Implementación del filtro de Kalman para el seguimiento de objetos

O. A. Agustín Aquino
Instituto de Física y Matemáticas
Universidad Tecnológica de la Mixteca
Huajuapan de León, Oaxaca, México
octavioalberto@mixteco.utm.mx

A. L. Rodríguez Santiago
División de Estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapan de León, Oaxaca, México
levid.rodriguez@gmail.com

E. López Jiménez
División de Estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapan de León, Oaxaca, México
ps2018280002@ndikandi.utm.mx

H. I. Hernández-Martínez
Instituto de Electrónica y Mecatrónica
Universidad Tecnológica de la Mixteca
Huajuapan de León, Oaxaca, México
hhdez@mixteco.utm.mx

***Abstract*—El filtro de Kalman propuesto en 1960, es una técnica de estimación de estados. Su aplicación incluye diversas áreas como son robótica, mecatrónica, informática y electrónica, entre otras, debido a que para su implementación requiere pocos recursos informáticos y proporciona gran capacidad para extraer información útil de datos ruidosos. El filtro de Kalman por lo general se centra en aplicaciones específicas como el movimiento de robots o la estimación de estados en sistemas lineales con ruido gaussiano.**

En este trabajo se presenta la implementación del filtro de Kalman mediante OpenCV y Python para el seguimiento de un objeto seleccionado en una escena proporcionada por una cámara web. Adicionalmente, con la información proporcionada por el software sobre el movimiento del objeto en pantalla, se programó un control reactivo a movimientos mediante la tarjeta de desarrollo Arduino Uno.

***Index Terms*—Filtro de Kalman, OpenCV, Python, Arduino.**

I. Introducción

El filtrado de Kalman es una técnica de estimación de estados propuesta por Rudolf E. Kálmán en 1960. Se usa en muchas áreas como: navegación de naves espaciales, planificación de movimientos en robótica, procesamiento de señales y redes de sensores inalámbricas, entre otras, debido a que para su implementación requiere pocos recursos informáticos y proporciona gran capacidad para extraer información útil de datos ruidosos. En la actualidad muchos trabajos muestran cómo usar el filtro de Kalman en controladores para sistemas informáticos [1] [2] [4] [6] [7] [8] [9] [10].

En la literatura se describen diversas formas de implementar el filtro de Kalman, que por lo general se centran en aplicaciones específicas como el movimiento de robots o la estimación de estados en sistemas lineales con ruido gaussiano. Lo anterior dificulta la implementación del filtro de Kalman a otros problemas, por lo que existen alternativas como son el filtro extendido de Kalman [5] [9] y el Filtro de partículas [3].

El filtro de Kalman se puede ver como un algoritmo para combinar estimaciones imprecisas de algún valor desconocido con la intención de obtener una estimación más precisa de dicho valor. Si las dos estimaciones son diferentes, como es probable, una solución es tomar el promedio de las dos estimaciones; si estas estimaciones son x_1 y x_2, se combinan utilizando la fórmula $0.5x_1 + 0.5x_2$ para asignar igual peso a las estimaciones.

Sin embargo, si se tiene información adicional se puede tener más confianza en alguna de las dos estimaciones, por ejemplo, se puede asignar más peso a la segunda estimación utilizando una fórmula como: $0.25x_1 + 0.75x_2$. En general, se puede considerar una combinación convexa de las dos estimaciones, mediante una fórmula de la forma: $(1 - \alpha)x_1 + \alpha x_2$, donde $0 \leq \alpha \leq 1$; intuitivamente, cuanta más confianza se tiene en la segunda estimación, α debería estar más cercano a 1. En el caso extremo, cuando $\alpha = 1$, se descarta la primera estimación y se usa solo la segunda.

La expresión $(1-\alpha)x_1+\alpha x_2$ es un ejemplo de un estimador lineal. Las estadísticas detrás del filtrado de Kalman indican cómo elegir el valor óptimo de α; el peso dado a una estimación debe ser proporcional a la confianza que se tiene en esa estimación. Para cuantificar estas ideas, es necesario formalizar los conceptos de estimaciones y confianza en las estimaciones. En el modelo que se utiliza del filtro de Kalman, las estimaciones se modelan como muestras aleatorias de ciertas distribuciones, y la confianza en las estimaciones se cuantifica en términos de las varianzas y covarianzas de dichas distribuciones.

Existen dos ideas estadísticas clave detrás del filtrado de Kalman:

1) El problema de fusionar más de dos estimaciones puede reducirse al problema de fusionar dos estimaciones a la vez, sin ninguna pérdida en la calidad de la estimación final. Los resultados obtenidos se pueden extender a estimaciones vectoriales.

Como los vectores de estado que representan la posición y la velocidad estimadas de un robot o nave espacial. Las matemáticas son más complicadas que en el caso escalar, pero las ideas básicas siguen siendo las mismas, excepto que, en lugar de trabajar con las varianzas de las estimaciones escalares, se debe trabajar con matrices de covarianza de las estimaciones vectoriales.

2) En algunas aplicaciones, las estimaciones son vectores, pero solo una parte del vector puede ser directamente observable. Por ejemplo, el estado de una nave espacial puede representarse por su posición y velocidad, pero solo su posición puede ser directamente observable.

A. Estimación vectorial del filtro de Kalman

En este trabajo las estimaciones de los estados del filtro de kalman son vectoriales. Por ejemplo, el estado de un robot que se mueve a lo largo de una sola dimensión podría estar representado por un vector que contenga su posición y velocidad. La matriz de covarianza de una variable aleatoria $\mathbf{x} : p(\mathbf{x})$ con media μ_x es la matriz $E[(x - \mu_x)(x - \mu_x)^T]$.

Estimaciones: una estimación x_i es una muestra aleatoria extraída de una distribución con media μ_i y matriz de covarianza Σ_i, escrita como $x_i : p_i \sim (\mu_i \Sigma_i)$. La inversa de la matriz de covarianza Σ_i^{-1} se denomina matriz de precisión o información. Se debe considerar que si la dimensión de x_i es uno, la matriz de covarianza se reduce a la varianza.

LEMA: Sea $\mathbf{x}_1 : p_1 \sim (\mu_1, \Sigma_1), ..., \mathbf{x}_n : p_n \sim (\mu_n, \Sigma_n)$ un conjunto de vectores aleatorios no correlacionados de pares de longitud m. Sea $\mathbf{y} = \Sigma_{i=1}^{n} A_i \mathbf{x}_i$. Entonces, la media y la varianza de $\mathbf{y}$ son las siguientes:

$$\mu_y = \sum_{i=1}^{n} A_i \mu_i \tag{1}$$

$$\Sigma_y = \sum_{i=1}^{n} A_i \Sigma_i A_i^T \tag{2}$$

II. Filtro de Kalman para sistemas lineales

Para calcular la evolución en el tiempo del estado de un sistema es necesario conocer con precisión el estado inicial x_0 y el modelo de la dinámica del sistema. El tiempo avanza en pasos discretos mientras que el estado del sistema en cualquier paso de tiempo es una función del estado del sistema en el paso de tiempo anterior y las entradas de control aplicadas al sistema durante ese intervalo. Esto generalmente se expresa mediante una ecuación de la forma $\mathbf{x}_{t+1} = f_t(\mathbf{x}_t, \mathbf{u}_t)$ donde $\mathbf{u}_t$ es la entrada de control. La función f_t no es lineal en el caso general. Si el sistema es lineal, la relación para la evolución del estado en el tiempo puede escribirse como $\mathbf{x}_{t+1} = F_t \mathbf{x}_t + B_t \mathbf{u}_t$, donde F_t y B_t son matrices (dependientes del tiempo) que pueden determinarse a partir de la física del sistema.

Por lo tanto, si el estado inicial $\mathbf{x}_0$ se conoce exactamente y la dinámica del sistema está modelada perfectamente por las matrices F_t y B_t, la evolución del estado con el tiempo se puede calcular con precisión. Sin embargo, es posible que no se conozca exactamente el estado inicial y que la dinámica del sistema y las entradas de control no se conozcan con precisión. Estas inexactitudes pueden hacer que los estados calculados y reales divergan de manera inaceptable a lo largo del tiempo. Para evitar esto, se pueden realizar mediciones del estado después de cada paso de tiempo. Si estas mediciones son exactas y se pudiera observar el estado completo después de cada paso de tiempo, no habría necesidad de modelar la dinámica del sistema. En general, (i) las mediciones en sí mismas son imprecisas, y (ii) algunos componentes del estado pueden no ser directamente observables mediante mediciones.

Si se puede observar el estado completo a través de mediciones, se tienen dos estimaciones imprecisas para el estado después de cada paso de tiempo, una desde el modelo de la dinámica del sistema y otra desde la medición. Si estas estimaciones no están correlacionadas y se conocen sus matrices de covarianza, se puede calcular la matriz de covarianza de esta estimación. La estimación del estado se introduce en el modelo del sistema para calcular la estimación del estado y su covarianza en el siguiente paso de tiempo, y se repite todo el proceso utilizando la siguiente notación.

- El estado inicial se denota por $\mathbf{x}_0$ y su covarianza por $\Sigma_{0|0}$.
- La incertidumbre en el modelo del sistema y las entradas de control se representan haciendo de $\mathbf{x}_{t+1|t}$ una variable aleatoria e introduciendo un término de ruido medio cero $\mathbf{w}_t$ en la ecuación de evolución del estado, que se convierte en

$$\mathbf{x}_{t+1|t} = F_t \mathbf{x}_{t|t} + B_t \mathbf{u}_t + \mathbf{w}_t \tag{3}$$

  La matriz de covarianza $\mathbf{w}_t$ se denota por Q_t y se supone que $\mathbf{w}_t$ no está correlacionada con $\mathbf{x}_{t|t}$.
- La medición imprecisa en el tiempo $t + 1$ se modela mediante una variable aleatoria

$$\mathbf{z}_{t+1} = \mathbf{x}_{t+1} + \mathbf{v}_{t+1} \tag{4}$$

  donde $\mathbf{v}_{t+1}$ es un término de ruido. $\mathbf{v}_{t+1}$ tiene una matriz de covarianza R_{t+1} y no está correlacionada con $\mathbf{x}_{t+1|t}$.

Se puede resumir que cuando se usa un modelo lineal para escalares, el peso que se le asigna a cada estimación debe ser inversamente proporcional a la varianza de esa estimación. Formalmente, los resultados para estimaciones escalares a menudo se expresan en términos de la ganancia de Kalman, como se muestra a continuación; estas ecuaciones se pueden aplicar recursivamente para fusionar múltiples estimaciones.

$$x_1 : p_1 \sim (\mu_1, \sigma_1^2), x_2 : p_2 \sim (\mu_2, \sigma_2^2)$$

$$K = \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} = \frac{\nu_2}{\nu_1 + \nu_2} \tag{5}$$

$$\widehat{y}(x_1, x_2) = x_1 + K(x_2 - x_1) \tag{6}$$

$$\mu_{\widehat{y}} = \mu_1 + K(\mu_2 + \mu_1) \tag{7}$$

$$\sigma^2_{\widehat{y}} = \sigma^2_1 - K\sigma^2_1 \quad o \quad \nu_{\widehat{y}} = \nu_1 + \nu_2 \tag{8}$$

A. Ejemplo: Estado 2D

La Figura 1 ilustra los pasos anteriores para un problema bidimensional en el que el vector de estado tiene dos componentes, y solo el primer componente se puede medir directamente. Se usa la siguiente notación simplificada para enfocarse en lo esencial.

- estimación del estado *a priori*: $\mathbf{x}_i = \binom{h_i}{c_i}$
- matriz de covarianza de la estimación *a priori*: $\Sigma_i = \begin{pmatrix} \sigma^2_h & \sigma_{hc} \\ \sigma_{ch} & \sigma^2_c \end{pmatrix}$
- estimación del estado *a posteriori*: $\mathbf{x}_0 = \binom{h_0}{c_0}$
- medición: z
- varianza de la medición: r^2

Siguiendo los tres pasos siguientes se pueden obtener la estimación *a posteriori*.

(i) La estimación *a priori* del estado observable es h_i. La estimación *a posteriori* se obtiene a partir de la ecuación 6.

$$h_0 = h_i + \frac{\sigma^2_h}{(\sigma^2_h + r^2)}(z - h_i) = h_i + K_h(z + h_i)$$

(ii) La estimación *a priori* del estado oculto es c_i. La estimación *a posteriori* se obtiene a partir de

$$\begin{aligned} c_0 &= c_i + \frac{\sigma_{hc}}{\sigma^2_h}\frac{\sigma^2_h}{(\sigma^2_h + r^2)}(z - h_i) \\ &= c_i + \frac{\sigma_{hc}}{(\sigma^2_h + r^2)}(z - h_i) = c_i + K_c(z - h_i) \end{aligned}$$

(iii) Al unirlos, se obtiene

$$\binom{h_0}{c_0} = \binom{h_i}{c_i} + \binom{K_h}{k_c}(z - h_i)$$

Como una generalización de este resultado es útil reescribir esta expresión usando matrices. Definir $H = (1 \quad 0)$ y $R = (r^2)$. Entonces $\mathbf{x}_0 = \mathbf{x}_i + K(z - H\mathbf{x}_i)$ donde $K = \Sigma_i H^T (H\Sigma_i H^t + R)^{-1}$.

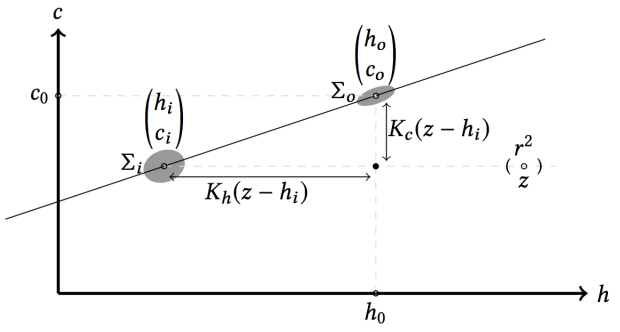


Fig. 1: Calculo de la estimación *a posteriori* cuando una parte del estado es no observable.

III. Implementación del filtro de Kalman

En resumen, el filtro de Kalman (KF) busca estimar el estado de un sistema lineal considerando que el ruido del proceso y la medición se pueden caracterizar mediante una distribución de probabilidad Gaussiana. El filtro de Kalman se basa en las siguientes suposiciones:

- El sistema es lineal y se tiene una ecuación de estado que describe su comportamiento dinámico y una ecuación de medición que describe la relación entre las variables de estado y la salida de los sensores. Ambas ecuaciones están descritas en tiempo discreto.
- Se conoce la estadística del ruido de medición y del ruido del sistema. Ambas son independientes entre sí, con media igual a cero y distribución de probabilidad normal (Gaussiana).

La ecuación de estado del sistema está dada por (3) y la ecuación de medición dada por (4). El algoritmo de Kalman calcula de manera recursiva (en tiempo real) la estimación de las variables de estado del sistema. Este algoritmo se encuentra dentro de un ciclo infinito en el que cada iteración del ciclo corresponde a un periodo de muestreo nuevo. El algoritmo tiene dos fases: a) la de predicción y b) la de corrección, modeladas por:

Predicción:

$$\mathbf{x}_{t+1} = F_t\mathbf{x}_t + B_t\mathbf{u}_t \tag{9}$$

$$\Sigma_{t+1} = A\Sigma_t A^T + Q \tag{10}$$

Corrección:

$$K = \Sigma_t H^T (H\Sigma_t H^T + R)^{-1} \tag{11}$$

$$\mathbf{x}_k = \mathbf{x}_t + K(z - H\mathbf{x}_t) \tag{12}$$

El *framework* de visión por computadora OpenCV [11], fue desarrollado en C/C++ y es compatible con diversos lenguajes de programacion, ofrece una implementación del filtro de Kalman. En este trabajo la implementación se desarrolla en lenguaje Python [12] de acuerdo al siguiente pseudocódigo.

```
Algoritmo Filtro_Kalman
        Leer Cuadro_delimitador
        Leer Nombre_ventanas
        Si Archivo_Video Entonces
                Leer Archivo_Video
        SiNo
                Leer Camara
        FinSi
        Objeto <- Seleccion_Objeto
        Matriz_Medicion <- Medicion_Inicial_Objeto
        Matriz_Prediccion <-
            Prediccion_Inicial_Objeto
        Matriz_Covarianza <-
            Covarianza_Inicial_Objeto
        Mientras Exista_Objeto_Escena Hacer
                Matriz_Medicion <- OpenCV[
                    Matriz_Medicion]
                Matriz_Prediccion <- OpenCV[
                    Matriz_Prediccion]
                Matriz_Covarianza <- OpenCV[
                    Matriz_Covarianza]
                Escribir Cuadros_Frame
                Escribir Ventanas
        FinMientras
FinAlgoritmo
```

A. Codificación en Python

En Python, la estructura de la función de filtro de Kalman $cv :: KalmanFilter :: KalmanFilter()$ se escribe de la siguiente manera:

```
<KalmanFilter object>=cv.KalmanFilter()
<KalmanFilter object>=cv.KalmanFilter(dynamParams,
    measureParams[, controlParams[, type]])
```

En donde $KalmanFilter$ recibe como parámetros: $dynamParams$, el cual representa la dimensionalidad del estado; $measureParams$, la dimensionalidad de la medición; $controlParams$, la dimensionalidad del vector de control; y $type$, es el tipo de matrices creados, el cual puede ser CV_32F o CV_64F.

La función $correct()$ actualiza el estado de predicción a partir de la medición, en Python se escribe de la siguiente manera:

```
retval=cv.KalmanFilter.correct(measurement)
```

La función $predict$ calcula el estado de predicción.

```
retval=cv.KalmanFilter.predict([,control])
```

A continuación se describe el código, en Python, utilizado para el filtro de Kalman. En esta primera parte del código se definen las bibliotecas de OpenCV que servirán para realizar el procesamiento de las imágenes. Para las operaciones numéricas y funciones matemáticas se usan $math$ y $numpy$, respectivamente. Es importante mencionar la utilidad de $importserial$ para realizar la comunicación entre Python y Arduino, a través de él se envían y se reciben órdenes para realizar algún tipo de acción.

```
#En este programa se implementa el filtro de kalman
    mediante opencv
import numpy as np
import math
import argparse
import sys
import cv2
import serial
```

En la siguiente parte del código se muestra la inicialización de la cámara de la computadora para acceder a ella. Además, se define que se usará en modo video.

```
procesamiento = True;
selecc_en_progreso = False; #apoyo a la seleccion de
    region en el video
pantalla_completa = False; # modo full screen
parser = argparse.ArgumentParser(description='
    Perform ' + sys.argv[0] + ' example operation on
    incoming camera/video image')
parser.add_argument("-c", "--camera_to_use", type=
    int, help="specify camera to use", default=0)
parser.add_argument('video_file', metavar='
    video_file', type=str, nargs='?', help='specify
    optional video file')
args = parser.parse_args()
```

En la siguiente parte del código se explica la selección de la región usando el ratón de la computadora. Se crea una función *on_mouse*, la cual determina la posición en la que se colocará el cursor del ratón.

```
#selecciona una region con el mouse
boxes = []
current_mouse_position = np.ones(2, dtype=np.int32);
def on_mouse(event, x, y, flags, params):
global boxes;
global selecc_en_progreso;
current_mouse_position[0] = x;
current_mouse_position[1] = y;
if event == cv2.EVENT_LBUTTONDOWN:
boxes = [];
sbox = [x, y];
selecc_en_progreso = True;
boxes.append(sbox);
elif event == cv2.EVENT_LBUTTONUP:
ebox = [x, y];
selecc_en_progreso = False;
boxes.append(ebox);
```

En la siguiente parte del código se muestra el centro del objeto de interés mediante un conjunto de puntos y representado por un rectángulo.

```
#regresa el centro de un conjunto de puntos
    representando un rectangulo
def center(points):
x = np.float32((points[0][0] + points[1][0] + points
    [2][0] + points[3][0]) / 4.0)
y = np.float32((points[0][1] + points[1][1] + points
    [2][1] + points[3][1]) / 4.0)
return np.array([np.float32(x), np.float32(y)], np.
    float32)
def nothing(x):
pass
```

A continuación, se define el objeto de captura y se inicializa el video a través de la cámara. A su vez, se define el nombre de las ventanas que contendrá el *frame*.

```
#define el objeto de captura en el video
cap = cv2.VideoCapture();
#define el nombre de la ventana
windowName = "Reconocimiento de objeto con kalman";
    # nombre de ventana
windowName2 = "proyeccion del histograma"; # nombre
    de ventana
windowNameSelection = "region seleccionada inicial";
```

La matriz de medición está dada mediante el siguiente arreglo.

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{vmatrix}$$

La matriz de transición está dada por:

$$\begin{vmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

La matriz de covarianza del ruido está dada por el siguiente arreglo:

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

En las siguientes líneas de código se inicia con el filtro de Kalman, definiendo cada función como se explica previamente.

```
#Inicia filtro de kalman para el objeto seleccionado
kalman = cv2.KalmanFilter(4,2)
kalman.measurementMatrix = np.array([[1,0,0,0],
[0,1,0,0]],np.float32) #se define la matriz de
    medicion de kalman
kalman.transitionMatrix = np.array([[1,0,1,0],
[0,1,0,1],
[0,0,1,0],
[0,0,0,1]],np.float32)
kalman.processNoiseCov = np.array([[1,0,0,0],
[0,1,0,0],
[0,0,1,0],
[0,0,0,1]],np.float32) * 0.03
```

A continuación, se muestran dos mensajes para señalar el objeto a identificar y la aproximación.

```
print("\nImagen en observacion: Azul");
print("Prediccion con Kalman: Verde\n");
```

Se lee un archivo de video o, en su defecto, la cámara incorporada en la computadora.

```
if (((args.video_file) and (cap.open(str(args.
    video_file))))
or (cap.open(args.camera_to_use))):
```

En las siguientes lineas de código muestra la función para llamar al puntero del mouse en la ventana.

```
cv2.namedWindow(windowName, cv2.WINDOW_NORMAL);
cv2.namedWindow(windowName2, cv2.WINDOW_NORMAL);
cv2.namedWindow(windowNameSelection, cv2.
    WINDOW_NORMAL);
```

```
#llamada al mouse
cv2.setMouseCallback(windowName, on_mouse, 0);
cropped = False;
#criterios de terminacion para la busqueda, 10
    iteraciones, o
# mueve por al menos 1 pix
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.
    TERM_CRITERIA_COUNT, 10, 1 )
```

Se realiza la búsqueda del objeto a través de la selección del objeto en 10 iteraciones y moviendo al menos un pixel. Mientras realiza el procesamiento se verifica si es ejecutado éxitosamente, además, se mide el tiempo en la que tarda el procesamiento.

```
#criterios de terminacion para la busqueda, 10
    iteraciones, o
# mueve por al menos 1 pix
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.
    TERM_CRITERIA_COUNT, 10, 1 )
while (procesamiento):
#si el archivo de video es ejecutado exitosamente
    entonces lee el frame del video
if (cap.isOpened):
ret, frame = cap.read();
# inicia un timer (para ver cuanto tiempo tarda el
    procesamiento)
start_t = cv2.getTickCount();
```

Se selecciona una región con el *mouse* y se define el tamaño del *frame* para desplegar el objeto.

```
# selecciona una region usando el mouse y
    desplegarlo
if (len(boxes) > 1) and (boxes[0][1] < boxes[1][1])
    and (boxes[0][0] < boxes[1][0]):
crop = frame[boxes[0][1]:boxes[1][1],boxes[0][0]:
    boxes[1][0]].copy()
h, w, c = crop.shape;   # size of template
if (h > 0) and (w > 0):
cropped = True;
```

A continuación, se define la tonalidad, saturación y valor de la región a identificar. Además, se selecciona el rango de saturación para eliminar los valores de saturación muy bajos.

```
#Convertir la region a HSV (Tonalidad, Saturacion y
    valor)
hsv_crop =  cv2.cvtColor(crop, cv2.COLOR_BGR2HSV);
#Selecciona toda la saturacion (0->180) y Sat,
    elimina valores
#con muy baja saturacion o valor
mask = cv2.inRange(hsv_crop, np.array((0., float(
    s_lower),float(v_lower))), np.array((180.,float(
    s_upper),float(v_upper))));
#construye un histograma de tonalidad y saturacion y
     lo normaliza
crop_hist = cv2.calcHist([hsv_crop],[0, 1],mask
    ,[180, 255],[0,180, 0, 255]);
cv2.normalize(crop_hist,crop_hist,0,255,cv2.
    NORM_MINMAX);
```

En el siguiente código se delimita el cuadro en la ventana, después se recorta y se muestra en otra ventana la selección del objeto a la que se aplicará el filtro de Kalman.

```
#posicion inicial del objeto
track_window = (cuadro[0][0],cuadro[0][1],cuadro
    [1][0] - cuadro[0][0],cuadro[1][1] - cuadro
    [0][1]);
cv2.imshow(windowNameSelection,crop);
#reinicia la lista de cajas
cuadro = [];
```

Una vez realizado lo anterior, se define la posición de la selección para mostrar el cuadro, así como las coordenadas del rectángulo.

```
#reinicia la lista de cajas
cuadro = [];
#despliega la seleccion de la caja
if (selecc_en_progreso):
top_left = (cuadro[0][0], cuadro[0][1]);
bottom_right = (current_mouse_position[0],
    current_mouse_position[1]);
cv2.rectangle(frame,top_left, bottom_right,
    (0,255,0), 2);
```

Posteriormente, se realiza una condición para el recorte del objeto para convertir la imagen de entrada a una proyección basada en la tonalidad y la saturación.

```
if (cropped):
#convierte la imagen de entrada a HSV
img_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV);
# proyeccion del histograma basada en tonalidad y
    saturacion
img_bproject = cv2.calcBackProject([img_hsv],[0,1],
    crop_hist,[0,180,0,255],1);
cv2.imshow(windowName2,img_bproject);
#aplica camshift para predecir la nueva ubicacion (
    observacion)
```

```
ret, track_window = cv2.CamShift(img_bproject,
    track_window, term_crit);
#dibuja la observacion en la imagen en azul
x,y,w,h = track_window;
frame = cv2.rectangle(frame, (x,y), (x+w,y+h),
    (255,0,0),2);
```

Una vez realizado lo anterior, se muestra el punto de observación del centro del cuadro delimitador para usar el filtro de kalman, desde la corrección, predicción y finalmente pasarla en formato *cv*2.

```
#extrae el centro de la observacion como punto
pts = cv2.boxPoints(ret)
pts = np.int0(pts)
#usa el filtro de kalman correcto
kalman.correct(center(pts));
#obtiene la nueva prediccion de kalman
prediction = kalman.predict();
#dibula la prediccion en la imagen en verde
frame = cv2.rectangle(frame, (prediction[0]-(0.5*w),
    prediction[1]-(0.5*h)), (prediction[0]+(0.5*w),
    prediction[1]+(0.5*h)), (0,255,0),2);
```

De igual forma, se realiza lo mismo para la segunda ventana usando una máscara para convertir el color a la componente de tonalidad, saturación y valor. Por último, se define la condición para terminar el proceso, es decir, cerrar la ventana cuando se desea salir de la ejecución del programa.

```
else:
#se muestra la mascara usando SV del componente HSV
img_hsv =  cv2.cvtColor(frame, cv2.COLOR_BGR2HSV);
#selecciona todos los valores de 0->180
mask = cv2.inRange(img_hsv, np.array((0., float(
    s_lower),float(v_lower))), np.array((180.,float(
    s_upper),float(v_upper))));
cv2.imshow(windowName2,mask);
#despliega la imagen
cv2.imshow(windowName,frame);
cv2.setWindowProperty(windowName, cv2.
    WND_PROP_pantalla_completa, cv2.
    WINDOW_pantalla_completa & pantalla_completa);
#termina el timer en ms
stop_t = ((cv2.getTickCount() - start_t)/cv2.
    getTickFrequency()) * 1000;
key = cv2.waitKey(max(2, 40 - int(math.ceil(stop_t))
    )) & 0xFF;
if (key == ord('x')):
procesamiento = False;
elif (key == ord('f')):
pantalla_completa = not(pantalla_completa);
#cierra la ventana
cv2.destroyAllWindows()
else:
print("Camara no conectada o archivo no especificado
    ");
```

B. Integración con Arduino

Para el control de los desplazamientos se utilizó un servomotor Sg90 (ver figura 2), utilizando la tarjeta de desarrollo Arduino Uno (ver figura 3) para el control del actuador. Además, la programación necesaria para la comunicación del filtro de Kalman, programado en Python y los movimientos del servomotor.

El entorno de progrmacion Arduino (ver figura 4) es una interfaz sencilla, intuitiva y fácil de usar. La codificación se realizó en lenguaje C/C++ de forma intuitiva.

Fig. 2: Servomotor-Sg90. Figura tomada de la página oficial de Sg90

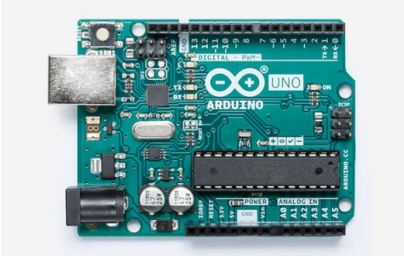


Fig. 3: Arduino Uno. Figura tomada de la página oficial de Arduino

El código del control para el servomotor se muestra a continuación y su implementación se muestra en la figura 5.

```
#include <Servo.h>
Servo servo_1;
// al presionar reset inicia la secuencia
void setup() {
  Serial.begin(9600);
  servo_1.attach(3);
}
// ciclo infinito
void loop() {
 if(Serial.available()) {
    char in = Serial.read();
 if(in == 'B'){
   servo_1.write(120);
 }
 if (in == 'W'){
    servo_1.write(0);
 }}}
```

IV. Experimentos y resultados

Como se ha mencionado, se usó Python y OpenCV, instalados previamente en el sistema operativo Ubuntu 18.04. Al ejecutar el programa se despliegan los comandos mostrados en la figura 6.

En las siguientes imágenes se muestra la implementación del filtro de Kalman. Se usó la cámara adherida a una computadora DELL G7. Primero, se seleccionó el objeto a aproximar con el *mouse* y posteriormente dicho objeto es aproximado con el filtro de Kalman. En las Figuras 7 se muestra en un cuadro verde la predicción del filtro y en un cuadro azul el objeto seleccionado.

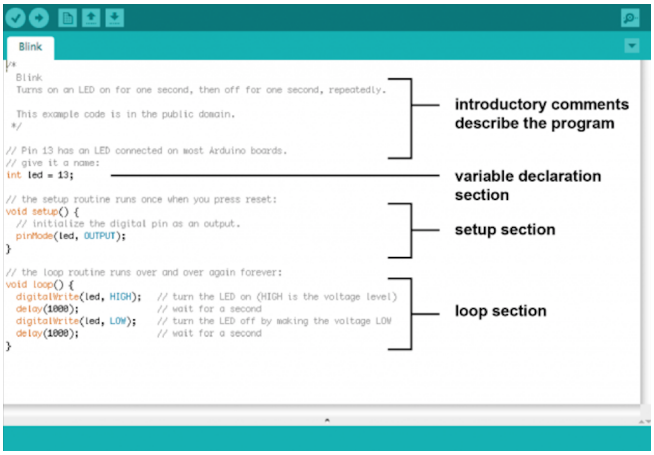


Fig. 4: IDE de Desarrollo de Arduino

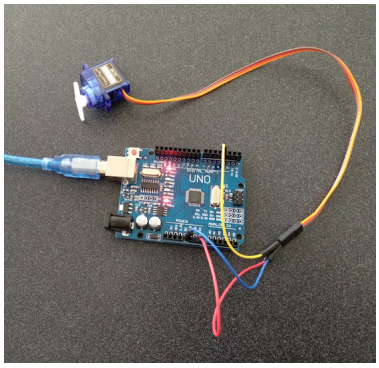


Fig. 5: Implementacion del control del servomotor mediante Arduino

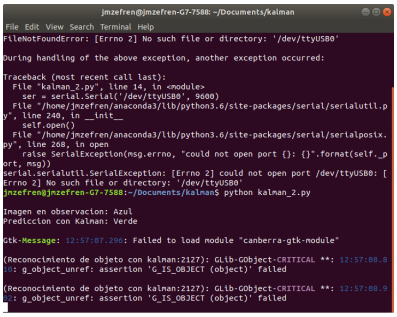
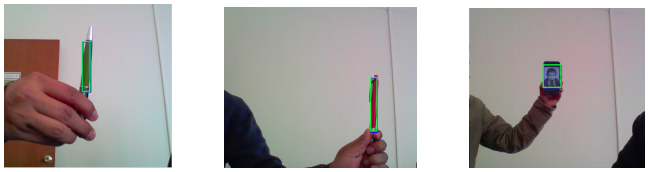


Fig. 6: Línea de comandos de Ubuntu

V. Conclusiones

Una vez que se ha estimado el modelo del sistema a controlar, es necesario determinar si el sistema se considera lineal o no lineal. Pues con base a esta consideración se aplica el filtro de Kalman o una de sus variantes. El filtro de Kalman discreto es robusto y proporciona resultados confiables para la estimación de las variables a controlar de sistemas lineales. Cabe destacar que se requiere de un correcto modelo del sistema a controlar, pues el filtro de Kalman puede presentar resultados poco confiables ante perturbaciones no Gaussianas de ruido.



(a) (b) (c)

Fig. 7: Aplicación del filtro de Kalman para seguimiento de un objetos

References

[1] Patel, Rajvi and Rajput, Manali and Parekh, Pramit, "Comparative study on multi-focus image fusion techniques in dynamic scene," International Journal of Computer Applications, vol. 109, no. 2, 2015.

[2] Lamsal, Bikash and Matsumoto, Naofumi, "Effects of the Unscented Kalman filter process for high performance face detector," International Journal of Information and Electronics Engineering, vol. 5, no. 6, p.454, 2015.

[3] Sánchez, Luis and Ordonez, Joan and Infante, Saba, "Filtro de Kalman extendido y filtro de partículas Kalman extendido para problemas de estimación No Lineal," Revista Ingeniería Universidad de Carabobo, vol. 20, no. 1, pp. 7–16, Abril 2013.

[4] Kumar, DVAN Ravi and Rao, S. Koteswara and Raju, K Padma, "Integrated Unscented Kalman filter for underwater passive target tracking with towed array measurements," Optik-International Journal for Light and Electron Optics, vol. 127, no. 5, pp.2840–2847, 2016.

[5] Atali, G and Garip, Z and Karayel, D and Ozkan, SS, "Localization of Mobile Robot using Odometry, Camera Images and Extended Kalman Filter," Acta Physica Polonica, A., vol. 134, no. 1, 2018.

[6] Shantaiya, Sanjivani and Verma, Kesari and Mehta, Kamal, "Multiple object tracking using kalman filter and optical flow," European Journal of Advances in Engineering and Technology, vol. 2, no. 2, pp.34–39, 2015.

[7] Raja, Thota Vinod and Tirupathamma, M, "Object Detection and Tracking in video using Kalman Filter," International Journal of Research In advanced Engineering Technology, vol. 5, no. 5, 2016.

[8] Ertürk, Sarp, "Real-time digital image stabilization using Kalman filters," Real-Time Imaging, vol. 8, no. 4, pp.317–328, 2002.

[9] Pascual, Alejandro, "EKF y UKF: dos extensiones del filtro de Kalman para sistemas no lineales aplicadas al control de un péndulo invertido," Monografía para el curso: Tratamiento Estadístico de Señales, p. 35, 2004.

[10] Pei, Yan and Biswas, Swarnendu and Fussell, Donald S and Pingali, Keshav, "An Elementary Introduction to Kalman Filtering," arXiv preprint arXiv:1710.04055, 2017.

[11] "OpenCV library," About - OpenCV library. [Online]. Available: https://opencv.org/. [Accessed: 24-Jan-2019].

[12] "Welcome to Python.org," Python.org. [Online]. Available: https://www.python.org/. [Accessed: 24-Jan-2019].