CONTROL SERVO VISUAL DE UN SISTEMA MULTIROBOTS

LUIS FERNANDO VALENCIA RUIZ

UNIVERSIDAD DE PAMPLONA
FACULTAD DE INGENIERÍAS Y ARQUITECTURA
DEPARTAMENTO MMI
PAMPLONA
2020

LUIS FERNANDO VALENCIA RUIZ

TRABAJO DE GRADO MODALIDAD INVESTIGACIÓN PARA OPTAR POR EL TÍTULO DE INGENIERO EN MECATRÓNICA

CESAR AUGUSTO PEÑA CORTÉS
Dr. Automática y Robótica

UNIVERSIDAD DE PAMPLONA
FACULTAD DE INGENIERÍAS Y ARQUITECTURA
DEPARTAMENTO MMI
PAMPLONA
2020

Nota de Aceptación

_____

_____

_____

_____
Director de tesis

_____

Jurado

_____

Jurado

Pamplona, 27 de mayo 2020

Dedico este trabajo a mis padres, ellos han estado apoyándome durante todo el proceso de formación universitario, motivándome a seguir adelante en la vida.

AGRADECIMIENTOS

Agradezco primeramente a Dios por haberme privilegiado con la vida y las capacidades para llevar este trabajo adelante, a mis padres por su apoyo y esfuerzo incondicional para conmigo, a cada profesor del programa de ingeniería mecatrónica que han aportado conocimiento y herramientas para hacer esto posible, en especial al Phd. Cesar Peña por el conocimiento impartido para culminar con éxito el trabajo, a la Universidad de Pamplona por brindar la formación académica a cada estudiante que pasa por las aulas de clase.

CONTENIDO

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE ANEXOS

Pág

RESUMEN

En este trabajo se realizó un control servo visual de un sistema multirobots, en el que se implementó una estrategia de juego para dirigir tres robots diferenciales sobre un ambiente estructurado. Se tomó una plataforma robótica libre de bajo costo a la cual se le realizaron modificaciones, añadiéndole características necesarias que le permitan patear una pelota dentro de una cancha con dimensiones de 120cm x 180cm. Se hace la identificación de cada robot, pelota y área de trabajo por medio del tratamiento digital imágenes, se trabajó este proyecto bajo el sistema servo visual basado en imágenes (IBVS). De las imágenes adquiridas por medio de una cámara, se extraen las características visuales, que identifican a cada objeto dentro del área de trabajo, calculando cada posición y orientación de los robots dentro del sistema y con respecto al objetivo del juego que es la pelota. A la cancha se le realiza una homografía para alinear la imagen entrante. La posición y orientación es enviada a cada robot mediante la implementación de un sistema de comunicación WiFi entre el ordenador y los robots, a través del protocolo HTTP con transporte de datos TCP/IP. Estas señales de control son analizadas internamente por cada robot ejecutando un movimiento relacionado a la señal de control entrante.

PALABRAS CLAVE: Control servo visual, robot diferencial, visión artificial, transformación de homografía.

SUMMARY

In this work was carried out with a visual servo control of a multi-robot system, in which a game strategy was implemented to direct three differential robots over a structured environment. A free and low-cost robotic platform was used, and some modifications were made to it, adding necessary features that allow it to kick a ball into a court with dimensions of 120cm x 180cm. The identification of each robot, ball and work area was by means of digital image processing, this project was worked under the visual servo system, based on images (IBVS). The images were acquired by means of a camera, from which the visual characteristics were extracted, that identify each object within the work area, calculating each position and orientation of the robots within the system and with respect to the objective of the game which is the ball. A homography was performed on the court in order to align the incoming image. The position and orientation are sent to each robot through the implementation of a WiFi communication system between the computer and the robots, through the HTTP protocol with TCP / IP data transport. These control signals are analyzed internally by each robot executing a movement related to the incoming control signal.

KEYWORDS: visual servoing control, wheeled mobile robot, artificial vision, homograph transformation.

1. INTRODUCCIÓN

El control servo visual ha sido un tema de mucho desarrollo, dado que permite tener un control realimentado por imágenes, y ser muy preciso en los trabajos que se necesite desenvolver. El control servo visual permite generar las leyes de control que le posibiliten al robot generar los movimientos respecto a un objetivo definido por el sistema. Se han desarrollado actualmente varias arquitecturas de control servo visual, las más importantes son: IBVS (image-based visual servoing) que extrae características visuales de imagen 2D, PVBS (position-based visual servoing) que estima las posiciones con respecto a un plano 3D y HVS (hybrid visual servoing) que es una arquitectura que mezcla las dos anteriormente nombradas. Estas imágenes entrantes a menudo necesitan alineaciones para una correcta interpretación del espacio de trabajo, mediante la homografía de imágenes se alinean estas, cambiando el espacio coordenado del mundo a un espacio coordenado relativo a la cámara. Los robots móviles son ampliamente usados ya que pueden realizar tareas complejas en espacios de trabajo. es necesario que sean provistos de sensores que le permitan tener un buen desempeño en la tarea asignada. Una cámara como sensor les permite ubicarse en el ambiente y también hacer movimientos precisos en el seguimiento de una ruta. Actualmente se cuenta con competencias que involucran a los robots móviles y el sistema de visión artificial, como lo es el fútbol robótico, que es un ejemplo de trabajo cooperativo entre robots el cual involucra un sistema de toma de decisiones que permitan dirigir varios robots mediante una estrategia de juego.

2. OBJETIVOS

2.1 OBJETIVO GENERAL

- Diseñar un control servo visual para definir las posiciones y orientaciones de un grupo de tres robots móviles con base a una estrategia de juego en un ambiente estructurado.

2.2 OBJETIVOS ESPECÍFICOS

- Identificar por medio de visión artificial la localización de los robots y la pelota dentro del entorno estructurado.
- Seleccionar una plataforma robótica de bajo costo y modificarla para jugar fútbol.
- Implementar un sistema de comunicaciones entre el algoritmo de control de alto nivel y los robots.
- Crear un algoritmo de alto nivel que permita dirigir los robots con una estrategia de juego.

3 PLANTEAMIENTO DEL PROBLEMA

3.1 DEFINICIÓN DEL PROBLEMA

La robótica de competencia a nivel mundial es un tema de mucho desarrollo, dado que permite la integración de múltiples conocimientos de robótica para su ejecución. El fútbol robótico permite la cooperación de robots empleando estrategias de decisión para ganar un juego. La Universidad de Pamplona cuenta con distintas modalidades de competencias diseñadas por el programa de ingeniería mecatrónica, donde los estudiantes se destacan por sus diseños propios y eficiencia durante los encuentros, pero aún, la categoría de fútbol robótico no está implementada dentro de las competencias robóticas de la Universidad de Pamplona, entonces, ¿Es posible implementar la categoría de fútbol robótico dentro de las competencias del programa de ingeniería mecatrónica en la Universidad de Pamplona?

3.2 JUSTIFICACIÓN

El presente trabajo se enfocará en controlar un equipo de robots mediante el análisis de imágenes, detectando a los robots y pelota, generando ubicaciones para los robots a través de estrategias de juego según la posición de una pelota dentro del área de trabajo. Se pretende darle un impulso a la categoría de fútbol robótico dentro de las competencias robóticas de la Universidad de Pamplona, logrando conformar un equipo que logre representar interna y externamente en esta modalidad de competencia. Esta categoría de competencia permite afianzar los conocimientos adquiridos en el programa de ingeniería mecatrónica. Las plataformas robóticas y códigos del sistema de tratamiento de imágenes serán dejados a disposición del programa de ingeniería mecatrónica para el uso y optimización.

4. MARCO TEÓRICO Y ESTADO DEL ARTE

La teoría del control aporta las reglas con las que un sistema funcionará de la mejor manera posible, haciendo que los tiempos de ejecución, planificación de rutas en los movimientos de los robots sean óptimos [1]. La retroalimentación de un sistema de lazo cerrado mediante sensores visuales proporciona grandes beneficios a escalas industriales dado que permiten la detección de objetos y se consideran altamente confiables por la precisión con la que se puede desenvolver un sistema [2], [3].

El control servo visual es una técnica de control que usa imágenes como retroalimentación y a partir de ellas se extrae características del ambiente las cuales son analizadas computacionalmente, luego estas señales de control son enviadas a los robots. El sistema servo visual permite dar leyes de control al robot para dirigir su movimiento y posición mediante la planeación de trayectorias con respecto al objetivo dentro del espacio de trabajo evitando posibles obstáculos [4].

La principal ventaja de este método es que los datos visuales adquiridos son similares a la manera en que el ojo humano capta imágenes del ambiente. El dispositivo de visión puede ser montado de 2 dos maneras diferentes, en la primera se acomoda la cámara estacionariamente de tal manera que logre observar al robot y al blanco que tiene, los movimientos producidos dentro del ambiente no mueven la cámara, esta configuración es llamada “*eye-to-hand*”. La segunda forma, es montar la cámara directamente en el robot, de esta manera la cámara se mueve a medida que el robot se mueve y solo se puede observar el blanco, esta manera de montar la cámara se llama “eye-in-hand” [5].

El sistema servo visual actualmente está dividido varias arquitecturas, pero solo se nombrarán las dos principales que son: 1. PBVS nombrada así por sus siglas en inglés “Position-Based Visual Servoing” que es una manera en la cual se estima la posición del robot con respecto al objetivo en el plano coordenado 3D mediante la extracción de las características de la imagen. En PBVS la imagen es muy sensible a los parámetros de la cámara. 2. IBVS llamada así por sus siglas en ingles “Image-Based Visual Servoing”, Donde la señal del error es extraída de la imagen 2D, para determinar la posición del efector final o del robot móvil con respecto al objetivo, aunque considerado menos intuitivo, es muy usado ya que elimina posibles inconvenientes que tiene el PBVS calculando la profundidad y con los ajustes de calibración de la cámara [6].

A menudo las imágenes adquiridas están desalineadas y en muchos campos de aplicación es necesaria una alineación para hacer ajustes correctos en el sistema de control, estos ajustes se logran a través de una transformación de homografía,

que es la combinación de transformaciones simples como traslación, rotación, escalado y cambio de perspectiva; este tipo de transformación geométrica es muy común en los sistemas de visión puesto que permiten adaptar la imagen a un campo de vista relativo [7].

Los robots móviles a menudos son muy usados por su amplio campo de aplicación, y pueden realizar tareas muy complejas en grandes espacios de trabajo colaborando en lugares hostiles que afectarían la salud y el bienestar de los humanos [8], [9], Este tipo de robot necesitan proveerse de muchos sensores y elementos que ayuden a realizar su trabajo de la manera más precisa posible, incluso sin la supervisión constante de humanos, para esto, el sistema debe contar con la planificación de rutas y seguimiento de trayectoria evitando posibles obstáculos presentados dentro del ambiente, hacer estos movimientos precisos el robot debe contar con varios grados de autonomía y sistema de navegación que le permita hacer una auto localización en el espacio de tarea [10]–[12].

El fútbol robótico es una manera de evaluar plataformas cooperativas de sistemas multirobots dentro de un ambiente dinámico. En estos sistemas, varios autores describen a cada robot de un equipo como agentes, estos agentes deben ser capaces de reaccionar ante los cambios que sean presentados dentro del entorno. Los sistemas completos de tomas de decisiones se dividen en dos partes que son: la valoración del entorno de juego y la toma de decisiones que selecciones la estrategia más eficiente con el fin de obtener un buen resultado de juego. Hay dos tipos de arquitecturas empleadas en el fútbol robótico, la primera es de control centralizado, donde hay una cámara global para el entorno, una unidad de procesamiento central y un solo sistema de tomas de decisiones, la comunicación entre la unidad de procesamiento central y los robots son inalámbricas. El segundo tipo de arquitectura es de control distribuido, en el cual cada integrante del equipo es autónomo y la percepción del ambiente es individual [13], [14].

En el juego, la estrategia que se emplea se define como aquel grupo de reglas que se pueden aplicar en las diferentes situaciones del juego para poder lograr el objetivo que es ganar el partido. Para lograr ejecutar una estrategia se hace necesario mapear el entorno donde se mueven los robots y extraer las características de estos, como lo son la posición y orientación de estos en el entorno y con respecto a la pelota, también se recomienda conocer la posición relativa que tienen los robots de un equipo con respecto al equipo contrario. La estrategia puede ser aplicada a diferentes zonas del entorno y las reglas usadas pueden variar con respecto al desarrollo del juego y a la distribución de los agentes en el juego [15], [16].

La RoboCup es una competencia internacional que pretende demostrar la adaptación que de sistemas robóticos inteligentes a través de un juego, donde las decisiones y planeaciones de rutas sean en tiempo real, logrando así la cooperación

distribuida entre los equipos, previniendo posibles colisiones dentro del entorno [17], [18].

Diversos trabajos reportan avances en los temas relacionados al tratamiento digital de imágenes usados para controlar distintos tipos de robot en ambientes muy diferentes, tal como es el caso del novedoso modelo de control adaptativo descrito por C. Hua, Y. Wang y X. Guan en su artículo “Visual tracking control for an uncalibrated robot system with unknown camera parameters”, este controlador lo usan sobre un manipulador al cual adaptan una cámara en el efector final. Esta implementación permite al manipulador seguir un objeto 3D mediante las características que este objeto tiene. El manipulador logra calcular la posición y orientación de su efector final con respecto al objetivo mediante las coordenadas extraídas de la imagen tomada. Concluyen con pruebas satisfactorias en el seguimiento de trayectorias incluso desconociendo los parámetros de la cámara, la dinámica del manipulador y añadiendo más perturbaciones al sistema [19].

Z. Ma y J. Su describen en su investigación “Robust uncalibrated visual servoing control based on disturbance observer” un control servo visual con alto rechaso a perturbaciones (DOB). Implementan un sistema binocular servo visual sobre el manipulador teniendo como reto llevar el brazo del robot hasta el objetivo con ayuda del sistema binnocular de cámaras. Desconocen los parámetros de la cámara. Las imágenes extraídas del sistema de visión permite calcular el error de posición del efector final con la posición deseada, ya calculado el error de posición, el controlador PI calcula instrucciones que son enviadas el manipulador con valores deseados. El manipulador es el brazo derecho del robot NAO, Se concluye con un seguimiento de la trayectoria con error reducido a cero [20].

K. Ahlin, B. Joffe, A. P. Hu, G. McMurray, and N. Sadegh en “Autonomous Leaf Picking Using Deep Learning and Visual-Servoing” hablan de la tarea difícil que es agarrar objetos sin una forma estructurada. Describen que el desafio radica en escoger qué objetos son deseables para captar y ubicar el lugar relativo en el que están con respecto al efector final. Implementan un brazo robótico con la finalidad de poder sujetar elementos con geotría irregular. Combinan redes neuronanles para el procesamieinto de las imágenes de dos cámaras y su análisis de la profundidad. Se describe la constante necesidad de rastrear la hoja objetivo a medida que el manipulador se mueve, además, deben triangular las coordenadas 3D medienate puntos caracteríStos dentro de la hoja. Impletan el sistema IBVS y el análisis de profundidad monoscópica (MDA). Estos métodos le permitieron al brazo robótico agarrar una hoja en un ambiente destructurado [21].

En el trabajo “Robust fulfillment of constraints in robot visual servoing”, P. Muñoz-Benavent, L. Gracia, J. E. Solanes, A. Esparza, y J. Tornero en el modo deslizante que satisface las restricciones de un robot servo visual en el cual combinan IBVS y

PBVS. Usan restricciones para evitar que robot exceda limítaciones del robot como son: velocidad y rango articular. Consideran seguir un objeto mediante imágenes en tres partes: tomar las caracteristicas visuales del plano, luego convertir coordenadas del plano en coordendas de pixeles y por último estimar la posición de robot. Usaron el control deslizante para satisfacer restricciones del robot y restricciones de visibilidad [22].

G. Allibert, M. D. Hua, S. Krupínski y T. Hamel proponen un control IBVS para el seguimiento de tuberias de vehiculos subamarinos autónomos. El controlador usa las caracteristicas de las imágenes obtenidas sin tener en cuenta la posición del robot con respecto al ambiente puesto que no consideran obligatorio conocer el mundo cartesiano [23].

Y. Zhang, C. Hua, Y. Li, and X. Guan en el artículo “Adaptive neural networks-based visual servoing control for manipulator with visibility constraint and dead-zone input” un control en linea servo visual con redes neuronales adaptativas para aproximar la dinámica no lineal desconocida con restricciones de estado completo y entrada de zona muerta para evaluar su rendimiento en el seguimiento de trayectorias mediante simulaciones [24].

5 DESARROLLO DEL PROYECTO

En esta sección del libro de describirá cada uno de los procesos que se llevaron a cabo para la realización del proyecto.

5.1. TRATAMIEINTO DIGITAL DE IMAGENES

En esta sección del libro se hace uso del tratamiento digital de imágenes para extraer las características visuales que tiene el ambiente estructurado. Para esto primero se debe leer y visualizar en pantalla la imagen de la cancha. Se lee una imagen mediante el comando "*imread()*" y se muestra en pantalla con "*imshow()*".

```
RGB = imread('n1.jpg');
f1=figure(1);, set(f1,'Color',[1,1,1]);
imshow(RGB);
```

Figura 1. Ambiente estructurado compuesto de la cancha de fútbol, los 2 equipos con su respectivo color y marcas, y la pelota.

Teniendo la imagen almacenada en una variable, se debe localizar todo lo que es la cancha para seleccionar el área de trabajo en el que se van a desarrollar los movimientos de los robots. En este caso se selecciona los datos de un pixel correspondiente al verde de la cancha y con ayuda del error cuadrático medio se hayan los pixeles parecidos a ese color. Se hace un recorte de la cancha para luego sacar el histograma del color verde de la cancha.

Figura 2. Recorte de una sección de cancha que solo muestre el color verde

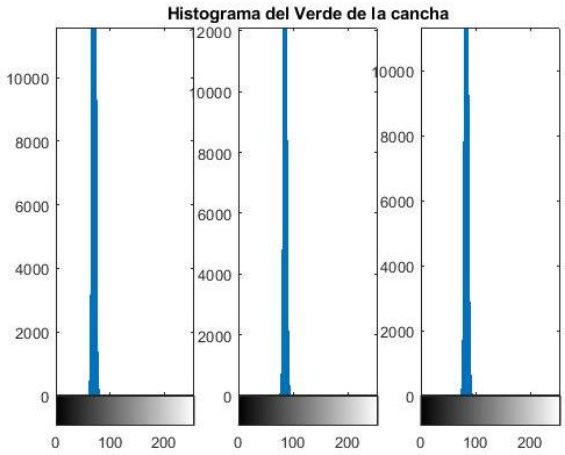


Figura 3. Histograma del color verde de la cancha.

```
RGB1 = double(RGB);           % Crea una copia de la imagen original
color1 = [80, 90, 85];        % Pixel RGB de la cancha
Error = ( (RGB1(:,:,1)- color1(1)).^2 + (RGB1(:,:,2)- color1(2)).^2 +
(RGB1(:,:,3)- color1(3)).^2 ).^0.5; %ECM
Error_tolerable1 = 82;        % Margen de error para la detección de la
cancha
[f,c]= size(Error);
Resultado = zeros(f,c);
Resultado_G= Resultado;
Resultado_G(Error < Error_tolerable1)=120;
Resultado_G = bwareaopen(Resultado_G,50000); % Filtro de imagen
Resultado_RGBin = im2bw(Resultado_G); % Imagen a blanco y negro
(Binaria)
f2 = figure(2);, set(f2,'Color',[1,1,1]);
imshow(Resultado_G);% Muestra en pantalla el resultado de la operación
```

En este caso se ubica un pixel RGB con los valores de 87 en la matriz de rojos, 99 en la matriz de verdes y 99 en la matriz de azules.
La ecuación del error cuadrático medio quedaría de la siguiente forma:

$$Error = \sqrt[2]{(R_a - 87)^2 + (G_a - 99)^2 + (B_a - 99)^2} \quad (1)$$

Donde:
Ra: Valor de los n pixeles en la matriz de rojos.
Ga: Valor de los n pixeles en la matriz de verdes.
Ba: Valor de los n pixeles en la matriz de azules.

El resultado obtenido de la operación hecha se muestra en la Figura 4.

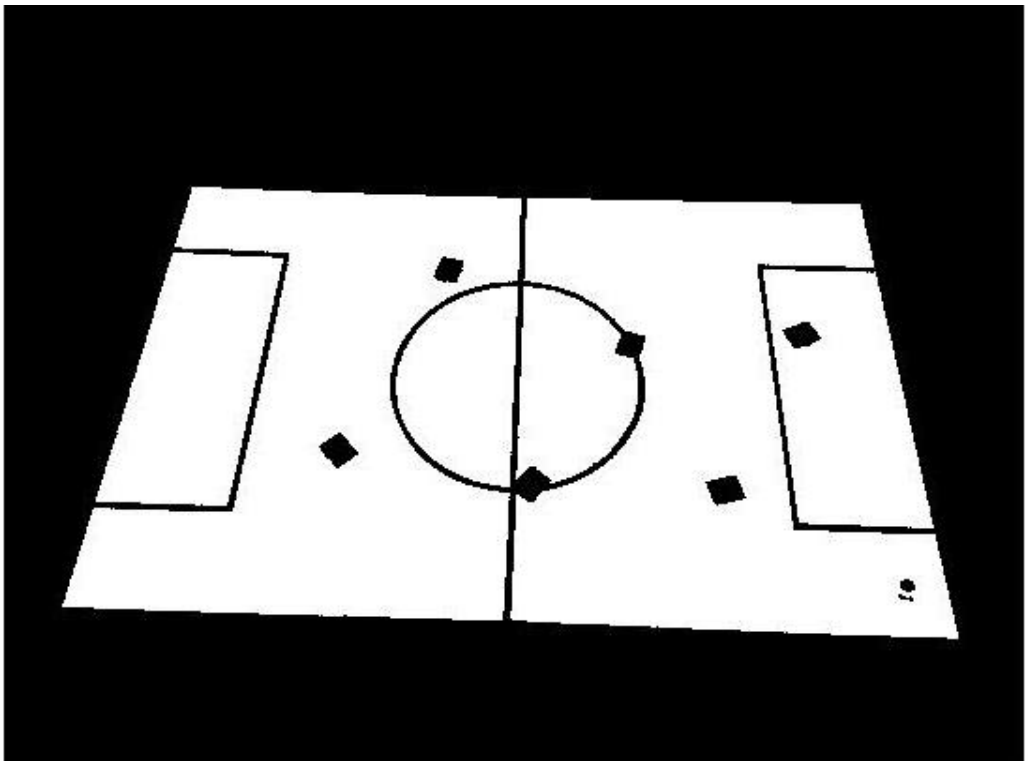


Figura 4. Imagen binaria de la cancha.

Para hacer una correcta identificación de los colores inmersos dentro de la cancha se debe hacer uso del histograma de colores y así asignar umbrales de los colores de cada objeto. Para eso se usa el comando "*imhist()*".

```
RGBr=RGB(1545:1577,1649:1691,:);    %Recorte de una sección de imagen
f2=figure(2);, set(f2,'Color',[1,1,1]);
imshow(RGBr);
f3=figure(3);, set(f3,'Color',[1,1,1]); %Muestra en pantalla el
histograma
subplot(1,3,1);imhist(RGBr(:,:,1));title('Histograma Componente R');
subplot(1,3,2);imhist(RGBr(:,:,2));title('Histograma Componente G');
subplot(1,3,3);imhist(RGBr(:,:,3));title('Histograma Componente B');
```

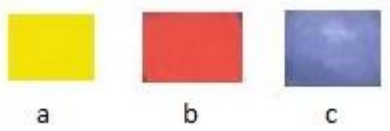


Figura 5. Recortes de los colores inmersos en cada objeto dentro de la cancha.

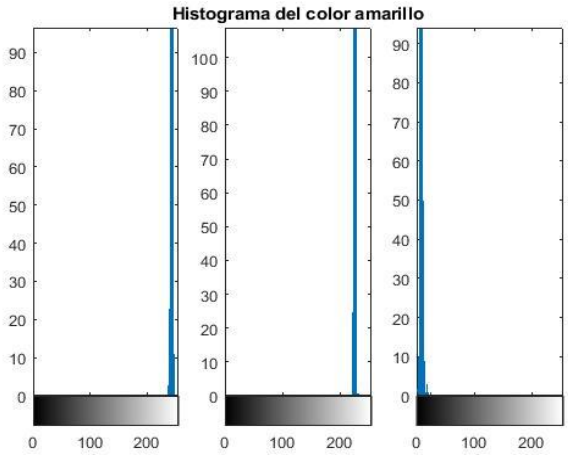


Figura 6. Histograma del recorte de la sección amarilla, ver figura 3a.

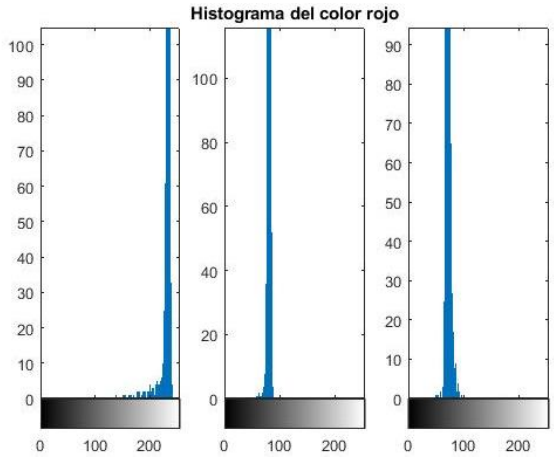


Figura 7. Histograma del recorte de la sección roja, ver figura 3b.

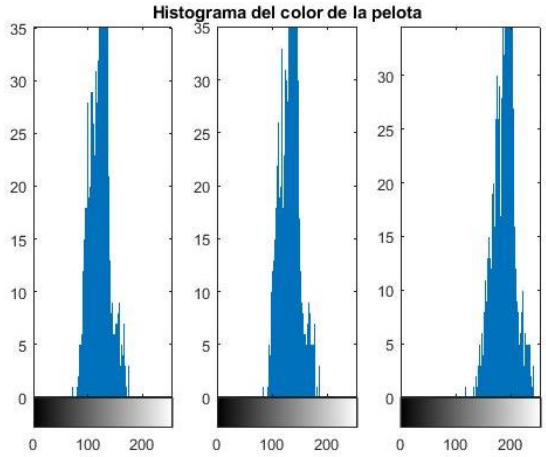


Figura 8. Histograma del recorte de la sección azul, ver figura 3c.

A partir de los resultados obtenidos se procede a asignar umbrales en los códigos de colores para la identificación de cada color y objeto dentro del ambiente. Estos umbrales es un rango de valores desde un mínimo hasta un máximo en cada matriz de colores, quedaría de la siguiente manera:

```
vur_min = 225;, vur_max = 255;      % umbrales asignados para la
vug_min = 203;, vug_max = 237;      % identificación de los colores
vub_min = 1;, vub_max = 65;          % amarillos

vur_min = 200;, vur_max = 250;      % umbrales asignados para la
vug_min = 65;, vug_max = 110;       % identificación de los colores
vub_min = 57;, vub_max = 105;       % rojos

vur_min = 100;, vur_max = 150;      % umbrales asignados para la
vug_min = 130;, vug_max = 170;       % identificación de la pelota
vub_min = 160;, vub_max = 220;
```

Una vez asignados los umbrales de los colores se procede a ubicar cada objeto dentro de la cancha en su sitio (ver anexos, *Identificación de equipos y pelota*), el resultado es el siguiente:

Figura 9. Detección de objetos con sus respectivos colores dentro del área de trabajo.

5.1.1 HOMOGRAFÍA

A partir de los resultados obtenidos, se puede notar que la imagen está desalineada y que necesita una corrección, esta corrección se puede hacer con la ayuda de una homografía visual. Esta transformación permite hacer un cambio de coordenadas relativas al campo de visión por la cámara y de esta manera obtener la correcta orientación de la imagen y así lograr apreciar las distancias que tienen cada robot con respecto a la pelota. Para esto se deben encontrar las cuatro esquinas de la cancha. En este trabajo se usaron dos métodos de búsqueda de esas esquinas, el primero el método de Harris y el segundo método se usó a partir de la imagen binaria de la cancha (Figura 4) se buscan los pixeles más cercanos y más lejanos con respecto a las dos esquinas superiores de la imagen.

El primer método fue implementado de la siguiente manera:

```
A = corner(Resultado_RGBin,'Harris',1200); % Método para buscar
esquinas por Harris de hasta 1200 esquinas
A = unique(A(:,:),'rows'); % Ordena filas de menor a mayor
B = find(A(:,2)<659); % Busca las esquinas con un valor menor a 659 en
el eje Y
X = zeros(4,1); % Crea vector de 4 posiciones
Y = zeros(4,1); % Crea vector de 4 posiciones
X(1) = A(1,1);  % Almacena esquina inferior izquierda
Y(1) = A(1,2);
X(2) = A(end,1); % Almacena esquina inferior derecha
Y(2) = A(end,2);
X(3) = A(B(1),1); % Almacena esquina superior izquierda
Y(3) = A(B(2),2);
X(4) = A(B(end),1); % Almacena esquina superior derecha
Y(4) = A(B(end),2);
imshow(RGB); hold on; % Imprime imagen en pantalla con cada esquina
plot(X,Y,'o','Color','blue');
```

Entre los parámetros de entrada de la herramienta de búsqueda de esquinas por Harris, se tuvo que permitir encontrar hasta 1200 esquinas para que entre ellas aparecieran las cuatro (4) esquinas de la cancha, en ocasiones con un valor de 300 el algoritmo era capaz de encontrarlas, pero en otras fallaba encontrando al menos una (1) de las cuatro (4) esquinas.

Figura 10. Búsqueda de las cuatro esquinas de la cancha por el método de Harris con un valor de 1200 esquinas.

Este método no es del todo eficiente pues tarda alrededor de 3.9 segundos en hallar las cuatro esquinas de la cancha. Por ese motivo se requiere otras formas de encontrar esas esquinas por código y se buscan aquellos pixeles que están más cercanos y más alejados con respecto a las esquinas de las imágenes. Se emplea de la siguiente manera:

```
RGBe = RGB; % se guarda la imagen en otra variable
[nf,nc]=size(Resultado_G); % Guarda el tamaño de la imagen original
[X,Y] = meshgrid(1:nc,1:nf); % Crea matriz X donde las columnas son
idénticas y Y una matriz donde la filas son idénticas. Ambas del tamaño
de la imagen.
Xm = X.*Resultado_G; % Multiplica los datos de la matriz binaria por
los datos de la matriz X.
Ym = Y.*Resultado_G; % Multiplica los datos de la matriz binaria por
los datos de la matriz Y.
D = Xm.^2+Ym.^2;
eID=[X(find(D==max(max(D)))), Y(find(D==max(max(D))))]; % Busca esquina
máxima de izquierda a derecha
D(D==0)=1000000000000;
eSI=[X(find(D==min(min(D)))), Y(find(D==min(min(D))))]; % Busca esquina
mínima de izquierda a derecha
D(D==1000000000000)=0;
Xm2 = (X-nc).*Resultado_G; % invierte las columnas, ahora mayor a menor
D2 = Xm2.^2+Ym.^2;
eII=[X(find(D2==max(max(D2)))), Y(find(D2==max(max(D2))))]; % Busca
esquina máxima de derecha a izquierda
D2(D2==0)=1000000000000;
eSD=[X(find(D2==min(min(D2)))), Y(find(D2==min(min(D2))))]; % Busca
esquina mínima de derecha a izquierda
D2(D2==1000000000000)=0;Esquinas = [eSI;eSD;eID;eII];
```

Figura 11. Esquinas de la cancha por el método de mínimos y máximos.

En la implementación de este método se puede observar una mejora en la optimización del código ya que es este solo demora aproximadamente 1.7 segundos en su ejecución. De esta manera se pueden hallar de forma directa las cuatro esquinas de la cancha, se almacenan en una matriz 4x2 ordenadas en sentido horario que luego será usada para el cambio de perspectiva de la imagen. La ejecución del código para encontrar las cuatro esquinas de la cancha se realiza una sola vez durante la calibración.

Ahora se debe hallar la matriz H de transformación 2D – 2D, la cual transforma un punto en el plano $\boldsymbol{X_{x,y}}$ al plano de la cámara $\boldsymbol{X_{i,j}}$, de tal forma que:

$$\boldsymbol{X_{i,j} = HX_{x,y}} \qquad (2)$$

Donde la matriz $\boldsymbol{H_{3x3}}$ se puede llamar matriz de transformación mundo-cámara, así tenemos:

$$\begin{bmatrix} \boldsymbol{i} \\ \boldsymbol{j} \\ \boldsymbol{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{h_{11}} & \boldsymbol{h_{12}} & \boldsymbol{h_{13}} \\ \boldsymbol{h_{21}} & \boldsymbol{h_{22}} & \boldsymbol{h_{23}} \\ \boldsymbol{h_{31}} & \boldsymbol{h_{32}} & \boldsymbol{h_{33}} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{1} \end{bmatrix} \qquad (3)$$

Para una correcta correspondencia de puntos, la ecuación anterior se puede reescribir de la siguiente forma:

$$\begin{bmatrix} \boldsymbol{x} & \boldsymbol{y} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & -\boldsymbol{ix} & -\boldsymbol{iy} & -\boldsymbol{i} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{x} & \boldsymbol{y} & \mathbf{1} & -\boldsymbol{jx} & -\boldsymbol{jy} & -\boldsymbol{j} \end{bmatrix} \begin{bmatrix} \boldsymbol{h_{11}} \\ \boldsymbol{h_{12}} \\ \boldsymbol{h_{13}} \\ \boldsymbol{h_{21}} \\ \boldsymbol{h_{22}} \\ \boldsymbol{h_{23}} \\ \boldsymbol{h_{31}} \\ \boldsymbol{h_{32}} \\ \boldsymbol{h_{33}} \end{bmatrix} = \mathbf{0} \qquad (4)$$

De esta forma el problema se resume en resolver un sistema matricial $\boldsymbol{AX} = \mathbf{0}$, que puede ser visto como un problema de autovalores para $\boldsymbol{B} = \boldsymbol{A^T A}$. Donde se debe buscar el autovector relacionado con el menor autovalor obtenido para B, para calcular los valores de los elementos de H.

Se usa directamente la ecuación (4) y se resuelve el sistema de ecuaciones, hallando la relación entre las coordenadas de entrada y las salidas esperadas, queda de la siguiente manera:

```
x=[1;1800;1800;1]; y=[1;1;1200;1200];  %Vectores con el tamaño del eje
X y Y
A=zeros(8,8);
A(1,:)=[X(1),Y(1),1, 0,   0,   0, -1*X(1)*x(1), -1*Y(1)*x(1)];
A(2,:)=[0,   0,   0, X(1),Y(1),1, -1*X(1)*y(1), -1*Y(1)*y(1)];
A(3,:)=[X(2),Y(2),1, 0,   0,   0, -1*X(2)*x(2), -1*Y(2)*x(2)];
A(4,:)=[0,   0,   0, X(2),Y(2),1, -1*X(2)*y(2), -1*Y(2)*y(2)];
A(5,:)=[X(3),Y(3),1, 0,   0,   0, -1*X(3)*x(3), -1*Y(3)*x(3)];
A(6,:)=[0,   0,   0, X(3),Y(3),1, -1*X(3)*y(3), -1*Y(3)*y(3)];
A(7,:)=[X(4),Y(4),1, 0,   0,   0, -1*X(4)*x(4), -1*Y(4)*x(4)];
A(8,:)=[0,   0,   0, X(4),Y(4),1, -1*X(4)*y(4), -1*Y(4)*y(4)];
%En un vector columna se guardan los datos de los vectores X e Y para
cada par de coordenadas cartesianas
H=[x(1);y(1);x(2);y(2);x(3);y(3);x(4);y(4)]; %Salidas esperadas
u=A\H; %Ahora resuelve el sistema de ecuaciones
%Escoger el tipo de forma projective2d e ingresa la matriz de
%transformación U transpuesta para crear la estructura de
%transformación proyectiva
T=projective2d(U');
%Se hace la transformación de la imagen original usando T y
%xWorldLimits y yWorldLimits que son dos vectores de un par de
%elementos reales que especifican la ubicación espacial de la imagen
%de salida P2 en el espacio de salida 2-D X-Y.
xWorldLimits = [1 1800]; yWorldLimits = [1 1200];
P2=imwarp(RGB,T,'FillValues',255,'OutputView',imref2d([1200 1800
3],xWorldLimits,yWorldLimits));
```

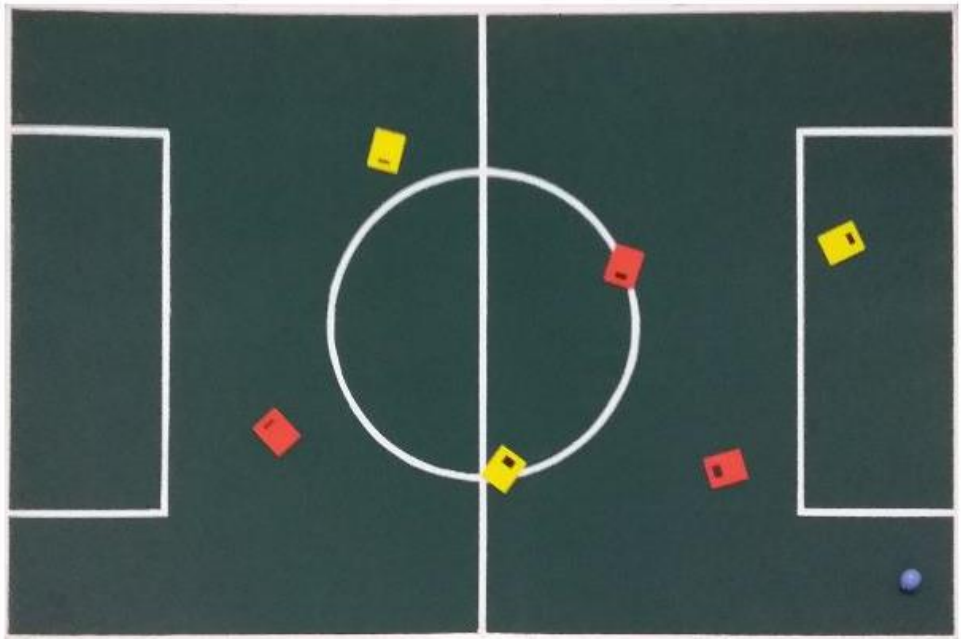


Figura 12. Homografía de la imagen de la cancha 2D conservando todos los colores.

5.1.2 POSICIÓN Y ORIENTACIÓN DE CADA ROBOT

Una vez terminado el proceso de homografía sobre la imagen original del entorno de juego, se debe identificar el rango de cada robot dentro del equipo. Es necesario seguir aplicando tratamiento digital de imágenes para hallar la numeración del robot por tamaño del área en la sección del cuadro negro, también se hace necesario calcular la posición y orientación de los robots en el área de juego.
Primero es necesario hacer una copia de cada banda de colores dentro de la imagen RGB.

```
% Se descompone la imagen en cada una de sus bandas y se guardan en
% matrices
R_i = RGB_M(:,:,1);
G_i = RGB_M(:,:,2);
B_i = RGB_M(:,:,3);
```

Luego, extrae los robots que tienen el color rojo por medio de sus umbrales máximos y mínimos. Se usa la operación lógica AND para tomar los pixeles que cumplan con la condición de rojos y luego se filtra la imagen por posibles pixeles erróneos.

```
% Se guardan las componentes para evaluar los umbrales de colores de
TeamR =R_i;   % los objetos dentro del ambiente
TeamG =G_i;
TeamB =B_i;
TeamR(TeamR < vur_minR) = 0;, TeamR(TeamR > vur_maxR) = 0;
TeamG(TeamG < vug_minR) = 0;, TeamG(TeamG > vug_maxR) = 0;
TeamB(TeamB < vub_minR) = 0;, TeamB(TeamB > vub_maxR) = 0;
Team = TeamR & TeamG & TeamB; % evalúa que se cumpla la condición de
colores
Team = bwareaopen(Team,3000); % filtra la imagen de posible ruido
```

Una vez hallado los tres robots de un equipo se debe ubicar los centroides de cada uno para conocer la posición con respecto al sistema de referencia XY de la imagen. También se debe identificar la marca que simboliza el rango de cada robot (rectángulo negro dentro la sección de color de cada equipo), por medio de la operación lógica NOT en la imagen binaria y filtro de áreas, para solamente dejar las marcas de cada robot, organizándolas por tamaño.
Luego de esto se calculan los ángulos de cada robot por medio de trigonometría como se observa en la figura 13.

```
% Centroide
TeamFill = imfill(Team,'holes'); % Rellena los huecos dentro la imagen
binaria
% Marcas
Team_not = not(Team); % Negación de la imagen binaria original, para
detectar las marcas
Team_marc = bwareafilt(Team_not,[0,1200]); % Filtra el resultado de la
negación
Team_marc = bwareaopen(Team_marc,80); % Filtra ruido en imagen
% Areas y centoides
Pos_Fill = regionprops(TeamFill,'Centroid'); % centroide de techo del
robot
PosArR_marc = regionprops(Team_marc,'Centroid','Area'); % centroide de
la marca del robot
AreaMarca = [PosArR_marc.Area]; % Guarda el área de la marca de los
robots
PosMarca = [PosArR_marc.Centroid]; PosMarca =
                                    [PosMarca(1),PosMarca(2);...
                                     PosMarca(3),PosMarca(4);...
                                     PosMarca(5),PosMarca(6)]; % Guarda
los centroides de cada marca
PosFill = [Pos_Fill.Centroid]; PosFill = [PosFill(1),PosFill(2);...
                                      PosFill(3),PosFill(4);...
                                      PosFill(5),PosFill(6)]; % Guarda
los centroides de cada robot
% Encuentrar el rango de los robots
Ar_Max = find(AreaMarca == max(AreaMarca));
Ar_min = find(AreaMarca == min(AreaMarca));
Ar_med = find(AreaMarca ~= min(AreaMarca) & AreaMarca ~=
max(AreaMarca));
```

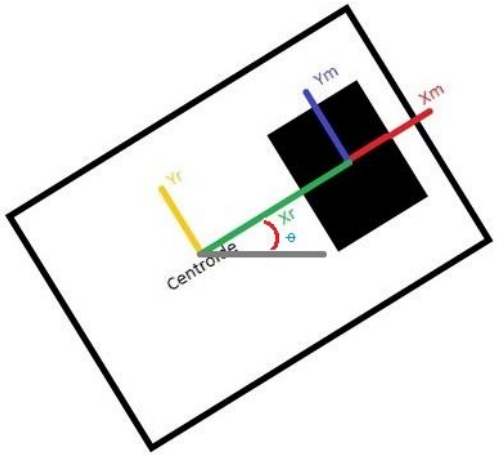


Figura 13. Orientación θ del robot con el propio marco de referencia con respecto a la horizontal.

Figura 13 muestra la vista superior del robot, con el centroide del robot y centroide de la marca que tiene cada robot (área negra), se asume que el robot está a un ángulo θ con respecto a la horizontal.

Donde:
Y_r: Eje vertical del robot.
X_r: Eje horizontal del robot.
Y_m: Eje vertical de la marca del robot.
X_m: Eje horizontal de la marca del robot.
θ: Ángulo en el que el robot se encuentra girado.
Anteriormente se calcularon los centroides tanto del robot como de la marca y esos son usados para hallar la diferencia que hay en la vertical y horizontal, teniendo esa diferencia se puede calcular el ángulo por la siguiente ecuación:

$$\theta = \tan^{-1}\left(\frac{Y_m - Y_r}{X_m - X_r}\right) \quad (5)$$

```
Resta_Pos = [PosFill - PosMarca];
An_T = [round(atan2d(Resta_Pos(1,1),Resta_Pos(1,2)));...
        round(atan2d(Resta_Pos(2,1),Resta_Pos(2,2)));...
        round(atan2d(Resta_Pos(3,1),Resta_Pos(3,2)))]+90;
```

Luego, todos los datos de cada robot son almacenados en un arreglo guardando la información del área de la marca, la posición en XY y el ángulo al cual está girado cada robot.

```
Team_Red =
[AreaMarca(Ar_min),PosFill(Ar_min,1),PosFill(Ar_min,2),An_T(Ar_min);...

AreaMarca(Ar_med),PosFill(Ar_med,1),PosFill(Ar_med,2),An_T(Ar_med);...

AreaMarca(Ar_Max),PosFill(Ar_Max,1),PosFill(Ar_Max,2),An_T(Ar_Max)];
```

Este procedimiento se aplica tanto para el equipo rojo como para el equipo amarillo, con respecto a la detección de la pelota solo debemos extraer la posición XY en la que se encuentra con respecto al plano coordenado de la imagen.

```
BallR =R_i;
BallG =G_i;
BallB =B_i;
BallR(BallR < vur_minP) = 0;, BallR(BallR > vur_maxP) = 0;
BallG(BallG < vug_minP) = 0;, BallG(BallG > vug_maxP) = 0;
BallB(BallB < vub_minP) = 0;, BallB(BallB > vub_maxP) = 0;
Ball = BallR & BallG & BallB;
Ball = bwareaopen(Ball,300); % Filtro a la imagen
Ball = imfill(Ball, 'holes'); % Rellena espacios vacíos en la imagen
Pos_ball = regionprops(Ball,'Centroid');
XY_ball = Pos_ball.Centroid; % Aposición XY
```

Almacenados los datos de cada robot como son sus posiciones y orientaciones se debe calcular el error de posición que hay entre estos y la pelota, para eso se restan sus posiciones relativas, también se debe calcular el ángulo de error que hay entre la orientación que tiene el robot y el deseado para estar de frente a la pelota.
Se realizan las mismas líneas de código para calcular el error de posición de los robots amarillos con respecto a la pelota, dado como resultado lo mostrado en la Figura 14.

```
PuntoD = [1800 600]; % Punto medio en que se ubica la meta Derecha
% Angulo de salida
Ang_D = round([atan2d(-1*(PuntoD(2) - Z(1,2)),PuntoD(1) - Z(1,1));...
               atan2d(-1*(PuntoD(2) - Z(2,2)),PuntoD(1) - Z(2,1));...
               atan2d(-1*(PuntoD(2) - Z(3,2)),PuntoD(1) - Z(3,1))]);
% calculo punto D (posicion de pateo) del robot hacia meta
D = 120;  % 10
Ar = [-1*(cosd(Ang_D(1))*D) + Z(1,1), (sind(Ang_D(1))*D) + Z(1,2);...
      -1*(cosd(Ang_D(2))*D) + Z(2,1), (sind(Ang_D(2))*D) + Z(2,2);...
      -1*(cosd(Ang_D(3))*D) + Z(3,1), (sind(Ang_D(3))*D) + Z(3,2)];
% Angulo de trayectoria robot respecto a D
Error_R = [Ar(1,1) - Team_Red(1,2), Ar(1,2) - Team_Red(1,3);...
           Ar(2,1) - Team_Red(2,2), Ar(2,2) - Team_Red(2,3);...
           Ar(3,1) - Team_Red(3,2), Ar(3,2) - Team_Red(3,3)];
% Magnitud de la distancia entre robots y pelota
Mag_R = round([norm(Error_R(1,:));...
               norm(Error_R(2,:));...
               norm(Error_R(3,:))]);
% Corrección del angulo de los robots
Ang_pend_R = [round(atan2d(-1*Error_R(1,2),Error_R(1,1)));...
              round(atan2d(-1*Error_R(2,2),Error_R(2,1)));...
              round(atan2d(-1*Error_R(3,2),Error_R(3,1)))];
Ang_Cr = [Ang_pend_R(1) - Team_Red(1,4);...
          Ang_pend_R(2) - Team_Red(2,4);...
          Ang_pend_R(3) - Team_Red(3,4)];
```

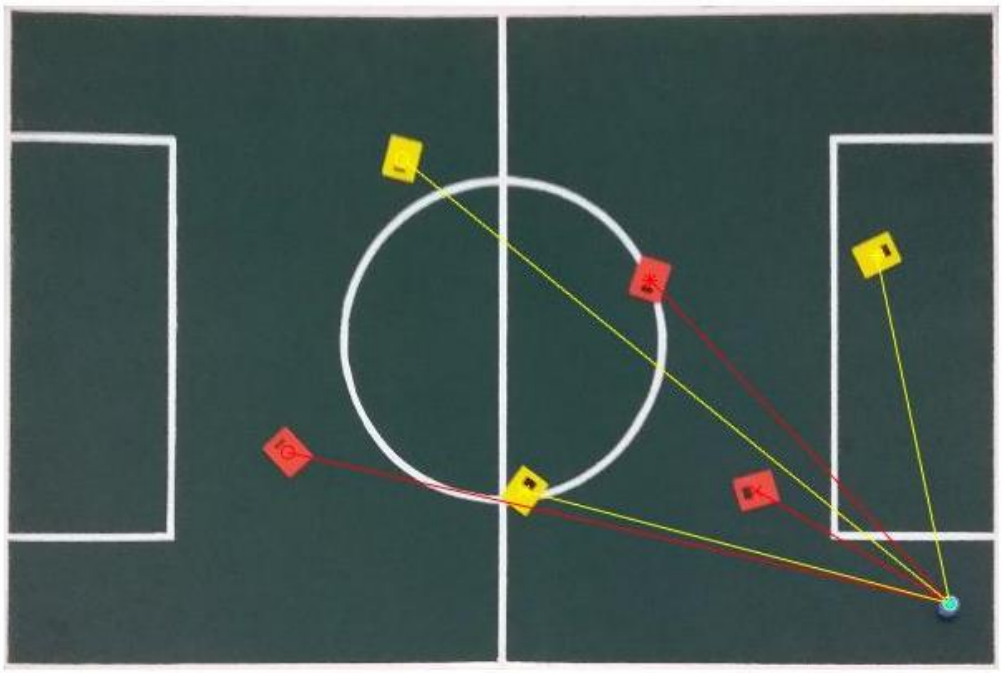


Figura 14. Error de posición de los robots con respecto a la pelota.

Tabla 1. Resultados obtenidos del error de posición y error de orientación de los robots con respecto a la pelota.

	Equipo rojo		**Equipo amarillo**	
	Angulo a corregir (grados)	Magnitud de la distancia (pixeles)	Angulo a corregir (grados)	Magnitud de la distancia (pixeles)
Robot 1	-147	1228	64	1276
Robot 2	112	797	-105	643
Robot 3	134	405	-69	796

En la tabla 1 se muestra el resultado obtenido en la figura 14, donde se mide el ángulo que debe girar cada robot para estar de frente a la pelota y la distancia en pixeles que le falta al robot para llegar hasta la pelota. En el caso de del robot número 1 del equipo, rojo debe hacer un giro de 147 grados hacia la derecha y avanzar 1228 pixeles para encontrarse con la pelota y el robot número 1 del equipo amarillo debe hacer un giro de 64 grados hacia la izquierda y avanzar 1276 pixeles para alcanzar la pelota.

5.1.3 DIVISIÓN DEL ENTORNO DE JUEGO POR ZONAS, ESTRATEGIA DE JUEGO

Teniendo el error de posición se debe hacer una estrategia de equipo, que dependa de la zona en la que se encuentre la pelota. Como no sirve que todos los robots ataquen al mismo tiempo la pelota, entonces, cada robot debe tener asignado un lugar para estar a medida que se desarrolla la estrategia, un solo robot debe atacar la pelota y los otros dos deben estar a la defensiva o uno a la defensiva y otro en un punto de ofensiva también. La cancha se divide en las siguientes 18 zonas:

Figura 15. Zonas en que la cancha es dividida.

Cada zona será ocupada solamente por un (1) solo robot de un mismo equipo, cada robot se posicionará en un espacio correspondiente dependiendo el lugar donde se encuentre la pelota.

Tabla 2. Zonas que deben cubrir cada robot según la posición en que esté la pelota.

ESTRATEGIA DE EQUIPO			
Ubicación de la pelota	Robot 2	Robot 1	Robot 3
Zona 1	Pelota	Zona 2	Zona 6
Zona 2	Zona 4	Pelota	Zona 6
Zona 3	Zona 4	Zona 2	Pelota
Zona 4	Pelota	Zona 5	Zona 9
Zona 5	Zona 7	Pelota	Zona 9
Zona 6	Zona 7	Zona 5	Pelota
Zona 7	Pelota	Zona 8	Zona 12
Zona 8	Zona 10	Pelota	Zona 13
Zona 9	Zona 10	Zona 8	Pelota
Zona 10	Pelota	Zona 11	Zona 15
Zona 11	Zona 13	Pelota	Zona 15
Zona 12	Zona 13	Zona 11	Pelota
Zona 13	Pelota	Zona 11	Zona 14
Zona 14	Zona 16	Zona 11	Pelota
Zona 15	Zona 14	Zona 11	Pelota
Zona 16	Pelota	Zona 11	Zona 14
Zona 17	Zona 16	Zona 11	Pelota
Zona 18	Zona 17	Zona 11	Pelota

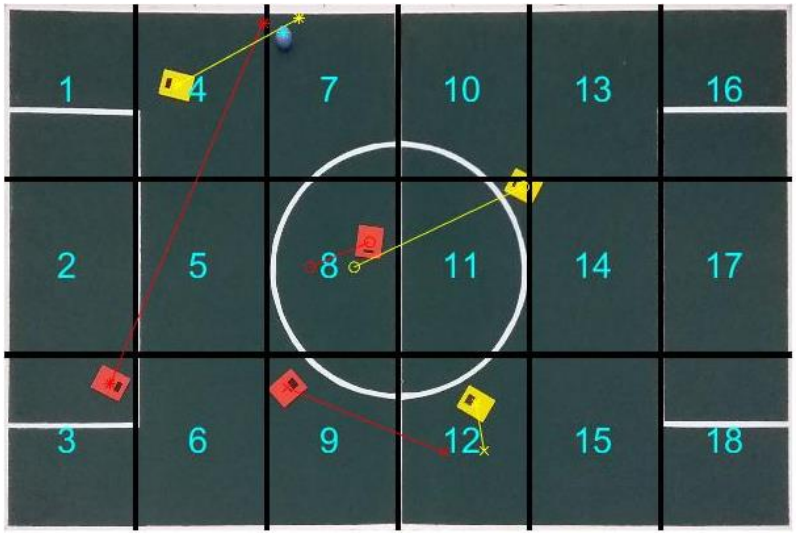


Figura 16. Estrategia de equipo con pelota en zona 7.

La figura 16 se muestra la pelota en la zona 7 y los robots que deben atacar la pelota son el número 2 de cada equipo, los robots número 1 ubicarse en zona 8 y los robots 3 esperar en zona 12.

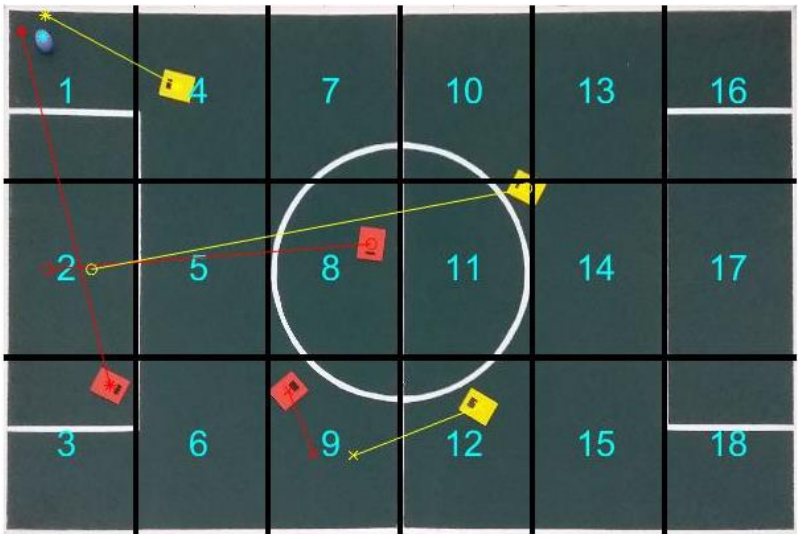


Figura 17. Estrategia de equipo con pelota en zona 1.

La figura 17 muestra la estrategia que se ejecuta cuando la pelota está en la zona 1 de juego, los robots que atacan son el número dos de cada equipo, mientras que los número uno esperan en zona 2 y los número 3 deben estar posicionados en zona 9.

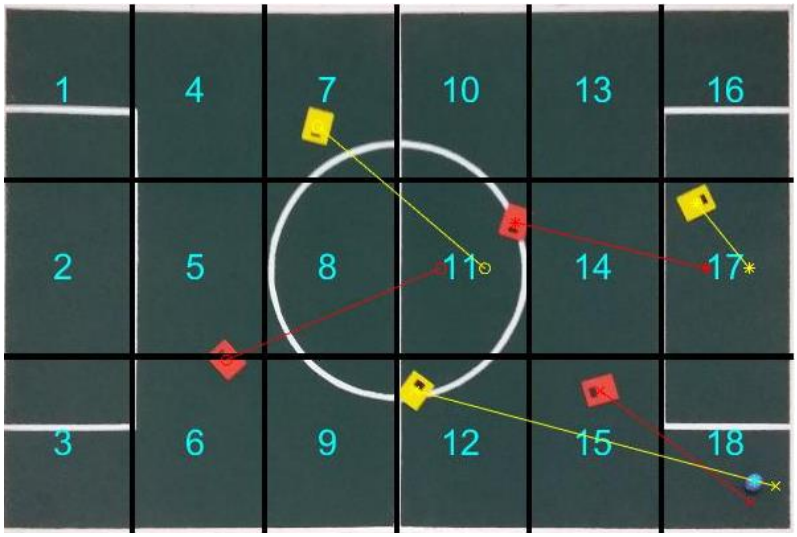


Figura 18. Estrategia de equipo con pelota en zona 18.

La figura 18 muestra la estrategia que se ejecuta con la pelota en la zona 18, la cual atacan los robots número tres de ambos equipos mientas que los número uno van a la zona 11 y los número dos en la zona 17.

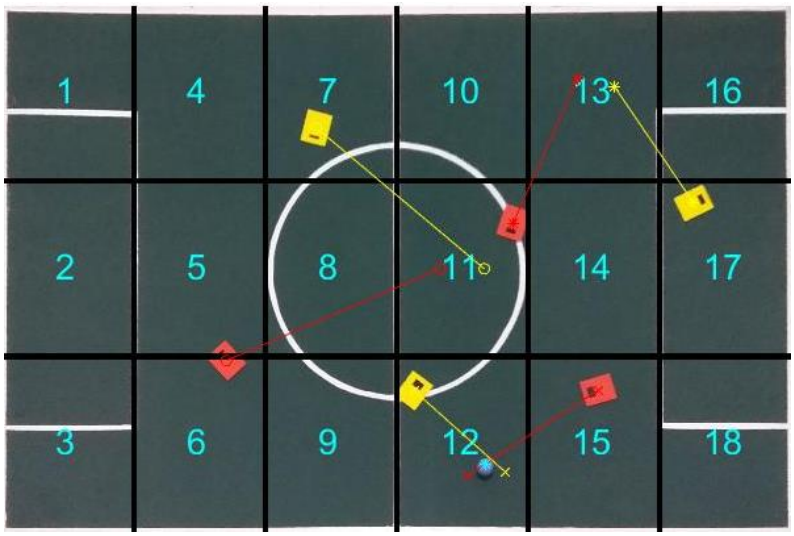


Figura 19. Estrategia de equipo con pelota en zona 12.

La figura 19 muestra el desarrollo del juego cuando la pelota se encuentra en zona 12, los robots número tres atacan la pelota, mientras los número dos ocupa la zona 13 y los número uno la zona 11.

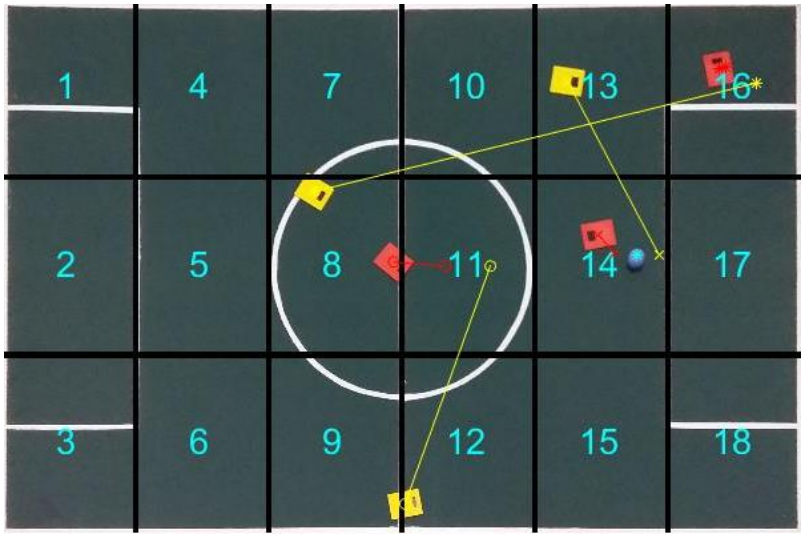
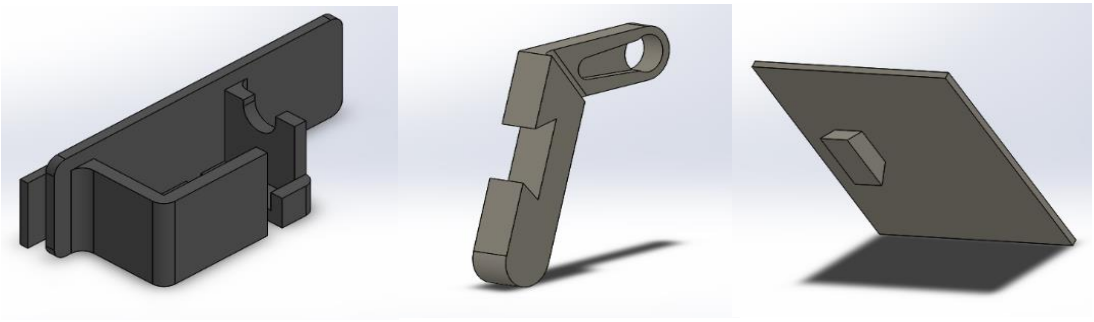


Figura 20. Estrategia de equipo con pelota en zona 14.

La figura 20 muestra el desarrollo del juego cuando la pelota se encuentra en zona 14, los robots número tres atacan la pelota, mientras los número dos ocupa la zona 16 y los número uno la zona 11.
(Ver código en anexos, "*Estrategia de juego con identificación de cada robot y pelota*")

5.2 PLATAFORMA ROBÓTICA LIBRE

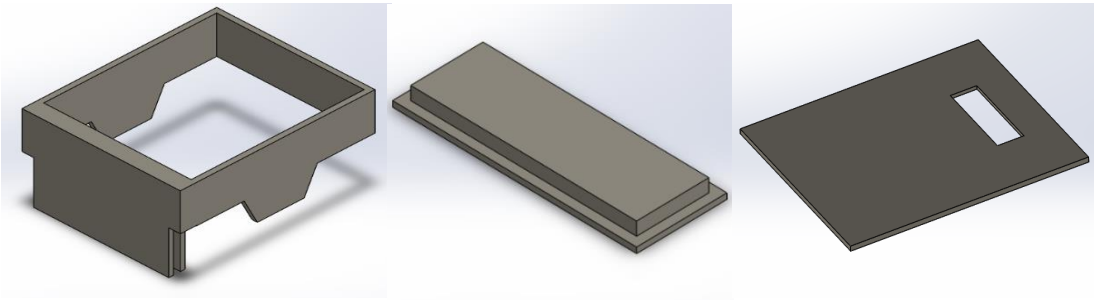
En esta sección se describe que plataforma robótica se escoge y los motivos, las modificaciones que esta lleva y las características de movilidad. SMARS "Screwless Modular Assemblable Robotic System" (Sistema robótico ensamblable modular sin tornillos), es un robot diferencial modular diseñado para el campo de la educación de bajo costo, al cual se le pueden montar o desmontar piezas, incluso permite montar piezas de diseño de terceras personas. El robot no necesita de tornillería, remaches ni de ningún tipo de pegantes para fijar herramientas a este. Dado a todas estas características que tiene el robot modular SMARS es buena opción para implementarlo. En este caso las modificaciones que se le deben hacer son tales que le permitan golpear una pelota.
Para esto se decide diseñar herramientas como: base para sostener un servo motor, una pala que servirá para golpear la pelota y un techo que pueda llevar las marcas distintivas de cada robot.



A B C

Figura 21. Diseño del sistema de pateo para implementar en el robot modular

En la figura 21A se muestra la base que sostendrá el servo motor, la figura 21B es el brazo que se acopla al servo motor y sostiene la pala, la figura 21C es la pala con la que se patea la pelota. Ver figura 29, sistema de pateo construido y ensamblado.



A B C

Figura 22. Diseño del techo que será implementado en el robot.

La figura 22A es la estructura que se acopla al chasis del robot, ver figura 28, la figura 22B es la marca que lleva el techo, la figura 22C es el techo que será amarillo y rojo, ver figura 26 y 27.

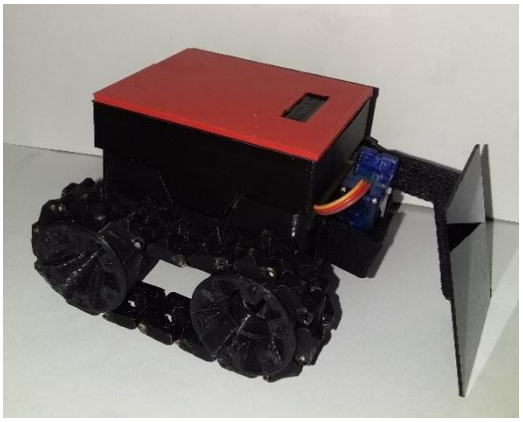


Figura 23. Ensamble terminado del robot.

El robot modular se compone de dos motores excéntricos y viéndolo de esta forma el robot no sería diferencial, para ser diferencial necesitaría que los dos motores estuvieran en línea, por este motivo se implementa el sistema de oruga ya que este sistema entra en el tipo de robot diferencial y así no se ve la necesidad de poner los motores en línea.

En la figura 23 se observa el techo del robot que llevará una marca, se diseñaron las marcas por CAD de tal manera que se pudieran imprimir en 3D y que su color negro fuera uniforme, también se pensó en colocar esa sección negra de áreas distintas y de un tamaño suficientemente grande para que el sistema de visión pudiera hacer una correcta identificación de cada una de ellas.

Para escoger los componentes electrónicos se hace una comparación con los artículos disponibles en el mercado y cada ventaja o desventaja que tendría cada uno. Se comienza con la placa de desarrollo que se debe elegir.

Tabla 3. Comparación de placas de desarrollo.

COMPARACIÓN ENTRE PLACAS DE DESARROLLO							
Placa	Pines I/O	Módulo wifi	Pines PWM	Velocidad de reloj (MHz)	Dimensiones (mm^2)	Voltaje de entrada (V)	Memoria flash (MB)
Wemos d1 mini	8	Sí	8	80 o 160	34.2x25.6	3 - 7	4
NodeMCU	10	Sí	10	80 o 160	48x26	5 - 9	4
ESP-01	4	Sí	4	80 o 160	25x14	3.3	1

Tabla 4. Comparación en driver puente H.

COMPARACIÓN DE DRIVER PARA MOTORES				
Driver	Voltaje de alimentación (V)	Corriente de salida por canal (A)	Dimensiones (mm^2)	Motores que controla
DRV8833	3 - 10	1.5	18.5x16	2
TB6612FNG	2.7 - 15	1.2	20.32x20.32	2
L298	2 - 10	1.5	24.7 x 21	2

Para escoger la placa de desarrollo que se empleará en el circuito del robot se tiene en cuenta la tabla 3, de la cual se concluye con la selección de la wemos d1 mini ya que esta asegura el mínimo de 5 pines I/O que no los tiene la ESP-01 y un tamaño más reducido que la NodeMCU. En cuanto al driver para el par de motores se escoge el DRB8833 pues es el de tamaño más reducido y cumple con la corriente nominal exigida por el motor N20 de 600 rpm que es de 70 mA con carga. La energía eléctrica del circuito es suministrada por una batería de 9v voltios a 250 mAh recargable, como la placa de desarrollo elegida no acepta como voltaje máximo de entrada 7v, entonces se incorpora al circuito un regulador de voltaje LM7805 como se muestra en la figura 24. Un micro servomotor se usa para el sistema de pateo

del robot.

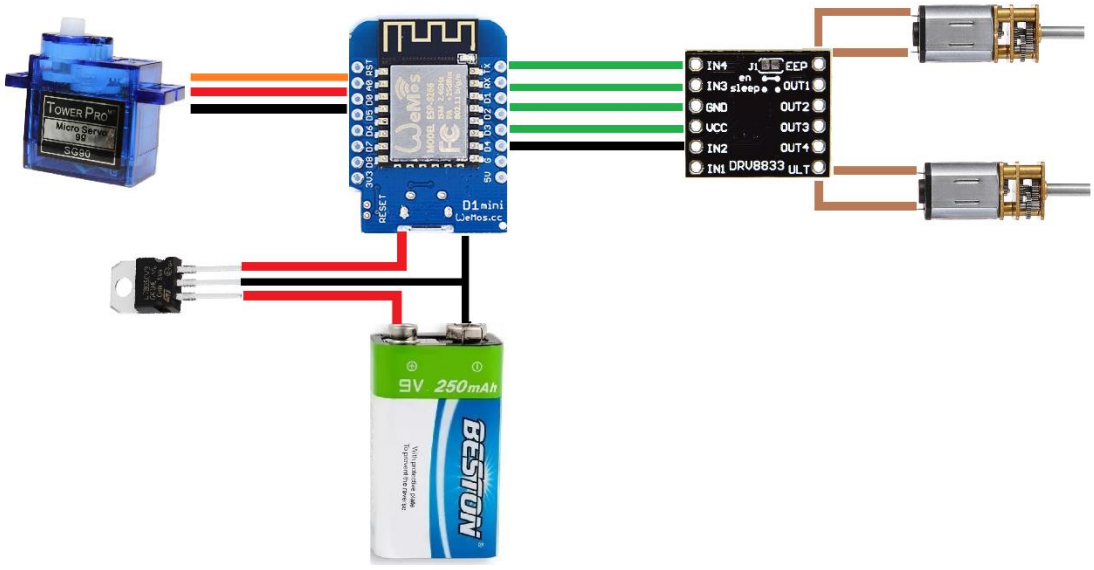


Figura 24. Esquema de conexiones electrónicas del robot.

Un último cambio también se hace necesario al diseño mecánico luego de hacer las pruebas de movimiento del robot, ya que, en su movimiento la oruga no permitió el movimiento del sistema porque quedaba pegada a la llanta, se intentó primero modificar los dientes de la rueda del robot, pero no se cumplió el objetivo de mover de manera fluida la rueda sin tanto esfuerzo por parte del motor. Entonces, se decide eliminar la oruga del sistema y dejar los motores en línea, para eso se necesitó hacer el rediseño del chasis del robot para ubicar de esta manera los motores, de resto todo siguió igual con respecto al robot, se pueden observar estos cambios en la figura 25.

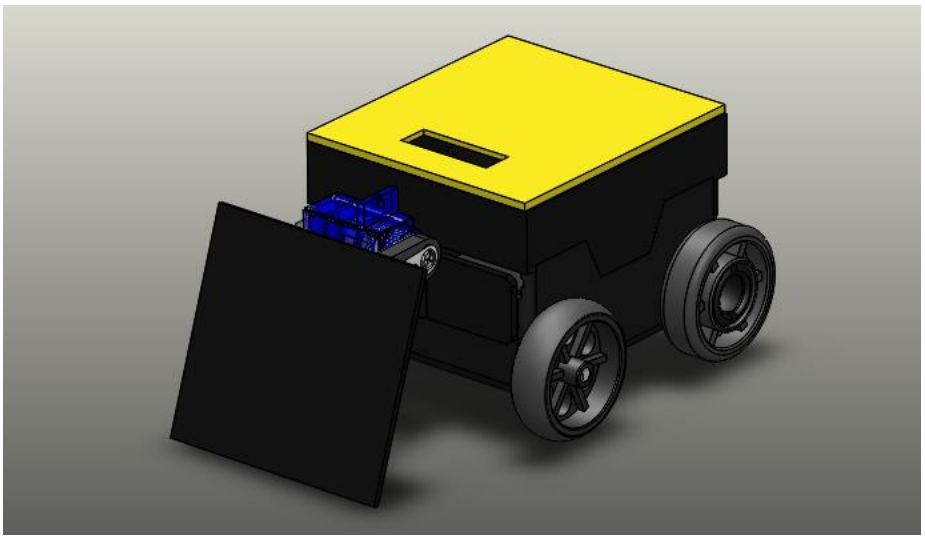


Figura 25. Rediseño del robot diferencial, con ruedas en lugar de oruga.

Este nuevo sistema les permitió un mejor movimiento a los robots en la prueba y de esta manera pudo llegar hasta el objetivo sin problemas mecánicos, se deja este diseño como definitivo para implementarlo en el sistema. El robot debe ser negro exceptuando la superficie superior del techo para que sus colores no interfieran en el tratamiento digital de las imágenes.

Luego de haber diseñado las piezas que se ensamblarán en el robot, se procede a imprimir en 3D todas las piezas del robot para construir los modelos. Las piezas impresas se muestran a continuación:

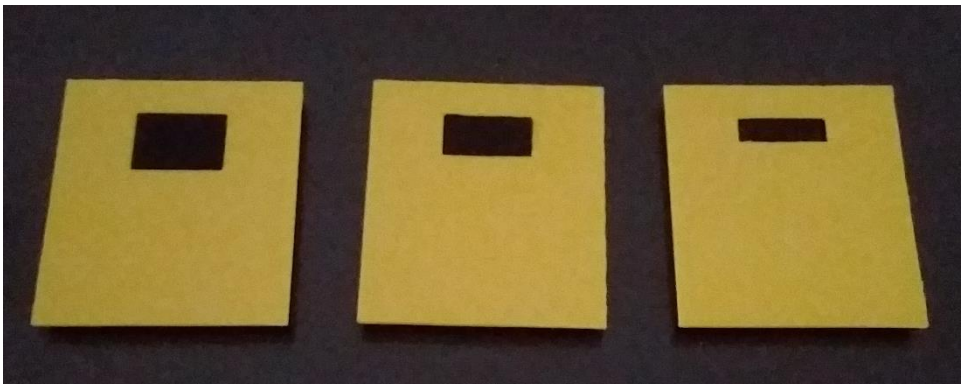


Figura 26. Techo amarillo.

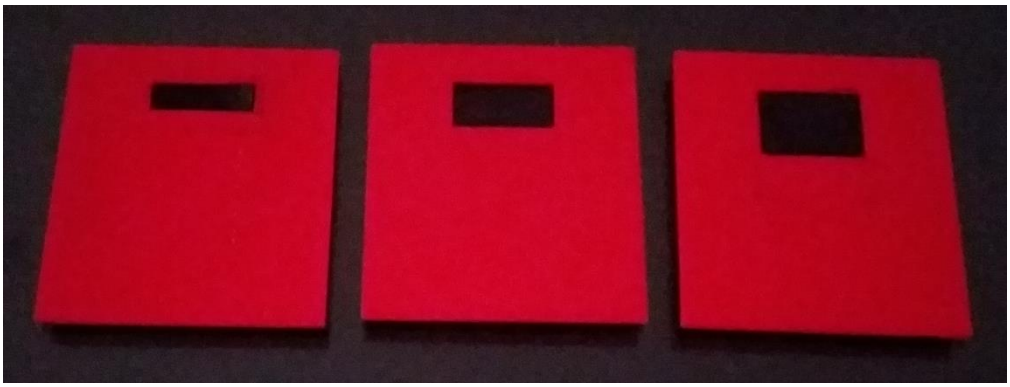


Figura 27. Techo Rojo

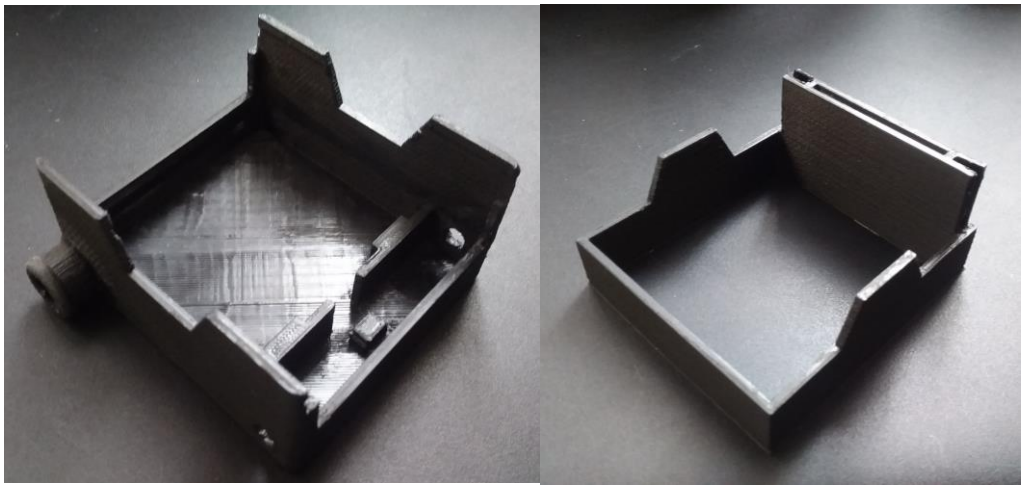


Figura 28. Chasis y soporte del techo para el robot.

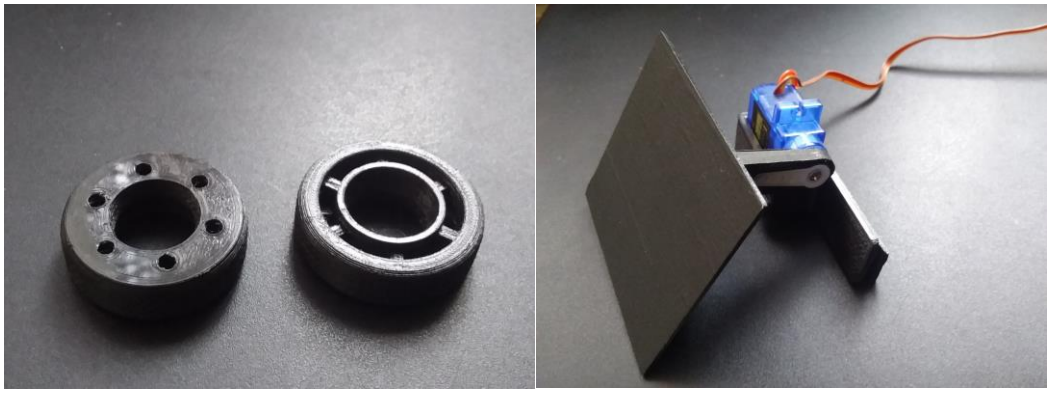


Figura 29. Ruedas esclavas y sistema de pateo.

Figura 30. Ruedas maestras.

Las ruedas de la figura 29 fueron impresas en 3D y las ruedas maestras que muestra la figura 30 fueron compradas y se pueden ensamblar en los ejes de los motorreductores N20.
Cuando se tienen las piezas impresas se procede a ensamblar cada una de ellas en su respectivo sitio. Primero se inicia ensamblando el sistema de pateo como se ve en la figura 31.

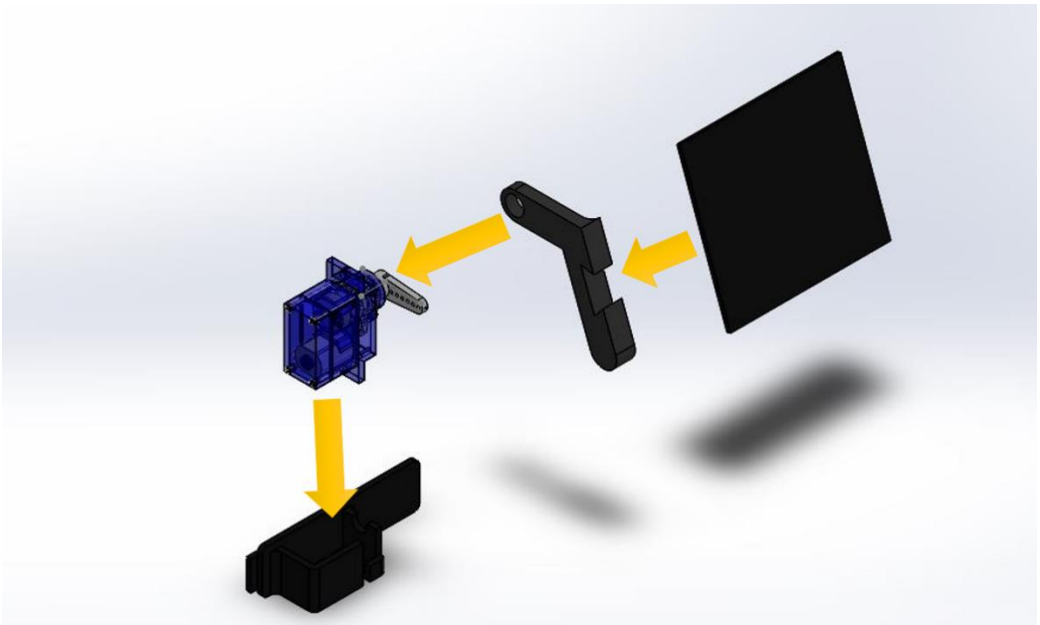


Figura 31. Ensamble del sistema de pateo.

Teniendo el sistema de pateo ensamblado se puede continuar ensamblando las demás piezas del robot, dando como resultado la figura 25.

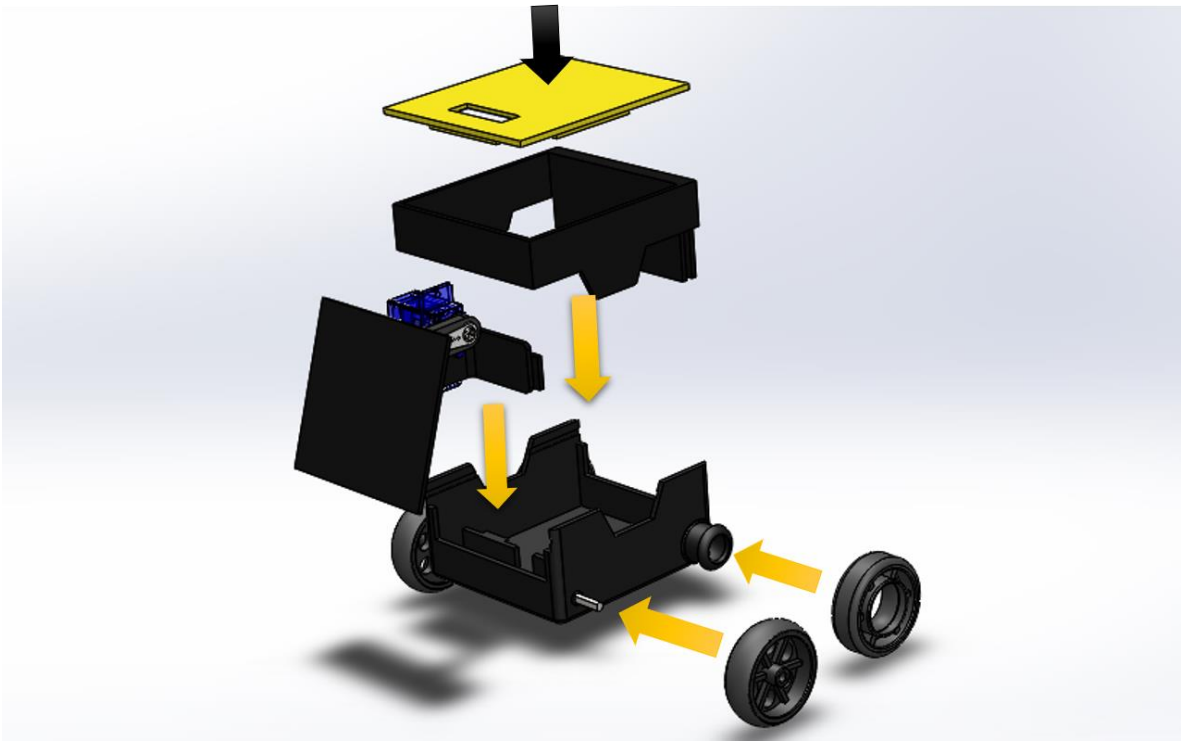


Figura 32. Ensamble de las piezas del robot.

5.3 PROGRAMACIÓN DEL ROBOTS E INICIACIÓN DEL SERVIDOR

En este apartado se describe paso a paso la manera en que se inicializa el servidor y la programación básica del robot. Crear el servidor web permite enviar y recibir datos en muchos lenguajes o aplicaciones por medio de un navegador web en diferentes tipos de protocolos. Se debe ajustar el protocolo a usar y se debe definir el tipo de transporte ya sea TCP (protocolo de control de transmisión), esta garantiza el flujo de datos sin errores o UDP (protocolo de datagrama de usuarios) no garantiza la llegada de datos sin errores. Se decide usar el tipo de transporte TCP por red IP asignada a cada robot para el correcto envío y recepción de datos. Entonces, Las peticiones al servidor web se harán mediante el protocolo HTTP la cual usa el puerto TCP 80 para este caso.

5.3.1 INICIACIÓN DEL SERVIDOR Y CREACIÓN DEL DOMINIIO WEB

El siguiente código debe ser almacenado en la tarjeta de desarrollo del robot, para eso, el programa empieza incluyendo la librería imprescindible.

```
#include <ESP8266WiFi.h>
```

El servidor solo responde a los clientes (navegadores web) en el puerto 80, que es un puerto estándar en el que los navegadores se comunican con servidores web.

```
WiFiServer server(80);
```

Conectar a una red WiFi es sencillo, para eso se debe definir el nombre de la red y la contraseña de esta manera:

```
// Datos de red
const char* ssid = "*******";       // Nombre de la red
const char* password = "*******";   // Contraseña de la red de WiFi
```

Luego de definir los datos de la red, se debe conectar a la red de internet WiFi e iniciar el servidor web:

```
WiFi.begin(ssid, password); // Ingresar a la red WiFi
while (WiFi.status() != WL_CONNECTED)
   {
       delay(500);
       Serial.print(".");
   }
server.begin(); //Inicia el servidor

Serial.println(WiFi.localIP()); // Impresión de la IP con la que se
                                // debe ingregar al host
```

Después de la iniciación se debe mostrar en el monitor serial algo similar a lo siguiente:

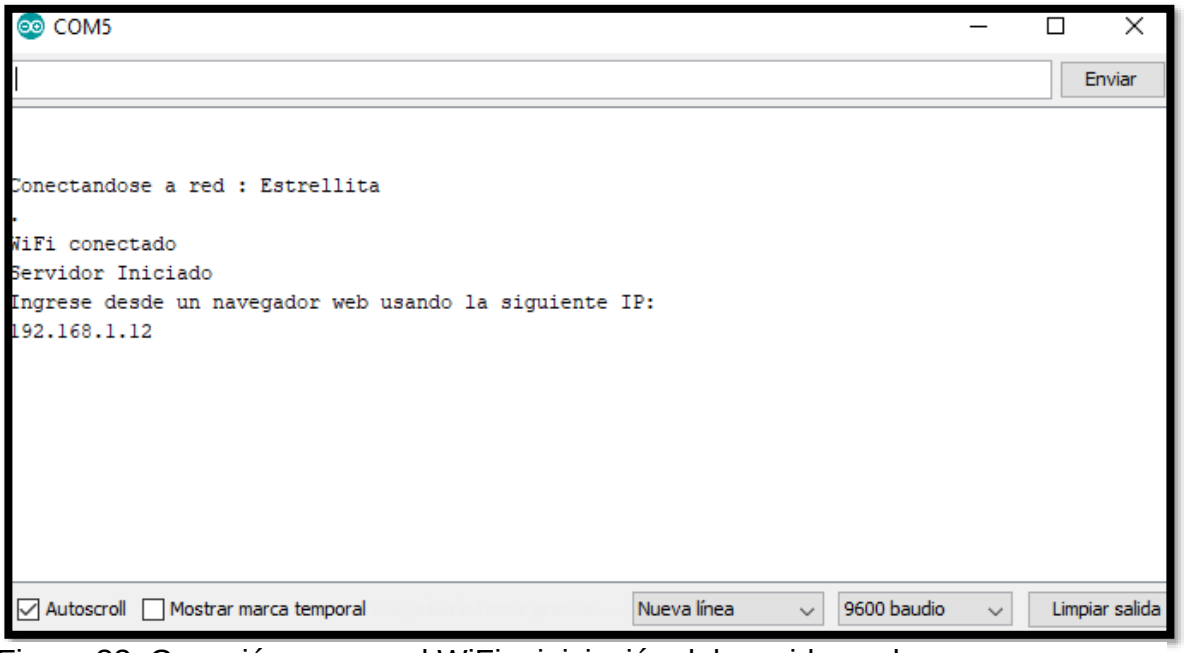


Figura 33. Conexión a una red WiFi e iniciación del servidor web.

Seguido de iniciar el servidor web, se debe comprobar la existencia de un cliente conectado para el envío y recepción de datos, en el módulo WiFi ESP8266 se debe ubicar de la siguiente manera:

```
WiFiClient client = server.available();
  if (client) //Si hay un cliente presente
  {
   Serial.println("Nuevo Cliente");
    //esperamos hasta que haya datos disponibles
    while(!client.available()&&client.connected())
    {
    delay(1);
    }
    // Leemos la primera línea de la petición del cliente.
    String linea1 = client.readStringUntil('r');
   .
   .
   .
   client.flush();//Espera hasta que se hayan enviado todos los
                  //caracteres salientes del buffer.
  }
```

Se crea una interfaz de en la página web con algunas características como el encabezado y el cuerpo de la página web.

```
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/html");
    client.println("Connection: close");// La conexión se cierra
                                        //después de finalizar de la
                                        //respuesta
    client.println();
    //Pagina html  para en el navegador

    client.println("<!DOCTYPE HTML>");
    client.println("<html>");
    client.println("<head><title>NOMBRE DE LA PESTAÑA WEB </title>");

    client.println("<body>");//etiqueta cuerpo de la página

    client.println("<h1 align='center'>TÍTULO DE LA WEB </h1>");
    client.println("<div style='text-align:center;'>");
    client.println("<br />");                // deja un renglón
    .
    .
    .
    client.println("</div>");
    client.println("</body>");
    client.println("</html>");
```

En este proyecto se implementan botones y formularios los cuales se pueden dejar por defecto sencillos o agregarles colores, bordes y muchos más.
La implementación de botones se puede hacer de la siguiente manera:

```
client.println("<button onClick=location.href='./dato>"
Nombre"</button>");
```

Un botón sencillo donde solo se coloca el nombre y el tipo de dato que va a enviar:
Para insertar un formulario o caja de texto, se hace de la siguiente manera:

```
client.println("<form/ >"); // LINEA PARA ESTABLECER FORMULARIO Y
ENVIAR DATOS TIPO "SUBMIT"
client.println("<label for=\"T\">Introducir tiempo: </label>");
client.println("<input type=text name=\"T\" placeholder=\"tiempo\"
id=\"T\" SIZE=\"10\" maxlength=\"4\">");
client.println("<input type=submit value=\"enviar\"</button>");
client.println("<form/>");
```

El primer resultado de diseño de la página web para controlar los movimientos de cada robot queda de la siguiente manera (ver anexos, “*Programación del servidor web*”):

Figura 34. Diseño de página web con botones y formulario para control de los movimientos del robot durante un tiempo específico.

5.3.2 PROGRAMACIÓN DEL ROBOT DIFERENCIAL

Una vez definida la página web con cada botón que ejercerá una acción de control en el robot se procede a programar el robot y para eso debe incluir las librerías de servo y la del módulo de control esp8266.

```
#include <ESP8266WiFi.h>    //Librería ESP8266
#include <Servo.h>          //Librería de servomotor
```

El robot consta de dos motores y un servomotor como elementos finales de control, los cuales deben ser declarados dentro del programa. Se debe aclarar que los pines I/O de la placa de desarrollo WEMOS tienen nombres diferentes dentro del IDE de

ARDUINO y se declaran de la siguiente manera:

```
// ROBOT
int Motor_Ib=14;  //D5
int Motor_Ia=5;   //D1
int Motor_Da=4;   //D2
int Motor_Db=0;   //D3
Servo Patada;
```

Tabla 5. Pines I/O Wemos en IDE Arduino.

I/O WEMOS	I/O ARDUINO
D0	16
D1	5
D2	4
D3	0
D4	2
D5	14
D6	12
D7	13
D8	15
Tx	3
Rx	1

Cabe resaltar que el pin digital (D4 o 2) es quien activa el servidor.
Como los datos de control provienen de un protocolo de control de transmisión vía internet se debe extraer ese dato tipo **String** y decodificarlo para obtener el movimiento que debe hacer el robot y el tiempo por el cual se va a mover el robot. Para esto se usa la siguiente línea de código que lee un dato hasta el final de la línea de código:

```
String linea1 = client.readStringUntil('r');
```

```
 char TIEMPO_E[3]; // creación de un vector de 4 posiciones
 int TIEMPO_N;  // Variable que contendrá el valor numérico del tiempo
                // descomposición del vector
    TIEMPO_E[0] = linea1[8];
    TIEMPO_E[1] = linea1[9];
    TIEMPO_E[2] = linea1[10];
    TIEMPO_E[3] = linea1[11];
    TIEMPO_N = atoi(TIEMPO_E); //Se emplea de la palabra reservada
                               //atoi para convertir de string a int

if (linea1.indexOf("IZQUIERDA")>0) //Busca la palabra "IZQUIERDA"
    {
                // Control sobre el robot
      digitalWrite(Motor_Ib,LOW);
      digitalWrite(Motor_Db,LOW);
      digitalWrite(Motor_Ia,HIGH);
      digitalWrite(Motor_Da,HIGH);
      Serial.println("va atras");
      delay(TIEMPO_N); //Tiempo en el que el robot se va a mover
    }
```

Luego de tener el dato proveniente del cliente se procede a decodificarlo y para eso se hace uso de la descomposición de vectores para hallar el tiempo, los cuales se deben convertir a tipo entero *int*, y también uso de una palabra reservada por ARDUINO para encontrar datos tipo *char* o *String* dentro de otro *String.*

5.4 CONTROL SERVO VISUAL

Para el control servo visual que se pretende plasmar en este trabajo se hace uso de todo lo descrito en las secciones anteriores. Cada imagen es tomada desde una cámara ubicada a un costado lateral de la cancha, a una altura de 1.8 metros. Esa imagen es enviada vía wifi al ordenador que se encargará de analizar las características visuales que tiene la imagen, ubicando cada robot, la pelota, y la posición y orientación deseada que debe tomar cada robot dentro de la cancha. Una vez que se analiza la imagen, extrayendo los parámetros de control para cada robot, se procede a enviar la información mediante la comunicación wifi que se estableció entre el ordenador y los robots a través del servidor web TCP/IP. Los datos enviados a los robots son la posición y orientación del mismo con relación a la pelota, este dato es descompuesto y analizado por el robot para la correcta ejecución de los

movimientos dentro de la estrategia empleada (figura 24).
Se implementa en el robot un control proporcional a la entrada del error con respecto a la distancia y el ángulo relativo entre el robot y la pelota.

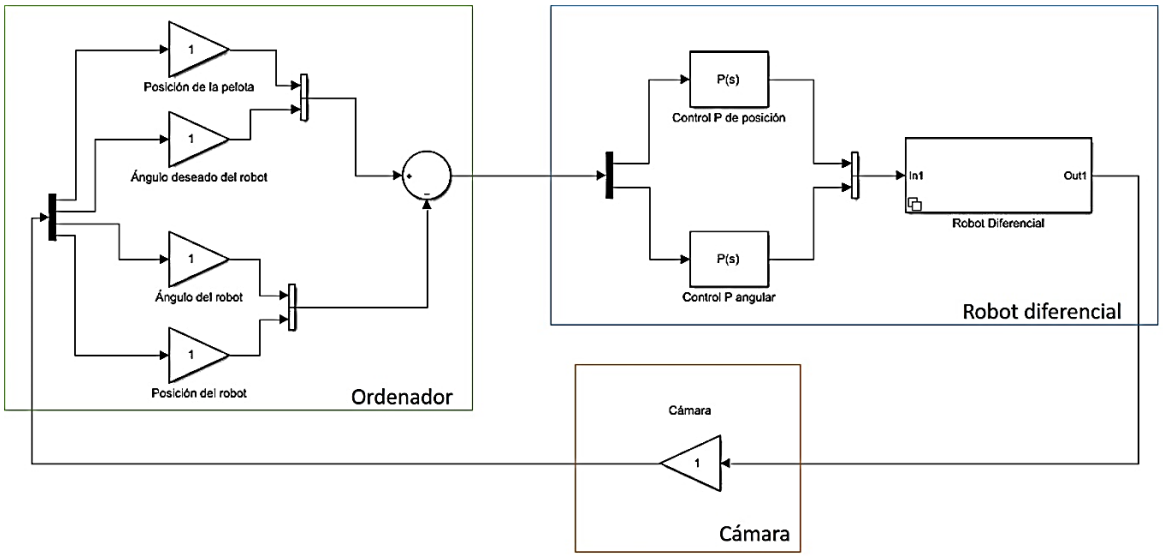


Figura 35. Esquema de control servo visual.

Con este esquema de control P se garantiza una corrección del ángulo de error en la trayectoria que debe seguir el robot y la disminución de la velocidad a medida que llega a la pelota y pueda ubicarse y patearla hacia adelante.

5.5 ANÁLISIS DEL DESARROLLO DEL PROYECTO

En este trabajó de desarrolló un control servo visual basado en imágenes 2D con cámara fija. Se analizaron cada una de las características de las imágenes entrantes para ubicar la zona de juego, cada robot y la pelota. El ambiente estructurado cuenta con una luz de ambiente sin cambios, esto ayudó a realizar una detección de los objetos bajo los mismos parámetros de análisis. A medida del movimiento realizado por cada robot se calcula el ángulo y la distancia faltante para llegar al objetivo. Los datos son analizados y pasados al robot para la corrección angular y corrección de velocidades para llegar al robot. A continuación, se muestra la evolución que tiene un robot al seguir la ruta trazada hasta llegar a la pelota.

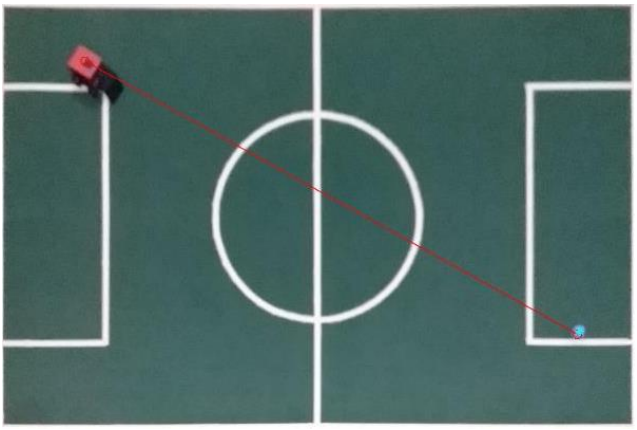


Figura 36. Posición inicial del robot con respecto a la pelota

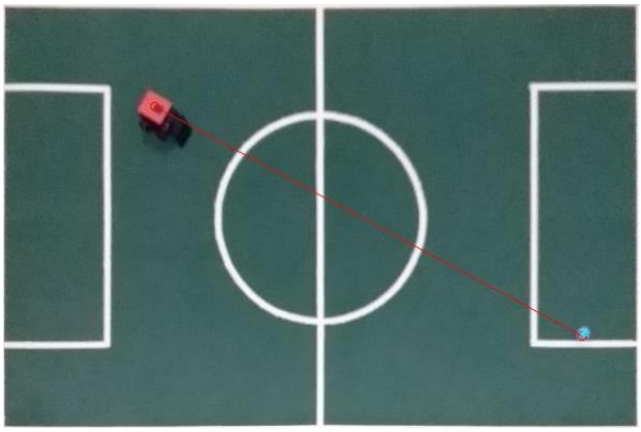


Figura 37. Avance del robot

En la figura 29 se muestra la posición inicial del robot y la pelota en la cancha, el algoritmo verifica que el robot se encuentre orientado hacia la pelota y ordena que avance, como se muestra en la figura 30 que ha avanzado rumbo a la pelota.

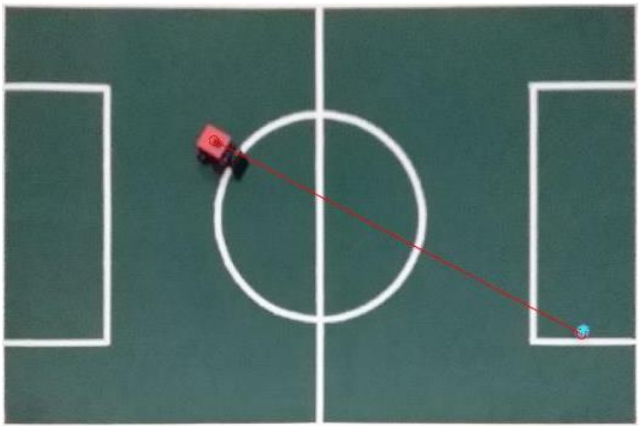


Figura 38. Seguimiento del robot.

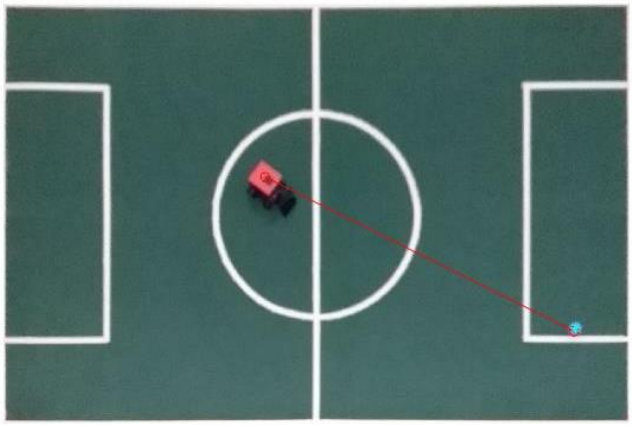


Figura 39. Desviación del angular del robot.

La figura 31 muestra el desarrollo del movimiento del robot a medida que se desarrolla el algoritmo de alto nivel, la figura 32 deja ver que el robot debe corregir el ángulo que tiene y ese ángulo se ve corregido en la figura 33. Luego que se corrige el ángulo, el robot procede a seguir su trayectoria hacia adelante tal como se observa en la figura 34.

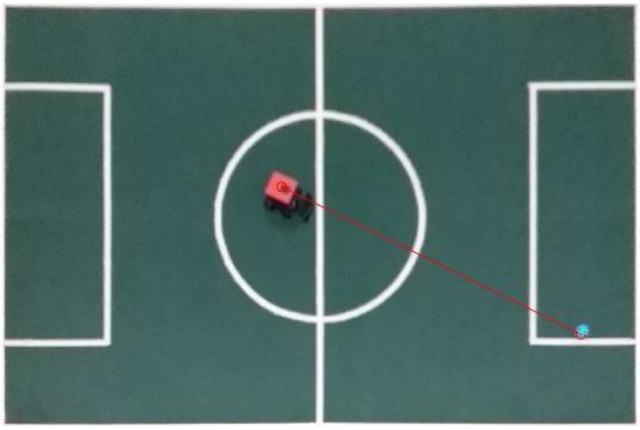


Figura 40. Corrección de angulo.

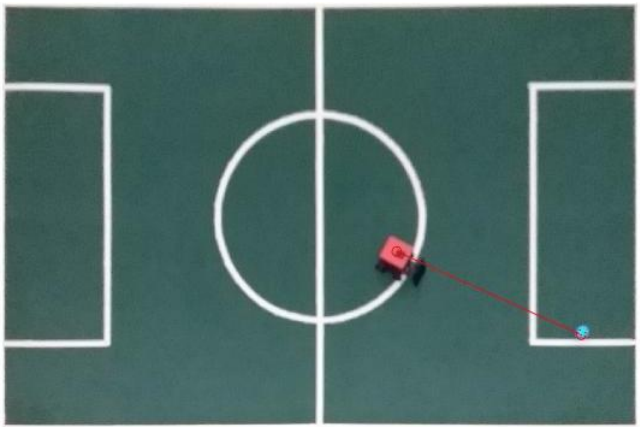


Figura 41. Avance del robot para llegar a la pelota.

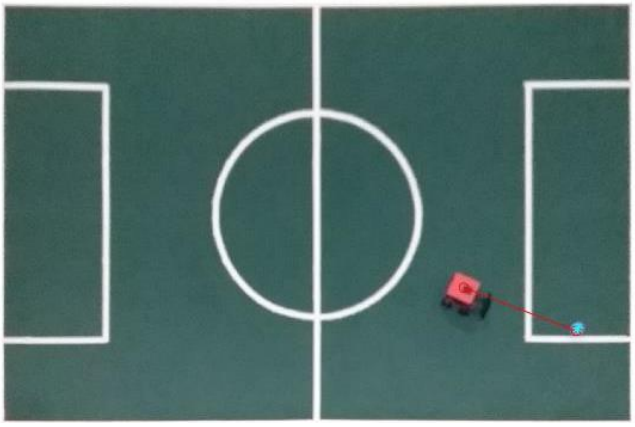


Figura 42. Robot a poca distancia de la pelota.

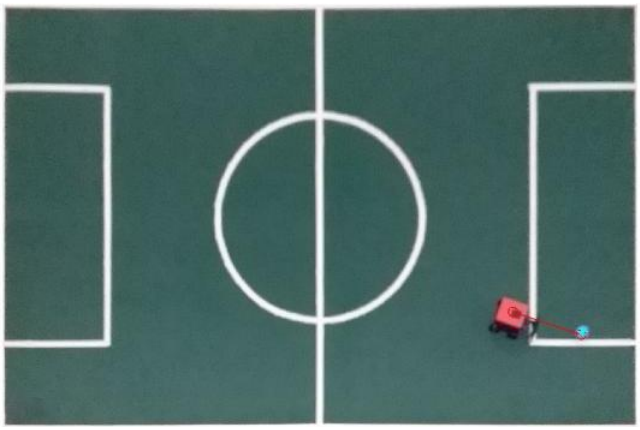


Figura 43. Error angular del robot con respecto a la pelota.

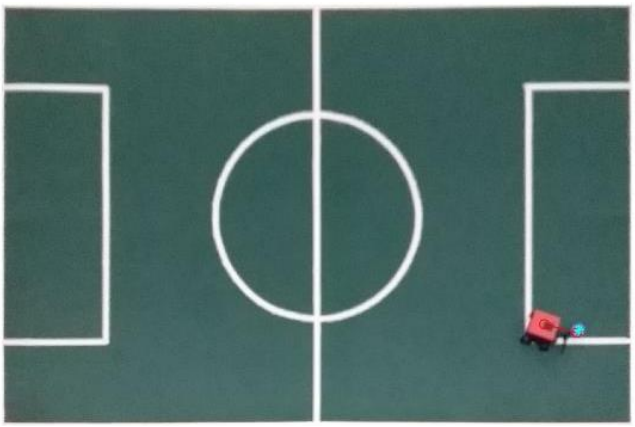


Figura 44. El robot llega a la pelota con error de angulo.

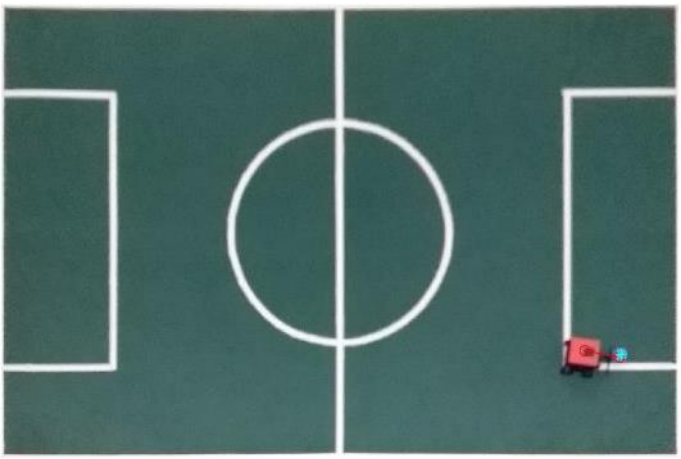


Figura 45. Corrección de ángulo para tener la pelota de frente.

En las figuras 35, 36 y 37 el robot hace el acercamiento final a la pelota para tenerla en el punto de pateo, pero se puede notar que tiene un ángulo desviado ligeramente que se debe corregir, esta corrección se refleja en la figura 38, quedando la pelota de frente y así lograr golpear la pelota hacia otro punto de la cancha.

La figura 39 muestra la trayectoria deseada en comparación con la trayectoria que realizó el robot.

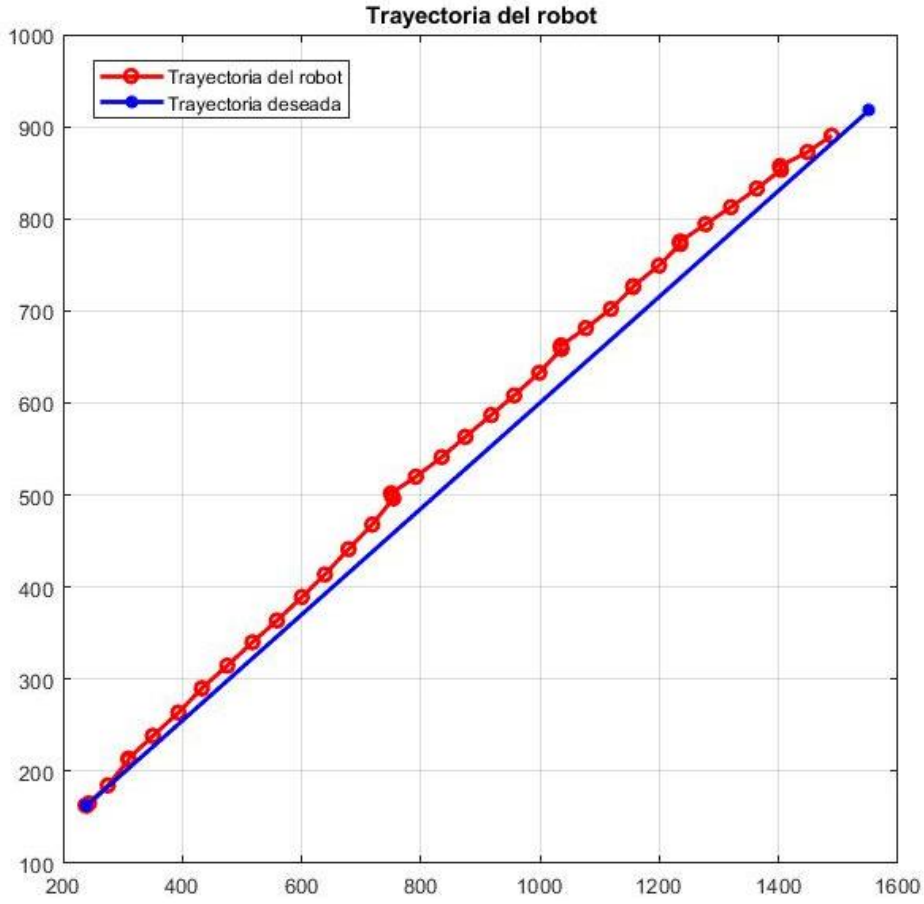


Figura 46. Trayectoria deseada vs trayectoria real.

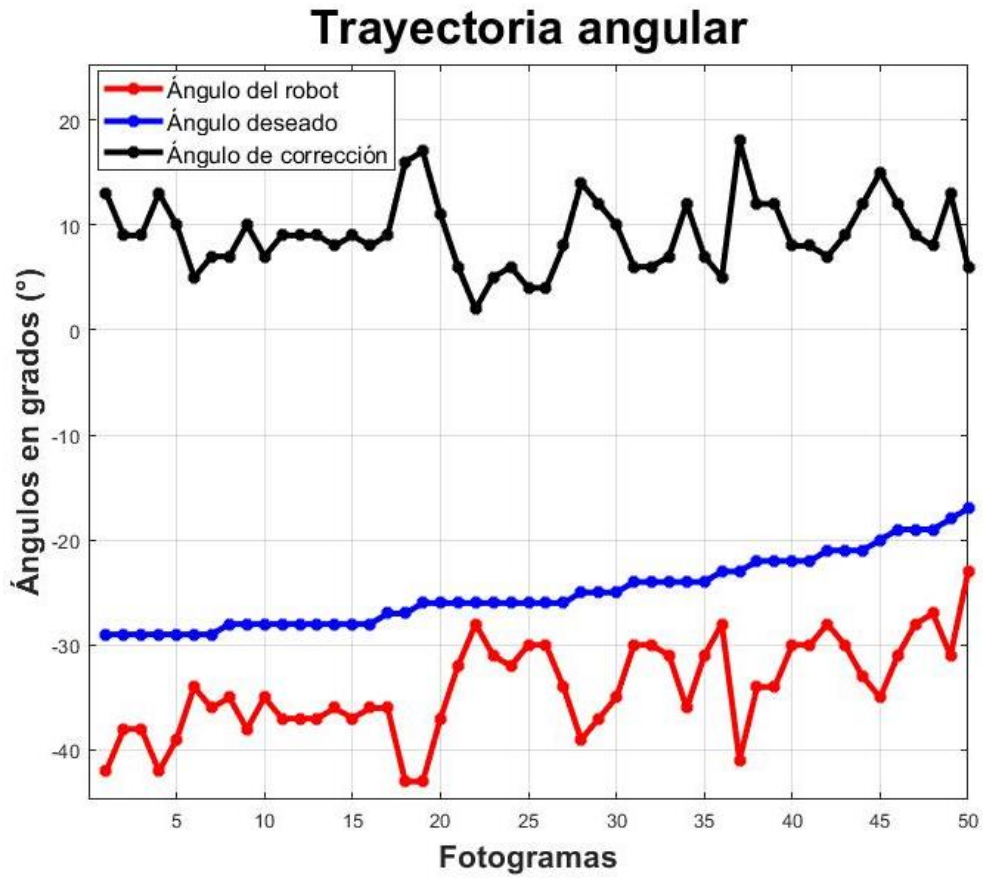


Figura 47. Evolución angular del robot durante la trayectoria.

La figura 40 muestra el desarrollo angular del robot donde la línea de color rojo representa el ángulo que tiene el robot, la línea azul señala el ángulo deseado por el sistema que debe tomar el robot y la línea negra simboliza el ángulo de debe girar el robot desde su ángulo actual para alcanzar el ángulo deseado. Se debe aclarar que el robot solo corrige ángulos mayores a 10 grados.

Para hacer más eficiente el algoritmo se realizan varios pasos para la configuración del programa, la detección de las esquinas de la cancha, los parámetros de la homografía y los umbrales de color de cada objeto se realizan una vez antes de iniciar el juego. Almacenadas todas esas constantes se usan para general el algoritmo de control de alto nivel que se encarga extraer el ángulo y posición del robot con respecto a la pelota y enviarlo a cada robot. Estos análisis se realizan dentro de un ciclo de repeticiones donde se les aplica la homografía a las imágenes entrantes con los cálculos previamente realizados.

RESULTADOS

Los experimentos del control de alto nivel fueron llevados a cabo en una cancha de largo 180cm y 120cm de ancho. Primeramente, se prueba con un equipo de robot siguiendo la pelota según sea la zona en que esta se encuentre.

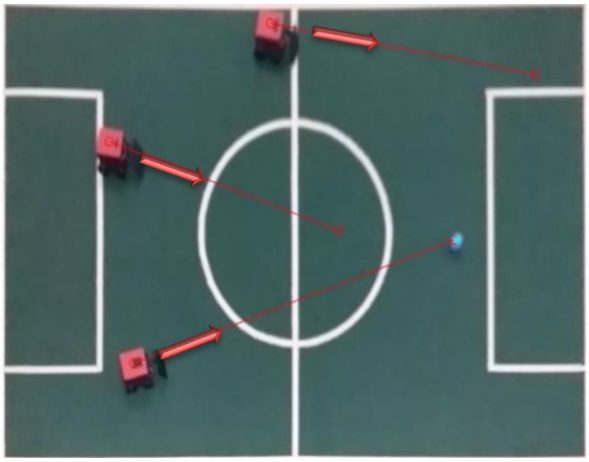


Figura 48. Posición inicial de los robots.

Se puede observar en la figura 41 la posición inicial del equipo rojo de robots con una línea trazada hasta la posición deseada que deben estar, siendo el robot 3 quien debe patear la pelota. El robot 1 está ubicado en el centro de la cancha, el robot 2 va por el costado izquierdo y el robot 3 se mueve por la parte derecha de la cancha.

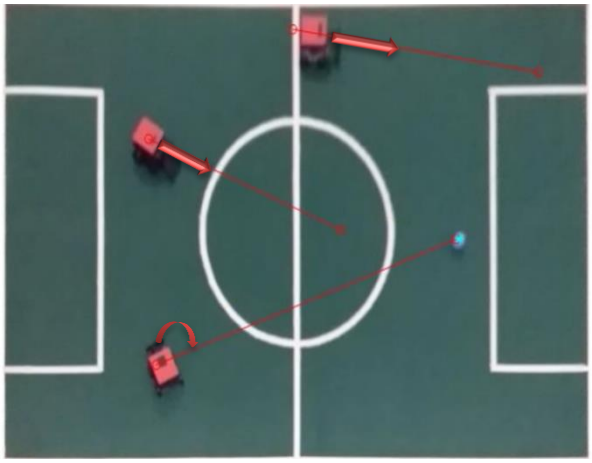


Figura 49.Corrección de ángulos y avance 1.

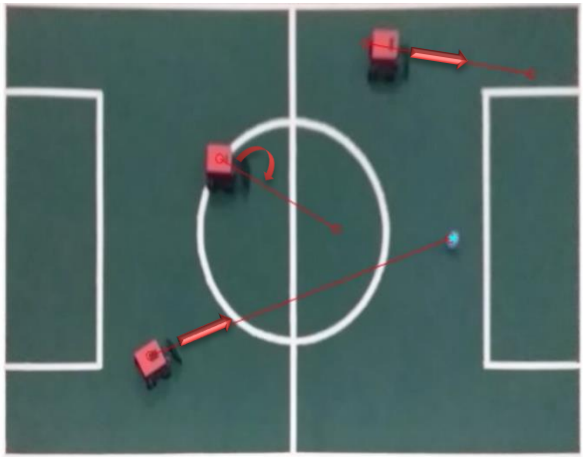


Figura 50. Corrección de ángulos y avance 2.

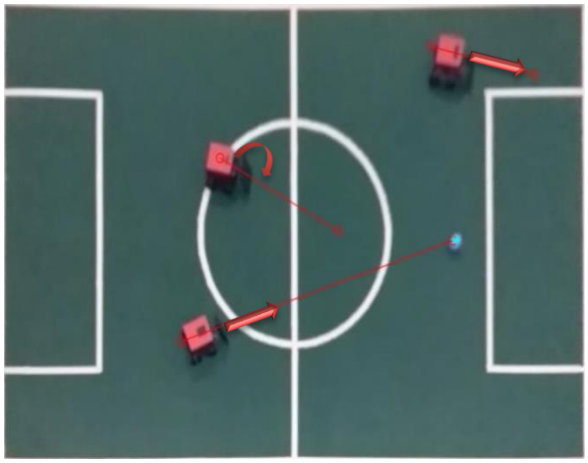


Figura 51. Corrección de ángulos y avance 3.

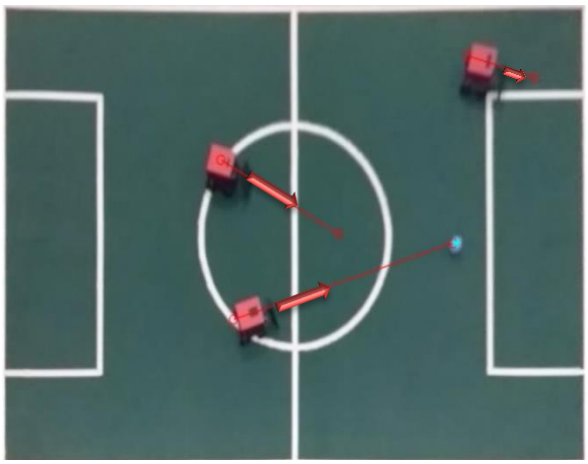


Figura 52. Corrección de ángulos y avance 4.

En las figuras 42, 43, 44 y 45 se puede apreciar el avance que tiene cada robot hacia la zona que debe llegar, y los ajustes de ángulos para seguir la trayectoria.

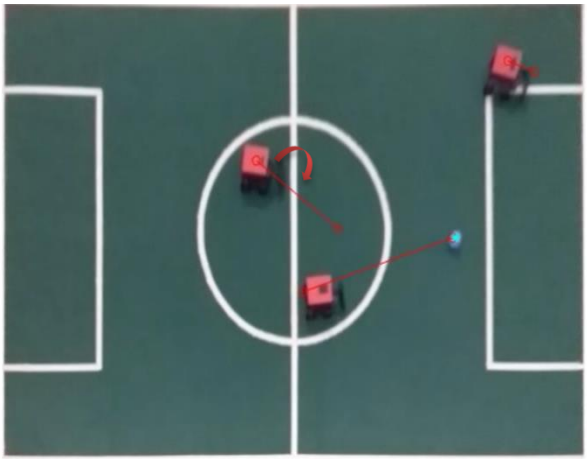


Figura 53. Corrección de ángulos y avance 5.

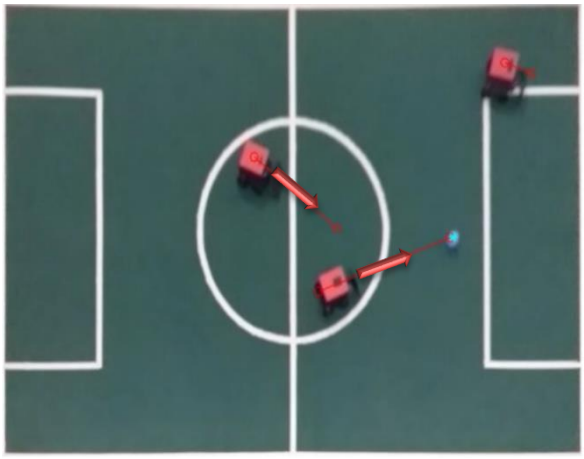


Figura 54. Corrección de ángulos y avance 6.

Las figuras 46 y 47 se nota que el robot 2 ya está en la zona correspondiente, mientras el robot 1 y 3 ajustan ángulos para continuar moviéndose.

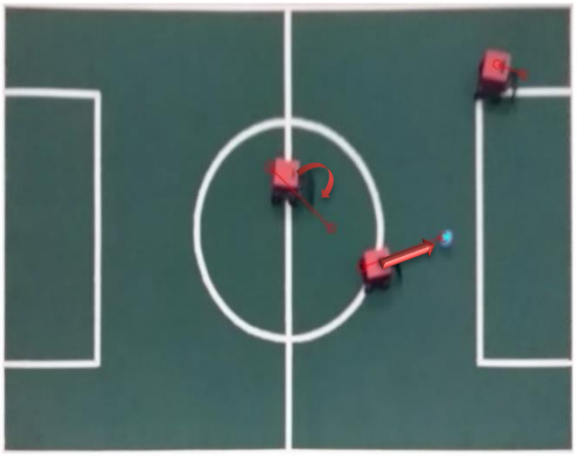


Figura 55. Corrección de ángulos y avance 7.

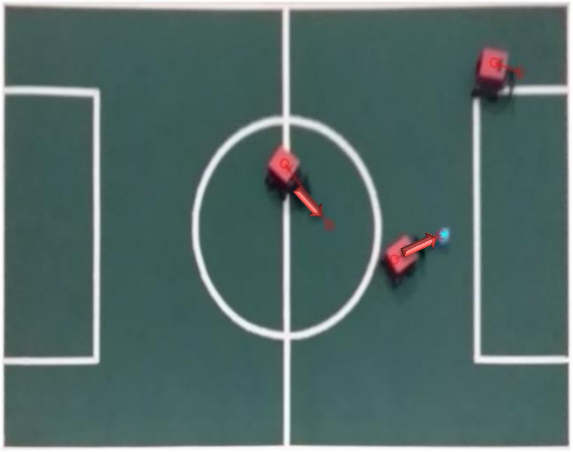


Figura 56. Corrección de ángulos y avance 8.

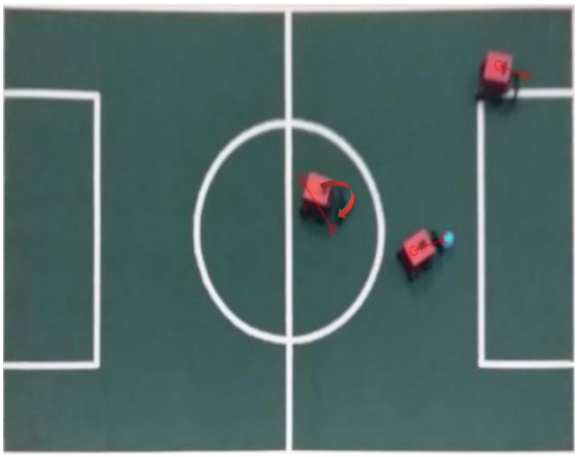


Figura 57. Robots ubicados en zona y pateo de pelota.

Las figuras 48 y 49 detallan el desarrollo del juego de la manera en cómo los robots hacen las aproximaciones a los lugares en que deben ubicarse, haciendo una corrección de ángulos cuando están desorientados y en caso que lleguen a la zona antes que los otros jugadores esperan hasta que haya un cambio de posición de la pelota y se le indique que debe avanzar, este es el caso del robot 2 que una vez que llega a la zona exigida por el algoritmo, espera quieto. La figura 50 se ve que el robot 3 llega la pelota y se dispone a patearla, mientras que el robot 1 debe hacer una corrección en su ángulo.

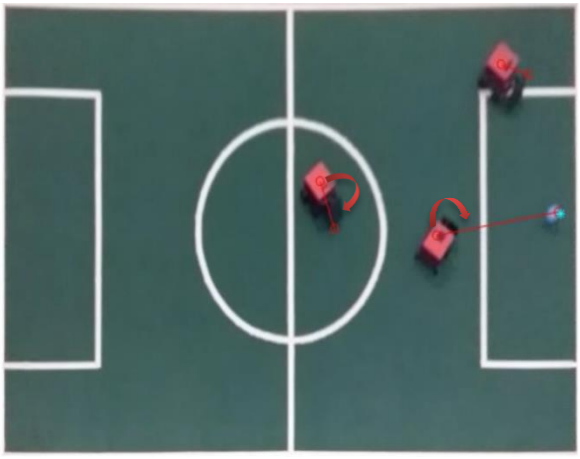


Figura 58. Corrección de ángulos y avance 9.

La figura 51 es posterior al pateo de la pelota, donde se puede notar el cambio de la ubicación de la pelota dentro del terreno de juego.

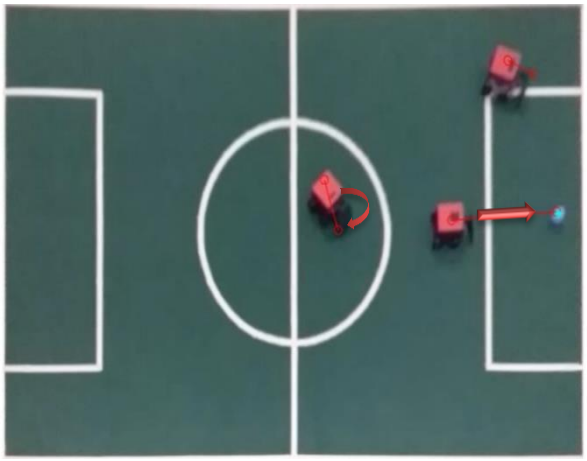


Figura 59. Corrección de ángulos y avance 10.

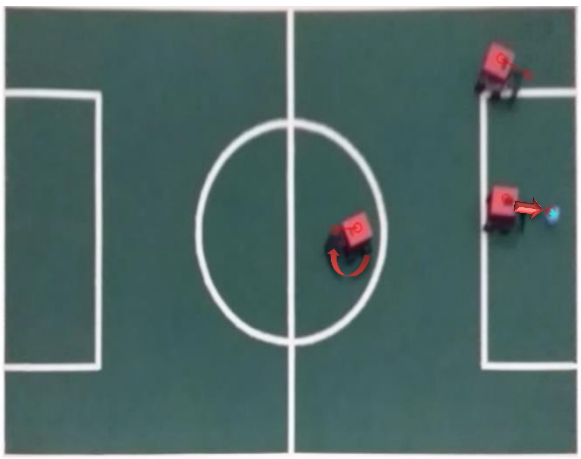


Figura 60. Corrección de ángulos y avance 11.

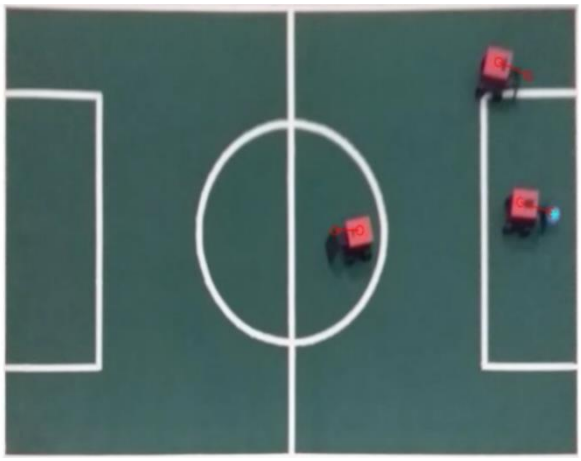


Figura 61. Pateo final de la pelota.

En las figuras 52 y 53 se continua con la supervisión del juego y se nota las correcciones que el algoritmo le indica a los robots que deben hacer y el acercamiento que el robot 3 le hace hacia la pelota. La figura 54 muestra el acercamiento final del robot 3 a la pelota previo al golpe que le da a la pelota.

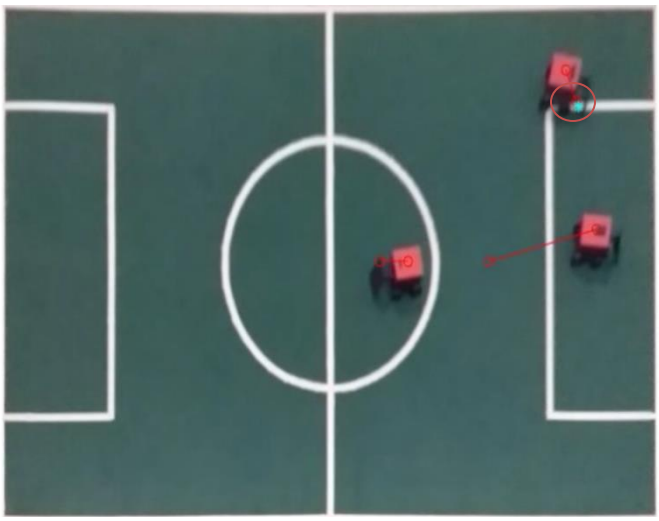


Figura 62. Error de detección.

La figura 55 muestra un error de detección de la pelota, un fotograma después de que la pelota saliera de la cancha detecta la pelota junto al robot 2. Ver anexos, *Control grupal*, para conocer el código con el que se le hizo control al grupo de tres robots.
Terminada la prueba de equipo se procede a hacer un juego entre dos equipos, rojo y amarillo. Este juego se realiza solo con dos robots por cada equipo, dado que el algoritmo no evita obstáculos y se pueden generar colisiones.

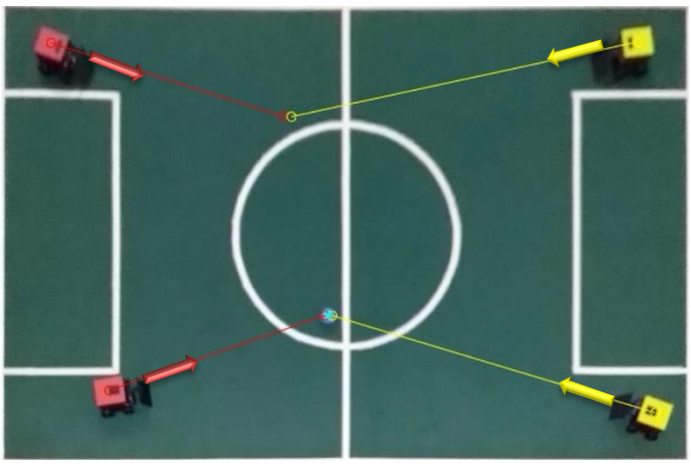


Figura 63. Juego de fútbol entre dos equipos de robots.

En la figura 56 se muestra la posición inicial de los equipos de fútbol robótico, de la pelota y la trayectoria que deben seguir los robots para poder atacar la pelota.

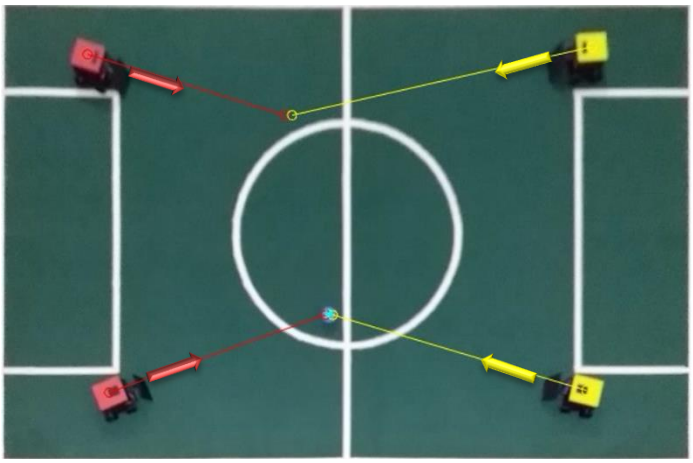


Figura 64. Corrección inicial de ángulos por cada robot.

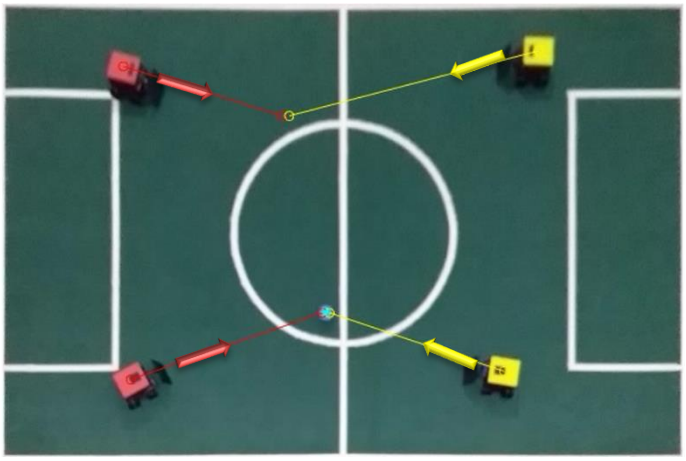


Figura 65. Avance de los equipos hacia la pelota y zonas de espera.

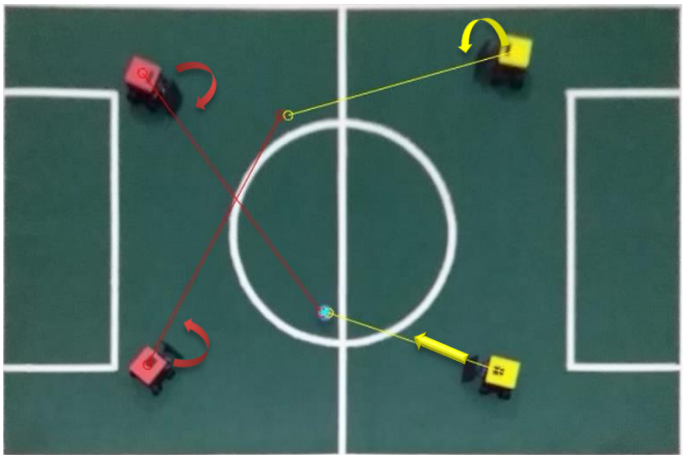


Figura 66. Error de detección del equipo rojo.

Las figuras 57 y 58 la corrección de los ángulos que deben hacer los robots inicialmente para dirigirse a las zonas ya sea de espera o de ataque. Se nota también el avance de los robots hacia la pelota. La figura 59 describe un error en la detección de los rangos del equipo rojo. El robot 1 va por la banda izquierda y el robot 2 por la banda derecha pero el sistema en ese fotograma los identifica mal, pero ya la figura 60 muestra la detección correcta de los equipos.

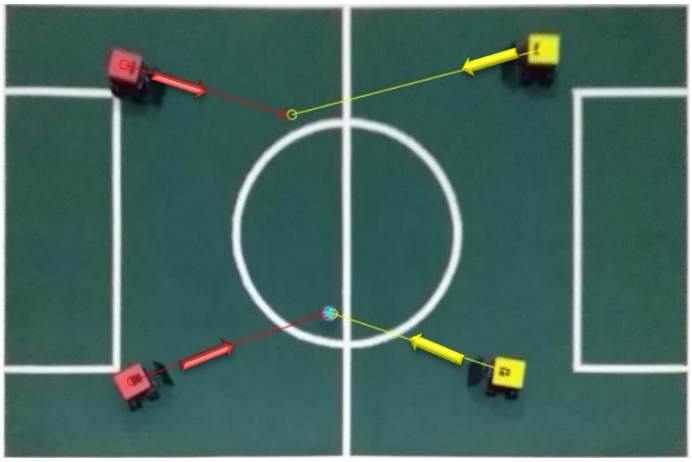


Figura 67. Detección correcta

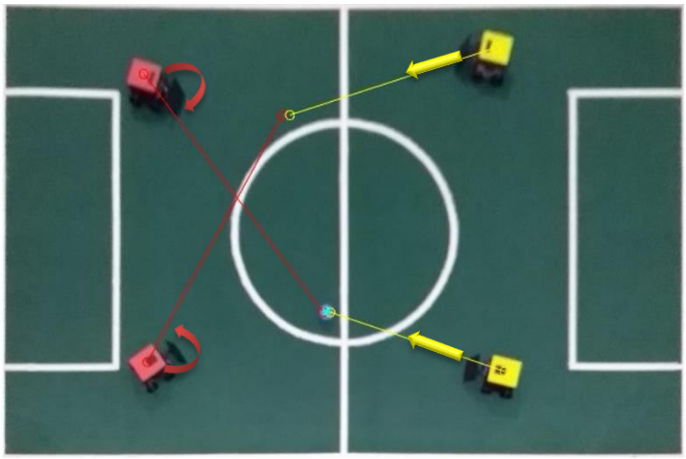


Figura 68. Detección incorrecta del equipo rojo durante el juego.

Luego de otros movimientos el sistema vuelve a hacer una detección errónea en el equipo rojo, y hace que los jugadores tiendan a corregir el ángulo hacia la posición nueva demandada por el algoritmo.
Se inicia de nuevo el juego esta vez ajustando algunos parámetros en la identificación del color rojo para que el algoritmo no tienda a confundir a los robots, ver figura 62.

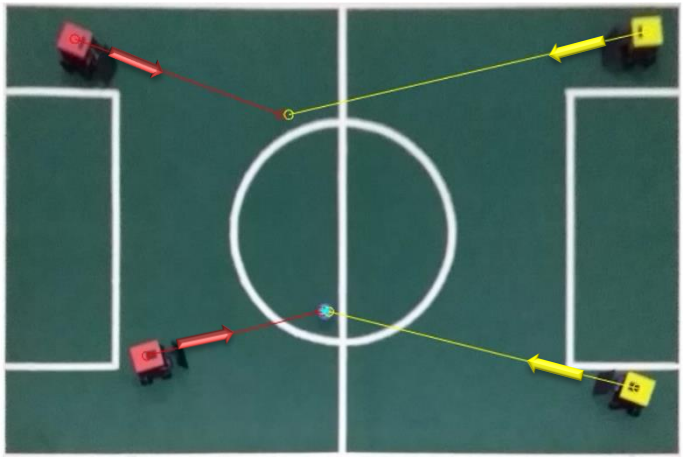


Figura 69. Inicio del segundo juego.

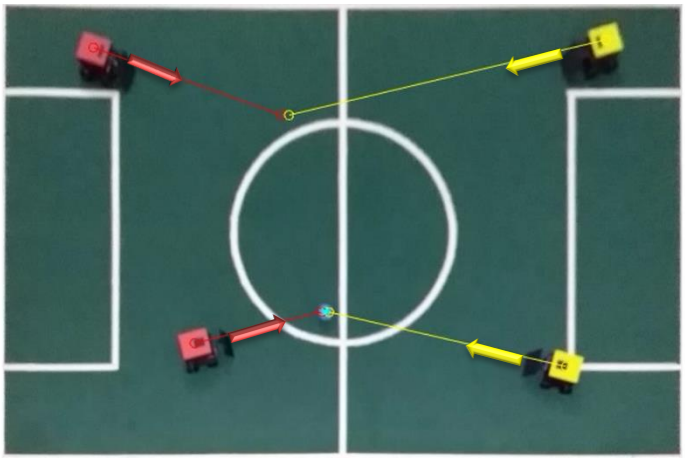


Figura 70. Acercamiento a la pelota y ajustes de ángulos

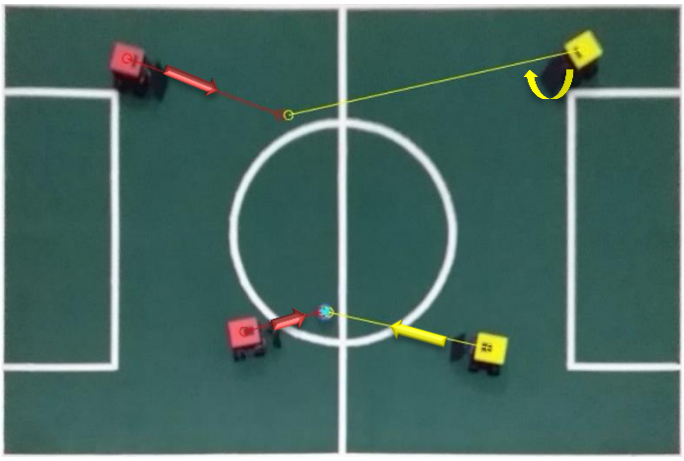


Figura 71. Acercamiento a la pelota y ajustes de cada robot.

Las figuras 62, 63 y 64 detallan el desarrollo del segundo juego en los equipos, el acercamiento a la pelota, las correcciones que hacen los robots para tener el ángulo correcto de ataque o movimiento.

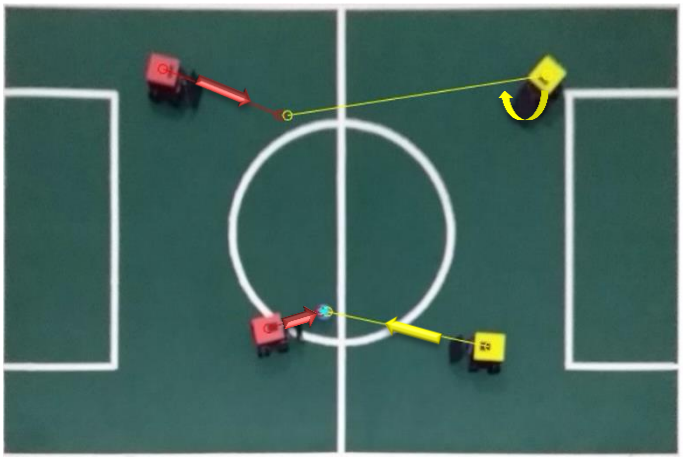


Figura 72. Avance de cada equipo.

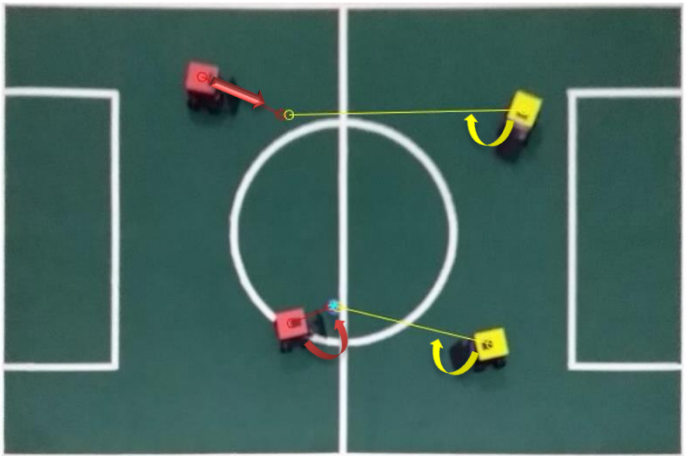


Figura 73. Aproximación a la pelota por parte del equipo rojo.

Figuras 65 y 66 se evidencia el acercamiento a la pelota por parte del equipo rojo, mientras los demás robots llegan a sus respectivas áreas dentro del juego. También los ajustes angulares realizados por los robots para acomodarse de una mejor manera frente a la pelota.

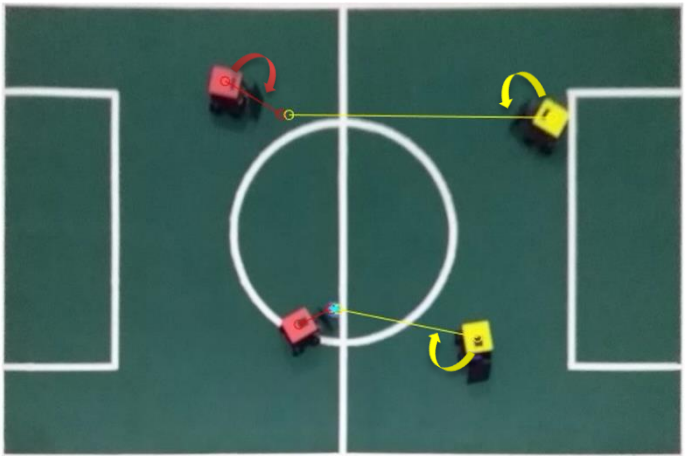


Figura 74. Robot 2 del equipo rojo llega a la pelota.

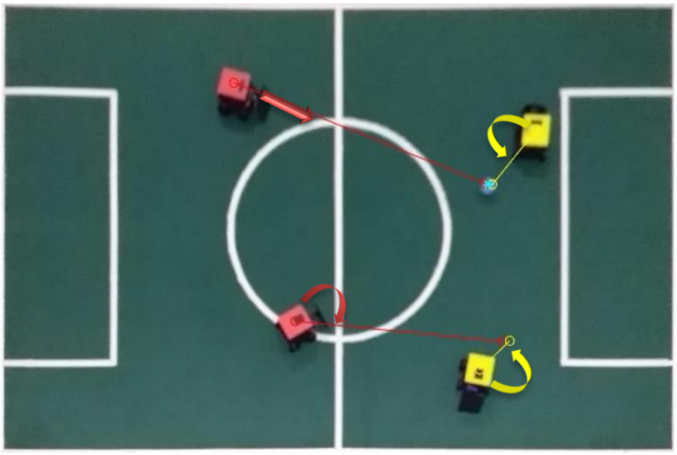


Figura 75. Seguimiento del juego posterior al pateo.

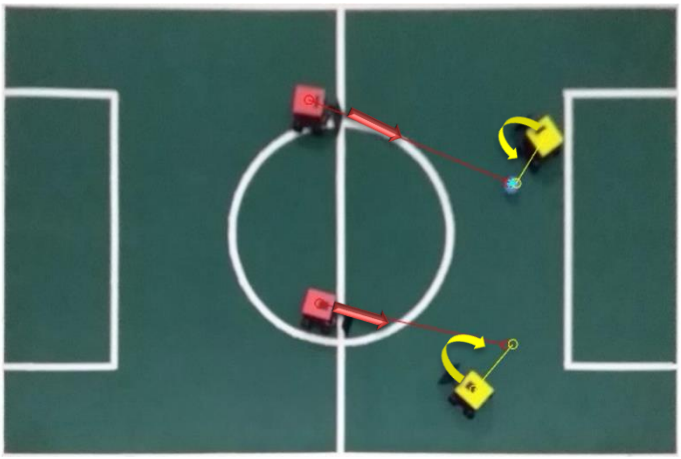


Figura 76. Avance de los robots en el juego.

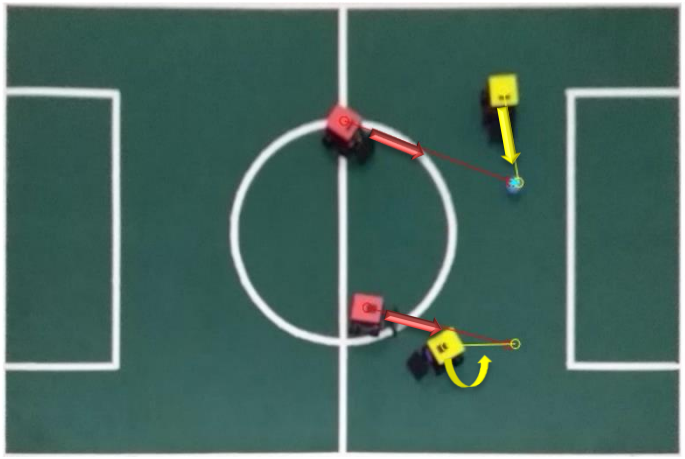


Figura 77. Avance de los robots hacia las áreas marcadas.

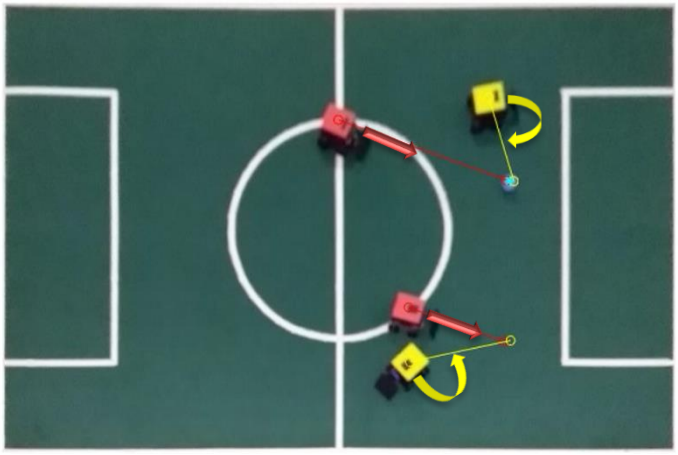


Figura 78. Ajuste de ángulos y avance.

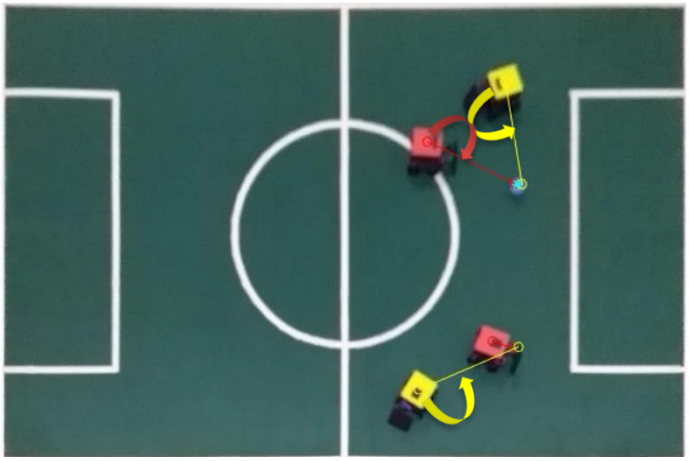


Figura 79. Aproximación del equipo rojo a la pelota.

La figura 67 describe el primer acercamiento del equipo rojo a la pelota donde ocurre el primer pateo por parte de este equipo, la pelota se desplaza por una parte de la cancha quedando posicionada para que sean los números 1 de cada equipo los que deban atacar la pelota, se muestra en la figura 68. De la figura 69 a la 72 se detalla el avance que tienen los robots hacia la pelota y hacia las zonas de espera, haciendo las correcciones pertinentes en los ángulos con respecto al objetivo del juego.

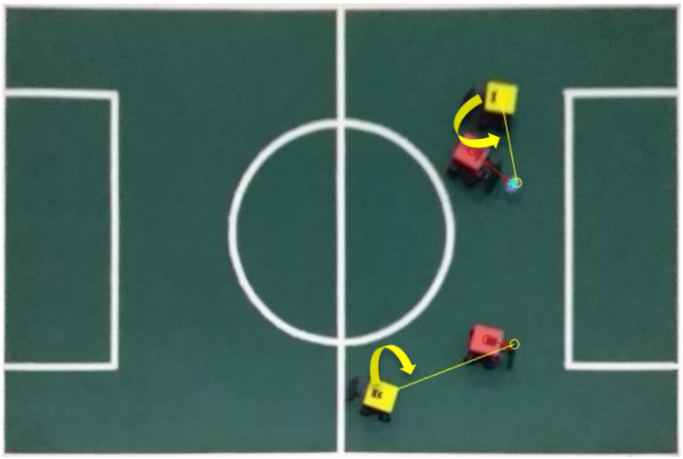


Figura 80. Segundo pateo a la pelota por parte del equipo rojo.

La figura 73 muestra el posicionamiento de cada robot dentro del ambiente de juego, ubicando al equipo rojo con la posesión de la pelota y el equipo amarillo con un bajo rendimiento. Se debe resaltar que el algoritmo de alto nivel se encargó en este caso de dirigir ambos equipos, lo que aumenta el tiempo de ejecución del programa.
Como se sabe que el programa está dividido en varias etapas: La primera etapa consta de la entrada de una imagen inicial, aquí se calcula el tiempo de lectura de la imagen, con la cual se realizan los ajustes del sistema como lo son la detección de la cancha, búsqueda automática de las cuatro esquinas de la cancha, cálculo y almacenamiento de los parámetros necesarios para la homografía, y una primera homografía a esa imagen entrante, posteriormente se calcula el tiempo que tarda el sistema en hallar estos datos y homografía. La segunda etapa es la detección de ambos equipos, se halla el tiempo que demora el ordenador en detectar ambos equipos. La tercera etapa busca el error de posición y de orientación de los robots con respecto a cada objetivo, aquí se cuenta el tiempo en ejecutar esta parte del código. La cuarta etapa consta de mostrar en pantalla los datos enviados a cada robot con la respectiva línea de error entre la posición actual y la deseada. La quinta y última etapa es leer una nueva imagen y aplicar nuevamente la homografía con los parámetros previamente calculados. Cabe aclarar que la primera etapa se ejecuta una sola vez para los ajustes y no se vuelve a ejecutar.

Tabla 6. Tiempos de ejecución del algoritmo de alto nivel.

TIEMPOS EN EJECUTAR CADA ETAPA						
ETAPA	Tiempo (s)					
Primera	2.0688					
Segunda	1.1511	0.7525	0.7326	0.6658	0.6751	0.6765
Tercera	0.0151	0.0031	0.0025	0.0031	0.0153	2.374e-4
Cuarta	0.6262	0.0725	0.0749	0.0742	0.1187	0.0694
Quinta	0.8143	0.7369	0.7113	0.8282	0.6973	0.6889
Tiempo total	2.6067	1.565	1.4545	1.5713	1.5064	1.4350

El algoritmo inicialmente demora alrededor 4.6 segundos en ejecutar todas las líneas la primera vez, luego baja a un tiempo de 1.5 segundos en ejecutar las etapas segunda a la quinta sucesivamente.

CONCLUSIONES

El control servo visual presentado en este trabajo permite la identificación correcta de los elementos dentro de la imagen como lo son la cancha, los robots y la pelota de manera eficiente. El algoritmo de control está diseñado para dirigir un grupo de robots a lo largo de un partido siempre que la pelota se encuentre dentro de la cancha, asignando los puntos en los que los robots deben tomar posición durante todo el partido por medio de una estrategia de juego que es ejecutada a medida que se desarrolla el juego, siendo capaz de responder a la entrada del sistema, extrayendo los parámetros de control y enviándolos a los robots mediante el servidor web. El flujo de datos del ordenador central hacia los robots logró implementarse de manera inalámbrica tardando 0.1 segundos enviarlos a todos los robots.

Se realizan las pruebas de movilidad de un equipo de tres robots y se observó una respuesta adecuada de estos ante el cambio de la posición de la pelota, ajustando los movimientos de rotación y traslación que deben generar para el seguimiento de la trayectoria deseada. En la prueba, el robot que tuvo como objetivo golpear la pelota, fue capaz de hacer un acercamiento y una corrección en la orientación con respecto a la pelota, para ubicarse de frente a esta y golpearla. En la validación de la estrategia se comprobó el resultado de esta ubicando a los robots en las áreas demandadas sin colisiones con miembros del mismo equipo, sin embargo, en ocasiones se producen colisiones cuando se ejecuta un juego entre dos equipos pues el algoritmo no está dotado de un sistema que evite a los miembros del primero chocar contra miembros del segundo. El sistema bajo las mismas condiciones de luz es eficiente, pero cambios de luz en el entorno, pueden generar variaciones en los parámetros de colores de cada equipo, en ocasiones, haciendo que el sistema de detección confunda a dos robots del mismo equipo.

BIBLIOGRAFÍA

[1] M. Allen, E. Westcoat, y L. Mears, “Optimal path planning for image based visual servoing”, *Procedia Manuf.*, vol. 39, núm. 2019, pp. 325–333, 2019, doi: 10.1016/j.promfg.2020.01.364.

[2] M. Laranjeira, C. Dune, y V. Hugel, “Catenary-based visual servoing for tether shape control between underwater vehicles”, *Ocean Eng.*, vol. 200, núm. January, p. 107018, 2020, doi: 10.1016/j.oceaneng.2020.107018.

[3] D. I. Kosmopoulos, “Robust Jacobian matrix estimation for image-based visual servoing”, *Robot. Comput. Integr. Manuf.*, vol. 27, núm. 1, pp. 82–87, 2011, doi: 10.1016/j.rcim.2010.06.013.

[4] A. H. Abdul Hafez, P. Mithun, V. V. Anurag, S. V. Shah, y K. M. Krishna, “Reactionless visual servoing of a multi-arm space robot combined with other manipulation tasks”, *Rob. Auton. Syst.*, vol. 91, pp. 1–10, 2017, doi: 10.1016/j.robot.2016.12.010.

[5] A. Taherian, A. H. Mazinan, y M. Aliyari-Shoorehdeli, “Image-based visual servoing improvement through utilization of adaptive control gain and pseudo-inverse of the weighted mean of the Jacobians”, *Comput. Electr. Eng.*, vol. 83, 2020, doi: 10.1016/j.compeleceng.2020.106580.

[6] X. Song y F. Miaomiao, “CLFs-based optimization control for a class of constrained visual servoing systems”, *ISA Trans.*, vol. 67, pp. 507–514, 2017, doi: 10.1016/j.isatra.2016.11.018.

[7] C. Je y H. M. Park, “Homographic p-norms: Metrics of homographic image transformation”, *Signal Process. Image Commun.*, vol. 39, pp. 185–201, 2015, doi: 10.1016/j.image.2015.08.009.

[8] A. Amirkhani, M. Shirzadeh, M. H. Shojaeefard, y A. Abraham, “Controlling wheeled mobile robot considering the effects of uncertainty with neuro-fuzzy cognitive map”, *ISA Trans.*, núm. xxxx, 2020, doi: 10.1016/j.isatra.2019.12.011.

[9] P. Sudhakara, V. Ganapathy, B. Priyadharshini, y K. Sundaran, “Obstacle Avoidance and Navigation Planning of a Wheeled Mobile Robot using Amended Artificial Potential Field Method”, *Procedia Comput. Sci.*, vol. 133, pp. 998–1004, 2018, doi: 10.1016/j.procs.2018.07.076.

[10] M. Velasco-Villa, E. Aranda-Bricaire, H. Rodríguez-Cortés, y J. González-Sierra, “Trajectory tracking for awheeled mobile robot using a vision based positioning system and an attitude observer”, *Eur. J. Control*, vol. 18, núm. 4, pp. 348–355, 2012, doi: 10.3166/EJC.18.348-355.

[11] M. Haddad, T. Chettibi, S. Hanchi, y H. E. Lehtihet, “A random-profile approach for trajectory planning of wheeled mobile robots”, *Eur. J. Mech. A/Solids*, vol. 26, núm. 3, pp. 519–540, 2007, doi: 10.1016/j.euromechsol.2006.10.001.

[12] Z. F. Li, J. T. Li, X. F. Li, Y. J. Yang, J. Xiao, y B. W. Xu, “Intelligent Tracking Obstacle Avoidance Wheel Robot Based on Arduino”, *Procedia Comput. Sci.*, vol. 166, pp. 274–278, 2020, doi: 10.1016/j.procs.2020.02.100.

[13] J. G. Guarnizo y M. Mellado, “Arquitectura Basada en Roles Aplicada en Equipos de Fútbol de Robots con Control Centralizado”, *RIAI - Rev. Iberoam. Autom. e Inform. Ind.*, vol. 13, núm. 3, pp. 370–380, 2016, doi: 10.1016/j.riai.2016.05.005.

[14] H. Shi, Z. Lin, S. Zhang, X. Li, y K. S. Hwang, “An adaptive decision-making method with fuzzy Bayesian reinforcement learning for robot soccer”, *Inf. Sci. (Ny).*, vol. 436–437, pp. 268–281, 2018, doi: 10.1016/j.ins.2018.01.032.

[15] V. Svatoň, J. Martinovič, K. Slaninová, y T. Bureš, “Improving strategy in robot soccer game by sequence extraction”, *Procedia Comput. Sci.*, vol. 35, núm. C, pp. 1445–1454, 2014, doi: 10.1016/j.procs.2014.08.204.

[16] K. G. Jolly, K. P. Ravindran, R. Vijayakumar, y R. Sreerama Kumar, “Intelligent decision making in multi-agent robot soccer system through compounded artificial neural networks”, *Rob. Auton. Syst.*, vol. 55, núm. 7, pp. 589–596, 2007, doi: 10.1016/j.robot.2006.12.011.

[17] E. R. M. Aleluya, A. D. Zamayla, y S. L. M. Tamula, “Decision-making system of soccer-playing robots using finite state machine based on skill hierarchy and path planning through Bezier polynomials”, *Procedia Comput. Sci.*, vol. 135, pp. 230–237, 2018, doi: 10.1016/j.procs.2018.08.170.

[18] G. Yang, “Research of strategy for RoboCup soccer robots competition”, *Procedia Eng.*, vol. 15, pp.

649–654, 2011, doi: 10.1016/j.proeng.2011.08.121.
[19] C. Hua, Y. Wang, y X. Guan, “Visual tracking control for an uncalibrated robot system with unknown camera parameters”, *Robot. Comput. Integr. Manuf.*, vol. 30, núm. 1, pp. 19–24, 2014, doi: 10.1016/j.rcim.2013.06.002.
[20] Z. Ma y J. Su, “Robust uncalibrated visual servoing control based on disturbance observer”, *ISA Trans.*, vol. 59, pp. 193–204, 2015, doi: 10.1016/j.isatra.2015.07.003.
[21] K. Ahlin, B. Joffe, A. P. Hu, G. McMurray, y N. Sadegh, “Autonomous Leaf Picking Using Deep Learning and Visual-Servoing”, *IFAC-PapersOnLine*, vol. 49, núm. 16, pp. 177–183, 2016, doi: 10.1016/j.ifacol.2016.10.033.
[22] P. Muñoz-Benavent, L. Gracia, J. E. Solanes, A. Esparza, y J. Tornero, “Robust fulfillment of constraints in robot visual servoing”, *Control Eng. Pract.*, vol. 71, núm. November 2017, pp. 79–95, 2018, doi: 10.1016/j.conengprac.2017.10.017.
[23] G. Allibert, M. D. Hua, S. Krupínski, y T. Hamel, “Pipeline following by visual servoing for Autonomous Underwater Vehicles”, *Control Eng. Pract.*, vol. 82, núm. October 2018, pp. 151–160, 2019, doi: 10.1016/j.conengprac.2018.10.004.
[24] Y. Zhang, C. Hua, Y. Li, y X. Guan, “Adaptive neural networks-based visual servoing control for manipulator with visibility constraint and dead-zone input”, *Neurocomputing*, vol. 332, pp. 44–55, 2019, doi: 10.1016/j.neucom.2018.11.058.

ANEXOS

ANEXO A. Identificación de equipos y pelota.

```
% Leer imagen original
RGB = imread('n1.jpg'); % ingresa la imagen y se guarda en la variable
RGB
% Division por bandas
RGB_M=RGB;  % Guarda la imagen original en una matriz nueva
modificable
% Se descompone la imagen en cada una de sus bandas
R_i = RGB_M(:,:,1);
G_i = RGB_M(:,:,2);
B_i = RGB_M(:,:,3);
% Asignación de los respectivos umbrales para cada código de colores
% Umbral para la identificación del color verde de la cancha
vur_minP = 100;, vur_maxP = 150;     % Pelota
vug_minP = 130;, vug_maxP = 170;
vub_minP = 160;, vub_maxP = 220;
% Umbral para la identificación del color en la marca roja
vur_minR = 200;, vur_maxR = 250;     % Rojo
vug_minR = 65;, vug_maxR = 110;
vub_minR = 57;, vub_maxR = 105;
% Umbral para la identificación del color en la marca Amarillo
vur_minA = 225;, vur_maxA = 255;     % Amarillo
vug_minA = 203;, vug_maxA = 237;
vub_minA = 1;,   vub_maxA = 65;
% Deteccion de la cancha
R2 = R_i;
G2 = G_i;
B2 = B_i;
R2(R2<vur_minP)=0;, R2(R2>vur_maxP)=0;
G2(G2<vug_minP)=0;, G2(G2>vug_maxP)=0;
B2(B2<vub_minP)=0;, B2(B2>vub_maxP)=0;
CBIN_P = R2 & G2 & B2; % Función lógica 'and' para cumplir condiciones
de marca
CBIN_P = bwareaopen(CBIN_P,300); % Aplica filtro de la imagen
CBIN_P = imfill(CBIN_P, 'holes');
% Detección de marcadores rojos
R2 =R_i;
G2 =G_i;
B2 =B_i;
R2(R2<vur_minR)=0;, R2(R2>vur_maxR)=0;
G2(G2<vug_minR)=0;, G2(G2>vug_maxR)=0;
B2(B2<vub_minR)=0;, B2(B2>vub_maxR)=0;
CBIN_R = R2 & G2 & B2; % Función lógica 'and' para cumplir condiciones
de marca
```

```
CBIN_R = bwareaopen(CBIN_R,3000); % Aplica filtro de la imagen
% Detectar marca azul
R2 =R_i;
G2 =G_i;
B2 =B_i;
R2(R2<vur_minA)=0;, R2(R2>vur_maxA)=0;
G2(G2<vug_minA)=0;, G2(G2>vug_maxA)=0;
B2(B2<vub_minA)=0;, B2(B2>vub_maxA)=0;
CBIN_A = R2 & G2 & B2; % Función lógica 'and' para cumplir condiciones
de marca
CBIN_A = bwareaopen(CBIN_A,2000); % Aplica filtro de la imagen
% Estadísticas de regiones - Centroides y conversión de estructura a
matriz
ESTADISTICAS_P = regionprops(CBIN_P,'Centroid')
Conver_p = cell2mat(struct2cell(ESTADISTICAS_P)');
ESTADISTICAS_r = regionprops(CBIN_R,'Centroid')
Conver_r = cell2mat(struct2cell(ESTADISTICAS_r)');
ESTADISTICAS_a = regionprops(CBIN_A,'Centroid')
Conver_a = cell2mat(struct2cell(ESTADISTICAS_a)');
% Asignación de las componentes en X y Y
% componentes para la identificación de la cancha
Xp = Conver_p(:,2);
Yp = Conver_p(:,1);
stc=('pelota');
% componentes para la identificación de la marca roja
Xr = Conver_r(:,2);
Yr = Conver_r(:,1);
str=('Rojo');
% componentes para la identificación de la marca azul
Xa = Conver_a(:,2);
Ya = Conver_a(:,1);
sta=('Amarillo');
% Mostrar imagen con su respectiva identificación
f1=figure(1);, set(f1,'Color',[1,1,1]);
imshow(RGB);
if (isempty(ESTADISTICAS_a) == 0)
    text(Ya,Xa,sta,'Color','Yellow','FontSize',16)
end
if (isempty(ESTADISTICAS_r) == 0)
    text(Yr,Xr,str,'Color','red','FontSize',16)
end
if (isempty(ESTADISTICAS_P) == 0)
    text(Yp,Xp,stc,'Color','black','FontSize',16)
end
```

ANEXO B. Estrategia de juego con identificación de cada robot

```
RGB = imread('n11.jpg');
% Detección de cancha como color 1
Resultado_G = Detecta_cancha(RGB);
Resultado_RGBin = im2bw(Resultado_G);
%% DETECTAR LAS CUATRO ESQUINAS DE LA CANCHA
Esquinas = Detecta_4Esquinas(Resultado_G);
X = [Esquinas(:,1)];
Y = [Esquinas(:,2)];
indc1 = 12;
indc2 = 18;
X(1) = X(1) - indc1;
Y(1) = Y(1) - indc1;
X(2) = X(2) + indc1;
Y(2) = Y(2) - indc1;
X(3) = X(3) + indc2;
Y(3) = Y(3) + indc2;
X(4) = X(4) - indc2;
Y(4) = Y(4) + indc2;
%% Transformar perspectiva - HOMOGRAFÍA
x=[1;1800;1800;1];
y=[1;1;1200;1200];
T = Homografia2D(X,Y,x,y);
xWorldLimits = [1 1800];
yWorldLimits = [1 1200];
RGB_M = imwarp(RGB,T,'FillValues',255,'OutputView',imref2d([1200 1800
3],xWorldLimits,yWorldLimits));
%% Detección de equipos y marcas
vur_minA = 225;, vur_maxA = 255;
vug_minA = 203;, vug_maxA = 237;
vub_minA = 1;, vub_maxA = 65;
% Umbral para la identificación del color en la marca Roja
vur_minR = 200;, vur_maxR = 250;
vug_minR = 65;, vug_maxR = 110;
vub_minR = 57;, vub_maxR = 105;
% Umbral para la identificación del color de la Pelota
vur_minP = 100;, vur_maxP = 150;
vug_minP = 130;, vug_maxP = 170;
vub_minP = 160;, vub_maxP = 220;
%% Equipo rojo
Team_Red =
Detecta_Equipo_Rojo(RGB_M,vur_minR,vur_maxR,vug_minR,vug_maxR,vub_minR
,vub_maxR);
%% Equipo amarillo
Team_Yell =
Detecta_Equipo_Amarillo(RGB_M,vur_minA,vur_maxA,vug_minA,vug_maxA,vub_
minA,vub_maxA);
%% Detección de pelota
XY_ball =
```

```
Detecta_Pelota(RGB_M,vur_minP,vur_maxP,vug_minP,vug_maxP,vub_minP,vub_
maxP);
%% Cálculo del error de los robots con respecto a la pelota y zonas de
defensa o ataque
PuntoI = [1 600];
PuntoD = [1800 600];
Z = ZonaBall (XY_ball);

% Amarillo
[Mag_A,Ang_Ca,Am] = Error_Robot_Ball_Amarillo_2(PuntoI, XY_ball,
Team_Yell, Z(:,1:2));
% rojo
[Mag_R,Ang_Cr,Ar] = Error_Robot_Ball_Rojo_2(PuntoD, XY_ball, Team_Red,
Z(:,1:2));
```

Funciones usadas en el código anterior:
Detecta_cancha(RGB)

```
function Resultado_G = Detecta_cancha(RGB)
    RGB1 = double(RGB);
    color1 = [87, 99, 99];
    Error = ( (RGB1(:,:,1)- color1(1)).^2 + (RGB1(:,:,2)-
color1(2)).^2 + (RGB1(:,:,3)- color1(3)).^2 ).^0.5;
    Error_tolerable1 = 82;
    [f,c]= size(Error);
    Resultado = zeros(f,c);
    Resultado_G= Resultado;
    Resultado_G(Error < Error_tolerable1)=120;
    Resultado_G = bwareaopen(Resultado_G,50000);
end
```

Homografía2D

```
function T = Homografia2D(X,Y,x,y)
    A=zeros(8,8);
    A(1,:)=[X(1),Y(1),1, 0,   0,   0, -1*X(1)*x(1), -1*Y(1)*x(1)];
    A(2,:)=[0,   0,   0, X(1),Y(1),1, -1*X(1)*y(1), -1*Y(1)*y(1)];
    A(3,:)=[X(2),Y(2),1, 0,   0,   0, -1*X(2)*x(2), -1*Y(2)*x(2)];
    A(4,:)=[0,   0,   0, X(2),Y(2),1, -1*X(2)*y(2), -1*Y(2)*y(2)];
    A(5,:)=[X(3),Y(3),1, 0,   0,   0, -1*X(3)*x(3), -1*Y(3)*x(3)];
    A(6,:)=[0,   0,   0, X(3),Y(3),1, -1*X(3)*y(3), -1*Y(3)*y(3)];
    A(7,:)=[X(4),Y(4),1, 0,   0,   0, -1*X(4)*x(4), -1*Y(4)*x(4)];
    A(8,:)=[0,   0,   0, X(4),Y(4),1, -1*X(4)*y(4), -1*Y(4)*y(4)];
    v=[x(1);y(1);x(2);y(2);x(3);y(3);x(4);y(4)];
    u=A\v;
    U=reshape([u;1],3,3)';
    T=projective2d(U');
end
```

Detecta_4Esquinas(Resultado_G)

```
function Esquinas = Detecta_4Esquinas(Resultado_G)
    [nf,nc]=size(Resultado_G);
    [X,Y] = meshgrid(1:nc,1:nf);
    Xm = X.*Resultado_G;
    Ym = Y.*Resultado_G;
    D = Xm.^2+Ym.^2;
    eID=[X(find(D==max(max(D)))), Y(find(D==max(max(D))))];
    D(D==0)=1000000000000;
    eSI=[X(find(D==min(min(D)))), Y(find(D==min(min(D))))];
    D(D==1000000000000)=0;
    %%%%
    Xm2 = (X-nc).*Resultado_G;
    D2 = Xm2.^2+Ym.^2;
    eII=[X(find(D2==max(max(D2)))), Y(find(D2==max(max(D2))))];
    D2(D2==0)=1000000000000;
    eSD=[X(find(D2==min(min(D2)))), Y(find(D2==min(min(D2))))];
    D2(D2==1000000000000)=0;
    indc1 = 12;
    indc2 = 18;
    Esquinas = [eSI;eSD;eID;eII];
end
```

Detecta_Pelota

```
function Esquinas = Detecta_4Esquinas(Resultado_G)
    [nf,nc]=size(Resultado_G);
    [X,Y] = meshgrid(1:nc,1:nf);
    Xm = X.*Resultado_G;
    Ym = Y.*Resultado_G;
    D = Xm.^2+Ym.^2;
    eID=[X(find(D==max(max(D)))), Y(find(D==max(max(D))))];
    D(D==0)=1000000000000;
    eSI=[X(find(D==min(min(D)))), Y(find(D==min(min(D))))];
    D(D==1000000000000)=0;
    %%%%
    Xm2 = (X-nc).*Resultado_G;
    D2 = Xm2.^2+Ym.^2;
    eII=[X(find(D2==max(max(D2)))), Y(find(D2==max(max(D2))))];
    D2(D2==0)=1000000000000;
    eSD=[X(find(D2==min(min(D2)))), Y(find(D2==min(min(D2))))];
    D2(D2==1000000000000)=0;
    indc1 = 12;
    indc2 = 18;
    Esquinas = [eSI;eSD;eID;eII];
end
```

Detecta_Equipo_Rojo()

```
function Team_Red =
Detecta_Equipo_Rojo(RGB_M,vur_minR,vur_maxR,vug_minR,vug_maxR,vub_minR
,vub_maxR)
    R_i = RGB_M(:,:,1);
    G_i = RGB_M(:,:,2);
    B_i = RGB_M(:,:,3);
    TeamR =R_i;
    TeamG =G_i;
    TeamB =B_i;
    TeamR(TeamR < vur_minR) = 0;, TeamR(TeamR > vur_maxR) = 0;
    TeamG(TeamG < vug_minR) = 0;, TeamG(TeamG > vug_maxR) = 0;
    TeamB(TeamB < vub_minR) = 0;, TeamB(TeamB > vub_maxR) = 0;
    Team = TeamR & TeamG & TeamB;
    Team = bwareaopen(Team,3000);
    % Centroide
    TeamFill = imfill(Team,'holes');
    % Marcas
    Team_not = not(Team);
    Team_marc = bwareafilt(Team_not,[0,1200]);
    Team_marc = bwareaopen(Team_marc,80);
    % Areas y centoides
    Pos_Fill = regionprops(TeamFill,'Centroid');
    PosArR_marc = regionprops(Team_marc,'Centroid','Area');
    AreaMarca = [PosArR_marc.Area];
    PosMarca = [PosArR_marc.Centroid];
    PosMarca = [PosMarca(1),PosMarca(2);...
                PosMarca(3),PosMarca(4);...
                PosMarca(5),PosMarca(6)];
    PosFill = [Pos_Fill.Centroid];
    PosFill = [PosFill(1),PosFill(2);...
               PosFill(3),PosFill(4);...
               PosFill(5),PosFill(6)];
    Ar_Max = find(AreaMarca == max(AreaMarca));
    Ar_min = find(AreaMarca == min(AreaMarca));
    Ar_med = find(AreaMarca~=min(AreaMarca) & AreaMarca ~=
max(AreaMarca));
    % Cálculo de ángulos
    Resta_Pos = [PosFill - PosMarca];
    An_T = [round(atan2d(Resta_Pos(1,1),Resta_Pos(1,2)));...
            round(atan2d(Resta_Pos(2,1),Resta_Pos(2,2)));...
            round(atan2d(Resta_Pos(3,1),Resta_Pos(3,2)))]+90;
    % Información de cada Robot [Area,pos_X,pos_Y,Ángulo]
    Team_Red =
[AreaMarca(Ar_min),PosFill(Ar_min,1),PosFill(Ar_min,2),An_T(Ar_min);..
.

AreaMarca(Ar_med),PosFill(Ar_med,1),PosFill(Ar_med,2),An_T(Ar_med);...

AreaMarca(Ar_Max),PosFill(Ar_Max,1),PosFill(Ar_Max,2),An_T(Ar_Max)];
end
```

Detecta_Equipo_Amarillo()

```
function Team_Yell =
Detecta_Equipo_Amarillo(RGB_M,vur_minA,vur_maxA,vug_minA,vug_maxA,vub_m
inA,vub_maxA)
    R_i = RGB_M(:,:,1);
    G_i = RGB_M(:,:,2);
    B_i = RGB_M(:,:,3);
    TeamR =R_i;
    TeamG =G_i;
    TeamB =B_i;
    TeamR(TeamR < vur_minA) = 0;, TeamR(TeamR > vur_maxA) = 0;
    TeamG(TeamG < vug_minA) = 0;, TeamG(TeamG > vug_maxA) = 0;
    TeamB(TeamB < vub_minA) = 0;, TeamB(TeamB > vub_maxA) = 0;
    Team = TeamR & TeamG & TeamB;
    Team = bwareaopen(Team,2000);
    % Centroide
    TeamFill = imfill(Team,'holes');
    % Marcas
    Team_not = not(Team);
    Team_marc = bwareafilt(Team_not,[0,3000]);
    Team_marc = bwareaopen(Team_marc,80);
    % Areas y centoides
    Pos_Fill = regionprops(TeamFill,'Centroid');
    PosArR_marc = regionprops(Team_marc,'Centroid','Area');
    AreaMarca = [PosArR_marc.Area];
    PosMarca = [PosArR_marc.Centroid];
    PosMarca = [PosMarca(1),PosMarca(2);...
                PosMarca(3),PosMarca(4);...
                PosMarca(5),PosMarca(6)];
    PosFill = [Pos_Fill.Centroid];
    PosFill = [PosFill(1),PosFill(2);...
               PosFill(3),PosFill(4);...
               PosFill(5),PosFill(6)];
    Ar_Max = find(AreaMarca == max(AreaMarca));
    Ar_min = find(AreaMarca == min(AreaMarca));
    Ar_med = find(AreaMarca~=min(AreaMarca) & AreaMarca ~=
max(AreaMarca));
    % Cálculo de ángulos
    Resta_Pos = [PosFill - PosMarca];
    An_T = [round(atan2d(Resta_Pos(1,1),Resta_Pos(1,2)));...
            round(atan2d(Resta_Pos(2,1),Resta_Pos(2,2)));...
            round(atan2d(Resta_Pos(3,1),Resta_Pos(3,2)))]+90;
    % Información de cada Robot [Area,pos_X,pos_Y,Ángulo]
    Team_Yell =
[AreaMarca(Ar_min),PosFill(Ar_min,1),PosFill(Ar_min,2),An_T(Ar_min);...

AreaMarca(Ar_med),PosFill(Ar_med,1),PosFill(Ar_med,2),An_T(Ar_med);...

AreaMarca(Ar_Max),PosFill(Ar_Max,1),PosFill(Ar_Max,2),An_T(Ar_Max)];
end
```

Error_Robot_Ball_Amarillo2

```
function [Mag_A,Ang_Ca,Am] = Error_Robot_Ball_Amarillo_2(PuntoI,
XY_ball, Team_Yell, Z)
    % Angulo de salida
    Ang_I = round([atan2d(-1*(PuntoI(2) - Z(1,2)),PuntoI(1) -
Z(1,1));...
                   atan2d(-1*(PuntoI(2) - Z(2,2)),PuntoI(1) -
Z(2,1));...
                   atan2d(-1*(PuntoI(2) - Z(3,2)),PuntoI(1) -
Z(3,1))]);
    % calculo punto D (posicion de pateo) del robot hacia meta
    D = 50; %10 %120
    Am = [-1*(cosd(Ang_I(1))*D) + Z(1,1), (sind(Ang_I(1))*D) +
Z(1,2);...
          -1*(cosd(Ang_I(2))*D) + Z(2,1), (sind(Ang_I(2))*D) +
Z(2,2);...
          -1*(cosd(Ang_I(3))*D) + Z(3,1), (sind(Ang_I(3))*D) + Z(3,2)];
    % Angulo de trayectoria robot --> D
    Error_A = [Am(1,1) - Team_Yell(1,2), Am(1,2) - Team_Yell(1,3);...
               Am(2,1) - Team_Yell(2,2), Am(2,2) - Team_Yell(2,3);...
               Am(3,1) - Team_Yell(3,2), Am(3,2) - Team_Yell(3,3)];
    % Magnitud de la distancia entre robots y pelota
    Mag_A =  round([norm(Error_A(1,:));...
                    norm(Error_A(2,:));...
                    norm(Error_A(3,:))]);
    % Corrección del angulo de los robots
    Ang_pend_A = [round(atan2d(-1*Error_A(1,2),Error_A(1,1)));...
                  round(atan2d(-1*Error_A(2,2),Error_A(2,1)));...
                  round(atan2d(-1*Error_A(3,2),Error_A(3,1)))];
    Ang_Ca = [Ang_pend_A(1) - Team_Yell(1,4);...
              Ang_pend_A(2) - Team_Yell(2,4);...
              Ang_pend_A(3) - Team_Yell(3,4)];
end
```

Error_Robot_Ball_Rojo2()

```
function [Mag_R,Ang_Cr,Ar] = Error_Robot_Ball_Rojo_2(PuntoD, XY_ball,
Team_Red, Z)
    % Ángulo de salida
    Ang_D = round([atan2d(-1*(PuntoD(2) - Z(1,2)),PuntoD(1) -
Z(1,1));...
                   atan2d(-1*(PuntoD(2) - Z(2,2)),PuntoD(1) -
Z(2,1));...
                   atan2d(-1*(PuntoD(2) - Z(3,2)),PuntoD(1) -
Z(3,1))]);
    % cálculo punto D (posicion de pateo) del robot hacia meta
    D = 50;  % 10 %120
    Ar = [-1*(cosd(Ang_D(1))*D) + Z(1,1), (sind(Ang_D(1))*D) +
Z(1,2);...
          -1*(cosd(Ang_D(2))*D) + Z(2,1), (sind(Ang_D(2))*D) +
Z(2,2);...
          -1*(cosd(Ang_D(3))*D) + Z(3,1), (sind(Ang_D(3))*D) +
Z(3,2)];
    % Ángulo de trayectoria robot --> D
    Error_R = [Ar(1,1) - Team_Red(1,2), Ar(1,2) - Team_Red(1,3);...
               Ar(2,1) - Team_Red(2,2), Ar(2,2) - Team_Red(2,3);...
               Ar(3,1) - Team_Red(3,2), Ar(3,2) - Team_Red(3,3)];
    % Magnitud de la distancia entre robots y pelota
    Mag_R = round([norm(Error_R(1,:));...
                   norm(Error_R(2,:));...
                   norm(Error_R(3,:))]);
    % Corrección del angulo de los robots
    Ang_pend_R = [round(atan2d(-1*Error_R(1,2),Error_R(1,1)));...
                  round(atan2d(-1*Error_R(2,2),Error_R(2,1)));...
                  round(atan2d(-1*Error_R(3,2),Error_R(3,1)))];
    Ang_Cr = [Ang_pend_R(1) - Team_Red(1,4);...
              Ang_pend_R(2) - Team_Red(2,4);...
              Ang_pend_R(3) - Team_Red(3,4)];
end
```

ZonaBall()

```
function Z = ZonaBall (XY_ball)
    X = XY_ball(1); Y = XY_ball(2);
    if(X >= 1 && X <= 300 && Y >= 1 && Y <= 400)
        Z = [  X,    Y, 1;...
             450, 1000, 2;...
             750,  200, 2];
    elseif(X >= 1 && X <= 300 && Y >= 401 && Y <= 800)
        Z = [  X,    Y, 1;...
             450, 1000, 2;...
             750,  200, 2];
    elseif(X >= 1 && X <= 300 && Y >= 801 && Y <= 1200)
        Z = [450,  200, 2;...
               X,    Y, 1;...
             750, 1000, 2];
```

```
    elseif(X >= 301 && X <= 600 && Y >= 1 && Y <= 400)
        Z = [  X,    Y, 1;...
             150, 1000, 2;...
             750, 1000, 2];
    elseif(X >= 301 && X <= 600 && Y >= 401 && Y <= 800)
        Z = [ 750,  200, 2;...
                X,    Y, 1;...
             1050, 1000, 2];
    elseif(X >= 301 && X <= 600 && Y >= 801 && Y <= 1200)
        Z = [ 750,  200, 2;...
                X,    Y, 1;...
             1050, 1000, 2];
    elseif(X >= 601 && X <= 900 && Y >= 1 && Y <= 400)
        Z = [   X,    Y, 1;...
              450,  600, 2;...
             1050, 1000, 2];
    elseif(X >= 601 && X <= 900 && Y >= 401 && Y <= 800)
        Z = [   X,    Y, 1;...
              450,  600, 2;...
             1050, 1000, 2];
    elseif(X >= 601 && X <= 900 && Y >= 801 && Y <= 1200)
        Z = [ 450, 600, 2;...
                X,   Y, 1;...
             1050, 200, 2];
    elseif(X >= 901 && X <= 1200 && Y >= 1 && Y <= 400)
        Z = [   X,    Y, 1;...
              750,  600, 2;...
             1350, 1000, 2];
    elseif(X >= 901 && X <= 1200 && Y >= 401 && Y <= 800)
        Z = [ 750, 1000, 2;...
                X,    Y, 1;...
             1350,  200, 2];
    elseif(X >= 901 && X <= 1200 && Y >= 801 && Y <= 1200)
        Z = [ 750, 1000, 2;...
                X,    Y, 1;...
             1350,  200, 2];
    elseif(X >= 1201 && X <= 1500 && Y >= 1 && Y <= 400)
        Z = [ 750, 1000, 2;...
             1350,  200, 2;...
                X,    Y, 1];
    elseif(X >= 1201 && X <= 1500 && Y >= 401 && Y <= 800)
        Z = [ 750, 1000, 2;...
             1350,  200, 2;...
                X,    Y, 1];
    elseif(X >= 1201 && X <= 1500 && Y >= 801 && Y <= 1200)
       Z = [ 750, 1000, 2;...
            1350,  200, 2;...
               X,    Y, 1];
    elseif(X >= 1501 && X <= 1800 && Y >= 1 && Y <= 400)
        Z = [ 750, 600, 2;1350, 600, 2;X,   Y, 1];
    elseif(X >= 1501 && X <= 1800 && Y >= 401 && Y <= 800)
       Z = [ 1050, 600, 2;1350, 600, 2;X,   Y, 1];
    elseif(X >= 1501 && X <= 1800 && Y >= 801 && Y <= 1200)
        Z = [1350,  200, 2;1050, 1000, 2;X,    Y, 1];
```

ANEXO C. Control grupal

```
clc,clear all,close all;
url = 'http://192.168.43.1:8080/shot.jpg';
RGB  = imread(url);
%% Detección de Cancha
Resultado_G = Detecta_cancha(RGB);
%% DETECTAR LAS CUATRO ESQUINAS DE LA CANCHA
Esquinas = Detecta_4Esquinas(Resultado_G);
X = [Esquinas(:,1)];
Y = [Esquinas(:,2)];
X(1) = X(1) - indc1;
Y(1) = Y(1) - indc1;
X(2) = X(2) + indc1;
Y(2) = Y(2) - indc1;
X(3) = X(3) + indc2;
Y(3) = Y(3) + indc2;
X(4) = X(4) - indc2;
Y(4) = Y(4) + indc2;

%% Transformar perspectiva - HOMOGRAFÍA

x=[1;1800;1800;1];
y=[1;1;1200;1200];

Esquinas = Detecta_4Esquinas(Resultado_G) % Detecta las cuatro
esquinas de la cancha

xWorldLimits = [1 1800];
yWorldLimits = [1 1200];
RGB=imwarp(RGB,T,'FillValues',255,'OutputView',imref2d([1200 1800
3],xWorldLimits,yWorldLimits));
f9=figure(9);, set(f9,'Color',[1,1,1]);
imshow(RGB)
%% Detección de equipos y marcas
% Umbral para la identificación del color en la marca Roja
vur_minR = 100;, vur_maxR = 255;
vug_minR = 63;, vug_maxR = 145;
vub_minR = 95;, vub_maxR = 161;
% Umbral para la identificación del color de la Pelota
vur_minP = 112;, vur_maxP = 179;
vug_minP = 142;, vug_maxP = 190;
```

Ciclo repetitivo:

```
while(1)
    tic
    % Detección de equipo rojo
    Team_Red =
Detecta_Equipo_Rojo2(RGB_M,vur_minR,vur_maxR,vug_minR,vug_maxR,vub_min
R,vub_maxR)

    % Detección de pelota
    XY_ball =
Detecta_Pelota2(RGB_M,vur_minP,vur_maxP,vug_minP,vug_maxP,vub_minP,vub
_maxP)

    % Calculo del error de los robots con respecto a la pelota
    Z = ZonaBall (XY_ball);

    % Angulo de salida
    PuntoD = [1800 600];
    % Cálculo de error de posición de losrobots con respecto al
objetivo
    [Mag_R,Ang_Cr,Ar] = Error_Robot_Ball_Rojo_3(PuntoD, XY_ball,
Team_Red, Z)
    t = toc
    % Volemos a leer la imagen y se aplica homografia
    RGB = imread(url);
    RGB = imwarp(RGB,T,'FillValues',255,'OutputView',imref2d([1200
1800 3],xWorldLimits,yWorldLimits));
    % Ejecución de los movimientos de los robots
    tic
    Movimiento1(Ang_Cr(1), Mag_R(1), Z(1,3));
    Movimiento2(Ang_Cr(2), Mag_R(2), Z(2,3));
    Movimiento3(Ang_Cr(3), Mag_R(3), Z(3,3));
    % Muestra en pantalla el análisis del juego
    f9=figure(9), set(f9,'color',[1,1,1]);
    imshow(RGB), hold on;
    plot([Ar(1,1);Team_Red(1,2)],[Ar(1,2);Team_Red(1,3)],'r-o');
    plot([Ar(2,1);Team_Red(2,2)],[Ar(2,2);Team_Red(2,3)],'r-o');
    plot([Ar(3,1);Team_Red(3,2)],[Ar(3,2);Team_Red(3,3)],'r-o');
    plot(XY_ball(1),XY_ball(2),'c*');
    t2 = toc

end
```

ANEXO D. Programación del servidor web

```
#include <ESP8266WiFi.h>
#include <Servo.h>

const char* ssid = "Nombre de red";
const char* password = "contraseña";
// establece la red en el puerto 80
WiFiServer server(80);

void setup() {
  Serial.begin(115200);

  delay(10);
/Configuración  del GPIO2
  pinMode(2, OUTPUT);
  digitalWrite(2,LOW);

  // Configuración de red
  Serial.println();
  Serial.println();
  Serial.print("Conectandose a red : ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);   //Conexion a la red

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi conectado");

  server.begin(); //Iniciamos el servidor
  Serial.println("Servidor Iniciado");

  Serial.println("Ingrese desde un navegador web usando la siguiente IP:");
  Serial.println(WiFi.localIP()); //Obtenemos la IP
}
```

```
void loop() {

  WiFiClient client = server.available();
  if (client) //Si hay un cliente presente
  {
    Serial.println("Nuevo Cliente");

    //esperamos hasta que hayan datos disponibles
    while(!client.available()&&client.connected())
    {
    delay(1);
    }
    // Leemos la primera línea de la petición del cliente.
    String linea1 = client.readStringUntil('r');

    if (linea1.indexOf("ATRAS")>0)
    {
    Serial.println("va atras");
    }
    if (linea1.indexOf("ADELANTE")>0)
    {
    Serial.println("va atras");
    }
    if (linea1.indexOf("IZQUIERDA")>0)
    {
    Serial.println("va atras");
    }
    if (linea1.indexOf("DERECHA")>0)
    {
    Serial.println("va atras");
    }
    client.flush();
```

```
Serial.println("Enviando respuesta...");
    //Encabesado http
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/html");
    client.println("Connection: close");// La conexión se cierra después de finalizar de la
respuesta
    client.println();
    //Pagina html  para en el navegador
    client.println("<!DOCTYPE HTML>");
    client.println("<html>");
    client.println("<head><title>Mecatronica - robot</title>");
    client.println("<body>");
    client.println("<h1 align='center'>CONTROL DE MOVIMIENTOS DEL
ROBOT</h1>");
    client.println("<div style='text-align:center;'>");
    client.println("<br />");
    client.println("<br />");
    client.println("<button onClick=location.href='./ADELANTE'>ADELANTE</button>");
    client.println("<br />");
    client.println("<br />");
    client.println("<button onClick=location.href='./IZQUIERDA'>IZQUIERDA</button>");
    client.println("<button onClick=location.href='./PARADA'>PARADA</button>");
    client.println("<button onClick=location.href='./DERECHA'>DERECHA</button>");
     client.println("<br />");
    client.println("<br />");
    client.println("<br />");
    client.println("<br />");
    client.println("<button onClick=location.href='./PATADA'>PATADA</button>");
    client.println("<br />");
    client.println("<br />");
    client.println("</div>");
    client.println("</body>");
    client.println("</html>");

     delay(1);
    Serial.println("respuesta enviada");
    Serial.println();
  }
}
```