

Task 1 Perception

Planning and Perception Module

Diploma in Robotics

Brayan Gerson Duran Toconas

November, 2025

Contents

1. Introduction	2
2. Discussion and Results	5
3. Conclusion	6
A. Codes	7

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

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

Listing 2: Select the colour to calculate mean and covariance

```
1 def select_color_model(frame):
2     r = cv2.selectROI("Seleccione la pelota", frame, fromCenter=False, showCrosshair=True)
3     x, y, w, h = r
4     roi = frame[y:y+h, x:x+w]
5     hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
6     pixels = hsv_roi.reshape((-1, 3))
7     mean = np.mean(pixels, axis=0)
8     cov = np.cov(pixels, rowvar=False)
9     print("Media HSV:", mean)
10    print("Covarianza HSV:", cov)
11    cv2.destroyAllWindows("Seleccione la pelota")
12    return mean, cov
13 print("Mark the colors...")
14 for c in range(len(colors)):
15     print(f"Color {c+1}: {colors[c]}")
16     for i in range(argscaptures):
17         print(f"Captura {i+1} de {argscaptures}")
18         frame_number=int(input(f"Frame number of
{int(cap.get(cv2.CAP_PROP_FRAME_COUNT))}: "))
19         try:
20             cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
21             ret, frame=cap.read()
22         except:
23             print("Error: Not valid frame number")
24             continue
25         if not ret:
26             print("Error: No se pudo leer el frame")
27             continue
28         men_i,cov_i=ball_tracker.select_color_model(frame)
29         means[c].append(men_i)
30         covs[c].append(cov_i)
```

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

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

```

3     diff = hsv - mean.reshape((1, 1, 3))
4     dist = np.sqrt(np.sum((diff @ inv_cov) * diff, axis=2))
5     mask = (dist < threshold).astype(np.uint8) * 255
6     return mask
7     mean_hsv[c] = np.mean(means[c], axis=0)
8     cov_hsv[c] = np.mean(covs[c], axis=0)
9     hsv=cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
10    combined_mask = np.zeros(hsv.shape[:2], dtype=np.uint8)
11    for c in range(len(colors)):
12        masks[c] = ball_tracker.gaussian_mask(hsv, mean_hsv[c], cov_hsv[c], args.threshold)
13        masks[c]=cv2.erode(masks[c],kernel,iterations=1)
14        masks[c]=cv2.dilate(masks[c],kernel,iterations=2)
15        combined_mask = cv2.bitwise_or(combined_mask, masks[c])
16    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.

```

1 for i, (name, color) in enumerate(draw_info):
2     contours, _ = cv2.findContours(masks[i], cv2.RETR_EXTERNAL,
3                                     cv2.CHAIN_APPROX_SIMPLE)
4     if contours:
5         if name != "Green":
6             c = max(contours, key=cv2.contourArea) # max
7             ((x, y), radius) = cv2.minEnclosingCircle(c)
8             print(radius)
9             if radius > 9.0 and radius < 15.5: # 0.005
10                 detected[name] = True
11                 measured_x[name] = int(x)
12                 measured_y[name] = int(y)
13                 # Dibujo en pantalla
14                 cv2.circle(frame, (measured_x[name], measured_y[name]),
15                            int(radius), color, 2)
16                 cv2.putText(frame, name, (measured_x[name] + 10,
17                               measured_y[name]), cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, 1)
18             else:
19                 for c in contours:
20                     ((x, y), radius) = cv2.minEnclosingCircle(c)
21                     if radius > 8.0 and radius < 15.2:
22                         detected[name] = True
23                         if measured_x[name] is None:
24                             measured_x[name] = []
25                             measured_y[name] = []
26                             measured_x[name].append(int(x))
27                             measured_y[name].append(int(y))
28                             cv2.circle(frame, (int(x), int(y)), int(radius),color, 2)
29                             cv2.putText(frame, name, (int(x) + 10, int(y)),
30                                       cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, 1)

```

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

```

1 kalman_positions = {}
2     for name, tracker in trackers.items():
3         if name in ["Red", "Blue"]:
4             pred_x, pred_y = tracker.predict()
5             if detected[name]:
6                 mx = measured_x[name]
7                 my = measured_y[name]

```

```

8             tracker.correct(mx, my)
9             current_pos = (mx, my)
10            else:
11                current_pos = (pred_x, pred_y)
12                cv2.putText(frame, f"{name} P(Ocuido)", (int(pred_x) + 10,
13                                         int(pred_y)),
14                                         cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
15                kalman_positions[name] = current_pos
16                # Dibujo del marcador del Kalman
17                cv2.drawMarker(frame, (int(current_pos[0])), int(current_pos[1])), (0, 0, 255
18 ), cv2.MARKER_CROSS, 20, 2)
19                elif name == "Green":
20                    if detected[name]:
21                        mx_list = measured_x[name]
22                        my_list = measured_y[name]
23                        if isinstance(mx_list, list):
24                            for x, y in zip(mx_list, my_list):
25                                cv2.drawMarker(frame, (x, y), (0, 0, 255),
26                               cv2.MARKER_TILTED_CROSS, 15, 2)

```

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

```

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

```

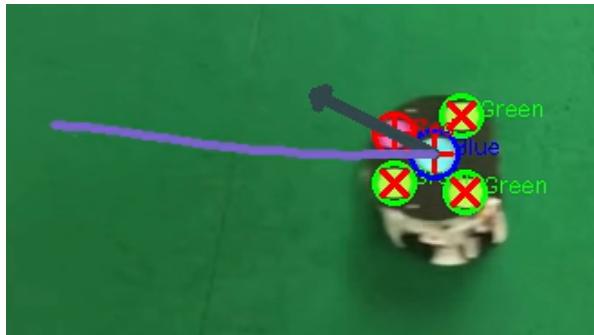
All previous operations are described in the following listing:

- 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.



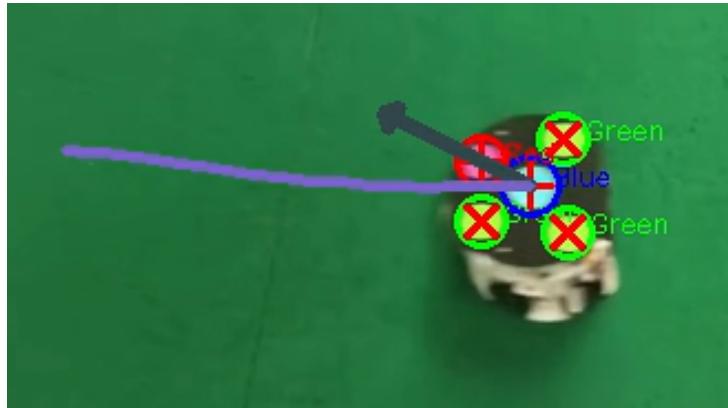
(b) Example of the Gaussian mask output.

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=8wJ0W9GZnzo70F9H>



(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 Functions

```

1 import cv2
2 import numpy as np
3 import argparse
4 from time import sleep
5 class KalmanTracker:
6     def __init__(self):
7
8         self.kf = cv2.KalmanFilter(4, 2)
9
10        self.kf.measurementMatrix = np.array([[1, 0, 0, 0],
11                                              [0, 1, 0, 0]], np.float32)
12
13        self.kf.transitionMatrix = np.array([[1, 0, 1, 0],
14                                              [0, 1, 0, 1],
15                                              [0, 0, 1, 0],
16                                              [0, 0, 0, 1]], np.float32)
17
18        self.kf.processNoiseCov = np.array([[1, 0, 0, 0],
19                                              [0, 1, 0, 0],
20                                              [0, 0, 1, 0],
21                                              [0, 0, 0, 1]], np.float32) * 0.001 #0.03 0.001
22
23        self.kf.measurementNoiseCov = np.array([[1, 0],
24                                              [0, 1]], np.float32) * 0.4 #1 0.7
25
26
27    def predict(self):
28        prediction = self.kf.predict()
29        return (int(prediction[0]), int(prediction[1]))
30
31    def correct(self, x, y):
32        measurement = np.array([[np.float32(x)], [np.float32(y)]])
33        self.kf.correct(measurement)
34
35
# ----- NUEVA FUNCIÓN: selecciona ROI y aprende el color
36 def select_color_model(frame):
37     r = cv2.selectROI("Seleccione la pelota", frame, fromCenter=False, showCrosshair=True)
38     x, y, w, h = r
39     roi = frame[y:y+h, x:x+w]
40
41
42     hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
43     pixels = hsv_roi.reshape((-1, 3))
44
45     mean = np.mean(pixels, axis=0)
46     cov = np.cov(pixels, rowvar=False)
47
48     print("Media HSV:", mean)
49     print("Covarianza HSV:", cov)
50
51     cv2.destroyAllWindows("Seleccione la pelota")
52
53     return mean, cov
54

```

```
55
56 # ----- NUEVA FUNCIÓN: máscara Bayesiana Gaussiana
57 def gaussian_mask(hsv, mean, cov, threshold=6.0):
58     inv_cov = np.linalg.inv(cov + np.eye(3) * 1e-6)
59     diff = hsv - mean.reshape((1, 1, 3))
60     dist = np.sqrt(np.sum((diff @ inv_cov) * diff, axis=2))
61     mask = (dist < threshold).astype(np.uint8) * 255
62     return mask
63
64
65 def main():
66     parser = argparse.ArgumentParser(description='Detector de pelota robusto con Kalman
67     Filter + Modelo Bayesiano.')
68     parser.add_argument('video_path', help='Ruta al archivo de video')
69     parser.add_argument('--captures', type=int, default=1,
70                         help='Número de capturas que hará el usuario para entrenar el color')
71     parser.add_argument('--threshold', type=float, default=7.456,
72                         help='Umbral para la máscara Bayesiana')
73
74     args = parser.parse_args()
75
76     cap = cv2.VideoCapture(args.video_path)
77
78     if not cap.isOpened():
79         print(f"Error: No se pudo abrir el video {args.video_path}")
80         return
81
82     tracker = KalmanTracker()
83     trajectory = []
84
85     # ----- : capturar
86     cap.set(cv2.CAP_PROP_POS_FRAMES, 20)
87     ret, frame = cap.read()
88     if not ret:
89         print("Error al leer primer frame")
90         return
91
92     print("Seleccione la pelota para aprender el color...")
93     mean_hsv, cov_hsv = select_color_model(frame)
94     '''
95     # ===== NUEVO: múltiples capturas para aprender el color =====
96
97     means = []
98     covs = []
99
100    for i in range(args.captures):
101        print(f"\n== Captura {i+1}/{args.captures} ==")
102
103        # Usuario elige frame
104        frame_number = int(input("Número de frame a usar: "))
105
106        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
107        ret, frame = cap.read()
108        if not ret:
109            print("Error leyendo frame. Saltando captura...")
110            continue
111
```

```

112     mean_i, cov_i = select_color_model(frame)
113     means.append(mean_i)
114     covs.append(cov_i)
115
116     # Modelo final promediado
117     mean_hsv = np.mean(means, axis=0)
118     cov_hsv = np.mean(covs, axis=0)
119
120     print("\n==== MODELO FINAL ===")
121     print("Media HSV:", mean_hsv)
122     print("Covarianza HSV:", cov_hsv)
123     sleep(5)
124
125
126
127     print("Procesando video con Kalman Filter... Presiona 'q' para salir.")
128
129     while True:
130         ret, frame = cap.read()
131         if not ret:
132             break
133
134         hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
135
136         # ----- NUEVO: máscara usando clasificación bayesiana
137         mask = gaussian_mask(hsv, mean_hsv, cov_hsv, threshold=args.threshold) #9.0    7.456
17.456
138
139         kernel = np.ones((5, 5), np.uint8)
140         mask = cv2.erode(mask, kernel, iterations=2)
141         mask = cv2.dilate(mask, kernel, iterations=2)
142
143         contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
144
145         detected = False
146         measured_x, measured_y = 0, 0
147
148         if contours:
149             c = max(contours, key=cv2.contourArea)
150             ((x, y), radius) = cv2.minEnclosingCircle(c)
151
152             if radius > 0.005:      #
153                 detected = True
154                 measured_x, measured_y = int(x), int(y)
155
156                 cv2.circle(frame, (measured_x, measured_y), int(radius), (0, 255, 0), 2)
157                 cv2.putText(frame, "Pelota", (measured_x + 10, measured_y),
158                             cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
159
160         pred_x, pred_y = tracker.predict()
161
162         if detected:
163             tracker.correct(measured_x, measured_y)
164             current_pos = (measured_x, measured_y)
165         else:
166             current_pos = (pred_x, pred_y)
167             cv2.putText(frame, "P(Ocubido)", (pred_x + 10, pred_y),
168                         cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

```

```
169
170     cv2.drawMarker(frame, (pred_x, pred_y), (0, 0, 255), cv2.MARKER_CROSS, 20, 2)
171
172     trajectory.append(current_pos)
173     if len(trajectory) > 15:
174         trajectory.pop(0)
175
176     for i in range(1, len(trajectory)):
177         if trajectory[i - 1] is None or trajectory[i] is None:
178             continue
179         cv2.line(frame, trajectory[i - 1], trajectory[i], (255, 0, 0), 2)
180
181     cv2.imshow("Detector de Pelota (Kalman + Bayes)", frame)
182
183     if cv2.waitKey(30) & 0xFF == ord('q'):
184         break
185
186     cap.release()
187     cv2.destroyAllWindows()
188
189
190 if __name__ == "__main__":
191     main()
192
193
```

Listing 8: Main Script

```

1 #!/usr/bin/env python3
2 #Python script to track a robot soccer with 3 colors using Kalman Filter and Bayes
3
4 import cv2
5 import numpy as np
6 import argparse
7 import sys
8 import os
9 from time import sleep
10 os.environ["QT_QPA_PLATFORM"] = "xcb"
11 os.environ["QT_LOGGING_RULES"] = "*.warning=false"
12
13
14 # Add python_scripts to path relative to this file
15 # We need to go up from src -> task1_perception -> perception_and_planning_lab
16 script_dir = os.path.dirname(os.path.abspath(__file__))
17 python_scripts_path = os.path.join(script_dir, '../..../python_scripts')
18 sys.path.append(python_scripts_path)
19
20 import ball_tracker
21
22 colors=["green","blue","red"] #Colors to track in the robot soccer
23
24 # Tabla fija de colores BGR para OpenCV
25 bgr_map = {
26     "green": (0, 255, 0),
27     "blue": (255, 0, 0),
28     "red": (0, 0, 255)
29 }
30
31 trackers = {
32     "Blue": ball_tracker.KalmanTracker(),
33     "Green": ball_tracker.KalmanTracker(),
34     "Red": ball_tracker.KalmanTracker()
35 }
36
37 # Crear draw_info automáticamente
38 draw_info = [(c.capitalize(), bgr_map[c]) for c in colors]
39
40 def main():
41     parser = argparse.ArgumentParser(description='Detector de pelota robusto con Kalman
42     Filter + Modelo Bayesiano.')
43     parser.add_argument('video_path', type=str, help='Path to video file')
44     parser.add_argument('--captures', type=int, default=1,
45                         help='Número de capturas que hará el usuario para entrenar el
46                         color')
47     parser.add_argument('--threshold', type=float, default=7.456,
48                         help='Umbral para la máscara Bayesiana')
49     args = parser.parse_args()
50     kalaman=ball_tracker.KalmanTracker()
51     #kalaman.track_video(args.video_path, args.captures, args.threshold)
52
53     args=parser.parse_args()
54
55     kalaman=ball_tracker.KalmanTracker()
56     trajectory = {name: [] for name in trackers.keys()}
57     means=[[[] for _ in range(len(colors))]]
58     covs=[[[] for _ in range(len(colors))]]

```

```
56     mean_hsv=[[] for _ in range(len(colors))]
57     cov_hsv=[[] for _ in range(len(colors))]
58
59
60     cap=cv2.VideoCapture(args.video_path)
61     if not cap.isOpened():
62         print(f"Error: No se pudo abrir el video {args.video_path} . Revisa el path")
63         return
64     print("Mark the colors...")
65
66     for c in range(len(colors)):
67         print(f"Color {c+1}: {colors[c]}")
68
69         for i in range(args.captures):
70             print(f"Captura {i+1} de {args.captures}")
71             frame_number=int(input(f"Frame number of
{int(cap.get(cv2.CAP_PROP_FRAME_COUNT))}: "))
72             try:
73                 cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
74                 ret, frame=cap.read()
75             except:
76                 print("Error: Not valid frame number")
77                 continue
78
79             if not ret:
80                 print("Error: No se pudo leer el frame")
81                 continue
82             men_i,cov_i=ball_tracker.select_color_model(frame)
83             means[c].append(men_i)
84             covs[c].append(cov_i)
85
86             print("\n")
87             print("\n")
88
89             mean_hsv[c] = np.mean(means[c], axis=0)
90             cov_hsv[c] = np.mean(covs[c], axis=0)
91             sleep(1)
92             print("Media color: ",colors[c],mean_hsv[c])
93             print("Cov color: ",colors[c],cov_hsv[c])
94             sleep(3)
95
96             print("Tracking...")
97             #cap.set(0, frame_number)
98
99             cap.set(cv2.CAP_PROP_POS_FRAMES, 2)
100            hsv=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
101            combined_mask = np.zeros(hsv.shape[:2], dtype=np.uint8)
102            masks=[[] for _ in range(len(colors))]
103            kernel=np.ones((5,5),np.uint8)
104
105            while True:
106                ret, frame = cap.read()
107                if not ret:
108                    break
109                hsv=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
110                combined_mask = np.zeros(hsv.shape[:2], dtype=np.uint8)
111                for c in range(len(colors)):
112                    masks[c] = ball_tracker.gaussian_mask(hsv, mean_hsv[c], cov_hsv[c],
```

```
args.threshold)

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

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

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

120     blue_contour,_=cv2.findContours(masks[1],cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
121
122     green_contours,_=cv2.findContours(masks[0],cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
123     red_contour,_=cv2.findContours(masks[2],cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

124
125     # Diccionarios para guardar los resultados por color
126     detected = { name: False for name, _ in draw_info }
127     measured_x = { name: None for name, _ in draw_info }
128     measured_y = { name: None for name, _ in draw_info }

129     for i, (name, color) in enumerate(draw_info):
130         contours, _ = cv2.findContours(masks[i], cv2.RETR_EXTERNAL,
131                                         cv2.CHAIN_APPROX_SIMPLE)
132         if contours:
133             if name != "Green":
134                 c = max(contours, key=cv2.contourArea) # max
135                 ((x, y), radius) = cv2.minEnclosingCircle(c)
136                 print(radius)
137                 if radius > 9.0 and radius < 15.5: # 0.005
138
139                     detected[name] = True
140                     measured_x[name] = int(x)
141                     measured_y[name] = int(y)
142
143                     # Dibujo en pantalla
144                     cv2.circle(frame, (measured_x[name], measured_y[name]),
145                               int(radius), color, 2)
146                     cv2.putText(frame, name, (measured_x[name] + 10,
147                                            measured_y[name]), cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, 1)
148             else:
149                 for c in contours:
150                     ((x, y), radius) = cv2.minEnclosingCircle(c)
151
152                     if radius > 8.0 and radius < 15.2:
153                         detected[name] = True
154                         if measured_x[name] is None:
155                             measured_x[name] = []
156                             measured_y[name] = []
157                             measured_x[name].append(int(x))
158                             measured_y[name].append(int(y))
159
160                             cv2.circle(frame, (int(x), int(y)), int(radius),color, 2)
161                             cv2.putText(frame, name, (int(x) + 10, int(y)),
162                                         cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, 1)
```

```

165 #cv2.imshow("frame",frame)
166 print("pos: ",measured_x," | ",measured_y,"\n")
167
168 #pred_x, pred_y = kalaman.predict()
169
170
171 kalman_positions = {}
172
173 for name, tracker in trackers.items():
174     if name in ["Red", "Blue"]:
175         pred_x, pred_y = tracker.predict()
176
177         if detected[name]:
178             mx = measured_x[name]
179             my = measured_y[name]
180             tracker.correct(mx, my)
181             current_pos = (mx, my)
182         else:
183             current_pos = (pred_x, pred_y)
184             cv2.putText(frame, f"{name} P(Ocuido)", (int(pred_x) + 10,
185 int(pred_y)),
186                         cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
187
188         kalman_positions[name] = current_pos
189         # Dibujo del marcador del Kalman
190         cv2.drawMarker(frame, (int(current_pos[0])), int(current_pos[1])), (0, 0, 255),
191         ), cv2.MARKER_CROSS, 20, 2)
192
193         elif name == "Green":
194             if detected[name]:
195                 mx_list = measured_x[name]
196                 my_list = measured_y[name]
197                 if isinstance(mx_list, list):
198                     for x, y in zip(mx_list, my_list):
199                         cv2.drawMarker(frame, (x, y), (0, 0, 255),
200                         cv2.MARKER_TILTED_CROSS, 15, 2)
201
202             # Draw Arrow from Red to Blue
203             if "Red" in kalman_positions and "Blue" in kalman_positions:
204                 start_point = kalman_positions["Blue"]
205                 end_point = kalman_positions["Red"]
206
207                 # Extender la flecha
208                 scale = 3.0
209                 dx = end_point[0] - start_point[0]
210                 dy = end_point[1] - start_point[1]
211                 new_end_point = (int(start_point[0] + dx * scale), int(start_point[1] + dy *
212 scale))
213
214                 cv2.arrowedLine(frame, start_point, new_end_point, (75, 70, 50), 6)
215
216             # Update and Draw Trajectory for Blue only
217             if "Blue" in kalman_positions:
218                 trajectory["Blue"].append(kalman_positions["Blue"])
219                 if len(trajectory["Blue"]) > 45: #15
220                     trajectory["Blue"].pop(0)
221
222                 for i in range(1, len(trajectory["Blue"])):

```

```
219         if trajectory["Blue"][i - 1] is None or trajectory["Blue"][i] is None:
220             continue
221         cv2.line(frame, trajectory["Blue"][i - 1], trajectory["Blue"][i], (200, 100
222             , 120), 3)
223
224     cv2.imshow("frame", frame)
225
226     if cv2.waitKey(30) & 0xFF == ord('q'):
227         break
228
229     cap.release()
230     cv2.destroyAllWindows()
231
232
233 if __name__ == '__main__':
234     main()
235
```