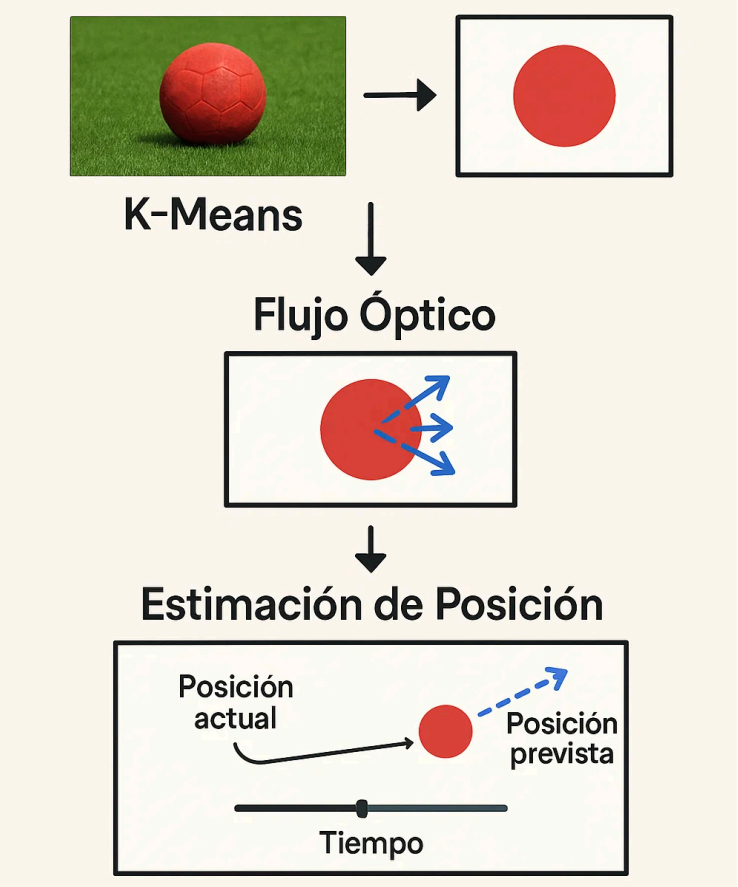


Seguimiento y predicción de la trayectoria de una pelota usando Flujo Óptico y K-Means

Diego García Currás
Alberto León Luengo

ÍNDICE

[] 1. INTRODUCCIÓN

} 1.1 OBJETIVOS

El objetivo de esta práctica consiste en desarrollar un sistema de Visión Artificial capaz de realizar las siguientes tareas:

- Segmentar una pelota en movimiento utilizando el algoritmo K-means.
- Rastrear su movimiento mediante el Flujo Óptico utilizando el método de Lucas-Kanade.
- Predecir su posición futura en un tiempo configurable mediante un trackbar.

} 1.2 CONCEPTOS CLAVE

Flujo Óptico: Técnica para estimar el movimiento de objetos entre frames consecutivos.
K-Means: Algoritmo de clustering utilizado para separar los píxeles de la pelota del fondo.
Segmentación: Proceso de dividir una imagen en regiones de interés (en este caso, la pelota).

} 1.3 APLICACIONES

La aplicación diseñada en esta práctica se puede utilizar en los siguientes campos:

- **Robótica**: Seguimiento predictivo de objetos en movimiento (Mejora de navegación, eficiencia…)
- **Deportes**: Análisis de la trayectoria en deportes de pelota (fútbol, tenis, etc).
- **Vigilancia**: Detección de movimientos anómalos.

[] 2. METODOLOGÍA

} 2.1 SEGMENTACIÓN CON K-MEANS

Para aislar la pelota del fondo, es necesario aplicar K-Means dentro del espacio de color HSV, ya que es más robusto a cambios de iluminación que el espacio de color RGB.

El proceso de segmentación con K-Means se puede dividir en 3 fases:

FASE 1: PREPROCESAMIENTO

En primer lugar, se pasa la imagen al espacio de color HSV y se aplica un desenfoque gaussiano con el objetivo de reducir todo el ruido posible.

FASE 2: CLUSTERING

Una vez hecho esto, se agrupan todos los píxeles de la imagen en dos o más clústers: Al menos uno para los píxeles de la pelota y otros para el resto. Después, mediante un sistema de ponderación se selecciona aquel cluster que contenga una tonalidad media más parecida al color HSV que estamos buscando, ya que en este caso, se asume que la pelota es de un color vistoso.

En el caso de que ningún clúster cumpla con un mínimo de confiabilidad (Una puntuación suficientemente alta) se concluye que no se ha detectado la pelota.

FASE 3: POSTPROCESAMIENTO

Y por último, se aplican las operaciones de dilatación y de erosión correspondientes hasta limpiar la máscara.

} 2.2 FLUJO ÓPTICO (LUCAS KANADE)

Habiendo segmentado ya la imagen con K-Means, se aplica Flujo Óptico, concretamente el método de Lucas Kanade, para estimar el movimiento de los píxeles de la pelota entre frame. Para ello, se siguen una serie de pasos explicados brevemente a continuación:

- **Detección de características**: Se determinan puntos de interés dentro del contorno de la pelota (Centro y puntos de los bordes)
- **Cálculo del Flujo Óptico**: Utiliza la función calcOpticalFlowPyrLK() para estimar el desplazamiento de los puntos.
- **Filtrado**: Realmente no es necesario, ya que solo se están siguiendo puntos que pertenecen a la pelota..

} 2.3 PREDICCIÓN DE TRAYECTORIA

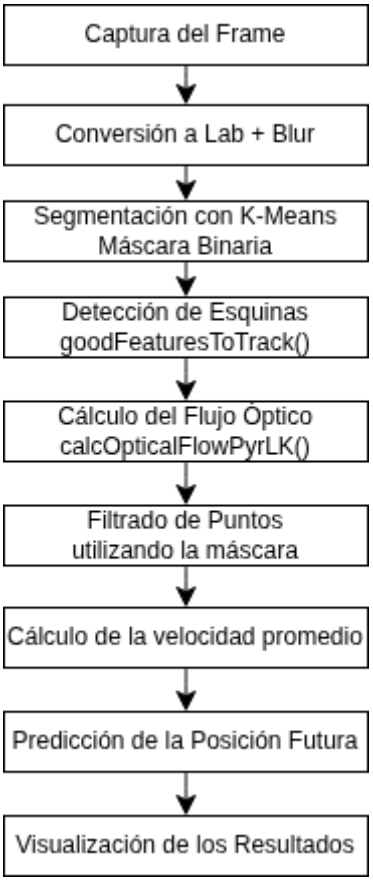
Para predecir la trayectoria que seguirá la pelota y su posición estimada en un tiempo X, se calcula el desplazamiento promedio de los píxeles de la pelota y se extrapola su posición futura teniendo en cuenta la siguiente fórmula:

Posición Futura = Posición Actual + Desplazamiento Promedio x Frames por delante

Es importante mencionar que el tiempo de predicción se ajusta con un trackbar / slider implementado.

[] 3. IMPLEMENTACIÓN

} 3.1 DIAGRAMA DE FLUJO



} 3.2 FRAGMENTOS DE CÓDIGO CLAVE

SEGMENTACIÓN CON K-MEANS

```
cv::Mat kmeansSegmentation(const cv::Mat & frame)
{
 int cluster_count = cv::getTrackbarPos(CVParams::CLUSTERS,
     CVParams::WINDOW_NAME) + 1;

 cv::Mat hsv;
 cv::cvtColor(frame, hsv, cv::COLOR_BGR2HSV);

 cv::Mat samples = hsv.reshape(1, hsv.rows * hsv.cols);
 samples.convertTo(samples, CV_32F);

 cv::Mat labels, centers;
 cv::kmeans(samples, cluster_count, labels,
            cv::TermCriteria(cv::TermCriteria::EPS +
```

```
    cv::TermCriteria::MAX_ITER, 10, 1.0),
            3, cv::KMEANS_PP_CENTERS, centers);

const float hue_center = (CVParams::lower_hsv_bound[0] +
  CVParams::upper_hsv_bound[0]) / 2.0f;
const float hue_range = (CVParams::upper_hsv_bound[0] -
  CVParams::lower_hsv_bound[0]) / 2.0f;
const float score_threshold = 0.6f;

int target_cluster_idx = -1;
float best_score = -1.0f;

for (int i = 0; i < centers.rows; ++i) {
  float hue = centers.at<float>(i, 0);
  float sat = centers.at<float>(i, 1);
    // float val = centers.at<float>(i, 2); // Not used in scoring

  float hue_diff = std::abs(hue - hue_center);
  float hue_score = 1.0f - (hue_diff / hue_range);
  hue_score = std::max(0.0f, std::min(hue_score, 1.0f));

  float sat_score = sat / 255.0f;
  float final_score = (hue_score * 0.8f) + (sat_score * 0.2f);

  if (final_score > best_score) {
    best_score = final_score;
    target_cluster_idx = i;
  }
}

if (best_score < score_threshold) {
  std::cerr << "[INFO] No confident cluster found (best_score = " <<
    best_score << "). No ball detected." << std::endl;
  return cv::Mat::zeros(frame.size(), CV_8UC1);
}

std::cout << "[INFO] Ball detected in cluster " << target_cluster_idx <<
  " (score = " << best_score << ")" << std::endl;

cv::Mat mask(hsv.rows, hsv.cols, CV_8UC1, cv::Scalar(0));

for (int i = 0; i < labels.rows; ++i) {
```

```
    int label = labels.at<int>(i);
    if (label == target_cluster_idx) {
      mask.at<uchar>(i / hsv.cols, i % hsv.cols) = 255;
    }
  }

  return mask;
}
```

FLUJO ÓPTICO Y PREDICCIÓN

```
cv::Mat processOpticalFlow(
  const cv::Mat & frame,
  const CVParams::OPTICAL_FLOW_MODE mode)
{
  std::vector<cv::Point2f> next_points, good_new_points;
  cv::Mat processed_frame = frame.clone();
  cv::Mat mask = CVUtils::inRangeSegmentation(frame);
  std::vector<cv::Vec3f> circles = CVUtils::detectCircles(mask);
  int radius = 0;
  cv::Point2f center(0.f, 0.f);
  if (!circles.empty()) {
    center = cv::Point2f(circles[0][0], circles[0][1]);
    radius = static_cast<int>(circles[0][2]);
  }
  cv::Mat masked_frame;
  cv::bitwise_and(frame, frame, masked_frame, mask);
  cv::Mat current_gray = CVUtils::preprocessFrame(masked_frame);

  if (!CVParams::tracking_initialized) {
    CVParams::previous_gray_frame = current_gray.clone();
    CVParams::previous_points = CVUtils::getBallPoints(frame);
    CVParams::tracking_initialized = true;
    return processed_frame;
  }

  if (CVParams::previous_points.empty()) {
    std::cerr <<
      "[ERROR] No points detected for optical flow initialization." <<
      std::endl;
    CVParams::tracking_initialized = false;
    return processed_frame;
```

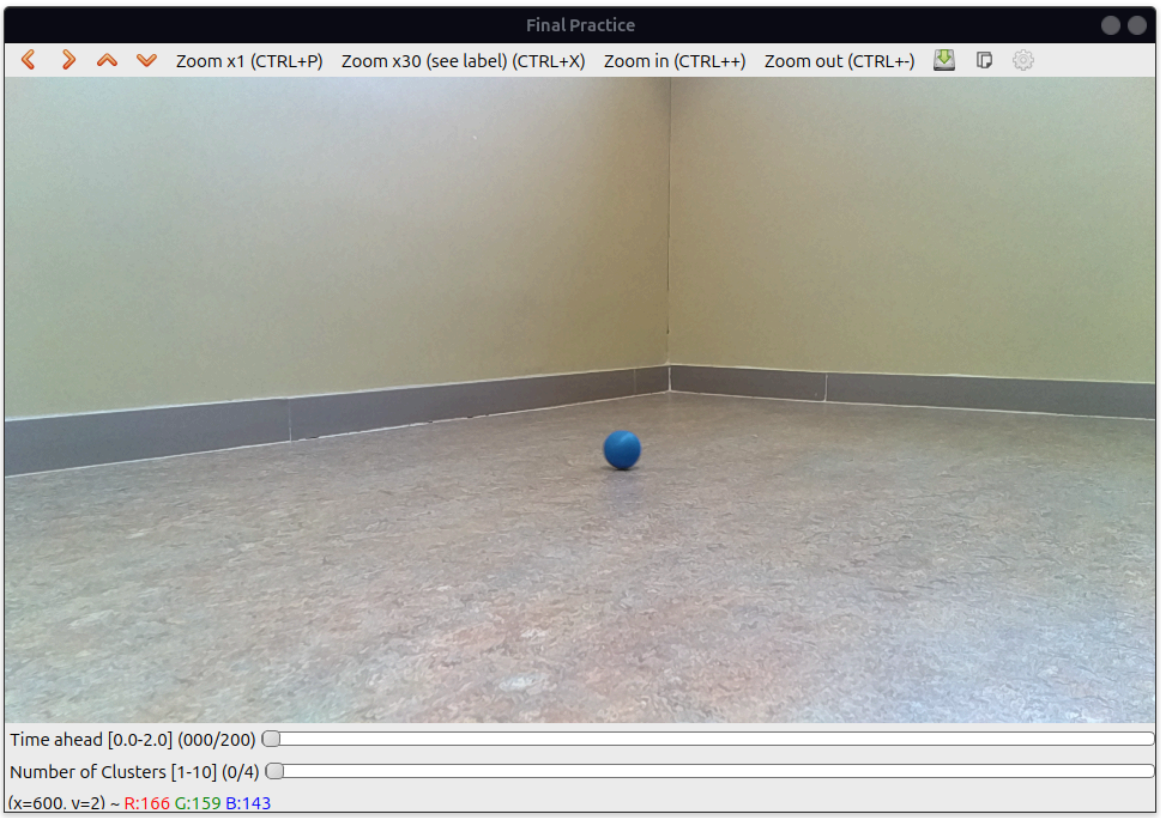
```
 }

 // Optical Flow
 std::vector<uchar> status;
 std::vector<float> err;
 cv::Size win_size(41, 41);
 int max_level = 5;
 cv::TermCriteria criteria(cv::TermCriteria::COUNT +
cv::TermCriteria::EPS, 50,
   0.03);

 cv::calcOpticalFlowPyrLK(
     CVParams::previous_gray_frame, current_gray,
     CVParams::previous_points, next_points,
     status, err,
     win_size, max_level, criteria
 );

 cv::Point2f total_displacement(0.f, 0.f);
 cv::Point2f center_current(0.f, 0.f);
 int valid_points = 0;

 for (size_t i = 0; i < CVParams::previous_points.size(); ++i) {
   if (status[i]) {
     good_new_points.push_back(next_points[i]);
     cv::circle(processed_frame, next_points[i], 5, cv::Scalar(0, 255, 0),
-1);
     cv::line(processed_frame, CVParams::previous_points[i],
next_points[i],
         cv::Scalar(0, 255, 0), 2);

     total_displacement += (next_points[i] -
CVParams::previous_points[i]);
     center_current += next_points[i];
     valid_points++;
   }
 }

 if (valid_points > 0) {
   total_displacement *= (1.0f / valid_points);
   center_current *= (1.0f / valid_points);
```

```
    CVParams::displacement_history.push_back(total_displacement);
    if (CVParams::displacement_history.size() > CVParams::MAX_HISTORY) {
      CVParams::displacement_history.pop_front();
    }

    cv::Point2f mean_displacement(0.f, 0.f);
    for (const auto & d : CVParams::displacement_history) {
      mean_displacement += d;
    }
    mean_displacement *= (1.0f / CVParams::displacement_history.size());

    cv::arrowedLine(
        processed_frame,
        center_current,
        center_current + mean_displacement * 5.0f,
        cv::Scalar(0, 0, 255), 3, cv::LINE_AA
    );

    int future_frames = cv::getTrackbarPos(CVParams::FRAMES_AHEAD,
        CVParams::WINDOW_NAME);
    cv::Point2f predicted_position = center_current + mean_displacement *
      future_frames;

    cv::circle(processed_frame, predicted_position, radius,
        cv::Scalar(255, 0, 0), 3);
  } else {
    std::cerr << "[WARNING] No valid points tracked for movement vector."
<<
      std::endl;
  }

  CVParams::previous_points = good_new_points;
  CVParams::previous_gray_frame = current_gray.clone();

  return processed_frame;
}
```

[] 4. RESULTADOS Y DISCUSIONES

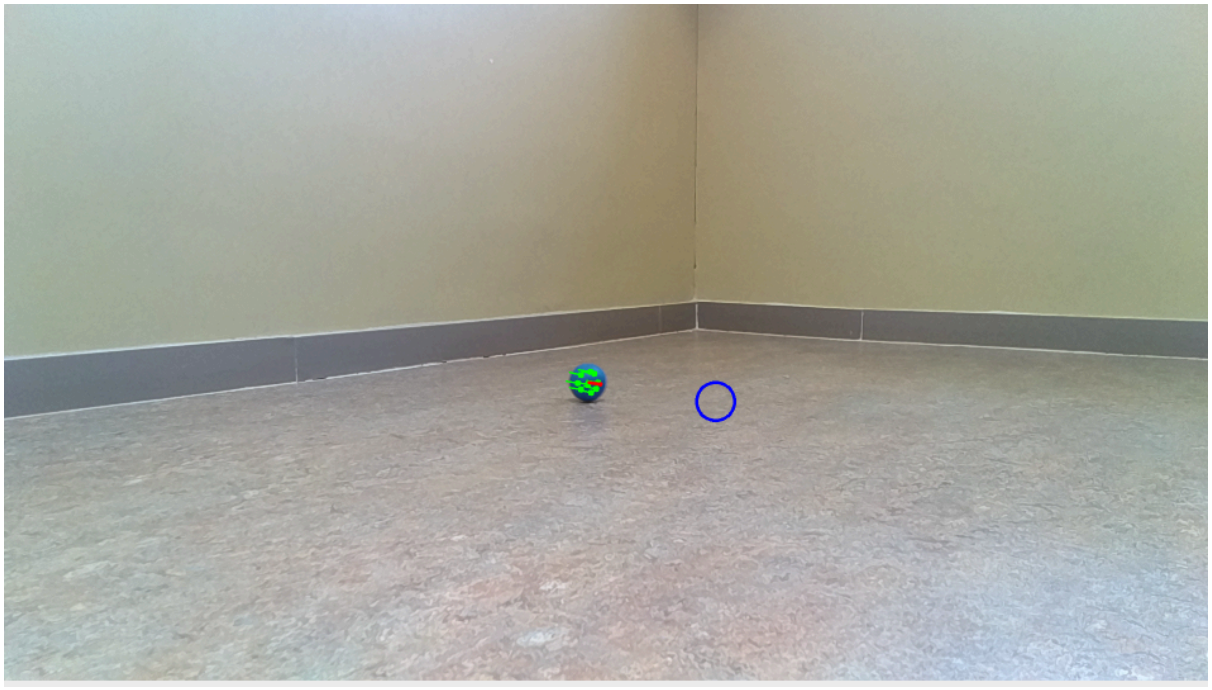
} 4.1 CAPTURAS DE PANTALLA

FRAME ORIGINAL



MÁSCARA K-MEANS

FLUJO ÓPTICO + PREDICCIÓN



} 4.2 MÉTRICAS DE RENDIMIENTO

TIEMPO DE PREDICCIÓN (S / MS)	ERROR OBTENIDO (PÍXELES)
10	e < 10%
15	10% < e < 15%
20	e > 15%

Como se puede observar, la predicción es más precisa cuando se utilizan tiempos cortos. Sin embargo, el error puede aumentar con el tiempo debido a cualquier tipo de aceleración anómala que aparezca.

} 4.3 LIMITACIONES

Sin embargo, esta aplicación posee limitaciones, las cuales se detallan a continuación:

- Dependencia del color de la pelota (K-Means puede fallar si hay partes del fondo con un color similar al de la pelota).
- K-Means es muy costoso para una aplicación de tiempo real de estas características. Se debería valorar otro tipo de algoritmo de segmentación. En esta situación tan controlada, un simple filtro de color y forma funciona mucho mejor.
- La aparición de oclusiones pueden causar pérdidas a la hora de realizar el tracking de la pelota, por la similitud de texturas con algunas partes del fondo.
- La realización de algún movimiento no lineal puede afectar a la precisión obtenida a la hora de calcular la posición estimada de la pelota.
- Los cambios bruscos de dirección afectan a la predicción durante unos frames.

[] 5. CONCLUSIONES

En resumen, se ha demostrado la viabilidad desarrollar un sistema capaz de seguir una pelota y de predecir su posición estimada en un intervalo de tiempo X, mediante técnicas de segmentación y el flujo óptico, que en este caso permite estimar la velocidad a la que se mueve la pelota, aunque posee algunas limitaciones en caso de que la pelota realice movimientos más complejos.

Como mejoras a implementar de cara al futuro, se podrían utilizar redes neuronales para mejorar la segmentación e implementar un filtro de Kalman para obtener predicciones más robustas. También se podría aumentar la tasa de frames de captura para suavizar los movimientos rápidos. También se podrían medir otras magnitudes del sistema internacional calibrando adecuadamente la cámara y añadiendo medidas reales del objeto a seguir.