Task 1 Perception
Planning and Perception Module

Diploma in Robotics

Brayan Gerson Duran Toconas

November, 2025

Contents

1. Introduction

This report presents the imageprocessing pipeline used to detect and track the color markers of an autonomous soccer robot. The procedure includes Gaussian-based color thresholding, masking operations, circle detection, and the application of a Kalman filter to obtain a stable and reliable estimation of the robot's position.

Procedure

Only the most relevant and significant parts of the code will be described, focusing on their functionality.

Listing 1: Read the video

```
cap=cv2.VideoCapture(args.video_path)
```

Listing 2: Select the colour to calculate mean and covariance

```
def select_color_model(frame):
    r = cv2.selectROI("Seleccione la pelota", frame, fromCenter=False, showCrosshair=True)
    x, y, w, h = r
    roi = frame[y:y+h, x:x+w]
    hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
    pixels = hsv_roi.reshape((-1, 3))
    mean = np.mean(pixels, axis=0)
    cov = np.cov(pixels, rowvar=False)
    print("Media HSV:", mean)
    print("Covarianza HSV:", cov)
    cv2.destroyWindow("Seleccione la pelota")
    return mean, cov
print("Mark the colors...")
    for c in range(len(colors)):
        print(f"Color {c+1}: {colors[c]}")
        for i in range(args.captures):
            print(f"Captura {i+1} de {args.captures}")
            frame_number=int(input(f"Frame number of {int(cap.get(cv2.CAP_PROP_FRAME_COUNT))}: "))
            try:
                cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
                ret, frame=cap.read()
            except:
                print("Error: Not valid frame number")
                continue
            if not ret:
                print("Error: No se pudo leer el frame")
                continue
            men_i,cov_i=ball_tracker.select_color_model(frame)
            means[c].append(men_i)
            covs[c].append(cov_i)
```

Listing 3: Traing the gaussian mask from mean and covariance and apply morphological operations

```
def gaussian_mask(hsv, mean, cov, threshold=6.0):
    inv_cov = np.linalg.inv(cov + np.eye(3) * 1e-6)
```

```
    diff = hsv - mean.reshape((1, 1, 3))
    dist = np.sqrt(np.sum((diff @ inv_cov) * diff, axis=2))
    mask = (dist < threshold).astype(np.uint8) * 255
    return mask
mean_hsv[c] = np.mean(means[c], axis=0)
cov_hsv[c] = np.mean(covs[c], axis=0)
hsv=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
combined_mask = np.zeros(hsv.shape[:2], dtype=np.uint8)
for c in range(len(colors)):
  masks[c] = ball_tracker.gaussian_mask(hsv, mean_hsv[c], cov_hsv[c], args.threshold)
  masks[c]=cv2.erode(masks[c],kernel,iterations=1)
  masks[c]=cv2.dilate(masks[c],kernel,iterations=2)
  combined_mask = cv2.bitwise_or(combined_mask, masks[c])
frame_masked=cv2.bitwise_and(frame,frame,mask=combined_mask)
```

Listing 4: Individual color contour detections: all green contours are shown, along with one representative contour for blue and red.

```
for i, (name, color) in enumerate(draw_info):
            contours, _ = cv2.findContours(masks[i], cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            if contours:
                if name != "Green":
                    c = max(contours, key=cv2.contourArea) # max
                    ((x, y), radius) = cv2.minEnclosingCircle(c)
                    print(radius)
                    if radius > 9.0 and radius < 15.5: # 0.005
                        detected[name] = True
                        measured_x[name] = int(x)
                        measured_y[name] = int(y)
                        # Dibujo en pantalla
                        cv2.circle(frame, (measured_x[name], measured_y[name]), int(radius), color, 2)
                        cv2.putText(frame, name, (measured_x[name] + 10, measured_y[name]),cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, 1)
                else:
                    for c in contours:
                        ((x, y), radius) = cv2.minEnclosingCircle(c)
                        if radius > 8.0 and radius < 15.2:
                            detected[name] = True
                            if measured_x[name] is None:
                                measured_x[name] = []
                                measured_y[name] = []
                            measured_x[name].append(int(x))
                            measured_y[name].append(int(y))
                            cv2.circle(frame, (int(x), int(y)), int(radius),color, 2)
                            cv2.putText(frame, name, (int(x) + 10, int(y)), cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, 1)
```

Listing 5: Prediction and marking of the color and robot positions using a Kalman filter.

```
kalman_positions = {}
        for name, tracker in trackers.items():
            if name in ["Red", "Blue"]:
                pred_x, pred_y = tracker.predict()
                if detected[name]:
                    mx = measured_x[name]
                    my = measured_y[name]
```

```
                    tracker.correct(mx, my)
                    current_pos = (mx, my)
                else:
                    current_pos = (pred_x, pred_y)
                    cv2.putText(frame, f"{name} P(Ocluido)", (int(pred_x) + 10, int(pred_y)),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
                kalman_positions[name] = current_pos
                # Dibujo del marcador del Kalman
                cv2.drawMarker(frame, (int(current_pos[0]), int(current_pos[1])), (0, 0, 255), cv2.MARKER_CROSS, 20, 2)
            elif name == "Green":
                if detected[name]:
                    mx_list = measured_x[name]
                    my_list = measured_y[name]
                    if isinstance(mx_list, list):
                        for x, y in zip(mx_list, my_list):
                            cv2.drawMarker(frame, (x, y), (0, 0, 255), cv2.MARKER_TILTED_CROSS, 15, 2)
```

Listing 6: Rendering of the trail, mark, and Kalman filter updates.

```
# Draw Arrow from Red to Blue
        if "Red" in kalman_positions and "Blue" in kalman_positions:
            start_point = kalman_positions["Blue"]
            end_point = kalman_positions["Red"]
            # Extender la flecha
            scale = 3.0
            dx = end_point[0] - start_point[0]
            dy = end_point[1] - start_point[1]
            new_end_point = (int(start_point[0] + dx * scale), int(start_point[1] + dy * scale))
            cv2.arrowedLine(frame, start_point, new_end_point, (75, 70, 50), 6)
        # Update and Draw Trajectory for Blue only
        if "Blue" in kalman_positions:
            trajectory["Blue"].append(kalman_positions["Blue"])
            if len(trajectory["Blue"]) > 45: #15
                trajectory["Blue"].pop(0)
            for i in range(1, len(trajectory["Blue"])):
                if trajectory["Blue"][i - 1] is None or trajectory["Blue"][i] is None:
                    continue
                cv2.line(frame, trajectory["Blue"][i - 1], trajectory["Blue"][i], (200, 100, 120), 3)
```

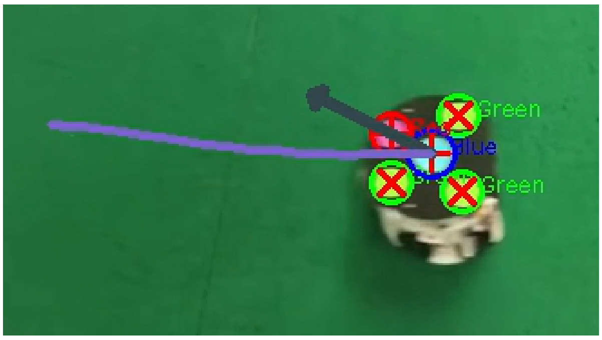
All previous operations are described in the following listing:

- The video is read and stored frame by frame.
- The HSV channels are used to compute the mean and covariance of the colors selected by the user for subsequent training.
- A Gaussian mask is computed and morphological closing is applied.
- The positions of the colored circles are detected using maximum contours and a radius constraint.

- The Kalman filter is computed for the blue and red circles to estimate the robot's position, and all color detections are visually marked.
- The robot's trajectory is drawn and the filter is updated.

2. Discussion and Results

The tracking results are as follows:



(a) Example 1 of tracking using the color markers.

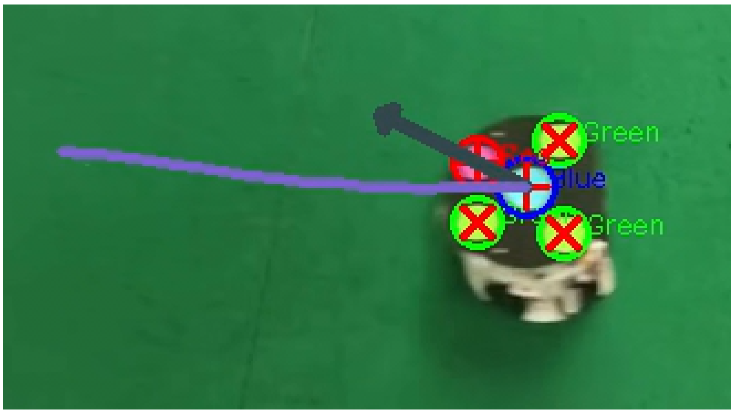
(b) Example of the Gaussian mask output.

Figure 1: Examples of function

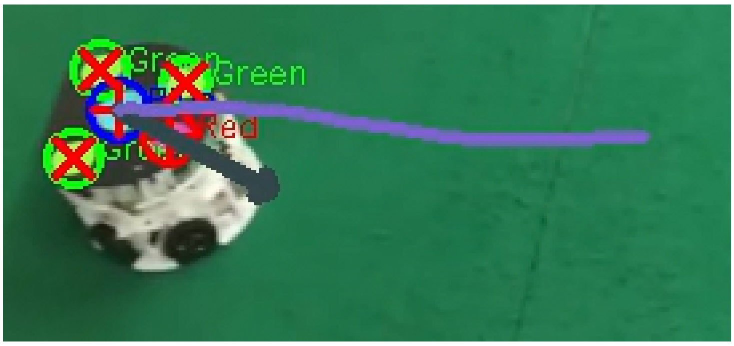
In Figure 1a, all colored circles are marked with an X indicating their detected positions, and each circle is also highlighted. Additionally, a Kalman filter is applied to the blue and red markers, and a line is drawn between their estimated positions. Figure 1b shows the corresponding Gaussian masks for all color channels.

Furthermore, Figures 2a and 2b show the trajectory of the robot based on the detection of its color markers. **To view the full video evidence, please visit the following link:**

`https://youtu.be/Wf4FbxHWbGE?si=8wJOW9GZnzo7OF9H`.



(a) Example 2 of tracking using the color markers.



(b) Example 3 of tracking using the color markers.

Figure 2: Examples of function

3. Conclusion

In conclusion, color detection remains consistent, and the application of a threshold computed from the Gaussian curve contributes significantly to its accuracy. This is further reinforced by circle detection and the use of HSV channels, which help mitigate the effects of lighting variations. Additionally, the application of a Kalman filter to predict the positions of the color markers and the robot smooths the tracking and prevents potential marker loss. Although marker loss did not occur during the tests, the Kalman filter clearly provides a smoother and less oscillatory motion prediction.

A. Codes

Listing 7: Python Funtions

```python
import cv2
import numpy as np
import argparse
from time import sleep
class KalmanTracker:
    def __init__(self):

        self.kf = cv2.KalmanFilter(4, 2)

        self.kf.measurementMatrix = np.array([[1, 0, 0, 0],
                                              [0, 1, 0, 0]], np.float32)

        self.kf.transitionMatrix = np.array([[1, 0, 1, 0],
                                             [0, 1, 0, 1],
                                             [0, 0, 1, 0],
                                             [0, 0, 0, 1]], np.float32)

        self.kf.processNoiseCov = np.array([[1, 0, 0, 0],
                                            [0, 1, 0, 0],
                                            [0, 0, 1, 0],
                                            [0, 0, 0, 1]], np.float32) * 0.001 #0.03  0.001

        self.kf.measurementNoiseCov = np.array([[1, 0],
                                                [0, 1]], np.float32) * 0.4 #1  0.7


    def predict(self):
        prediction = self.kf.predict()
        return (int(prediction[0]), int(prediction[1]))

    def correct(self, x, y):
        measurement = np.array([[np.float32(x)], [np.float32(y)]])
        self.kf.correct(measurement)


# --------------------------- NUEVA FUNCIÓN: selecciona ROI y aprende el color
def select_color_model(frame):
    r = cv2.selectROI("Seleccione la pelota", frame, fromCenter=False, showCrosshair=True)
    x, y, w, h = r
    roi = frame[y:y+h, x:x+w]

    hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
    pixels = hsv_roi.reshape((-1, 3))

    mean = np.mean(pixels, axis=0)
    cov = np.cov(pixels, rowvar=False)

    print("Media HSV:", mean)
    print("Covarianza HSV:", cov)

    cv2.destroyWindow("Seleccione la pelota")

    return mean, cov

```

```python

# --------------------------- NUEVA FUNCIÓN: máscara Bayesiana Gaussiana
def gaussian_mask(hsv, mean, cov, threshold=6.0):
    inv_cov = np.linalg.inv(cov + np.eye(3) * 1e-6)
    diff = hsv - mean.reshape((1, 1, 3))
    dist = np.sqrt(np.sum((diff @ inv_cov) * diff, axis=2))
    mask = (dist < threshold).astype(np.uint8) * 255
    return mask


def main():
    parser = argparse.ArgumentParser(description='Detector de pelota robusto con Kalman Filter + Modelo Bayesiano.')
    parser.add_argument('video_path', help='Ruta al archivo de video')
    parser.add_argument('--captures', type=int, default=1,
                        help='Número de capturas que hará el usuario para entrenar el color')
    parser.add_argument('--threshold', type=float, default=7.456,
                        help='Umbral para la máscara Bayesiana')

    args = parser.parse_args()

    cap = cv2.VideoCapture(args.video_path)

    if not cap.isOpened():
        print(f"Error: No se pudo abrir el video {args.video_path}")
        return

    tracker = KalmanTracker()
    trajectory = []

    # --------------------------- : capturar
    '''
    cap.set(cv2.CAP_PROP_POS_FRAMES, 20)
    ret, frame = cap.read()
    if not ret:
        print("Error al leer primer frame")
        return

    print("Seleccione la pelota para aprender el color...")
    mean_hsv, cov_hsv = select_color_model(frame)
    '''
    # ======== NUEVO: múltiples capturas para aprender el color ========

    means = []
    covs = []

    for i in range(args.captures):
        print(f"\n=== Captura {i+1}/{args.captures} ===")

        # Usuario elige frame
        frame_number = int(input("Número de frame a usar: "))

        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
        ret, frame = cap.read()
        if not ret:
            print("Error leyendo frame. Saltando captura...")
            continue

```

```
        mean_i, cov_i = select_color_model(frame)
        means.append(mean_i)
        covs.append(cov_i)

    # Modelo final promediado
    mean_hsv = np.mean(means, axis=0)
    cov_hsv = np.mean(covs, axis=0)

    print("\n=== MODELO FINAL ===")
    print("Media HSV:", mean_hsv)
    print("Covarianza HSV:", cov_hsv)
    sleep(5)



    print("Procesando video con Kalman Filter... Presiona 'q' para salir.")

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # --------------------------- NUEVO: máscara usando clasificación bayesiana
        mask = gaussian_mask(hsv, mean_hsv, cov_hsv, threshold=args.threshold) #9.0   7.456 17.456

        kernel = np.ones((5, 5), np.uint8)
        mask = cv2.erode(mask, kernel, iterations=2)
        mask = cv2.dilate(mask, kernel, iterations=2)

        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        detected = False
        measured_x, measured_y = 0, 0

        if contours:
            c = max(contours, key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)

            if radius > 0.005:     # ---------------------------
                detected = True
                measured_x, measured_y = int(x), int(y)

                cv2.circle(frame, (measured_x, measured_y), int(radius), (0, 255, 0), 2)
                cv2.putText(frame, "Pelota", (measured_x + 10, measured_y),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

        pred_x, pred_y = tracker.predict()

        if detected:
            tracker.correct(measured_x, measured_y)
            current_pos = (measured_x, measured_y)
        else:
            current_pos = (pred_x, pred_y)
            cv2.putText(frame, "P(Ocluido)", (pred_x + 10, pred_y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
```

```
            cv2.drawMarker(frame, (pred_x, pred_y), (0, 0, 255), cv2.MARKER_CROSS, 20, 2)

            trajectory.append(current_pos)
            if len(trajectory) > 15:
                trajectory.pop(0)

            for i in range(1, len(trajectory)):
                if trajectory[i - 1] is None or trajectory[i] is None:
                    continue
                cv2.line(frame, trajectory[i - 1], trajectory[i], (255, 0, 0), 2)

            cv2.imshow("Detector de Pelota (Kalman + Bayes)", frame)

            if cv2.waitKey(30) & 0xFF == ord('q'):
                break

        cap.release()
        cv2.destroyAllWindows()


    if __name__ == "__main__":
        main()
```

Listing 8: Main Script

```
#!/usr/bin/env python3
#Python script to track a robot soccer with 3 colors using Kalman Filter and Bayes

import cv2
import numpy as np
import argparse
import sys
import os
from time import sleep
os.environ["QT_QPA_PLATFORM"] = "xcb"
os.environ["QT_LOGGING_RULES"] = "*.warning=false"


# Add python_scripts to path relative to this file
# We need to go up from src -> task1_perception -> perception_and_planning_lab
script_dir = os.path.dirname(os.path.abspath(__file__))
python_scripts_path = os.path.join(script_dir, '../../python_scripts')
sys.path.append(python_scripts_path)

import ball_tracker

colors=["green","blue","red"] #Colors to track in the robot soccer

# Tabla fija de colores BGR para OpenCV
bgr_map = {
    "green": (0, 255, 0),
    "blue":  (255, 0, 0),
    "red":   (0, 0, 255)
}
trackers = {
    "Blue": ball_tracker.KalmanTracker(),
    "Green": ball_tracker.KalmanTracker(),
    "Red": ball_tracker.KalmanTracker()
}

# Crear draw_info automáticamente
draw_info = [(c.capitalize(), bgr_map[c]) for c in colors]

def main ():
    parser = argparse.ArgumentParser(description='Detector de pelota robusto con Kalman
  Filter + Modelo Bayesiano.')
    parser.add_argument('video_path', type=str, help='Path to video file')
    parser.add_argument('--captures', type=int, default=1,
                        help='Número de capturas que hará el usuario para entrenar el
  color')
    parser.add_argument('--threshold', type=float, default=7.456,
                        help='Umbral para la máscara Bayesiana')
    args = parser.parse_args()
    kalaman=ball_tracker.KalmanTracker()
    #kalaman.track_video(args.video_path, args.captures, args.threshold)

    args=parser.parse_args()

    kalaman=ball_tracker.KalmanTracker()
    trajectory = {name: [] for name in trackers.keys()}
    means=[[] for _ in range(len(colors))]
    covs=[[] for _ in range(len(colors))]
```

```
    mean_hsv=[[] for _ in range(len(colors))]
    cov_hsv=[[] for _ in range(len(colors))]


    cap=cv2.VideoCapture(args.video_path)
    if not cap.isOpened():
        print(f"Error: No se pudo abrir el video {args.video_path} . Revisa el path")
        return
    print("Mark the colors...")

    for c in range(len(colors)):
        print(f"Color {c+1}: {colors[c]}")

        for i in range(args.captures):
            print(f"Captura {i+1} de {args.captures}")
            frame_number=int(input(f"Frame number of {int(cap.get(cv2.CAP_PROP_FRAME_COUNT))}: "))
            try:
                cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
                ret, frame=cap.read()
            except:
                print("Error: Not valid frame number")
                continue

            if not ret:
                print("Error: No se pudo leer el frame")
                continue
            men_i,cov_i=ball_tracker.select_color_model(frame)
            means[c].append(men_i)
            covs[c].append(cov_i)

        print("\n")
        print("\n")

        mean_hsv[c] = np.mean(means[c], axis=0)
        cov_hsv[c] = np.mean(covs[c], axis=0)
        sleep(1)
        print("Media color: ",colors[c],mean_hsv[c])
        print("Cov color: ",colors[c],cov_hsv[c])
    sleep(3)

    print("Tracking...")
    #cap.set(0, frame_number)

    cap.set(cv2.CAP_PROP_POS_FRAMES, 2)
    hsv=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
    combined_mask = np.zeros(hsv.shape[:2], dtype=np.uint8)
    masks=[[] for _ in range(len(colors))]
    kernel=np.ones((5,5),np.uint8)

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        hsv=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
        combined_mask = np.zeros(hsv.shape[:2], dtype=np.uint8)
        for c in range(len(colors)):
            masks[c] = ball_tracker.gaussian_mask(hsv, mean_hsv[c], cov_hsv[c],
```

```
args.threshold)

            masks[c]=cv2.erode(masks[c],kernel,iterations=1)
            masks[c]=cv2.dilate(masks[c],kernel,iterations=2)
            combined_mask = cv2.bitwise_or(combined_mask, masks[c])

        frame_masked=cv2.bitwise_and(frame,frame,mask=combined_mask)

        cv2.imshow("frame_masked",frame_masked)
        #cv2.imshow("Masks",mask)
        #cv2.imshow("frame",frame)

        blue_contour,_=cv2.findContours(masks[1],cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

green_contours,_=cv2.findContours(masks[0],cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        red_contour,_=cv2.findContours(masks[2],cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

        # Diccionarios para guardar los resultados por color
        detected = { name: False for name, _ in draw_info }
        measured_x = { name: None for name, _ in draw_info }
        measured_y = { name: None for name, _ in draw_info }

        for i, (name, color) in enumerate(draw_info):
            contours, _ = cv2.findContours(masks[i], cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
            if contours:
                if name != "Green":
                    c = max(contours, key=cv2.contourArea) # max
                    ((x, y), radius) = cv2.minEnclosingCircle(c)
                    print(radius)
                    if radius > 9.0 and radius < 15.5: # 0.005

                        detected[name] = True
                        measured_x[name] = int(x)
                        measured_y[name] = int(y)

                        # Dibujo en pantalla
                        cv2.circle(frame, (measured_x[name], measured_y[name]),
int(radius), color, 2)
                        cv2.putText(frame, name, (measured_x[name] + 10,
measured_y[name]),cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, 1)
                else:
                    for c in contours:
                        ((x, y), radius) = cv2.minEnclosingCircle(c)

                        if radius > 8.0 and radius < 15.2:
                            detected[name] = True
                            if measured_x[name] is None:
                                measured_x[name] = []
                                measured_y[name] = []
                            measured_x[name].append(int(x))
                            measured_y[name].append(int(y))

                            cv2.circle(frame, (int(x), int(y)), int(radius),color, 2)
                            cv2.putText(frame, name, (int(x) + 10, int(y)),
cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, 1)
```

```
        #cv2.imshow("frame",frame)
        print("pos: ",measured_x,"   |   ",measured_y,"\n")

        #pred_x, pred_y = kalaman.predict()


        kalman_positions = {}

        for name, tracker in trackers.items():
            if name in ["Red", "Blue"]:
                pred_x, pred_y = tracker.predict()

                if detected[name]:
                    mx = measured_x[name]
                    my = measured_y[name]
                    tracker.correct(mx, my)
                    current_pos = (mx, my)
                else:
                    current_pos = (pred_x, pred_y)
                    cv2.putText(frame, f"{name} P(Ocluido)", (int(pred_x) + 10, int(pred_y)),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

                kalman_positions[name] = current_pos
                # Dibujo del marcador del Kalman
                cv2.drawMarker(frame, (int(current_pos[0]), int(current_pos[1])), (0, 0, 255), cv2.MARKER_CROSS, 20, 2)

            elif name == "Green":
                if detected[name]:
                    mx_list = measured_x[name]
                    my_list = measured_y[name]
                    if isinstance(mx_list, list):
                        for x, y in zip(mx_list, my_list):
                             cv2.drawMarker(frame, (x, y), (0, 0, 255), cv2.MARKER_TILTED_CROSS, 15, 2)

        # Draw Arrow from Red to Blue
        if "Red" in kalman_positions and "Blue" in kalman_positions:
            start_point = kalman_positions["Blue"]
            end_point = kalman_positions["Red"]

            # Extender la flecha
            scale = 3.0
            dx = end_point[0] - start_point[0]
            dy = end_point[1] - start_point[1]
            new_end_point = (int(start_point[0] + dx * scale), int(start_point[1] + dy * scale))

            cv2.arrowedLine(frame, start_point, new_end_point, (75, 70, 50), 6)

        # Update and Draw Trajectory for Blue only
        if "Blue" in kalman_positions:
            trajectory["Blue"].append(kalman_positions["Blue"])
            if len(trajectory["Blue"]) > 45: #15
                trajectory["Blue"].pop(0)

            for i in range(1, len(trajectory["Blue"])):
```

```
                    if trajectory["Blue"][i - 1] is None or trajectory["Blue"][i] is None:
                        continue
                    cv2.line(frame, trajectory["Blue"][i - 1], trajectory["Blue"][i], (200, 100
    , 120), 3)


            cv2.imshow("frame",frame)

            if cv2.waitKey(30) & 0xFF == ord('q'):
                break

        cap.release()
        cv2.destroyAllWindows()


if __name__ == '__main__':
    main()
```