



Visión Artificial

Prácticas: entorno de desarrollo

- ▼ OpenCV
- ▼ Proyecto software

OpenCV

- ▼ Introducción
- ▼ Estructuras básicas
- ▼ Módulos

OpenCV: introducción

- ▼ Open Source Computer Vision Library.
- ▼ Incluye un amplio número de funciones relacionadas con la visión artificial.
- ▼ Continuo desarrollo.
- ▼ Disponible para varias plataformas.
- ▼ Uso muy extendido.
- ▼ Versión 4.5.1
- ▼ Web: <http://opencv.org>
- ▼ Documentación: <http://docs.opencv.org/>

OpenCV: introducción

- ▼ Estructura modular: diferentes librerías para diferentes grupos de funciones.
 - ▼ *core*: estructuras de datos básicas y funciones genéricas usadas por otros módulos
 - ▼ *imgproc*: procesamiento de imágenes.
 - ▼ *imgcodecs*: lectura y escritura de imágenes en fichero
 - ▼ *videoio*: entrada/salida de vídeo
 - ▼ *highgui*: funciones de gestión de GUIs y de entrada/salida.
 - ▼ *video*: análisis de vídeo.
 - ▼ *calib3d*: calibración de cámara y geometría multi-vista.
 - ▼ *features2d*: características de imagen.
 - ▼ *objdetect*: detección de objetos.
 - ▼ ...

OpenCV: estructuras básicas (la clase *Mat*)

- ▼ Definición de arrays multidimensionales.
- ▼ Clase básica para almacenar y manejar imágenes.
- ▼ Consta de una cabecera y un puntero a la zona de memoria que contiene la imagen.
- ▼ Creación de un objeto *Mat*:
 - ▼ Constructor *Mat(filas, columnas, tipo [, valor inicial])*
 - ▼ *tipo*: *CV_[bits][S/U/F]C[canales]* (ej: *CV_8UC3*)
 - ▼ Método *create(filas, columnas, tipo)*
 - ▼ Asignación de otro objeto *Mat*: sólo copia la cabecera, no los datos (varios objetos apuntando a la misma imagen).
 - ▼ Métodos *clone()* y *copyTo(destino)*: realiza una copia completa del objeto (cabecera y datos).

OpenCV: estructuras básicas (la clase *Mat*)

- ▼ Atributos principales:
 - ▼ *int rows, cols*: filas y columnas del array (2 dimensiones)
 - ▼ *int dims*: número de dimensiones del array.
 - ▼ *uchar * data*: puntero a la zona de memoria que contiene los datos del array.
 - ▼ *MStep step*: array que contiene la distancia en bytes de dos elementos consecutivos en una determinada dimensión.
- ▼ Acceso a un elemento de la matriz:
 - ▼ Método *at<T>(...)*: $v = M.at<uchar>(f, c); v = M.at<uchar>(p)$
 - ▼ Dirección del elemento $(i_0, i_1, \dots, i_N) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + \dots + M.step[N]*i_N$
 - ▼ Píxel situado en $(f, c) = M.data[M.step[0]*f + M.step[1]*c]$

OpenCV: estructuras básicas (la clase *Mat*)

▼ Algunos métodos:

- ▼ *row(...)*: crea una matriz con la fila especificada (*).
- ▼ *col(...)*: crea una matriz con la columna especificada (*).
- ▼ *rowRange(...)*: crea una matriz con el rango de filas indicado (*).
- ▼ *colRange(...)*: crea una matriz con el rango de columnas indicado (*).
- ▼ *convertTo(...)*: convierte el tipo de los elementos de la matriz al indicado por parámetro.
- ▼ *setTo(...)*: inicializa todos los elementos al valor especificado.
- ▼ *resize(...)*: modifica el número de filas de la matriz.
- ▼ *release()*: libera, si es posible, la memoria ocupada por los datos de la matriz.

(*) *sólo genera la cabecera*

OpenCV: otras estructuras básicas

- ▼ *Point_*: plantilla para definir puntos 2D (*Point2i*, *Point*, *Point2f*). Las coordenada se representan a través de los atributos *x* e *y*.
- ▼ *Point3_*: plantilla para definir puntos 3D (*Point3i*, *Point3f*, *Point3d*). Coordenadas en *x*, *y* y *z*.
- ▼ *Size_*: plantilla para especificar el tamaño de una imagen o un rectángulo (*Size2i*, *Size*, *Size2f*). Tamaño en *width* y *height*.
- ▼ *Rect_*: plantilla para definir regiones de interés de una imagen.
- ▼ *Matx*: plantilla para definir matrices de pequeño rango con tamaño y tipo conocidos en tiempo de compilación (*Matx33f*).
- ▼ *Vec*: plantilla para definir vectores numéricos de pocos elementos (*Vec2i*, *Vec3b*).
- ▼ *Scalar_*: plantilla para definir vectores de 4 elementos: *Vec<T, 4>*
- ▼ *Range*: clase que permite especificar un rango continuo de valores. Los límites del rango en *start* y *end*.

OpenCV: *InputArray* *OutputArray*

- ▼ Clases *proxy* utilizadas como parámetros en muchas funciones de OpenCV para representar arrays genéricos.
- ▼ *InputArray*: parámetro de entrada (sólo lectura).
- ▼ *OutputArray*: parámetro de salida.
- ▼ No es necesario definir objetos de estos tipos de manera explícita.
- ▼ Se utilizan como clases intermediarias de *Mat*, *std::vector<>*, *Matx<>*, *Vec<>* o *Scalar*.
- ▼ Si el parámetro es opcional y no se desea especificar ninguno, debe indicarse *cv::noarray*.
- ▼ Otras clases similares: *InputOutputArray*, *InputArrayOfArrays*, *OutputArrayOfArrays*.

OpenCV: módulos - *core*

- ▼ Definición de clases (C++), estructuras (C) y operaciones básicas.
- ▼ Operaciones sobre matrices (imágenes) y vectores:
 - ▼ *void **absdiff**(InputArray src1, InputArray src2, OutputArray dst):* valor absoluto de la diferencia entre *src1* y *src2*.
 - ▼ *void **compare**(InputArray src1, InputArray src2, OutputArray dst, int cmpop):* compara *src1* y *src2* según el operador indicado en *cmpop* (resultado 0 ó 255).
 - ▼ *double **invert**(InputArray src, OutputArray dst, int method=DECOMP_LU):* calcula la inversa de *src*.
 - ▼ *bool **solve**(InputArray src1, InputArray src2, OutputArray dst, int flags=DECOMP_LU):* resuelve el sistema de ecuaciones $src1 * X = src2$.

OpenCV: módulos - *imgproc*

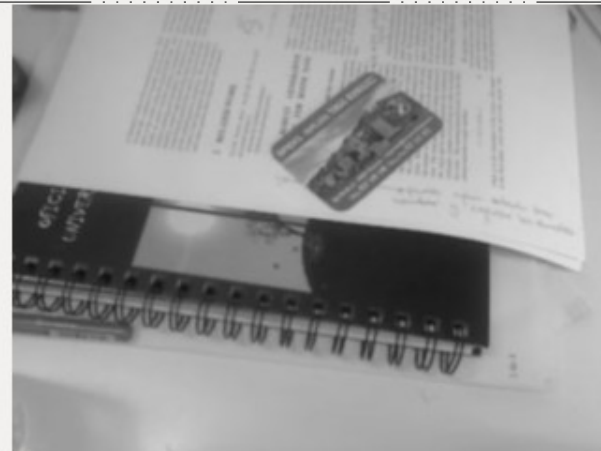
- ▼ Procesamiento de imágenes.
- ▼ **Operaciones de filtrado:** operaciones sobre el entorno local de un píxel para suavizado, realce, ...
 - ▼ Filtros lineales:
 - ▼ *void **filter2D**(InputArray src, OutputArray dst, int ddepth, InputArray kernel, Point anchor=Point(-1,-1), double delta=0, int borderType=BORDER_DEFAULT):* convolución con el kernel especificado.
 - ▼ *void **blur**(InputArray src, OutputArray dst, Size ksize, Point anchor=Point(-1,-1), int borderType = BORDER_DEFAULT):* aplica el filtro media utilizando un kernel del tamaño especificado por *ksize*.
 - ▼ *void **GaussianBlur**(InputArray src, OutputArray dst, Size ksize, double sigmaX, double sigmaY=0, int borderType=BORDER_DEFAULT):* filtro gaussiano

OpenCV: módulos - *imgproc*

```
Size kSize(5,5);  
blur(grayImage, destImage, kSize);
```



```
Size kSize(5,5);  
GaussianBlur(grayImage, destImage, kSize, 0.8, 0.8);
```



OpenCV: módulos - *imgproc*

▼ Operaciones de filtrado:

▼ Filtros no lineales:

- ▼ *void **medianBlur**(InputArray src, OutputArray dst, int ksize):* suavizado mediante el filtro mediana.

▼ **Transformaciones geométricas:** modifican la posición de los píxels dentro de la imagen.

- ▼ *void **remap**(InputArray src, OutputArray dst, InputArray mapX, InputArray mapY, int interpolation, int borderMode=BORDER_CONSTANT, const Scalar& borderValue=Scalar()):* construye la imagen *dst* copiando a cada posición (x,y) el píxel de *src* situado en la posición indicada por *mapX(x,y)* y *mapY(x,y)*.

OpenCV: módulos - *imgproc*

▼ Transformaciones geométricas:

- ▼ *void **warpAffine**(InputArray src, OutputArray dst, InputArray M, Size dsize, int flags=INTER_LINEAR, int borderMode=BORDER_CONSTANT, const Scalar& borderValue=Scalar()):* aplica una transformación afín a *src*.

```
Size sDest(320, 240);  
MT(0,0) = cos(0.3); MT(0,1) = sin(0.3); MT(0,2) = -20;  
MT(1,0) = -sin(0.3); MT(1,1) = cos(0.3); MT(1,2) = 10;  
warpAffine(grayImage, destImage, MT, sDest);
```

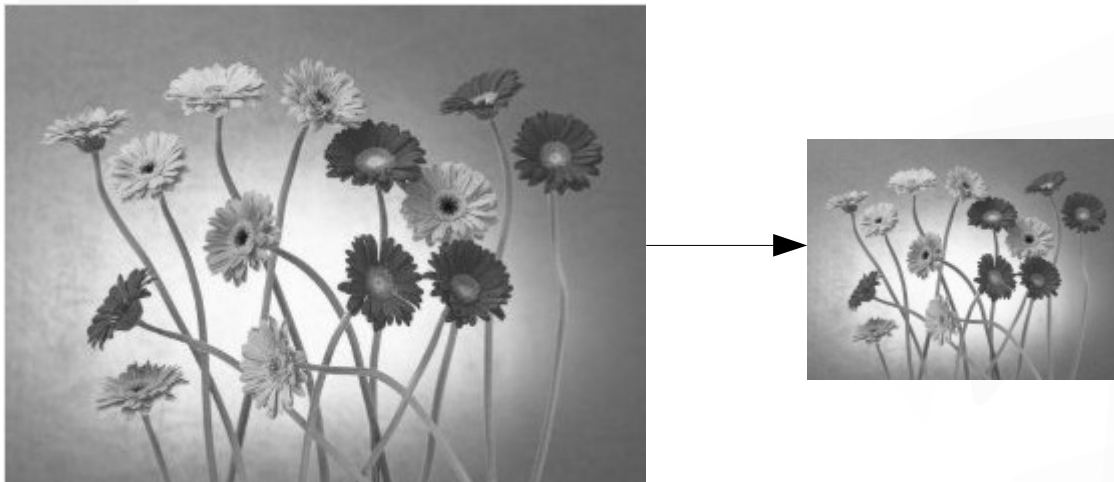


OpenCV: módulos - *imgproc*

▼ Transformaciones geométricas:

- ▼ *void **resize**(InputArray src, OutputArray dst, Size dsize, double fx=0, double fy=0, int interpolation=INTER_LINEAR)*:
redimensiona la imagen *src* al tamaño especificado en *dsize* o aplicando el factor de escala indicado en *fx* y *fy*. La imagen redimensionada se almacena en *dst*.

```
resize(grayImage, destImage, Size(160, 120));
```

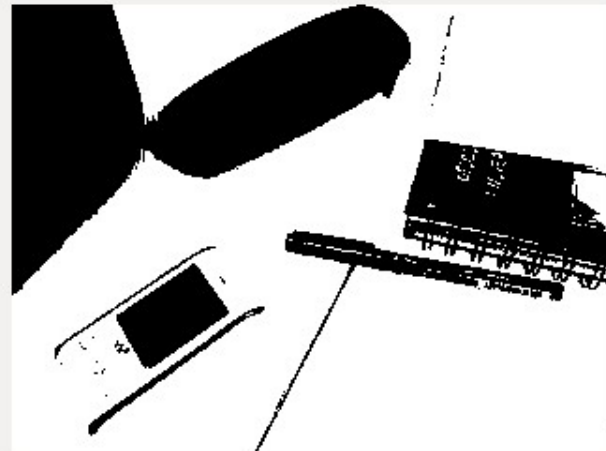


OpenCV: módulos - *imgproc*

▼ Otras transformaciones:

- ▼ *void **cvtColor**(InputArray src, OutputArray dst, int code, int dstCn=0)*: transforma el espacio de color de *src* según la conversión indicada en *code* (p.e., `COLOR_RGB2GRAY`).
- ▼ *double **threshold**(InputArray src, OutputArray dst, double thresh, double maxVal, int thresholdType)*: transforma los píxeles de *src* aplicando el umbral indicado.

```
threshold(grayImage, binaryImage, 120, 255, CV_THRESH_BINARY);
```



OpenCV: módulos - *imgproc*

▼ Operaciones con histogramas:

- ▼ *void **equalizeHist**(InputArray src, OutputArray dst):* ecualiza el histograma de *src*.

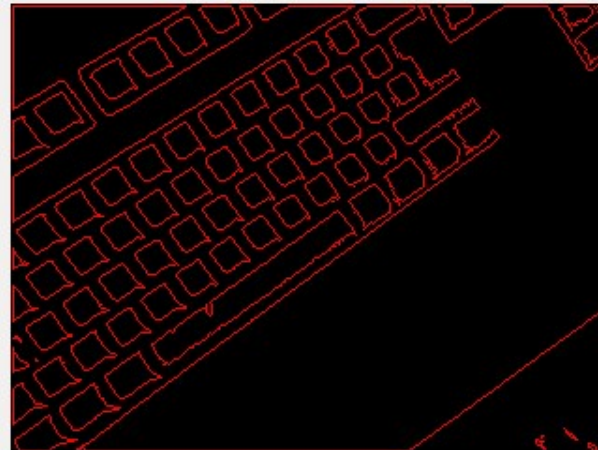
```
equalizeHist(grayImage, destImage);
```



OpenCV: módulos - *imgproc*

- ▼ **Análisis estructural:** detección de contornos, aproximaciones poligonales, propiedades de formas, ...
- ▼ *void **findContours**(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point()):* detecta los contornos de una imagen binaria. Cada contorno es representado a través de un array de puntos.

```
std::vector< std::vector<Point2i> > contours;  
findContours(binaryImage, contours, noArray(),  
CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);
```



OpenCV: módulos - *imgproc*

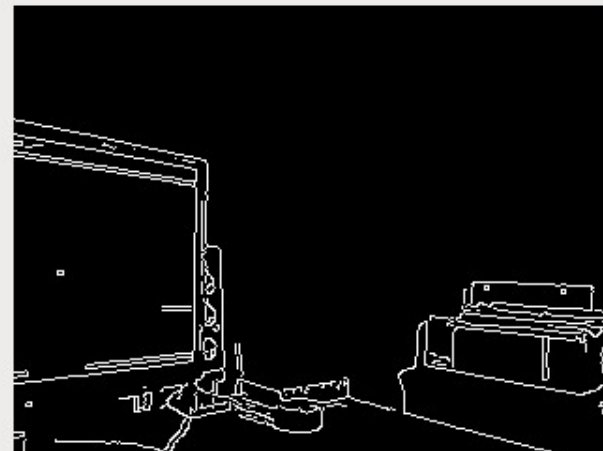
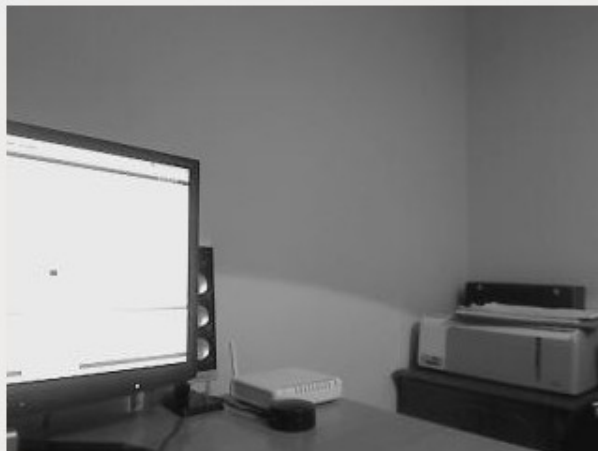
▼ Análisis estructural:

- ▼ *void **fitLine**(InputArray points, OutputArray line, int distType, double param, double reps, double aeps):* ajusta el conjunto de puntos indicado a una línea, minimizando la distancia de los puntos a la línea según la función indicada en *distType*.
- ▼ *void **convexHull**(InputArray points, OutputArray hull, bool clockwise=false, bool returnPoints=true):* calcula la envolvente convexa que contiene el conjunto de puntos indicado.
- ▼ *void **approxPolyDP**(InputArray curve, OutputArray approxCurve, double epsilon, bool closed):* aproxima el contorno indicado a un polígono.
- ▼ *bool **isContourConvex**(InputArray contour):* determina si el contorno indicado es o no convexo.

OpenCV: módulos - *imgproc*

- ▼ **Detección de características:** características de bajo nivel (bordes, esquinas, líneas, ...)
- ▼ *void **Canny**(InputArray image, OutputArray edges, double threshold1, double threshold2, int apertureSize=3, bool L2gradient=false)*: detecta los bordes de la imagen *image*, siguiendo el algoritmo de *Canny*, y marca los puntos correspondientes en la imagen *edges*.

```
Canny(grayImage, destImage, 120., 50.);
```



OpenCV: módulos - *imgproc*

▼ Detección de características:

- ▼ ***void goodFeaturesToTrack***(*InputArray image*, *OutputArray corners*, *int maxCorners*, *double qualityLevel*, *double minDistance*, *InputArray mask=noArray()*, *int blockSize=3*, *bool useHarrisDetector=false*, *double k=0.04*): detecta esquinas muy distintivas en la imagen indicada. El número máximo será el indicado en *maxCorners*.

```
std::vector<Point2f> corners;  
goodFeaturesToTrack(grayImage, corners, 50, 0.01, 10);
```

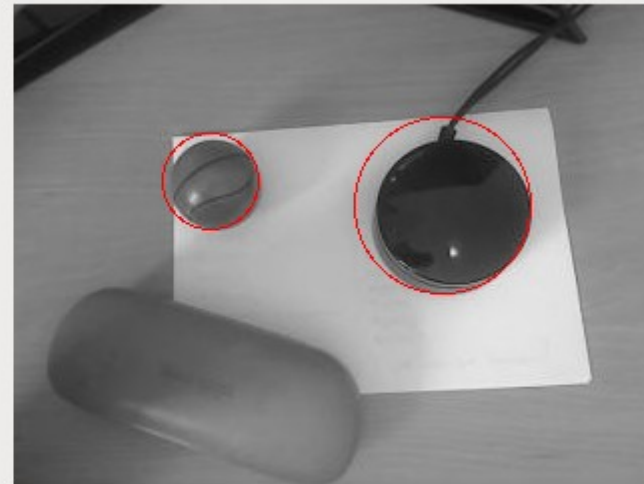
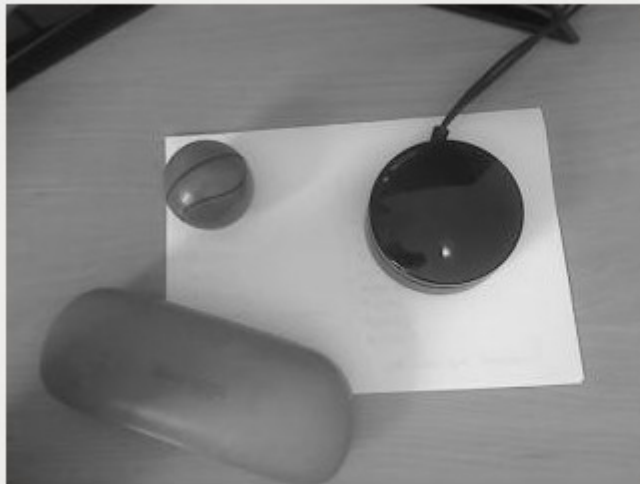


OpenCV: módulos - *imgproc*

▼ Detección de características:

- ▼ ***void HoughCircles***(*InputArray image*, *OutputArray circles*, *int method*, *double dp*, *double minDist*, *double param1=100*, *double param2=100*, *int minRadius=0*, *int maxRadius=0*): detecta los círculos en *image* utilizando la transformada de Hough. Cada círculo se representa como un vector de 3 elementos (x, y, radio).

```
std::vector<Vec3f> circles;  
HoughCircles(grayImage, circles, CV_HOUGH_GRADIENT, 2, 20, 200, 100);
```



OpenCV: módulos - *imgproc*

▼ Funciones de dibujo:

- ▼ *void **line**(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int lineType=8, int shift=0):* dibuja una línea sobre *img* delimitada por *pt1* y *pt2*.
- ▼ *void **putText**(Mat& img, const string& text, Point org, int fontFace, double fontScale, Scalar color, int thickness=1, int lineType=8, bool bottomLeftOrigin=false):* imprime una cadena de texto (*text*) sobre *img*.

▼ Utilidades y funciones del sistema:

- ▼ *int **cvFloor**(double value):* redondea *value* al entero menor más cercano.
- ▼ *void **setNumThreads**(int nthreads):* inicializa el número de hilos utilizados por OpenCV.

OpenCV: módulos - *imgcodecs*

▼ Operaciones de E/S sobre fichero:

- ▼ *Mat **imread**(const string& filename, int flags=IMREAD_COLOR):* lee una imagen de fichero.
- ▼ *bool **imwrite**(const string& filename, InputArray image, const vector<int>& params=vector<int>()):* almacena la imagen indicada en fichero. El formato queda especificado por la extensión del fichero.

OpenCV: módulos - *videoio*

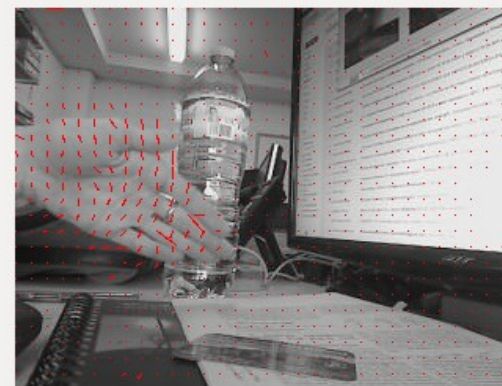
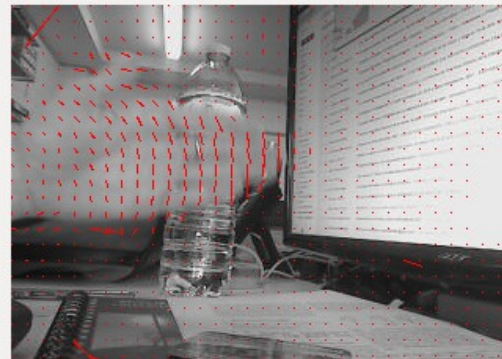
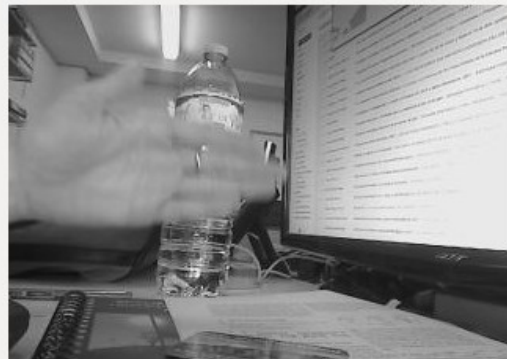
- ▼ Lectura/escritura de vídeo o de secuencia de imágenes.
- ▼ Captura de vídeo:
 - ▼ La clase ***VideoCapture***: clase de captura de vídeo desde cámara o desde fichero.
 - ▼ Constructores:
 - ▼ *VideoCapture(const string& filename)*
 - ▼ *VideoCapture(int device)*
 - ▼ Establecer y recuperar propiedades de la captura:
 - ▼ *bool set(int propertyId, double value)*
 - ▼ *double VideoCapture::get(int propId)*
 - ▼ Capturar:
 - ▼ *VideoCapture& operator>>(Mat& image)*
 - ▼ *bool read(Mat& image)*

OpenCV: módulos - *video*

- ▼ Análisis de una secuencia de imágenes: estimación del movimiento y seguimiento de objetos.
- ▼ *void **calcOpticalFlowPyrLK**(InputArray prevImg, InputArray nextImg, InputArray prevPts, InputOutputArray nextPts, OutputArray status, OutputArray err, ...)*: calcula el flujo óptico del conjunto de puntos indicado en *prevPts* a partir de dos instantes de captura reflejados en *prevImg* y *nextImg*.

```
VideoCapture *cap;  
std::vector<Point2f> prevPts, nextPts;  
Mat status, err;  
...  
for(int r=0; r<240; r+=10)  
    for(int c=0; c<320; c+=10)  
        prevPts.push_back(Point2f(c,r));  
  
nextPts = prevPts;  
...  
grayImage.copyTo(prevImage);  
*cap >> grayImage;  
calcOpticalFlowPyrLK(prevImage, grayImage, prevPts, nextPts, status, err);
```

OpenCV: módulos - *video*



OpenCV: módulos - *calib3d*

- ▼ Calibración de cámara y reconstrucción 3D.
- ▼ **Operaciones de calibración:**
 - ▼ *double **calibrateCamera**(InputArrayOfArrays objectPoints, InputArrayOfArrays imagePoints, Size imageSize, InputOutputArray cameraMatrix, InputOutputArray distCoeffs, OutputArrayOfArrays rvecs, OutputArrayOfArrays tvecs, int flags=0)*: determina los parámetros intrínsecos y extrínsecos de la cámara.
- ▼ **Estimación de transformaciones:**
 - ▼ *Mat **findFundamentalMat**(InputArray points1, InputArray points2, ...)*: determina la matriz fundamental entre dos vistas.
 - ▼ *Mat **findHomography**(InputArray srcPoints, InputArray dstPoints, ...)*: estima la homografía que relaciona dos vistas de un mismo plano.

OpenCV: módulos - *calib3d*

▼ Estimación de 3D:

- ▼ Las clases **StereoBM** y **StereoSGBM**: realizan el cálculo de correspondencias entre las dos imágenes de un par estéreo. Proporcionan una imagen de disparidad (diferencia de posición entre cada par de puntos homólogos).
- ▼ *void **reprojectImageTo3D**(InputArray disparity, OutputArray _3dImage, InputArray Q, bool handleMissingValues=false, int depth=-1)*: cálculo del 3D a partir de una imagen de disparidad.

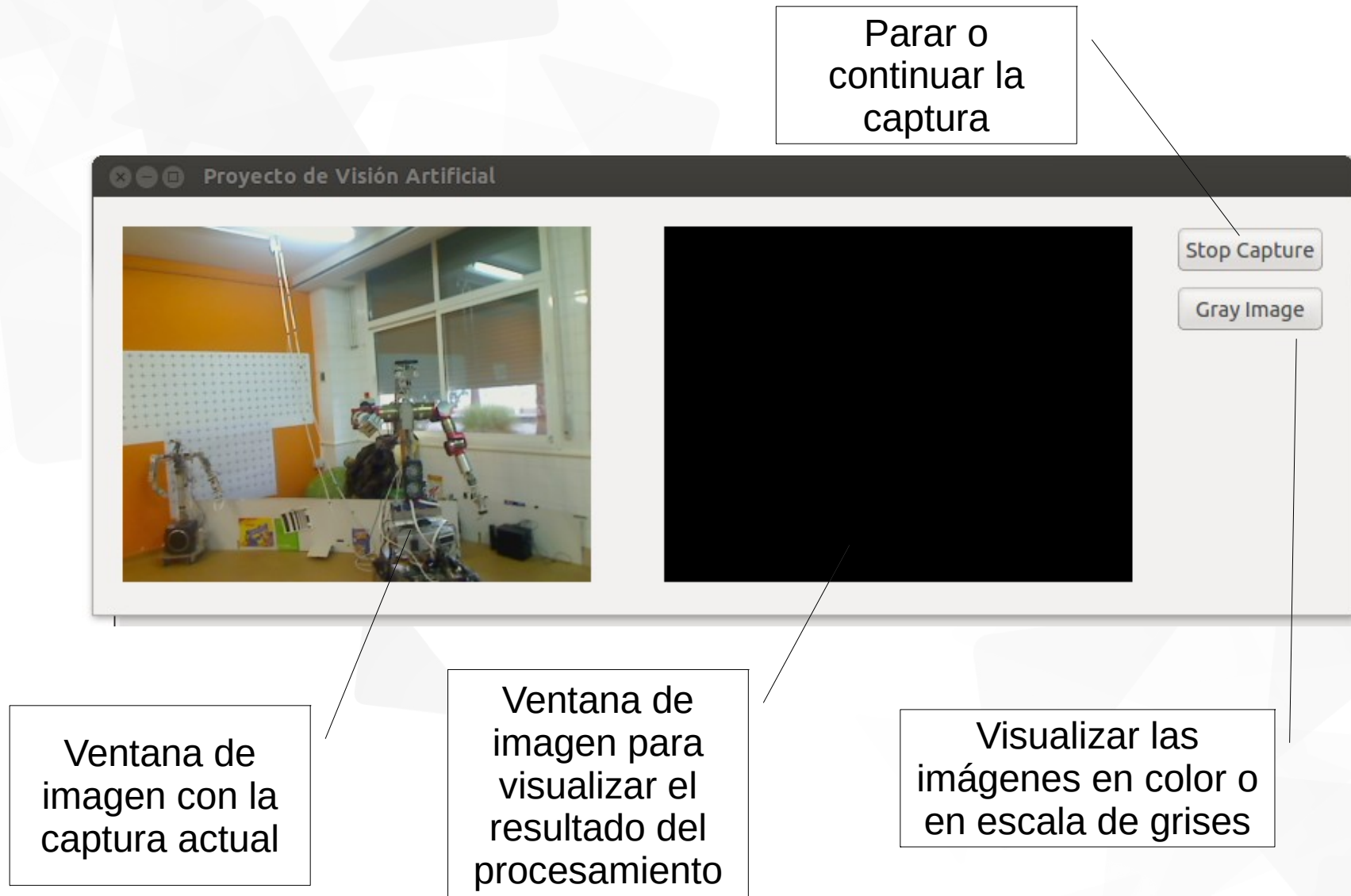
Proyecto software

- ▼ Descripción
- ▼ Interfaz de usuario
- ▼ Estructura del proyecto

Proyecto software: descripción

- ▼ Proyecto Qt: `proyVA.pro`
- ▼ Entorno de desarrollo: Qt Creator
- ▼ Paquetes y librerías necesarias:
 - ▼ *qtcreator*: IDE de Qt
 - ▼ *qt5-default*, *qttools5-dev-tools*: librerías y herramientas de Qt5
 - ▼ OpenCV:
 - ▼ Instalación desde paquetes: *libopencv-dev*
 - ▼ Versión 3.5 en Ubuntu 18.04
 - ▼ Versión 4.2 en Ubuntu 20.04
 - ▼ Instalación desde fuentes: <http://opencv.org>

Proyecto software: interfaz de usuario



Proyecto software: estructura del proyecto

- ▼ Fichero de descripción proyecto: *proyVA.pro*
- ▼ Formulario de Qt: *mainwindow.ui*
- ▼ Clases:
 - ▼ *MainWindow*: clase principal encargada de la gestión del proyecto. Contiene el formulario de Qt y realiza el control de los posibles eventos.
 - ▼ *ImgViewer*: visor de imágenes con diversos métodos de dibujo.

Proyecto software: estructura del proyecto

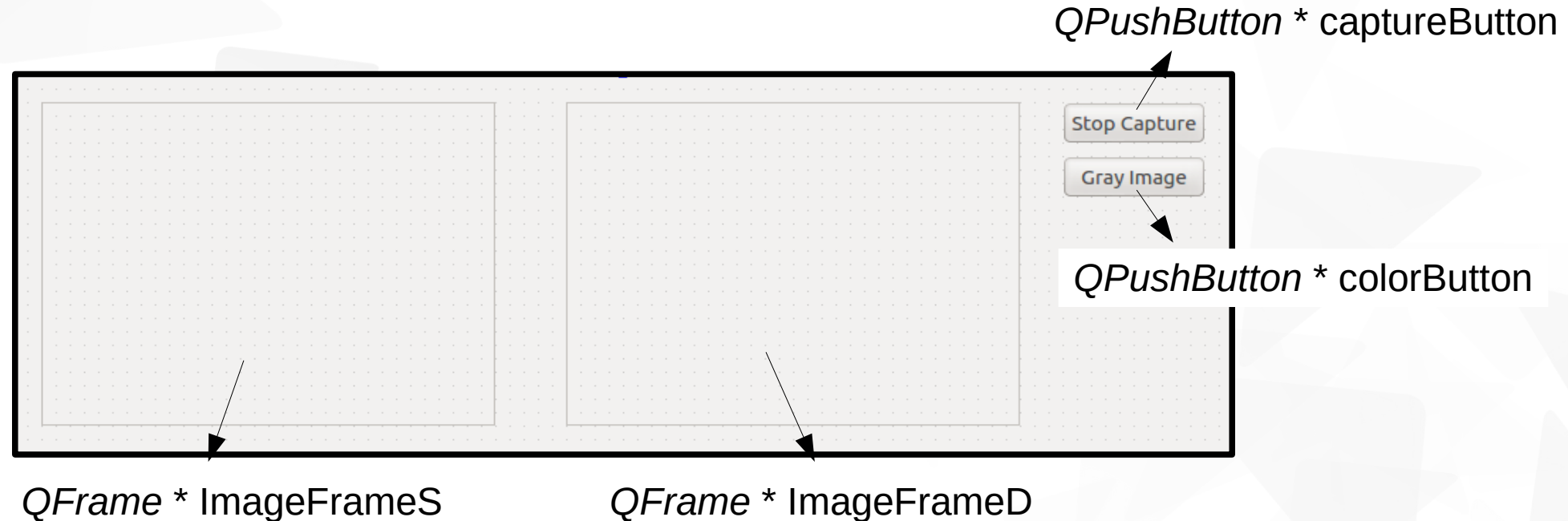
ImgViewer

- ▼ Proporciona métodos para visualizar imágenes y dibujar diferentes elementos sobre ellas.
- ▼ Métodos de interés:
 - ▼ *void drawSquare(const QPoint &, int sideX, int sideY, const QColor &, bool fill=false, int id= -1, float rads=0, float width=0):* dibuja el cuadrado especificado del color indicado por parámetro.
 - ▼ Ejemplo: `drawSquare(QPoint(200, 120), 50, 40, Qt::green)`
 - ▼ *void drawLine(const QLine &line, const QColor & c, float width=0):* dibuja la línea especificada del color indicado
 - ▼ Ejemplo: `drawLine(QLine(20, 12, 180, 110), Qt::red)`

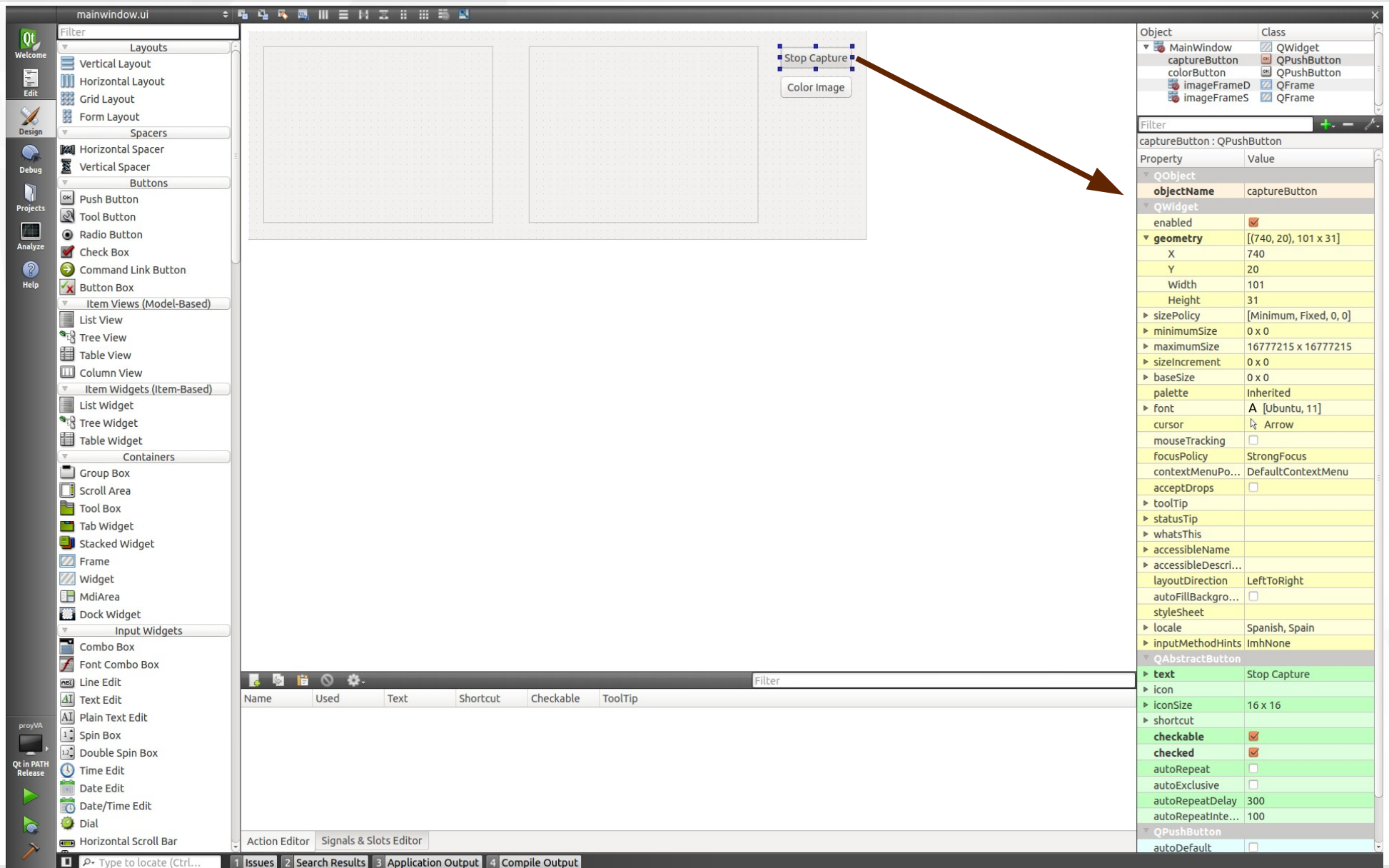
Proyecto software: estructura del proyecto

MainWindow

- ▼ Clase principal encargada de la gestión de la aplicación.
- ▼ Atributos:
 - ▼ *Ui::MainWindow *ui*: formulario de Qt. Cada elemento del formulario es un puntero al objeto de la clase que corresponda.



Proyecto software: estructura del proyecto



Proyecto software: estructura del proyecto

▼ Atributos:

- ▼ ***QTimer timer***: produce un evento de tiempo de manera periódica.
- ▼ ***ImgViewer *visorS, *visorD***: visores de la imagen original y la imagen procesada.
- ▼ ***VideoCapture *cap***: capturador de OpenCV.
- ▼ ***Mat colorImage, grayImage***: imágenes en color y en grises resultantes de la captura.
- ▼ ***Mat destColorImage, destGrayImage***: imágenes en color y en grises resultantes del procesamiento.
- ▼ ***bool winSelected***: bandera que indica si se ha seleccionado alguna ventana a través del visor de la imagen capturada.
- ▼ ***Rect imageWindow***: posición y tamaño de la ventana seleccionada.

Proyecto software: estructura del proyecto

▼ Métodos:

▼ Constructor y destructor

```
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    cap = new VideoCapture(0);
    winSelected = false;

    colorImage.create(240,320,CV_8UC3);
    grayImage.create(240,320,CV_8UC1);
    destColorImage.create(240,320,CV_8UC3);
    destGrayImage.create(240,320,CV_8UC1);

    visorS = new ImgViewer(&grayImage, ui->imageFrameS);
    visorD = new ImgViewer(&destGrayImage, ui->imageFrameD);

    connect(&timer,SIGNAL(timeout()),this,SLOT(compute()));
    connect(ui->captureButton,SIGNAL(clicked(bool)),this,SLOT(start_stop_capture(bool)));
    connect(ui->colorButton,SIGNAL(clicked(bool)),this,SLOT(change_color_gray(bool)));
    connect(visorS,SIGNAL(mouseSelection(QPointF, int, int)),this,SLOT(selectWindow(QPointF, int, int)));
    connect(visorS,SIGNAL(mouseClick()),this,SLOT(deselectWindow()));
    timer.start(30);
}
```

Proyecto software: estructura del proyecto

▼ Métodos:

▼ Constructor y destructor

```
MainWindow::~MainWindow()
{
    delete ui;
    delete cap;
    delete visorS;
    delete visorD;
    colorImage.release();
    grayImage.release();
    destColorImage.release();
    destGrayImage.release();
}
```

Proyecto software: estructura del proyecto

▼ Métodos:

▼ Slots

```
void MainWindow::compute()
{
    //Captura de imagen
    if(ui->captureButton->isChecked() && cap->isOpened())
    {
        *cap >> colorImage;
        cv::resize(colorImage, colorImage, Size(320,240));
        cvtColor(colorImage, grayImage, CV_BGR2GRAY);
        cvtColor(colorImage, colorImage, CV_BGR2RGB);
    }

    //En este punto se debe incluir el código asociado con el procesamiento de cada captura

    //Actualización de los visores
    if(winSelected)
        visorS->drawSquare(QRect(imageWindow.x, imageWindow.y, imageWindow.width, imageWindow.height), Qt::green );

    visorS->update();
    visorD->update();
}
```


Proyecto software: estructura del proyecto

▼ Métodos:

▼ Slots

```
void MainWindow::start_stop_capture(bool start)
{
    if(start)
        ui->captureButton->setText("Stop capture");
    else
        ui->captureButton->setText("Start capture");
}

void MainWindow::change_color_gray(bool color)
{
    if(color)
    {
        ui->colorButton->setText("Gray image");
        visorS->setImage(&colorImage);
        visorD->setImage(&destColorImage);
    }
    else
    {
        ui->colorButton->setText("Color image");
        visorS->setImage(&grayImage);
        visorD->setImage(&destGrayImage);
    }
}
```

Proyecto software: estructura del proyecto

▼ Métodos:

▼ Slots

```
void MainWindow::selectWindow(QPointF p, int w, int h)
{
    QPointF pEnd;
    if(w>0 && h>0)
    {
        imageWindow.x = p.x()-w/2;
        if(imageWindow.x<0)
            imageWindow.x = 0;
        imageWindow.y = p.y()-h/2;
        if(imageWindow.y<0)
            imageWindow.y = 0;
        pEnd.setX(p.x()+w/2);
        if(pEnd.x()>=320)
            pEnd.setX(319);
        pEnd.setY(p.y()+h/2);
        if(pEnd.y()>=240)
            pEnd.setY(239);
        imageWindow.width = pEnd.x()-imageWindow.x+1;
        imageWindow.height = pEnd.y()-imageWindow.y+1;

        winSelected = true;
    }
}

void MainWindow::deselectWindow()
{
    winSelected = false;
}
```