Mapeo del robot con encoders.

El encoder que se maneja en el carro da 600 pulsos por revolución y el diámetro de la llanta es de 6 cm. Por la fórmula 2\*pi\*r que es igual al perímetro de la llanta tenemos que:

L=2\*Pi\*r 🡪 2\*Pi\*3= 18.84954 cm.

Quiere decir que por cada revolución de la llanta se avanza 18.84 cm. Por regla de tres determinamos los pulsos requeridos para avanzar en la cuadrícula.

18.84954 🡪 600 pulsos

25 🡪 X pulso

Sabemos que cada cuadro por lado mide 25 cm, es decir que para recorrer la distancia de 25cm se necesitan 795.7753 pulsos aproximadamente. Con esto y la librería de motor\_turn determinamos el recorrido del carro sobre la cuadrícula usando los encoders.

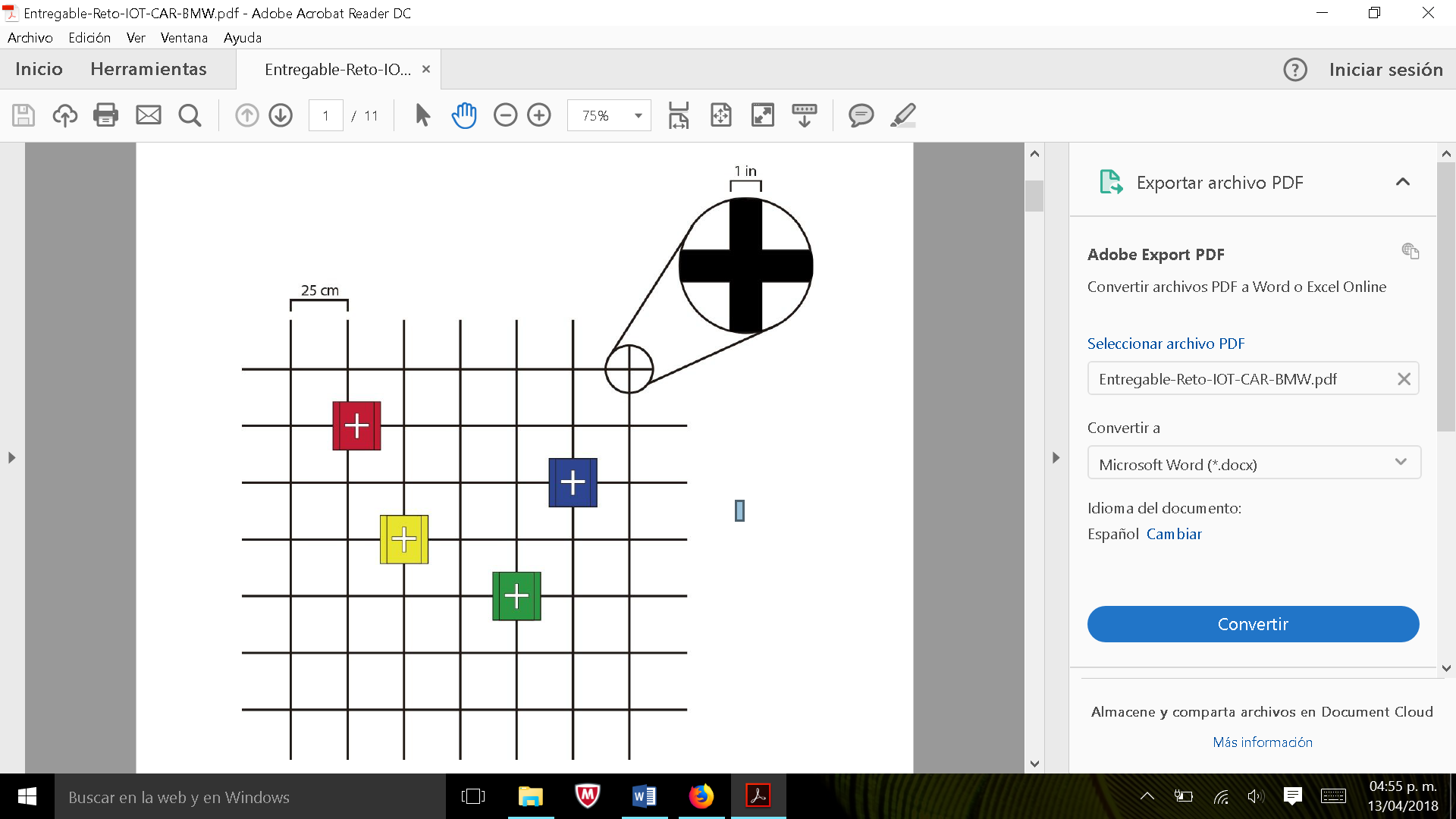


Figura . Tamaño de cuadros de 25 cm.

Para que el robot siga la línea recta, se hace una corrección con los sensores de piso al determinar el error cuando el robot no está centrando la línea. Para contar los cruces y las cajas, se checa cuando hay un cambio de color en los dos sensores de piso. Cuenta como cruce cuando ambos sensores están en negro.

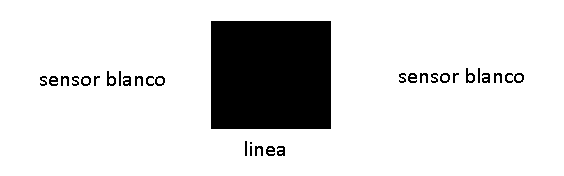


Figura 2. Segmento de línea y detección de sensores en color blanco.

Detección de línea y cruces usando Open CV.

También se probó determinar la corrección de línea recta usando la cámara y OpenCV. Se cambio el sentido de la cámara para que viera hacia la cuadrícula. Al adquirir la imagen, esta se transforma a escalas de grises y se pone un umbral binario ya que solo nos interesa detectar las líneas negras. Para determinar en que momento el robot debe corregir, se corta la imagen para solo distinguir una línea recta y un cruce como se ejemplifica a continuación.

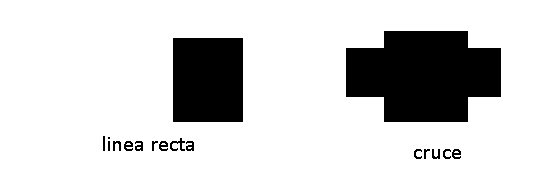


Figura . Imagen muestra de línea y cruce usando Open CV.

Para hacer la corrección y el conteo de cruces se saca el centroide de la imagen y el área de la figura detectada. Se usó la función findContours de Open Cv, para detectar los contornos en la imagen. Los contornos se pueden ejemplificar como una curva que une todos los puntos continuos (a lo largo del límite), teniendo el mismo color o intensidad.

En dicha función se usan tres parámetros donde el primero es la imagen de origen, el segundo es el modo de recuperación de contorno, el tercero es el método de aproximación de contorno y tiene como salida la imagen, los contornos y la jerarquía, con esto podemos detectar diferentes objetos en una imagen. En ocasiones los objetos están en diferentes lugares, pero en algunos casos, algunas formas están dentro de otras formas. Al igual que las figuras anidadas. En este caso, llamamos a uno externo como “padre” y uno interno como “hijo”. De esta forma, los contornos en una imagen tienen alguna relación entre sí. Y podemos especificar cómo un contorno está conectado entre sí, por ejemplo, si es “hijo” de algún otro contorno, o es un “padre”, etc.

La representación de esta relación se llama Jerarquía.

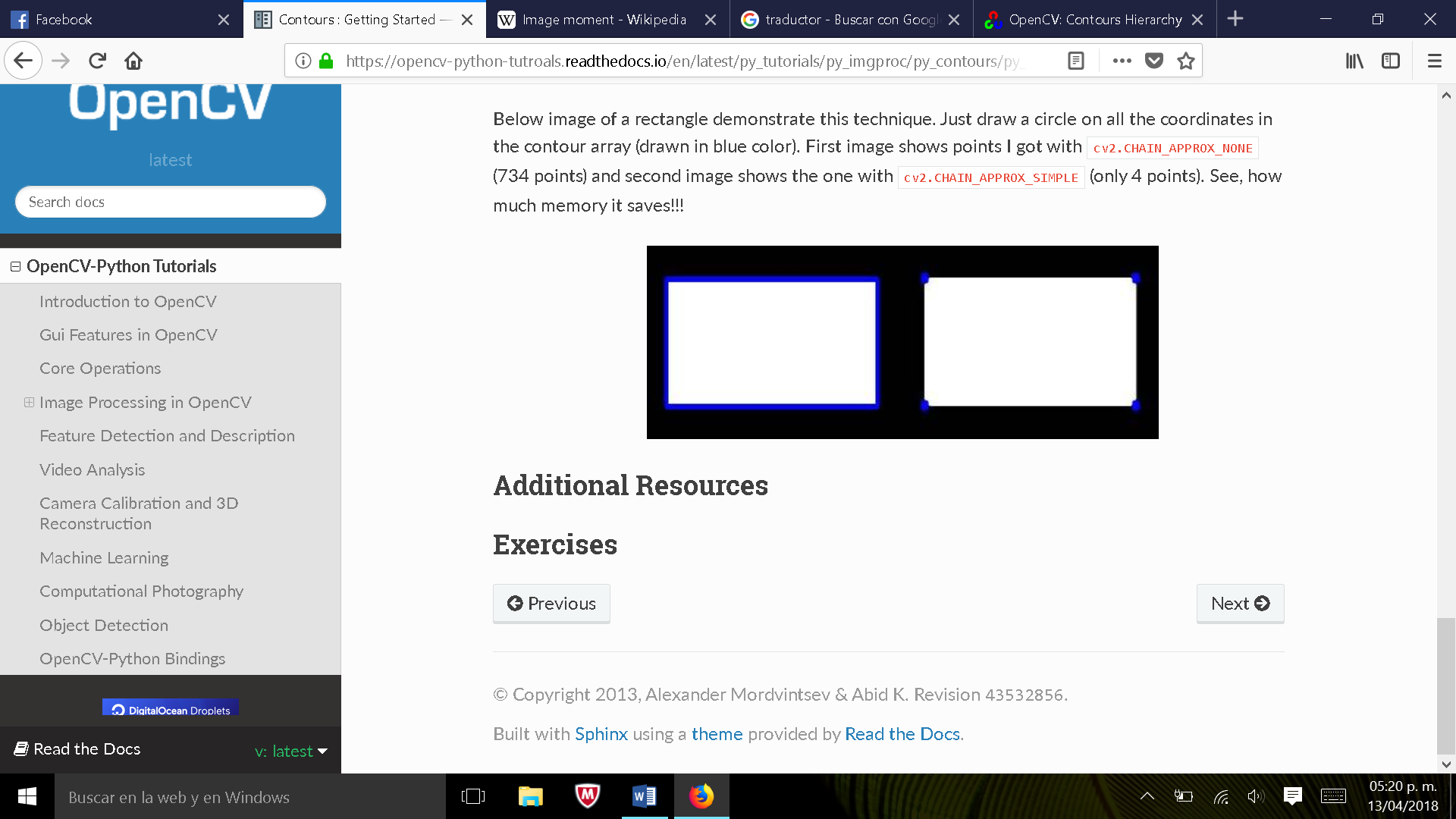


Figura 4. Detección de contorno de un rectángulo.

Teniendo los contornos obtenemos diferentes datos como área, perímetro, centroide, etc. Los momentos de imagen ayudan a calcular algunas características como el centro de masa y el área del objeto, etc.

La función cv2.moments () nos proporciona todos los valores de momento calculados. A partir de estos obtenemos el centroide que viene dado por las relaciones:

Ecuación. Cálculo de centroides del objeto.

Con la función cv2.contourArea() obtenemos el área del objeto. El área de las cruces debe ser mayor que el área de la línea recta. Con estos datos, obtenemos la posición de los cruces en la cuadrícula y los guardamos en la matriz de mapeo. Usando lo sensores piso verificamos que en el cruce no exista una caja de color, y si existe guarda la posición con un multiplicador que llamaremos peso para identificar una caja en la matriz.

Determinamos el error cada que el robot se mueve hacia alguno de los lados haciendo la resta del centroide en el eje Y menos 80 (pixeles de la línea) y del mismo modo se corrige usando las funciones de movimiento del motor.

En ambos métodos se emplea una corrección usando un control proporcional al error.

Detección de las cajas usando Open CV.

Para detectar las cajas, se hace un cambio de espacio de color (inicialmente BGR) a HSV para determinar el rango de valores de color de las cajas. El espacio de color HSV tiene las siguientes componentes.

H - Hue (longitud de onda dominante).

S - Saturación (Pureza / tonos del color).

V - Valor (Intensidad).

Lo que lo hace ideal para trabajar es que usa solo un canal para describir el color (H), por lo que es muy intuitivo especificar el color.

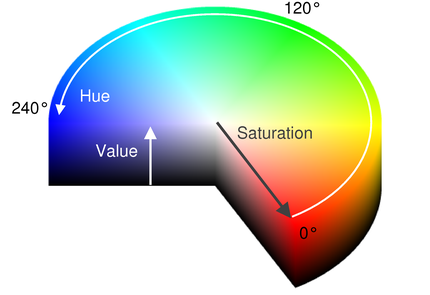


Figura 5. Espacio de color HSV.

En nuestro programa hacemos uso de un trackbar para detectar y calibrar el color de cada caja ya que estos valores varían dependiendo de la luz ambiente. Se usa la siguiente instrucción:

hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

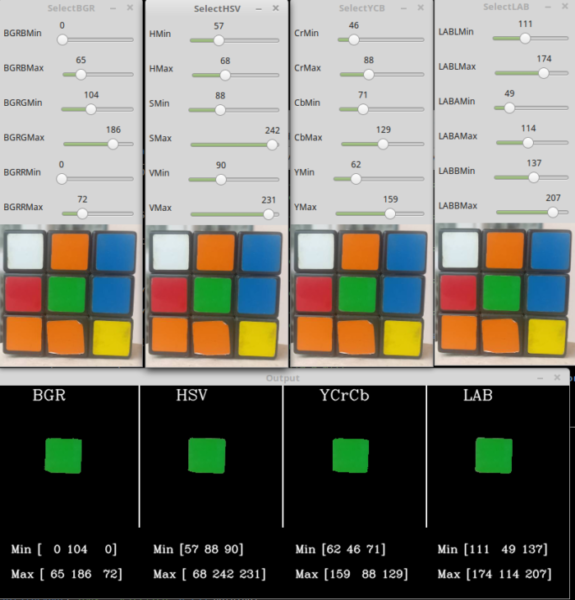
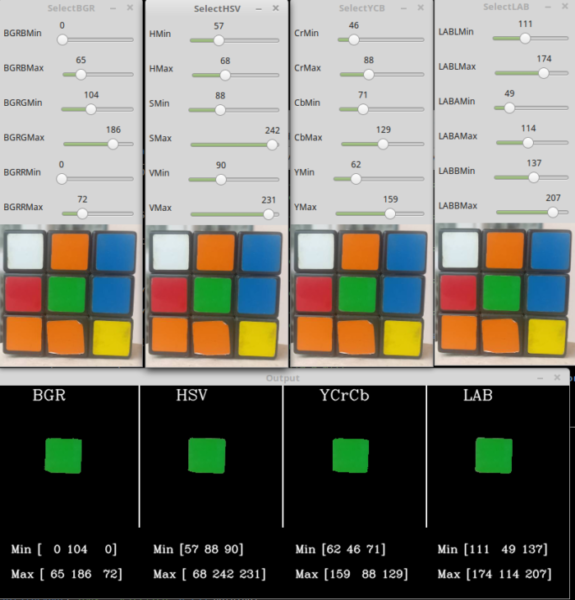


Figura 6. Ejemplo de calibración de color verde usando trackbars en el espacio HSV.

Se guardan los valores máximo y mínimo de H, S y V de cada caja. Al momento de implementar la detección de color de cada caja junto con el sensor ultrasónico que la encontraba y esto pasaba cuando había 5 cm de separación entre sensor y caja, por la sombras que el mismo carro generaba la detección de color no era del todo efectiva por lo cual declinamos por detectar las cajas usando la cámara.

**Código implementado**

# importa los paquetes a usar

from picamera.array import PiRGBArray

from picamera import PiCamera

import time

import cv2

 # inicializa la cámara y sus parámetros

camera = PiCamera()

camera.resolution = (640, 480) #resolución

camera.framerate = 32

rawCapture = PiRGBArray(camera, size=(640, 480))

 # tiempo de inicialización

time.sleep(0.1)

 # captura de los frames

for frame in camera.capture\_continuous(rawCapture, format="bgr", use\_video\_port=True):

image = frame.array

#cambio a escala de grises y umbral binario

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

#hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV) espacio HSV

thresh = cv2.threshold(gray, 60, 255, cv2.THRESH\_BINARY)[1]

#se corta la imagen

thresh = thresh[70:170, 440:540]

# encuentra los contornos de la imágen

cnts = cv2.findContours(thresh.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

for c in cnts:

# encuentra el centroide

M = cv2.moments(c)

cX = int(M["m10"] / M["m00"])

cY = int(M["m01"] / M["m00"])

# dibuja el centroide en la imagen

cv2.circle(image, (cX, cY), 7, (255, 255, 255), -1)

# cálculo del área

area = cv2.contourArea(c)

# muestra el frame

cv2.imshow("Frame", image)

key = cv2.waitKey(1) & 0xFF

# limpia para recibir el siguiente frame

rawCapture.truncate(0)

# salir en caso de que la letra q se apriete

if key == ord("q"):

break