Automatización y control Clasificación por visión artificial.

PROYECTO FINAL

Introducción

La visión artificial se aplica en distintos campos un ejemplo la medicina en la cual se usa mucho para hacer comparaciones de radiografías, resonancias magnéticas y tomografías. También se aplica en la biología debido a que se pueden distinguir entre aplicaciones microscópicas y macroscópicas. Incluso para el reconocimiento y clasificación de objetos que de hecho se usara en la realización de este proyecto para poder identificar un objeto según el tamaño y su forma.

Sin duda alguna la importancia que genera la inteligencia artificial es muy grande en la actualidad y por eso hemos querido implementarla en un sistema de reconocimiento de objetos óptimo que al momento de querer reconocer imágenes nos dé un excelente resultado, y pueda satisfacer nuestras necesidades de acuerdo al problema que tengamos en el momento.

Objetivos

General

Implementar un sistema de selección a escala que permita la clasificación dependiendo de sus características.

Especificos.

- Diseñar un sistema de control y clasificación de objetos mediante la implementación de un sistema de visión artificial.

- Minimizar los costos de operación del proceso mediante la implementación del sistema.

- Realizar inspecciones en procesos donde existe diversidad de piezas

Visión Artificial

Es una rama de la inteligencia artificial que tiene por objetivo modelar matemáticamente los procesos de percepción visual en los seres vivos y generar programas que permitan simular estas capacidades visuales por computadora. La visión artificial permite la detección automática de la estructura y propiedades de un posible mundo dinámico en 3 dimensiones a partir una o varias imágenes bidimensionales del mundo. Las imágenes pueden ser monocromáticas o a color; pueden ser capturadas por una o varias cámaras, y cada cámara puede ser estacionaria o móvil. La estructura y propiedades del mundo tridimensional que se intentan deducir en la visión

artificial incluyen no sólo propiedades geométricas (tamaños, formas, localización de objetos, etc.), sino también propiedades del material (sus colores, sus texturas, la composición, etc.) y la luminosidad u oscuridad de las superficies.

Herramientas utilizadas

Para la realización de este proyecto se utilizaron una serie de herramientas las cuales fueron de gran utilidad para la realización del proyecto entre esas:

python

Python es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

El intérprete de Python y la extensa biblioteca estándar están a libre disposición en forma binaria y de código fuente para las principales plataformas desde el sitio web de Python, http://www.python.org/, y puede distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de Python de terceros, programas y herramientas, y documentación adicional.

OpenCV

Librería desarrollada por Intel para el uso de visión artificial la cual maneja detección de rostros e imágenes en tiempo real. Esta librería ha sido usada desde el manejo de sistemas de seguridad para poder detectar el movimiento hasta el reconocimiento de objetos. Actualmente esta librería es gratuita debido a que la licencia es tipo BSD (Libre Uso), esto nos permitiría hacer uso de esta en la parte comercial e investigativa.

Herramientas utilizadas

-potencimetro de tres patillas

-motor de carro

-LM317

-material MDF

-tornillos-

-balineras de rodamiento

-resistencias

-rodillos torniados

-material poroso

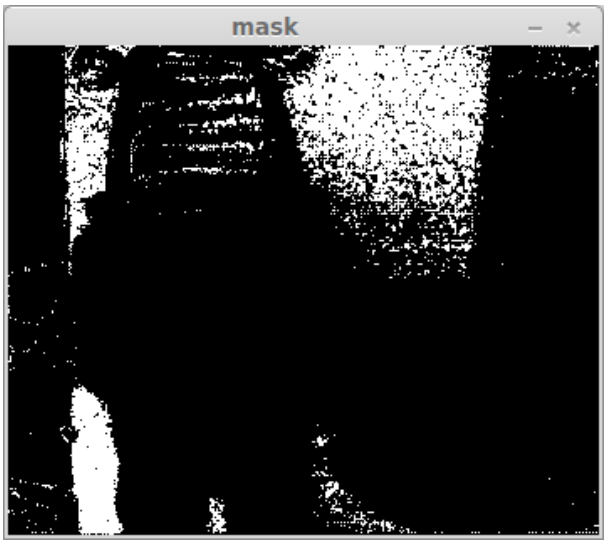
-base MDF

Resumen

En el presente proyecto se diseñara e implementara un sistema de clasificación de objetos por medio de visión artificial para realizar su debido control, El funcionamiento básico del sistema es emplear una banda transportadora para realizar el transporte de los objetos de los colores básicos que componen en modelo RGB que son rojo verde o azul, los cuales serán detectados al paso del sensor óptico de barrera y de esta forma capturar la imagen con la cámara, la cual será procesada por el software de ingeniería python-arduino en tiempo real y de esta forma clasificarla en alguno de los contenedores según el tipo de clasificación que se desee realizar en el momento ya sea por forma o color.

Los sistemas de visión artificial efectúan tareas repetitivas con precisión y rapidez y permiten trabajar fuera del espectro visible distinguiendo detalles no visibles por el ojo humano y aportando numerosos beneficios, siendo los más inmediatos el incremento de la calidad y del rendimiento de la producción y la reducción de costes de mano de obra.

cómo identificar objetos de color con OpenCV y Python. Uno de los “problemas” que tiene esta librería a la hora de trabajar con detección de colores es que hay infinitud de formas de hacerlo: por contornos, por momentos, calculando áreas, etc. Además, no nos basta con aplicar las funciones de detección de color, ya que se genera una gran cantidad de ruido del que habrá que encargarse para conseguir un resultado fiable.



Al analizar la imagen, se produce gran cantidad de ruido que deberemos eliminar

El concepto

Para reconocer y marcar un objeto de un color determinado seguiremos estos pasos:

1. **Capturar una imagen con la cámara**.
2. **Convertir la imagen de RGB a HSV**.
3. **Buscar objetos del color deseado**.
4. **Eliminar ruido**
5. **Mostrar la imagen**

---

1. Capturar Imagen

Cargamos las librerías cv2 y numpy. Activamos la cámara y guardamos lo que lee.

```
1  import cv2
```

```
2    import numpy as np
3
4    captura = cv2.VideoCapture(0)
```

---

2. Convertir la imagen

Creamos un bucle infinito (while(1) o while(true)). Dentro, leemos un frame y lo guardamos dentro de una variable que llamaremos ‘imagen’, después convertimos este frame a HSV, ya que es más fácil analizar imágenes en este modelo de color.

```
1    while(1):
2        _, imagen = captura.read()
3
4        hsv = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)
```

---

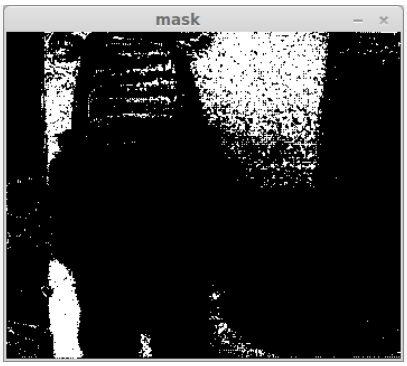
3. Buscar objetos verdes

Necesitamos dos arrays para guardar el rango de colores que detectamos. El límite inferior será 49, 50, 50, un verde oscuro. El límite superior será 80, 255, 255, un verde marino muy claro. Nuestro programa detectará todos los colores dentro de este rango.

```
1    verde_bajos = np.array([49,50,50])
2    verde_altos = np.array([80, 255, 255])
```

Necesitamos saber qué pixeles de la imagen están dentro del rango. Para ello crearemos una máscara que los guarde.

¿Pero qué es una máscara? Se trata de una imagen que contiene únicamente dos colores: blanco y negro. En nuestro caso pintará de blanco los píxeles verdes y de negro el resto. Por ejemplo, la imagen que he puesto al inicio es una máscara que detecta color verde:




```
1    mask = cv2.inRange(hsv, verde_bajos, verde_altos)
```

---

4. Eliminar ruido

Necesitamos desechar todos aquellos objetos que no lleguen a un tamaño determinado (ruido).

Para empezar calcularemos el momento de los objetos que hemos detectado. El momento se utiliza en matemáticas para encontrar la forma que adoptan un conjunto de puntos.

La función cv2.moments() nos da como output un diccionario. Nos interesa la clave ‘m00’, que guarda el valor del área del momento.

```
1    moments = cv2.moments(mask)
2    area = moments['m00']
3
4    #Para visualizar el area descomentamos la siguiente linea
5    #print area
```

A fin de eliminar el ruido, nos quedaremos únicamente con aquéllos objetos cuya área supere un determinado valor. Para ello utilizaremos un condicional. Después de jugar con distintos números encontré que 2000000 es una cifra bastante estándar que a mí me funciona.

Buscamos el centro del objeto en cuestión y mostramos sus coordenadas en pantalla. Para visualizarlo en la imagen dibujaremos un pequeño rectángulo rojo.

```
1      if(area > 2000000):
2        #Buscamos los centros
3        x = int(moments['m10']/moments['m00'])
```

```
4         y = int(moments['m01']/moments['m00'])
5
6         #Escribimos el valor de los centros
7         print "x = ", x
8         print "y = ", y
9
10        #Dibujamos el centro con un rectangulo
11        cv2.rectangle(imagen, (x, y), (x+2, y+2),(0,0,255), 2)
```

---

5. Mostrar la imagen

Mostraremos dos ventanas. En la primera aparecerá la imagen original con el centro del objeto. La segunda será la máscara en blanco y negro.

```
1     cv2.imshow('mask', mask)
2     cv2.imshow('Camara', imagen)
```

Con ***escape*** se cierra el programa:

```
1       #Se sale con ESC
2       tecla = cv2.waitKey(5) & 0xFF
3       if tecla == 27:
4          break
5
6     cv2.destroyAllWindows()
```

El código completo

He aquí el código completo que he escrito. Sóis libres de modificarlo para adaptarlo a vuestras necesidades.

```
1       #Algoritmo de deteccion de colores
2       #Detecta objetos verdes, elimina el ruido y busca su centro
3
4       import cv2
5       import numpy as np
6
7       #Iniciamos la camara
8       captura = cv2.VideoCapture(0)
9
```

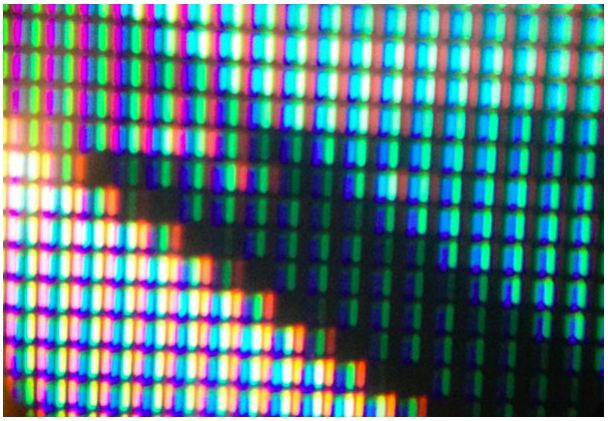
```
10  while(1):
11
12      #Capturamos una imagen y la convertimos de RGB -> HSV
13      _, imagen = captura.read()
14      hsv = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)
15
16      #Establecemos el rango de colores que vamos a detectar
17      #En este caso de verde oscuro a verde-azulado claro
18      verde_bajos = np.array([49,50,50], dtype=np.uint8)
19      verde_altos = np.array([80, 255, 255], dtype=np.uint8)
20
21      #Crear una mascara con solo los pixeles dentro del rango de verdes
22      mask = cv2.inRange(hsv, verde_bajos, verde_altos)
23
24      #Encontrar el area de los objetos que detecta la camara
25      moments = cv2.moments(mask)
26      area = moments['m00']
27
28      #Descomentar para ver el area por pantalla
29      #print area
30      if(area > 2000000):
31
32          #Buscamos el centro x, y del objeto
33          x = int(moments['m10']/moments['m00'])
34          y = int(moments['m01']/moments['m00'])
35
36          #Mostramos sus coordenadas por pantalla
37          print "x = ", x
38          print "y = ", y
39
40          #Dibujamos una marca en el centro del objeto
41          cv2.rectangle(imagen, (x, y), (x+2, y+2),(0,0,255), 2)
42
43
44      #Mostramos la imagen original con la marca del centro y
45      #la mascara
46      cv2.imshow('mask', mask)
47      cv2.imshow('Camara', imagen)
48      tecla = cv2.waitKey(5) & 0xFF
49      if tecla == 27:
```

```
50          break
51
52  cv2.destroyAllWindows()
53
54
55
```

Como creemos que a muchos nos está costando entender cómo funciona el espacio de color HSV y qué significan los arrays de valores máximos y mínimos que usamos en OpenCV para detectar colores, voy a dar una explicación más detallada sobre el tema. Además, voy a añadir un código en Python que me pasó un amigo para poder ajustar estos valores HSV en tiempo real con barras de desplazamiento.

¿Qué es el espacio de color HSV?

Cuándo vemos un color en la pantalla, lo que en realidad estamos viendo son miles de píxeles que brillan con una cierta intensidad. Cada píxel está formado por tres luces: una roja, una verde y una azul. La sensación de color se produce variando la intensidad de estas tres luces y alterando así la cantidad de luz roja, azul y verde que reciben los ojos del usuario.

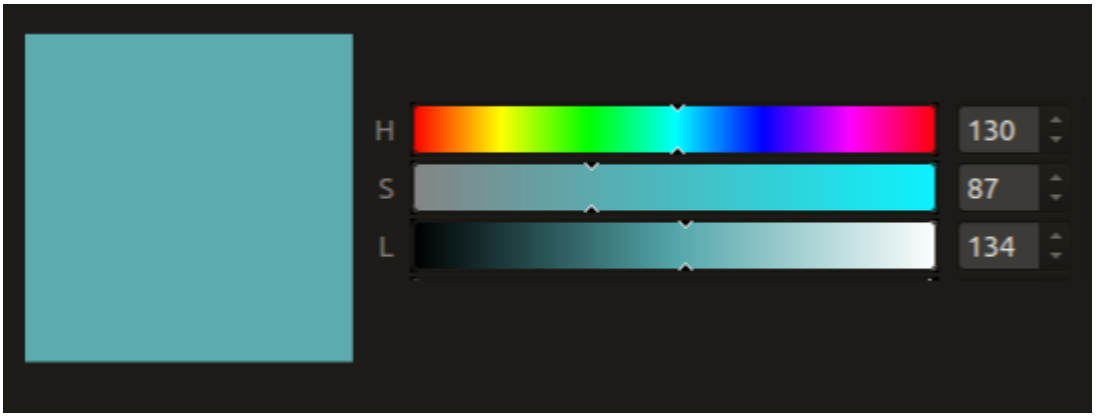


Dentro del ordenador, los colores vienen codificados por números. Hay varias formas de codificar los colores. La más popular es el sistema de color RGB, que sin duda todos conocéis. Este sistema asigna a cada color una cantidad de Rojo, Verde y Azul entre 0 y 255. Por ejemplo, el rojo puro es [255,0,0]. Hasta aquí todo correcto, ¿no?

Pero hay otras formas de codificar los colores. El **HSV** (del inglés Hue,Saturation,Value – Tono, Saturación y Valor) es ideal para reconocimiento de colores.

¿Qué significan estos tres valores?

- **Hue (H)** es el tono de color (por ejemplo verde, rojo, morado…)
- **Saturation (S)** es la intensidad de esta tonalidad. Cuanta menos saturación, más gris es el color.
- **Value** (**V**) es la luminosidad del color**.**



Ejemplo de color HSV

<u>Nota importante:</u> en algunos programas el rango de Hue es de 0 a 100 o de 0 a 360. También hay veces en que Saturation y Value van de 0 a 100. **El rango de OpenCV es de <u>0 a 255</u> para H, S y V.**

OpenCV: rango de valores máximo y mínimo

Conceptualmente, ¿cómo lo hace OpenCV para detectar colores? El programa busca los píxeles de la imagen que estén entre un valor mínimo y máximo.

Fíjemonos en esta parte del código del tutorial:

```
1       verde_bajos = np.array([49,50,50], dtype=np.uint8)
2       verde_altos = np.array([80, 255, 255], dtype=np.uint8)
3
4       #Crear una mascara con solo los pixeles dentro del rango de verdes
5       mask = cv2.inRange(hsv, verde_bajos, verde_altos)
```

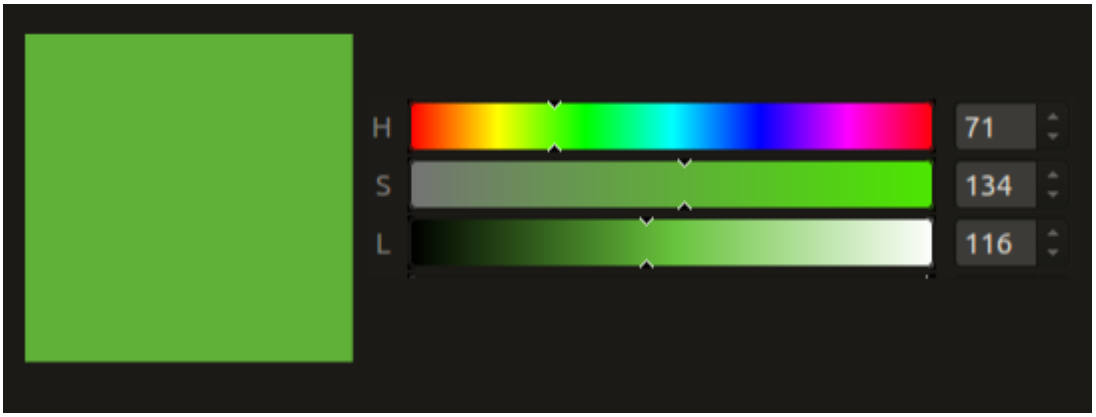
Las dos primeras líneas crean un array. El primero son los valores mínimos de H, S y V que buscará el programa, y el segundo son los valores máximos. En la última línea se llama la función para detectar colores. Recibe como parámetros la imagen (‘hsv’) y el array de valores mínimos y máximos.

Fíjemonos en el array verde_bajos y verde_altos:

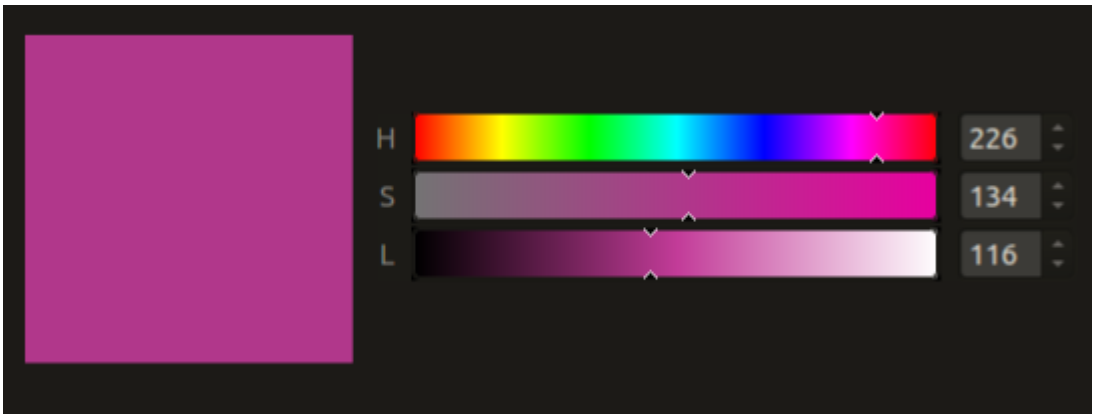
```
verde_bajos: [49,50,50]

verde_altos: [80, 255, 255]
```

Eso siginifica que el programa buscará píxeles que cumplan estas tres condiciones a la vez:

1. Su valor **Hue** esté entre **49 y 80**
2. Su valor **Saturation** esté entre **50 y 255**
3. Su valor **Value** esté entre **50 y 255**



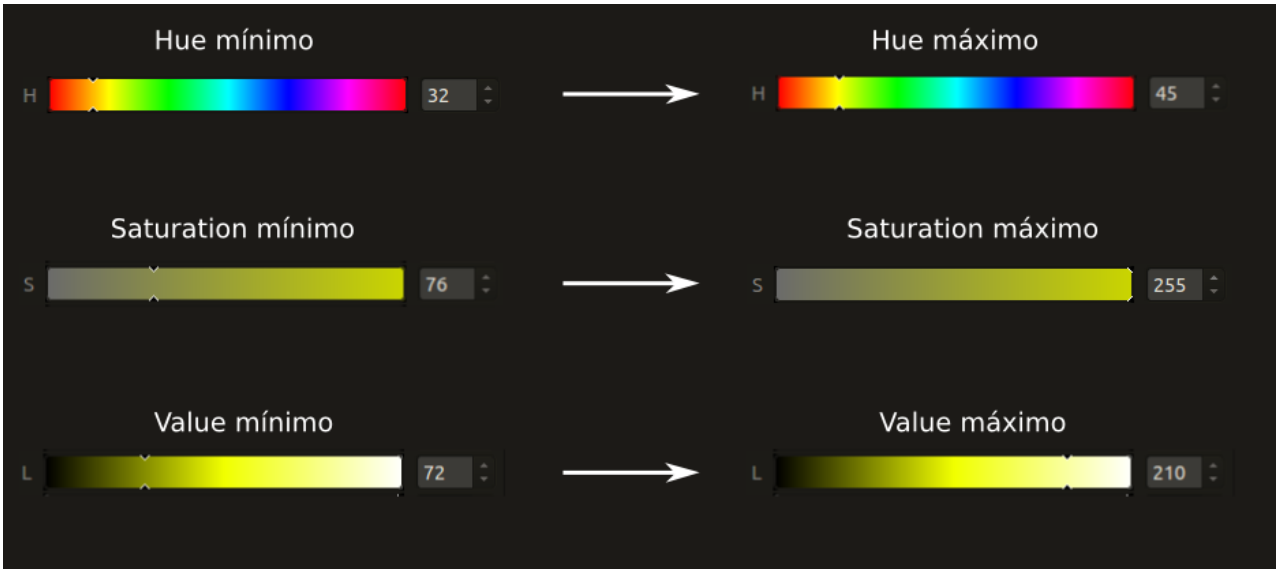
Este color tiene los valores H,S y V dentro del rango, por lo tanto será detectado



En cambio, este púrpura tiene el valor H fuera del rango. No será detectado.

Si queremos cambiar el color a detectar, tenemos que modificar los valores de los arrays. ¿Cómo podemos ver cuál es el rango HSV de nuestro color?

Lo más fácil es abrir un programa de diseño gráfico como Gimp, Inkscape o Photoshop (o Paint, si no tenéis orgullo) y mirar el rango de colores en la paleta. Supongamos que queremos detectar amarillos:



La otra opción es utilizar este código. Se trata de un código de detección de colores que permite ajustar los valores HSV en tiempo real mediante barras de desplazamiento. Hay que ajustarlas hasta que en la máscara aparezcan sólo los colores deseados.

```
1   import serial
2   import cv2
3   import numpy as np
4
5   cap = cv2.VideoCapture(0)
6
7   def nothing(x):
8     pass
9
10  #Creamos una ventana llamada 'image' en la que habra todos los sliders
11  cv2.namedWindow('image')
12  cv2.createTrackbar('Hue Minimo','image',0,255,nothing)
13  cv2.createTrackbar('Hue Maximo','image',0,255,nothing)
14  cv2.createTrackbar('Saturation Minimo','image',0,255,nothing)
15  cv2.createTrackbar('Saturation Maximo','image',0,255,nothing)
16  cv2.createTrackbar('Value Minimo','image',0,255,nothing)
17  cv2.createTrackbar('Value Maximo','image',0,255,nothing)
18
19  while(1):
20    _,frame = cap.read() #Leer un frame
21    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) #Convertirlo a espacio de color H
22
23    #Los valores maximo y minimo de H,S y V se guardan en funcion de la posicion de los
24    hMin = cv2.getTrackbarPos('Hue Minimo','image')
25    hMax = cv2.getTrackbarPos('Hue Maximo','image')
26    sMin = cv2.getTrackbarPos('Saturation Minimo','image')
27    sMax = cv2.getTrackbarPos('Saturation Maximo','image')
28    vMin = cv2.getTrackbarPos('Value Minimo','image')
29    vMax = cv2.getTrackbarPos('Value Maximo','image')
30
31    #Se crea un array con las posiciones minimas y maximas
32    lower=np.array([hMin,sMin,vMin])
33    upper=np.array([hMax,sMax,vMax])
34
35    #Deteccion de colores
36    mask = cv2.inRange(hsv, lower, upper)
37
38    #Mostrar los resultados y salir
```

```
39      cv2.imshow('camara',frame)
40      cv2.imshow('mask',mask)
41      k = cv2.waitKey(5) & 0xFF
42      if k == 27:
43        break
44    cv2.destroyAllWindows()
```

Utilidad de las cintas transportadoras

Las cintas transportadoras son **máquinas diseñadas para mover grandes volúmenes** de materiales a un ritmo mucho más rápido y a una fracción del precio que lo que se habría incurrido con la mano de obra manual.

Razones por las que utilizar las cintas transportadoras

Su uso más significativo puede ser observado en la industria alimentaria, donde se **usan sistemas transportadores tanto neumáticos como mecánicos** y en esta ocasión nos centraremos en las cintas transportadoras mecánicas.

Cintas transportadoras mecánicas

Las cintas transportadoras mecánicas también **se denominan caballos de carga de la industria** de los sistemas de manipulación de materiales.

Pueden ser inclinados, horizontales o verticales y se utilizan para transportar materiales a granel de un punto a otro, incluso los transportadores también pueden mover materiales de un nivel a otro con mucha facilidad.

El mismo trabajo, si lo hicieran los humanos, habría resultado ser un ejercicio muy caro y agotador, así que una de las mayores ventajas que ofrecen las cintas transportadoras es que **pueden instalarse prácticamente en cualquier luga**r para mover cargas de prácticamente todas las formas y tamaños.
En una planta de procesamiento de alimentos, los sistemas de cinta transportadora pueden usarse en una amplia variedad de formas, algunas de las cuales son las siguientes

- Movimiento de materias primas en dispositivos de almacenamiento dedicados
- Transporte de materias primas a la línea de producción
- Trasladar el producto fabricado a la línea de envasado
- Transporte de productos envasados a otra área para su envío.

Cintas transportadoras para diferentes sectores

Las cintas transportadoras que se utilizan en la industria de la manipulación de alimentos, se construyen e instalan de acuerdo con los principios de diseño sanitario, estas deben instalarse de manera que permita una fácil limpieza e inspección.

Es importante conocer la utilidad del cinturón, ya que estas son las cintas transportadoras más utilizadas y es que en la industria alimentaria, se utilizan para el procesamiento y **envasado de productos alimenticios,** siendo la superficie del cinturón muy suave, lo que lo hace ideal para la manipulación de alimentos.
Las cintas se hacen generalmente de materiales fuertes y duraderos tales como nilón, el poliéster y el caucho que se pueden limpiar fácilmente.

Utilidades de las cintas transportadoras

Las cintas transportadoras tienen muchas utilidades por lo que las podemos ver en muchos lugares, solo que no las detallamos.

Una de las mayores utilidades es en las fábricas, donde se transporta el producto de un lado al otro sin la necesidad de que esté una persona llevando el producto.

Pero también **se suelen ver en los aeropuertos,** donde se encargan de llevar las maletas hasta el avión y desde el avión hasta la cinta donde el dueño recogerá sus pertenencias.

Una de las utilidades más normales que podemos ver en la vida diaria es en los supermercados, ya que donde se encuentran las cajas, **suelen haber estas cintas para llevar los productos desde el carrito hasta la cajera.** Igualmente hay lugares que tienen estas cintas para trasladar personas, su utilidad es como la de una escalera mecánica pero de forma horizontal, con la finalidad de acelerar el paso.

CONTROLAR ARDUINO CON PYTHON Y LA LIBRERÍA PYSERIAL

Python es uno de los lenguajes de programación que mayor auge han experimentado en los últimos tiempos. Su sencillez de uso permite hacer rápidamente pequeños programas y scripts, con tiempos de desarrollo muy cortos.

Esta sencillez ha hecho que Python se gane un hueco en el Internet de las cosas (IoT), donde destaca por la facilidad para comunicar con diferentes

dispositivos (ordenadores, tablet, smarthones), tanto por cable, bluetooth, o Internet.

Por supuesto, el mundo de Arduino no resulta una excepción y resulta muy sencillo conectar Arduino con Python, empleando el puerto serie y la librería PySerial.