Referencia del Archivo imgServer.cpp

```
#include <ImageServer.h>
#include <cstring>
```

Dependencia gráfica adjunta para imgServer.cpp:



Clases

- struct cornerData

  Esta estructura contiene información de esquinas que se utiliza durante la captura de posiciones en la pantalla usando el raton.

Funciones

- void dibujaPuntos (Mat &frame, Point2f ∗p, int n, Scalar color=Scalar(0, 0, 255))

  Esta función dibuja una lista de puntos en una imagen.
- void findMapping (int source, Point2f ∗vP, Mat &H)

  Esta funcion se utiliza para obtener la información necesaria para rectificar las imagenes que el servidor va a capturar. La funcion captura imágenes de la fuente de video "source", permite que usuario capture 4 coordenadas en la imagen utilizando el mouse, y a partir de esas cuatro coordenadas, calcula la transformación proyectiva que describe la transformación de esas 4 coordenadas al plano de la imagen. Se presume que el usuario va a elegir cuatro esquinas que yacen en un plano en la escena, y que la tranformación producida, permitira dedicar la atención de esa región únicamente.

- int main (int argc, char **argv)
- void onMouseEvent (int event, int x, int y, int flags, void *data)

  Funcion que atiende evantos generados por el ratón. Esta funcion es invocada automáticamnete por el sistema cuando se registra usando la función setMouseCallback de la biblioteca highgui.

9.1.1. Documentación de las funciones

9.1.1.1. void dibujaPuntos (Mat & *frame*, Point2f * *p*, int *n*, Scalar *color* = `Scalar(0,0,255)`)

Esta función dibuja una lista de puntos en una imagen.

Parámetros

Mat	&frame Imagen en donde se van a dibujar los puntos.
Point2f	*p Apuntador al arreglo de objetos de tipo Point2f que contiene las coordenadas de los puntos a dibujar.
int	n Cantidad de puntos a dibujar.
Scalar	color Objeto de tipo scalar que codifica en fomrto BGR el color de los puntos a dibujar.

Definición en la línea 73 del archivo imgServer.cpp.

```
74 {
75
76     for (int i=0;i<n; ++i)
77         circle(frame, Point((int)p[i].x, (int)p[i].y), 5, color);
78 }
```

9.1.1.2. void findMapping (int *source*, Point2f * *vP*, Mat & *H*)

Esta funcion se utiliza para obtener la información necesaria para rectificar las imagenes que el servidor va a capturar. La funcion captura imágenes de la fuente de video "source", permite que usuario capture 4 coordenadas en la imagen utilizando el mouse, y a partir de esas cuatro coordenadas, calcula la transformación proyectiva que describe la transformación de esas 4 coordenadas al plano de la imagen. Se presume que el usuario va a elegir cuatro esquinas que yacen en un plano en la escena, y que la tranformación producida, permitira dedicar la atención de esa región únicamente.

Parámetros

int	source El identificador de la cámara que se va a utilizar. Point2f *vp

Definición en la línea 86 del archivo imgServer.cpp.

```
87 {
88     Point2f rP[4];
89     Mat Frame;
90     VideoCapture cap(source);
91     cornerData cD;
92     int i = 0;
93
94     if (!cap.isOpened())
95     {
96         cerr « "no se pudo abrir la cámara" « endl;
97         return;
98     }
99
100     namedWindow("Introduce esquinas");
101     setMouseCallback("Introduce esquinas", onMouseEvent, (void*) &cD);
102     cD.cont = 0;
103     while (cD.cont < 4)
104     {
105         cap » Frame;
106         if (Frame.empty())
```

```
107             cerr « "capturaPuntos: Error capturando el Frame" « endl;
108         imshow("Introduce esquinas", Frame);
109         dibujaPuntos (Frame, cD.crn, cD.cont);
110         if (waitKey(30) >= 0)
111             break;
112     }
113     cvSetMouseCallback("Introduce esquinas", 0, 0);
114     if (cD.cont < 4)
115       cerr « "La captura de puntos se aborto." « endl;
116     for (int i=0;i<4;++i)
117         rP[i] = cD.crn[i];
118     cap.release();
119
120     cout « "Se capturaron los siguientes puntos:" « endl;
121     for (i=0;i<4;++i)
122         cout « "(" « rP[i].x « ", " « rP[i].y « ")" « endl;
123     H = getPerspectiveTransform(rP, vP);
124     destroyWindow("Introduce esquinas");
125 }
```

9.1.1.3. int main (int *argc*, char ** *argv*)

Definición en la línea 127 del archivo imgServer.cpp.

```
128 {
129          int port = 8888, camId = 0;
130          char ipAddress[64] = "127.0.0.1";
131          Point2f vP[4], mP[4];
132      Mat Hrv, Hmv, Maze;
133
134          Maze = 255 * Mat::ones(Size(640,480), CV_8UC3);
135          if (argc < 2)
136          {
137             cerr « "Uso: imgServer camId ipAddress port Maze" « endl;
138             return -1;
139      }
140      else
141         camId = atoi(argv[1]);
142          if (argc > 2)
143          {
144                         strncpy (ipAddress, argv[2], 63);
145                         if (argc > 3)
146                         {
147                             port = atoi (argv[3]);
148                             if (argc > 4)
149                                 Maze = imread(argv[4]);
150                 }
151         }
152
153      vP[0].x = 0; vP[0].y = 0;
154      vP[1].x = 639; vP[1].y = 0;
155      vP[2].x = 639; vP[2].y = 479;
156      vP[3].x = 0; vP[3].y = 479;
157      findMapping(camId, vP, Hrv);
158      cout « "Calculamos la Hrv como:"« endl « Hrv « endl;
159
160      mP[0].x = 0; mP[0].y = 0;
161      mP[1].x = Maze.cols; mP[1].y = 0;
162      mP[2].x = Maze.cols; mP[2].y = Maze.rows;
163      mP[3].x = 0; mP[3].y = Maze.rows;
164      Hmv = getPerspectiveTransform(mP, vP);
165      cout « "[" « Maze.cols « ", " « Maze.rows « "]" « endl;
166      cout « "Calculamos la Hmv como:"« endl « Hmv « endl;
167
168          ImageServer *imS = new ImageServer (camId, port, ipAddress, &Hrv, &Hmv, &
    Maze);
169
170          imS->start ();//Esta funcion bloquea la ejecución del programa. ¿debiera ocurrir?
171
172          return 0;
173 }
```

9.1.1.4. void onMouseEvent (int *event*, int *x*, int *y*, int *flags*, void * *data*)

Funcion que atiende evantos generados por el ratón. Esta funcion es invocada automáticamnete por el sistema cuando se registra usando la función setMouseCallback de la biblioteca highgui.

Definición en la línea 52 del archivo imgServer.cpp.

```
53 {
54     cornerData *cD;
55
56     cD = (cornerData *)data;
57     if (event == CV_EVENT_LBUTTONDOWN)
58     {
59         cD->crn[cD->cont].x = x;
60         cD->crn[cD->cont].y = y;
61         cD->cont++;
62     }
63 }
```

9.2. Referencia del Archivo include/Camera.h

Archivo de encabezado donde se define la clase Camera. Esta Clase tiene como función capturar imágenes una camara. Cada cuadro capturado se almacena en una cola, para que pueda ser procesados posteriormente.

```
#include <imageBuffer.h>
#include <cv.h>
#include <highgui.h>
#include <pthread.h>
#include <unistd.h>
#include <stdint.h>
#include <iostream>
```

Dependencia gráfica adjunta para Camera.h:

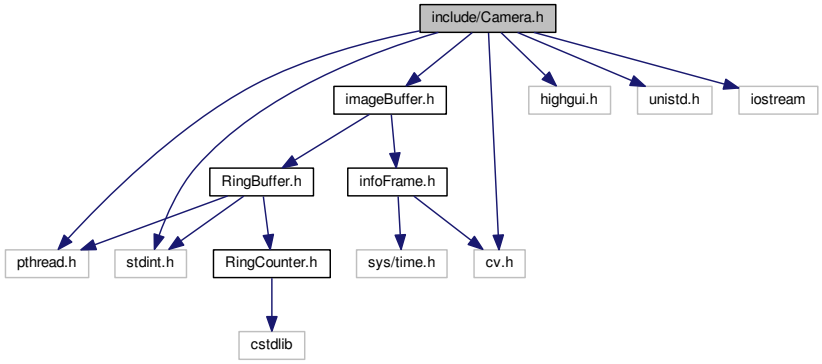
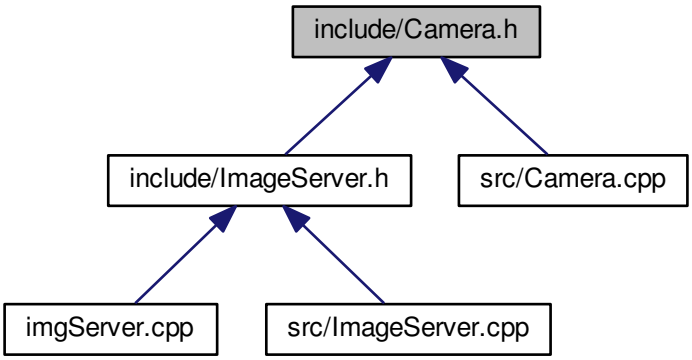


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class Camera

  Esta clase permite capturar imágenes de una cámara, utilizando los métodos provisto por la biblioteca highgui. Las imagenes capturadas son almacenadas en una cola. Se proveen mecanismos de sincronización para el acceso concurrente a dicha cola.

9.2.1. Descripción detallada

Archivo de encabezado donde se define la clase Camera. Esta Clase tiene como función capturar imágenes una camara. Cada cuadro capturado se almacena en una cola, para que pueda ser procesados posteriormente.

Al instanciar un objeto de esta clase, se genera para asegurar la ejecución continua del ciclo de captura.

Definición en el archivo Camera.h.

9.3. Referencia del Archivo include/Client.h

En este archivo de encabezado se define la clase Client. Dicha clase crea al ser instanciada un objeto que facilita la parte cliente de un sistema de comunicación basado en sockets.

```
#include <cv.h>
#include <SockIO.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <iostream>
#include <string.h>
#include <structures.h>
```

Dependencia gráfica adjunta para Client.h:

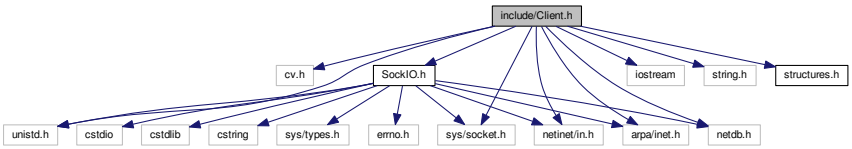
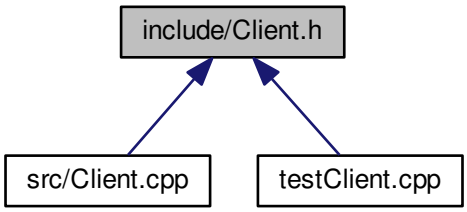


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class Client

  Esta clase crea un objeto que permite conectarse via un socket con un servidor, para transmitir información.

'defines'

- #define MAXDATASIZE 6220800

9.3.1. Descripción detallada

En este archivo de encabezado se define la clase Client. Dicha clase crea al ser instanciada un objeto que facilita la parte cliente de un sistema de comunicación basado en sockets.

Definición en el archivo Client.h.

9.3.2. Documentación de los 'defines'

9.3.2.1. #define MAXDATASIZE 6220800

Definición en la línea 25 del archivo Client.h.

9.4. Referencia del Archivo include/ConnServer.h

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <pthread.h>
#include <iostream>
#include <cstdio>
#include <vector>
```

Dependencia gráfica adjunta para ConnServer.h:

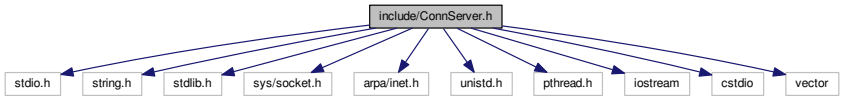
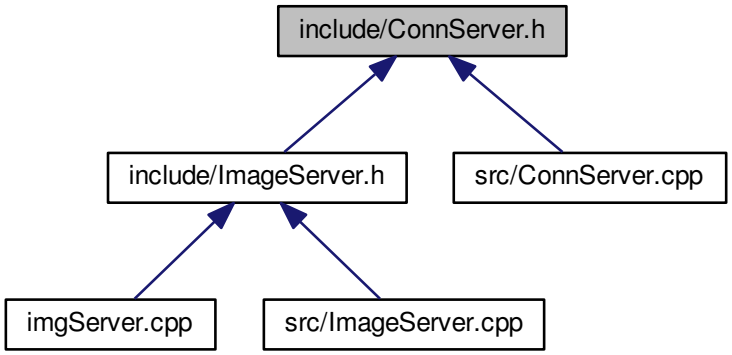


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class ConnServer

  Esta clase crea un "stream socket" y lo asocia a una direccion IP y un puerto, y provee servicios básicos para atender de manera concurrente a multiples clientes.

'defines'

- #define BACKLOG 5

9.4.1. Documentación de los 'defines'

9.4.1.1. #define BACKLOG 5

Definición en la línea 18 del archivo ConnServer.h.

9.5. Referencia del Archivo include/imageBuffer.h

```
#include <RingBuffer.h>
#include <infoFrame.h>
```

Dependencia gráfica adjunta para imageBuffer.h:

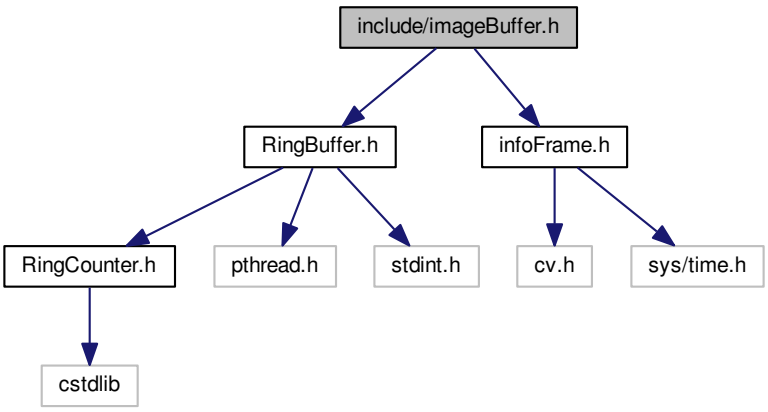
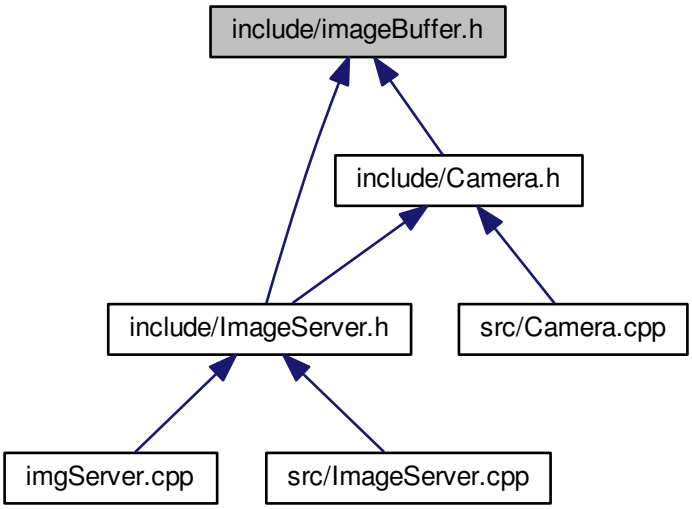


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class ImageBuffer

  Esta clase especializa la clase Ringbuffer para operar on objetos de tipo infoFrame. Aparte añade dos métodos nuevos: getLast y advHead.

9.6. Referencia del Archivo include/ImageServer.h

Este archivo contiene las definición de la clase ImageServer. Objetos instanciados de esta clase capturan imágenes de una cámara, y los ofrecen a traves de un socket a clientes que se conectan a el. Ademas provee capacidades, para transformar la imagenes capturadas (homografias), y mezclar éstas con imagenes predefinidas.

```
#include <ConnServer.h>
#include <imageBuffer.h>
#include <structures.h>
#include <Camera.h>
#include <vector>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <iostream>
```

Dependencia gráfica adjunta para ImageServer.h:

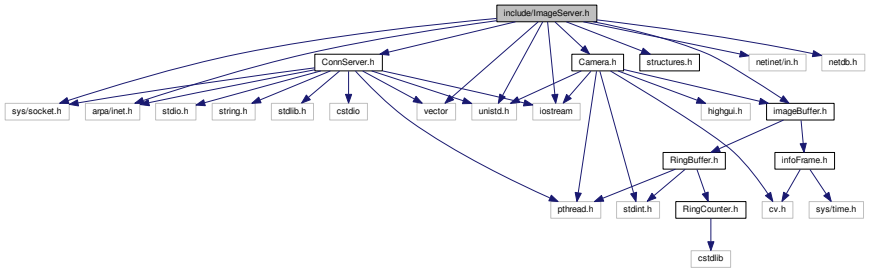
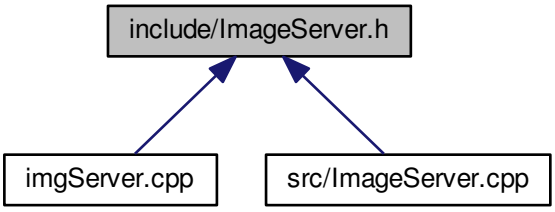


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class ImageServer

  Objetos instanciados de esta clase, capturan imágenes de una cámara y las almacena en un buffer, y permite la conexión via sockets de clientes a quienes provee de dichas imágenes.

'defines'

- #define MAX_CONNECTIONS 1000

Funciones

- void paintMaze (Mat &I, Mat &Mask)

  Aplica una mascara a color a una imagen. En aquellos pixeles de la máscara son diferentes a RGB[255,255,255] (color blanco), el pixel correspondiente de la imagen se sustituye por el valor de la máscara. De caso contrario, el valor original del pixel, se preserva,.

9.6.1. Descripción detallada

Este archivo contiene las definición de la clase ImageServer. Objetos instanciados de esta clase capturan imágenes de una cámara, y los ofrecen a traves de un socket a clientes que se conectan a el. Ademas provee capacidades, para transformar la imagenes capturadas (homografias), y mezclar éstas con imagenes predefinidas.

Definición en el archivo ImageServer.h.

9.6.2. Documentación de los 'defines'

9.6.2.1. #define MAX_CONNECTIONS 1000

Definición en la línea 25 del archivo ImageServer.h.

9.6.3. Documentación de las funciones

9.6.3.1. void paintMaze (Mat & *I*, Mat & *Mask*)

Aplica una mascara a color a una imagen. En aquellos pixeles de la máscara son diferentes a RGB[255,255,255] (color blanco), el pixel correspondiente de la imagen se sustituye por el valor de la máscara. De caso contrario, el valor original del pixel, se preserva,.

Parámetros

Mat	&I Matriz que contiene la imagen a Enmascarar.
Mat	&Mask Mascara que se debe a plicar a la imagen I.

Definición en la línea 65 del archivo ImageServer.cpp.

```
66 {
67     int i, j;
68     Vec3b *ptrI, *ptrM;
69     Vec3b val(255,255,255);
70 
71     assert (I.rows == Mask.rows && I.cols == Mask.cols);
72     for (i=0;i<I.rows;++i)
73     {
74         ptrI = I.ptr<Vec3b>(i);
75         ptrM = Mask.ptr<Vec3b>(i);
76         for (j=0;j<I.cols;++j,++ptrI,++ptrM)
77             if (*ptrM != val)
78                 *ptrI = *ptrM;
79     }
80 }
```

9.7. Referencia del Archivo include/infoFrame.h

En este archivo se encuentra la defincion de la estructura infoFrame, que se utiliza para almacenar una imagen junto con el tiempo en que fue capturada.

```
#include <cv.h>
#include <sys/time.h>
```

Dependencia gráfica adjunta para infoFrame.h:

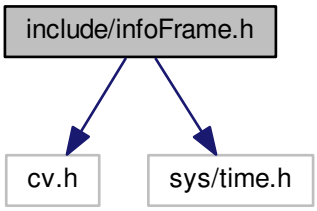
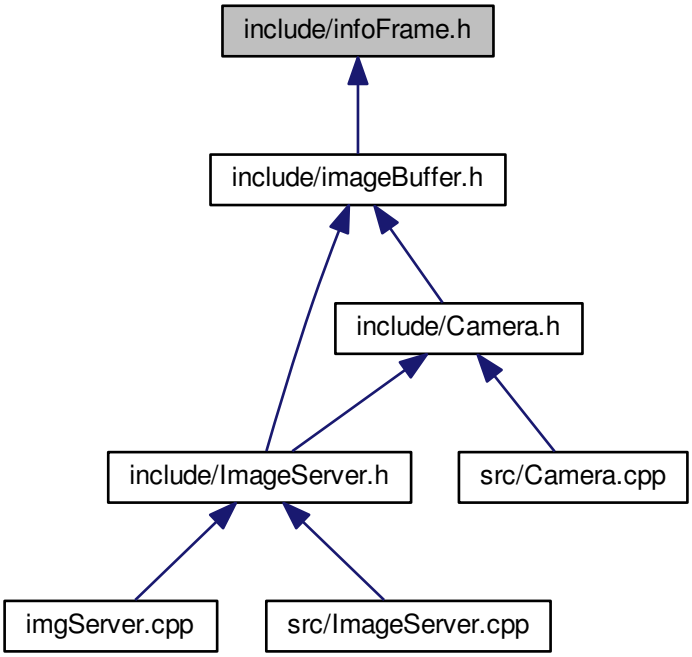


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- struct infoFrame

  Objetos instanciados a partir de esta estructura almacenan una imagen utilizando un objeto cv::Mat y el tiempo en que fue capturada (un 'timestamp').

9.7.1. Descripción detallada

En este archivo se encuentra la defincion de la estructura infoFrame, que se utiliza para almacenar una imagen junto con el tiempo en que fue capturada.

Definición en el archivo infoFrame.h.

9.8. Referencia del Archivo include/RingBuffer.h

Archivo de encabezado en donde se define ela clase RingBuffer.

```
#include <RingCounter.h>
#include <pthread.h>
#include <stdint.h>
```

Dependencia gráfica adjunta para RingBuffer.h:

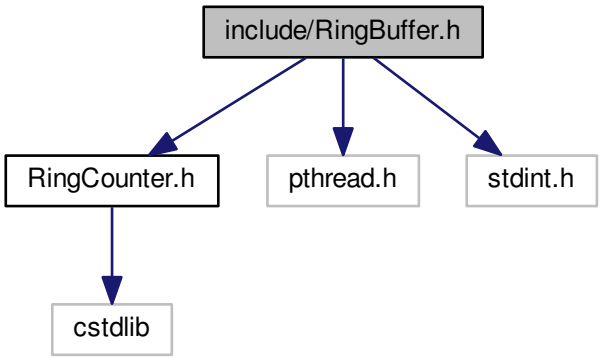
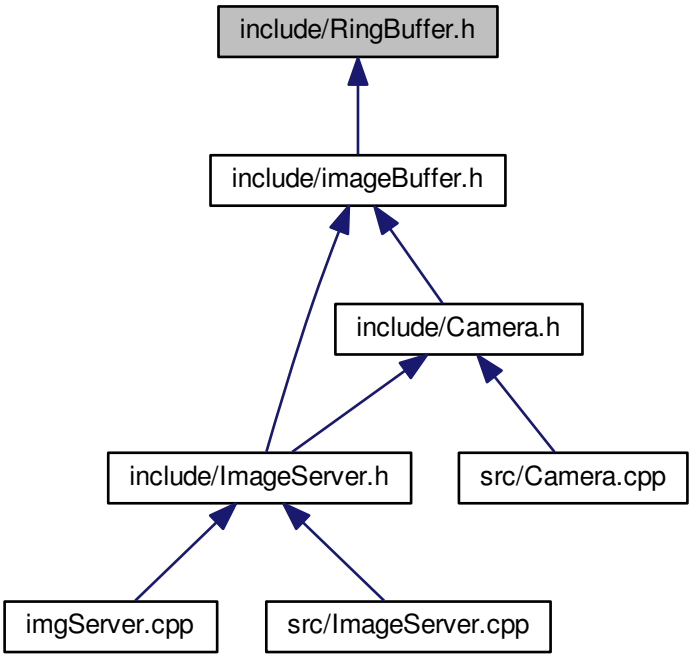


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class RingBuffer< X >

  Esta clase implementa un contador modular. El objeto funciona como un entero, que aritmeticamente opera usando aritmética modular.

9.8.1. Descripción detallada

Archivo de encabezado en donde se define ela clase RingBuffer.

Definición en el archivo RingBuffer.h.

9.9. Referencia del Archivo include/RingCounter.h

Este archivo contiene la definición de un contador módular.

```
#include <cstdlib>
```

Dependencia gráfica adjunta para RingCounter.h:

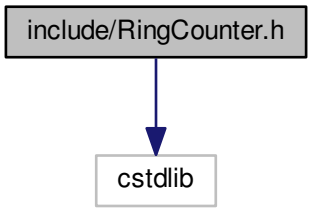
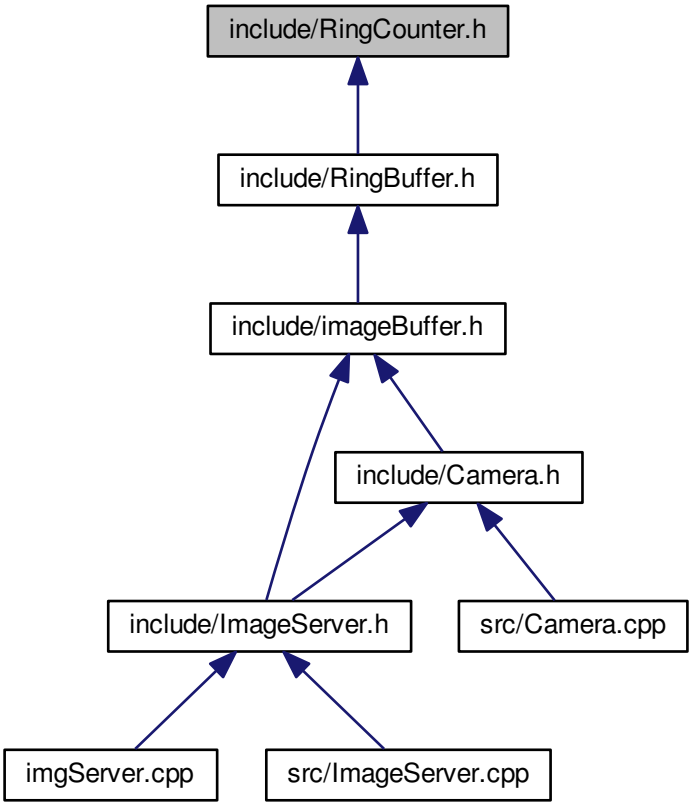


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class RingCounter

9.9.1. Descripción detallada

Este archivo contiene la definición de un contador módular.

Definición en el archivo RingCounter.h.

9.10. Referencia del Archivo include/SockIO.h

En este archivo de encabezado se definen dos funciones que permiten enviar y recibir una cantidad arbitrariamente grande de datos a través de un socket.

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <errno.h>
```

Dependencia gráfica adjunta para SockIO.h:

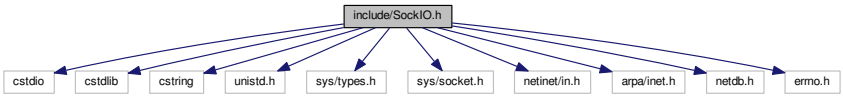
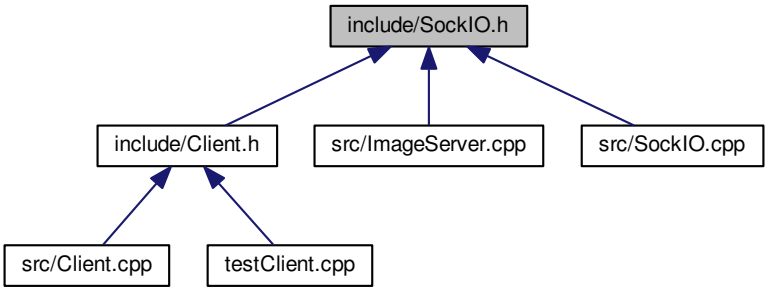


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



'defines'

- #define MAXCHUNK 4096

Funciones

- int Read (int sock, u_long tam, unsigned char *buffer)

  Esta funcion lee tam bytes del socket sock y lo almacena en el arreglo apuntado por buffer.
- int Write (int sock, u_long tam, unsigned char *buffer)

  Esta función escribe tam bytes, que se encuentran almacenados en la cadena de caracteres referida por el apuntador buffer en el socket identificado por sock.

9.10.1. Descripción detallada

En este archivo de encabezado se definen dos funciones que permiten enviar y recibir una cantidad arbitrariamente grande de datos a través de un socket.

Definición en el archivo SockIO.h.

9.10.2. Documentación de los 'defines'

9.10.2.1. #define MAXCHUNK 4096

Definición en la línea 9 del archivo SockIO.h.

9.10.3. Documentación de las funciones

9.10.3.1. int Read (int *sock*, u_long *tam*, unsigned char * *buffer*)

Esta funcion lee tam bytes del socket sock y lo almacena en el arreglo apuntado por buffer.

Parámetros

int	sock Descriptor del socket de donde se va a leer.
u_long	tam Número de bytes a leer.
unsigned	char *buffer Apuntador al arreglo en donde se van a escribir los datos leidos

Devuelve

La funcion regresa -1 en caso de que haya habido un error en la lectura, y 0 en caso de exito.

Definición en la línea 8 del archivo SockIO.cpp.

```
9 {
10          u_long leeidos = 0;
11          int cnt, leer;
12
13          //double tiempo;
14
15          //  tiempo = clock()*1.0/CLOCKS_PER_SEC;
16          do
17          {
18                          // max chuncks of MAXCHUNK bytes
19                          if (tam - leeidos > MAXCHUNK)
20                                  leer = MAXCHUNK;
```

```
21                          else
22                                  leer = tam - leeidos;
23
24                          if ((cnt = read (sock, buffer + leeidos, leer)) < 0)
25                          {
26                                  fprintf (stderr, "ERROR FATAL. falla en read %d, EA=%d\n", errno,
27                                                                                  EAGAIN);
28                                  fprintf (stderr, "EI=%d EIO=%d\n", EINTR, EIO);
29                                  // close(sock);
30                                  /* no necesariamente hay que cerrar */
31                                  printf ("Error, Didn't read all the bytes");
32                                  return -1;                                                  //
      exit(5);
33                          }
34                          else
35                          {
36                                  leeidos += cnt;
37                          }
38          }
39          while (leeidos < tam);
40          return 0;
41  }
```

9.10.3.2. int Write (int *sock*, u_long *tam*, unsigned char * *buffer*)

Esta función escribe tam bytes, que se encuentran almacenados en la cadena de caracteres referida por el apuntador buffer en el socket identificado por sock.

Parámetros

int	sock Descriptor del socket de donde se va a escribir.
u_long	tam Número de bytes a escribir.
unsigned	char *buffer Apuntador al arreglo en donde se van a tomar los datos a enviar.

Devuelve

La funcion regresa -1 en caso de que haya habido un error en la escritura, y 0 en caso de exito.

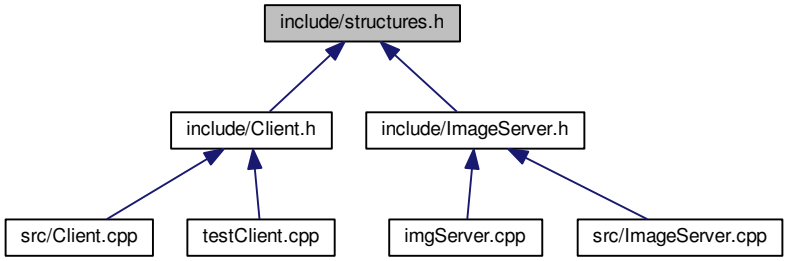
Definición en la línea 43 del archivo SockIO.cpp.

```
44  {
45          u_long escritos = 0;
46          int cnt, escribe;
47
48          do
49          {
50                          // max chuncks of MAXCHUNK bytes
51                          if (tam - escritos > MAXCHUNK)
52                                  escribe = MAXCHUNK;
53                          else
54                                  escribe = tam - escritos;
55
56                          if ((cnt = write (sock, buffer + escritos, escribe)) < 0)
57                          {
58                                  fprintf (stderr, "ERROR FATAL. falla en write %d\n", errno);
59                                  // close(sock);
60                                  /* no necesariamente hay que cerrar */
61                                  printf ("Error, Didn't write all the bytes");
62                                  return -1;                                                  //
      exit(5);
63                          }
64                          else
65                          {
66                                  escritos += cnt;
67                          }
68          }
69          while (escritos < tam);
70          return 0;
71  }
```

9.11. Referencia del Archivo include/structures.h

En este archivo se define la estructura ImageInfo, que se utiliza en el servidor de imágenes, así como se definen varios valores por defecto.

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- struct ImageInfo

  Esta clase define la estructura ImagenInfo, que define el encabezado que se utiliza para la transmisión de imágenes.

'defines'

- #define FORMAT_ERROR 3
- #define MSG_LENGTH 10
- #define SEND_FAILURE 0
- #define SEND_SUCCESS 1

9.11.1. Descripción detallada

En este archivo se define la estructura ImageInfo, que se utiliza en el servidor de imágenes, así como se definen varios valores por defecto.

Definición en el archivo structures.h.

9.11.2. Documentación de los 'defines'

9.11.2.1. #define FORMAT_ERROR 3

Definición en la línea 12 del archivo structures.h.

9.11.2.2. #define MSG_LENGTH 10

Definición en la línea 14 del archivo structures.h.

9.11.2.3. #define SEND_FAILURE 0

Definición en la línea 9 del archivo structures.h.

9.11.2.4. #define SEND_SUCCESS 1

Definición en la línea 10 del archivo structures.h.

9.12. Referencia del Archivo README.dox

9.13. Referencia del Archivo README.md

9.14. Referencia del Archivo remoteFrame.py

Clases

- class remoteFrame

Namespaces

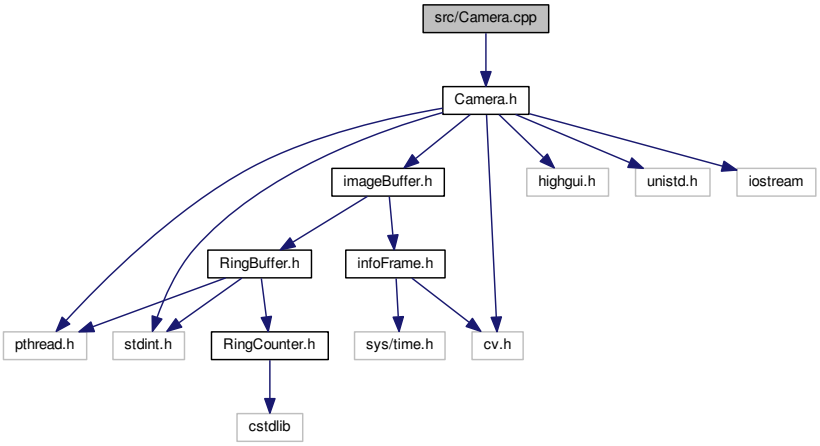
- remoteFrame

9.15. Referencia del Archivo src/Camera.cpp

Este archivo contiene el código que de los métodos de la clase Camara.

```
#include <Camera.h>
```

Dependencia gráfica adjunta para Camera.cpp:



9.15.1. Descripción detallada

Este archivo contiene el código que de los métodos de la clase Camara.

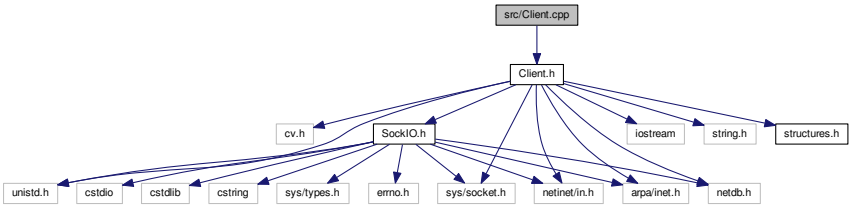
Definición en el archivo Camera.cpp.

9.16. Referencia del Archivo src/Client.cpp

Este archivo contiene el código que de los métodos de la clase Client.

```
#include <Client.h>
```

Dependencia gráfica adjunta para Client.cpp:



9.16.1. Descripción detallada

Este archivo contiene el código que de los métodos de la clase Client.

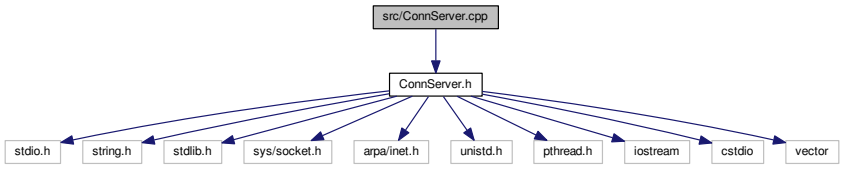
Definición en el archivo Client.cpp.

9.17. Referencia del Archivo src/ConnServer.cpp

Este archivo contiene el código que de los métodos de la clase ConnServer.cpp.

```
#include <ConnServer.h>
```

Dependencia gráfica adjunta para ConnServer.cpp:



9.17.1. Descripción detallada

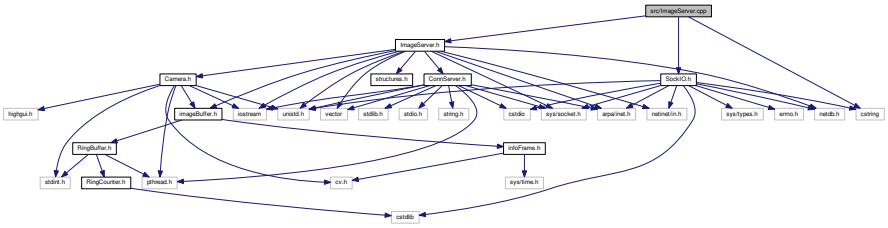
Este archivo contiene el código que de los métodos de la clase ConnServer.cpp.

Definición en el archivo ConnServer.cpp.

9.18. Referencia del Archivo src/ImageServer.cpp

```
#include <ImageServer.h>
#include <SockIO.h>
#include <cstring>
```

Dependencia gráfica adjunta para ImageServer.cpp:



Funciones

- void paintMaze (Mat &I, Mat &Mask)

  Aplica una mascara a color a una imagen. En aquellos pixeles de la máscara son diferentes a RGB[255,255,255] (color blanco), el pixel correspondiente de la imagen se sustituye por el valor de la máscara. De caso contrario, el valor original del pixel, se preserva,.

9.18.1. Documentación de las funciones

9.18.1.1. void paintMaze (Mat & *I*, Mat & *Mask*)

Aplica una mascara a color a una imagen. En aquellos pixeles de la máscara son diferentes a RGB[255,255,255] (color blanco), el pixel correspondiente de la imagen se sustituye por el valor de la máscara. De caso contrario, el valor original del pixel, se preserva,.

Parámetros

Mat	&I Matriz que contiene la imagen a Enmascarar.
Mat	&Mask Mascara que se debe a plicar a la imagen I.

Definición en la línea 65 del archivo ImageServer.cpp.

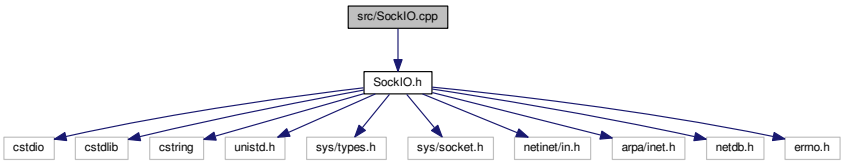
```
66 {
67     int i, j;
68     Vec3b *ptrI, *ptrM;
69     Vec3b val(255,255,255);
70 
71     assert (I.rows == Mask.rows && I.cols == Mask.cols);
72     for (i=0;i<I.rows;++i)
73     {
74         ptrI = I.ptr<Vec3b>(i);
75         ptrM = Mask.ptr<Vec3b>(i);
76         for (j=0;j<I.cols;++j,++ptrI,++ptrM)
77             if (*ptrM != val)
78                 *ptrI = *ptrM;
79     }
80 }
```

9.19. Referencia del Archivo src/SockIO.cpp

Este archivo contiene el código que de los funciones Read y Write definidias en SockIO.h.

```
#include <SockIO.h>
```

Dependencia gráfica adjunta para SockIO.cpp:



Funciones

- int Read (int sock, u_long tam, unsigned char ∗buffer)

  Esta funcion lee tam bytes del socket sock y lo almacena en el arreglo apuntado por buffer.
- int Write (int sock, u_long tam, unsigned char ∗buffer)

  Esta función escribe tam bytes, que se encuentran almacenados en la cadena de caracteres referida por el apuntador buffer en el socket identificado por sock.

9.19.1. Descripción detallada

Este archivo contiene el código que de los funciones Read y Write definidias en SockIO.h.

Definición en el archivo SockIO.cpp.

9.19.2. Documentación de las funciones

9.19.2.1. int Read (int *sock,* u_long *tam,* unsigned char ∗ *buffer*)

Esta funcion lee tam bytes del socket sock y lo almacena en el arreglo apuntado por buffer.

Parámetros

int	sock Descriptor del socket de donde se va a leer.
u_long	tam Número de bytes a leer.
unsigned	char ∗buffer Apuntador al arreglo en donde se van a escribir los datos leidos

Devuelve

La funcion regresa -1 en caso de que haya habido un error en la lectura, y 0 en caso de exito.

Definición en la línea 8 del archivo SockIO.cpp.

```
9 {
10          u_long leeidos = 0;
11          int cnt, leer;
12
13          //double tiempo;
```

```
14
15          //  tiempo = clock()*1.0/CLOCKS_PER_SEC;
16          do
17          {
18                          // max chuncks of MAXCHUNK bytes
19                          if (tam - leeidos > MAXCHUNK)
20                                  leer = MAXCHUNK;
21                          else
22                                  leer = tam - leeidos;
23
24                          if ((cnt = read (sock, buffer + leeidos, leer)) < 0)
25                          {
26                                  fprintf (stderr, "ERROR FATAL. falla en read %d, EA=%d\n", errno,
27                                                                                  EAGAIN);
28                                  fprintf (stderr, "EI=%d EIO=%d\n", EINTR, EIO);
29                                  // close(sock);
30                                  /* no necesariamente hay que cerrar */
31                                  printf ("Error, Didn't read all the bytes");
32                                  return -1;                                                      //
      exit(5);
33                          }
34                          else
35                          {
36                                  leeidos += cnt;
37                          }
38          }
39          while (leeidos < tam);
40          return 0;
41  }
```

9.19.2.2. int Write (int *sock*, u_long *tam*, unsigned char * *buffer*)

Esta función escribe tam bytes, que se encuentran almacenados en la cadena de caracteres referida por el apuntador buffer en el socket identificado por sock.

Parámetros

int	sock Descriptor del socket de donde se va a escribir.
u_long	tam Número de bytes a escribir.
unsigned	char *buffer Apuntador al arreglo en donde se van a tomar los datos a enviar.

Devuelve

La funcion regresa -1 en caso de que haya habido un error en la escritura, y 0 en caso de exito.

Definición en la línea 43 del archivo SockIO.cpp.

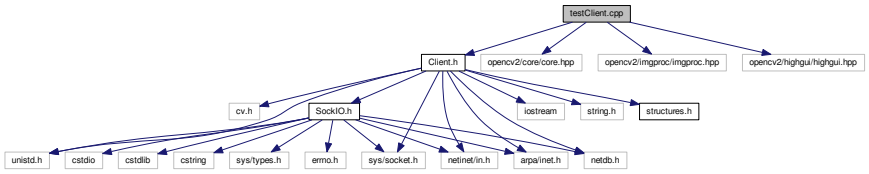
```
44  {
45          u_long escritos = 0;
46          int cnt, escribe;
47
48          do
49          {
50                          // max chuncks of MAXCHUNK bytes
51                          if (tam - escritos > MAXCHUNK)
52                                  escribe = MAXCHUNK;
53                          else
54                                  escribe = tam - escritos;
55
56                          if ((cnt = write (sock, buffer + escritos, escribe)) < 0)
57                          {
58                                  fprintf (stderr, "ERROR FATAL. falla en write %d\n", errno);
59                                  // close(sock);
60                                  /* no necesariamente hay que cerrar */
61                                  printf ("Error, Didn't write all the bytes");
62                                  return -1;                                                      //
      exit(5);
63                          }
64                          else
65                          {
```

```
66                                  escritos += cnt;
67                          }
68          }
69          while (escritos < tam);
70          return 0;
71 }
```

9.20. Referencia del Archivo testClient.cpp

```
#include <Client.h>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
```

Dependencia gráfica adjunta para testClient.cpp:



Funciones

- int main (int argc, char **argv)

9.20.1. Documentación de las funciones

9.20.1.1. int main (int *argc*, char ** *argv*)

Definición en la línea 11 del archivo testClient.cpp.

```
12 {
13          int port = 8888;
14          char ipAddress[64] = "127.0.0.1";
15
16          if (argc > 1)
17          {
18                          strncpy (ipAddress, argv[1], 63);
19                          if (argc > 2)
20                                  port = atoi (argv[2]);
21          }
22
23          Client *client = new Client (port, ipAddress);
24          client->connectSocket ();
25
26          Mat frame;
27          while (true)
28          {
29                          client->getFrame (frame);
30                          imshow ("rec", frame);
31                          if (waitKey (30) >= 0)
32                                  break;
33          }
34
35
36          return 0;
37 }
```

9.21. Referencia del Archivo testClient.py

Namespaces

- testClient

Variables

- string address = '127.0.0.1'
- tuple argc = len(sys.argv)
- tuple img = rF.getFrame()
- Mask = None
- int port = 8888
- tuple rF = remoteFrame(address, port, Mask)

Índice alfabético