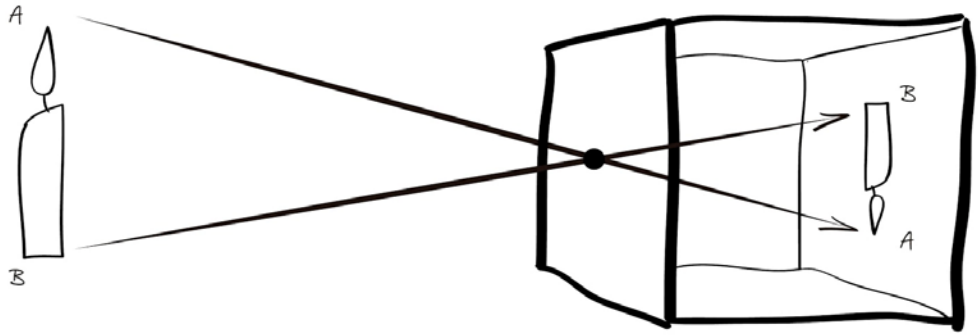
Universidad
Rey Juan Carlos
Escuela Técnica Superior
Ingeniería de Telecomunicación



Visión Artificial

2. Formación de la imagen

JOSÉ MIGUEL GUERRERO HERNÁNDEZ

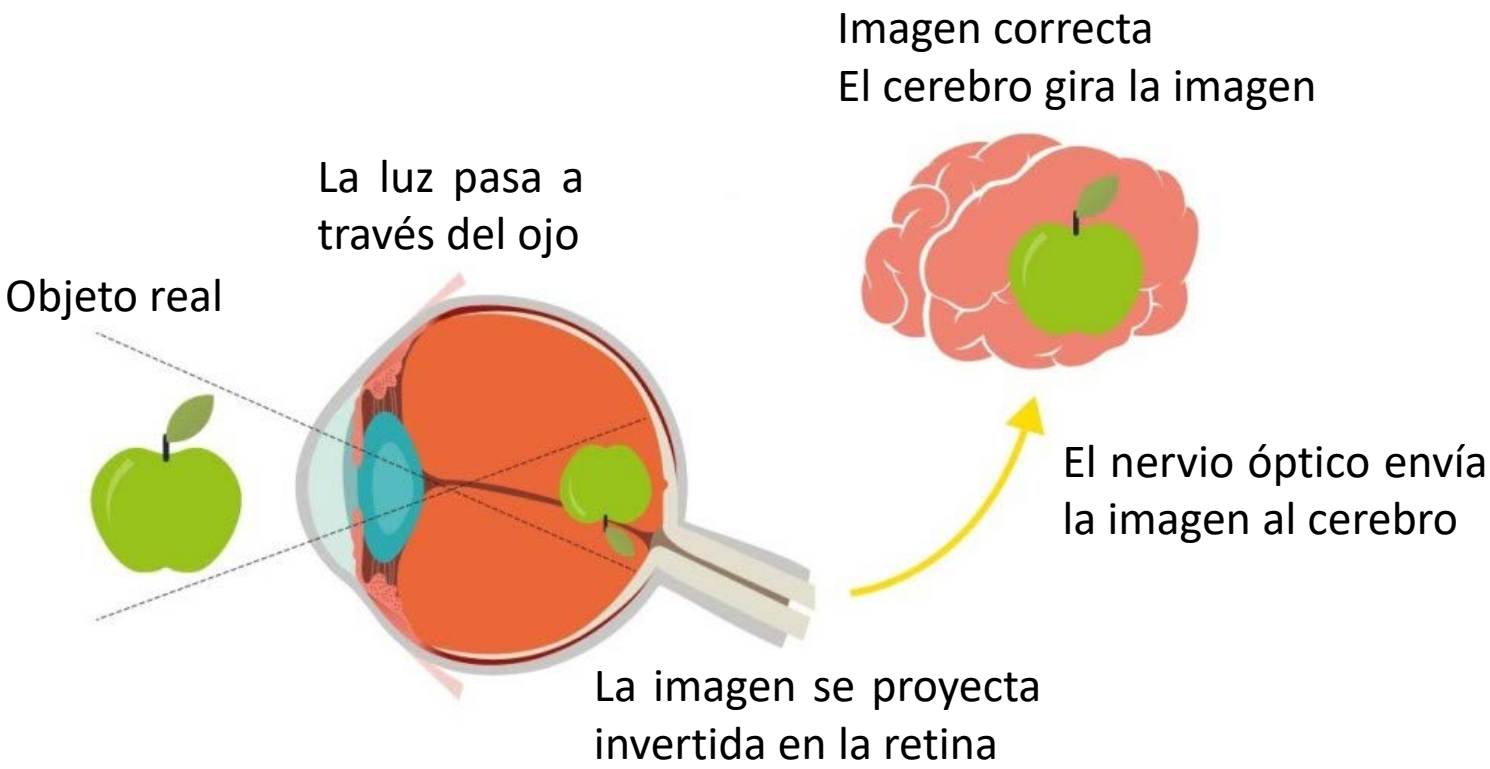
EMAIL: JOSEMIGUEL.GUERRERO@URJC.ES

Índice de contenidos

Índice de contenidos

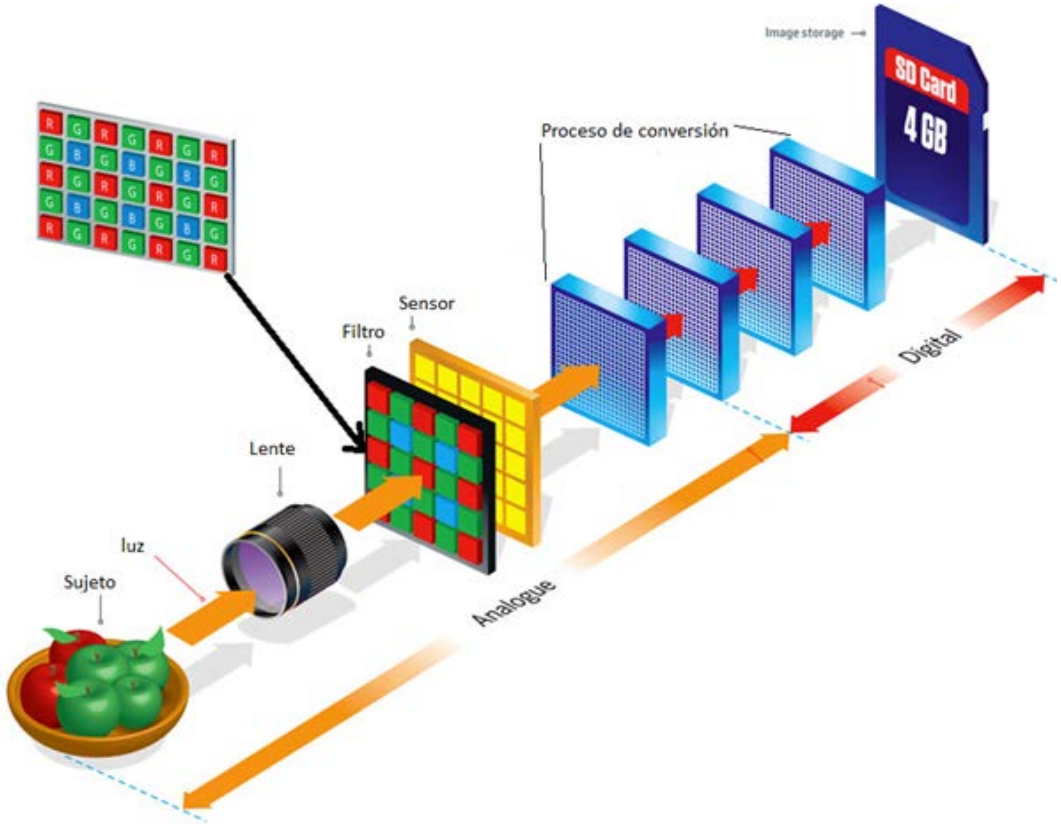
1. Ojo humano vs cámara

- Formación de la imagen a través del ojo humano:



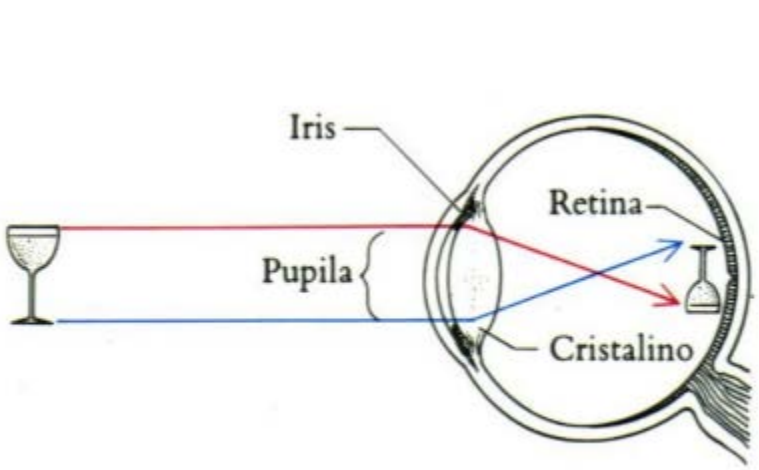
1. Ojo humano vs cámara

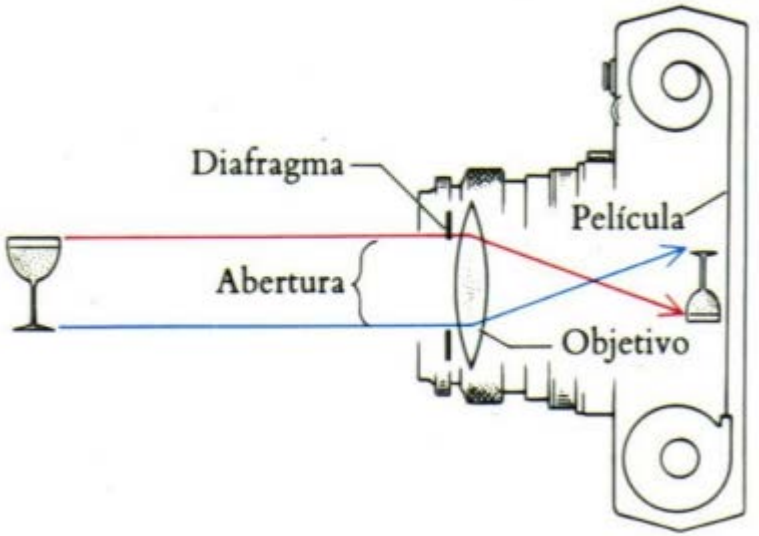
- Formación de la imagen a través de una cámara:



1. Ojo humano vs cámara

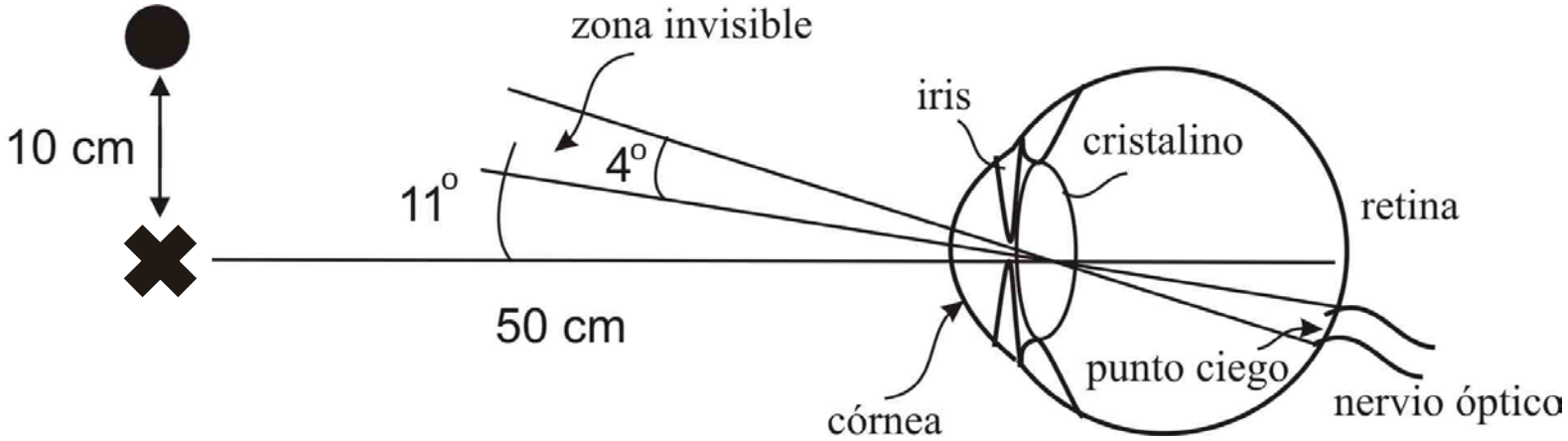
- Similitudes:





1. Ojo humano vs cámara

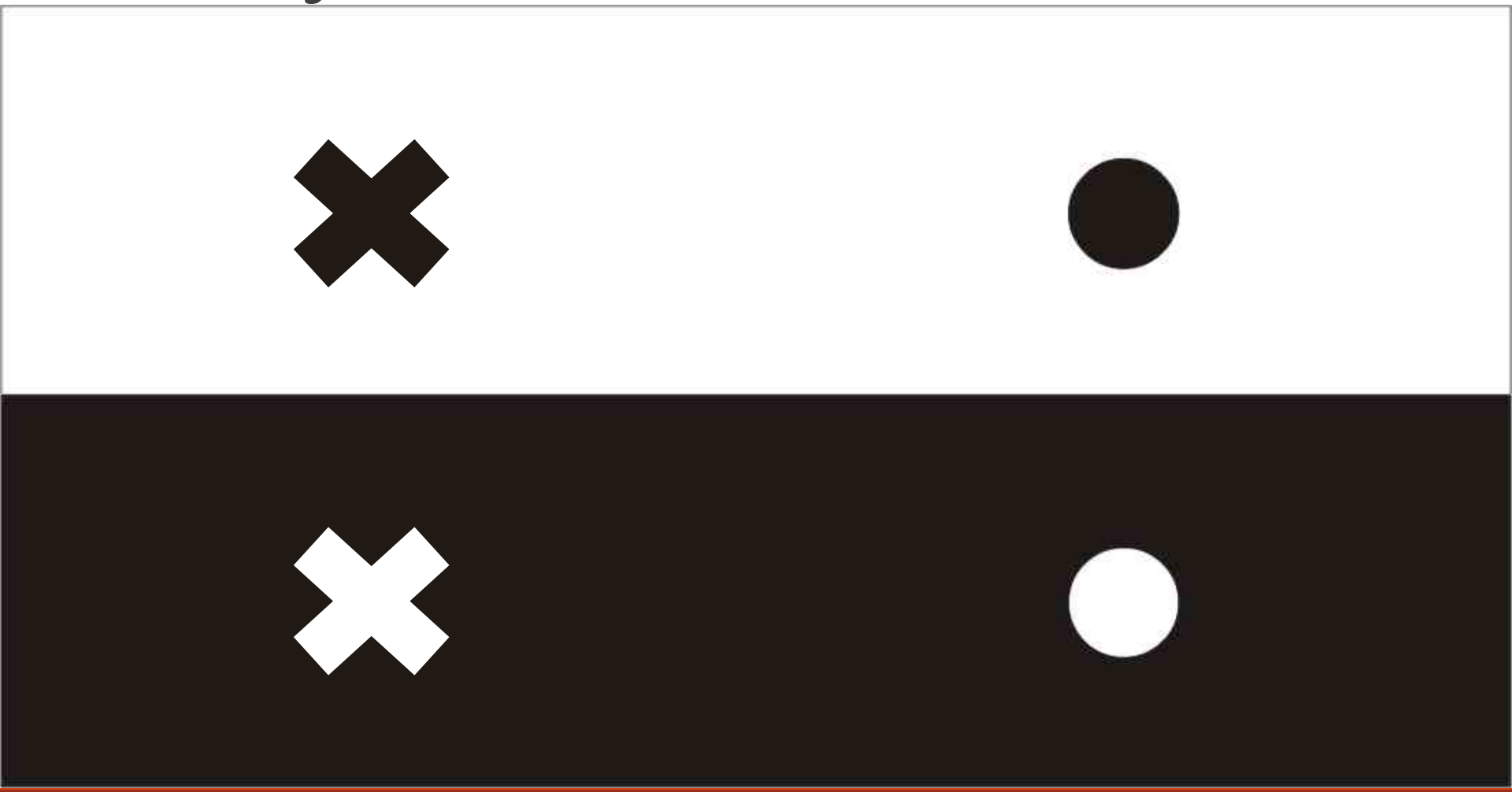
- Nuestro ojo tiene un punto ciego, que la cámara no, la inserción del nervio óptico
- Nuestro cerebro reemplaza ese punto ciego con un relleno lo más parecido al área que rodea ese punto



1. Ojo humano vs cámara

- ¿No te lo crees? Prueba
- El test de la siguiente página está diseñado para revelar ese punto ciego:
  - Cierra ojo izquierdo, coloca el ojo derecho a unos 50 centímetros de la cruz negra y mira la cruz con atención
  - Varía ligeramente la distancia a la pantalla, y cuando te encuentres a cierta distancia de la pantalla el círculo negro desaparece de tu campo visual y en su lugar ves blanco
  - Ahora, quédate a esa distancia y fíjate en la cruz blanca, en este caso el círculo blanco desaparece y se rellena con el color negro del fondo

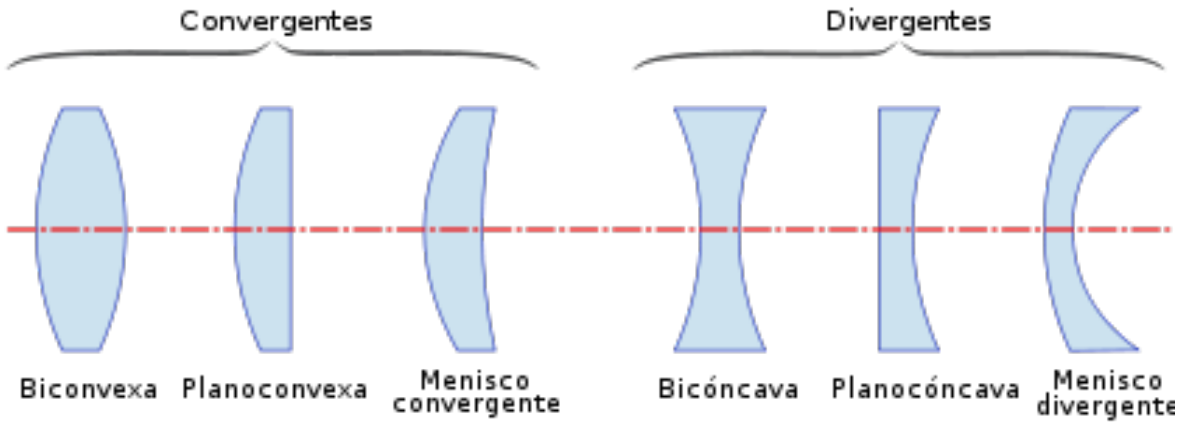
1. Ojo humano vs cámara



Índice de contenidos

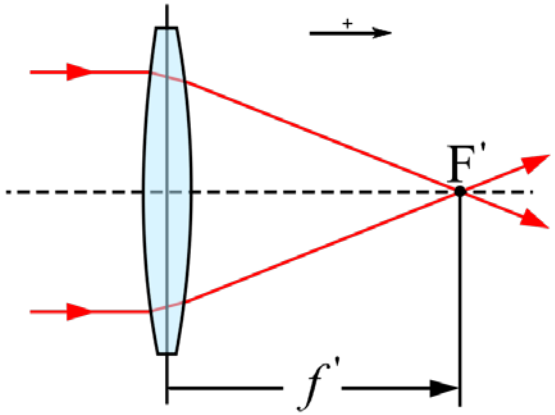
2. Lentes

- Para hablar de la formación de la imagen es necesario primero, hacer una introducción sobre la óptica y las lentes
- La óptica es la rama de la física que estudia la luz, su propagación y su interacción con la materia. Así, incluye la reflexión, la refracción y la absorción en instrumentos ópticos como lentes, espejos, prismas
- También abarca los aspectos fisiológicos de la visión



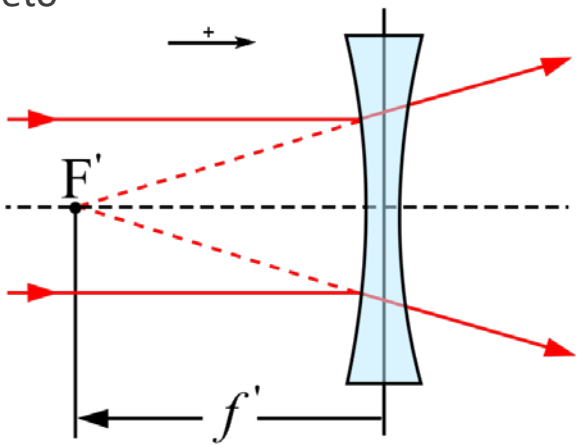
2. Lentes

- Convergentes o convexas:
  - Se las llama también lentes positivas
  - Los rayos de luz, al atravesar una de estas lentes van a converger en un mismo punto, que es el foco
  - Son más gruesas en el centro que en los extremos
  - Las imágenes de las lentes convergentes son reales e invertidas
  - Cuanto más cerca está el objeto más grande será la imagen del mismo
  - El foco está siempre detrás de la lente



2. Lentes

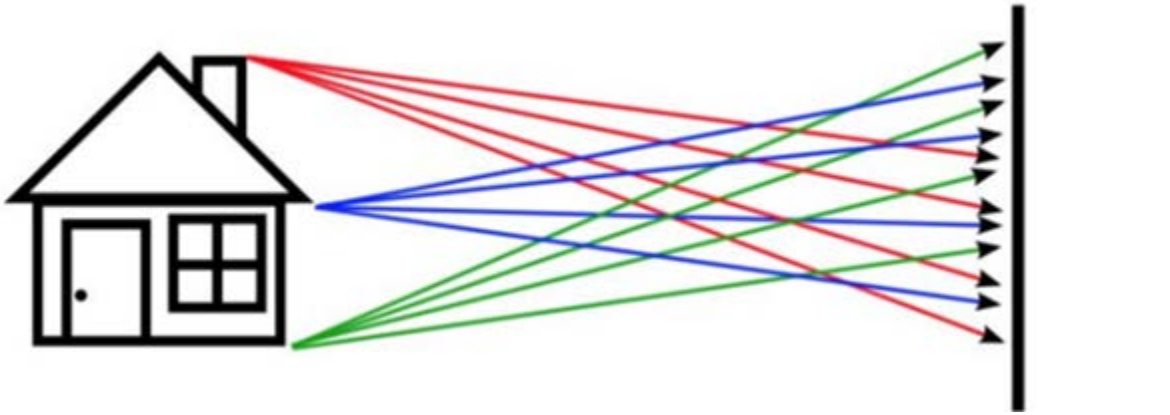
- Divergentes o convexas:
  - Se les llama también lentes negativas
  - Por la forma de la lente los rayos no convergen en un punto, sino que al atravesar la lente divergen en todas las direcciones
  - Son más delgadas en la parte central que en los extremos
  - Las imágenes de este tipo de lentes son virtuales, es decir, no se pueden recoger sobre ninguna pantalla
  - Las imágenes son siempre más reducidas que el objeto
  - El foco está delante de la lente
  - La imagen se encuentra entre el foco y la lente



Índice de contenidos

3. Modelos

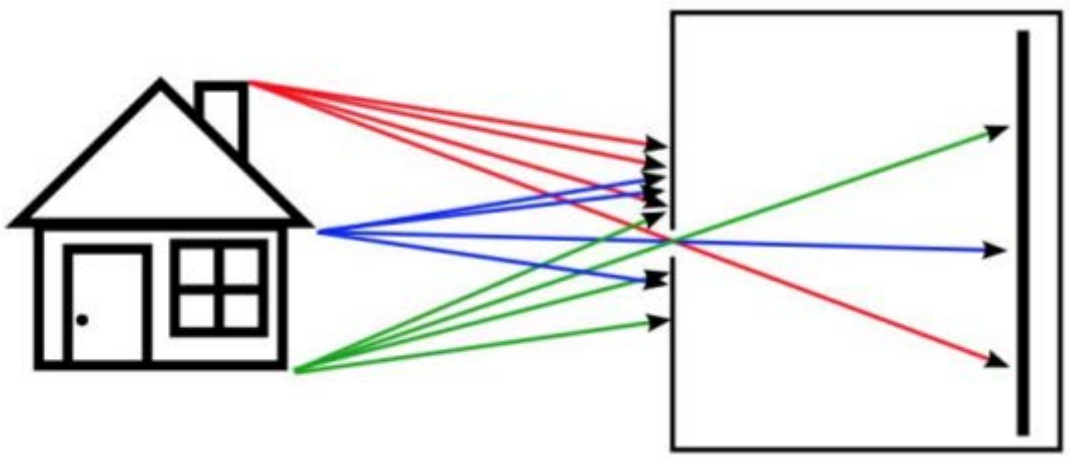
- Intentemos construir un dispositivo simple



- No podemos obtener una imagen razonable

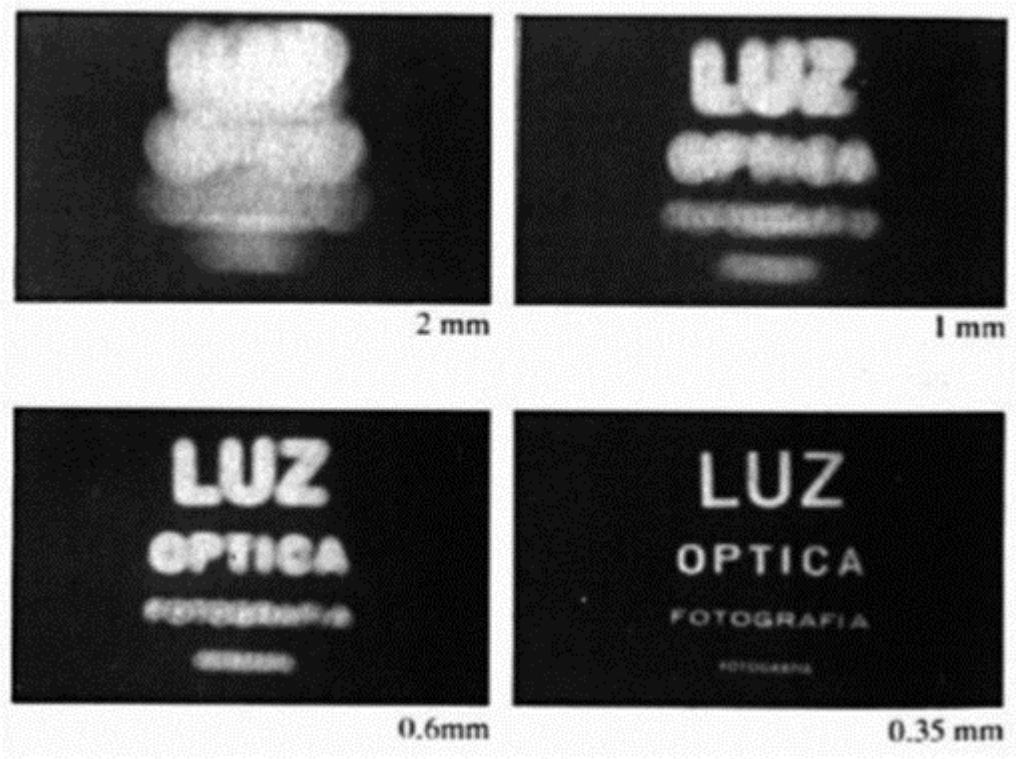
3. Modelos: pinhole

- Poner una barrera con un pequeño orificio (apertura) entre el objeto y el sensor
  - Se reduce el desenfoque pero para ello tenemos que hacer que la apertura sea lo más pequeña posible
  - Esto es lo que se conoce como cámara estenopeica (pinhole)



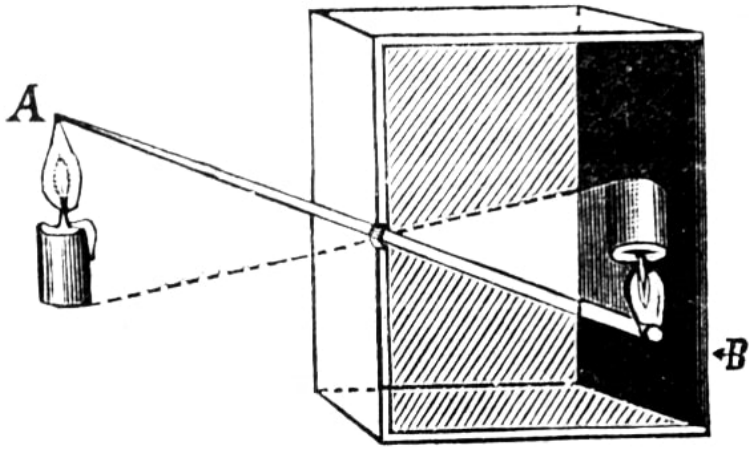
3. Modelos: pinhole

- ¿Cuánto de pequeño hay que hacer ese agujero?



3. Modelos: pinhole

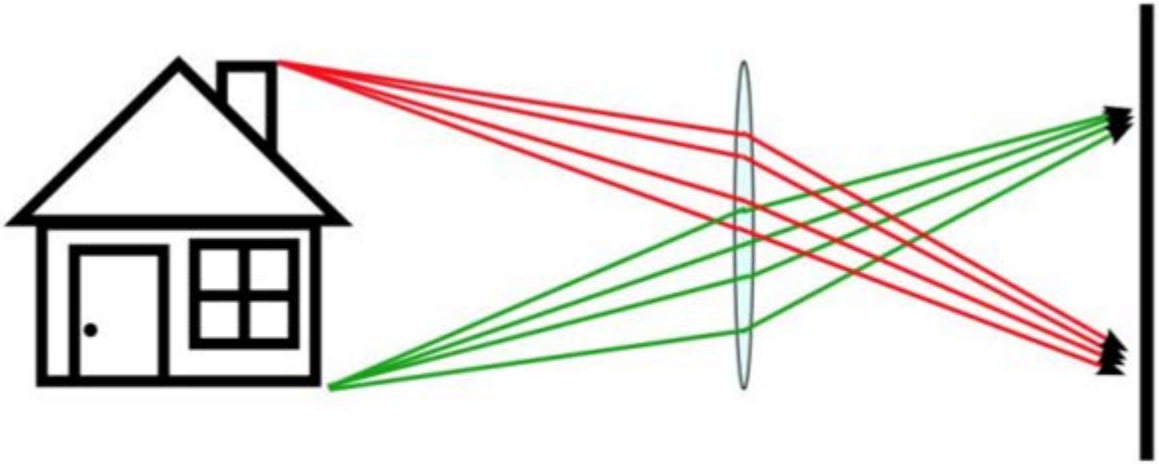
- Cuando la luz de una imagen pasa a través de este agujero entonces se forma una imagen invertida en el lado opuesto



- Problemas del modelo pinhole:
  - Si dejamos un agujero grande, la imagen se verá borrosa
  - Cuanto más pequeño es el agujero por el que atraviesa la luz, más nítida y definida es la imagen, pero se hace más oscura

3. Modelos: lente delgada

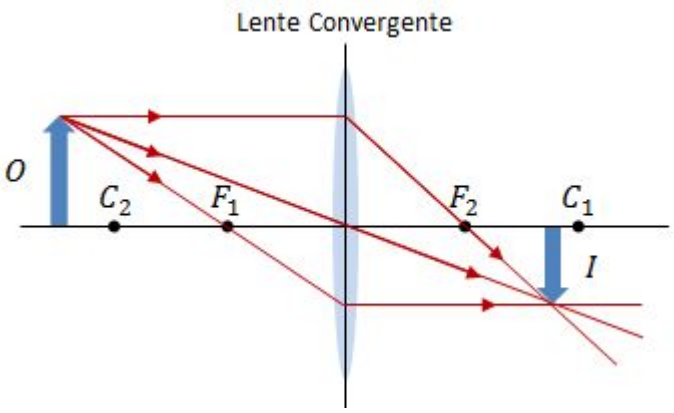
- ¿Solución? Utilizar lentes

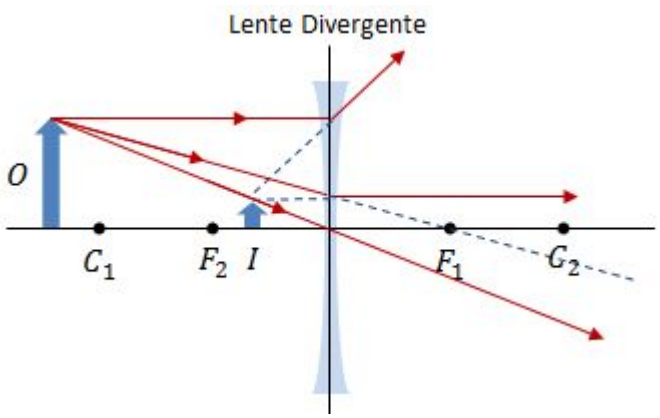


- Esta lente enfoca la luz sobre la película/sensor

3. Modelos: lente delgada

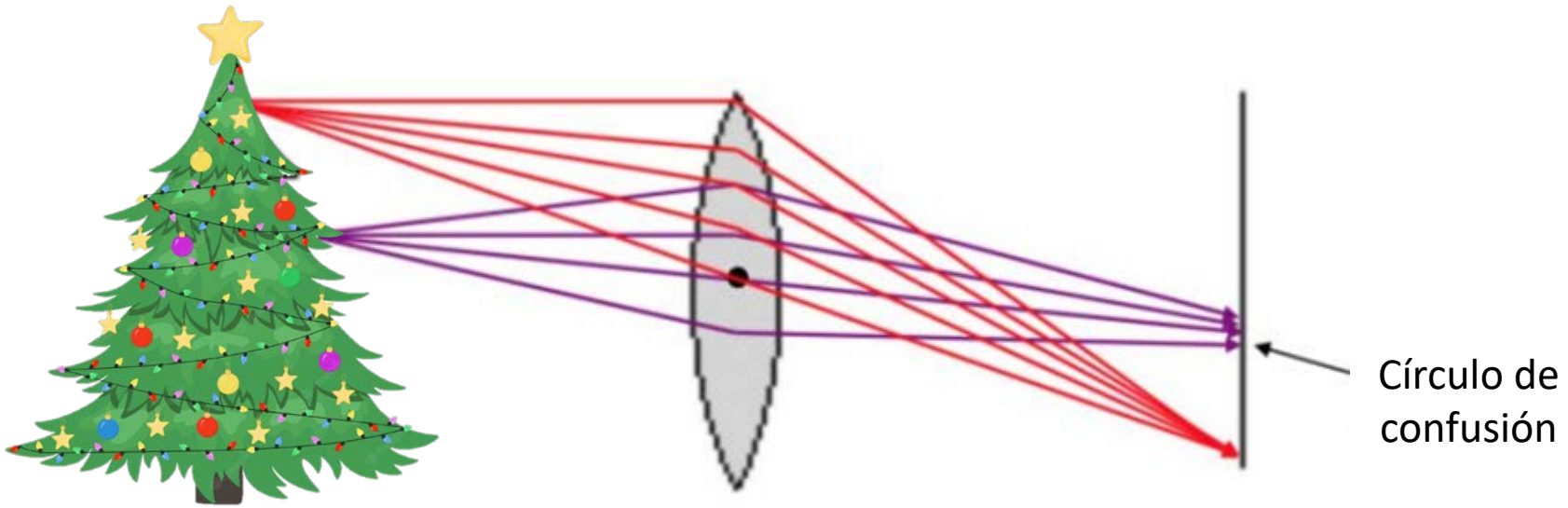
- Este modelo se conoce como modelo de lente delgada:
  - Los rayos que atraviesan el centro óptico no se desvían
  - Concentra en un punto los infinitos rayos luminosos procedentes de un punto del espacio
  - Todos los rayos paralelos al eje óptico convergen en un punto que es el foco y depende de la lente





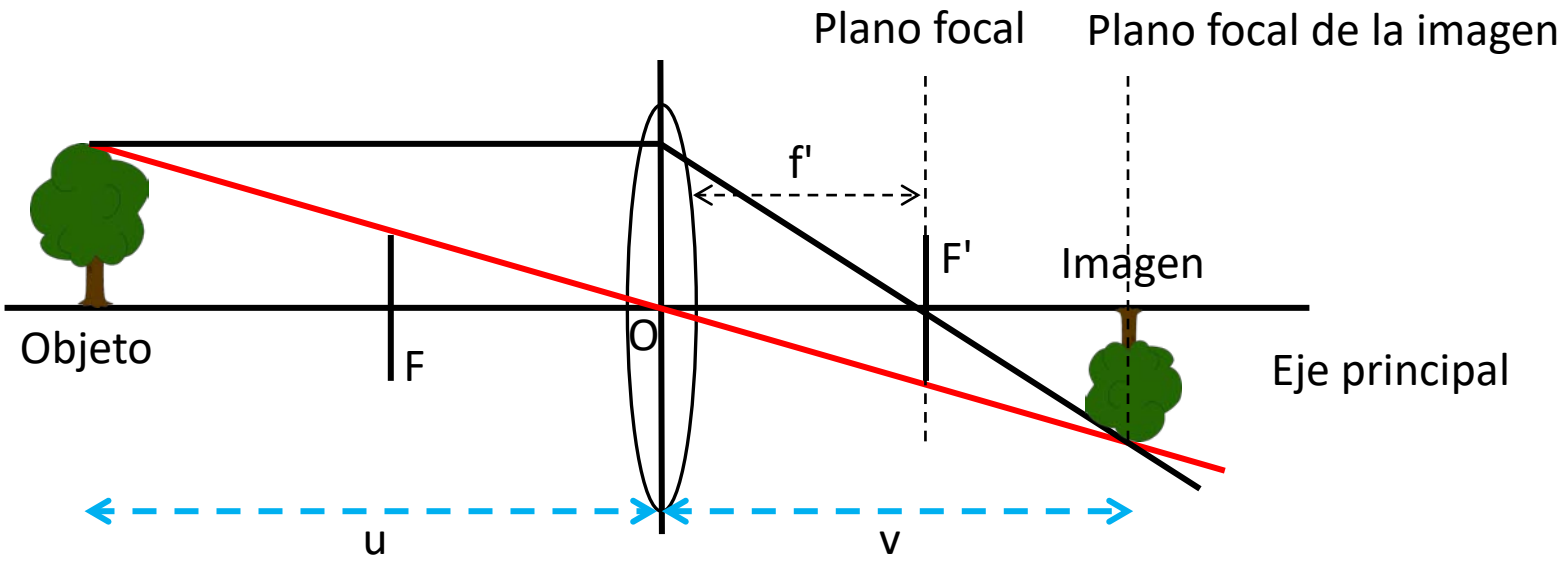
3. Modelos: lente delgada

- A diferencia de la cámara estenopeica ideal, hay una distancia específica a la que los objetos están enfocados



3. Modelos: lente delgada

- La fórmula de las lentes delgadas permite relacionar la posición del objeto y de la imagen con la distancia focal



3. Modelos: lente delgada

- La fórmula de las lentes delgadas permite relacionar la posición del objeto y de la imagen con la distancia focal

$$\frac{1}{F} = \frac{1}{v} - \frac{1}{u}$$

  - F = distancia focal
  - u = distancia del objeto, distancia conjugada objeto
  - v = distancia de la imagen, distancia conjugada imagen

- Pero este modelo no es perfecto:
  - Las lentes imperfectas pueden causar distorsión radial (las desviaciones son más notorias para los rayos que pasan por el borde de la lente)
  - La solución: calibrar (lo veremos más adelante)

Índice de contenidos

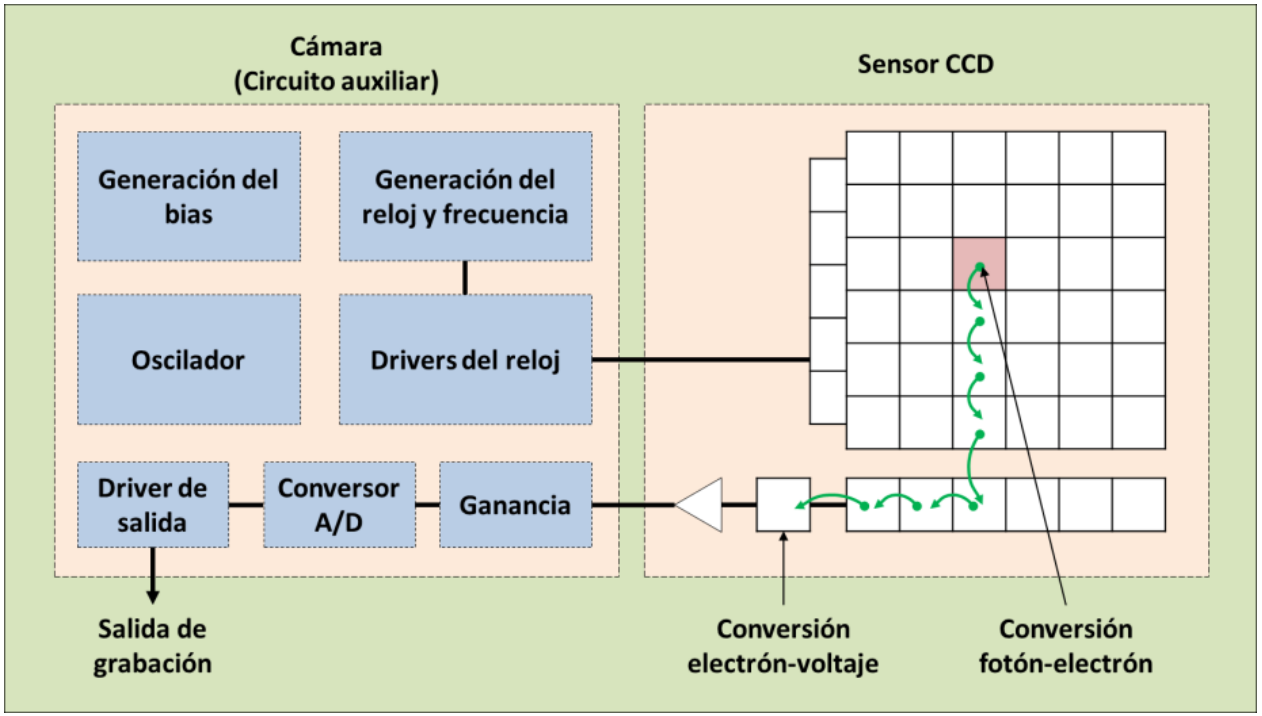
4. Formación de la imagen

- **Parámetros intrínsecos**, son valores nominales proporcionados por los respectivos fabricantes:
  - Longitud focal
  - Centro del plano de la imagen
  - Tamaño del píxel, normalmente en micras
  - Resolución horizontal-vertical del CCD, normalmente en micras

- **Parámetros extrínsecos**, aquellos que dependen de la posición del sistema de visión y que depende de cada escenario:
  - Altura
  - Inclinación (ángulos de cabeceo - pitch, alabeo - roll, y guiñada - yaw)

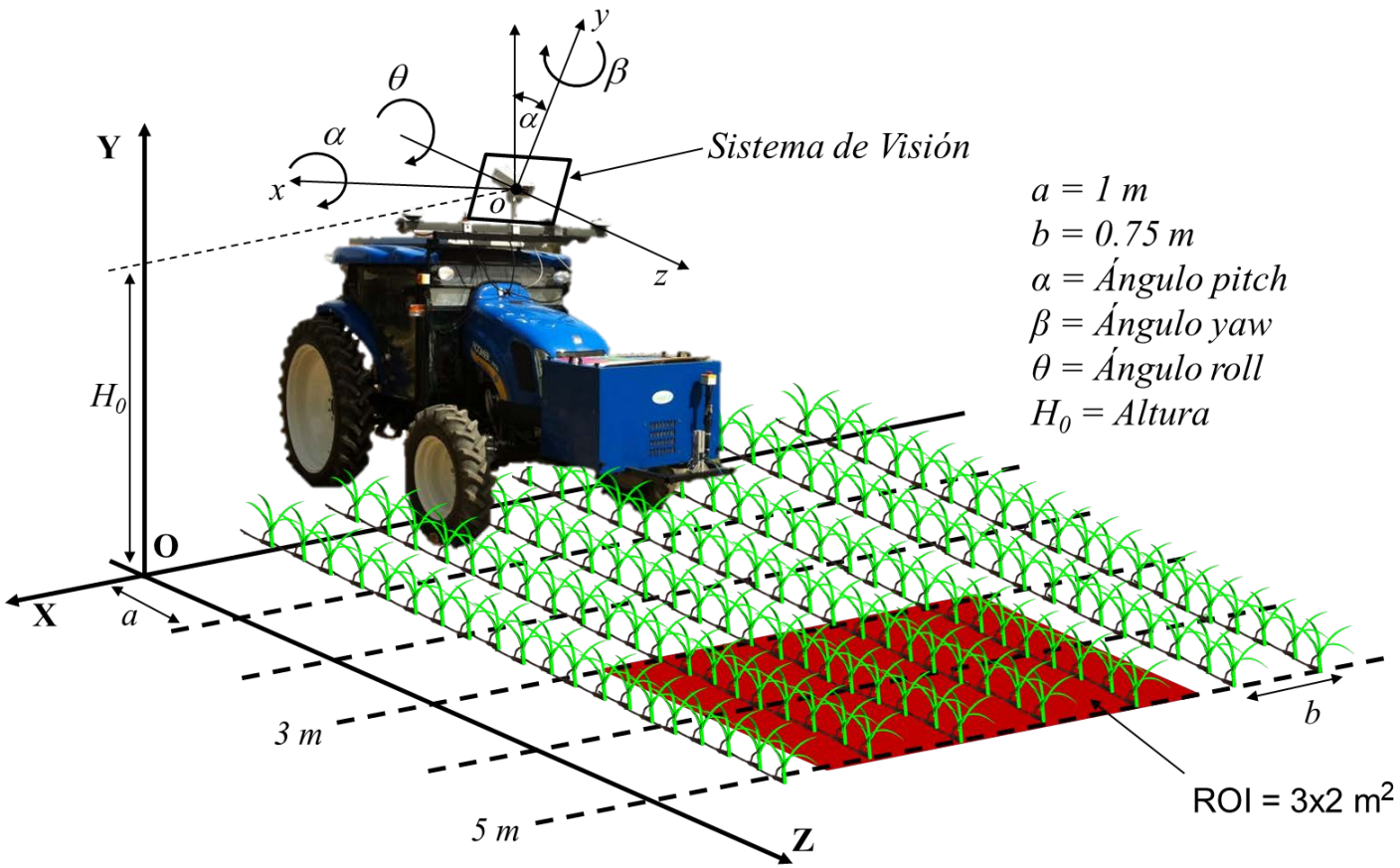
4. Formación de la imagen

- Parámetros intrínsecos



4. Formación de la imagen

- Parámetros extrínsecos



Índice de contenidos

5. Imagen digital

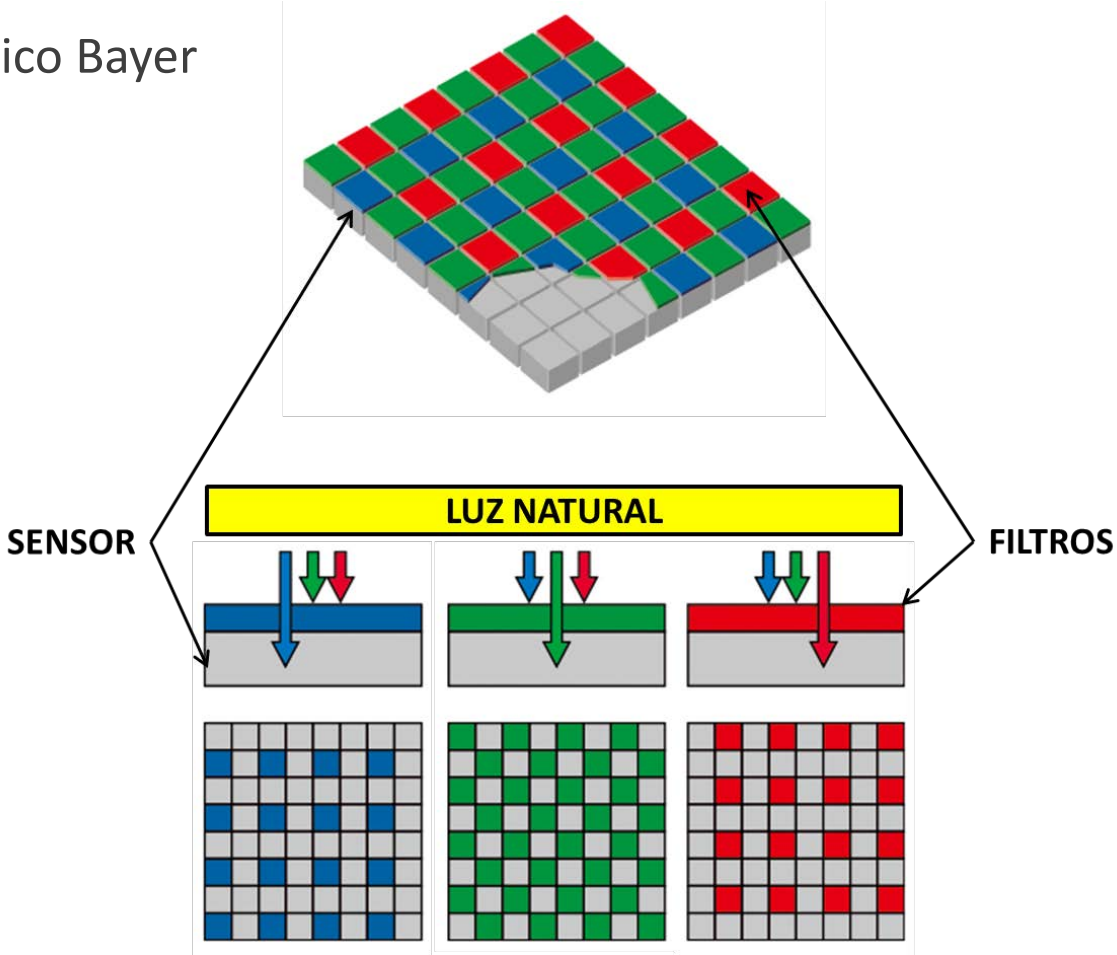
- El sensor, y por tanto el plano, se suele dividir en partes iguales (píxeles) típicamente de forma rectangular
- Cada píxel es capaz de captar la luz que le incide
- El valor de cada píxel es proporcional a la cantidad de luz reflejada por la parte de la superficie del objeto que se proyecta sobre ese píxel, por lo que depende de:
  - Material del objeto
  - Posición de las luces en la escena
  - Reflejo de otros objetos en la escena
- Todo esto hace que la imagen real obtenida únicamente por el sensor sea monocromo, no tiene color

5. Imagen digital

- Para captar la imagen a color es necesario disponer de un sistema de filtros que cubran dicho sensor
- Cada píxel, en el sensor, va a ser más o menos sensible a la radiación recibida en función de las longitudes de onda en el espectro visible
- Los filtros tendrán las longitudes de onda correspondientes al rojo, verde y azul
- Uno de los filtros más utilizados es el conocido como mosaico Bayer:
  - Para cada píxel existe un filtro que limita la incidencia de la radiación a una única longitud de onda (roja, verde o azul)
  - Para formar la imagen se recorre todo el mosaico en grupos de cuatro píxeles, que aportan los datos de color a cada porción de imagen, y cuya transformación da lugar a una imagen en el espacio de color RGB

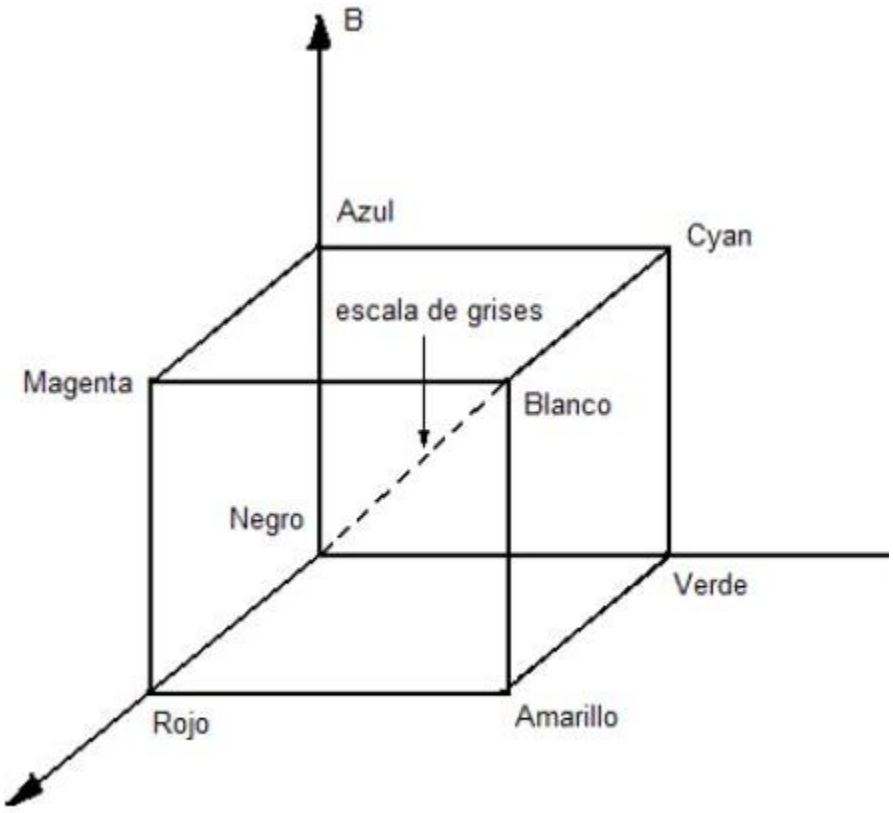
5. Imagen digital

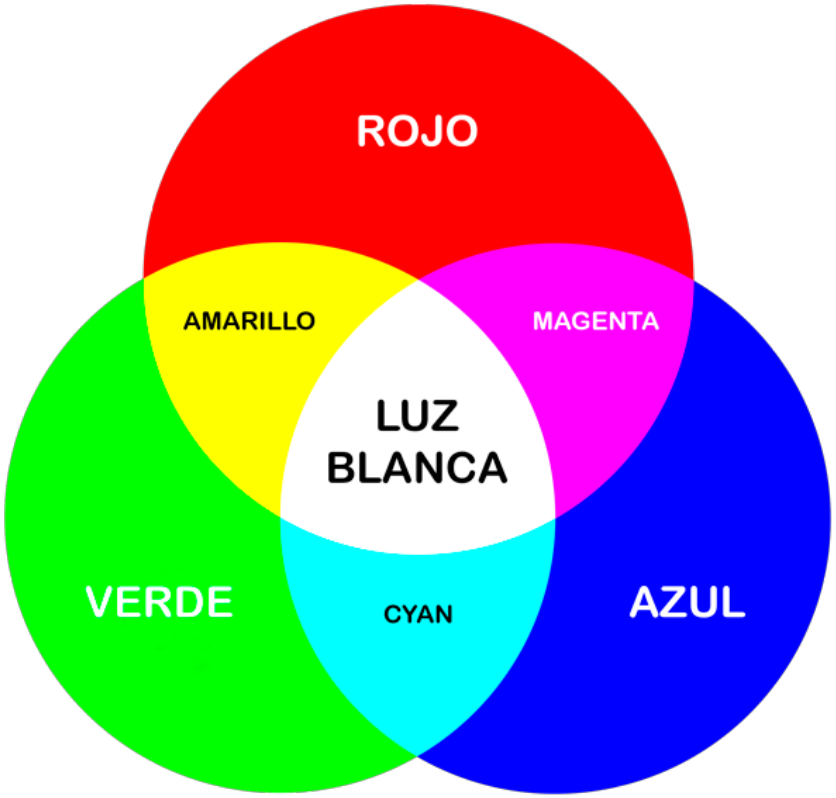
- Mosaico Bayer



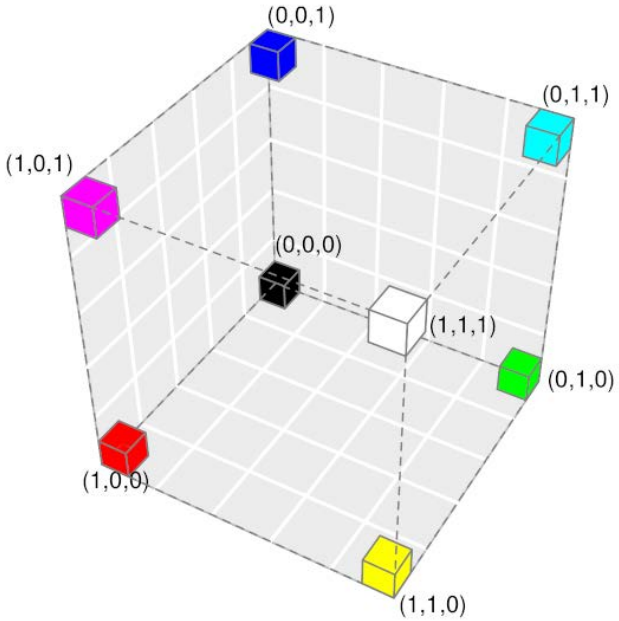
Índice de contenidos

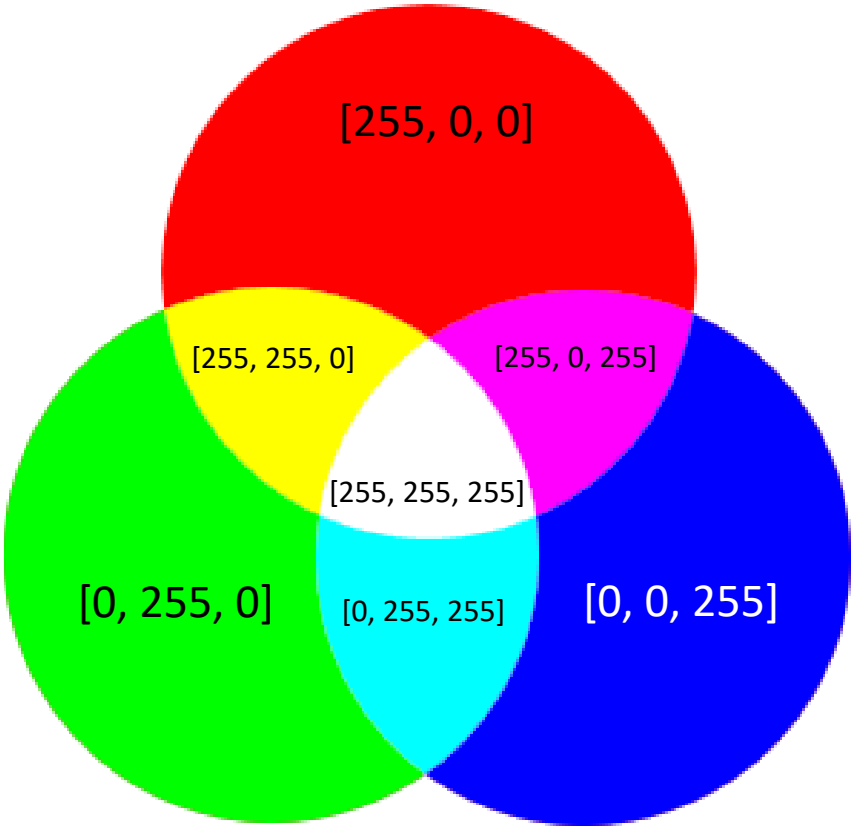
6. Espacios de color: RGB



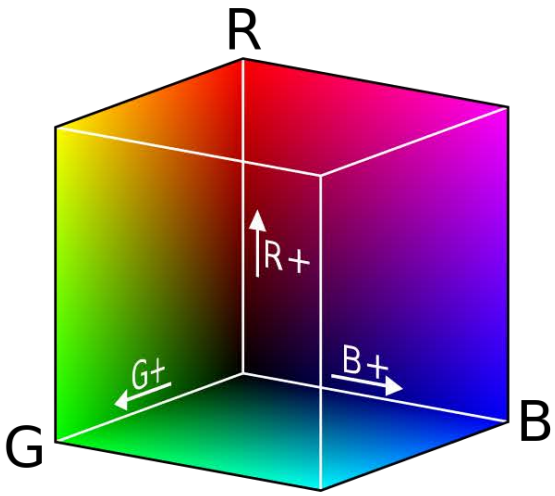


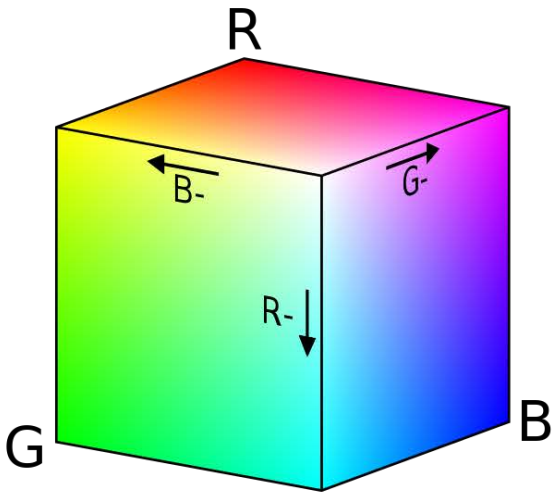
6. Espacios de color: RGB

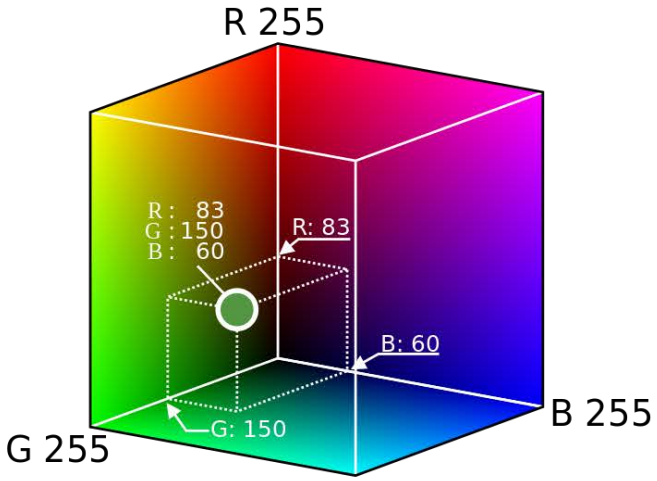




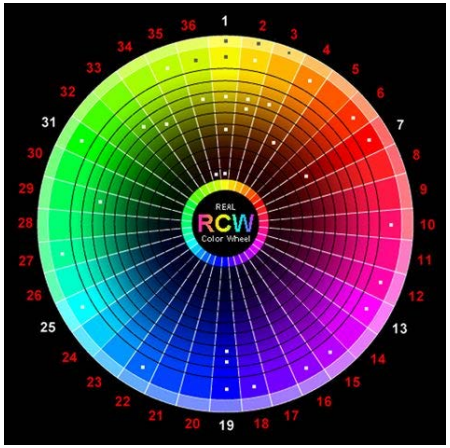
6. Espacios de color: RGB



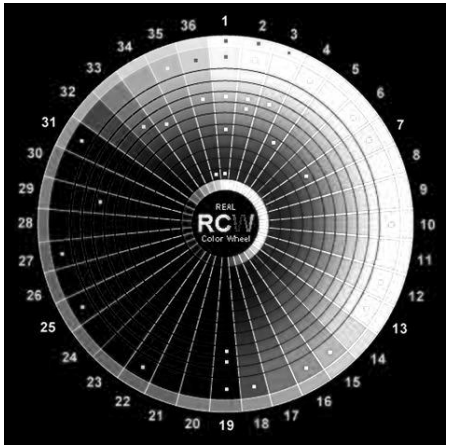




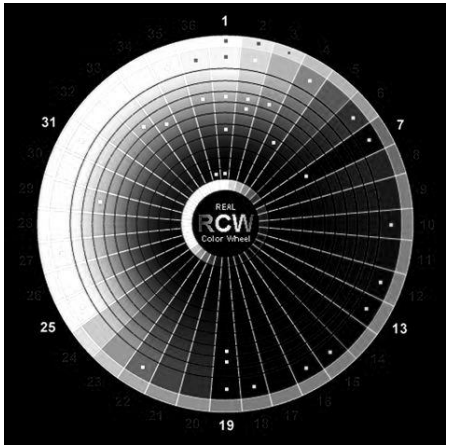
6. Espacios de color: RGB



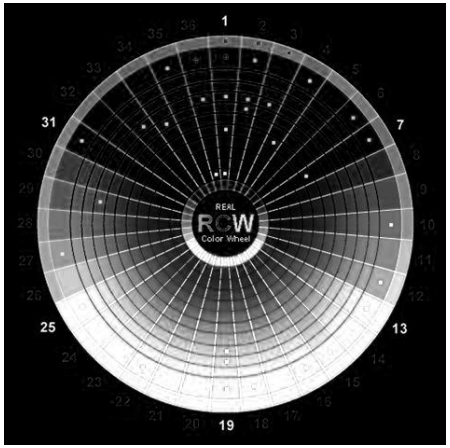
RGB



R

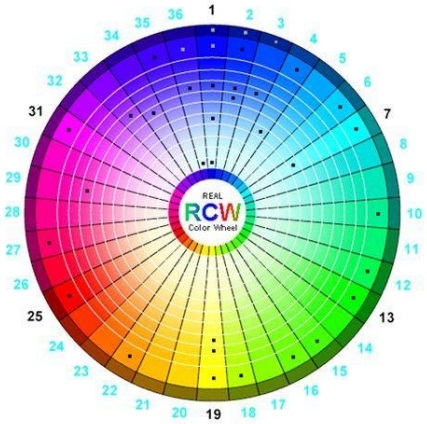


G



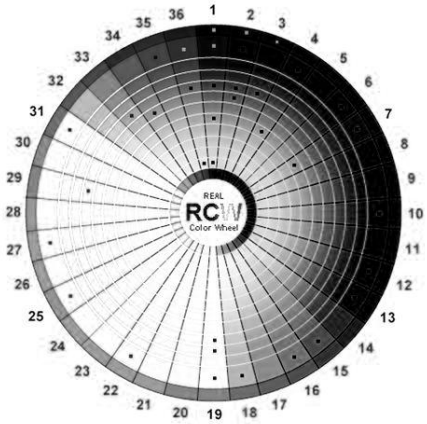
B

6. Espacios de color: CMY

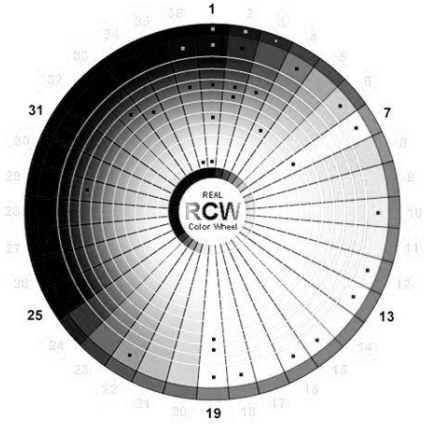


CMY

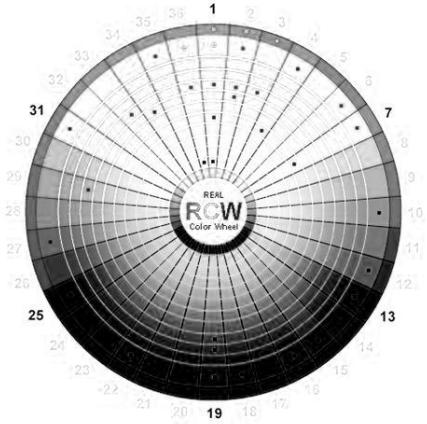
$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 255 \\ 255 \\ 255 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



C

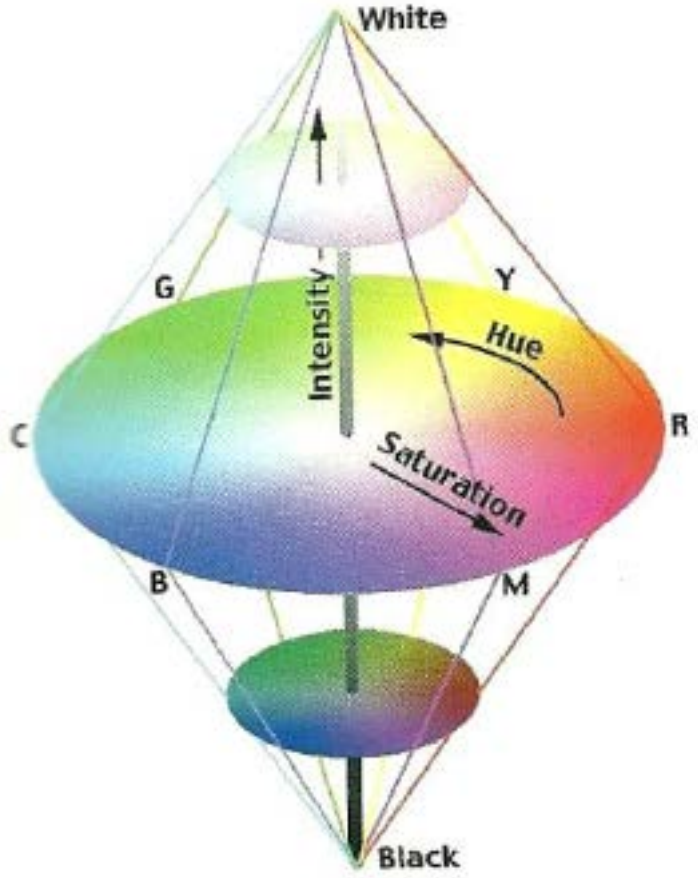


M



Y

6. Espacios de color: HSI



Matiz:

$$H = \cos^{-1}\left[\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-B)^2+(R-B)(G-B)]^{1/2}}\right]$$

Saturación:

$$S = 1 - \frac{3}{R+G+B}[min(R,G,B)]$$

Intensidad:

$$I = \frac{1}{3}(R+G+B)$$

6. Espacios de color: HSI

- Los valores HSI parten de estar normalizados entre 0 y 1:
  - Para ello, cada valor R, G, B tiene que normalizarse dividiendo entre 255

$$R = \frac{R}{255}; G = \frac{G}{255}; B = \frac{B}{255}$$

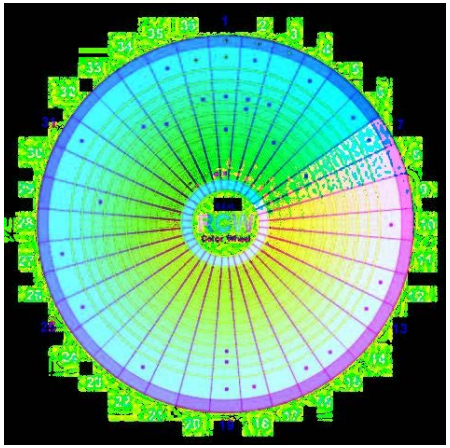
- Hay que tener en cuenta que cuando el color está en la mitad superior del triángulo de cromaticidad, debemos restar el ángulo H a 360 para obtener el tono, es decir:

$$Si\ (B > G) \rightarrow H = 360 - H$$

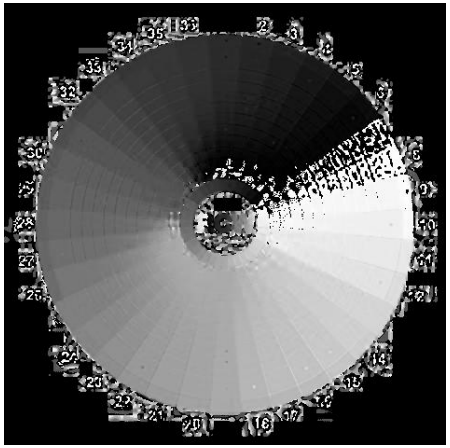
- Una vez realizados los cálculos, conviene pasar del intervalo [0,1] al [0,255]

$$H = H/360 * 255; S = S * 255; I = I * 255$$

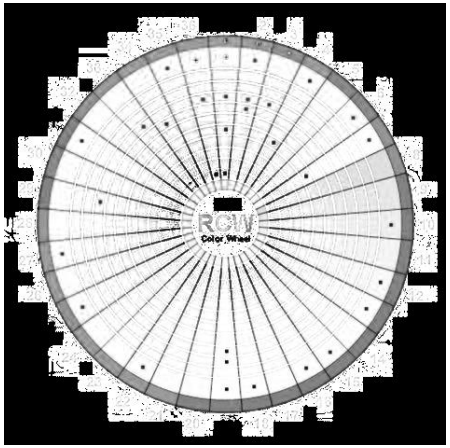
6. Espacios de color: HSI



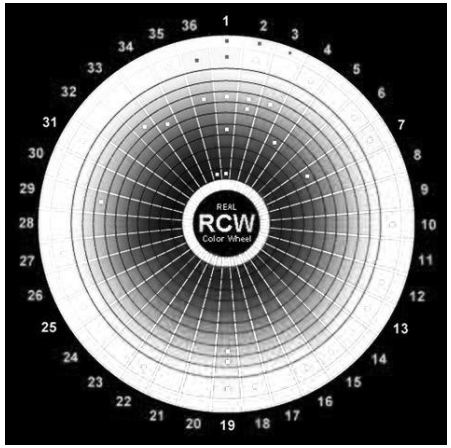
HSI



H



S



I

Índice de contenidos

7. Ejemplo: leer imagen

```cpp
#include <opencv2/highgui.hpp>

int main () {
    // Create image variable
    cv::Mat image;

    // Read image
    image = cv::imread("../../images/lenna.jpg", cv::IMREAD_COLOR);

    // Show image
    cv::imshow("TEST IMAGE", image);

    // Wait to press a key
    cv::waitKey(0);

    return 0;
}
```

Las imágenes en OpenCV son tipo cv::Mat

Mostrar la imagen

Lectura de la imagen:
Mat cv::imread (const String &filename, int flags=IMREAD_COLOR)

7. Ejemplo: leer imagen

Usando namespace

```
#include <opencv2/highgui.hpp>

using namespace cv;

int main () {
    // Create image variable
    Mat image;

    // Read image
    image = imread("../../images/lenna.jpg", IMREAD_COLOR);

    // Show image
    imshow("TEST IMAGE", image);

    // Wait to press a key
    waitKey(0);

    return 0;
}
```

Igual que el ejemplo anterior, pero usando el namespace evitamos tener que utilizar elementos de OpenCV con cv::

7. Ejemplo: leer píxeles (Vec3b)

```
int main( int argc, char** argv ) {
  // Load an image
  Mat src = imread( "../../images/lenna.jpg", IMREAD_COLOR );
  if ( src.empty() ) {
    cout << "Could not open or find the image!\n" << endl;
    cout << "Usage: " << argv[0] << " <Input image>" << endl;
    return -1;
  }

  // Show image
  namedWindow( "Pixel Demo", WINDOW_AUTOSIZE );
  imshow("Pixel Demo", src);


  // Method 1:
  // Read pixel values using Vec3b: vector of 3 values
  for ( int i=0; i<src.rows; i++ )
        for ( int j=0; j<src.cols; j++ )
            // You can now access the pixel value with cv::Vec3b
            cout << (uint)src.at<Vec3b>(i,j)[0] << " " << (uint)src.at<Vec3b>(i,j)[1]
                        << " " << (uint)src.at<Vec3b>(i,j)[2] << endl;


  waitKey(0);
  return 0;
}
```

Lectura de la imagen

Mostrar la imagen

Acceso a los elementos utilizando Vec3b. Los elementos i y j son las coordenadas de cada píxel. Los accesos a cada array [] son el canal en el cual se trabaja, 0 – Blue, 1 – Green, 2 – Red.

7. Ejemplo: leer píxeles (split)

```
int main( int argc, char** argv ) {
  // Leer y mostrar imagen
  // ...

  // ------------
  // Method 2:
  // Read pixel values using split channels
  vector<Mat> three_channels;
  split( src, three_channels );

  // Now I can access each channel separately
  for( int i=0; i<src.rows; i++ )
        for( int j=0; j<src.cols; j++ )
              cout << (uint)three_channels[0].at<uchar>(i,j) << " "
                   << (uint)three_channels[1].at<uchar>(i,j) << " "
                   << (uint)three_channels[2].at<uchar>(i,j) << endl;

  imshow( "Blue channel", three_channels[0] );
  imshow( "Green channel", three_channels[1] );
  imshow( "Red channel", three_channels[2] );

  // ...

}
```

En este caso se crea un vector de matrices vector<Mat> y se utiliza la función split para separar los canales

Como el caso anterior, los elementos i y j son las coordenadas de cada píxel. Los accesos a cada canal se realiza a través de [] : , 0 – Blue, 1 – Green, 2 – Red.

7. Ejemplo: leer píxeles (split)

```
int main( int argc, char** argv ) {
  // Leer y mostrar imagen
  // ...

  // ------------
  // Method 2:
  // Read pixel values using split channels
  vector<Mat> three_channels;
  split( src, three_channels );
  // Now I can access each channel separately
  // ...
  // Create new image combining channels
  vector<Mat> channels;
  channels.push_back(three_channels[0]);
  channels.push_back(three_channels[1]);
  channels.push_back(three_channels[2]);

  Mat new_image;
  merge(channels, new_image);
  imshow("New image", new_image);


  waitKey(0);
  return 0;
}
```

Para unir se crea un vector de matrices vector<Mat>. Se agregan las matrices a combinar

Para obtener la imagen final, se combinan las tres matrices en una utilizando la función merge

7. Ejemplo: cambio de espacio

```
int main( int argc, char** argv ) {
  // Load an image
  Mat src = imread( "../../images/RGB.jpg", IMREAD_COLOR );
  if ( src.empty() ) {
    cout << "Could not open or find the image!\n" << endl;
    cout << "Usage: " << argv[0] << " <Input image>" << endl;
    return -1;
  }

  // ...

  // Changing original image to HSV using cvtColor
  Mat HSV_opencv;
  cvtColor(src, HSV_opencv, COLOR_RGB2HSV);
  imshow("HSV OpenCV", HSV_opencv);


  waitKey(0);
  return 0;
}
```

Para realizar un cambio de espacio de color, se utiliza la función cvtColor(im_origen, im_destino, formato)

https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html