UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Visión de robots humanoides para detección de pelota, portería y postes en cancha de futbol de RoboCup.

PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA (COMPUTACIÓN)

PRESENTA:

ABNER QUIROZ CLEMENTE

TUTOR: **DR. FERNANDO ARÁMBULA COSIO**
CCADET, UNAM

MÉXICO, D.F. JULIO 2013.

Tabla de Contenidos

Resumen

RoboCup es una iniciativa científica internacional cuya meta es hacer progresos en el estado del arte de la robótica de robots inteligentes.

Dentro de este evento actualmente existe la categoría de RoboCup Soccer, la cual consiste en que cada equipo debe crear un equipo de robots completamente autónomos que muestren comportamiento competitivo y estrategias avanzadas.

La UNAM ya ha participado en dicho evento en años anteriores, pero en 2011 se decidió no asistir para poder preparar un equipo competitivo para el torneo del 2012, cuya sede fue México. El trabajo realizado desde enero de 2011 ha consistido en conocer la estructura del equipo, la construcción de los robots, sus componentes, los programas que hasta esta fecha se tienen para la inteligencia y visión y también en envolverse en el ambiente de la RoboCup y los equipos internacionales y mexicanos.

A partir de esa fecha se empezó la creación de software nuevo para los robots que se construyeron y competieron en 2012. Estos cuentan con nuevos componentes electrónicos que nos dan más capacidad de procesamiento, lo cual brinda la posibilidad de desarrollar código específico con mejores algoritmos para encontrar los objetos en la cancha de juego a través de la visión del robot.

En este trabajo se presenta la teoría detrás de cada componente del programa, después se muestra la implantación de cada uno y porque se dejaron o desecharon en el producto final.

El programa consiste en entender cómo se capturan las imágenes a través de *cámaras digitales* (webcams) integradas en los robots. Una vez capturadas se debe elegir el *espacio de color* más adecuado para trabajar con ellas, pues parte del reconocimiento que se hace es por *segmentación de colores*. Para dicha segmentación se probaron distintas técnicas (*umbrales, distancias, clasificadores de k medias, bayesianos*) hasta llegar a una propia que es combinación de algunas anteriores, a la que se llamó *K-Cubos*.

A partir de las imágenes segmentadas tenemos que obtener la posición de los objetos en la cancha como son *porterías, pelota y postes laterales*, si es que están en la imagen. En este punto se aplican filtros morfológicos para eliminar ruido en la segmentación y obtener una mejor clasificación de objetos. Para esto cada objeto se trata de distinta manera. La pelota sabemos que es naranja y además es circular, por lo que en este trabajo se propone usar *circularidad y cierre convexo* para clasificarla y se ve una solución a la oclusión que sufre en la parte inferior de su cuerpo debido a la sombra auto provocada.

Las porterías son los objetos más grandes en la cancha y son de color azul o amarillo. Basado en un poco de trabajos anteriores se decidió buscarlas a través de su forma, por lo que un descriptor de forma sería el *centro de gravedad*, el cual es distinto para cada figura irregular. Usando esto, verificamos la posición de dicho centroide respecto a los puntos de toda la figura para saber si estamos o no en presencia de la portería. Se añadieron casos para cuando la portería está ocluida parcialmente y poder detectarla aun así.

Los postes laterales son usados para localización y posicionamiento de los robots por lo que es importante poder distinguirlos. Estos están conformados de dos colores, azul y amarillo en una configuración de dos partes de uno por una parte del otro, siempre el singular en medio de los otros dos. Los métodos que se usan para encontrarlos son basados en buscar *sus puntos extremos y su centro*, y otro en buscar *blobs* de estos colores y conforme a su posición respecto uno del otro tratar de checar si el objeto es o no un poste.

Aunque varias técnicas se proponen, no todas son factibles de implementar en los robots, pues estos poseen capacidades de almacenamiento y procesamiento *limitadas*. Por esto aunque sean métodos confiables, se tienen que desechar algunos o adaptar para poder ser puestos en marcha sobre la competencia, ya que recordemos que son sistemas que trabajan en *tiempo real*.

Al final del documento se exponen una serie de códigos base en lenguaje C++ que se implementaron en los robots, un poco de Matlab con lo que se hicieron pruebas, imágenes muestras de laboratorio y competencia, así como resultados obtenidos al clasificar pixeles y objetos.

Introducción.

La RoboCup.

"Es una iniciativa científica internacional establecida en 1997, con la finalidad de lograr en comunidad avances en el área de la robótica inteligente y su mayor atractivo al público en general fue el objetivo de tener listo para el año 2050 d.C. un equipo de robots jugadores de futbol (soccer) que pudiera vencer al equipo de humanos campeón de la copa del mundo" [1].

Existen varias áreas actualmente en las que RoboCup llama a participar, entre ellas están Soccer, Rescue, @Home y Junior. Esto promueve entre la comunidad internacional el desarrollo de nuevas tecnologías y resultados de investigaciones que desde el laboratorio se puedan plasmar en el campo. La categoría de soccer consiste en crear equipos de robots autónomos que cooperen entre sí exhibiendo comportamientos que pueden pasar por inteligentes para ejecutar una estrategia. Todo esto debe realizarse en un ambiente adverso dinámico, en el cual muchas variables del medio entran en juego como iluminación, público, condiciones de cancha, etcétera. En la categoría de humanoides estos robots deben tener una semejanza con la estructura humana, por ello las reglas de la competencia establecen cada vez que estos tiendan a parecerse más al poner mayor número de limitaciones en medidas y formas del humanoide. Además los sensores que están permitidos son aquellos que imiten el que poseemos los humanos, como cámaras en lugar de ojos, giroscopios para el equilibrio, oído con micrófono, bocinas, etc.

Esta liga conlleva a muchos ámbitos de investigación como caminado dinámico, correr, patear la pelota, ver los objetos en la cancha como pelota, porterías y rivales, auto localización y juego de equipo entre muchos otros.

Existen 3 tipos de robots en la liga de humanoides. Están clasificados de acuerdo a su tamaño en "Kid Size", "Teen Size" y "Adult Size". Aunque el trabajo presente se puede aplicar para cualquiera de las tres categorías, las pruebas y el trabajo está dedicada a Kid Size que son robots de 30 a 60 cm de altura por ser la competencia en la que se participa con el equipo pUNAMoids y dotMX en RoboCup.

El problema de la visión.

La visión en los robots humanoides que juegan soccer en la RoboCup está basado en el sistema de percepción que tenemos los humanos, es decir los ojos. Para imitar estos componentes en los jugadores se ponen una o dos cámaras que reciban una secuencia de imágenes (video) a través de las cuales el robot podrá identificar elementos en su entorno y tomar decisiones a partir de esta información, como planear una estrategia y realizar cierto movimiento (correr, girar o patear) entre otros.

Por eso es de tanta importancia tener un buen sistema de visión, ya que es por medio de este que el robot podrá entender que objetos están cerca y lejos de él. En un juego de futbol es necesario saber en donde está la pelota y dependiendo de la posición del

jugador y su función como delantero, defensa o portero, este puede usar su conocimiento para decidir si ir por la pelota, patearla, posicionarse en otra área de la cancha, sólo cubrir a un rival o la pelota.

Además existen varios objetos que debe de poder encontrar un robot. Entre estos y primero que todos está la pelota. Luego existen más como las porterías, líneas de cancha, robots compañeros y rivales. En la liga de humanoides actualmente están usándose dos postes a media cancha con una combinación de colores los cuales llamaremos "beacons" que sirven como ayuda para localizarse a los robots. En la cancha de RoboCup para KidSize un sistema de colores ha sido impuesto para ayudar a los equipos a identificar los objetos. Este posee azul, amarillo, naranja, verde, magenta, cian y blanco. Son colores muy distintos entre sí y por ahora es la información principal en que los robots se apoyan para identificar los objetos.

Las porterías están pintadas una de azul y otra de amarillo, los beacons poseen una combinación de dos colores en dos patrones distintos basado en azul y amarillo también. La pelota es una bola de tenis naranja. La cancha es verde y las líneas del campo son blancas.

Existen varios factores que alteran la percepción del robot aún con los colores ya descritos y sabiendo de ante mano cuales son. La iluminación de la cancha, la distancia de los objetos, obstrucción de la vista por rivales o compañeros, presencia de público, cambio de posición entre otros, son algunos motivos para complicar el problema de la visión en los robots.

En el presente trabajo se plantea usar información de la geometría de los objetos y no sólo el color de estos para distinguirlo para de esta forma poder proporcionar mayor robustez a los algoritmos detectores y por supuesto construir robots más confiables.

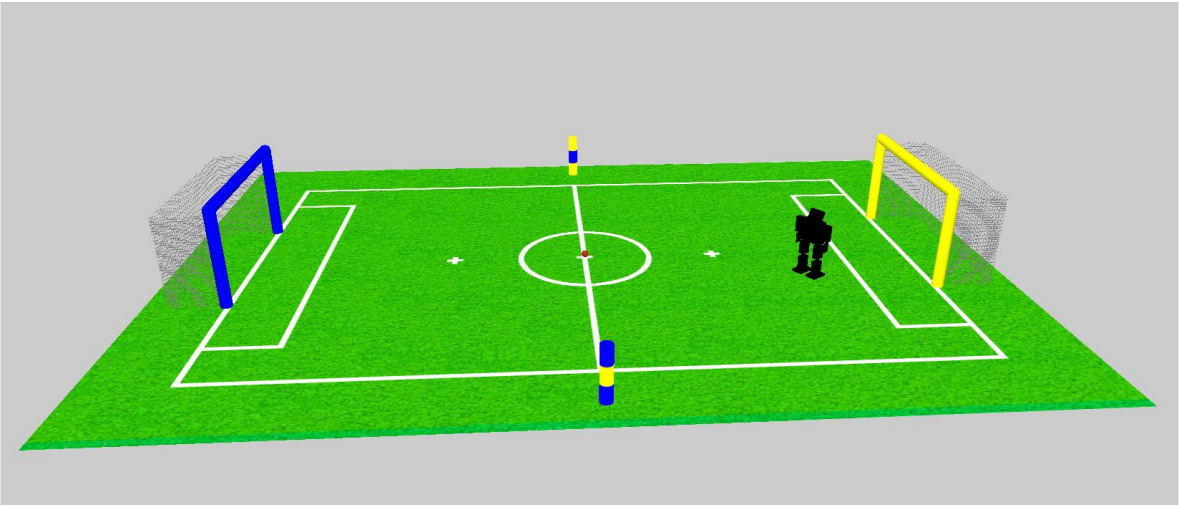


Figura 1. Descripción de la cancha de RoboCup 2012.[1]

[1] Imagen obtenida de [2]

En la Figura 1 se pueden apreciar gráficamente los elementos de la cancha de soccer del torneo de RoboCup según las reglas para el torneo 2012 llevado a cabo en la Ciudad de México [2].

Descripción del robot.

El robot en el que se prueban los algoritmos y que es el que juega en competencia es el DarwinOP, una plataforma abierta de hardware que tiene una computadora FitPC y una tarjeta controladora de motores.

El robot posee una cámara web Logitech como sistema de captura de video en su cabeza.

Objetos a reconocer.

En resumen, el conjunto de objetos básicos que el sistema de visión debe de reconocer son la pelota, las porterías y los postes laterales.

De acuerdo a las reglas de la RoboCup, la pelota debe ser naranja, las porterías son una azul y otra amarilla, y los postes laterales son uno dos partes de azul con una de amarillo en medio y otra de dos partes de amarillo con azul en el centro.

Descripción del contenido.

A continuación se presenta un breve resumen del contenido de los temas de la tesis.

Capítulo I. Obtención de las imágenes.

En este apartado se comenta sobre la forma en que se adquiere la información de la vista del robot a través de cámaras digitales. Se explica que problemas conlleva el capturar imágenes a través de estos dispositivos surgidos por la manera en que están construidas las cámaras y la forma en que se resuelve en código a través de la librería OpenCV.

Se explica además como la información del video puede ser manipulada, ya sea en espacio de color BGR o en HSV. Se explican estos dos espacios y se empieza a hablar de las ventajas y desventajas de usar cada uno.

Capítulo II. Segmentación.

Después de adquiridas las imágenes se debe separar la información útil del resto. En este capítulo se explican varios procedimientos para escoger los pixeles que se consideran como información útil, empezando por segmentación a través de distancias, que es el método más sencillo. Después se explican métodos más complicados, como *K-medias* y el uso de un clasificador de *Bayes*, que también son útiles para este procedimiento.

Luego se explica un método propuesto en este trabajo basado en segmentación por umbrales, llamado K-*Cubos*, el cual toma un poco la idea de K-medias al crear varias clases, pero en este caso, se usa para un mismo objeto.

Capítulo III. Reconocimiento de objetos.

En este apartado se usan los pixeles que son útiles obtenidos de los algoritmos anteriores para aplicar sobre ellos métodos de reconocimiento de objetos.

Se plantean tres objetos clave en el trabajo a identificar: la pelota, las porterías y los postes laterales. Cada uno de estos objetos posee características únicas que se usan para detectarlos.

Para la pelota se plantean métodos descriptores de forma, que son por centroide, cierre convexo y defectos del cierre convexo.

Para las porterías se usan los métodos de detección de blobs, ya que estas son los objetos más grandes dentro de la cancha de juego por lo tanto producen mayor detección de pixeles en la imagen de la cámara. También se usa el concepto de centro de masa debido a la forma única de las porterías y se plantea una solución para la oclusión parcial de estos objetos en la imagen capturada.

Para los postes laterales se presentan dos métodos. El primero es a través de la detección de puntos característicos de estos objetos y el segundo es por medio la detección de los centros de los blobs que generan las partes distintas de los postes y la discriminación de dichos centros a través de su posición vertical y horizontal en la imagen.

Capítulo IV. Experimentos y resultados.

Aquí se enlistan las herramientas con las que se contó para realizar el trabajo, tanto de hardware y software, las limitaciones del robot en cada uno de los algoritmos y el espacio de muestras usado para realizar pruebas de los métodos.

En este capítulo se explica la elección de trabajar en un espacio de color determinado por medio de pruebas en cambios de iluminación sobre imágenes semejantes y el cambio de los datos en ellas.

Después se ponen a prueba los diferentes métodos de segmentación de color. Se ven las ventajas de trabajar con cada uno de ellos, los tiempos que tardan en trabajar y el número de pixeles clasificados de forma correcta e incorrecta respecto a la imagen original.

El siguiente paso es elegir los métodos de detección de objetos. Esto se hace a través de su eficiencia medida en base al número de detecciones correctas en la imagen y a la robustez basada en detecciones hechas con ruido en la imagen, es decir, con oclusiones o cambios de posición de los objetos.

En cada uno de estos se explica además el uso de otros métodos de tratamiento de imágenes digitales y reconocimiento de patrones necesarios para procesar la visión como eliminar ruido, aplicar filtros morfológicos y detección de bordes, y que esto no comprometa la rapidez de los algoritmos al ejecutarse en el robot. Las pruebas y resultados se muestran en los anexos.

Al final de esta sección se da un breve comentario de la implementación sobre el robot de los algoritmos.

Capítulo V. Conclusiones.

En este apartado se comenta sobre el trabajo realizado, las ventajas del uso de los algoritmos seleccionados y que resultados se obtuvieron al ponerlos a prueba en competencia y en laboratorio.

Por último se plantean metas a futuro y los cambios en las reglas que se deben de tomar en consideración para la competencia de RoboCup.

Capítulo 1. Obtención de imágenes.

1.1 Filtro Bayer en cámaras digitales.

Las cámaras digitales aunque poseen mayor resolución siguen usando el mismo principio en general. En estos sistemas es necesario capturar los valores o densidades de cada pixel rojo, verde y azul por separado. Las cámaras usan generalmente un filtro de Bayer o mosaico, que consiste en un arreglo cuadrado de foto sensores. Este filtro tiene una proporción de 50% sensores verdes, 25% de azules y 25% de rojos, basados en la vista humana la cual tiene una mayor sensibilidad a los tonos verdes que a los azules o rojos. Esto fue propuesto en la patente descrita en [3].

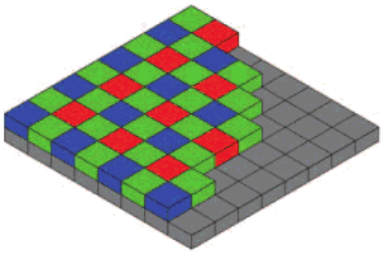


Figura 2. Arreglo Bayer.[2]

Ahora tenemos una imagen en mosaico, en donde cada pixel tiene una medida espectral, lo cual significa que los demás colores tienen que ser estimados de acuerdo a sus vecinos para producir una imagen de alta resolución. A este proceso se le conoce como deshacer el mosaico o sintetizar el color. En la Figura 2. Arreglo Bayer. de arriba podemos ver como están organizadas las celdas, donde la predominancia del verde se mantiene.

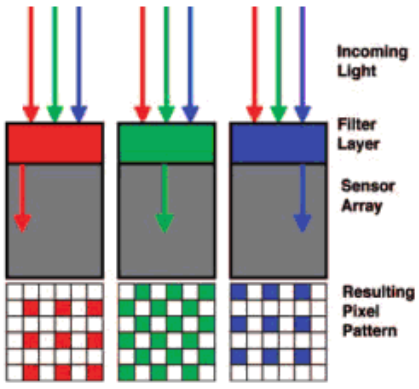


Figura 3. Arreglo resultado del filtro Bayer.[3]

[2]

[3] Figuras de https://en.wikipedia.org/wiki/Bayer_filter

En la Figura 3. Arreglo resultado del filtro Bayer. vemos como los colores son percibidos por una celda con un filtro de cada color lo que provoca solo recibir la luz de ese tipo y son colocados en el arreglo o rejilla. Existen varios métodos para obtener una imagen a partir de este arreglo. "Esto se hace a través de algoritmo de descomposición de mosaico o "Bayer filter demosaicing algorithms" que convierten la imagen RAW (cruda) en valores RGB (rojo, verde y azul). En particular la interpolación bilinear da buenos resultados y tiene la siguiente estructura" [3] [4].

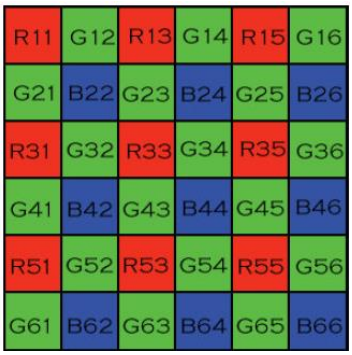


Figura 4. Ejemplo Patrón de Bayer

$$pixel\ R33\ (red) = \begin{cases} R_{ed} = R_{33} \\ G_{reen} = \dfrac{G_{23} + G_{34} + G_{32} + G_{43}}{4} \\ B_{lue} = \dfrac{B_{22} + B_{24} + B_{42} + B_{44}}{4} \end{cases}$$

Ecuación 1

$$pixel\ B44\ (blue) = \begin{cases} B_{lue} = B_{44} \\ G_{reen} = \dfrac{R_{33} + R_{35} + R_{53} + R_{55}}{4} \\ R_{ed} = \dfrac{R_{33} + R_{35} + R_{53} + R_{55}}{4} \end{cases}$$

Ecuación 2

$$pixel\ G43\ (green\ in\ blue\ row) = \begin{cases} G_{reen} = G_{43} \\ R_{ed} = \dfrac{R_{33} + R_{53}}{2} \\ B_{lue} = \dfrac{B_{42} + B_{44}}{2} \end{cases}$$

Ecuación 3

$$pixel\ G34\ (green\ in\ red\ row) = \begin{cases} G_{reen} = G_{34} \\ R_{ed} = \dfrac{R_{33} + R_{35}}{2} \\ B_{lue} = \dfrac{B_{24} + B_{44}}{2} \end{cases}$$

Ecuación 4

Donde B, G y R se refieren al pixel de color azul, verde y rojo respectivamente. Cada R_{xy}, B_{ij} y G_{kl} se refieren al valor de la celda obtenida de la cámara en esa posición para cada componente. Por medio de las ecuaciones anteriores, se puede obtener cada pixel para la imagen a tratar y están basadas en la estructura de la Figura 4. Ejemplo Patrón de Bayer Figura 4, en donde se pueden ver las posiciones de los pixeles. Pero ese trabajo se deja a la librería OpenCV.

1.2 Espacios de color.

Existen varias formas de codificar u organizar los diferentes colores a partir de componentes básicas, a esto se le conoce como espacios de color. Como tales existen los modelos RGB, HSV, YUV, etc. [5]

El modelo RGB se basa en los sensores humanos, considerando que los colores son una combinación de los tres colores básicos (R)rojo, (G)verde y (B)azul. Estos se representan en un rango de 0 a 255 donde 0 es la ausencia del color y 255 es el máximo de presencia.

El HSV está conformado por H(matiz o hue), S(saturación o inverso de blanco) y V(valor de matiz o brillo). El matiz se representa como un ángulo en grados, y la saturación e intensidad en valores de 0 a 100. En el caso de este trabajo para este espacio de color los valores en la aplicación serán de 0 a 180 para H, y 0 a 255 para S y V, esto debido a que son los rangos manejados por la librería de OpenCV y resulta conveniente dejarlo así.

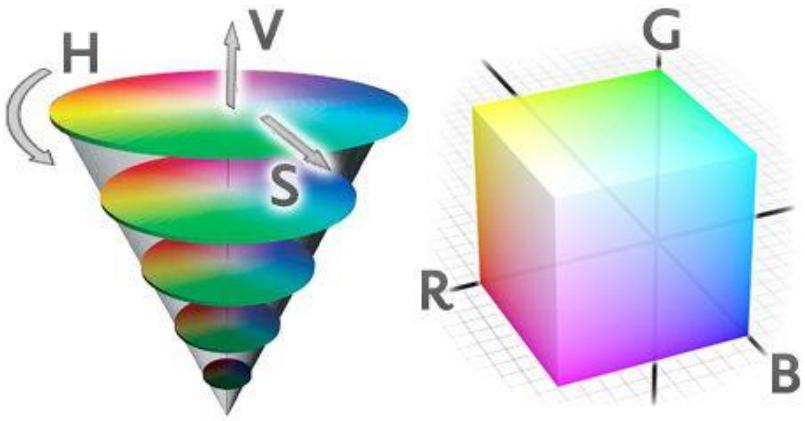


Figura 5. Espacios HSV y RGB[4].

Estos dos espacios tridimensionales se pueden entender mejor con la Figura 5. Espacios HSV y RGB. donde se puede apreciar la conceptualización de estos espacios de color como espacios tridimensionales, donde RGB son 3 ejes coordenados y HSV son distancias y ángulo.

En la aplicación los métodos a usar se probarán con estos dos espacios de color.

4 FUENTE: Modelos de color. http://aprende.colorotate.org/color-models.html

Para cambiar entre ambos espacios se aplican las siguientes fórmulas.

Para pasar de RGB a HSV [6]:

$$\mathrm{H} = \begin{cases} \frac{120(B-\min)}{B+G-2min}, & si\ min = R \\ \frac{120(R-\min)}{R+B-2min}, & si\ min = G \\ \frac{120(G-\min)}{R+G-2min}, & si\ min = B \end{cases}$$

Ecuación 5. Obtención de Hue

$$S = \begin{cases} 0, & si\ MAX = \min \\ \frac{MAX-min}{MAX}, & en\ otro\ caso \end{cases}$$

Ecuación 6. Obtención de Saturación

$$V = MAX$$

Ecuación 7. Obtención de Valor

Donde $MAX = Max(R,G,B)$ y $min = Min(R,G,B)$.

V (valor) y S (saturación) se normalizan entre 0 y 1. H (tono) está definido entre 0 y 360 grados.

Para pasar de HSV a RGB [7]:

Primero obtenemos un color primario H_i.

$$H_i = \frac{H}{60}$$

Ecuación 8

Recordar que H está entre 0 y 360.

Luego obtenemos un color secundario f.

$$f = \frac{H}{60} - H_i$$

Ecuación 9

Calculamos p, q y t usando V y S.

$$p = V(1-S)$$

Ecuación 10

$$q = V(1-fS)$$

Ecuación 11

$$t = V(1 - (1 - f)S)$$

Ecuación 12

Finalmente obtenemos R, G y B y de la siguiente manera:

$$si\ H_i = \begin{cases} 0, & R = V; G = t; B = p \\ 1, & R = q; G = V; B = p \\ 2, & R = p; G = V; B = t \\ 3, & R = p; G = q; B = V \\ 4, & R = t; G = p; B = V \\ 5, & R = V; G = p; B = q \end{cases}$$

Ecuación 13. HSV a RGB

Capítulo 2. Segmentación.

"La segmentación de una imagen consiste en particionar a esta en regiones que pueden tener o no algún significado relativo a la escena que se desea tratar. Esto se hace para estudiar con mayor facilidad aquellos patrones de interés al separarlos de la imagen." [8]

1.3 Segmentación por distancia.

Consiste en medir la distancia entre los pixeles de la imagen a partir de un pixel o conjunto de pixeles que sean de nuestro interés.

Para esta aplicación en particular se obtendrá una muestra de pixeles a partir de la cual se sacará una media que será el pixel desde el que se partirá para calcular las distancias.

Dados dos pixeles $p1 = (r1, g1, b1)$ y $p2 = (r2, g2, b2)$, donde $r1$ y $r2$ son los componentes de color rojo, $g1$ y $g2$ los componentes de color verde y $b1$ y $b2$ los componentes de color azul, las distancias tomadas en cuenta para este trabajo son las siguientes:

1.3.1 Euclidiana.

La distancia Euclidiana es la más sencilla pero a la vez bastante eficiente. Se entiende como la distancia ordinaria que hay entre dos puntos que uno mediría con una regla [9].

$$\delta = \sqrt{(r1 - r2)^2 + (g1 - g2)^2 + (b1 - b2)^2}$$

Ecuación 14.

1.3.2 Manhattan.

Consiste en la diferencia entre dos puntos dada la suma de las diferencias absolutas de sus coordenadas [9]:

$$\delta m = |r1 - r2| + |g1 - g2| + |b1 - b2|$$

Ecuación 15.

1.3.3 Mahalanobis.

También conocida como distancia estadística es una distancia en R^n definida por [9]:

$$\delta M = \sqrt{(p1 - p2)^T S^{-1} (p1 - p2)}$$

Ecuación 16.

Donde S es la matriz de covarianza de los vectores de observación.

Este método permite calcular distancias entre colores ponderadas por la importancia del matiz (color principal) y la varianza en cada componente (R, G y B o H, S y V). Al incluir la matriz de covarianza inversa se logra dicha "ponderación" [10].

1.4 Segmentación por K-Medias (K-Means)

K-Medias es un algoritmo de agrupamiento o "clustering" sin supervisión en el cual el usuario elige de forma arbitraria un número de centros los cuales rápidamente terminan en medio de grupos de datos. Estos datos son asignados a un "clúster" dependiendo de sus rasgos y semejanza con los centros.

Este es un método iterativo por lo cual a medida que pasa cada repetición los datos son ajustados mejor a cada grupo. El algoritmo consiste en:

1. *Tomar un conjunto de datos A de entrada para clasificar.*
2. *Tomar un número K de grupos o clústeres arbitrarios.*
3. *Asignar de forma aleatoria centros para cada clúster.*
4. *Asociar cada punto del conjunto A con su centro más cercano.*
5. *Mover los centros de los clústeres al centroide de sus grupos de datos.*
6. *Regresar al punto 4 hasta que haya convergencia (centroide permanezca quieto).*

El ajuste o compacidad de este algoritmo se da por una función computada después de cada intento como [11]:

$$\sum_i ||muestras_i - centros_{clase_i}||^2$$

Ecuación 17

Donde $muestras_i$ son los pixeles muestra.

A partir de esta compacidad se elige aquella que haya sido la mejor (mínima) y su clase correspondiente es la ganadora, es decir aquella a la que el valor corresponde.

Este algoritmo aunque eficiente posee algunas desventajas que no se pueden dejar fuera en un sistema de tiempo real como es el caso de esta aplicación robótica.

Para empezar existen casos en los que el método converge rápidamente, sin embargo no siempre es así y puede tomar un número bastante alto de iteraciones para hacerlo.

Además k-medias no asegura encontrar los mejores centroides aunque si encontrara siempre convergencia.

Otro problema es que no se especifica el número de clústeres que se deben emplear, ya que cada selección tendrá una solución distinta [12].

1.5 Clasificador Bayesiano.

Un clasificador de Bayes basado en el teorema del mismo (Bayes, 1764) nos es útil para tomar en cuenta probabilidades a "priori" o sucesos que han pasado y combinarlas con probabilidades a "posteriori" o sucesos que llegarán.

El teorema nos dice que dados dos sucesos A y B aleatorios cuyas probabilidades se denotan como p(A) y p(B) respectivamente. Estas son las probabilidades a priori del evento. La probabilidad a posteriori de que un suceso A se verifique o la probabilidad a posteriori de que B pertenezca a A es:

$$p(A|B) = \frac{p(A)p(B|A)}{p(B)}$$

Ecuación 18

Viendo a los conjuntos A y B como patrones se entiende de la fórmula anterior (Ecuación 8) que:

- P(A|B) es la probabilidad posteriori de que dado un patrón B pertenezca a A.
- P(A) es la probabilidad a priori de que cualquier patrón pertenezca a A.
- P(B|A) es la probabilidad condicional de la clase A y es la probabilidad de que una muestra tenga un valor B dado que pertenece a la clase A.

Dado que la formulación de Bayes se puede aplicar para datos aleatorios de una o varias dimensiones resulta conveniente aplicarla a conjuntos de datos en imágenes como es el caso de este trabajo, ya que se obtienen varios puntos o pixeles con 3 componentes de color, ya sea en HSV o RGB, que deben ser asignados a alguna clase de color que nos interese, que en este caso son los colores usados en la competencia.

La probabilidad a posteriori nos ayuda a hacer esta asignación aunque no garantiza una efectividad del 100%. Esto lo hacemos con una regla de discriminación para clasificar varias clases. En este caso usando la misma notación, las clases serían $(A1, A2, \dots, Aj)$ y asignaríamos un patrón Bn a una clase Ak si:

$$\frac{p(B|Ak)p(Ak)}{p(B)} > \frac{p(B|Aj)p(Aj)}{p(B)} \; \forall j \neq k$$

Ecuación 19

Si descartamos p(B) entonces:

$$p(B|Ak)p(Ak) > p(B|Aj)p(Aj) \; \forall j \neq k$$

Ecuación 20

Ahora para poder hacer los cálculos podemos suponer basados en [13] que las probabilidades condicionales de clases $p(B|Ak)\; y\; p(B|Aj)$ tienen una distribución

normal o Gaussiana y de esta forma usar las funciones discriminantes de Bayes de manera simple. Esta distribución para un vector x de varias dimensiones es:

$$G_N(x) = \frac{1}{2\pi^{N/2}|\Sigma|^{1/2}} \exp\left(\frac{-1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Ecuación 21

Donde $|\Sigma|$ es el determinante de la matriz de covarianza y al ser x un vector:

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)(x_i-\mu)^T$$

Ecuación 22

Sustituyendo en las funciones discriminantes por G_N tenemos:

$$Y_k = G_N p(Ak)$$

Ecuación 23

Y aplicando logaritmo:

$$\ln Y_k = \ln G_N + \ln p(Ak)$$

Ecuación 24

Sustituyendo y quitando constantes nos queda la función que nos servirá para evaluar la pertenencia de un pixel a una clase de color:

$$Y_k(x) = \frac{-1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) - \frac{1}{2}\ln|\Sigma_k| + \ln p(Ak)$$

Ecuación 25

Nos evitamos el cálculo del ln del lado izquierdo porque afecta igual a todas las Y_k para k=1, 2, …, M.

Ahora para realizar la asignación de un pixel x a la clase k comparamos la evaluación de dicho elemento en todas las clases y aquella con mayor valor será la ganadora y por lo tanto pertenecerá a dicho patrón, es decir si $Y_k(x) > Y_j(x)\ \forall j \neq k, j = 1, 2, \ldots, M$.

Este algoritmo eficaz resulta muy conveniente para obtener una clasificación muy precisa de lo que queremos, sin embargo su gran limitante y problema en la aplicación de un sistema en tiempo real como lo es esta (y se verá en la sección de Resultados) es su lentitud. Debido a las operaciones que debe realizar sobre cada dato que hay en la imagen, esta solución retarda mucho el tiempo de procesamiento de la información por lo que resulta no factible con las capacidades de cómputo actuales y teniendo en cuenta que el color no es el único factor a tomar en cuenta para la localización de los objetos en la cancha de futbol.

1.6 Segmentación por Umbrales o cubo.

Esta técnica consiste en establecer dos rangos dados por puntos o pixeles que delimiten los valores que se toman en cuenta de cada imagen obtenida. Así se establece un valor RGB o HSV máximo y otro mínimo, lo cual delimita lo que llamamos un cubo de color. Los datos que nos interesan están dentro de este cubo y se asignan a una clase de color.

Tenemos dos pixeles, uno máximo (Pmax) y otro mínimo (Pmin) cada uno con sus componentes de color en RGB o HSV.

Para cada pixel P de una imagen adquirida los valores a tomar en cuenta para una clase C_k están dados por:

$$Si\ (\ (Pmin_{Ck}[red] < P[red] < Pmax_{Ck}[red]) \wedge (Pmin_{Ck}[Ggreen] < P[green] < Pmax_{Ck}[green]) \wedge (Pmin_{Ck}[blue] < P[blue] < Pmax_{Ck}[blue])\ )$$

$$\text{Entonces el P}ixel\ RGB\ está\ dentro\ de\ la\ clase\ de\ color\ C_k$$

Ecuación 26.

Donde red, green y blue se refieren a las componentes roja, verde y azul del pixel P. Los valores Pmax y Pmin están definidos para cada clase C_k.

1.7 K-cubos.

El problema de tener un solo cubo que describa la clase de color que queremos buscar es que estas estructuras no alcanzan a describir completamente las propiedades de los colores lo cual puede llevar a una clasificación incorrecta de pixeles.

Para esto proponemos una técnica basada en el algoritmo de k medias. Los k-cubos determinan varios descriptores para una clase de color, lo cual permite que la clasificación de los pixeles sea más robusta y la segmentación se sobreponga a problemas de iluminación y posición de los objetos.

Determinamos una cantidad i de descriptores máximos y mínimos para la clase, $Pmax_{i,Ck}$ y $Pmin_{i,Ck}$, donde i = 1, 2... n número de cubos definidos.

En otras palabras, para la clase C_k existe un número i de cubos definidos para clasificar los pixeles. Por ejemplo, tenemos varios rangos mínimos y máximos para definir la clase naranja de la pelota.

De esta forma los pixeles de la imagen son clasificados de acuerdo a la Ecuación 26 para cada cubo, por lo que pueden existir datos que no entren en un cubo, pero si en otro para la misma clase.

En la figura 3 se muestra como hay zonas que por la cantidad de poca o mucha luz, no entrarían dentro de un solo cubo en la clase de la pelota naranja pero al tener dos cubos la segmentación de la pelota completa puede ser posible.

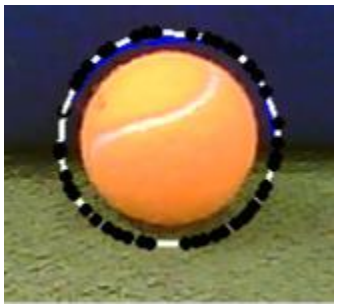


Figura 6. La luz cambia los valores RGB por lo que puede llevar a una mala clasificación.

En general el algoritmo para poder realizar este procedimiento es tan simple como efectivo y es el siguiente:

Para cada pixel P[H,S,V] del cuadro

Para cada cubo [Hmax,Hmin,Smax,Smin,Vmax,Vmin] de la Clase C_i

Si Hmin <= H <=Hmax & Smin<= S <=Smax & Vmin<= V <=Vmax

entonces

Inc(númeroDePixelesDeLaClaseC_i)

P entra en la clasificación de C_i

Fin Si

Fin Para

Fin Para

Capítulo 3. Reconocimiento de Objetos.

Una vez que obtuvimos información de la segmentación de color, utilizamos los pixeles resultantes para poder identificar los objetos que se pueden encontrar en las imágenes extraídas por la cámara del robot.

Se plantean 3 objetos importantes a identificar dentro de la cancha de RoboCup: la pelota, las porterías y los postes laterales. Cada uno de estos objetos posee características únicas que se usan para su clasificación.

1.8 Pelota.

El primero y más importante de los objetos es la pelota, debido a que un partido de futbol está basado en la manipulación de esta para poder anotar puntos o goles. Como ya se explicó en la Introducción., este objeto en la cancha es de color naranja y como toda pelota de futbol, es esférica.

A continuación se explican los métodos propuestos para su detección.

1.8.1 Centroide.

El centroide es básicamente el promedio de los puntos detectados con el color correspondiente a la pelota.

Es decir, después de la segmentación aquellos pixeles resultantes como naranjas de una bola de juego de RoboCup son promediados usando sus coordenadas (x, y) y el resultado es el centro de la pelota y nos indica en la imagen la posición supuesta del objeto.

$$Pelota(x, y) = (\overline{x_i}, \overline{y_i})$$

Ecuación 27. Centroide

Donde $i = 1, 2, \dots, ancho \times alto$, $\overline{x_i}$ y $\overline{y_i}$ son el promedio de las coordenadas de los pixeles en cada componente.

Este método es el que se usaba anteriormente, aunque se desechó al empezar este trabajo dado que no es exacto y tiene muchas fallas al tomar en cuenta el ruido que hay en la imagen, puesto que si hay más pixeles dentro de la segmentación que entren en la clasificación, la posición de estos se tomará en cuenta para calcular el centroide. Esto provocará que las coordenadas (x, y) de la pelota se ubiquen en una posición muy distinta a la real si se detectan cosas como rostros humanos o ropa de tonos rojos.

1.8.2 Cierre convexo.

El cierre convexo (convexhull) de un conjunto Q de puntos es el polígono convexo más pequeño P para el cual cada punto de Q está en su perímetro o en su interior [14].

Esto es como si tuviéramos una banda elástica y la pusiéramos alrededor del conjunto de puntos como si fueran clavos en una tabla y la soltáramos. La figura que describe es el cierre convexo.

El cierre convexo da una buena representación de la figura de los puntos y es la base para el siguiente paso del algoritmo.

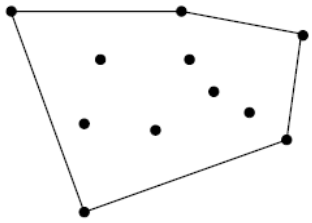


Figura 7. Cierre convexo de un conjunto de puntos.[5]

A partir de esto obtendremos los defectos del cierre convexo o de la convexidad.

Además de encontrar este contorno podemos calcular una secuencia de puntos del contorno entre dos vértices consecutivos del cierre convexo. Esta secuencia forma el defecto de convexidad y es posible calcular la profundidad de un defecto d_i [15]**.**

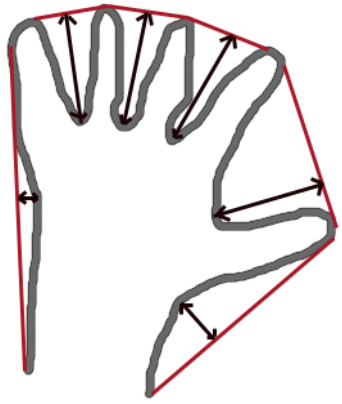


Figura 8. Defectos del cierre convexo.[6]

La idea detrás de la elección de esta técnica, es que un objeto con varios defectos o grandes no puede ser algo esférico como en la figura anterior. Es decir que a menor número de defectos o defectos pequeños el programa puede discriminar si está observando la pelota o no.

La dificultad de este método radica en que el objeto debió estar bien segmentado antes de llegar a este punto, de lo contrario incluso aplicando esta técnica podríamos obtener errores.

5

6 Figuras obtenidas de http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptdes.html (revisada 2013)

1.9 Porterías.

Otro elemento importante en la RoboCup son las porterías. Por supuesto si el robot va a cumplir su misión de ganar el partido debe saber a dónde tiene que meter gol o en otras palabras, hacia donde llevar la pelota.

Para este cometido una primera aproximación se da al detectar los colores que la componen, que son el azul y el amarillo hasta la competencia de 2012.

Ahora falta distinguir el objeto como tal, ya que únicamente usar los colores como formas de detección es mala idea, ya que puede haber otros objetos con estos colores, además de que los postes laterales también son azules con amarillo.

Para este fin se usa la idea de centro de gravedad o centroide de una figura.

1.9.1 Blobs.

Antes de usar solo los centros de gravedad, se tomaban en cuenta los llamados blobs. Estas son regiones de pixeles semejantes agrupados de cierta forma en la imagen. Esto se hacía ya que en torneos anteriores las porterías eran figuras amarillas o azules completamente rellenas de uno de esos colores, por lo que al encontrar un blob azul o amarillo que fuera el más grande, aseguraban que era la portería. Esto no es más así, pues ahora solo los postes de las porterías están coloreados.

Sin embargo, aún se usa un poco de esta teoría en el trabajo actual, pues nos será de utilidad al momento de encontrar otros objetos más adelante.

Para encontrar estos "blobs" en la imagen se ocupa el siguiente algoritmo:

1. *Convierte la imagen fuente en imágenes binarias ocupando umbrales superior e inferior.*
2. *Se extraen los componentes conectados para cada imagen binaria, es decir los* ***contornos*** [16] *y se calculan sus centros.*
3. *Se agrupan los centros de varias imágenes binarias por sus coordenadas. Los centros cercanos de un grupo corresponden a un "blob". La distancia entre blobs tiene que estar ajustada por un escalar.*
4. *Se forman los grupos, se estiman sus centros finales y su radio.*

Otra aproximación descrita en [17] y [18] usa detección de porterías por medio de la obtención de puntos en los extremos de estas y resuelve un problema de posicionamiento al trabajar en un espacio definido por un toroide, lo cual convierte la portería vista desde cualquier dirección a una vista de frente. El problema reside en que una vez más si no se ve por completo el objeto, el algoritmo no funciona o no es seguro que lo haga.

1.9.2 Centro de gravedad.

El centro de gravedad y el centro de masas están en el mismo lugar y estos coinciden con una media aritmética de un conjunto de puntos [19].

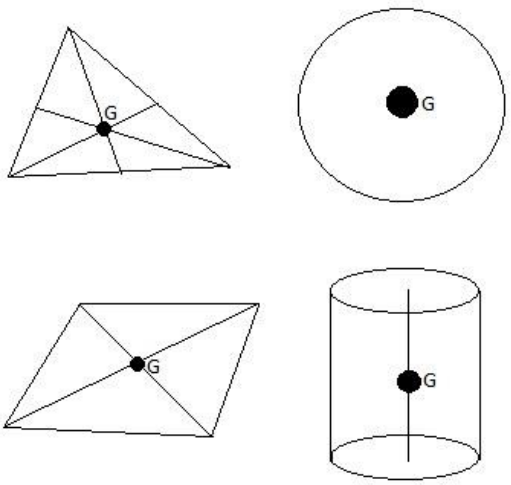


Figura 9. Ejemplos de centros de algunas figuras.[7]

En la Figura 9 se muestran ejemplos de los centros de ciertas figuras geométricas. En esta figura se aprecia que cada una posee dicho centro en una posición única, definida en base a las características de la forma de cada una.

Entonces si tenemos un conjunto de pares $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ el centro de masas o gravedad viene dado por [20]:

$$Xg = \bar{x}$$

$$Yg = \bar{y}$$

Ecuación 28. Centro de gravedad.

Donde $\bar{x}$ y $\bar{y}$ son las medias de los valores x_i y y_i respectivamente.

Basados en esto sabemos que cada figura tiene su centro en un lugar determinado por sus componentes y al tratarse de una figura irregular la portería, este centro tiene una posición característica, la cual nos ayuda a determinar mediante programación que efectivamente el robot está viendo dicho elemento. En la Figura 10 se muestra una aproximación visual al centroide de una portería.

[7] Figura de http://www.taringa.net/comunidades/ciencia-con-paciencia/5274252/I-Fuerzas-paralelas-centro-de-gravedad-y-peso-especifico.html

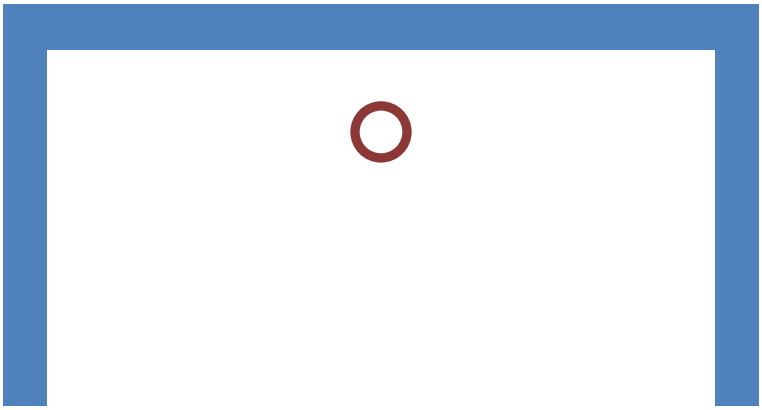


Figura 10. Centroide de portería.

Para esto lo que se ideó fue calcular el centro del objeto detectado y mediante reglas de discriminación de la posición de este centro respecto a los puntos de la figura, decidir si hay o no una portería.

Este método funciona bien para la portería completa, es decir cuando a partir de la segmentación se obtiene la vista del objeto completo. Sin embargo cuando hay obstrucción por algún elemento en la cancha o no se alcanza a ver el objeto en el rango de la cámara, esto provoca obviamente que el reconocimiento no funcione, pues se trata de otro caso u otro objeto.

Para esto el truco ideado en este trabajo consiste en generar dos casos más cuando solo se aprecia parte de la portería. Estos casos se aprecian en la siguiente figura.

Se grafica únicamente una parte de la portería, de esta manera suponemos que solo es visible la parte derecha del objeto. Esto provoca que estemos en presencia de un nuevo tipo de figura, que sigue siendo la portería, pero tiene otro centro de gravedad por contener una configuración diferente.

Lo que hacemos es simplemente calcular su centro de gravedad y localizarlo dentro del objeto de la misma forma que en el caso de la portería completa, sólo que tomamos en cuenta que no todo el objeto es visible. Con un cierto rango de tolerancia, podemos identificar que estamos ante la portería nuevamente.

Para el caso contrario, que sólo veamos la parte izquierda de la portería y el lado derecho esté bloqueado en nuestra visión, hacemos el mismo procedimiento pero localizando el centroide del lado correspondiente de la figura, a la izquierda.

En la Figura 11 se muestra un ejemplo de la posición del centroide y como varía si solo se detecta parte de la portería. En morado se muestra la falta del poste izquierdo y en azul la falta de casi la mitad de la portería. El centroide respectivo se mueve conforme la figura cambia.

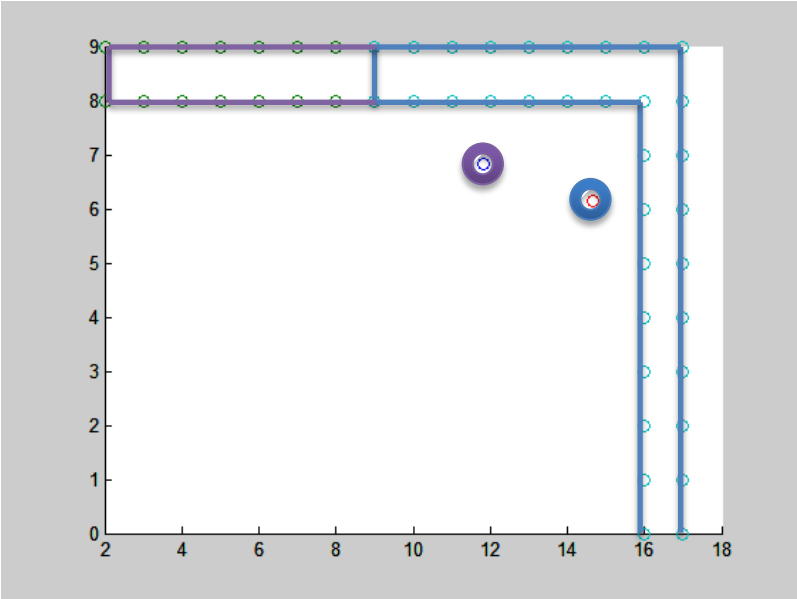


Figura 11. Suponemos que parte de la portería está bloqueada.

1.10 Postes laterales "landmarks".

Los dos postes laterales en RoboCup están determinados desde el año 2009 y son importantes debido a que estos son usados como referencias en la auto localización de los robots dentro de la cancha.

Casi siempre los robots inician el partido buscando estos objetos para saber donde están posicionados.

Estos postes tienen la configuración mostrada en la Figura 12. Dos partes de un color en los extremos por una parte de otro en el centro.

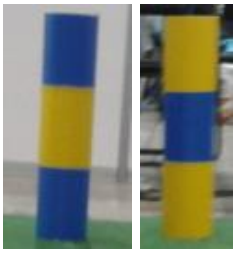


Figura 12. Configuración de postes laterales.

1.10.1 Método de 5 marcas.

Este método consiste en encontrar 5 marcas características de los "landmarks" basado en el trabajo de [21]. Estas son la esquina superior izquierda (X_1, Y_1), superior derecha (X_2, Y_2), la inferior izquierda (X_3, Y_3), la inferior derecha (X_4, Y_4) y el centro (X_c, Y_c). Usando estos cinco puntos se puede encontrar además la longitud horizontal F_H y la longitud vertical F_V.

Este método tiene la desventaja de que necesita ver por completo el poste, ya que de lo contrario los cinco puntos no corresponderán a lo buscado y con que una porción del

poste esté ocluida por otro objeto o no esté presente por completo en el campo de visión de la cámara, no se detectará el landmark.

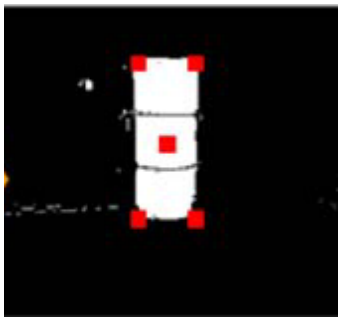


Figura 13. Detección de Landmark o poste lateral por cinco puntos.

La Figura 13 muestra los puntos característicos de un poste en este método. La ausencia de uno de ellos debido a una oclusión provoca que no se pueda detectar el objeto.

1.10.2 Método de reconocimiento de objeto muestra.

Un método mostrado en [22] para detectar los postes consiste en clasificar los colores que conforman el poste de la siguiente forma.

Para un poste con capas amarillas arriba y abajo y el centro en azul como en la parte derecha de la Figura 12. Configuración de postes laterales. es definido con la etiqueta YBY-landmark. El poste con la configuración contraria es definido como BYB-landmark.

Para este método se calculan etiquetas de YBY-landmark y BYB con la ecuación

$$P_{YBY}(x,y) = L_Y^i(x,y) \cup L_Y^j(x,y) \cup L_B^k(x,y)$$

$$if\{|L_Y^i(x_c) - L_Y^j(x_c)| < \beta_Y\} \cap \{L_Y^i(y_{max}) < L_B^k(y_c) < L_Y^j(y_{min})\}$$

Ecuación 29. Asignación de etiquetas para landmarks o postes laterales.

Donde P_{YBY} se refiere al poste amarillo, debido a su configuración amarillo-azul-amarillo, (x,y) son las coordenadas dentro de la imagen del objeto, L_Y^i se refiere al objeto i-ésimo de color amarillo detectado, L_Y^j al j-ésimo objeto amarillo y L_B^k al k-ésimo objeto azul detectado.

De acuerdo con el procedimiento de etiquetado mencionado anteriormente, se etiquetan todos los componentes de color amarillo y azul en la imagen y asigna números a los componentes.

L_Y^i define los píxeles de la i-ésima componente de color amarillo (Y), y_{max} y y_{min} el valor mínimo y el valor máximo para el objeto i en dirección del eje y, respectivamente, x_c y y_c el punto central del objeto en la dirección horizontal y vertical respectivamente. El valor de tolerancia horizontal β_Y lo fijan en 15. El poste se compone de dos objetos del mismo color en la línea vertical, y el centro es de un color diferente. Si se puede

encontrar un objeto con esta composición, el sistema puede tratar este objeto como el punto de referencia y envía los datos de las coordenadas.

De manera gráfica se puede entender este método con la Figura 14. Dos cuerpos amarillos están arriba y debajo de uno azul respectivamente y además los centros de los cuerpos amarillos respecto del eje x están cercanos entre sí bajo una tolerancia dada por β_Y.

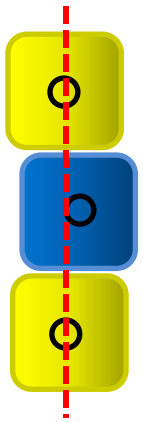


Figura 14. Idea gráfica de la detección de postes laterales por sus centros

Ahora sobre esta idea, se agregó una modificación a la ecuación de tal forma que también se contemplara la cercanía de los objetos en su eje horizontal. Esto consiste en tomar en cuenta la constante beta para medir la distancia entre las partes superior e inferior con respecto a la inferior.

De esta forma la ecuación quedó de la siguiente forma:

$$P_{YBY}(x,y) = L_Y^i(x,y) \cup L_Y^j(x,y) \cup L_B^k(x,y)$$

$$if(\{|L_Y^i(x_c) - L_B^k(x_c)| < \beta_Y\} \cap \{|L_Y^j(x_c) - L_B^k(x_c)| < \beta_Y\} \cap \{L_Y^i(y_{max}) < L_B^k(y_c) < L_Y^j(y_{min})\})$$

Ecuación 30. Modificación de etiquetado de postes laterales

Donde $|L_Y^i(x_c) - L_B^k(x_c)| < \beta_Y$ representa la distancia del objeto central, en este caso azul, al objeto amarillo superior. De manera similar se compara con la distancia sobre el eje x hacia el objeto inferior. Así se asegura que el programa no tome en cuenta objetos como los descritos en la siguiente Figura, donde se aprecia el error que tiene la ecuación sin la modificación.

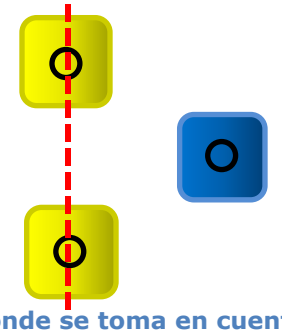


Figura 15. Error en el etiquetado, donde se toma en cuenta objetos como este y se marcan como landmarks.

Capítulo 4. Experimentos y resultados.

Ya planeados los pasos a ejecutar para hacer el programa de visión, se procedió a programar y ejecutar pruebas.

A continuación se presentan los métodos y plataformas de programación que se usaron para llevar a cabo las pruebas de los algoritmos propuestos y los resultados de estos. Así mismo se justifica la elección del método que elegimos para segmentar el color y reconocer objetos, mostrando su eficiencia y eficacia por el tiempo y resultados correctos.

1.11 Equipo usado.

Empezaré listando el equipo sobre el cual se realizaron las pruebas y las plataformas de desarrollo que se usaron:

1.11.1 Hardware

- Plataforma de desarrollo abierta DarwinOP. (El robot)
- FitPC 2 con un procesador Intel Atom a 1.6 GHz con 1 GB de RAM. (La computadora que llevan los robots para procesar los datos)
- PC con Pentium 4 a 3.4 GHz con 2GB de RAM.
- Cámara Minoru 3D con una resolución de 800 x 600 pixeles.
- Cámara Logitech C905 resolución de hasta 1600x1200. Aunque por velocidad se optó por 640x480 y obtención de casi 30 cuadros por segundo.

1.11.2 Software

- Distribución de Linux Khabelix.
- C++.
- Librería OpenCV para procesamiento de imágenes [23] [24].
- MATLAB para graficas explicativas.

1.11.3 Muestras

Se evaluó el desempeño con 54 imágenes tomadas con la cámara del robot a una resolución de 640x480 pixeles. Además se contó con videos de la vista del robot en competencia y en el laboratorio, pero para fines de experimentos se usaron las imágenes.

Se manejaron 43 imágenes de la bola, 25 imágenes de los postes y 35 imágenes de las porterías.

1.12 Obteniendo las imágenes y definiendo el espacio de color.

Lo primero para realizar las pruebas es obtener la información deseada, que en este caso son la secuencia de fotos que toma el robot a través de una cámara web, que le servirán para adquirir la información en su entorno.

Recordemos que en este tipo de cámaras digitales tenemos presente el inconveniente del ruido que es ingresado por el filtro Bayer. Esto aunque mínimo, puede resultar un problema al momento de buscar los objetos por color, ya que en la interpolación entre colores se producen artefactos que podríamos confundir con lo que nos interesa si es que estos llegan a encontrarse muy lejos.

Por ejemplo la pelota. A veces una pelota al otro lado del campo puede resultar en un pixel o muy pocos, que se pueden confundir con el ruido del filtro en otro lugar, y por lo tanto nos genera una información errónea.

Dado que el filtro Bayer viene ya de fábrica con las cámaras digitales se requiere poder acceder a los controladores de estas para poder manipularlo.

Esto representa un problema ya que también se debe programar las funciones y para esto se debe poseer una interfaz entre el controlador y el lenguaje de programación, en este caso C++.

Para cámaras que se conectan por firewire el CINVESTAV proporcionó un código que permite realizar esta operación, sin embargo las cámaras que tenemos para el DarwinOP y para el nuevo robot son de interfaz USB por lo que no son compatibles.

Patrón de Bayer en OpenCV.

Como el patrón de Bayer es usado en cámaras CMOS y CCD, OpenCV tiene la opción de que las imágenes adquiridas por estos dispositivos sean convertidas a cualquier tipo que maneje esta librería (BGR, HSV, YCrCb, etcétera) por medio de la función cv::cvtColor de C++ y cvCvtColor de C.

Las opciones que posee son : `CV_BayerBG2BGR`, `CV_BayerGB2BGR`, `CV_BayerRG2BGR`, `CV_BayerGR2BGR`, `CV_BayerBG2RGB`, `CV_BayerGB2RGB`, `CV_BayerRG2RGB`, `CV_BayerGR2RGB`.

Esta función permite obtener imágenes de un solo plano donde los pixeles R, G y B (sensores de un componente particular) están arreglados de la forma:

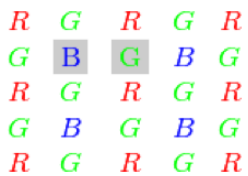


Figura 16. Arreglo Bayer en OpenCV[8]

[8] Figura de http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html

Los componentes de salida RGB de un pixel son interpolados de 1, 2 o 4 vecinos del pixel con el mismo color.

De esta forma podemos manipular estos datos si así se requiriera, sin embargo, esto no resulta indispensable, pues las funciones de segmentación son lo suficiente robustas para evitar errores importantes en este aspecto ya que al clasificar los pixeles, los artefactos que pudieran llegar a haber quedan fuera de las clases. Esto se verá más adelante.

Espacio de Color.

El segundo aspecto a considerar es la elección del espacio de color sobre el que se va a trabajar.

Por defecto, las cámaras usadas en este trabajo obtienen una imagen que es mostrada al usuario en el espacio RGB. Esto es un aspecto importante a considerar, puesto que al ser una aplicación en tiempo real necesitamos que la información se trate con la mayor velocidad posible.

Sin embargo este espacio de color presenta una desventaja importante y es que al más mínimo cambio de luz los valores que se obtienen de una misma imagen varían mucho en sus 3 componentes (Rojo, Verde y Azul), dado que se trata de un espacio que se basa en la combinación de colores primarios para obtener uno solo.

Figura 17. Una misma escena con distintos niveles de luz.

En la Figura de arriba se puede apreciar el mismo escenario con elementos de una cancha y se ve como un cambio de iluminación altera los colores de los objetos al ser percibidos por la cámara digital con la que se trabaja.

Para ver las variaciones en los datos se seleccionó una región en la imagen con datos del color amarillo y se graficaron para ver como variaban gracias a la luminosidad del ambiente.

En la Figura 17 se puede apreciar el cambio de niveles de iluminación entre las regiones. Cuando la luz es cambiada los datos en RGB sufren un cambio más significativo debido a que sus tres componentes tienen un cambio importante, lo que

resulta en una variación mayor. Sin embargo en el espacio HSV el cambio que se aprecia es menor. Esto se debe a que no todos los valores sufren este cambio significativo. Al variar la luminosidad del ambiente los objetos conservan su tonalidad pero varían en la cantidad de brillo que poseen, es decir el valor. Este es un dato importante para la elección del espacio de color a usar, ya que el proceso de calibración de colores es tedioso ante los cambios de horario y luz que hay en las competencias, y un cambio mínimo en los datos acelera este proceso.

En general el reconocimiento a través de HSV es más usado ya que este modelo es mucho menos influenciado por la iluminación o luz, ya que describe el color y brillo de los objetos [22].

El código de este graficado de datos se puede ver en el Anexo B. Código MATLAB para graficado de pixeles en RGB y HSV. de este documento.

A partir de estas pruebas se decidió optar por el espacio de colores HSV, ya que aunque implica transformar el espacio obtenido por la cámara, este proceso es rápido y se computa de manera fácil gracias a OpenCV con la función que trae incluida `cvtColor(fuenteBGR,destinoHSV,CV_BGR2HSV)`.

Cabe señalar que las imágenes adquiridas por la cámara del robot son de una resolución de 640 pixeles de ancho y 480 de alto, ya que de esta forma aumentamos mucho la cantidad de información adquirida de cámaras previamente usadas en otros modelos de robots anteriores, como la CMUcam3 con una resolución de 352 x 288 y que procesaba un máximo de 26 cuadros por segundo.

En la Figura 18 podemos ver como varían los datos dentro de la región seleccionada con un círculo rojo en la imagen. Corresponden a la parte central de un landmark azul, es decir a su parte amarilla. El primer renglón de gráficas muestra los datos adquiridos en el espacio RGB y el segundo renglón muestra los mismos datos transformados al espacio HSV. Aquí podemos apreciar que el cambio que sufren los pixeles debido a la iluminación es más notorio en el primer renglón, mientras que en HSV el cambio es menor, por lo tanto obtenemos menos variaciones.

1.1 Segmentación.

Una vez definidos los parámetros de entrada de la imagen y el formato para tratar la información tenemos que empezar a procesarla.

El primer paso es diferenciar los objetos que nos pueden interesar, para esto debemos separarlo del resto de la imagen. Este procedimiento lo haremos por medio de una segmentación de colores, pues cada elemento en la cancha se distingue de los demás notoriamente por sus colores.

Para esta segmentación se probaron los métodos vistos en la primera sección del documento para poder decidir cual es la mejor opción para el sistema de visión de los robots.

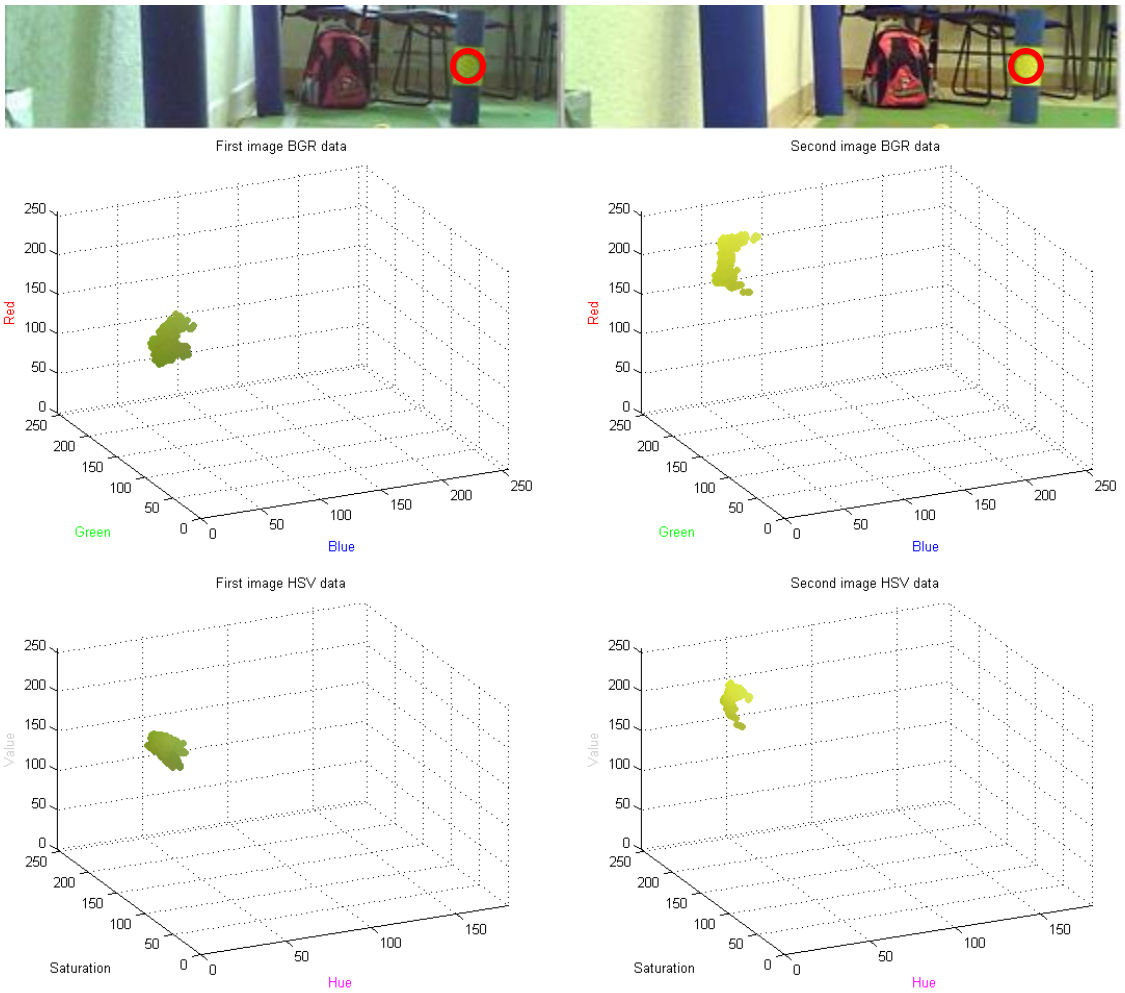


Figura 18. Comparación de los datos en un cambio de iluminación.

Los objetos y sus colores son los siguientes.

Objeto	Color correspondiente
Pelota	**Naranja**
Porterías	**Azul** y **Amarillo**
Postes laterales o beacons	**Azul** y **Amarillo**
Otros robots en la cancha	**Magenta** y **Cian**
Campo	**Verde**
Líneas del campo	**Blanco**

Tabla 1. Colores usados en RoboCup

Calibración.

Antes de empezar a clasificar los pixeles necesitamos una muestra que nos indique los valores con los que se debe trabajar, es decir un conjunto de pixeles con la información de cada elemento a clasificar. Para este trabajo se tomaron fotos usando la cámara del robot en el laboratorio de pruebas y en la cancha de competencia. Con esto tenemos imágenes con las condiciones para trabajar.

El primer método probado fue por medio de *umbrales* o un *cubo*. Este mecanismo consiste en crear niveles que serán aceptables para tomar en cuenta un pixel dentro de una clase. Básicamente para cada canal del espacio de color se establecen dos límites, uno mínimo y otro máximo. De esta forma solo tomamos en cuenta para un objeto sus pixeles si es que están dentro de estos rangos, en un espacio en tres dimensiones, esto imaginariamente lo podemos ver como aceptar aquellos datos que entren en un cubo delimitado por un ancho, alto y largo.

Por ejemplo, para encontrar los pixeles en una imagen que pudieran corresponder a una pelota naranja, primero tomamos una muestra de pixeles que correspondan a dicha pelota en una imagen, luego obtenemos una media de los valores que tomamos como muestra y sacamos los mínimos y máximos de cada canal de los pixeles. Es decir obtenemos el máximo H, mínimo H, máximo S, mínimo S, máximo V y mínimo V de todos los datos muestreados. Como la imagen varía mucho dependiendo del movimiento del robot en la cancha, necesitamos tomar muestras de imágenes desde varias perspectivas y a partir de todas esas muestras obtener nuestra referencia. En pruebas en laboratorio se tomaron varias imágenes de la cancha desde distintas vistas para así tener un conjunto de datos más confiable y con la mayor cantidad de variación posible para poder tomar en cuenta todos los cambios de luz posibles. Se hicieron tomas de día, tarde y noche con distintas fuentes de iluminación en cada tiempo.

Figura 19. Muestras de objetos en distintas situaciones de iluminación.

Para ver todas las imágenes dirigirse al Anexo 3 de este documento.

A partir de este conjunto de imágenes se obtuvieron miles de muestras de colores para empezar a hacer la clasificación por colores de los diferentes objetos que se quieren obtener.

El método de umbrales es muy rápido, pues para cada pixel de la imagen resulta en solo dos comparaciones para saber si será tomado en cuenta o no. La desventaja de este es que la misma variación de luz puede afectar los datos y dependiendo si uno deja muy abiertos los umbrales se pueden tomar datos de la imagen como objetos de

interés cuando no lo son, es decir resulta en falsos positivos. Y al contrario, si se dejan muy cerrados estos umbrales se pueden dejar de lado pixeles que son de interés para nosotros, aunque esto significa aumentar la confiabilidad de los datos clasificados.

Usando distancias.

Al momento de una competencia el calibrar o corregir una calibración con rapidez puede resultar en una ventaja durante el partido. Debido a esto ajustar 6 parámetros puede ser lento e incorrecto bajo presión.

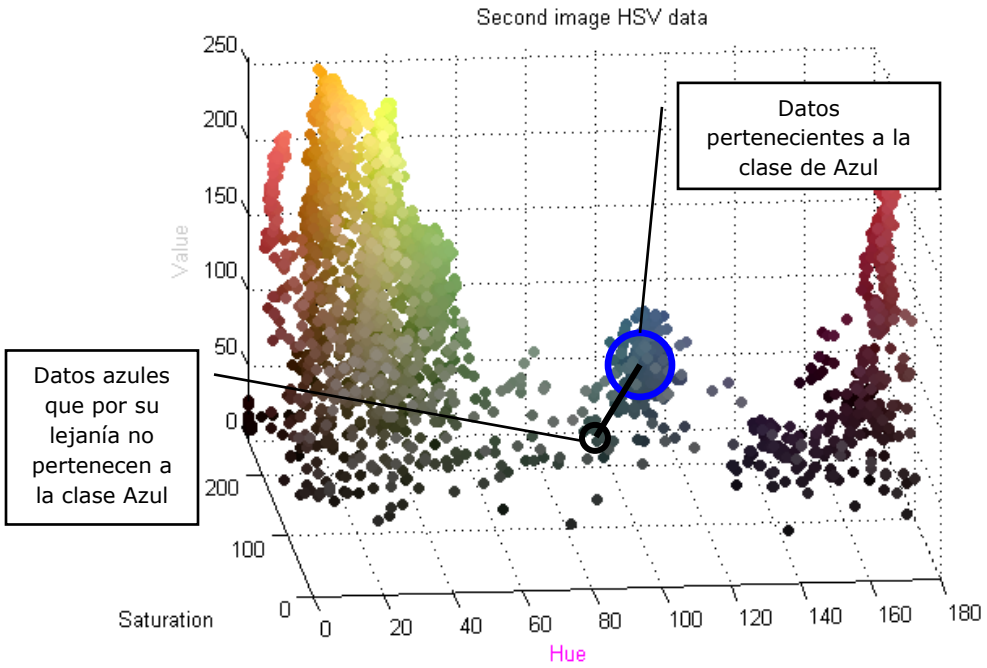


Figura 20. Cercanía de pixeles a sus semejantes. Solo aquellos suficientemente cercanos pertenecerán a una clase determinada.

Una solución propuesta para esto fue la utilización de segmentación por distancias. La idea consiste en que bajo la premisa de que estamos trabajando en espacios tridimensionales de color, cada pixel puede verse como una coordenada en el espacio. Entonces a partir de un punto medio que ya calculamos en la extracción de las muestras de las imágenes para cada clase, podemos elegir los pixeles que pertenecen a estas por medio de su proximidad al centro. Esto debido a que los pixeles cercanos a este punto medio son aquellos parecidos a los objetos de interés.

En la Figura 20, podemos ver como existen pixeles parecidos al resto de cada una de sus clases. Es decir, hay pixeles azules que podrían ser parte de una portería o poste, sin embargo como sabemos que el color de estos elementos es poco variable, deberíamos descartar aquellos azules que no sean parte de estos objetos y tal vez sean ruido provocado por otros involucrados en la imagen.

A partir de esto es como se establece la idea de distancia, pues si un pixel azul está muy cerca de la muestra o media obtenida, entonces podríamos decir que es parte del objeto que queremos identificar y si está muy lejos podríamos decir lo contrario.

La *ventaja* de usar distancias radica en la velocidad y simplicidad para poder calibrar los sistemas ya que como se explicó antes solo se requiere variar un parámetro, que es la distancia.

El problema es que de forma imaginaria estamos definiendo una esfera dentro de la cual existen aquellos pixeles o valores que suponemos entran en nuestra clasificación y al igual que en el método de umbrales, si la esfera es muy pequeña o muy grande podemos dejar fuera o tomar en cuenta datos que no corresponden a nuestros objetos y provocar error en el reconocimiento.

Clustering por K-Medias y clasificador de Bayes.

Un acercamiento a mejorar la clasificación de pixeles es usar algoritmos que usen más información que solo límites o distancias.

La primera propuesta fue utilizar un clasificador con k-medias. La teoría de este se ha explicado anteriormente.

Lo primero que tenemos que hacer es elegir el número de clústeres que deseamos, lo cual no es tarea trivial. Una idea nos la da el número de clases de colores que tenemos que clasificar, es decir 7 clústeres que son para naranja, azul, amarillo, cian, magenta, verde y blanco. Quizás uno extra para el resto.

Este algoritmo resulta interesante pues aunque nosotros directamente no proporcionamos los datos base para clasificar, obtiene resultados aceptables para imágenes en un ambiente controlado como es el laboratorio. La desventaja es el tiempo de procesamiento que este método implica ya que el mejor tiempo obtenido al hacer pruebas fue de 4 segundos para procesar una imagen con 7 clústeres. Es mucho tiempo para un solo cuadro considerando que debemos procesar a una velocidad mucho mayor pues es en tiempo real y quedarse parado procesando un cuadro por tanto tiempo simplemente no es factible.

En la Figura 21 se muestra el resultado de este procesamiento. Aunque de inicio parece funcionar, conforme se añaden más elementos, el algoritmo empieza a dejar fuera ciertos objetos. Esto se puede corregir añadiendo más iteraciones o más clases, pero implica un mayor tiempo de procesamiento del cuadro que resulta en pérdida de tiempo.

En la primera imagen la segmentación resulta buena aunque por el número de clases usadas, se ver forzada a separar el amarillo y verde en dos. Además la bola es clasificada como amarilla. Se usaron 7 clústeres. Esto provoca que tengamos una clase de amarillo claro y otra de amarillo oscuro debido a la sombra y a que obligamos al algoritmo a usar un número determinado de clases.

En la segunda imagen se añadieron artefactos del color de elementos en una cancha como son tonos de magenta y de cian. El resultado fue mejor aunque la bola y el suelo siguen con problemas. Se usaron 7 clústeres.

Y en la tercera imagen vemos una situación muy común en competencia. La imagen se ve saturada de elementos que meten ruido a la información. En este caso el algoritmo falla más y aunque se resuelve aumentando clústeres, se pierde tiempo en el procesamiento. En la primera prueba se usaron 7 clústeres y en la segunda 14.

Esto resulta en una dificultad pues no se puede predecir el número de clústeres que se deben emplear para obtener las clases que necesitamos. No se tienen en general condiciones óptimas o sin ruido durante el movimiento de un robot y el movimiento de su cabeza donde está la cámara provoca ver siempre aspectos extras.

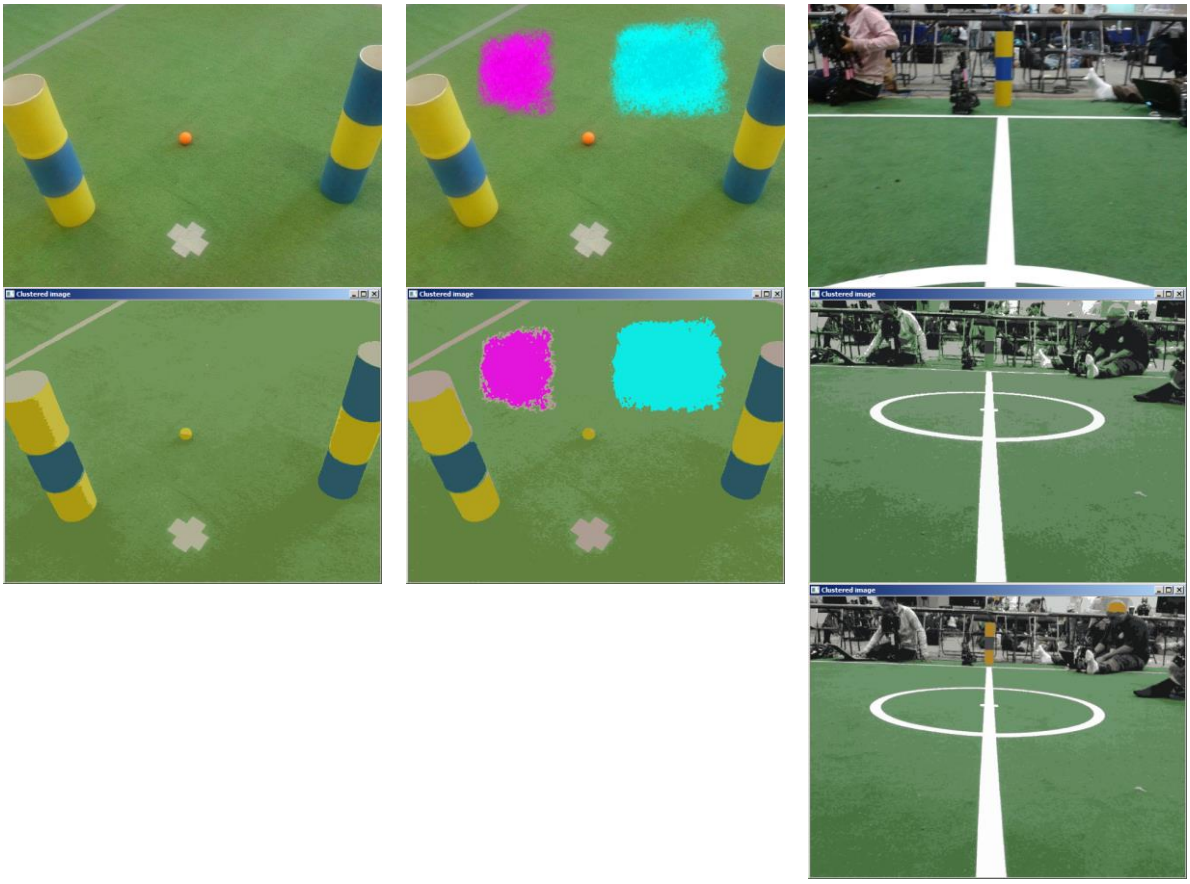


Figura 21. Resultado de K-Medias

Para esto se aprovechó la función `kmeans` integrada en OpenCV.

Clústeres	Iteraciones	Tiempo
7	5	4 segundos
14	5	15 segundos
14	10	51 segundos

Tabla 2. Tiempos k-medias.

En la tabla anterior se muestran los resultados de tiempos promedio que toma al algoritmo ejecutarse.

Otra alternativa fue usar un clasificador de Bayes, que se basa en la probabilidad de que un elemento esté presente en una clase con un previo conjunto de entrenamiento. Este conjunto se obtuvo a través de las muestras definidas a partir de las imágenes en laboratorio y en competencia.

Lo primero que se hizo fue calcular la matriz de covarianza y media para cada canal del conjunto de entrenamiento con la función `calcCovarMatrix()`.

Luego calculamos el determinante de la matriz de covarianza, su inversa y su logaritmo para la primera parte del algoritmo. Esto con las funciones `determinant()`, `invert()` y `log()`.

Posteriormente pasamos a calcular la función discriminante para cada pixel y cada clase:

```
for(a=0; a<NUMBER_OF_CLASSES; a++)
{
	//Pixel - mean
	pix_mean = pixel - meanMat[a];
	//Mul with inverse covariance matrix and pix_mean transpose
	multemp = (pix_mean * invCovar[a] * pix_mean.t());
	//rest of bayes
	y[a] = (float)((multemp.at<float>(0,0) / -2.0) - logCovar[a]/2.0);
}
```

El resto es clasificar de acuerdo a los valores obtenidos en y[], donde el valor más alto indica a que clase pertenece un solo pixel. Para ver el resto del código ir al Anexo 4.

Como ya es de imaginarse, tantos cálculos y operaciones por pixel en la imagen trae el primer problema a la vista. Este procedimiento aunque efectivo, es muy costoso en tiempo de procesamiento.

Veamos los resultados en la Figura 22. Como se aprecia, hay una segmentación muy buena, el algoritmo es bastante eficiente pues clasifica la mayoría de los pixeles de manera correcta al asignarlos a sus clases deseadas.

Sin embargo la gran desventaja es que tarda para cada imagen con 7 clases un promedio de 35 segundos. Esto prácticamente descarta el método para una aplicación en tiempo real.

Del lado izquierdo se ve la imagen original, del derecho el resultado y abajo el número de muestras para cada clase donde:

- *0 es Naranja*
- *1 es Azul*
- *2 es Amarillo*
- *3 es Verde*
- *4 es Blanco*
- *5 es Cian*

- *6 es Magenta*

Y se incluyó una clase más para el resto, es decir aquellos pixeles que por medio de un umbral definido no entraban en ninguna clasificación. Son los que resultan en color negro en la imagen.

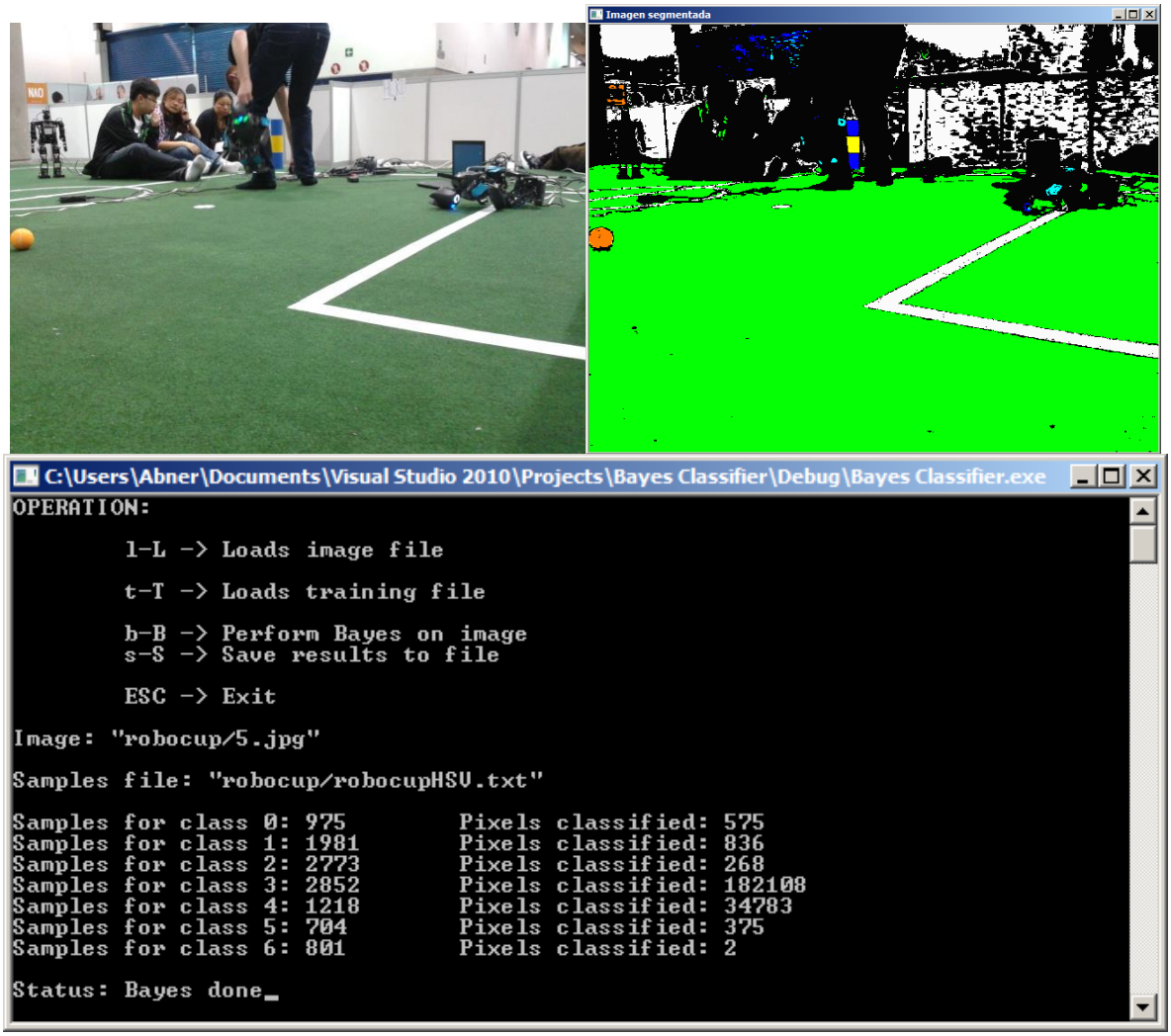


Figura 22. Ejemplo resultado clasificador de Bayes.

De esta forma aunque Bayes y K-Medias son algoritmos útiles para clasificar, no resulta conveniente usarlos para la aplicación por su tiempo de ejecución.

Para este inconveniente desarrollamos un nuevo método que combina la segmentación por umbrales con la teoría de k-medias.

Segmentación por k-cubos.

Ya se había explorado la utilización de umbrales para segmentar una imagen. Un problema de esta aproximación radica en la iluminación que tiene el ambiente y como influye en los objetos, en especial la pelota pues en general siempre se presenta una

luz desde la parte superior del lugar en donde se realizan las pruebas, ya sea por el Sol o lámparas de techo. Esto crea una zona de sombra que modifica los colores de los objetos lo cual no permite que se puedan clasificar completamente los pixeles que conforman algún elemento en la cancha.

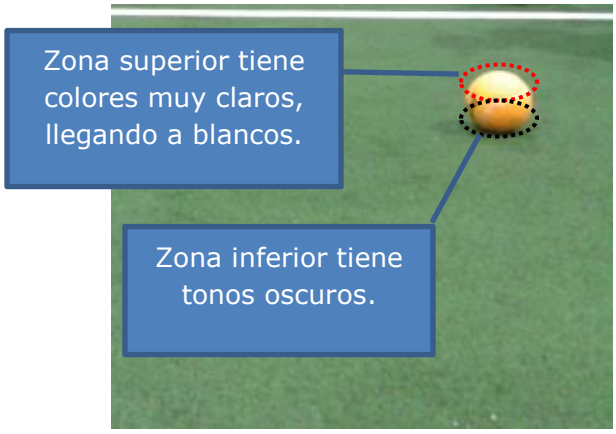


Figura 23. La luz superior crea sombras que modifican los valores percibidos por la cámara.

En la Figura 23 podemos apreciar como la sombra provoca que los pixeles correspondientes a la parte inferior de la pelota sean de otro rango que de los de la parte superior.

La ventaja de este método era su velocidad, pues se puede procesar casi 30 cuadros por segundo con una resolución de 640x480 pixeles.

La solución propuesta fue usar la idea detrás de k-medias al crear más de un centro de color para cada clase. Esta idea consiste en poder clasificar aquellos pixeles que queden fuera y sean necesarios para distinguir un objeto.

De esta forma regresando al ejemplo de la pelota, podemos crear un centro de color para la parte oscura y otro para la parte clara y clasificar los pixeles en la imagen por medio de umbrales por cada centro.

Veamos un ejemplo. En la siguiente imagen se estableció un cubo con los siguientes valores:

Mínimo Tono (H)	Máximo Tono (H)	Mínima Saturación	Máxima Saturación	Mínimo Valor	Máximo Valor
19	33	96	203	167	255

Tabla 3. Valores mínimos y máximos de HSV de un cubo.

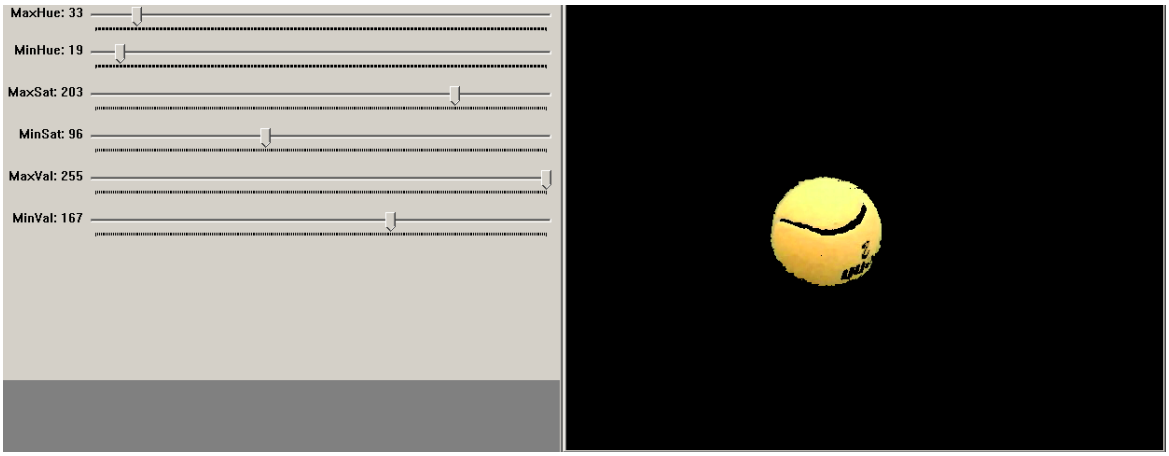


Figura 24. Resultado de calibración con un cubo solamente la pelota.

En la Figura 24 apreciamos una muy buena segmentación a través de un cubo definido para la pelota. Del lado derecho vemos el resultado y en el lado izquierdo vemos los rangos de inclusión para el cubo en HSV.

Y ahora usando el mismo cubo que cubre perfectamente los pixeles de la pelota se hace la segmentación sobre la siguiente imagen con su resultado:

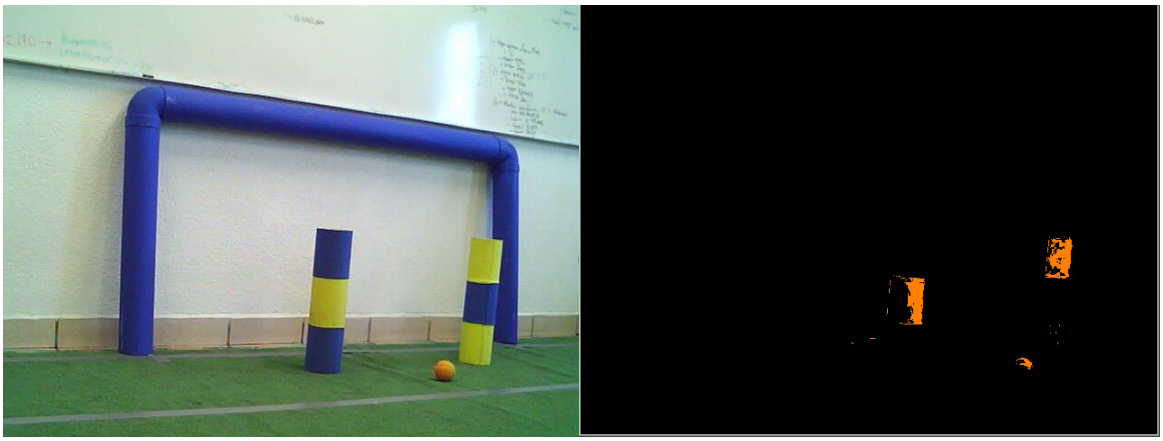


Figura 25. Resultado de calibración con un cubo con varios elementos.

En la Figura 25 se suscita un problema de segmentación, debido a que los objetos son parecidos. Esto provocado por iluminación y a que el rango incluía a los pixeles parecidos entre pelota y amarillo.

El problema es que los valores de algunos objetos, en este caso las partes amarillas, entran en los rangos establecidos anteriormente y por lo tanto obtenemos información falsa positiva. Además en la sección de la pelota, se puede apreciar que no está completa, pues por efectos de iluminación, algunos pixeles quedaron fuera, estos son falsos negativos pues debieron entrar en nuestra segmentación.

Ahora vamos a usar dos cubos distintos para el naranja, uno que cubra su parte superior y otro la inferior. El resultado y los valores para los rangos son los siguientes:

Mínimo Tono (H)	Máximo Tono (H)	Mínima Saturación	Máxima Saturación	Mínimo Valor	Máximo Valor
16	22	177	223	130	223
7	16	175	255	68	186

Tabla 4. Valores máximos y mínimos de HSV para dos cubos.

Figura 26. Resultado de calibración con dos cubos con varios elementos.

De esta forma gracias a que tenemos por separado ambos cubos, podemos crear con mayor exactitud los rangos para segmentar los pixeles correspondientes a cada elemento. En la Figura 26 se muestra que los objetos amarillos que aparecieron antes ya no siguen en el resultado, por el contrario aquellos pixeles de la parte inferior de la pelota que no se encontraban anteriormente ahora si están presentes.

El programa se arregló y compiló bajo la distribución de Linux Khabelix. El software también usa la librería de OpenCV y se corrió con distintas modalidades en distintas computadoras para medir su funcionamiento.

Se probó primero en una PC con Pentium 4 a 3.4 GHz con 2GB de RAM usando una cámara Minoru 3D con una resolución de 800 x 600 pixeles y se obtuvo la siguiente velocidad de procesamiento:

Con salida gráfica activada	Salida gráfica desactivada
4 FPS (Frames Per Second)	29 FPS

Tabla 5. Resultados para PC.

Luego se probó en la FitPC que lleva el robot en construcción para la RoboCup. Cuenta con un procesador Intel Atom a 1.6 GHz con 1 GB de RAM usando la misma cámara con los siguientes resultados:

Con salida gráfica activada	Salida gráfica desactivada
4 FPS	12 FPS

Tabla 6. Resultados para FitPC.

Y por último se cargó sobre la FitPC que está en el robot DarwinOP usando la cámara que trae integrada, Logitech C905.

Con salida gráfica activada	Salida gráfica desactivada
4 FPS	7 FPS

Tabla 7. Resultados para DarwinOP.

En todas las pruebas se detectaron las porterías y las marcas en el ambiente controlado del laboratorio de pUNAMoids en el IIMAS y la pelota se detectó de forma correcta a una distancia de 4 metros que es la medida del ancho oficial de la cancha de Kid Size.

Durante la competencia.

En la competencia de RoboCup se probó este método y los resultados fueron bastante positivos. A continuación se presenta una tabla de confusión en la cual se aprecia que la cantidad de pixeles detectados correctamente es muy alta y aunque hay errores de clasificación, estos son mínimos. Para un conjunto de imágenes de la competencia ir al Anexo C. Imágenes de muestra. Para ver la lista completa de tablas de confusión de pixeles clasificados de estas imágenes de muestra ir al Anexo E. Tablas de confusión.

En la Figura 27 apreciamos un ejemplo de la segmentación realizada en ambiente de la competencia de la RoboCup. Vemos como se realizó la segmentación correcta de las diferentes clases. La pelota, las marcas cian de los robots, la portería amarilla y elementos falsos positivos de azul. Esto está mostrado en la tabla de abajo también.

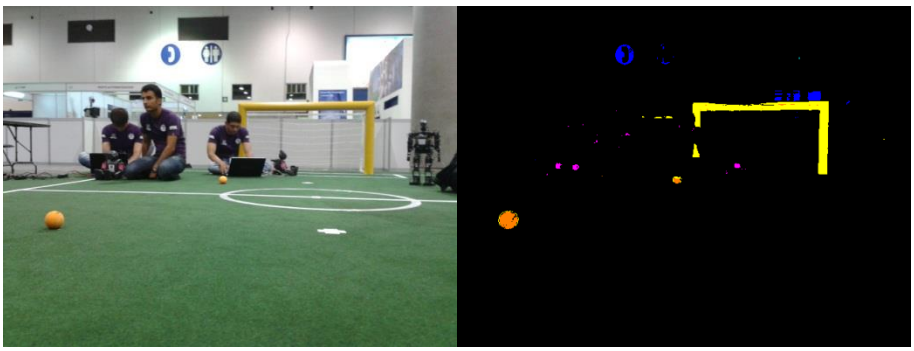


Figura 27. Imagen muestra de la competencia segmentada con K-Cubos.

Clase	Pixeles clasificados	% PC	Positivos Verdaderos	% PV	Positivos Falsos	% PF	Negativos Verdaderos	% NV
Naranja	721	0.235%	**683**	0.222%	38	0.012%	306517	99.778%
Azul	873	0.284%	**873**	0.284%	0	0.000%	306327	99.716%
Amarillo	3553	1.157%	**3419**	1.113%	134	0.044%	303781	98.887%
Cian	3	0.001%	**3**	0.001%	0	0.000%	307197	99.999%
Magenta	195	0.063%	**157**	0.051%	38	0.012%	307043	99.949%

Tabla 8. Tabla de confusión de la clasificación de pixeles en una imagen de muestra de la competencia de RoboCup.

En la siguiente tabla se expone una comparación de los colores más importantes para reconocer los elementos de la cancha, que son porterías, postes laterales y pelota. Los colores segmentados son el azul, amarillo y naranja. Los demás se toman como fondo.

Se tomó una imagen ejemplo para esta comparación. La imagen fue tomada en el laboratorio de trabajo en condiciones de iluminación de tarde entre las 12 y 14 horas.

Color / Segmentación	Azul	Amarillo	Naranja	Fondo
Azul	26432	0	0	83
Amarillo	0	435	0	1
Naranja	0	22	69	1
Fondo	2796	65	30	280264

Tabla 9. Tabla de todos contra todos.

Color / Segmentación	Azul	Amarillo	Naranja	Fondo
Azul	99.7%	0.0%	0.0%	0.3%
Amarillo	0.0%	99.8%	0.0%	0.2%
Naranja	0.0%	23.9%	75.0%	1.1%
Fondo	1.0%	0.0%	0.0%	99.0%

Tabla 10. Tabla de porcentajes de todos contra todos.

Las tablas anteriores se leen de la siguiente forma.

Para el renglón Azul: Un pixel de un elemento azul en la imagen original, se clasificó como azul, amarillo, naranja o fondo.

Es decir, hubo 26432 elementos azules clasificados como azules, ningún elemento azul clasificado como amarillo, ningún elemento azul clasificado como naranja y 83 elementos azules clasificados como fondo.

Para la Tabla 10 la información se lee como porcentajes respecto al total de elementos azules presentes en la imagen original. Es decir, el 99.7% de los elementos azules fueron clasificados como azules.

A continuación se presenta la imagen original y segmentada de la cual se hizo la prueba.

Figura 28. Del lado izquierdo la imagen segmentada y del lado derecho la imagen original.

En la Figura anterior, se muestra el resultado de la clasificación de pixeles a las clases azul, amarilla y naranja. Se incluye negro como fondo o resto.

1.2 Reconocimiento de Objetos.

1.2.1 La pelota.

Para calcular el cierre convexo se utilizó la función de OpenCV `convexHull()` que nos permite obtener a través del algoritmo de Sklansky [25] el conjunto de puntos que conforman el cierre que tiene complejidad O(N log N) en la implementación actual. Para ver el código en C++ de la rutina ir al Anexo 4.

Utilizando la función de OpenCV `cvConvexityDefects()` podemos obtener la secuencia de defectos del cierre convexo, con el punto inicial y final de cada uno, su tamaño y así saber cual es el más grande y en base a eso determinar si el objeto que buscamos está frente a nosotros.

Por medio de este método la detección de la pelota es eficiente, rápida y robusta, pues así como se aprecia en la Figura 6 el cierre convexo en una imagen permite encontrar correctamente la pelota.

Ahora falta probarlo para un conjunto de cuadros o imágenes, es decir el video o entradas que está recibiendo el robot en tiempo real con movimiento de los objetos y ver cómo reacciona el algoritmo.

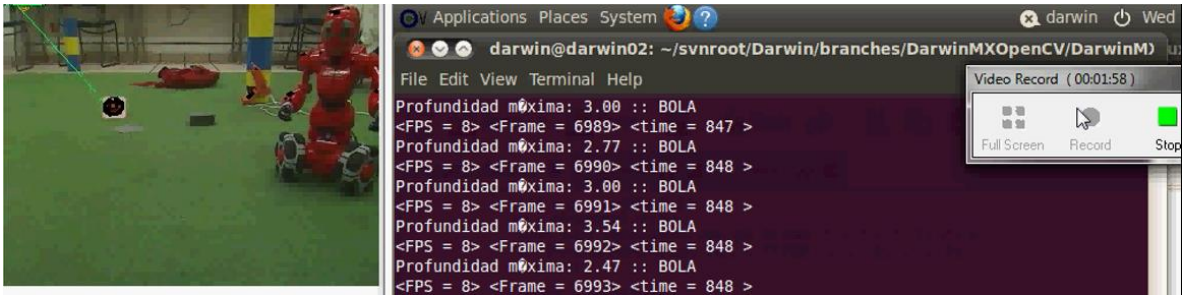


Figura 29. Detección con cubos y cierre convexo.

En la Figura 29 se muestra una captura de un video del algoritmo corriendo sobre la computadora del robot. La velocidad de procesamiento es de 8 cuadros por segundo, lo cual es rapidísimo considerando que está corriendo también la segmentación de 2 cubos y el calculando el cierre convexo. Prácticamente la bola está siempre siendo encontrada por nuestro algoritmo y rara vez es perdida. Las detecciones en imágenes muestra se pueden ver en el Anexo G. Resultados de detección de elementos en diferentes ambientes.

De hecho un problema común es que trabajando únicamente con colores, se pueda confundir algún elemento naranja, rojo o del tono de la piel humana como pelota. En la figura anterior se muestra que aun detectando algunos pixeles extraños a la bola, estos quedan fuera de consideración a tomar en cuenta solo aquellos que forman la pelota.

En la siguiente tabla se muestra el resultado de las detecciones de pelota en un conjunto de imágenes muestra de la competencia en las cuales se encontraba presente la pelota.

Pelota:	Naranja	Muestras:	9
Correctas:	9	100.0%	
Erróneas:	0	0.0%	

Tabla 11. Resultado de detección de pelota en conjunto muestra.

1.2.2 La portería.

Para la portería se optó por buscar el centroide de la imagen segmentada para el color amarillo y azul. A partir de esta segmentación y este centroide podemos saber si vemos portería al ubicar el centroide en base a los puntos del objeto en la imagen.

Para esto se toman los puntos extremos de la detección y se divide en regiones la zona detectada. Esto a fin de localizar el centro en alguna de esas regiones y como sabemos por experimentación la zonas donde se ubica el centroide de una portería podemos saber si estamos viendo una.

Sin embargo antes de hacer esto se trabaja un poco sobre la imagen, ya que esta puede contener ruido o pixeles basura que afecten la detección.

Se trabaja solo en un canal, ya sea azul o amarillo, de tal forma que se sepa que portería se está buscando. Después se aplica un algoritmo de detección de contornos a partir del cual se obtienen los distintos objetos en la imagen delimitados por estos y luego se calcula el área de cada contorno. El que posea el área más grande se asume que es la portería.

Una vez hecho esto, ahora se pueden tomar los puntos extremos mencionados antes, tomando como extremos los puntos más lejanos a la izquierda, derecha, arriba y abajo del contorno seleccionado.

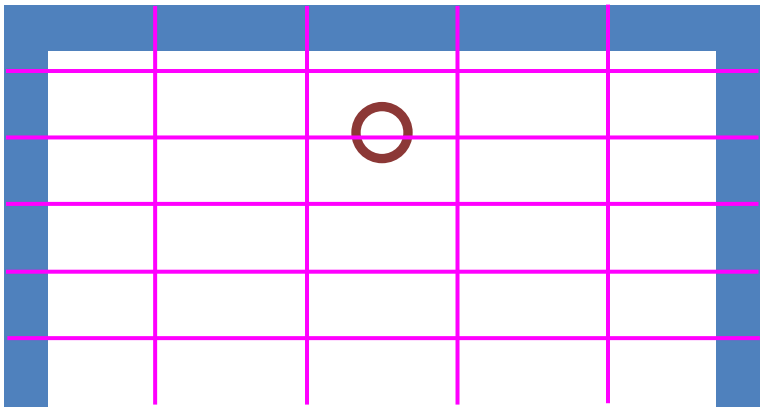


Figura 30. Portería dividida en regiones.

El centroide se ubica en una determinada región, la cual queda en nuestro trabajo delimitada en el eje X entre las posiciones 2*ancho/5 y 3*ancho/5. Mientras que en el eje Y está entre 2*alto/6 y 2*alto/5. Esto se puede ver mejor en la Figura 30.

El algoritmo para detectar la portería una vez segmentada la imagen queda de la siguiente manera.

1. *Seleccionar color a buscar (portería)*
2. *Obtener los contornos de la imagen para el color seleccionado.*
3. *Seleccionar el contorno con el área más grande.*
4. *Para determinar en donde está el primer poste (izquierda) buscar el primer pixel detectado de izquierda a derecha del contorno. Lo llamaremos col1.*
5. *Para determinar el segundo poste (derecha) buscamos el último pixel detectado de izquierda a derecha del contorno. Lo llamaremos col2.*
6. *Utilizando las coordenadas de col1 y col2 obtenemos el ancho del objeto detectado con* $\boldsymbol{col2 - col1}$.
7. *Buscar el primer pixel detectado de arriba hacia abajo del contorno. Lo llamaremos ren1.*
8. *Buscar el último pixel detectado de izquierda a derecha del contorno. Lo llamaremos ren2.*
9. *Utilizando las coordenadas de ren1 y ren2 obtenemos el alto del objeto detectado con* $\boldsymbol{ren2 - ren1}$.
10. *Obtener el centroide del objeto con* $\boldsymbol{xc = \bar{x}}$ *y* $\boldsymbol{yc = \bar{y}}$. *Donde* $\bar{\boldsymbol{x}}$ *y* $\bar{\boldsymbol{y}}$ *son las medias de los puntos correspondientes al objeto seleccionado.*
11. *Si* $\boldsymbol{col1 + \frac{2}{5}ancho < xc < col2 - \frac{2}{5}ancho}$ *y* $\boldsymbol{ren1 + \frac{1}{6}alto < yc < ren2 - \frac{2}{5}alto}$

    Entonces encontramos la portería.

Este algoritmo se emplea para cada clase de portería (una por cada color), es decir para la portería amarilla y para la portería azul ya que son distintas. Es importante diferenciarlas, puesto que ¡No deseamos meter un autogol!

El código de este proceso está en el Anexo D. Códigos C++.

En las pruebas realizadas se aprecia que la detección con este método de la portería fue correcta. En el ejemplo que sigue, vemos la segmentación de una portería amarilla. Al programa se le pide primero que detecte si es una portería azul y de forma correcta nos dice que no está viendo tal objeto. Esto se aprecia visualmente con un círculo rojo en la imagen.

Posteriormente pedimos que detecte si es una portería amarilla y mediante dos círculos blancos en la imagen puestos en el centroide del objeto el programa indica que efectivamente está viendo la portería amarilla.

En las pruebas que se hicieron sobre varias imágenes de porterías los resultados fueron bastante positivos, pues la detección resultó bastante eficiente en los casos donde las porterías se encontraban incluso parcialmente ocluidas.

En las siguientes tablas se exponen los resultados de estas detecciones, separadas por portería de cada color y el número de detecciones correctas e incorrectas, posteriormente una tabla con los resultados generales.

En la Figura 31 se muestra que en la detección correcta de una portería azul no existente, aunque hay elementos falsos positivos de color azul, no son tomados en cuenta para considerarse como elemento portería.

Aunque pareciera que el porcentaje de detección de porterías es bajo (66.7%), esto es un muy buen indicio para tener una correcta clasificación de este objeto, pues como veremos más adelante, solo ocurrió error cuando la portería no estaba presente en su totalidad. Es decir, si la portería es visible por completo, es detectable.

Portería:	**Amarilla**	**Muestras:**	7
Correctas:	5	71.4%	
Erróneas:	2	28.6%	

Portería:	**Azul**	**Muestras:**	5
Correctas:	3	60.0%	
Erróneas:	2	40.0%	

Portería:	**Ambas**	**Muestras:**	12
Correctas:	8	66.7%	
Erróneas:	4	33.3%	

Tabla 12. Resultados detección de porterías centroide.

En la Tabla 12 se exponen los resultados de evaluar un conjunto de imágenes de la competencia en las cuales estaba presente alguna de las porterías para ser detectada. Primero están los resultados de detección para porterías amarillas, luego azules y al último el resultado de ambas.

El problema surgió en las 4 imágenes erróneamente detectadas (2 porterías azules y 2 amarillas) dado que en las imágenes no se apreciaba desde el inicio el objeto completo como tal, por lo que esto resultaba en otra figura la cual no correspondía con una portería, y por lo tanto, su centroide no concordaba con el de estas.

Este problema se ejemplifica en las dos imágenes de la Figura 32. En cada una se aprecia la oclusión de una parte de las porterías.

Es en estos casos cuando hacemos la validación descrita en la sección de teoría. Lo que haremos será tomar dos nuevos objetos basados en la configuración de la Figura 33.

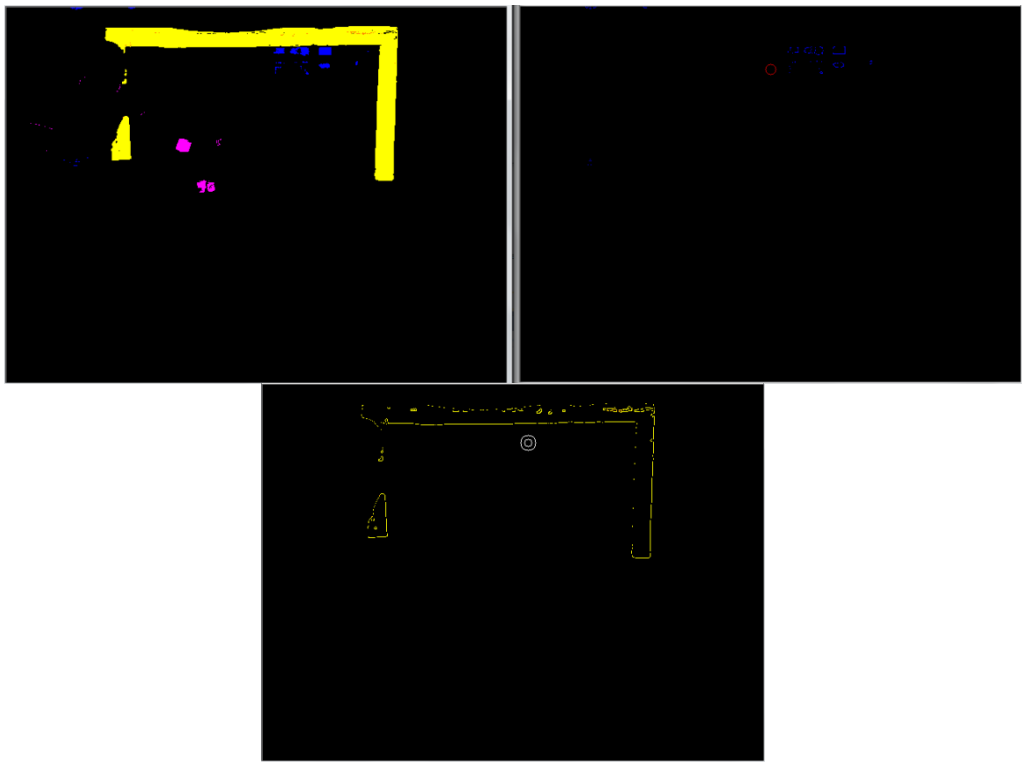


Figura 31. Detección de portería amarilla por centroide.[9]

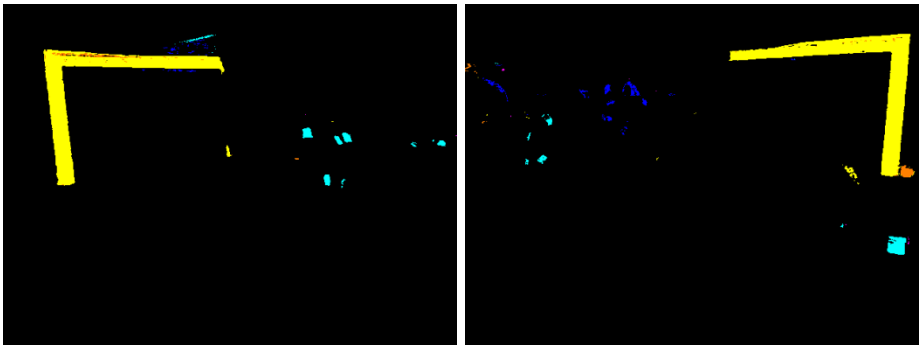


Figura 32. Segmentaciones donde no se ve la portería completa.

Básicamente la idea es generar nuevos rangos cuando se detecte un objeto. Si no se pudo detectar la portería con el método anterior pero hay un objeto azul entonces pasamos a hacer una validación con el mismo algoritmo variando los parámetros de búsqueda del centroide.

Esto queda así, para el punto 8 del algoritmo en el caso de que se vea la parte derecha de la portería:

[9] Para la detección de la portería se usa un detector contornos [16], ya que así podemos obtener la información más importante de la imagen y elimina elementos innecesarios.

- $Si\ \boldsymbol{col1 + \frac{3}{5}ancho < xc < col2 - \frac{1}{5}ancho}\ y\ \boldsymbol{ren1 + \frac{1}{6}alto < yc < ren2 - \frac{3}{5}alto}$

O para la parte izquierda de la portería:

- $Si\ \boldsymbol{col1 + \frac{1}{5}ancho < xc < col2 - \frac{3}{5}ancho}\ y\ \boldsymbol{ren1 + \frac{1}{6}alto < yc < ren2 - \frac{3}{5}alto}$

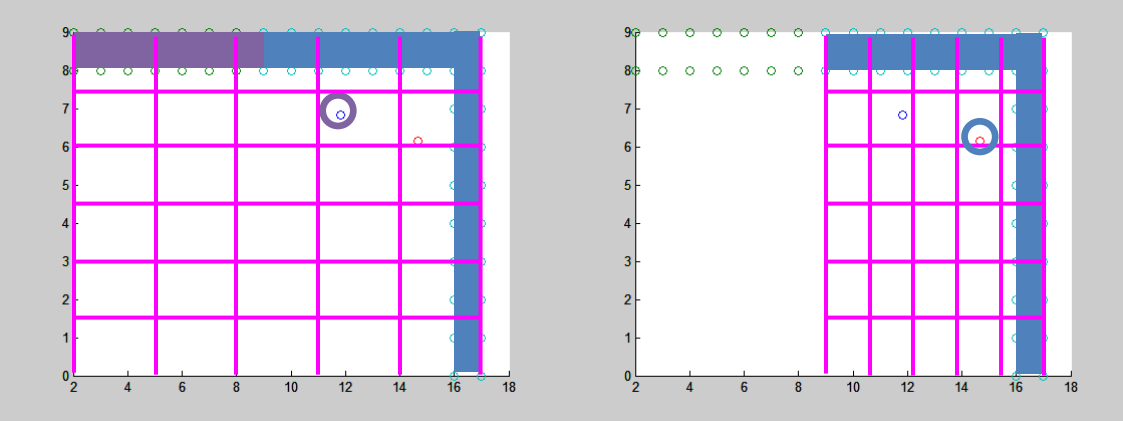


Figura 33. Regiones propuestas para portería obstruida.

En la Figura 33se aprecian dos imágenes de la portería obstruida. Esto es para tomar en cuenta que no siempre la misma región o rango estará fuera de vista.

Sin embargo podemos apreciar que el centroide para ambos casos está dentro de un rango similar, por lo cual podemos generalizar un poco la posición de dicho punto respecto a la detección del objeto azul o amarillo.

Una vez aplicadas estas reglas, los resultados de detección de la portería mejoran considerablemente, pues ahora la detección cubre los casos donde no podemos ver por completo el objeto y de esta forma podemos considerar una portería incompleta para usarla en la toma de decisiones.

Portería:	**Amarilla**	**Muestras:**	**7**
Correctas:	7	100.0%	
Erróneas:	0	0.0%	

Portería:	**Azul**	**Muestras:**	**5**
Correctas:	5	100.0%	
Erróneas:	0	0.0%	

Portería:	**Ambas**	**Muestras:**	**12**
Correctas:	12	**100.0%**	
Erróneas:	0	0.0%	

Tabla 13. Resultados de detección de porterías corrección de centroides.

En la tabla anterior vemos el número de detecciones hechas con el arreglo del algoritmo.

La detección de las porterías mejora considerablemente como se muestra en la tabla anterior, pues la detección funciona ahora correctamente en todos los casos.

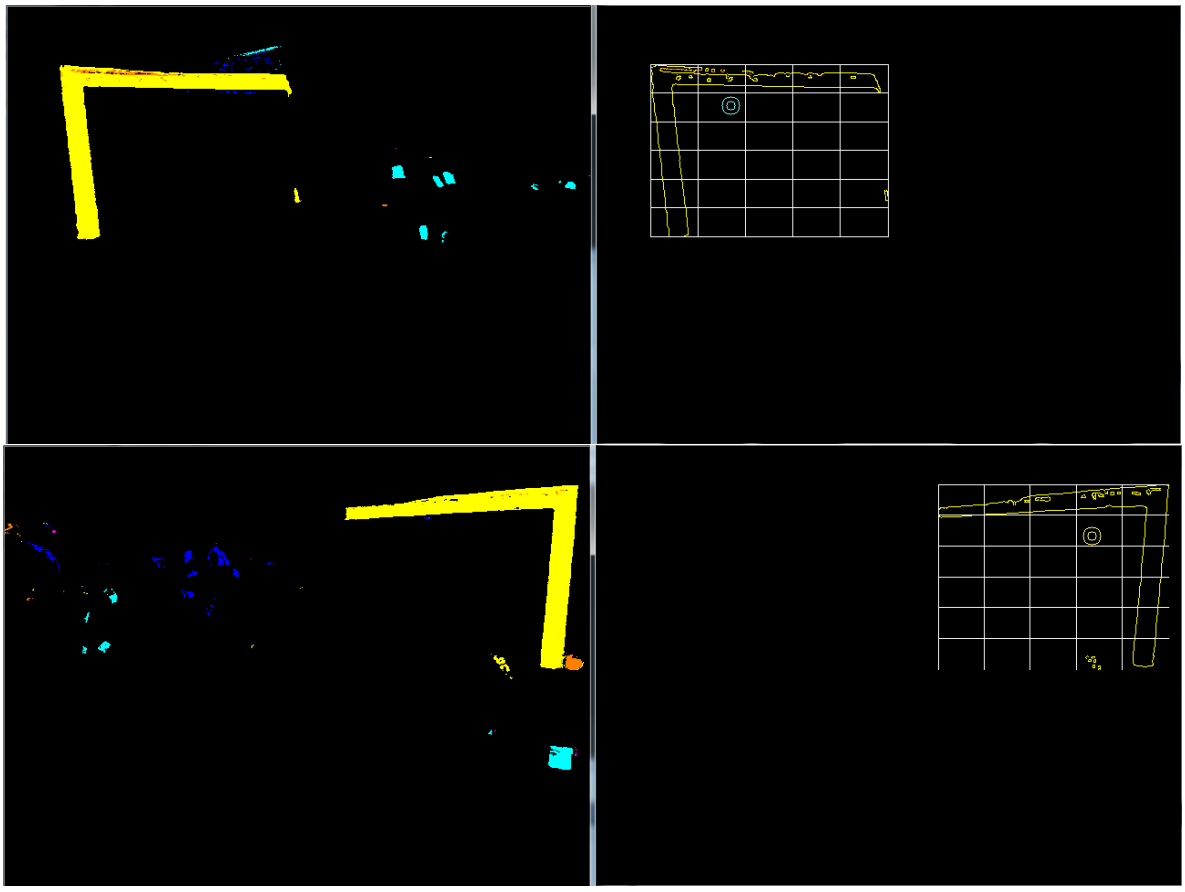


Figura 34. Detección de porterías incompletas

En la Figura 34 podemos ver los casos de ejemplo donde no se podía detectar la portería debido a que esta no se encontraba presente en su totalidad en la imagen. Ahora con las reglas agregadas al algoritmo, aún con las porterías parcialmente ocluidas el programa detecta correctamente su presencia.

Más aún, como sabemos que regla de discriminación estamos usando, podemos saber qué lado de la portería estamos viendo, por lo que podríamos generar caminado lateral o giros de acuerdo a esta posición.

1.2.3 Los postes laterales o landmarks.

Para los postes laterales se descartó el método de los 5 puntos pues si la imagen tiene obstrucciones en el poste, simplemente no se puede detectar y se busca que el programa sea lo más robusto posible, es decir, que pueda detectar los objetos en condiciones reales de juego, donde hay varios obstáculos que pueden obstruir la visión del robot.

Para la detección se optó por el método descrito por la Ecuación 29. Lo que se busca en esta ecuación es ver si dos objetos amarillos (o azules) están arriba y debajo de un objeto azul (o amarillo), además verifica que estos estén alineados verticalmente, de tal forma que efectivamente sea un poste.

Para esto el procedimiento fue primero detectar los "blobs" o áreas que había en la imagen segmentada y de estos sacar únicamente los amarillos y azules.

Para extraer los blobs el método que se emplea es el que se describió en la sección de Blobs. de la teoría.

Una vez detectados estos procedemos a aplicar la fórmula para saber si estamos viendo o no un poste lateral. A continuación se ve parte del código usado, para ver todo dirigirse a la sección de códigos al final.

```
// Go through all the detections (blue and yellow)
for(int p=0; p < blueKeyPoints.size() + yellowKeyPoints.size(); p++)
{
// We have a blue blob
if( p < blueKeyPoints.size())
{
	px = (int)blueKeyPoints[p].pt.x;
	py = (int)blueKeyPoints[p].pt.y;

	/// Search for other BLUE blobs
	// We only look on the points we have not already
	for(int q=p; q<blueKeyPoints.size(); q++)
	{
		qx = (int)blueKeyPoints[q].pt.x;
		qy = (int)blueKeyPoints[q].pt.y;

		/// Now search for a YELLOW blob
		for(int r=0; r< yellowKeyPoints.size(); r++)
		{
			rx = (int)yellowKeyPoints[r].pt.x;
			ry = (int)yellowKeyPoints[r].pt.y;

			/// Apply hsia formula
			// The blue is between yellows
			if( (py < ry && ry < qy) || (qy < ry && ry < py) )
				//Mod: See if yellow is aligned with The centers of blues
				if(abs(rx - qx) < BETA && abs(rx-px) < BETA)
				{
					// We found a landmark! Blue landmark!
					circle(resultImage, yellowKeyPoints[r].pt, (py - ry),
						Scalar(255,0,0), 2, 8);
				putText(resultImage, "BLUE LANDMARK", yellowKeyPoints[r].pt,
					FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0,255,0), 2,8);
					detection = 0;
					break;
				}
		}
	}
}
```

A través de este método la detección de los landmarks es bastante eficiente y robusta, pues su tiempo de procesamiento es rápido ya que hace una búsqueda de los puntos centrales de los "blobs" localizados y sólo toma en cuenta aquellos que sean amarillos y azules, evitando además buscar en aquellos puntos que se está visitando ya y repetir innecesariamente un punto.

El código toma un punto azul o amarillo y para cada otro punto del mismo color que no sea sí mismo realiza una comparación con los puntos correspondientes a un blob del color opuesto.

Una vez tomados tres puntos, se verifica que sean distintos, y que el color repetido, es decir dos amarillos o dos azules, estén uno arriba y otro abajo del color singular, un amarillo o un azul, que es la configuración de los landmarks de la cancha. Ya que esta condición no es suficiente, verificamos que los blobs que estén detectados se encuentren en una zona vertical para asegurar que estén sobre el mismo eje Y. Para esto se agregó una constante BETA que sirva de tolerancia para la posición de los puntos. La forma en que trabaja es medir la posición X de cada una de las partes del poste y luego asegurarnos que la distancia entre estas posiciones se encuentre dentro de la tolerancia establecida. Esta tolerancia se obtuvo a través de pruebas.

Este trabajo es muy eficiente y robusto. Del conjunto de imágenes muestra que se tienen en todas aquellas donde existe algún poste lateral, éste fue detectado correctamente. Es decir en el 100% de los casos la detección fue correcta.

Más aún, en aquellas imágenes donde no hay poste lateral, como comprobación, el algoritmo no detectó ningún poste, es decir no hubo detecciones falsas.

Un ejemplo de una detección está en la siguiente imagen. Más pruebas se pueden ver en el Anexo F. Localización de "landmarks".

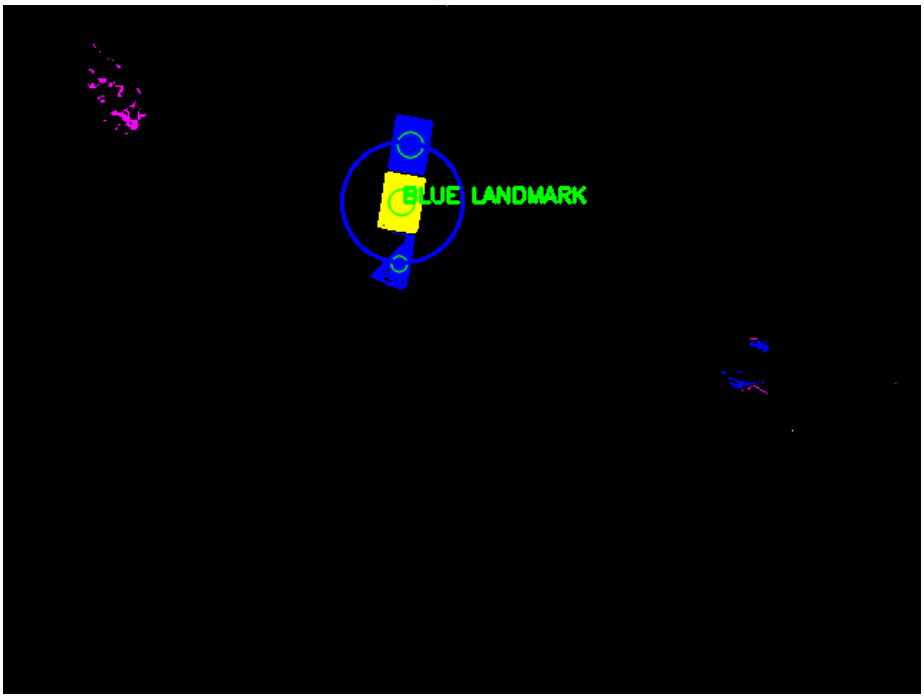


Figura 35. Landmark o poste lateral localizado.

En la imagen anterior, se muestra como a pesar de que el poste está ligeramente obstruido, la detección se hace de manera correcta.

Postes laterales:	**Ambos**	**Muestras:**	**8**
Correctas:	8	**100.0%**	
Erróneas:	0	0.0%	

Tabla 14. Porcentajes de detecciones de postes laterales sobre 8 imágenes del torneo.

En la Tabla 14 se muestra el resultado de detección de postes laterales sobre el conjunto de imágenes muestra de la competencia RoboCup 2012. En total se evaluaron 8 imágenes correspondientes a la competencia en las cuales existía un poste que detectar.

La limitante de este procedimiento es sin duda cuando el poste está demasiado lejos de la cámara, por lo que en la imagen el tamaño de los objetos azules y amarillos es muy pequeño para ser catalogado como tales.

Sin embargo este problema se presenta cuando el poste se encuentra a más de 8 metros de distancia, lo cual no es posible que pase durante el juego. Por esto el método planteado es suficiente para resolver nuestro problema de encontrar el landmark.

Para evitar errores debido a la segmentación, se usó un filtro morfológico de erosión, con el cual quitamos ruido de la imagen y además permite que los objetos o blobs estén mejor definidos y si puedan clasificar mejor.

Con esto tenemos todos los pasos para poder localizar los objetos dentro de la imagen si es que existen. Lo siguiente es implantar esto para el robot DarwinOP.

En las pruebas hechas sobre el conjunto de imágenes muestra se probó todo el procedimiento de segmentación y detección de objetos, con lo cual se obtuvieron los resultados siguientes.

Objeto	# de imágenes con objeto presente	# de detecciones correctas (Verdaderos Positivos)	% VP	# Verdaderos negativos	% VN	# Falsos positivos	% FP
Postes laterales	**23**	19	**82.6%**	21	**91.3%**	2	8.7%
Porterías	**35**	33	**94.3%**	29	**82.9%**	6	17.1%
Pelota	**43**	42	**97.7%**	38	**88.4%**	5	11.6%

Tabla 15. Resultado de detecciones sobre imágenes muestra

En la Tabla 15 se muestra una tabla de confusión, donde para cada objeto a detectar (postes, porterías y pelota) se da a conocer el número de imágenes en las cuales existen dichos objetos. Luego se muestra el número de detecciones hechas de manera efectiva (# de detecciones correctas) y el porcentaje de estas. Después se pone la columna de verdaderos negativos, donde nos indica en cuantas imágenes se espera que el objeto no sea detectado y sucede así. Por último la columna de Falsos positivos enseña el número de elementos detectados como poste, portería o pelota en las imágenes, pero que no debía de haber sido detectados como tales.

Muchos de los objetos detectados de forma errónea se debe a que en el laboratorio donde se hicieron las pruebas, existían elementos de color azul similares al azul de los postes y porterías, por lo que al tomar la imágenes y realizar las segmentación estos resultaron adentro del rango aceptable para ser tomados en cuenta, lo cual provocó ruido en la detección. Aun así, se muestra en la tabla anterior que el algoritmo de detección tuvo un porcentaje de efectividad al obtener los elementos presentes bastante alto. Incluso en el momento de la competencia, las pruebas realizadas fueron mejores ya que los resultados mostraron más efectividad al tener un porcentaje mayor de elementos detectados correctamente.

En el "Anexo G. Resultados de detección de elementos en diferentes ambientes." se muestra más detalle de los resultados.

1.3 Comentario de la implantación sobre el robot DarwinOP.

Después de tener lo métodos de procesamiento de la imagen el siguiente paso es implantar estos dentro del robot.

Para esto se usó la plataforma de desarrollo abierta que viene incluida dentro del framework de Darwin. Desde el código creado para detectar los objetos, se pasan como parámetros las posiciones de los objetos vistos en la imagen en forma de coordenadas (x, y).

Con las coordenadas de la imagen, se calcula ahora el pan y tilt de la cabeza del robot. Estos son los ángulos de giro arriba abajo y de un lado al otro. Con esta información el robot puede girar o caminar dependiendo de la situación.

Como el robot tiene integrada una tarjeta controladora para los motores se modificaron los códigos fuente de las librerías de tal forma que los códigos funcionaran correctamente. Además de agregarle movimientos de cuerpo y cabeza dependiendo de la situación.

Notas sobre las limitaciones.

En este punto es importante volver a remarcar porque se eligieron las técnicas usadas en este trabajo. Para esto hay que hablar primero de las limitaciones existentes en el hardware del robot. Las características de la computadora integrada a este ya fueron detalladas anteriormente en las especificaciones, pero a simple vista quizás no le digan mucho al lector.

Para ponerlo más claro, el robot posee una configuración parecida a la de una mini laptop de esas características, gracias a lo cual se le puede poner un Sistema Operativo (en este caso, Linux). Esto no se podía hacer con versiones anteriores de los robots. Bajo este S.O. se pueden usar códigos más extensos e incluir más bibliotecas de trabajo con funciones útiles para procesar las imágenes, también gracias a la memoria más amplia que se tiene.

Pero esto no significa que el poder de cómputo del robot sea el ideal. Una computadora con más memoria (8 GB de RAM) y mejor procesador (Intel core i5) tarda hasta 30 segundos en procesar completamente una imagen y detectar todos los objetos que se encuentran en ella usando un clasificador muy robusto, como lo es Bayes. Entonces hay que recordar que nuestro robot está jugando futbol, requiere tomar decisiones en tiempo real y recibir datos de manera continua mientras se procesan. Por esto no podemos darnos el lujo de usar los algoritmos más robustos planteados en cada capítulo del trabajo, pues el tiempo que lleva ejecutarlo es demasiado y una necesidad principal del software es que procese los cuadros tan rápido como sea posible y así el robot pueda tomar decisiones respecto al juego.

Al ser la visión de un jugador un proceso crítico en el sistema, es forzoso elegir métodos que sean rápidos, pero que además nos entreguen resultados tan confiables como sea posible. Para saber si un resultado es confiable, tenemos que realizar pruebas, medir tiempos y calificar detecciones hechas por el código. Estas pruebas se presentaron en los experimentos descritos en la tesis y se justifica el porqué de su elección para ser implantados sobre el Darwin.

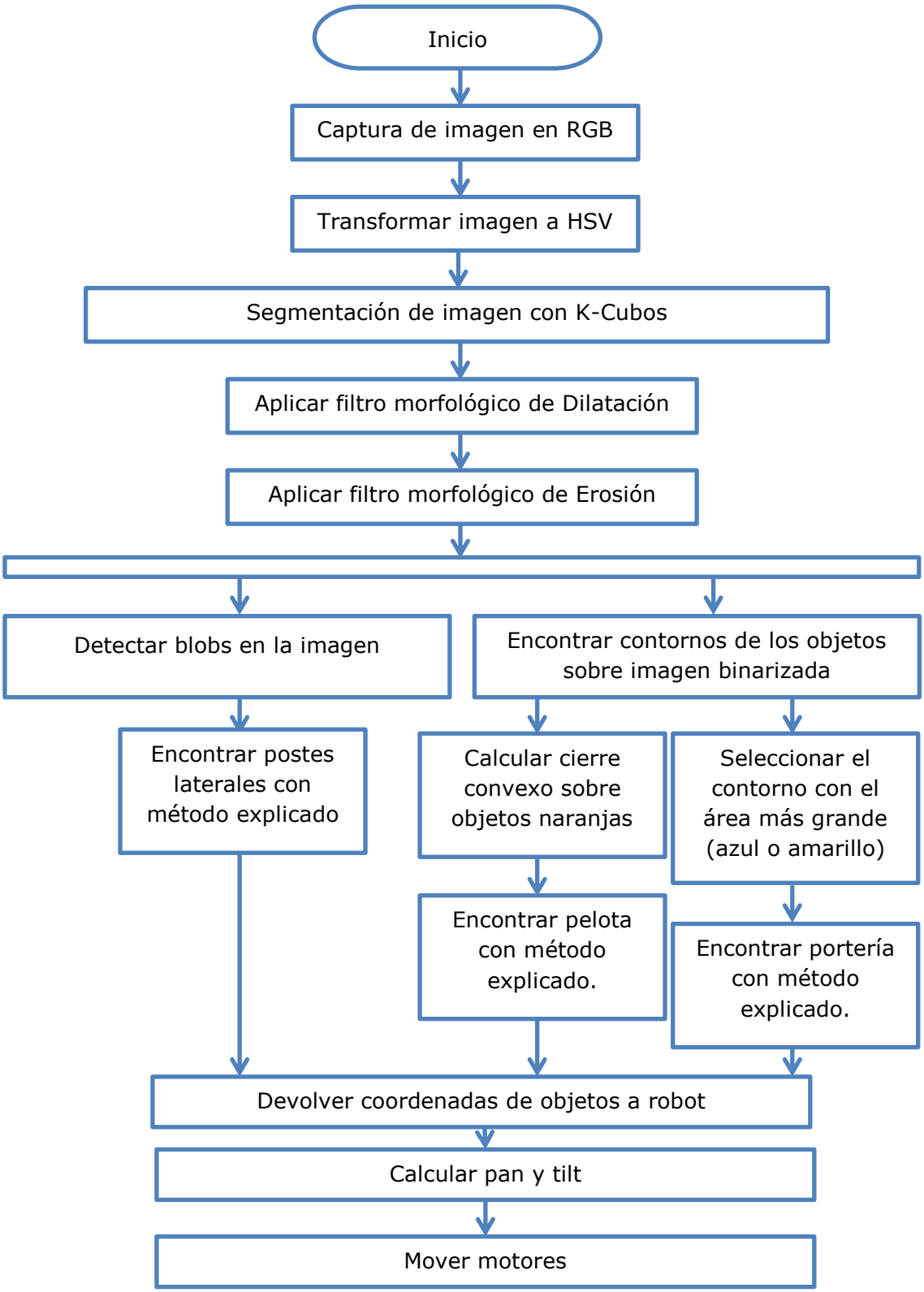
Conforme se vayan mejorando las partes electrónicas del robot, se podrán implantar métodos que requieran más memoria o tiempo para ser procesados pero que su eficiencia sea mejor para encontrar las cosas en la visión.

Esto no significa que los métodos elegidos sean malos, por el contrario como ya se describió, estos son muy buenos y bastante rápidos, pero sin duda pueden mejorar.

Ya que tenemos todo armado a continuación se presenta un diagrama del proceso de ejecución del algoritmo completo presentado en este trabajo.

1.4 Diagrama del Procesamiento de la información.

A continuación se presenta un diagrama del procesamiento que tiene un cuadro para detectar los objetos en él.



Capítulo 5. Conclusiones.

En este trabajo se exploraron varias formas de detectar objetos dentro de un cuadro recibido desde la cámara del robot. Después de probar varios de estos algoritmos se desecharon varios debido a tiempos de ejecución muy lentos en su mayoría. Un problema presente en este trabajo es la limitación en cuanto recursos de procesamiento y memoria, pues el DarwinOP posee un procesador de una computadora pequeña y memoria interna que sirve para apenas almacenar el sistema operativo del robot y los programas que en él corren.

Después de varias pruebas se pudo verificar la confiabilidad de los algoritmos planteados al comprobar que el número de detecciones correctas es alto. Además los tiempos de ejecución son aceptables tomando en cuenta los recursos ya mencionados.

Hay métodos que son bastante confiables en cuanto resultados, como el clasificador Bayesiano. Pero su problema es la lentitud con que se ejecutan, pues un cuadro necesitaba hasta 30 segundos para procesarse, lo cual para términos de la aplicación, es inaceptable ya que se deben tomar decisiones en tiempo real.

Por esto se deben encontrar formas de agilizar el procesamiento de los cuadros, ya que mientras más rápido se tome una decisión más rápido podemos hacer reaccionar al robot y en un juego de futbol, ser rápido es importante.

Aunque también es importante hacer las cosas bien. Por esto no se puede confiar la visión sólo a una segmentación de colores y ubicar los objetos de acuerdo a sus valores en RGB o HSV por los problemas que esto presenta. Un cambio de iluminación o ruido en la imagen, gente que se mete a la cancha o está en el público, luces ambientales, etcétera, son factores que hacen de la visión basada en colores un método poco efectivo y nada seguro para detectar objetos. Hacer la detección de esta forma es muy rápido pero poco fiable.

Es por esto que se deben usar otros algoritmos, de tal forma que podamos incrementar la confiabilidad del método sin sacrificar demasiado la velocidad.

Después de revisar ideas de la gente involucrada en este campo, se eligieron algunas que se pudieran modificar o adaptar a las necesidades y capacidades del robot. Además de generar algoritmos propios que nos permitieran efectuar el procesamiento de la visión.

Después de mucho trabajo se llegó a un punto en el cual el algoritmo de visión es robusto y eficiente.

Parte de este algoritmo se puso a concursar en la competencia de FIRA 2011 en Taiwán, donde fue la primera participación de un equipo mexicano en este evento. También se puso a prueba en el Torneo Mexicano de Robótica 2012, donde se obtuvo

el 3er lugar de la competencia (y esto debido a una falla técnica del robot). Por último se llevó la prueba más importante en el torneo de RoboCup 2012 donde se compitió a la par de los mejores equipos del mundo, y aunque no se logró un lugar, el robot y trabajo del equipo dotMX fueron motivos de halagos de colaboradores.

Trabajo a futuro.

Aunque este trabajo es la conclusión de un ciclo de trabajo, la idea del equipo dotMX sigue en pie. Y en colaboración con las instituciones compañeras y los amigos de trabajo se pretende que el trabajo siga avanzando.

Conforme vaya cambiando la estructura de la competencia de RoboCup también lo harán las estrategias, algoritmos, hardware y limitaciones que se tienen, por lo que los programas se deben ir adaptando para igualar y mejorar a los competidores, pero sobre todo hacer una contribución al trabajo comunitario de mucha gente involucrada en este campo.

Las ideas a plantear a continuación son las siguientes sin que sean la únicas a ejecutar o sean limitantes:

- Disminuir la dependencia de segmentación por colores para ubicar objetos.
- Localizar al robot dentro de la cancha usando elementos distintivos de un campo de futbol, es decir, quitar elementos como los postes laterales en la estrategia de juego y que las porterías no se detecten por su color azul o amarillo.
- Igualar las capacidades de visión del humano al integrar una segunda cámara al robot de tal forma que nos permita tener una percepción estereoscópica del entorno. De esta manera se puede identificar la profundidad del objeto con otros algoritmos y tomar decisiones distintas que mejoren el juego.

Para finalizar queda el gusto de un trabajo que trajo resultados al equipo, que contribuyó al desarrollo de la técnica de juego del robot. Se crearon algoritmos que permitieron una buena visión para el robot, que mientras son eficientes también son rápidos, las dos características necesarias para una aplicación como esta.

Bibliografía

[1] RoboCup, «RoboCup Page,» 2012. [En línea]. Available: http://www.robocup.org/about-robocup/. [Último acceso: 2012].

[2] Robcup, «RoboCup Soccer Humanoid League Rules and Setup For the 2012 Competition in Mexico City,» 2012. [En línea]. Available: http://www.tzi.de/humanoid/. [Último acceso: 2012].

[3] B. E. Bayer, «Color imagin array». US Patente 3971065, 1976.

[4] R. Jean, «Demosaicing with the Bayer Pattern,» [En línea]. Available: http://www.unc.edu/~rjean/demosaicing/demosaicing.pdf. [Último acceso: 2012].

[5] R. G. Kuehni, Color space and its divisions: color order from antiquity to the present, John Wiley and Sons, 2003.

[6] L. E. Sucar y G. Gómez, Introduction to Computer Vision, INAOE, 2008.

[7] A. Ford y A. Roberts, Colour Space Conversions, University of Westminster, 1998.

[8] J. L. Chávez, Tratamiento digital de imágenes multiespectrales, UNAM, 2011.

[9] M. M. Deza y E. Deza, Encyclopedia of Distances, Springer, 2009.

[10] P. R. B., «Ejemplo práctico de función de segmentación de color utilizando la distancia de Mahalanobis,» 2008. [En línea]. Available: http://www2.elo.utfsm.cl/~elo328/PDI14_EjemploMahalanobis.pdf. [Último acceso: 2012].

[11] A. a. S. Vassilvitskii, «k-means++: the advantages of careful seeding,» de *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007.

[12] J. M. I. Zannatha, R. C. Limón, A. D. G. Sánchez, E. H. Castillo, L. E. F. Medina y F. J. K. L. Leyva, «Monocular visual self-localization for humanoid soccer robots,» de *21st International Conference on Electrical Communications and Computers (CONIELECOMP)*, 2011.

[13] F. A. Cosio, Análisis de imágenes médicas, 2008.

[14] T. H. Cormen, Introduction to Algorithms, Segunda ed., The MIT Press, 2001.

[15] C. Manresa, J. Varona, R. Mas y F. J. Perales, «Real –Time Hand Tracking and Gesture Recognition for Human-Computer Interaction,» *Electronic Letters on Computer Vision and Image Analysis,* vol. 0, nº 0, pp. 1-7, 2000.

[16] S. Susuki y K. Abe, «Topological Structural Analysis of Digitized Binary Images by Border Following,» *COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING,* nº 30, pp. 32-46, 1985.

[17] J. M. Cañas, D. Puig, E. Perdices y T. González, «Visual Goal Detection for the RoboCup Standard Platform League,» de *X WORKSHOP DE AGENTES FÍSICOS*, Cáceres, 2009.

[18] J. M. Cañas, E. Perdices, T. González y D. Puig, «Recognition of Standard Platform RoboCup Goals,» *JOURNAL OF PHYSICAL AGENTS,* vol. 4, nº 1, 2010.

[19] G. R. Garzón, «El concepto estadístico de centro de gravedad,» *Números Revista Didáctica de las Matemáticas,* vol. 53, pp. 43-55, Marzo 2003.

[20] J. Kilian, «Simple Image Analysis By Moments Version 0.2,» 2001.

[21] C. H. Hsia, J. S. Chiang y S. H. Chang, «Adaptative Vision-Based Self-Localization System For Humanoid Robot Of Robocup,» *International Journal of Innovative Computing, Information and Control,* vol. 9, nº 3, pp. 991-1012, 2013.

[22] C.-H. Hsia, W.-H. Chang y J.-S. Chiang, «A Real-Time Object Recognition System Using Adaptive Resolution Method for Humanoid Robot Vision Development,» *Journal of Applied Science and Engineering,* vol. 15, nº 2, pp. 187-196, 2012.

[23] opencvDevTeam, «OpenCV v2.4.3 documentation,» 2012. [En línea]. Available: docs.opencv.org. [Último acceso: 2012].

[24] G. Bradsky y A. Kaehler, Learning OpenCV, O'REILLY, 2008.

[25] J. Sklansky, «Finding the Convex Hull of a Simple Polygon,» *Pattern Recognition Letters,* vol. 1, nº 2, pp. 79-83, Diciembre 1982.

[26] B. Kisačanin, S. S. Bhattacharya y C. Sek, Embedded Computer Vision, Springer, 2009.

[27] N. Salvaggio, L. Stroebel y R. D. Zakia, Basic Photographic Materials and Processes, Focal Press, 2008.

[28] M. Marghany y M. Hashim, «Comparison between Mahalanobis classification and neural network for oil spill detection using RADARSAT-1 SAR data,» *International Journal of the Physical Sciences,* vol. 6(3), pp. 566-576, 4 February 2011.

[29] S. S. Skiena, The algorithm design manual, Segunda ed., Springer, 2008.

[30] C. Manresa, J. Varona, R. Mas y F. J. Perales, «Real-Time Hand Tracking and Gesture Recognition for Human-Computer Interaction,» *Electronic Letters on Computer Vision and Image Analysis,* vol. 0(0), pp. 1-7, 2000.

[31] R. Álvarez, E. Millán, R. S. Oropeza y A. A. López, «Color Image Classification through Fitting of Implicit Surfaces,» *Lecture Notes in Computer Science,* vol. 3315, 2004.

[32] I. The MathWorks, «Función de Matlab 3D scatter plot: scatter3,» [En línea]. Available: http://www.mathworks.com/help/matlab/ref/scatter3.html. [Último acceso: 27 Noviembre 2012].

[33] I. The MathWorks, «Función de Matlab Load data from file: importdata,» [En línea]. Available: http://www.mathworks.com/help/matlab/ref/importdata.html. [Último acceso: 28 Noviembre 2012].

[34] S. F. Ramonde, Color image processing problems in digital photography, 2011.

[35] J. S. Cuenca, RECONOCIMIENTO DE OBJETOS POR DESCRIPTORES DE FORMA, Universitat de Barcelona, 2008.

[36] M. G. Borroto, Y. V. Rey, M. Á. M. Pérez y J. R. S. Juliett Martínez López, Selección y construcción de objetos para el mejoramiento de un clasificador supervisado: un análisis crítico., La Habana: CENATAV, 2008.

[37] P. Heinemann, F. Sehnke, F. Streichert y A. Zell, «Towards a Calibration-Free Robot: The ACT Algorithm for Automatic Online Color Training,» *RoboCup 2006,* p. 363–370, 2007.

Anexo A. Índice de figuras y tablas.

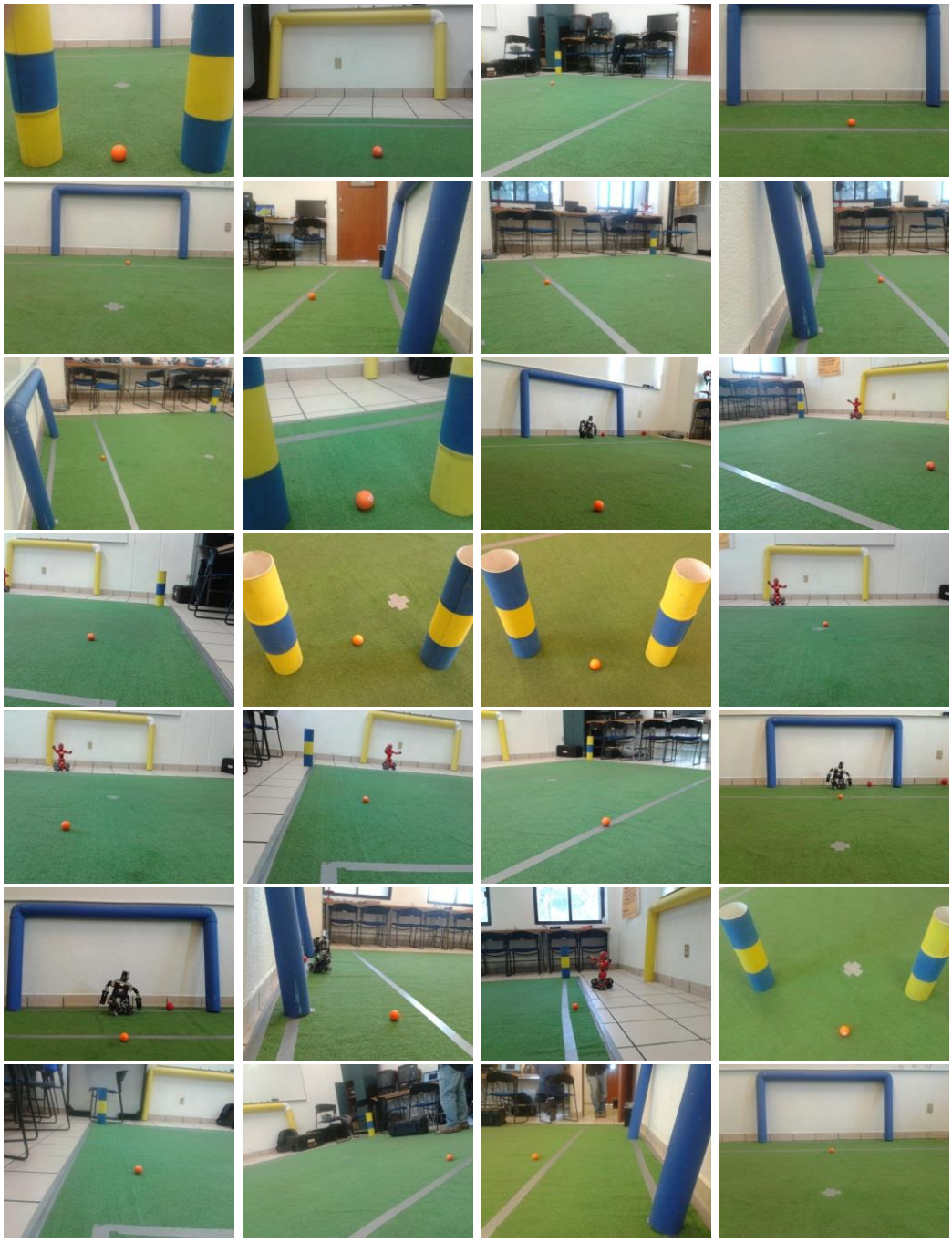
Anexo B. Código MATLAB para graficado de pixeles en RGB y HSV.

```
function A = lectorMuestrasBGR_HSV()

% This function reads a file with 12 headers:
% H1 S1 V1 B1 G1 R1 H2 S2 V2 B2 G2 R2
% These are the HSV and BGR values from a selection of the same area in 2
% images.
%
% Abner Quiroz Clemente 2012

filename = 'C:\Users\Abner\Documents\Visual Studio
2010\Projects\RecogeMuestrasRBG_HSV\RecogeMuestrasRBG_HSV\muestrasBGRHSV.
dat';
delimiterIn = ' ';
headerlinesIn = 1;
A = importdata(filename,delimiterIn,headerlinesIn);

% Create a 3D scatter plots using the scatter3 function

figure;
%       R               G               B
colors=[A.data(:,6), A.data(:,5), A.data(:,4)];
colors = colors./255;
%           X           Y           Z           size color  type
scatter3(A.data(:,4), A.data(:,5), A.data(:,6), 30, colors, 'filled')
view(-25, 30);
% Add title and axis labels
title('First image BGR data');
xlabel('Blue','Color','b');
ylabel('Green','Color','g');
zlabel('Red','Color','r');
% Set axis limits of 0 to 255
xlim([0 255])
ylim([0 255])
zlim([0 255])

figure;
scatter3(A.data(:,1), A.data(:,2), A.data(:,3), 30, colors, 'filled')
view(-25, 30);
title('First image HSV data');
xlabel('Hue','Color','m');
ylabel('Saturation','Color','k');
zlabel('Value','Color',[.8 .8 .8]);
% Set axis limits of HSV according to OpenCV values
xlim([0 180])
ylim([0 255])
zlim([0 255])

figure;
```

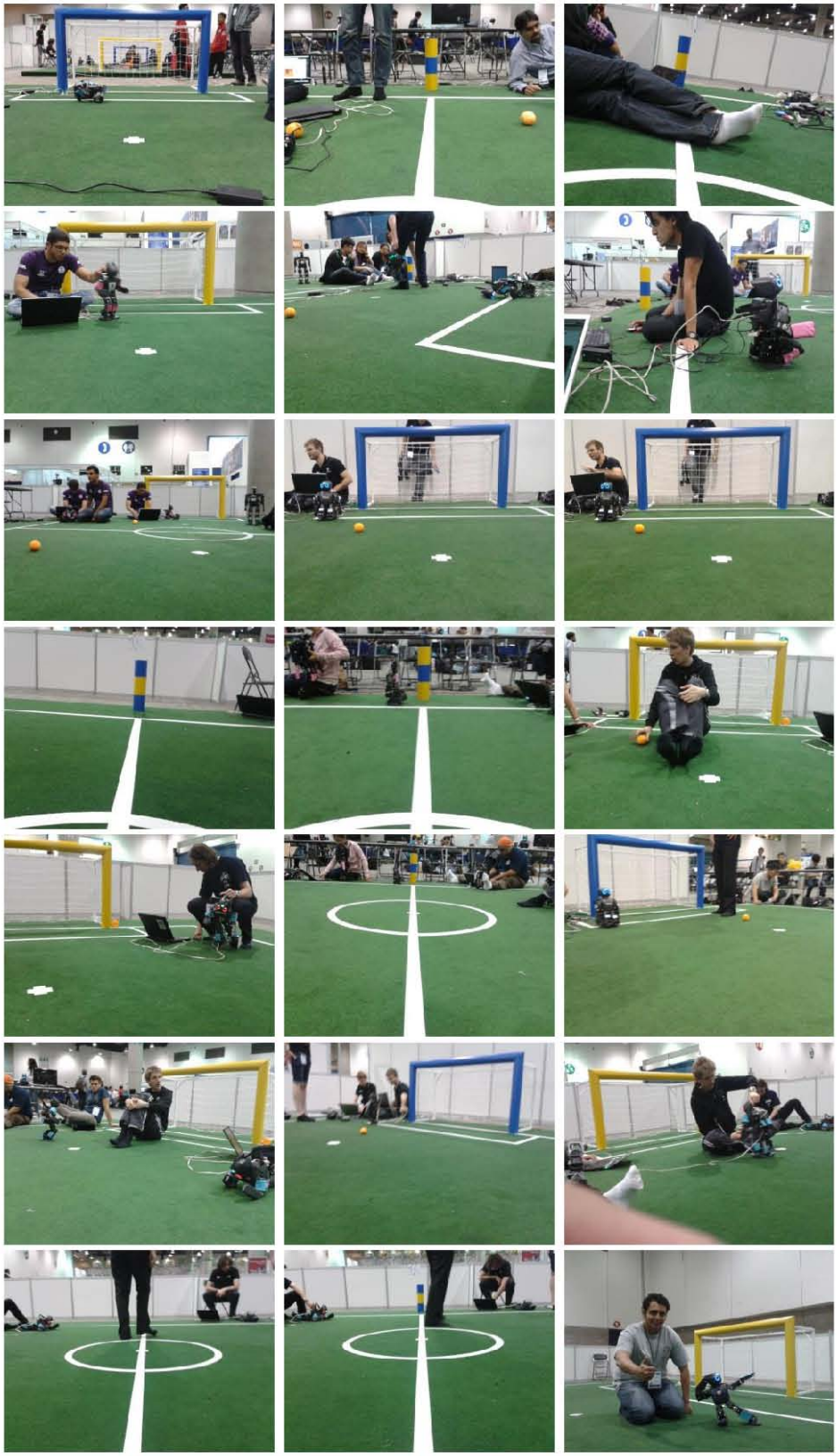
```
colors=[A.data(:,12), A.data(:,11), A.data(:,10)];
colors = colors./255;
scatter3(A.data(:,10), A.data(:,11), A.data(:,12), 30, colors, 'filled')
view(-25, 30);
title('Second image BGR data');
xlabel('Blue','Color','b');
ylabel('Green','Color','g');
zlabel('Red','Color','r');
xlim([0 255])
ylim([0 255])
zlim([0 255])

figure;
scatter3(A.data(:,7), A.data(:,8), A.data(:,9), 30, colors, 'filled')
view(-25, 30);
title('Second image HSV data');
xlabel('Hue','Color','m');
ylabel('Saturation','Color','k');
zlabel('Value','Color',[.8 .8 .8]);
xlim([0 180])
ylim([0 255])
zlim([0 255])
```

Anexo C. Imágenes de muestra.

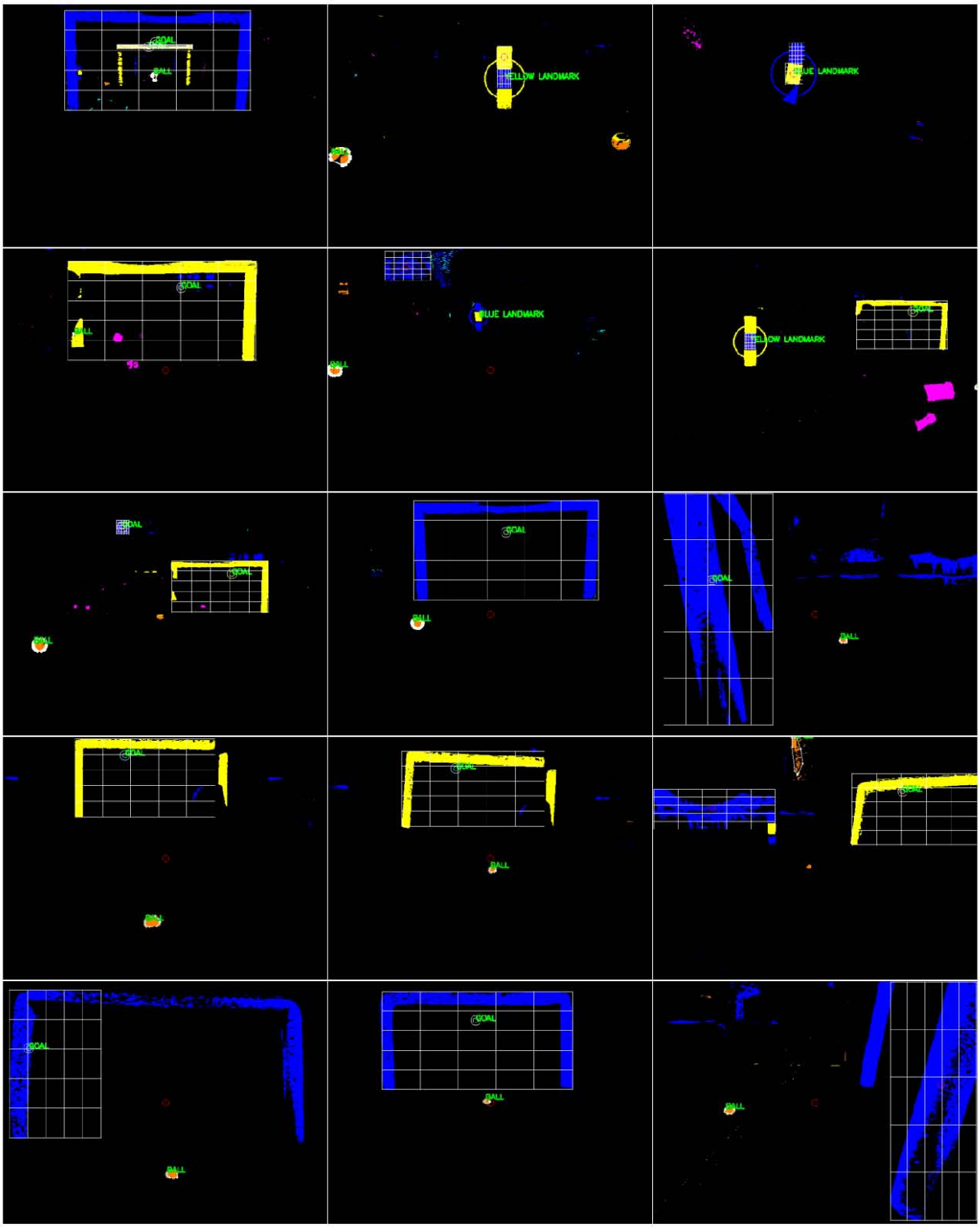
En el laboratorio de pruebas con variaciones de luz.

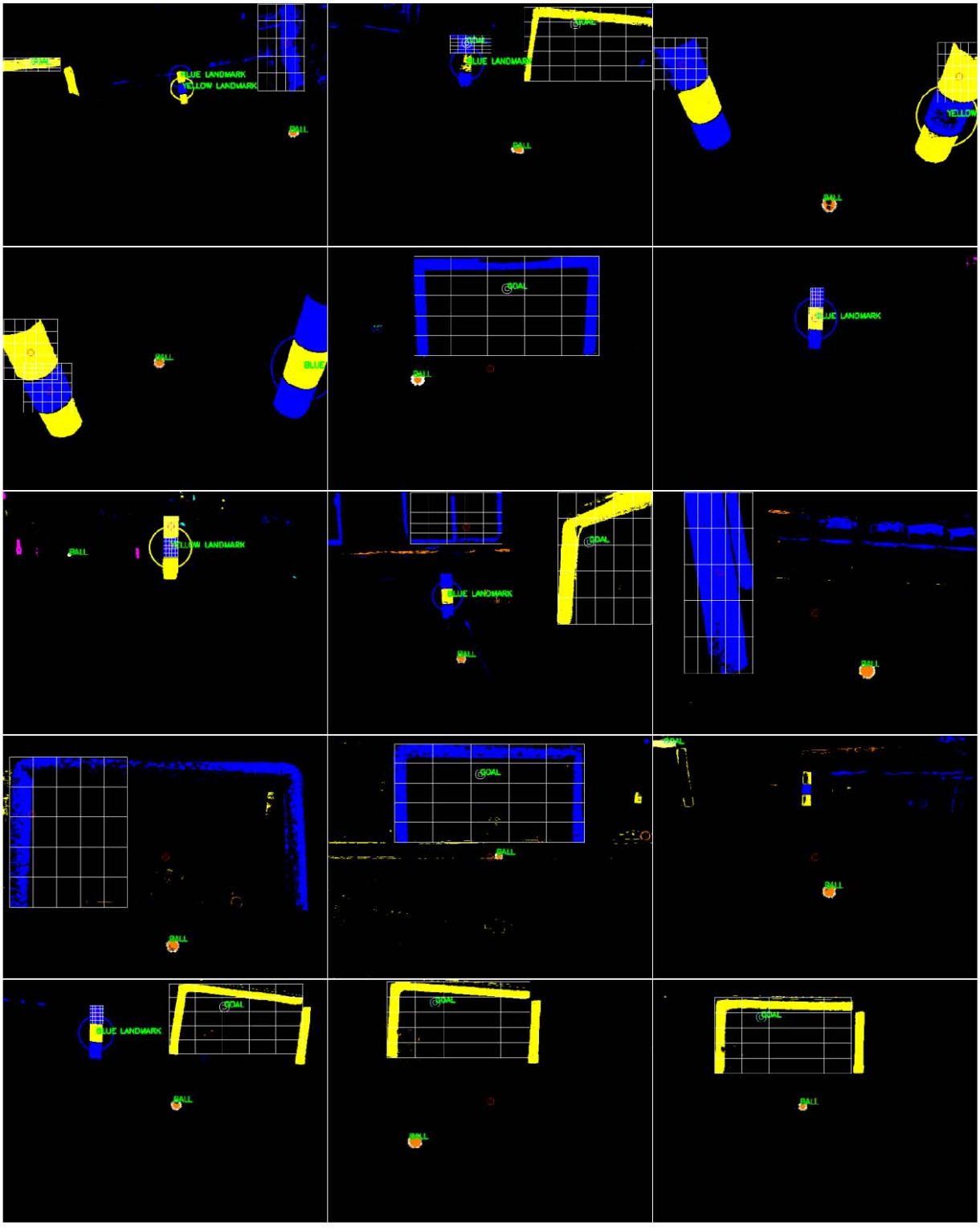


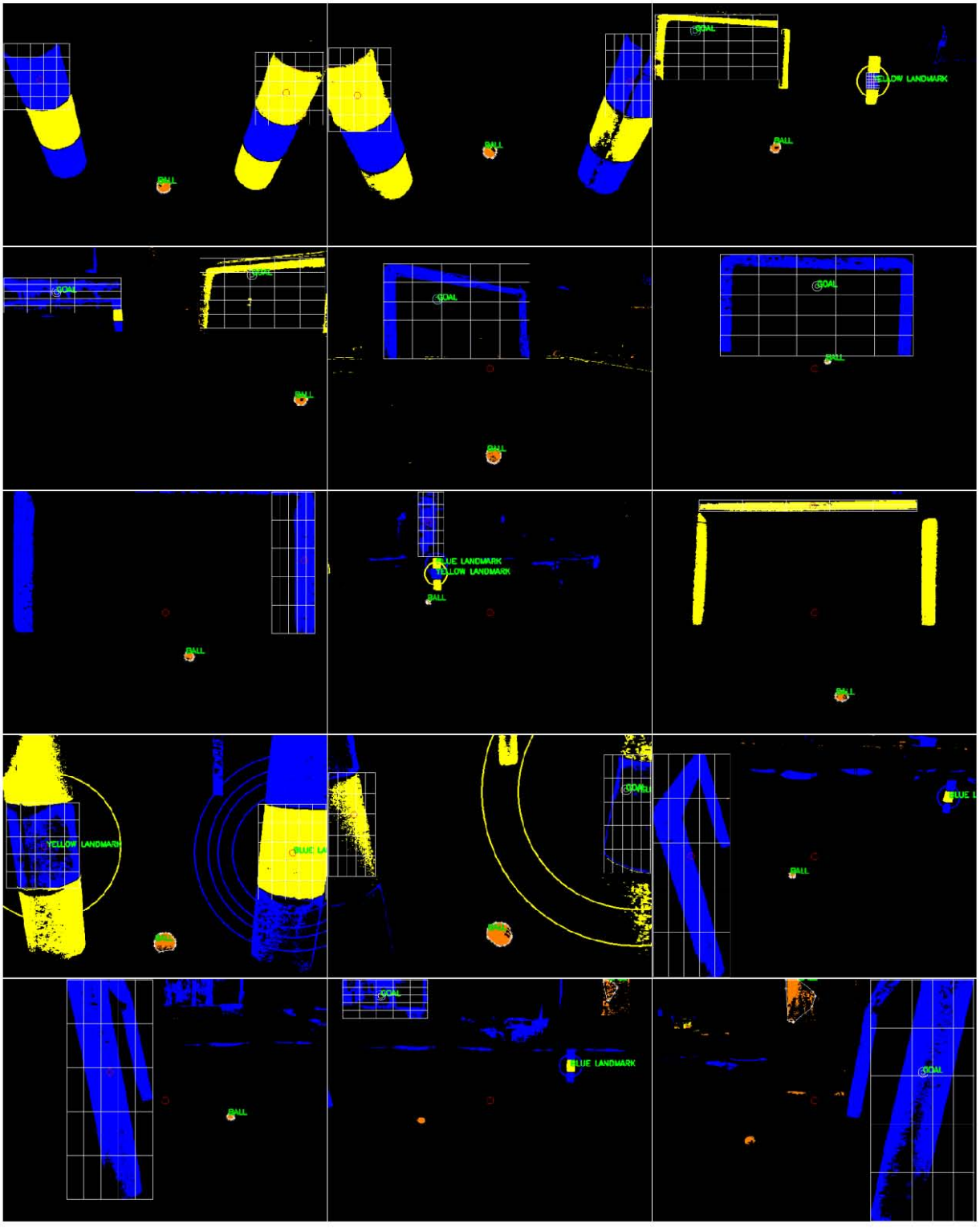
En competencia de RoboCup 2012.



Ejecución de los algoritmos para detectar objetos. Se muestran a continuación imágenes segmentadas con la detección de objetos en ellas.




GOAL
BLUE LANDMARK
YELLOW LANDMARK
BALL


BALL
GOAL
YELLOW LANDMARK
BLUE LANDMARK

Anexo D. Códigos C++

Rutinas de distancias.

```
/* -- métodos de segmentación por esferas o distancias -- */
```

Euclideana.

```
/**
Segmentates image using Euclidian distance.
*/
void euclidiana()
{
 Scalar dist = 0.0;

 for( int yy = 0; yy < video_segmentacion.rows; yy++ )
  for( int xx = 0; xx < video_segmentacion.cols; xx++ )
  {
        dist = pow((float)pt_medio[0]-video_segmentacion.at<Vec3b>(yy,xx)[0], 2)+
            pow((float)pt_medio[1]-video_segmentacion.at<Vec3b>(yy,xx)[1], 2)+
            pow((float)pt_medio[2]-video_segmentacion.at<Vec3b>(yy,xx)[2], 2);

        sqrt(dist,dist);
// -- pinta solo los pixeles que estén dentro del rango, lo demás lo pone en negro
        if(dist[0]<=(float)tol)
        {
         video_pruebas.at<Vec3b>(yy,xx) = video_segmentacion.at<Vec3b>(yy,xx);
         //circle(video_pruebas, Point(xx,yy), 2, CV_RGB( 200, 20, 20 ), 1, 8, 0);
         //cout<< "dist[]"+(int)dist[0];
        }
        else if(dist[0] > (float)tol)
         video_pruebas.at<Vec3b>(yy,xx) = Vec3b(0,0,0);
    }
}
```

Manhattan.

```
/**
Segmentates image using Manhattan distance.
*/
void manhattan()
{
 Scalar dist = 0.0;
 Vec3b temp;

 for( int yy = 0; yy < video_segmentacion.rows; yy++ )
  for( int xx = 0; xx < video_segmentacion.cols; xx++ )
  {
        dist =  abs(pt_medio[0]-video_segmentacion.at<Vec3b>(yy,xx)[0])+
                abs(pt_medio[1]-video_segmentacion.at<Vec3b>(yy,xx)[1])+
                abs(pt_medio[2]-video_segmentacion.at<Vec3b>(yy,xx)[2]);

// -- pinta solo los pixeles que estén dentro del rango, lo demás lo pone en negro
        if(dist[0]<=tol)
                video_pruebas.at<Vec3b>(yy,xx) = video_segmentacion.at<Vec3b>(yy,xx);
        else if(dist[0] > tol)
                video_pruebas.at<Vec3b>(yy,xx) = Vec3b(0,0,0);
  }
}
```

Mahalanobis.

```
/**
Segmentates image using Mahalobis distance.
*/
```

```
void  mahalanobis() //TODO
{
 Scalar medias;
 Mat matTemp;
 double bt = 0.0, gt= 0, rt = 0;

 /* -- media de los vectores -- */
 /*mediab = mean(blue);mediag = mean(green);mediar = mean(red);*/
 medias = mean(video_segmentacion.t());

 matTemp = Mat(3,3,CV_64F);

 /* -- matriz de covarianza -- */
 for (int i = 0; i < 3 ; i++)
  for (int j = 0; j < 3; j++)
  {
        for (int c = 0; c < 3; c++) //
        {
         //bt += (clicmedia[i][c] - medias[i]) * (clicmedia[j][c] - medias[j]) / 3;
        }
        matTemp.at<double>(i,j) = matTemp.at<double>(j,i) = bt;
        bt = 0.0;
  }
 //calcCovarMatrix(&video_segmentacion,3,matTemp,vecTemp,CV_COVAR_NORMAL,-1);
 //calcCovarMatrix(clicmedia,matTemp,medias,CV_COVAR_USE_AVG);

 /* -- invierte la matriz -- */
 matTemp = matTemp.inv();

 /* -- obtiene la distancia -- */
 /*             temp = mean(clicmedia);
 medias[1] = mean(clicmedia[1]);
 medias[2] = mean(clicmedia[2]);*/
 for( int yy = 0; yy < video_segmentacion.rows; yy++ )
  for( int xx = 0; xx < video_segmentacion.cols; xx++ )
  {
        //dist = Mahalanobis(video_segmentacion.at<Vec3b>(yy,xx),clicmedia, matTemp);
        /*bt = video_segmentacion.at<Vec3b>(yy,xx)[0] - clicmedia[0];
        gt = video_segmentacion.at<Vec3b>(yy,xx)[1] - clicmedia[1];
        rt = video_segmentacion.at<Vec3b>(yy,xx)[2] - clicmedia[2];*/

        //dist = Mahalanobis(video_segmentacion.at<Vec3b>(yy,xx),media, out);
                              //cout << "d m: " << dist[0] << endl;
  }
}
```

Clasificador de Bayes.

```
///Bayes classification
void Bayes()
{
        int max;
        int a;

        clasificar=true;

        for(a=0;a<NUMBER_OF_CLASSES;a++)
                y[a] = FLT_MIN;

        for(a=0; a<NUMBER_OF_CLASSES; a++)
        {
                if(nTrainingSamples[a] > 0)
                {
///Calculate covariance matrix and mean for every channel of training data
//We exclude the last one since is the rest, meaning everything that is not in the other classes
                        calcCovarMatrix(training_data[a], covar[a] ,meanMat[a] , CV_COVAR_NORMAL |
                        CV_COVAR_ROWS | CV_COVAR_SCALE, CV_32FC1);
```

```
                ///Calculate covariance matrix eterminant
                det[a] = (float)determinant(covar[a]);

                ///Calculate inverse of covariance matrix
                invert(covar[a],invCovar[a],DECOMP_CHOLESKY);

                ///Calculate logarithm of covariance matrix determinant
                logCovar[a] = log(det[a]);
        }
        else
        {
        }
}

///Clean count of classified pixels
for(int r=0; r<NUMBER_OF_CLASSES; r++)
        npixelsclassified[r] = 0;

///Calculate Bayes discrimination function
for(int r=0; r<original.rows; r++)
        for(int c=0; c<original.cols; c++)
        {
                //Pixel extraction
        pixel.at<float>(0,0) = (float)hsvOriginal.at<Vec3b>(r,c)[0];    //B H
        pixel.at<float>(0,1) = (float)hsvOriginal.at<Vec3b>(r,c)[1];    //G S
        pixel.at<float>(0,2) = (float)hsvOriginal.at<Vec3b>(r,c)[2];    //R V

        for(a=0; a<NUMBER_OF_CLASSES; a++)
        {
                //Pixel - mean
                pix_mean = pixel - meanMat[a];
                //Mul with inverse covariance matrix and pix_mean transpose
                multemp = (pix_mean * invCovar[a] * pix_mean.t());

                //rest of bayes
                y[a] = (float)((multemp.at<float>(0,0) / -2.0) - logCovar[a]/2.0);
        }

        max = 0;
        ///Classification of pixel
        //Search biggest
        for(a=0; a<NUMBER_OF_CLASSES; a++)
        {
                if(y[max] < y[a])
                        max = a;
        }

        //check if epsilon is enough
        if(y[max] <= epsilon[max])
                max = NUMBER_OF_CLASSES+1;

        npixelsclassified[max]++;

        switch(max)
        {
                case 0: //ball - orange
                        seg_color = Scalar(5,127,255);
                        break;
                case 1: //blue
                        seg_color = Scalar(255,5,5);
                        break;
                case 2: //yellow
                        seg_color = Scalar(5,255,255);
                        break;
                case 3: //green
                        seg_color = Scalar(5,255,5);
                        break;
                case 4: //gray/white
                        seg_color = Scalar(250,250,250);
```

```
                        break;
                case 5: //CYAN
                        seg_color = Scalar(255,255,5);
                        break;
                case 6: //MAGENTA
                        seg_color = Scalar(255,5,255);
                        break;
                default:
                        seg_color = Scalar(5,5,5);
                        break;
            }
            segmentadaPintada.at<Vec3b>(r,c)[0] = (uchar)seg_color[0];
            segmentadaPintada.at<Vec3b>(r,c)[1] = (uchar)seg_color[1];
            segmentadaPintada.at<Vec3b>(r,c)[2] = (uchar)seg_color[2];
            }
        clasificar=false;
        status = "Bayes done";
}
```

Umbrales o cubo.

```
/**
Sets the minimum and maximum values or parameters for the k cube of the ball.
*/
void cubo()
{
 ///Convierte a HSV
 cvtColor(video_segmentacion, video_segmentacion, CV_BGR2HSV);
 cvtColor(video_pruebas, video_pruebas, CV_BGR2HSV);

 int bt,gt,rt;
 for( int yy = 0; yy < video_segmentacion.rows; yy++ )
  for( int xx = 0; xx < video_segmentacion.cols; xx++ )
  {
        bt = video_segmentacion.at<Vec3b>(yy,xx)[0];
        gt = video_segmentacion.at<Vec3b>(yy,xx)[1];
        rt = video_segmentacion.at<Vec3b>(yy,xx)[2];

        // comparamos si el pixel actual, está dentro de los rangos para cada canal
        if(bt<=ballMaxHSV[0] && bt>=ballMinHSV[0] &&
                gt<=ballMaxHSV[1] && gt>=ballMinHSV[1] &&
                rt<=ballMaxHSV[2] && rt>=ballMinHSV[2])
// -- pinta solo los pixeles que estén dentro del rango, lo demás lo pone en negro
         video_pruebas.at<Vec3b>(yy,xx) = video_segmentacion.at<Vec3b>(yy,xx);
        //else
         //video_pruebas.at<Vec3b>(yy,xx) = Vec3b(0,0,0);
  }

 /// Regresa a BGR
 cvtColor(video_segmentacion, video_segmentacion, CV_HSV2BGR);
 cvtColor(video_pruebas, video_pruebas, CV_HSV2BGR);
}
```

K-Cubos.

```
/**
Merges the data of each cube in one structure.
For each cube where exists data that defines a part of a element, this rutine will take every pixel
and put them into one matrix.
This will obtain the current ball position
*/
void processAllCubes()
{
 int bt,gt,rt;
 Mat t;
 int totalx;
 int totaly;
```

```
 int ball_pixels;

 /// Convierte a HSV
 cvtColor(video_segmentacion, video_segmentacion, CV_BGR2HSV);

 //video_pruebas.convertTo(video_pruebas,CV_8UC1);
 totalx = totaly = ball_pixels = 0;
 video_pruebas = Mat::zeros(video_segmentacion.size(),video_segmentacion.type());

 for( int yy = 0; yy < video_segmentacion.rows; yy++ )
  for( int xx = 0; xx < video_segmentacion.cols; xx++ )
  {
        bt = video_segmentacion.at<Vec3b>(yy,xx)[0];
        gt = video_segmentacion.at<Vec3b>(yy,xx)[1];
        rt = video_segmentacion.at<Vec3b>(yy,xx)[2];
        for ( int kk= 0; kk < kBall; kk++)
        {
/// comparamos si el pixel actual está dentro de los rangos para cada canal para cada cubo definido
         if(bt<=(int)ballMaxs.at(kk)[0] && bt>=(int)ballMins.at(kk)[0] &&
                gt<=(int)ballMaxs.at(kk)[1] && gt>=(int)ballMins.at(kk)[1] &&
                rt<=(int)ballMaxs.at(kk)[2] && rt>=(int)ballMins.at(kk)[2])
         {
          ball_pixels++;
          totalx += xx;
          totaly += yy;
// -- pinta solo los pixeles que estén dentro del rango, lo demás lo pone en negro
          video_pruebas.at<Vec3b>(yy,xx) = Vec3b(0,127,255);
         }
        }
  }
 ///Obtain the ball position
 totalx /= ball_pixels;
 totaly /= ball_pixels;

 ballPos.X = totalx;
 ballPos.Y = totaly;

 cvtColor(video_segmentacion, video_segmentacion, CV_HSV2BGR);
}
```

Filtro de Cerradura (Dilatación y Erosión)

```
/**  @function Closing
We perform dilation and erosion
*/
void Closing()
{
        int morph_type;

        if( morph_elem == 0 )
                morph_type = MORPH_RECT;
        else if( morph_elem == 1 )
                morph_type = MORPH_CROSS;
        else if( morph_elem == 2)
                morph_type = MORPH_ELLIPSE;

        Mat element = getStructuringElement( morph_type,
                        Size( 2*morph_size + 1, 2*morph_size+1 ),
                        Point( morph_size, morph_size ) );

        /// Apply the dilation operation
        dilate( segmentedImage, erodedImage, element );
        /// Apply the erosion operation
        erode( erodedImage, erodedImage, element );
}
```

Detección de contornos y cálculo de centro de masa.

```
/**
Uses gray image to generate a contourImage and contour vector
*/
void ContourDetection(Mat GrayOrigin)
{
        //init for goal
        col1 = 0;
        col2 = 0;
        row1 = 0;
        row2 = 0;
        goalArea = 0;
        massCenter = Point(0,0);

        //Extract the contours so that
        GrayOrigin.copyTo(contoursImage);

findContours( contoursImage, contours0, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE);

        if(contours0.size() > 0)
    {
                contours.resize(contours0.size());
                for( size_t k = 0; k < contours0.size(); k++ )
                        approxPolyDP(Mat(contours0[k]), contours[k], 3, true);

                // To find the contour with biggest area
                int area = 0;
                int indexOfBiggest = 0;

                for( int i = 0; i< contours.size(); i++ )
                {
                        Scalar color = Scalar( 255,i,255) ;
drawContours( contoursImage, contours, i, color, 2, 8, hierarchy, 0, Point() );

                        area = (int)contourArea(contours.at(i));
                        if(goalArea <  area)
                        {
                                goalArea = area;
                                indexOfBiggest = i;
                        }
                }

                // Search for the extreme points, up, down, left and right
                // And calculate mass center
                int up = contours[indexOfBiggest][0].y;
                int left = contours[indexOfBiggest][0].x;
                int right = contours[indexOfBiggest][0].x;
                int down = contours[indexOfBiggest][0].y;

                int s = contours[indexOfBiggest].size();
                for(int c=0; c < s; c++)
                {
right = contours[indexOfBiggest][c].x > right ? contours[indexOfBiggest][c].x : right;
up =    contours[indexOfBiggest][c].y < up ?    contours[indexOfBiggest][c].y : up;
left =  contours[indexOfBiggest][c].x < left ?  contours[indexOfBiggest][c].x : left;
down =  contours[indexOfBiggest][c].y > down ?  contours[indexOfBiggest][c].y : down;

                }

                col1 = left;
                col2 = right;
                row1 = up;
                row2 = down;

                int xc = 0;
                int yc = 0;
                int count=0;
                for(int y = row1; y<row2; y++)
```

```
                        for(int x = col1; x<col2; x++)
                        {
                                if((int)GrayOrigin.at<uchar>(y, x) != 0 )
                                {
                                        xc += x;
                                        yc += y;
                                        count++;
                                }
                        }

                if(count > 0)
                        massCenter = Point(xc/count, yc/count );
                else
                        massCenter = Point(0, 0 );
        }
}
```

Detección de portería por centroide.

```
/**Goal detection
 */
void GoalDetection(char colorOfGoal)
{
        /**
La idea es que el objeto más grande es la portería.
En vez de que ubiquemos el centroide respecto a todos los elementos de la imagen
lo haremos con el objeto más grande.
Habría que sacar los contornos de la imagen y luego el área de estos. Así idealmente
tendría el objeto más grande.
Luego como conozco los puntos del contorno, busco las coordenadas más lejanas. Es lo
mismo que el procedimiento de abajo, que ya se tiene, pero en base al contorno.
        */

        if(colorOfGoal == 'b')
        {
                blueBlobsImage.copyTo(gray);
                ContourDetection(blueBlobsImage);
        }
        else
        {
                yellowBlobsImage.copyTo(gray);
                ContourDetection(yellowBlobsImage);
        }

        int lon = abs(col2-col1);
        int height = abs(row2-row1);

        // Look for a big goal, not small object
        if(goalArea > 400)
        {
/** I am going to establish a limit to determine if the center is indicating a goal
To do this, the centroid must be for a goal in a certain position.
        For the horizontal must be between 2/5 and 3/5
        For the vertical between 1/6 and 2/5  //1/2 (3/6)
*/
                if(massCenter.x >= col1+2*(lon/5) && massCenter.x <= col2-2*(lon/5) &&
                massCenter.y >= row1+height/6 && massCenter.y <= row2-3*(height/5) )
                // It is the goal
                {
                circle(resultImage, massCenter, 5, Scalar(255,255,255), 1, 8, 0);
                circle(resultImage, massCenter, 10, Scalar(255,255,255), 1, 8, 0);
                putText(resultImage, "GOAL", massCenter,
                FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0,250,0), 2,8);
                }
                else /** Perhaps the goal is occluded and we are seeing the right side
                        For the horizontal must be between 3/5 and 4/5
                        For the vertical between 1/6 and 2/5   //1/2 (3/6) */
                if (massCenter.x >= col1+3*(lon/5) && massCenter.x <= col2-1*(lon/5) &&
```

```
                massCenter.y >= row1+height/6 && massCenter.y <= row2-3*height/5)
                {
                circle(resultImage, massCenter, 5, Scalar(127,255,255), 1, 8, 0);
                circle(resultImage, massCenter, 10, Scalar(127,255,255), 1, 8, 0);
                putText(resultImage, "GOAL", massCenter,
                        FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0,250,0), 2,8);
                }
                else /** Or maybe it's the left side
                                For the horizontal must be between 1/5 and 2/5
                                For the vertical between 1/6 and 2/5    //1/2 (3/6) */
                if (massCenter.x >= col1+(lon/5) && massCenter.x <= col2-3*(lon/5) &&
                massCenter.y >= row1+height/6 && massCenter.y <= row2-3*height/5)
                {
                circle(resultImage, massCenter, 5, Scalar(255,255,127), 1, 8, 0);
                circle(resultImage, massCenter, 10, Scalar(255,255,127), 1, 8, 0);
                putText(resultImage, "GOAL", massCenter,
                FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0,250,0), 2,8);                               }
                else // No goal present
                {
                        circle(resultImage, massCenter, 7, Scalar(0,0,250), 1, 8, 0);
                }

                drawLines(col1, col2, row1, row2, 5, 5);
        }
        else
circle(resultImage, Point(resultImage.cols/2,resultImage.rows/2), 7, Scalar(0,0,250), 1, 8, 0);
}
```

Detección de blobs usando OpenCV.

```
/**
 * @function Blobs Detection
 */
void BlobsDetection()
{
        inRange(erodedImage,Scalar(254,0,0), Scalar(255,0,0), blueBlobsImage);
        inRange(erodedImage,Scalar(0,254,254), Scalar(0,255,255), yellowBlobsImage);

        // Preparation for detection of blobs
        params.filterByCircularity = false;
        params.filterByInertia = false;
        params.filterByConvexity = false;
        params.filterByArea = true;
        params.filterByColor = true;
        //params.minDistBetweenBlobs=0;
        //params.minRepeatability = 0;
        params.blobColor = 255;

        params.minThreshold = (float)minThreshold;
        params.maxThreshold = (float)maxThreshold;
        params.thresholdStep = 4;

        params.minArea = (float)minArea;
        params.maxArea = (float)maxArea;

        /// Blob detector declaration
        SimpleBlobDetector blobDetector( params );
        blobDetector.create("SimpleBlob");

        /// Blobs detection
        blobDetector.detect( blueBlobsImage, blueKeyPoints );
        blobDetector.detect( yellowBlobsImage, yellowKeyPoints );

        drawKeypoints( original, blueKeyPoints, blueyellow, CV_RGB(5,200,5),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
        drawKeypoints( original, yellowKeyPoints, blueyellow, CV_RGB(5,200,5),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
```

```
}
```

Detección de landmarks o postes laterales.

```
//LANDMARK OPERATIONS
int landmarkSearch()
{
	int detection = -1;
	/// Will return -1 if no detection
	//	0 if blue landmark
	//	1 if yellow landmark

	int px, py;
	int qx, qy;
	int rx, ry;

	// Go through all the detections (blue and yellow)
	for(int p=0; p < blueKeyPoints.size() + yellowKeyPoints.size(); p++)
	{
		// We have a blue blob
		if( p < blueKeyPoints.size())
		{
			px = (int)blueKeyPoints[p].pt.x;
			py = (int)blueKeyPoints[p].pt.y;

			/// Search for other BLUE blobs
			// We only look on the points we have not already
			for(int q=p; q<blueKeyPoints.size(); q++)
			{
				qx = (int)blueKeyPoints[q].pt.x;
				qy = (int)blueKeyPoints[q].pt.y;

				/// Now search for a YELLOW blob
				for(int r=0; r< yellowKeyPoints.size(); r++)
				{
					rx = (int)yellowKeyPoints[r].pt.x;
					ry = (int)yellowKeyPoints[r].pt.y;

					/// Apply hsia formula
					// The blue is between yellows
					if( (py < ry && ry < qy) || (qy < ry && ry < py) )
					//Mod: See if yellow is aligned with The centers of blues
						if(abs(rx - qx) < BETA && abs(rx-px) < BETA)
						{
							// We found a landmark! Blue landmark!
					circle(resultImage, yellowKeyPoints[r].pt, (py - ry),
								Scalar(255,0,0), 2, 8);
					putText(resultImage, "BLUE LANDMARK", yellowKeyPoints[r].pt,
						FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0,255,0), 2,8);
							detection = 0;
							break;
						}
				}
			}
		}
		else // a yellow blob
		{
			px = (int)yellowKeyPoints[p - blueKeyPoints.size()].pt.x;
			py = (int)yellowKeyPoints[p - blueKeyPoints.size()].pt.y;

			/// Search for other yellow blobs
			// We only look on the points we have not already
			for(int q=(p-blueKeyPoints.size()); q< yellowKeyPoints.size(); q++)
			{
				qx = (int)yellowKeyPoints[q].pt.x;
				qy = (int)yellowKeyPoints[q].pt.y;

				/// Now search for a blue blob
				for(int r=0; r< blueKeyPoints.size(); r++)
```

```
                                {
                                        rx = (int)blueKeyPoints[r].pt.x;
                                        ry = (int)blueKeyPoints[r].pt.y;

                                        /// Apply hsia formula
                                        // The blue is between yellows
                                        if( (py < ry && ry < qy) || (qy < ry && ry < py) )
                                        //Mod: See if yellow is aligned with The centers of blues
                                                if(abs(rx - qx) < BETA && abs(rx-px) < BETA)
                                                {
                                                        // We found a landmark! Yellow landmark!
                                        circle(resultImage, blueKeyPoints[r].pt, (py - ry),
                                                                Scalar(0,255,255), 2, 8);
                                        putText(resultImage, "YELLOW LANDMARK", blueKeyPoints[r].pt,
                                                FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0,255,0), 2,8);
                                                        detection = 1;

                                                        break;
                                                }
                                }
                        }
                }

        }
        return detection;
}
```

Detección de pelota (Defectos del Cierre convexo)

```
/**
Calculates convex hull and convexity defects using the precalculated image contours.
*/
void FindBall()
{
        // Only orange pixels
        inRange(erodedImage,Scalar(0,127,254), Scalar(0,128,255), orangeImage);

        findContours( orangeImage, ballContours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE);

        if(ballContours.size() > 0)
    {
                // To find the contour with biggest area
                float maxArea= contourArea(ballContours.at(0));
                float area = 0;
                int indexOfBiggest = 0;

                for( int i = 0; i< ballContours.size(); i++ )
                {
                        area = contourArea(ballContours.at(i));
                        if(maxArea <  area)
                        {
                                maxArea = area;
                                indexOfBiggest = i;
                        }
                }

                ballContour = ballContours.at(indexOfBiggest);

                /// Now the Convex Hull
                vector<int> ballHullIdx;
                convexHull(Mat(ballContour), ballHull, true);
                convexHull(Mat(ballContour), ballHullIdx, true);

                /* -- Dibuja cierre convexo -- */

                Point pto = ballHull.at(ballHull.size()-1);
                Point pt;
```

```
                for(int i = 0; i < ballHull.size(); i++)
                {
                        pt = ballHull.at(i);
                        line( resultImage, pto, pt, CV_RGB( 250, 255, 250 ),1,8,0);
                        circle(resultImage, pto, 2, CV_RGB( 255, 255, 255 ), 1, 8, 0);
                        pto = pt;
                }

                if(ballHull.size() > 0)
                {
                        //defectos_cierre_convexo
                        convexityDefects(ballContour, ballHullIdx, convexityDefectsSet);
                        // access the 4 parameters that this function calculates
                        //(start, end, depth/defect-point and depth)
                        double maxDepth = 0.0;
                        double depth;
                        for (int cDefIt = 0; cDefIt < convexityDefectsSet.size(); cDefIt++)
                        {
                        depth = (double)convexityDefectsSet[cDefIt].val[3]/256.0f;  // see
documentation
                        maxDepth = max(depth, maxDepth);
                        }
                        cout << endl << "maxdepth: " << maxDepth;

                        // Check if we can see ball through its defects
                        // A big defect means we are not seeing something round
                        if(maxDepth < 10.0)
                        // WE FOUND A BALL!!
                        putText(
                        resultImage, "ball", pto, FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0,220,0), 2,8);
                }

        }
}
```

Anexo E. Tablas de confusión.

Muestras de competencia RoboCup.

Tamaño imagen:	307200

Imagen 1.

Clase	%	Pixeles clasificados	%	Positivos Verdaderos	%	Positivos Falsos	%	Negativos Verdaderos
Naranja	0.056%	172	0.056%	172	0.000%	0	99.944%	307028
Azul	4.822%	14813	4.822%	14812	0.000%	1	95.178%	292387
Amarillo	0.479%	1473	0.477%	1464	0.003%	9	99.521%	305727
Cian	0.020%	62	0.017%	51	0.004%	11	99.980%	307138
Magenta	0.016%	48	0.004%	13	0.011%	35	99.984%	307152

Imagen 2.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.451%	1386	0.448%	1375	0.004%	11	99.549%	305814
Azul	0.372%	1142	0.372%	1142	0.000%	0	99.628%	306058
Amarillo	0.996%	3060	0.830%	2549	0.166%	511	99.004%	304140
Cian	0.005%	16	0.005%	16	0.000%	0	99.995%	307184
Magenta	0.006%	19	0.000%	0	0.006%	19	99.994%	307181

Imagen 3.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.000%	1	0.000%	0	0.000%	1	100.000%	307199
Azul	0.570%	1750	0.570%	1750	0.000%	0	99.430%	305450
Amarillo	0.371%	1140	0.371%	1139	0.000%	1	99.629%	306060
Cian	0.000%	0	0.000%	0	0.000%	0	100.000%	307200
Magenta	0.085%	261	0.078%	240	0.007%	21	99.915%	306939

Imagen 4.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.020%	60	0.000%	0	0.020%	60	99.980%	307140
Azul	0.197%	606	0.197%	606	0.000%	0	99.803%	306594
Amarillo	4.385%	13471	4.385%	13471	0.000%	0	95.615%	293729

Cian	0.000%	0	0.000%	0	0.000%	0	100.000%	307200
Magenta	0.167%	512	0.159%	488	0.008%	24	99.833%	306688

Imagen 5.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.207%	637	0.207%	637	0.000%	0	99.793%	306563
Azul	1.006%	3089	1.006%	3089	0.000%	0	98.994%	304111
Amarillo	0.124%	380	0.095%	291	0.029%	89	99.876%	306820
Cian	0.088%	270	0.088%	270	0.000%	0	99.912%	306930
Magenta	0.009%	29	0.008%	25	0.001%	4	99.991%	307171

Imagen 6.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.021%	66	0.002%	5	0.020%	61	99.979%	307134
Azul	0.277%	852	0.277%	852	0.000%	0	99.723%	306348
Amarillo	1.487%	4568	1.472%	4521	0.015%	47	98.513%	302632
Cian	0.003%	9	0.003%	9	0.000%	0	99.997%	307191
Magenta	0.836%	2567	0.827%	2540	0.009%	27	99.164%	304633

Imagen 7.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.235%	721	0.222%	683	0.012%	38	99.765%	306479
Azul	0.284%	873	0.284%	873	0.000%	0	99.716%	306327
Amarillo	1.157%	3553	1.113%	3419	0.044%	134	98.843%	303647
Cian	0.001%	3	0.001%	3	0.000%	0	99.999%	307197
Magenta	0.063%	195	0.051%	157	0.012%	38	99.937%	307005

Imagen 8.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.136%	417	0.136%	417	0.000%	0	99.864%	306783
Azul	4.635%	14240	4.623%	14201	0.013%	39	95.365%	292960
Amarillo	0.059%	182	0.007%	22	0.052%	160	99.941%	307018
Cian	0.007%	23	0.005%	16	0.002%	7	99.993%	307177
Magenta	0.001%	2	0.000%	0	0.001%	2	99.999%	307198

Imagen 9.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.115%	353	0.115%	353	0.000%	0	99.885%	306847

Azul	4.619%	14190	4.609%	14160	0.010%	30	95.381%	293010
Amarillo	0.063%	192	0.007%	21	0.056%	171	99.938%	307008
Cian	0.008%	26	0.007%	20	0.002%	6	99.992%	307174
Magenta	0.002%	5	0.000%	0	0.002%	5	99.998%	307195

Imagen 10.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.000%	0	0.000%	0	0.000%	0	100.000%	307200
Azul	0.639%	1963	0.639%	1963	0.000%	0	99.361%	305237
Amarillo	0.399%	1226	0.399%	1226	0.000%	0	99.601%	305974
Cian	0.000%	0	0.000%	0	0.000%	0	100.000%	307200
Magenta	0.023%	70	0.000%	0	0.023%	70	99.977%	307130

Imagen 11.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.007%	21	0.000%	1	0.007%	20	99.993%	307179
Azul	0.432%	1326	0.432%	1326	0.000%	0	99.568%	305874
Amarillo	0.863%	2651	0.858%	2636	0.005%	15	99.137%	304549
Cian	0.035%	109	0.035%	109	0.000%	0	99.965%	307091
Magenta	0.148%	454	0.146%	449	0.002%	5	99.852%	306746

Imagen 12.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.299%	917	0.296%	908	0.003%	9	99.701%	306283
Azul	0.017%	51	0.017%	51	0.000%	0	99.983%	307149
Amarillo	4.971%	15272	4.926%	15132	0.046%	140	95.029%	291928
Cian	0.000%	0	0.000%	0	0.000%	0	100.000%	307200
Magenta	0.009%	27	0.008%	25	0.001%	2	99.991%	307173

Imagen 13.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.105%	322	0.087%	268	0.018%	54	99.895%	306878
Azul	0.034%	104	0.034%	104	0.000%	0	99.966%	307096
Amarillo	3.056%	9387	3.048%	9364	0.007%	23	96.944%	297813
Cian	0.056%	172	0.025%	78	0.031%	94	99.944%	307028
Magenta	0.006%	18	0.000%	0	0.006%	18	99.994%	307182

Imagen 14.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.007%	23	0.002%	5	0.006%	18	99.993%	307177
Azul	0.185%	568	0.185%	568	0.000%	0	99.815%	306632
Amarillo	0.317%	974	0.295%	906	0.022%	68	99.683%	306226
Cian	0.031%	95	0.028%	86	0.003%	9	99.969%	307105
Magenta	0.052%	161	0.049%	150	0.004%	11	99.948%	307039

Imagen 15.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.109%	334	0.081%	249	0.028%	85	99.891%	306866
Azul	3.136%	9634	3.119%	9583	0.017%	51	96.864%	297566
Amarillo	0.256%	786	0.205%	630	0.051%	156	99.744%	306414
Cian	0.007%	21	0.004%	11	0.003%	10	99.993%	307179
Magenta	0.001%	3	0.001%	3	0.000%	0	99.999%	307197

Imagen 16.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.134%	413	0.119%	367	0.015%	46	99.866%	306787
Azul	0.190%	583	0.188%	577	0.002%	6	99.810%	306617
Amarillo	3.001%	9220	2.994%	9198	0.007%	22	96.999%	297980
Cian	0.255%	784	0.255%	783	0.000%	1	99.745%	306416
Magenta	0.008%	24	0.004%	11	0.004%	13	99.992%	307176

Imagen 17.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.044%	135	0.044%	135	0.000%	0	99.956%	307065
Azul	3.033%	9316	3.033%	9316	0.000%	0	96.967%	297884
Amarillo	0.022%	68	0.002%	5	0.021%	63	99.978%	307132
Cian	0.003%	8	0.000%	0	0.003%	8	99.997%	307192
Magenta	0.004%	13	0.000%	0	0.004%	13	99.996%	307187

Imagen 18.

Clase	%	PC	%	PV	%	PF	%	NV

Naranja	0.109%	335	0.004%	12	0.105%	323	99.891%	306865
Azul	0.056%	172	0.056%	172	0.000%	0	99.944%	307028
Amarillo	2.787%	8563	2.786%	8560	0.001%	3	97.213%	298637
Cian	0.234%	720	0.219%	674	0.015%	46	99.766%	306480
Magenta	0.001%	3	0.001%	3	0.000%	0	99.999%	307197

Imagen 19.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.000%	0	0.000%	0	0.000%	0	100.000%	307200
Azul	0.001%	3	0.000%	0	0.001%	3	99.999%	307197
Amarillo	0.018%	56	0.018%	56	0.000%	0	99.982%	307144
Cian	0.032%	99	0.032%	99	0.000%	0	99.968%	307101
Magenta	0.079%	242	0.000%	0	0.079%	242	99.921%	306958

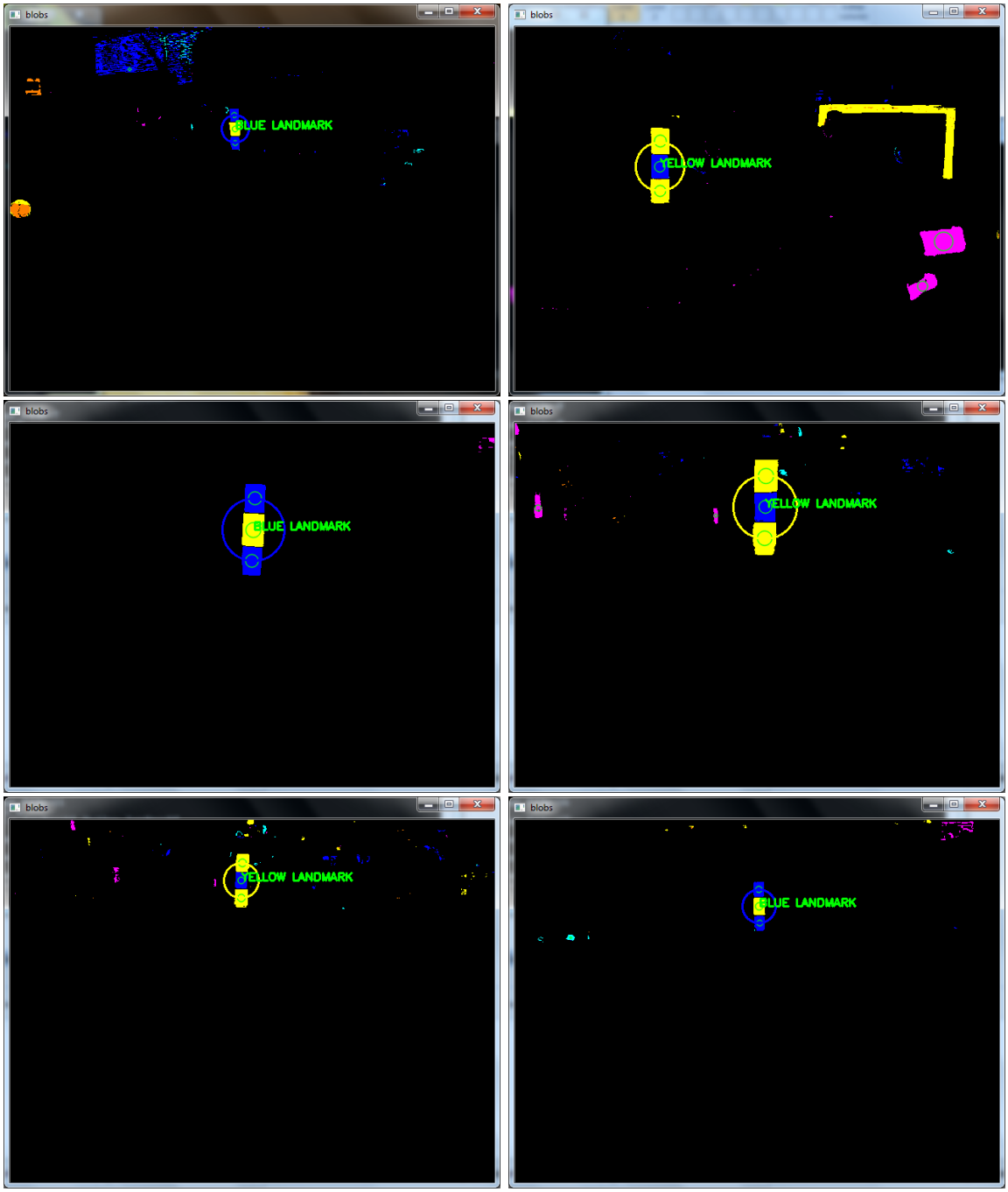
Imagen 20.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.002%	6	0.002%	5	0.000%	1	99.998%	307194
Azul	0.176%	540	0.174%	535	0.002%	5	99.824%	306660
Amarillo	0.129%	395	0.129%	395	0.000%	0	99.871%	306805
Cian	0.029%	88	0.028%	86	0.001%	2	99.971%	307112
Magenta	0.060%	185	0.060%	185	0.000%	0	99.940%	307015

Imagen 21.

Clase	%	PC	%	PV	%	PF	%	NV
Naranja	0.003%	9	0.003%	9	0.000%	0	99.997%	307191
Azul	0.110%	339	0.110%	339	0.000%	0	99.890%	306861
Amarillo	48.945%	150358	48.942%	150349	0.003%	9	51.055%	156842
Cian	0.226%	694	0.226%	694	0.000%	0	99.774%	306506
Magenta	0.000%	1	0.000%	1	0.000%	0	100.000%	307199

Anexo F. Localización de “landmarks”.




blobs
YELLOW LANDMARK
blobs
BLUE LANDMARK

Anexo G. Resultados de detección de elementos en diferentes ambientes.

En la primera tabla se muestra el total de imágenes en las cuales existía un elemento determinado a ser detectado.

Ambiente	Postes	Portería	Pelota
Laboratorio tarde	5	5	10
Laboratorio noche	3	8	11
Laboratorio mañana	7	9	13
Robocup	8	13	9

En la siguiente tabla se muestra el número de detecciones correctas de cada elemento en cada ambiente.

Ambiente	Postes	Portería	Pelota
Laboratorio tarde	5	3	10
Laboratorio noche	2	8	10
Laboratorio mañana	4	9	13
Robocup	8	13	9

Por último se muestra el número de detecciones falsas en cada ambiente para cada elemento. Esto se debe a ruido originado por cosas de color similar a los elementos de la cancha, como anuncios, luces y ropa.

Ambiente	Postes	Portería	Pelota
Laboratorio tarde	0	2	0
Laboratorio noche	1	0	1
Laboratorio mañana	1	1	0
Robocup	0	3	4