**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

**FACULTAD DE INGENIERÍA**

# Sistema de visión para estimar posición y velocidad de objetos para un robot bípedo

**TESIS**

Que para obtener el título de

**Ingeniero Mecatrónico**

**P R E S E N T A**

Luis Eduardo González Nava

**DIRECTOR DE TESIS**

Dr. Marco Antonio Negrete Villanueva

**Ciudad Universitaria, Cd. Mx., 2021**

# Índice general

**1. Introducción** **5**
1.1. Motivación . . . . . 6
1.2. Planteamiento del problema . . . . . 6
1.3. Hipótesis . . . . . 7
1.4. Objetivos . . . . . 7
1.5. Descripción del documento . . . . . 7

**2. Antecedentes** **9**
2.1. Robots bípedos . . . . . 9
2.2. Aplicaciones en el juego de Futbol Soccer . . . . . 12
2.3. Usos de la visión computacional . . . . . 13
2.4. El Filtro de Kalman . . . . . 14
2.5. Estado del Árte . . . . . 18

**3. Segmentación de imágenes** **21**
3.1. Modelo de cámara estenopeica *(Pinhole)* . . . . . 21
3.2. Corrección de la distorsión . . . . . 25
3.3. Los espacios de color RGB y HSV . . . . . 27
3.4. Operadores morfológicos . . . . . 29

**4. Estimación de posición y velocidad** **35**
4.1. Cálculo de la posición del objeto de interés . . . . . 35
4.2. Estimación mediante EKF . . . . . 39
4.3. Obtención de los parámetros . . . . . 47

**5. Implementación** **51**
5.1. El robot bípedo Nimbro OP . . . . . 51
5.2. La bilbioteca OpenCV . . . . . 53

5.3. La plataforma ROS . . . . . 54
5.4. Integración de los diferentes programas . . . . . 56

**6. Resultados** **63**
6.1. Descripción del experimento . . . . . 63
6.2. Parámetros del experimento . . . . . 64
6.3. Pruebas de estimación . . . . . 65

**7. Discusión** **71**
7.1. Conclusiones . . . . . 71
7.2. Trabajo Futuro . . . . . 72

**Apéndices** **77**

**A. Código del estimador de Kalman** **79**

# Capítulo 1

# Introducción

La robótica estudia los mecanismos físicos controlados por un programa de cómputo que interactúan con los seres humanos o el entorno y realizan tareas que involucran procesamiento de información y una acción motora o perceptual de manera coordinada. Aunque la palabra "robot" proviene de la ciencia ficción y tiene como antecedentes los autómatas mecánicos que se han construido a lo largo de la historia, la robótica como disciplina se inicia con la invención de los brazos autónomos para asistir la manufactura en líneas de producción, especialmente en la industria automotriz.

A partir de estos robots iniciales, en conjunto con el desarrollo de las disciplinas relacionadas, como el Procesamiento de Señales, la Teoría de Control y la Inteligencia Artificial, durante las últimas décadas se han diseñado y construido una gran cantidad de robots o dispositivos robóticos que se pueden conceptualizar y clasificar en relación a dos dimensiones principales: i)autonomía y ii)complejidad del entorno. La primera depende de la riqueza de sus sensores y de la variedad y estructura de sus mecanismos motores e informacionales. La segunda corresponde a la variedad y variabilidad del entorno en que el robot es capaz de enfrentar e incluso transformar. Pineda Cortés (2017).

Como dice Sakagami y cols. (2002), los humanoides son especialmente deseables por la sociedad humana ya que pueden trabajar bien dentro de ambientes que fueron diseñados para humanos, además que con su particular forma se hace más fácil para las personas identificarlos cuando se comparan con otros tipos de robots. Los autómatas como ASIMO (Figura 2.1) pueden potencialmente asistir a la gente en las tareas diarias, incrementando su valor dentro de la sociedad.

## 1.1. Motivación

Los robots humanoides tienen un gran potencial para la robótica de servicio debido a que su bioinspiración mecánica les permite realizar tareas muy similares a las de un ser humano. Sin embargo, con cada mejora de capacidades, la complejidad de los algoritmos incrementa considerablemente. Es por eso que los concursos de la RoboCup abrieron la categoría *Humanoid* para impulsar el desarrollo e investigación de los humanoides (Kitano y Asada, 1998).

Una de las propuestas más convenientes para buscar dicho desarrollo, es formar equipos de humanoides con retos específicos del fútbol soccer, ya que engloba sistemas de visión computacional, inteligencia artificial y control automático análogos a los del sistema humano (Gerndt, Seifert, Baltes, Sadeghnejad, y Behnke, 2015).

A lo largo de los últimos años, se han creado extraordinarios algoritmos para realizar comportamientos complejos y cada vez más parecidos a los de los jugadores, pese a esto, aún resta mucho por perfeccionar. Por ejemplo al interactuar con objetos en movimiento, un humano puede fácilmente patear, cabecear o interceptar un balón que se mueve, con los humanoides estas son tareas complejas y requieren gran capacidad de cómputo. Por esta razón, la motivación de esta tesis es crear algoritmos para imitar dichas capacidades y hacer partidos más dinámicos.

## 1.2. Planteamiento del problema

Actualmente el desarrollo de los sistemas de visión computacional para humanoides ha postergado un poco el estudio de objetos en movimiento, debido en parte, a que los partidos se ejecutan de manera lenta y pausada, donde el balón permanece inmóvil la mayor parte del tiempo. No obstante, el avance en la complejidad de los concursos, ha hecho necesario obtener la velocidad y la trayectoria del balón, sobretodo en los tiros penales y en la prueba llamada *kick from a moving ball*.

En el Laboratorio de Bio-robótica de la UNAM se tiene a disposición un humanoide tipo *Nimbro-OP* el cual es un prototipo en desarrollo que representa una gran oportunidad para implementar las herramientas anteriormente mencionadas. Con estas herramientas, el humanoide será capaz de competir en concursos tanto nacionales como internacionales.

## 1.3. Hipótesis

Mediante un sistema de segmentación por color y suponiendo que el balón sólo estará sobre el suelo, se puede estimar su posición a partir de una imagen RGB y una serie de cálculos de cinemática directa.

Teniendo un muestreo constante de posición y velocidad del balón, se pueden estimar y filtrar los datos empleando el Filtro de Kalman Extendido. Posteriormente, este filtro se podrá utilizar para hacer una extrapolación de posición en un tiempo determinado, lo suficiente para que el robot pueda patear con precisión objetos en movimiento.

## 1.4. Objetivos

- Determinar la posición de un balón con base en un sistema de referencia egocéntrico para un robot humanoide utilizando técnicas de visión computacional.
- Aplicar el Filtro de Kalman Extendido para obtener la estimación de posición y velocidad de un balón en movimiento.
- Desarrollar un algoritmo de pateo basado en posiciones predefinidas.
- Integrar los distintos programas utilizando la plataforma ROS para que dicho humanoide patee un balón en movimiento.
- Realizar pruebas en simulación utilizando el modelo del robot Nimbro-OP y el simulador Gazebo.

## 1.5. Descripción del documento

Este trabajo explica el sistema de visión computacional implementado en un robot bípedo para la medición de posición de objetos en movimiento circundantes a él, y hacer una predicción de estado dentro de un intervalo de tiempo.

En el Capítulo 2 se aborda el estado del arte de los robots bípedos actuales y el potencial que tienen para poder hacer tareas parecidas a las de los seres humanos, generalmente más visibles en el juego de Fútbol Soccer. También se describen y se hacen comparativas entre los distintos sistemas de visión

que se han empleado dentro de los concursos de la RoboCup. Al final de este capítulo, se plantea lo que es el Filtro de Kalman y la ventaja que puede representar al filtrar datos cuando se obtienen mediciones de posición.

Dentro del Capítulo 3 se expone el marco teórico de los principios físicos y matemáticos para implementar la visión computacional, así como el hardware y la adecuación de los modelos matemáticos al mundo físico.

El Capítulo 4 tiene las deducciones del modelo matemático del sistema que se quiere analizar, la comparación entre el Filtro de Kalman y el Filtro de Kalman Extendido, la descripción de cada estado y los parámetros convenientes a usar a fin de hacer una óptima predicción.

Las herramientas de software, su integración y las características del hardware del Humanoide están ampliamente explicadas dentro del Capítulo 5, a fin de conocer los alcances y limitantes de este robot.

En la parte de Resultados están descritas las diversas pruebas que se hicieron con base en la teoría que se vio en los capítulos previos dejando en claro, el desempeño del robot con resultados cualitativos.

La discusión se plantea en el Capítulo 7, dentro de éste se encuentran las conclusiones y el trabajo a futuro que propone posibles mejoras que se le pueden implementar tanto al robot como a los sistemas de medición.

# Capítulo 2

# Antecedentes

En este capítulo se describen los criterios para definir qué es un robot con piernas y sus variantes. También se hace mención de las características principales para su diseño, tomando en cuenta su control y robustez a la hora de ejecución, haciendo hincapié en los robots bípedos. Por último, se describen sus diferentes aplicaciones, y las ventajas que tienen a comparación de los robots con ruedas.

Para empezar, se hace una introducción de qué es la visión computacional, sus usos más prácticos y el por qué resulta conveniente aplicar el Filtro de Kalman en algunos de estos sistemas.

Las aplicaciones y las implementaciones más comunes dentro del mundo de los robots bípedos (muy similares a los de esta tesis) se describen ampliamente en la última sección llamada Estado del Árte.

## 2.1. Robots bípedos

### Conceptos básicos

Un robot con piernas es un robot móvil que debe tener un cuerpo, al menos una pierna (extremidad inferior) y un número arbitrario de brazos (extremidades superiores). Generalmente sus piernas deben tener un actuador final con el cual apoyarse e impulsarse y sus brazos, un actuador para manipular objetos (Siciliano y Khatib, 2016). Por lo tanto, un *robot bípedo* debe tener dos piernas las cuales usa para moverse, de forma similar al caminado humano.

Entre los *robots bípedos* más conocidos se encuentran los *robots humanoides* (Figura 2.1) que poseen las siguientes características:

- Tienen un par de piernas para asemejar la apariencia y forma de un ser humano, adicionalmente su cuerpo puede tener dos brazos y una cabeza ajustada a un tronco.
- Deber ser capaces de permanecer de pie sobre sus pies y caminar con sus piernas.
- Pueden interactuar con humanos usando reconocimiento de voz y/o de imágenes.
- Los movimientos que son capaces de hacer deben ser cinemáticamente equivalentes a los de un ser humano por ejemplo, las articulaciones de la rodilla no deben doblarse para atrás, la cabeza no debe girar a más de 180 grados, (Robocup Humanoid League Rulebook, 2019).

Uno de los mayores retos a la hora de diseñar y modelar a los robot bípedos es su condición de equilibrio. Los robots con piernas tienen un alto número de grados de libertad en sus extremidades y estos deben estar muy bien coordinados para evitar que el robot caiga mientras avanza. No obstante, aunque la complejidad aumenta se tienen algunas ventajas, por ejemplo, los robots con ruedas no pueden subir escaleras y les resulta extremadamente difícil cambiar de entorno como en superficies llanas o con pendientes pronunciadas.

### Características para el diseño de un robot con piernas

- **Tipo de marcha:** Es el patrón de movimientos de piernas del robot (caminata).
- **Biomímesis:** Es el diseño de algunos robots para imitar la estructura mecánica de un ser vivo de tal manera que sea tan precisa como se pueda.
- **Bioinspiración mecánica:** Es el diseño que sirve para reproducir la robustez y versatilidad de la locomoción de animales, algunos diseñadores prestan más atención en la dinámica esencial de la locomoción que en la mecánica.

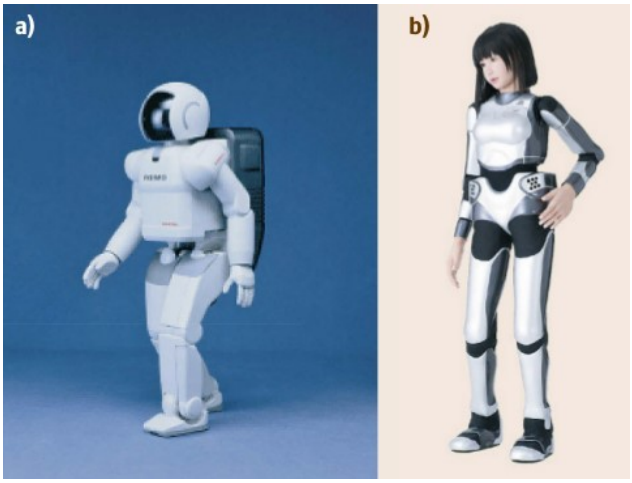


Figura 2.1: Ejemplos de robots humanoides (a) Asimov (2000); (b)HRP-4C (2009). Imagen tomada de: Siciliano y Khatib (2016).

- **Simplicidad mecánica:** Con la simplicidad se pretende usar el menor número de actuadores posibles para cumplir sólo con las tareas realizadas.
- **Espacio de trabajo de la extremidad:** Señala que una extremidad debería tener al menos 3GDL para moverse libremente en el espacio. Para que se pueda tener una arbitraria orientación en el efector final en un espacio 3-D se debe contar con al menos 6GDL.

### Análisis de estabilidad

Para el control del sistema dinámico no lineal hay un número de conceptos relativos para su seguridad y estabilidad:

- *Puntos fijos* : Representan las posturas estáticas en cuáles el robot puede estar de pie de manera segura.
- *Ciclos límites*: Proveen una natural extensión del análisis de los puntos fijos para movimientos de caminata periódica.
- *Viabilidad*: La viabilidad es un concepto de invarianza controlada, que analiza el conjunto de estados del cual el robot es capaz de mantenerse

de pie. Desafortunadamente esta propiedad puede ser intratable para el cómputo.

- *Controlabilidad*: La controlabilidad provee una ligera noción de restricción de viabilidad, analizando el conjunto de estados del cual el robot es capaz de retornar a un particular punto fijo.
- *Estabilidad robusta*: Examina las propiedades del sistema considerando el "peor de los casos" de las perturbaciones. Por instancias, un controlador robusto debería ser capaz de garantizar que un punto fijo es estable incluso si la estimación de masa del tronco tiene un error del $\pm 10\,\%$.
- *Estabilidad estocástica*: El análisis estocástico provee herramientas para investigar la probabilidad de caer. Para muchos modelos de perturbaciones en robots el sistema caerá eventualmente (con probabilidad uno), pero el análisis puede revelar la distribución del tiempo de vida metaestable.
- *Estabilidad de entrada-salida*: En este análisis se trata una particular perturbación como una entrada, un criterio de rendimiento como salida e intenta calcular una ganancia relativa o sensibilidad del rendimiento del robot debido a esta entrada.
- *Márgenes de estabilidad*: El análisis de robustez puede ser difícil. En la práctica, los diseñadores del control a menudo se conforman con que el sistema se mantenga cómodamente lejos de los límites de estabilidad determinista (Siciliano y Khatib, 2016).

## 2.2. Aplicaciones en el juego de Futbol Soccer

**Futbol Soccer en el concurso RoboCup**

La *RoboCup Federation* es una iniciativa internacional para promover la inteligencia artificial y la tecnología robótica a través de la organización de competencias y encuentros científicos. Una de las categorías más famosas dentro de este concurso es el de fútbol soccer con robots humanoides (Humanoid League) cuyo objetivo es lograr que para la mitad del siglo XXI un equipo de robots humanoides autónomos sean capaces de ganar una partida

de fútbol contra el recién campeón mundial de ese entonces, siguiendo las reglas oficiales de la FIFA.

Las competencias de soccer comenzaron en el año de 1997 con simples y simulados robots con ruedas, desde entonces el desarrollo fue creciendo hasta que en el 2002 se inauguró la liga de humanoides. Para ese entonces los principales retos eran comportamientos como caminar, patear y percibir objetos del ambiente, de esta manera, las mejoras en los mecanismos, la electrónica y control han hecho evolucionar el desarrollo para que la *biomímesis* sea la más parecida a la de un humano.(Gerndt y cols., 2015).

#### Liga de robots humanoides de categoría *TeenSize*

Recientemente, la introducción de robots estandarizados, accesibles y de bajo costo ha tenido una formidable aceptación para el desarrollo e investigación de robots para soccer. Un ejemplo, es el humanoide *DARwIn-OP* (Figura 2.2), este robot cuenta con una plataforma estable, y algunos comportamientos de soccer están ya implementados, de modo que los nuevos desarrolladores tienen la oportunidad de enfocarse en tareas de más alto nivel para hacer los partidos más dinámicos y complejos. (Schwarz y cols., 2013).

*DARwIn-OP* (categorizado dentro de *KidSize*) ha facilitado que los equipos lleguen a tener el número necesario de jugadores (robots) para los partidos, no obstante, para otras categorías en donde los robots tienden a tener mayor tamaño (*TeenSize* o *AdultSize*) no se llega a tener el número suficiente de jugadores para completar los equipos y algunos competidores se ven forzados a participar con robots hechos por ellos mismos. Consecuentemente esto altera el desempeño de los partidos y retrasa el desarrollo de nuevos comportamientos. Para solucionar este problema fue desarrollado un primer prototipo de plataforma abierta basado el *Darwin-OP* para la categoría *TeenSize* llamado *Nimbro-OP* (véase la Sección 5.1) el cual tiene un diseño de fácil manufactura, ensamblaje y mantenimiento.

## 2.3. Usos de la visión computacional

La *visión computacional* es la transformación de información lumínica desde una imagen o video a hacia una *nueva representación* de datos, Bradski y Kaehler (2008). Con esta nueva representación se pueden usar ciertas características de la luz capturada para transformarlas en variables numé-

Figura 2.2: Robot Humanoide Darwin-OP (Imagen tomada de: http://openrobot.cl/es/robot-darwin-op/).

ricas que la computadora pueda abstraer y procesar (tómese de ejemplo la figura 2.3).

Para los fines de esta tesis, no es será necesario hacer uso de software para reconocimiento de formas o patrones, solamente se utilizarán funciones para segmentar objetos con base en su color, lo cual es muy conveniente para determinar la posición del objeto en el espacio. Tal y como se puede ver en el análisis matemático de la Sección 4.1.

## 2.4. El Filtro de Kalman

### Filtro de Kalman Discreto

En 1960 Rudolf Emil Kalman publicó un artículo donde describe una forma sencilla para filtrar datos discretos de manera recursiva. Este método ha sido usado ampliamente en investigaciones y aplicaciones de ingeniería ya que resulta útil para la predicción de estados con base en modelos matemáticos y datos estadísticos. Por ejemplo, la posición y velocidad de un avión en vuelo, la temperatura de una caldera a la que se le está suministrando combustible


Figura 2.3: Representación de cómo una imagen es representada por la computadora. Imagen modificada de: Bradski y Kaehler (2008)

o las lecturas dispersas de posición en un GPS.

El Filtro de Kalman es un algoritmo computacionalmente eficiente cuyas ecuaciones matemáticas minimizan el error producido por el ruido inherente de las mediciones. Con la estimación es posible conocer con precisión el estado pasado, presente y futuro de un sistema, incluso si éste tiene un modelo de naturaleza desconocida (Welch, Bishop, y cols., 1995).

## Proceso de estimación

El Filtro de Kalman aborda el problema de tratar de estimar el estado $x \in \Re^n$ de un proceso de tiempo discreto que es modelado con la ecuación diferencial (2.1).

$$\hat{x}_{n+1,n} = F\hat{x}_{n,n} + G\hat{u}_{n,n} \tag{2.1}$$

En donde $F$ es una matriz de nxn que relaciona el estado presente $n$ con el estado futuro $n+1$. $G$ es una matriz de nx1 que relaciona la entrada de control $u \in R^1$ al estado $x$.

La medición $z \in \Re^m$, también conocida como vector de medición se puede modelar con la ecuación (2.2).

$$z_n = Hx_n + v_n \tag{2.2}$$

En donde la matriz de observación $H$ de mxn es la que relaciona el estado $x_n$ con la medición $z_n$, el vector $v_n$ representa el ruido aleatorio en la medición. El Filtro de Kalman supone que el error en las mediciones debe tener una distribución normal, de esta manera la predicción de los estados tendrá un mayor grado de confiabilidad, por lo tanto la expresión (2.3) representa cómo está distribuido el error del vector $v_n$.

$$p(v) \sim N(0, R) \tag{2.3}$$

Con el objetivo de encontrar una ecuación que calcule la estimación de un estado *a posteriori* $\hat{x}_{n,n}$ como una combinación lineal de un estado *a priori* $\hat{x}_{n,n-1}$ más un término proporcional a la diferencia entre la medición actual $z_n$ y la predicción $H\hat{x}_{n,n-1}$, se emplea la ecuación (2.4).

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n(z_n - H\hat{x}_{n,n-1}) \tag{2.4}$$

donde $K_n$ es una matriz de $n \times m$ llamada *Ganancia de Kalman*, esta ganancia está dada por la ecuación (2.5) y su valor determina si es más confiable el valor de la medición o la predicción previa del modelo matemático.

$$K_n = P_{n,n-1}H^T(HP_{n,n-1}H^T + R_n)^{-1} \tag{2.5}$$

Como puede observarse, cuando el error de covarianza de la medición $R_n$ se aproxima a cero, la ganancia $K_n$ tiene un mayor peso (cercano a uno). Por el contrario, cuando el error de covarianza *a posteriori* $P_{n,n-1}$ se aproxima a cero, la ganancia $K_n$ obtiene menor peso (cercano a cero). La matriz de $m \times m$ $Q$ o *matriz de error de proceso* describe el error causado por situaciones imprevistas, causados por ambientes irregulares o fallas mecánicas.

El algoritmo consta de dos fases, la fase *predictiva* y la fase *correctiva*. La fase predictiva consta de utilizar un modelo matemático para obtener las probables mediciones de un fenómeno. La fase correctiva obtiene los valores de la medición y las compara con las obtenidas con el modelo matemático, véase la Figura 2.4.

Como se vio, este filtro utiliza modelos matemáticos lineales, sin embargo en diversas situaciones, el modelo matemático no es lineal, y la predicción

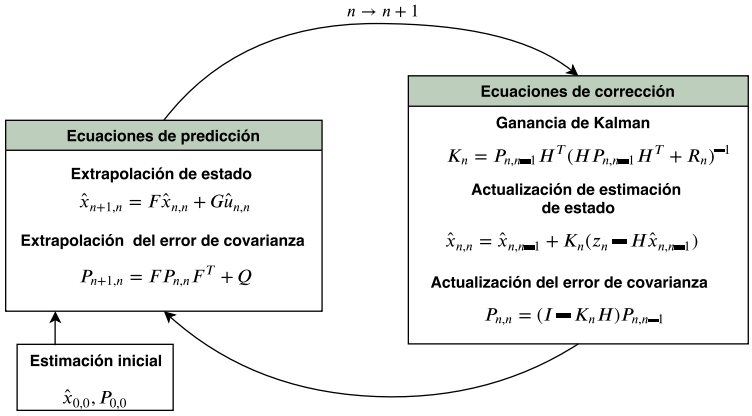


Figura 2.4: Esquema ilustrativo del Filtro de Kalman. Diagrama Modificado de: Welch y cols. (1995)

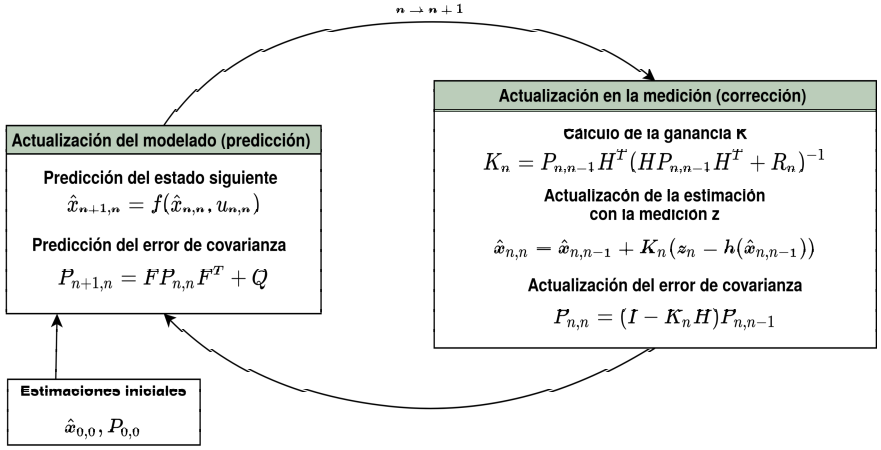


Figura 2.5: Esquema del Filtro de Kalman Extendido. Diagrama Modificado de: Welch y cols. (1995)

resulta ser errónea. Para solucionar esto, se puede implementar el Filtro de Kalman Extendido, que es básicamente el mismo tomando como lineal el modelo en ciertos intervalos de tiempo, y linealizando la predicción de la covarianza del proceso con base en la serie de Taylor. Véase la Figura 2.5.

En el Capítulo 4, se decide utilizar el Filtro de Kalman Extendido, debido a que hay una no linealidad dentro del modelo matemático para la predicción.

## 2.5. Estado del Árte

Como ya se ha mencionado, la detección del balón es el principal objetivo para la categoría de *Soccer Humanoids*. Esto representa un reto para los sistemas de visión computacional, el cual debe encargarse tanto de la detección como del cálculo de la posición con base en un sistema egocéntrico. Según los TDP *(Team Description Paper)* se puede saber cómo los equipos han abordado este desafío en los recientes años, por lo que ayuda en gran medida para saber qué técnicas se pueden reimplementar o mejorar según la necesidad.

En los siguientes párrafos, se resumen los métodos utilizados por los equipos de distintos países, así será fácil diferenciar con los métodos implementados de esta tesis.

### ITAndroids

El equipo *ITAndroids* del Instituto Tecnológico de Aereonáutica de Brasil implementó un sistema basado en los sistemas de visión de los humanoides de la plataforma estándar. Tomaron como punto de partida la segmentación por color con algoritmos hechos en MATLAB y funciones de OpenCV como *Hough Circle Transform* para la detección correcta del objetivo, tal y como se ve en la Figura 2.6.

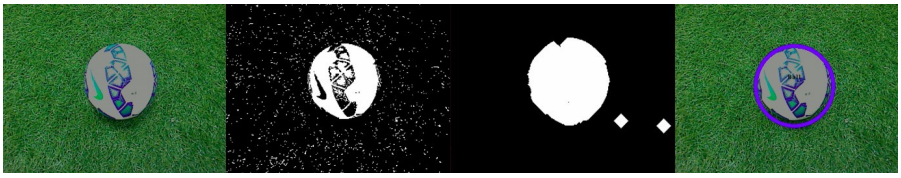

Figura 2.6: Proceso de visión implementado por ITAndroids. Imagen tomada de: Herculano y Samuel Pinto (2017)

Debido al ruido inherente en el procesamiento de imágenes, un Filtro de Kalman es empleado para estimar la posición más probable si es que un objeto del mismo color del balón es detectado.

Ya que se hace el reconocimiento del balón, se procede a calcular su posición relativa respecto al humanoide, transformando las coordenadas bidimensionales obtenidas por la imagen a tridimensionales haciendo uso de un método similar al de esta tesis (véase la Sección 4), suponiendo que la posición del balón en la coordenada $z$ es siempre 0 y calculando la cinemática directa desde la posición de la cámara a la posición de dicho objetivo, Herculano y Samuel Pinto (2017).

## NimbRo TeenSize

Debido a las nuevas reglas dadas por el comité organizador de la RoboCup en cuanto al color del balón (blanco como en las competencias de la FIFA), el equipo *NimbRo* de la Universidad de *Friedrich-Wilhelms* de Alemania optó por utilizar un sistema de reconocimiento que no se base sólo en color, pues las líneas divisorias de las canchas o la portería tienen la misma tonalidad. Es por eso que su sistema de reconocimiento está basado en histogramas HOG (histogram of oriented gradients).

Con el reconocimiento debidamente implementado, se usan las Transformaciones Homogéneas para obtener la posición del balón con cinemática directa, de este modo resulta muy útil la biblioteca *tf2* de ROS. No obstante, aunque las posiciones del robot están bien definidas, algunas variaciones del hardware pueden dar pauta al error en la medición. Para resolver estos fallos este equipo usó el método de Nelder-Mead que consiste en hacer una triangulación convergente para las probables posiciones del balón, Farazi, Allgeuer, Ficht, y Behnke (2016).

## T-Flow

Otra forma para realizar la detección de objetos es obteniendo una forma tridimensional específica con una cámara estéreo, tal y como lo hizo el equipo *T-Flow* del *Centro de Investigación de Robótica* del Politécnico de Indonesia.

Este equipo emplea una cámara estéreo para obtener un par de imágenes, remapear y formar una imagen tridimensional con profundidad aproximada, para posteriormente compararse con un patrón esférico como lo es el balón.

El sistema de estereo-visión resulta ser muy útil y sencillo para la detección y posicionamiento del objeto, no obstante, se requiere un elevado coste computacional, Suryanto y Dewi (s.f.).

## Ichiro

De una manera muy similar al robot del Laboratorio de Bio-robótica de la UNAM, los humanoides del equipo *Ichiro* del *Instituto Tecnológico Diez de Noviembre* de Indonesia, poseen una web-cam Logitech C922 con alta definición.

En la detección del balón este equipo emplea un método de Patrones de Binarios Locales el cual es un descriptor de texturas que toma como base la transformación de las imágenes a escala de grises, para posteriormente tomar como contorno los cambios de tonalidad, este método representa bajo costo computacional y hace la segmentación más independiente a los cambios de luz, Razi y cols. (s.f.).

Como se expresó en los trabajos citados TDPs, existen distintos procedimientos para la detección y localización tridimensional del objeto, las cuales resultan muy prometedoras. No obstante, poco se menciona sobre la detección de objetos en movimiento, empleando sólo objetos estáticos, por lo que mi trabajo de tesis busca ampliar las fronteras de los algoritmos comúnmente empleados.

# Capítulo 3

# Segmentación de imágenes con base en color

En este capítulo se desarrollan los conceptos necesarios para entender la visión computacional que se empleará en este trabajo. Comenzando con definiciones importantes y entendiendo cómo es que funcionan las cámaras, con sus capacidades y limitaciones.

Dentro de la Sección 3.2 se describen lo modelos matemáticos para corregir las diversas distorsiones que una cámara puede generar al momento de tomar una imagen (debido a sus irregularidades en la manufactura de las lentes).

Por último se explica el método de segmentación basado en color, tomando como base el espacio de color HSV (más adelante se explica por qué es más conveniente que el RGB) y los métodos matemáticos para la eliminación de ruido, más comúnmente conocidos como operadores morfológicos.

## 3.1. Modelo de cámara estenopeica *(Pinhole)*

De acuerdo con Bradski y Kaehler (2008) la visión es la detección de la luz del mundo. El proceso de visión empieza cuando un rayo de luz es emanado desde una fuente hacia un objeto. Cuando la luz choca con el objeto, mucha de esta luz es absorbida, la que no, se puede percibir como el color, de esta manera, la luz reflejada hace su camino hacia el sensor óptico.

El modelo más simple de cómo sucede la captura de luz, es el de la cámara

Figura 3.1: Ejemplo en la vida real de la proyección de una imagen con la cámara estenopeica. Imagen tomada de: https://www.pinterest.com.mx/pin/371828512962028619.

estenopeica o *pinhole*. Una cámara estenopeica se puede imaginar como una habitación sin ventanas en donde la luz únicamente entra por una pequeña apertura en el centro de la pared proyectando así una imagen dentro de la habitación, véase la Figura 3.1.

En la cámara estenopeica se considera que sólo un rayo de luz entra desde un punto en particular, este punto es *proyectado* sobre una superficie, generalmente plana. Como resultado, la imagen en este plano (también llamado plano proyectivo), está siempre en el foco y su tamaño se relaciona a la distancia del objeto por un sólo parámetro: *la distancia focal*. La principal diferencia entre la imagen real y la que aparece en una cámara Pinhole, es que la imagen aparece invertida.

El punto dentro del Pinhole es reinterpretado como el centro de proyección. Para este tipo de cámara, la distancia desde la abertura del Pinhole hacia la pantalla, es precisamente la distancia focal. Como puede verse en la Figura 3.2.

En la Figura 3.2, $f$ es representada como la distancia focal de la cámara,

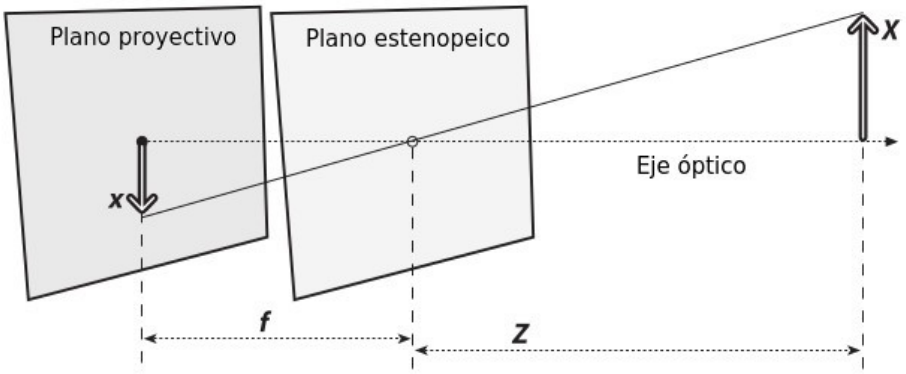


Figura 3.2: Esquema del modelo de cámara estenopeica. Imagen modificada de: Bradski y Kaehler (2008).

$Z$ es la distancia de la cámara al objeto, $X$ es la longitud del objeto, y $x$ es la longitud de la la imagen proyectada en el plano. Dentro de la figura, se pueden ver dos triángulos semejantes de los cuales se puede deducir la siguiente expresión:

$$x = -f\frac{X}{Z}$$

Para fines prácticos, el modelo *Pinhole* no es conveniente de usar en exposiciones rápidas, ya que toda su luz proviene de un solo punto. A fin de obtener otro modelo de cámara que recopile mayor cantidad de luz y tenga ecuaciones similares, pero sin signos negativos (correspondientes a la inversión de la imagen) Bradski y Kaehler (2008) propone hacer un rearreglo en el que se coloca al frente del centro de proyección el plano proyectivo. Tal y como se ve en la Figura 3.3.

Con este nuevo cambio, cada rayo de luz proveniente del objeto distante se dirige hacia el *centro de proyección* y deja un punto en el *plano proyectivo*. El punto de intersección del plano proyectivo y el eje óptico es conocido como *el punto principal*. La imagen proyectada en este nuevo plano de imagen, tiene exactamente el mismo tamaño que en el esquema mostrado en la Figura 3.2 pero en este caso, la imagen no queda invertida, por lo que su relación de triángulos quedaría de la siguiente manera:

$$\frac{x}{f} = \frac{X}{Z}$$

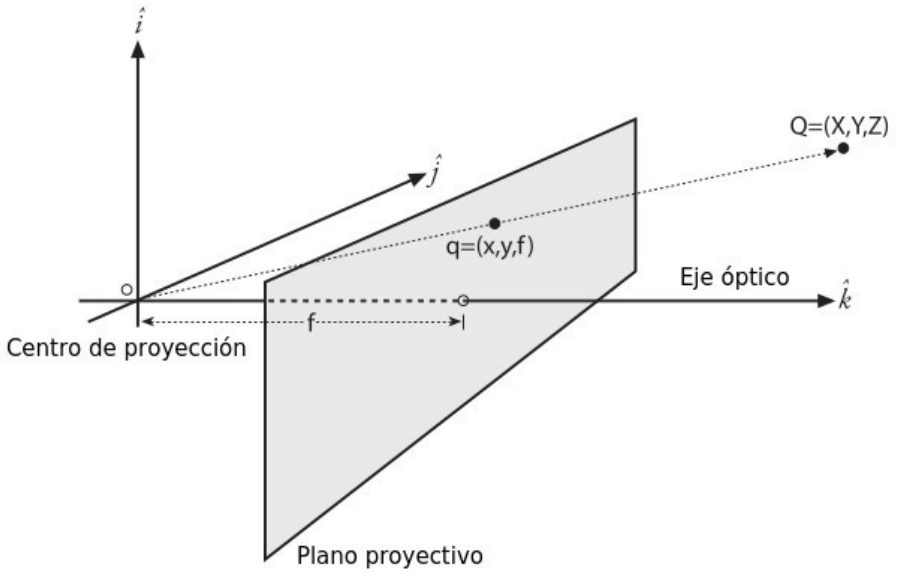


Figura 3.3: Rearreglo de una cámara estenopeica. Imagen modificada de: Bradski y Kaehler (2008).

Se podría pensar que que *el punto principal* es equivalente al centro de la imagen, sin embargo, este centro usualmente no está en el eje óptico, es por eso que se introducen dos nuevos parámetros, $c_x$ y $c_y$ para modelar un posible desplazamiento (perpendicular al eje óptico) del centro de coordenadas en el plano de proyección. El resultado es que un punto $Q$ en el mundo físico cuyas coordenadas son (X,Y,Z) es proyectado dentro de la pantalla en una localización de pixeles dada por $(x_{screen}, y_{screen})$ de acuerdo con las siguientes ecuaciones:

$$x_{screen} = f_x \left(\frac{X}{Z}\right) + c_x$$

$$y_{screen} = f_y \left(\frac{Y}{Z}\right) + c_y$$

Nótese que se han introducido dos diferentes distancias focales $f_x$ y $f_y$, la razón es porque generalmente los pixeles son rectangulares.

## 3.2. Corrección de la distorsión

### Geometría básica proyectiva

La relación que mapea los puntos $Q_i$ en el mundo físico con coordenadas $(X_i, Y_i, Z_i)$ a los puntos en el plano de proyección con coordenadas $(x_i, y_i)$ es llamada una *transformación proyectiva* (Forsyth y Ponce, 2011). Cuando se trabaja con estas transformaciones es conveniente usar las *coordenada homogéneas*. El plano de imagen es el espacio proyectado y tiene dos dimensiones, con lo que se pueden representar puntos en el plano como un vector tridimensional $q = (q_1, q_2, q_3)$ ó $q = (x, y, w)$, en donde w es la orientación. Una forma de hacer un arreglo con los parámetros que definen a la cámara $f_x, f_y, c_x$ y $c_y$ dentro de una matriz de 3x3 es la llamada *matriz de parámetros intrínsecos*. La proyección $q$ de los puntos del mundo físico en el plano de la imagen se puede calcular como:

$$q = MQ$$

donde

$$q = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \ , \ M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \ , \ Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

### Distorsión de las lentes

Debido a diversos factores en la manufactura, las lentes de una cámara no son perfectas, y al obtener una imagen, esta puede notarse distorsionada (véase la Figura 3.4). La distorsión es un fenómeno que todas las cámaras tienen, no obstante, es más evidente en las cámaras de baja calidad o en las que tienen lentes tipo *ojo de pescado*. Las distorsiones más características son: las *Distorsiones radiales* y las *Distorsiones tangenciales*. Las primeras surgen como resultado de la forma de las lentes y las segundas como resultado del proceso de ensamblado de la cámara.

Cuando la imagen se hace más pequeña conforme está más cerca de los bordes se está viendo una *distorsión radial* (véase la Figura 3.5). En la práctica, se puede caracterizar con la serie de Taylor, las ecuaciones son:

$$x_{corregida} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{corregida} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

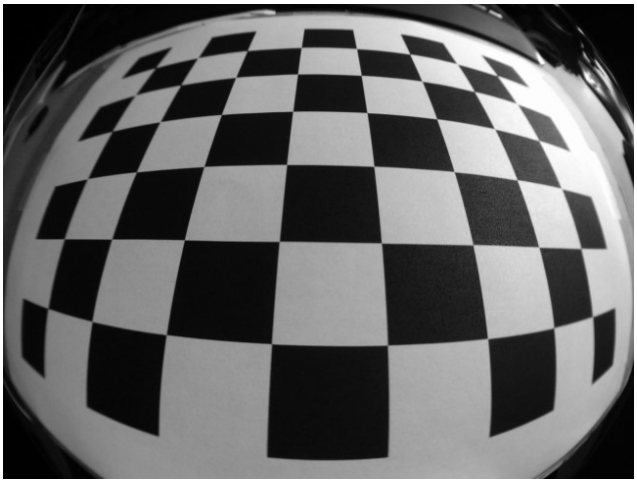

Figura 3.4: Ejemplo de distorsión de una cámara. Imagen tomada de: https://answers.opencv.org/question/114056/chessboard-detection-fails-on-fisheye-image/?sort=votes

En donde $(x, y)$ son las coordenadas de la imagen, $r$ es la distancia del punto de la imagen al centro y las constantes $k_1$, $k_2$ y $k_3$ son parámetros que están relacionados a cada cámara en específico.

Las *Distorsiones tangenciales* son ocasionadas por no tener paralelas las lentes con el plano de la imagen, vea la distribución de la distorsión tangencial en la Figura 3.6. Para obtener expresiones que ayuden a corregir este tipo de distorsión se introducen dos parámetros: $p_1$ y $p_2$ dentro de las siguientes ecuaciones:

$$x_{corregida} = x + [2p_1y + p_2(r^2 + 2x^2)]$$

$$y_{corregida} = y + [p_1(r^2 + 2y^2) + 2p_2x]$$

La estimación de los parámetros internos es importante en la calibración de las cámaras, de esto depende la exactitud de las mediciones de distancia dadas por la visión computacional. Bradski y Kaehler (2008).

Para poder hacer una corrección automática de todas las distorsiones, las herramientas de OpenCV (véase la Sección 5.2) usan los cinco parámetros antes mencionados dentro de una matriz de 1x5 llamada *matriz de coeficientes de distorsión*, en donde cada parámetro se acomoda de la siguiente forma:

$$[k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3]$$

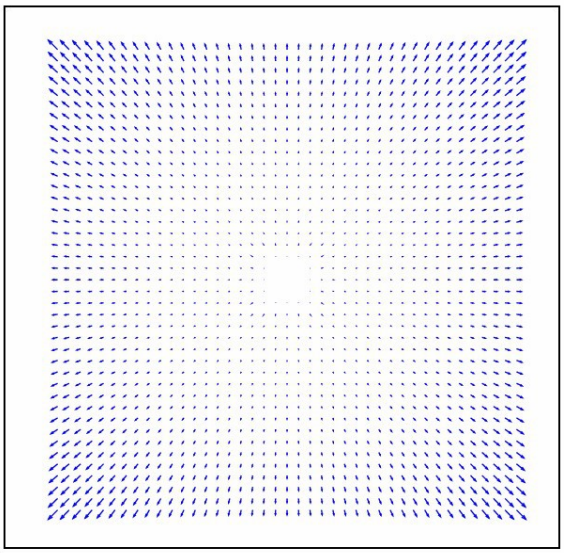

Figura 3.5: Distribución de la distorsión radial de una imagen. Imagen tomada de: https://www.i-ciencias.com/pregunta/96683/que-es-el-tangencial-distorsion-de-opencv-en-realidad-tangencial.

## 3.3. Los espacios de color RGB y HSV

Para el procesamiento de imágenes se utilizará un segmentador con base en el color. Aunque no es tan bueno como otros (por ejemplo con base en profundidad) es más fácil de utilizar y es mejor que el de escala de grises. Gonzalez, Woods, y Eddins (2004).

### Percepción del color

Según Agoston y Agoston (2005), en el caso más simple, la percepción del color tiene tres características principales llamadas *matiz, saturación y brillo.*

- *Matiz (Hue):* La matiz es un descriptor de qué tan combinados están los colores unitarios entre sí (si se entiende por color unitario a los colores rojo, amarillo, verde y azul).

- *Saturación (Saturation):* La saturación es la percepción de la relativa

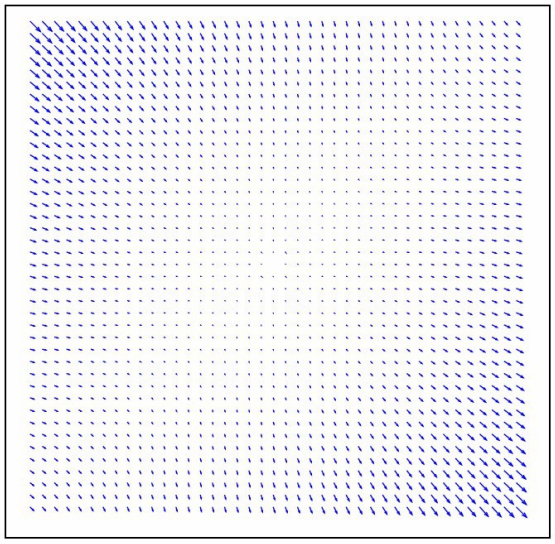

Figura 3.6: Distribución de la distorsión tangencial en una imagen. Imagen tomada de: https://www.i-ciencias.com/pregunta/96683/que-es-el-tangencial-distorsion-de-opencv-en-realidad-tangencial.

carga de color que tiene la matiz. Se puede decir que la saturación es una medida de qué tan puro es un color si éste es diluido en blanco.

- *Brillo: (Brightness)* El brillo es un atributo de la iluminación en la cual un objeto no aislado se ve afectado, es notable cuando un objeto de un mismo color cambia su tonalidad debido a las variaciones de iluminación de su entorno.

## Espacios de color

El *espacio de color* es un modelo utilizado para facilitar la especificación de cualquier color de una manera estandarizada. Uno de los sistemas más conocidos, es el espacio de color RGB (por sus siglas en inglés red, green y blue), que está basado en un sistema coordenado ortogonal (como se observa en la Figura 2.3) en donde la escala de los colores primarios está cada uno en los ejes.

El espacio de color *HSV (Hue-Saturation-Value)* es especificado por tres números que corresponden a la matiz (Hue), saturación (saturation) y el valor

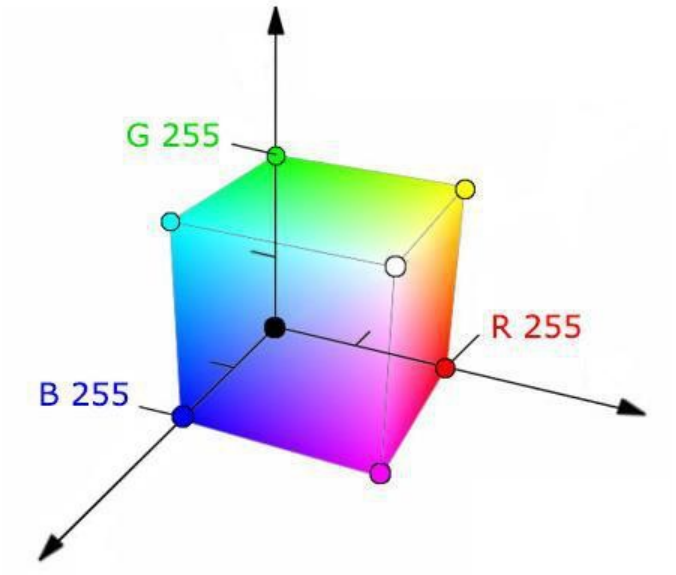


Figura 3.7: Modelo tridimensional del espacio de color RGB. Imagen tomada de: https://lpurpura.wordpress.com/2011/05/03/modo-de-color/ .

(value). La matiz corresponde a un ángulo de 0 a 360 grados. La saturación se toma entre valores de 0 a 1 que miden la salida de la matiz del blanco. El valor (value) que va del 0 al 1 mide la salida de la matiz del negro o *color de energía cero*, véase la Figura 2.4 en donde se observa el modelo tridimensional de este espacio.

Tanto el espacio de color RGB y el HSV pueden representar la totalidad de los colores del espectro de luz visible, no obstante para el procesamiento de imágenes se utiliza el HSV ya que es una forma más sencilla para segmentar colores y es una forma más parecida a la que los seres humanos percibimos los colores, Prince (2012).

## 3.4. Operadores morfológicos

La gran mayoría de las veces, cuando se hace procesamiento de imágenes, se necesitan ciertos procesos para eliminar el ruido circundante y dejar al elemento de interés aislado. Los *operadores morfológicos* (erosión y dilatación)

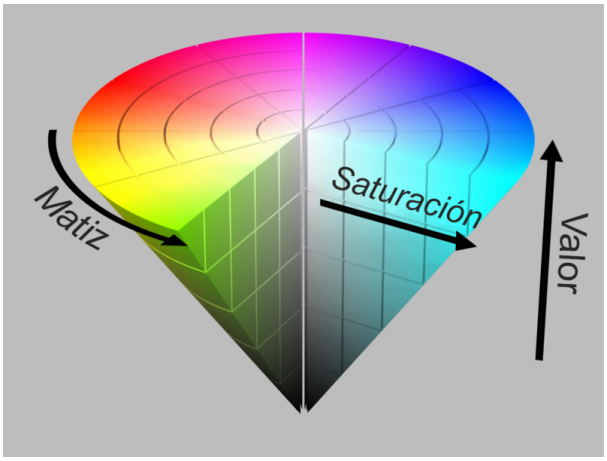


Figura 3.8: Modelo tridimensional del espacio de color HSV. Imagen tomada de: *https* : *//es.wikipedia.org/wiki/Modelo_de_color_HSV* .

se pueden usar para este propósito.

Los *operadores morfológicos* son operadores matemáticos basados en la forma de una imagen comúnmente binaria (con pixeles blancos y negros) y sirven para preservar sus características y eliminar las irrelevancias. Dado que toda imagen está compuesta por pixeles, se pueden crear ciertos conjuntos de pixeles dentro de un vector para así poder aplicar operaciones matemáticas (Heijmans, 1999).

La *erosión* es el primer operador que se aplica cuando se desea eliminar el ruido en una segmentación, eso es porque elimina una gran cantidad de pixeles que tienden a tener mucha menor área que la figura de interés. Para lograr esto se establecen dos conjuntos (de coordenadas de pixeles) $A$ y $B$. Si $A$ y $B$ son conjuntos dentro de un espacio euclidiano de N elementos $E^N$, entonces la erosión de $A$ por $B$ es el conjunto de elementos $x$ para el cual $x + b \in A$ para cada $b \in B$. Por lo tanto, de una manera más formal, la operación de erosión se puede definir con la expresión (3.1). En la Figura 3.9 se puede observar de una manera más visual cómo funciona la operación de erosión dentro de una imagen binaria.

$$A \ominus B = \left\{ x \in E^N \mid x + b \in A \ \forall \, b \in B \right\} \tag{3.1}$$

Para comprender de mejor manera cómo es esta operación se debe entender lo que es un *kernel*. Un *kernel* o *matriz de convolución* es un conjunto de

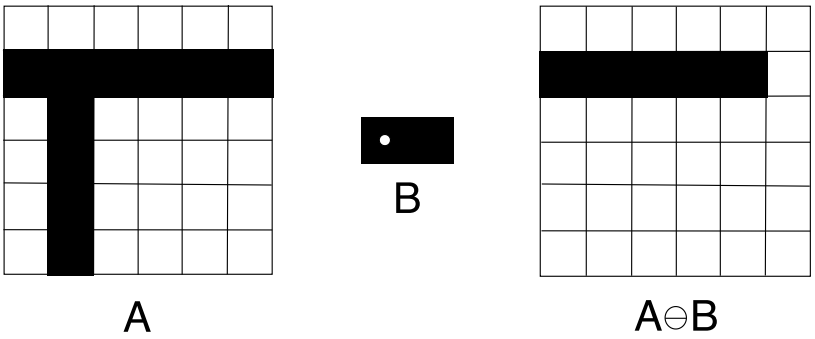


Figura 3.9: Ejemplo de la operación de erosión en una imagen binaria.

pixeles de cualquier forma o tamaño que se caracteriza por tener un pixel de anclaje dentro de sí (en la Figura 3.9 es el pixel con un punto blanco), esta matriz se utiliza para desenfocar, realzar y detectar bordes dentro de una imagen (Szeliski, 2010). En el ejemplo de la Figura 3.9 se le llamará *kernel* al conjunto $B$ y siempre debe ser de menor tamaño que el conjunto $A$. La intersección del conjunto $A$ con el $B$ al ir colocando el pixel de anclaje en cada pixel del conjunto $A$ es la imagen resultante para la erosión.

En la Figura 3.10 se ve que la erosión ha eliminado el ruido y la imagen se ve más limpia de puntos blancos que rodean la región de interés. El problema de hacer este procedimiento es que la imagen original suele verse afectada y tiende a encogerse o a deformarse, esto puede representar un problema para la detección de contornos. Para solucionarlo se procede a aplicar una operación opuesta a la erosión, mejor conocida como (*dilatación*).

Una vez eliminado el ruido gracias a la erosión, la *dilatación* procede a ensanchar la figura de interés utilizando la adición de dos conjuntos de elementos. De la misma manera en que se hizo la erosión se procede a nombrar dos conjuntos $A$ y $B$. Si $A$ y $B$ son conjuntos dentro de un espacio euclidiano de N elementos. Sean $a \in A$ y $b \in B$ en donde $a = (a_1, ..., a_n)$ y $b = (b_1, ..., b_n)$, es decir conjuntos con coordenadas de pixeles, entonces la dilatación de $A$ por $B$ es el conjunto de todas las posibles sumas de elementos, de $A$ y $B$. De forma formal se expresa como 3.2.

$$A \oplus B = \left\{ c \in E^N \mid c = a + b \quad \forall \quad a \in A \; y \;\; b \in B \right\} \tag{3.2}$$

En la Figura 3.11 se puede imaginar que si se coloca el *kernel* dentro de los pixeles pertenecientes al conjunto $A$ y se suman, queda como resultado de

Figura 3.10: Eliminación de ruido alrededor de una figura de interés usando la erosión. Imagen tomada de: *https* : *//docs.opencv.org/*3.4/*d*9/*d*61/*tutorial_py_morphological_ops.html* .

$A \oplus B$ una imagen ensanchada, lo que puede ayudar a que la figura erosionada tenga mayor parecido a la imagen original Haralick, Sternberg, y Zhuang (1987). En la Figura 3.12 está un ejemplo de cómo se ensancha una figura al aplicar la dilatación.

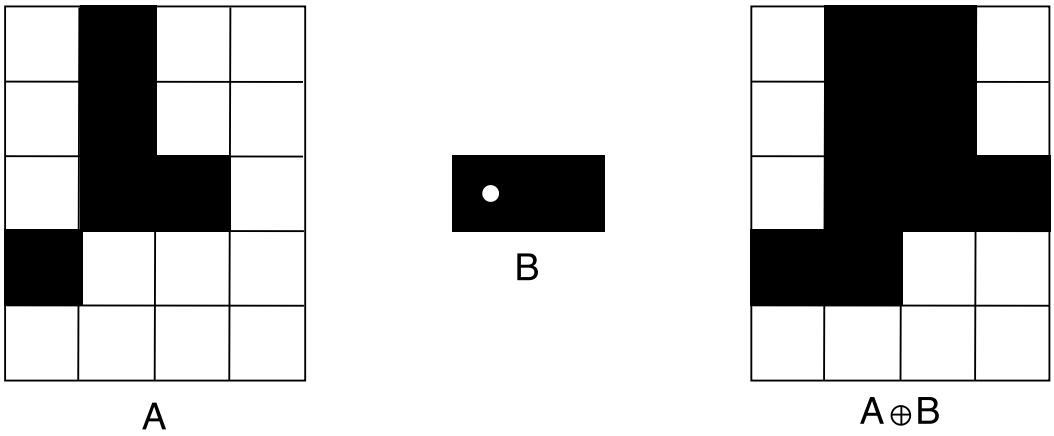


Figura 3.11: Ejemplificación de la operación de dilatación en una imagen binaria.

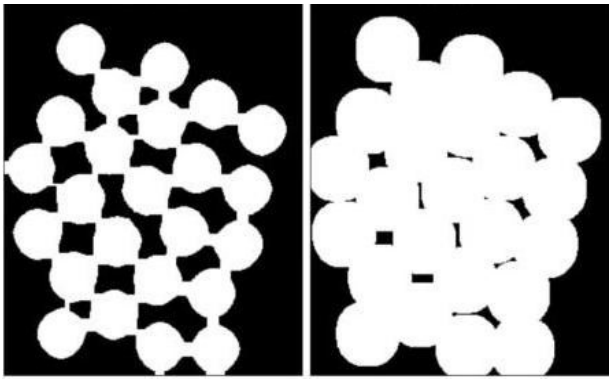

Figura 3.12: Ejemplificación de la dilatación aplicada en una segmentación de color. Imagen tomada de: $https://unipython.com/segmentacion-imagenes-algoritmo-watershed/$ .

# Capítulo 4

# Estimación de posición y velocidad

En este capítulo se presentan las deducciones para el modelo matemático de posición y velocidad del objeto a segmentar es decir, un balón de fútbol, utilizando herramientas de geometría analítica espacial y de la plataforma ROS para realizar las transformaciones homogéneas del robot. Con base en el modelado matemático se hace un análisis del Filtro de Kalman para obtener sus parámetros y se decide si se debe usar el normal o el Filtro de Kalman Extendido.

## 4.1. Cálculo de la posición del objeto de interés

Tal como se observa en la simulación de Gazebo (Figura 4.1), se decidió usar un sistema ortogonal derecho en la base de los pies como sistema de referencia que gobernará a todo el modelo.

La cámara está localizada en la cabeza del humanoide, por lo que su centro de visión se puede representar por un eje que va de la cámara al centroide del objeto, en este caso un balón de fútbol. Para estimar la posición de un objeto que cruce por el centro de visión de la cámara se necesita establecer un vector unitario:

$$\hat{u} = (u_x, u_y, u_z)$$

conocido en computación gráfica como vector *look at*, (ver Figura 4.2). Para

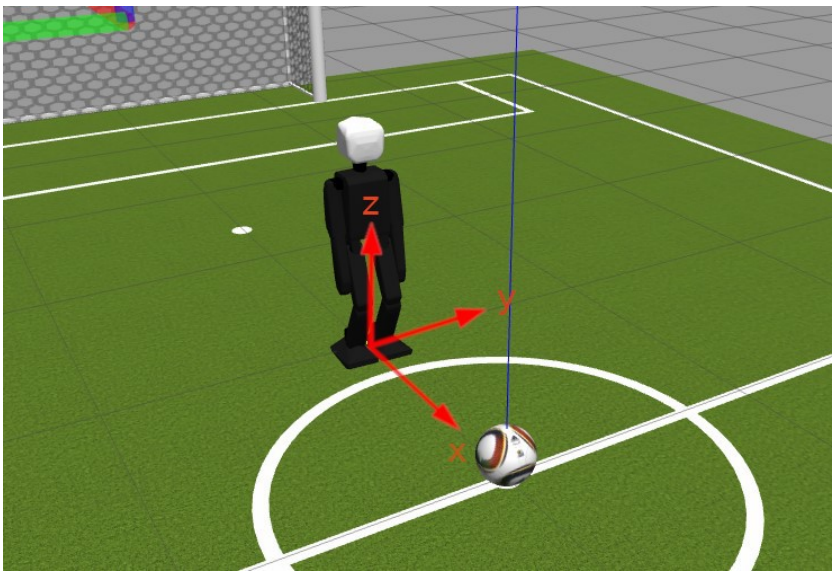


Figura 4.1: Representación del sistema de referencia usado en el robot.

obtener la ecuación vectorial de la recta paralela al vector *look at* se toma un punto que contenga la recta, en este caso la posición de la cabeza en donde se encuentra la cámara:

$$r(\lambda) = (r_x, r_y, r_z) \quad + \quad \lambda(u_x, u_y, u_z) \tag{4.1}$$

En donde $(r_x, r_y, r_z)$ es la posición en el espacio de la cámara utilizada, referida al sistema de referencia anteriormente mencionado. Se considera que el objetivo siempre estará en el suelo, por lo que el punto de intersección de la ecuación de la recta con el suelo hacen que la variable de altura sea igual a cero, conforme a la siguiente expresión:

$$r = (x, y, 0) = (r_x, r_y, r_z) + \lambda(u_x, u_y, u_z)$$

Despejando $\lambda$ del tercer término de la expresión anterior se obtiene:

$$\lambda = -\frac{r_z}{u_z}$$

De esta manera, substituyendo $\lambda$ en (4.1):

$$x = r_x - \frac{r_z u_x}{u_z}$$

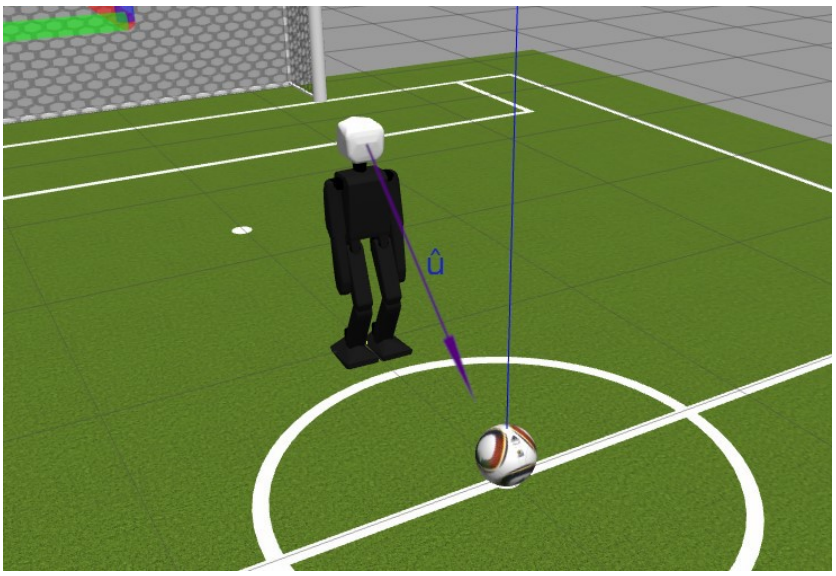


Figura 4.2: Representación del vector *look at* utilizado en visión computacional.

$$y = r_y - \frac{r_z u_y}{u_z}$$

Ya teniendo estas expresiones se procede a sustituir al vector unitario *look at* con coordenadas esféricas, tal y como se observa en la siguiente expresión:

$$\hat{u} = (u_x, u_y, u_z) = (\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta) \tag{4.2}$$

Sustituyendo la expresión (4.2) dentro de los valores $x$ y $y$ da como resultado:

$$x = r_x - \frac{r_z \sin\theta\cos\varphi}{\cos\theta}$$

$$y = r_y - \frac{r_z \sin\theta\sin\varphi}{\cos\theta}$$

Utilizando la identidad trigonométrica:

$$\tan\theta = \frac{\sin\theta}{\cos\theta}$$

Las ecuaciones para obtener la posición del objetivo siempre y cuando $z$ sea igual a cero, son:

$$x = r_x - r_z \tan\theta\cos\varphi \tag{4.3}$$

$$y = r_y - r_z \tan\theta \sin\varphi \tag{4.4}$$

No obstante, el objeto a considerar no es un objeto bidimensional, es un balón con forma esférica que está sobre el plano del suelo, debido a esto se pueden complementar las ecuaciones (4.3) y (4.4), para obtener la posición (x,y,z) del centro del balón.

En la Figura 4.3(a) se observa un diagrama del balón en donde el vector *look at* atraviesa su centro e intersecta con el suelo en un punto en donde el balón no está. Esto representa un problema, ya que la posición en $x$ sufre una proyección, la cual incrementa mientras el balón tenga mayor radio, de acuerdo a la expresión (4.5). A esta proyección se le pondrá la variable $x_c$. Véase la Figura 4.3(b)

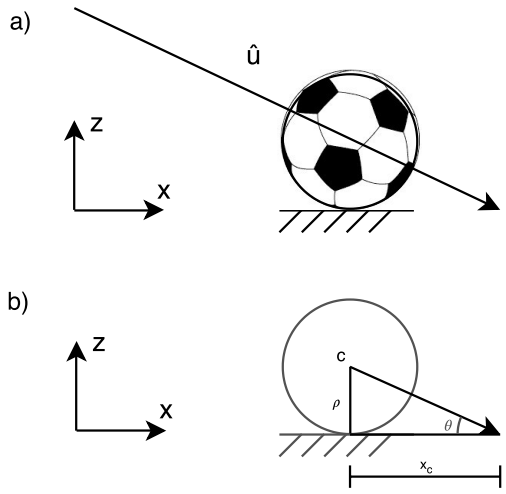


Figura 4.3: (a) Bosquejo del vector *look at* atravesando el centro del balón esférico; (b) Diagrama de la obtención de la distancia de proyección $x_c$.

Para obtener la magnitud de $x_c$ es necesario analizar el diagrama de la Figura 4.3(b), en donde $\theta$ es el ángulo *pitch* y $\rho$ es el radio del balón. Así se puede obtener la siguiente igualdad:

$$\cot\theta = \frac{x_c}{\rho}$$

Despejando $x_c$

$$x_c = \rho \cot \theta \tag{4.5}$$

Debido a que $x_c$ se ve afectada también por la posición *yaw* se deducen las siguientes igualdades para corregir el valor de la posición de $x$ y $y$ respectivamente:

$$x_{cx} = \rho \cot \theta \cos \varphi \tag{4.6}$$

$$x_{cy} = \rho \cot \theta \sin \varphi \tag{4.7}$$

De esta manera, se resta (4.6) a (4.3) y (4.7) a (4.4). Finalmente las ecuaciones resultantes son:

$$x = r_x - r_z \tan \theta \cos \varphi - \rho \cot \theta \cos \varphi$$

$$y = r_y - r_z \tan \theta \sin \varphi - \rho \cot \theta \sin \varphi$$

$$z = \rho$$

## 4.2. Estimación de estados mediante el Filtro de Kalman

Para poder implementar el Filtro de Kalman, se necesita primero tener un modelo matemático del sistema que se pretende tomar mediciones. Analizando el diagrama de cuerpo libre de la Figura 4.4 se puede hacer un análisis dinámico del balón utilizando la segunda ley de Newton.

$$\sum F = m\ddot{x} \tag{4.8}$$

$$F - f_{fricción} = m\ddot{x} \tag{4.9}$$

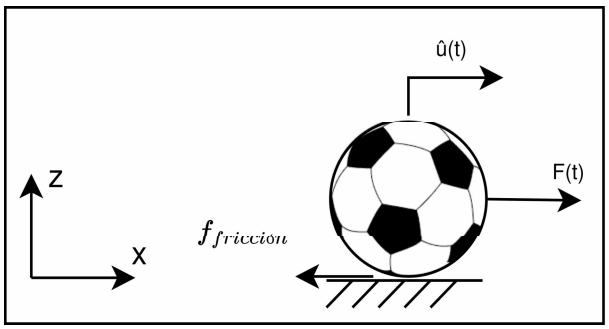


Figura 4.4: Diagrama de cuerpo libre del objeto de interés

Cambiando el nombre de las variables acorde de nuestro diagrama y tomando en como fuerza de fricción dinámica $f_{fricción} = \mu_d mg$ la igualdad de fuerzas es:

$$F - \mu_d mg = m\frac{\mathrm{d}\dot{x}}{\mathrm{d}t} \tag{4.10}$$

Donde $\mu_d$ es el coeficiente de fricción dinámica, $m$ es la masa del balón y $g$ es la aceleración de la gravedad. Debido a que la fuerza de fricción es la única que actúa sobre el balón, $F$ se iguala a cero, dejando el modelo el modelo cinemático de la expresión (4.11).

$$\frac{\mathrm{d}\dot{x}}{\mathrm{d}t} = -\mu_d g \tag{4.11}$$

Despejando la variable $\dot{x}$ para se puede integrar la ecuación para obtener la velocidad en el tiempo $t$ del balón.

$$\int_{\dot{x}_0}^{\dot{x}} \mathrm{d}\dot{x} = -\mu_d g \int_0^t \mathrm{dt} \tag{4.12}$$

Resolviendo la ecuación (4.12) y despejando $\dot{x}$ se puede obtener la siguiente expresión:

$$\dot{x} = \dot{x}_0 - \mu_d g t \tag{4.13}$$

Una vez teniendo el modelo para predecir la velocidad del estado siguiente, se procede a integrar nuevamente para obtener su posición:

$$\int_{x_0}^{x} \mathrm{dx} = \int_0^t (\dot{x}_0 - \mu_d g t)\mathrm{dt} \tag{4.14}$$

Integrando y despejando $x$ se obtiene:

$$x = x_0 + \dot{x}_0 t - \frac{1}{2}\mu_d g t^2 \tag{4.15}$$

Ya obtenida la ecuación para obtener la posición, despejamos la aceleración del modelo (4.10):

$$\ddot{x} = -\frac{1}{m}\mu_d m g + \frac{1}{m}F \tag{4.16}$$

Pero $F$ es igual a cero. Es decir, que simplemente el balón comienza con $v_0$ diferente de cero y se va deteniendo. Planteando esto en variables de estado:

$$[x_1, x_2] = [x, \dot{x}]$$

En donde:

$$\dot{x}_1 = x_2 \tag{4.17}$$

y

$$\dot{x}_2 = -\frac{1}{m}\mu_d m g + \frac{1}{m}F \tag{4.18}$$

### Descripción del proceso de filtrado en el sistema

Con el modelo dinámico que describe el movimiento del balón, se procede a escribir las ecuaciones matriciales del Filtro de Kalman (véase la Sección 2.4). De este modo, este filtro es una serie de pasos que iterativamente se deben ir completando para estimar posiciones y velocidades de un objeto en movimiento (Figura: 4.5).

### Modelado del sistema

Para desarrollar las ecuaciones de predicción de estado (ver Figura 4.5), se considerará que el balón se desplazará en el suelo sin elevarse o tener rebotes, por lo que el vector de estado $\hat{x}_n$ que describe la estimación de posición y velocidad en el plano *(x, y)* es (4.19), siguiendo la nomenclatura que se vio en la Sección 2.4.

$$\hat{x}_{n,n} = \begin{bmatrix} x_{1_{n,n}} \\ y_{1_{n,n}} \\ x_{2_{n,n}} \\ y_{2_{n,n}} \end{bmatrix} \tag{4.19}$$

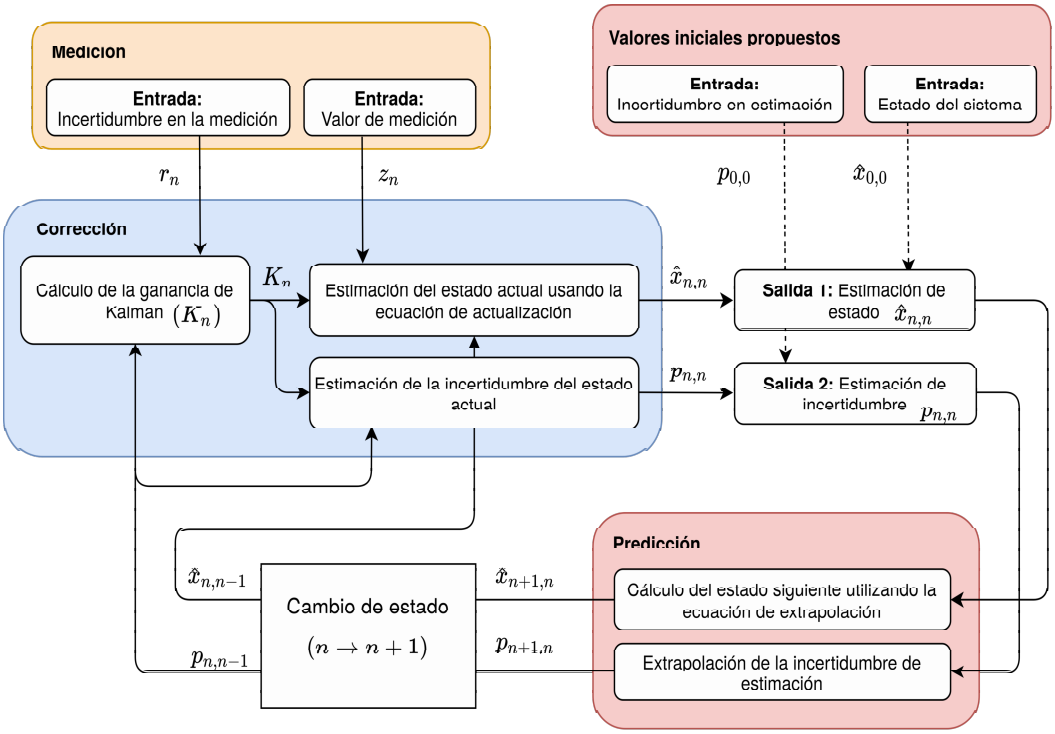


Figura 4.5: Diagrama extendido del funcionamiento paso a paso del Filtro de Kalman. Imagen modificada de: https://www.kalmanfilter.net/kalman1d.html

Dado que la única fuerza de entrada en el balón es la fricción, la cual depende del coeficiente de fricción dinámica y la gravedad el vector $\hat{u}$ es representado como:

$$\hat{u}_n = \begin{bmatrix} F_x \\ F_y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4.20}$$

Como se vio en la Sección 2.4, Figura 2.4, la ecuación de extrapolación de estado es:

$$\hat{x}_{n+1,n} = F\hat{x}_{n,n} + G\hat{u}_{n,n} + \omega_n \tag{4.21}$$

No obstante, con la deducción de la ecuación de posición, el modelo predictivo (4.15) no es lineal, por lo que se tiene emplear el Filtro de Kalman Extendido, es decir que la ecuación (4.21) se transforma en (4.22).

$$\hat{x}_{n+1,n} = f(\hat{x}_{n,n}, \hat{u}_{n,n}) \tag{4.22}$$

La igualdad que se encuentra en (4.18) es el modelo en variables de estado, continuo. Para el EKF se discretiza este modelo. Y la señal de entrada $F$ igual a cero.

El modelo discreto del sistema con ruido es:

$$\begin{aligned}
x_{1_{n+1,n}} &= x_{1_{n,n}} + \Delta t x_{2_{n,n}} + \omega_1 \\
y_{1_{n+1,n}} &= y_{1_{n,n}} + \Delta t y_{2_{n,n}} + \omega_2 \\
x_{2_{n+1,n}} &= x_{2_{n,n}} - \Delta t \frac{1}{m}\mu_d mg + \frac{1}{m}F_x + \omega_3 \\
y_{2_{n+1,n}} &= y_{2_{n,n}} - \Delta t \frac{1}{m}\mu_d mg + \frac{1}{m}F_y + \omega_4
\end{aligned}$$

O escrito en su forma matricial:

$$\bar{x}_{n+1,n} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1_{n,n}} \\ y_{1_{n,n}} \\ x_{2_{n,n}} \\ y_{2_{n,n}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\Delta t \mu_d g \\ -\Delta t \mu_d g \end{bmatrix} + \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \tag{4.23}$$

$\Omega = [\omega_1, ..., \omega_4]$ es un vector de ruido con distribución normal, media cero, y matriz de covarianza $Q \in \mathbb{R}^{4\times4}$.

Para estas ecuaciones se tienen dos entradas, que son las posiciones en $x$ y $y$ con su respectiva incertidumbre de medición, para que estas entradas tengan el mismo sistema de coordenadas y las mismas unidades, generalmente

deben estar dentro de una función de transformación $h$, sin embargo, como la entrada no requiere un mapeo de valores, el vector $z_n$ es:

$$z_1 = h_1(x_1) + v_1 = x_1 + v_1 \tag{4.24}$$

$$z_2 = h_2(y_1) + v_1 = y_1 + v_1 \tag{4.25}$$

o en forma matricial:

$$z_n = \begin{bmatrix} h_1(x_1) \\ h_2(y_1) \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{4.26}$$

$V = [v_1, v_2]$ es el vector de ruido de la medición, considerando que tiene una distribución normal, media cero y matriz de covarianza $R_n \in \mathbb{R}^{2\times 2}$.

Para poder definir la matriz de predicción $F$, la cual será de apoyo para el estado siguiente de la matriz $P$ se tiene que tomar en cuenta que la ecuación (4.15) no es lineal por no cumplir con la condición de homogeneidad, lo cual hace que distribución no sea gaussiana y la predicción resultaría errónea. Para solucionar esto, es necesario hacer un procedimiento para linealizar el modelo utilizando la serie de Taylor con sólo los dos primeros términos (dado que si se agregan más el modelo dejaría de ser lineal otra vez).

La serie de Taylor aplicada con los dos primeros términos es:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x - a) \tag{4.27}$$

Ahora al substituir $f(x)$ con nuestro modelo de predicción cercano a un punto $a$:

$$f(\hat{x}_{n,n}) = f(\hat{x}^a_{n,n}) + F(\hat{x}^a_{n,n})(\hat{x}_{n,n} - \hat{x}^a_{n,n}) \tag{4.28}$$

F es el Jacobiano:

$$F = \begin{bmatrix} \frac{\partial f_{x_1}}{\partial x_1} & \frac{\partial f_{x_1}}{\partial y_1} & \frac{\partial f_{x_1}}{\partial x_2} & \frac{\partial f_{x_1}}{\partial y_2} \\ \frac{\partial f_{y_1}}{\partial x_1} & \frac{\partial f_{y_1}}{\partial y_1} & \frac{\partial f_{y_1}}{\partial x_2} & \frac{\partial f_{y_1}}{\partial y_2} \\ \frac{\partial f_{x_2}}{\partial x_1} & \frac{\partial f_{x_2}}{\partial y_1} & \frac{\partial f_{x_2}}{\partial x_2} & \frac{\partial f_{x_2}}{\partial y_2} \\ \frac{\partial f_{y_2}}{\partial x_1} & \frac{\partial f_{y_2}}{\partial y_1} & \frac{\partial f_{y_2}}{\partial x_2} & \frac{\partial f_{y_2}}{\partial y_2} \end{bmatrix} \tag{4.29}$$

Substituyendo la función $f$ en la matriz $F$ da como resultado:

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.30}$$

**Extrapolación de la incertidumbre de estimación**

El proceso de estimación en el Filtro de Kalman es un modelo basado en la *esperanza* de una serie de variables aleatorias para obtener el valor *oculto* que se considera el real. Este proceso de filtrado considera que todos los errores tanto en medición como en estimación tienen una *distribución gaussiana* por lo que cada incertidumbre tiene que expresarse como la varianza de una recompilación de datos. De una manera análoga a la predicción de posición y velocidad en un tiempo $\Delta t$ la incertidumbre de estimación se puede calcular con la siguiente ecuación matricial:

$$P_{n+1,n} = F P_{n,n} F^T + Q \tag{4.31}$$

Donde $P$ es la matriz de covarianza:

$$P_{n,n} = \begin{bmatrix} \sigma_{x_1}^2 & 0 & \sigma_{x_1}\sigma_{x_2} & 0 \\ 0 & \sigma_{y_1}^2 & 0 & \sigma_{y_1}\sigma_{y_2} \\ \sigma_{x_2}\sigma_{x_1} & 0 & \sigma_{x_2}^2 & 0 \\ 0 & \sigma_{y_2}\sigma_{y_1} & 0 & \sigma_{y_2}^2 \end{bmatrix} \tag{4.32}$$

Nótese que a comparación de las otras matrices de covarianza, en ésta hay varianzas y covarianzas, esto se debe a que se tienen sólo entradas de posición es decir, las velocidades están correlacionadas a la medición de posición.
Q es una matriz diagonal cuadrada:

$$Q = \begin{bmatrix} q_{x_1} & 0 & 0 & 0 \\ 0 & q_{y_1} & 0 & 0 \\ 0 & 0 & q_{x_2} & 0 \\ 0 & 0 & 0 & q_{y_2} \end{bmatrix} \tag{4.33}$$

**Obtención de la ganancia de Kalman**

Una vez teniendo la predicción del estado siguiente, es posible comparar las mediciones con las predicciones previamente hechas para hacer un proceso

de corrección. Para esto es necesario obtener el peso de la ganancia de Kalman que se describe con la siguiente ecuación:

$$K_n = P_{n,n-1}H^T(HP_{n,n-1}H^T + R_n)^{-1} \tag{4.34}$$

La matriz de observación $H$ es la que mapea los valores de la estimación para que se puedan sumar con los valores de la medición, está definida como el siguiente jacobiano:

$$H = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial y_1} & \frac{\partial h_1}{\partial x_2} & \frac{\partial h_1}{\partial y_2} \\ \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial y_1} & \frac{\partial h_2}{\partial x_2} & \frac{\partial h_2}{\partial y_2} \end{bmatrix} \tag{4.35}$$

Substituyendo con los valores de la función $h$ el jacobiano es:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{4.36}$$

Como ya se había mencionado, $R_n$ es una matriz cuadrada de 2x2 que contiene la covarianza del ruido en la medición:

$$R_n = \begin{bmatrix} R_x & 0 \\ 0 & R_y \end{bmatrix} \tag{4.37}$$

### Corrección de estimación de estado

Con cada nueva medición, se procede a hacer la corrección del estado presente, haciendo uso del la ganancia del filtro para poder *decidir* si es más fiable la medición o la predicción.

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n(z_n - h(\hat{x}_{n,n-1})) \tag{4.38}$$

### Corrección del error de covarianza

De una manera muy similar el error de covarianza se puede actualizar utilizando la ganancia de Kalman:

$$P_{n,n} = (I - K_nH)P_{n,n-1} \tag{4.39}$$

## 4.3. Obtención de los parámetros

### Coeficiente de fricción dinámica

De acuerdo con la ecuación (4.15) y como se vio a lo largo de su deducción, el modelo de la posición del balón respecto al tiempo, depende únicamente del coeficiente de fricción dinámica entre el balón y el material el cual se desplaza. Para obtener dicho coeficiente se optó por medir directamente la fuerza que se opone al desplazamiento, por medio de un dinamómetro de resorte marca *Pasco* con rango de 0 a 10[N] y una resolución de 0.1[N]. Véase la Figura 4.6.

Figura 4.6: Dinamómetro de resorte utilizado para obtener la fuerza de fricción. Imagen tomada de: https://tecnoedu.com/Pasco/DinamometrosECYT.php

Las medidas del dinamómetro del peso del balón dio $2.2[N]$, de la fuerza de fricción estática $F_e = 1.5[N]$ y de la fuerza de fricción dinámica $F_d = 0.8[N]$.

Por lo que se puede aplicar la segunda ley de Newton (de manera simplificada) para despejar el coeficiente de fricción deseado:

$$F_d = N\mu_d \tag{4.40}$$

En donde $N$ es la fuerza normal a la superficie con la misma magnitud del peso del balón (por considerar que se está desplazando de manera horizontal). Por tanto la fuerza de fricción dinámica es:

$$\mu_d = \frac{F_d}{N} = 0.3636 \tag{4.41}$$

## Parámetros de Kalman

El diagrama de bloques de la Figura 4.7 muestra que el Filtro de Kalman es un sistema de control automático realimentado con su misma salida. A fin de obtener resultados parecidos a los de la realidad, los parámetros que se presentan a continuación se establecieron con base en la experiencia del error en el sensado y la estimación. En una implementación real, los parámetros se deben tomar con base en la estadística de un determinado número de muestras para encontrar el ruido en un ambiente controlado.

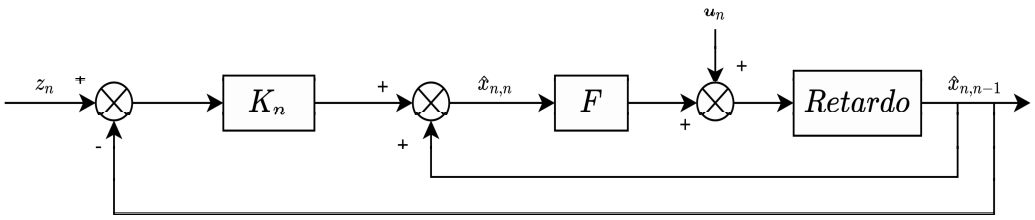


Figura 4.7: Diagrama de bloques del proceso de Kalman

### Matriz de covarianza del ruido del proceso

$$Q = \begin{bmatrix} 0.000002 & 0 & 0 & 0 \\ 0 & 0.000002 & 0 & 0 \\ 0 & 0 & 0.000002 & 0 \\ 0.0 & 0 & 0 & 0.000002 \end{bmatrix} \tag{4.42}$$

### Matriz de error de medición

$$R_n = \begin{bmatrix} 0.00002 & 0 \\ 0 & 0.00002 \end{bmatrix} \tag{4.43}$$

### Valores iniciales de entrada

Con base en la experimentación de diversas pruebas del algoritmo con el simulador, los valores iniciales de entrada (4.44) y (4.45) se seleccionaron como valores cercanos a la condición inicial, a fin que el filtro converja lo suficientemente rápido y poder predecir cuándo se debe patear al balón.

$$\hat{x}_{0,0} = \begin{bmatrix} -0.3 \\ -0.3 \\ 0.3 \\ 0.3 \end{bmatrix} \tag{4.44}$$

$$P_{0,0} = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix} \tag{4.45}$$

En el siguiente capítulo se discute la implementación del Filtro de Kalman Extendido con los resultados expresados de manera numérica y gráfica.

# Capítulo 5

# Implementación

Al inicio de este capítulo se hace descripción del hardware del humanoide que se toma como base para esta tesis, así como las características de software con las que cuenta. Seguido de esto, se hace una explicación de las bibliotecas de OpenCV y las principales funciones que se emplean como herramienta de software para hacer la segmentación por color y localización en pixeles del objeto segmentado.

Por último, se hace una breve explicación de la plataforma ROS, su historia y las características que tiene para la integración de los diferentes programas a fin de hacer una buena distribución de los diversos procesos y hacer más sencilla su programación.

## 5.1. El robot bípedo Nimbro OP

La implementación de los algoritmos de visión artificial está planeada para usarse para el robot bípedo Nimbro OP (Open Platform) que está diseñado para jugar fútbol, en competencias como la RoboCup *Humanoid League* y es compatible con la categoría *TeenSize*.
*Nimbro-OP* (Figura 5.1) fue el primer prototipo para modular a los humanoides *open-source* para investigación y educación, para más información visite su página inicial $https://www.nimbro.net/OP/$.

### Características de hardware

- Cuenta con una altura de 95cm y un peso aproximado de 6.6kg.

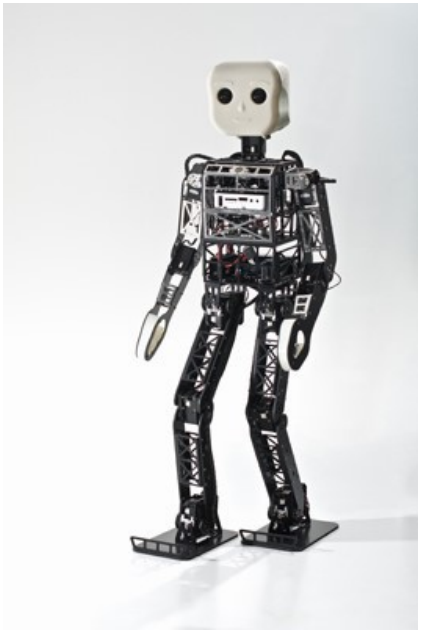

Figura 5.1: Robot humanoide tipo Nimbro-OP. Imagen tomada de: http://www.nimbro.net/OP/NimbRo-OP.html

- 20 actuadores interconectados (motores Dynamixel).
  - 6 por cada pierna (MX-106)
  - 3 por cada brazo (MX-64)
  - 2 en el cuello (MX-64)
- PC con doble núcleo (Zotac ZBOX nano XS)
  - Procesador AMD E-450(2x1.65GHz)
  - 2GB RAM, 64GB SSD
  - USB 3.0, HDMI, Gigabit Ethernet
- WiFi: IEEE 802.11b/g/n
- Cámara de amplio ángulo de visión (Logitech C905)
- Sensores de inercia (Dentro del controlador Ronotis CM-730)

  - Acelerómetro de 3 ejes
  - Giroscopio de 3 ejes
- Batería de Litio de 11.1V a 4.5Ah
- Chasis de fibra de carbono, aluminio y ABS plus

**Características de software**

- El software es compatible tanto para Linux como Windows.
- Tiene un desarrollo basado en la plataforma ROS para proporcionar la abstracción de hardware.
- Cuenta con software de código abierto que *Robotis* lanzó para DARwin-OP para comportamientos y habilidades básicas en fútbol.

## 5.2. La bilbioteca OpenCV

OpenCV (*Open Computer Vision*) es una biblioteca *open source* diseñada para visión computacional. Esta biblioteca está escrita principalmente en C y C++ y es capaz de ejecutarse en diversos sistemas operativos, tales como: *Windows*, *Linux*, o *Mac OS X*.

OpenCV fue desarrollada para hacer procesos más eficientes cuando se hace visión con aplicaciones en tiempo real. Una de las metas de OpenCV es proveer una infraestructura sencilla y sofisticada para diferentes tipos de usuarios, que van desde profesores, estudiantes, profesionistas, desarrolladores, y autodidactas. La librería cuenta con más de 500 funciones que se extienden en diversas áreas de visión, incluyendo inspección en la fabricación de productos, seguridad, calibración de cámaras, visión estéreo, robótica, etc. (Culjak, Abram, Pribanic, Dzapo, y Cifrek, 2012).

Las principales funciones de OpenCV que se utilizaron para la elaboración de este trabajo fueron: *cv::calibrateCamera()* para hacer la corrección de las distorsiones (véase la sección 3.2), *cv::cvtColor()* para cambiar del espacio de color RGB al HSV, *cv::InRange()* para realizar la segmentación de color del objeto de interés (véase la sección 3.3), *cv::erode()* y *cv::dilate()* para implementar los operadores morfológicos, junto con *cv::findNonZero()* para obtener el centroide de la figura proyectada en la imagen.

## 5.3. La plataforma ROS

### ¿Qué es ROS?

Citando a Pyo, Cho, Jung, y Lim (2015) ROS es un meta sistema operativo *open-source* que provee servicios a las aplicaciones de robótica, servicios que comúnmente se esperan de un sistema operativo, tales como: abstracción de hardware, control de dispositivos a *bajo nivel*, paso de mensajes entre procesos, ordenamiento y manejo de distintos tipos de paquetes. ROS también provee herramientas y bibliotecas para obtener, construir, escribir y ejecutar programas a través de múltiples computadoras.

ROS es la abreviación en inglés de *Robot Operating System* lo cual se podría traducir al español como Sistema Operativo de Robots. Se podría pensar que ROS es un sistema operativo, sin embargo el término mejor empleado es el de *Meta Sistema Operativo*, y aunque no está definido en el diccionario, se puede describir como un sistema que realiza procesos tales como programación, ejecución, monitoreo, y manejo de errores, utilizando una capa de visualización entre aplicaciones y recursos informáticos distribuidos.

Dicho lo anterior, ROS no es un sistema operativo convencional, tal como *Windows*, *Linux*, o *Android*, sino una plataforma que se ejecuta dentro del sistema operativo instalado. A menudo, para utilizar ROS se requiere tener instalado *Ubuntu*, que es un sistema basado en las distribuciones de *Linux*. No obstante, es posible usarse en distintos sistemas, tal y como se muestra la Figura 5.2.

### Objetivos al utilizar ROS

Aunque existen diversas plataformas de robótica (OpenRTM, OPRoS, Player, Orca, Microsoft Robotics Studio, etc.), ROS está orientado a construir entornos de desarrollo para software de robótica a un nivel global, con esto se espera que el código de diferentes desarrolladores se pueda usar, modificar o mejorar para hacer crecer el entorno mismo, es por eso que posee las siguientes características:

- **Distribución de procesos:** Están programados en unidades mínimas de procesamiento (nodos). Cada uno de estos procesos se ejecuta de


Figura 5.2: Esquema de la abstracción de hardware por ROS en distintos Sistemas Operativos. Imagen tomada de Pyo y cols. (2015).

manera independiente y es capaz de intercambiar datos con otros de manera sistemática.

- **Manejo de paqueterías:** Cuando varios procesos tienen propósitos similares, estos se manejan dentro de un *paquete* que los haga más ordenados y fáciles de desarrollar.

- **Repositorios públicos:** Cada paquete se hace público dentro de un repositorio (por ejemplo GitHub) para que la comunidad de desarrolladores puedan acceder a él.

- **API (Interfaz de Programación de Aplicaciones):** Cuando se desarrolla un programa en ROS, generalmente se llaman funciones ya existentes fácilmente insertarlas dentro del código que se esté construyendo.

- **Soporte de distintos lenguajes de programación:** La plataforma ROS posee una *biblioteca de clientes* para facilitar el trabajo de los pro-

gramadores. La biblioteca puede importar lenguajes de programación que son bastantes populares, tales como Python, C++, Java, Ruby, Lips, entre otros.

### Breve historia de ROS

*Robot Operating System* fue creado en Mayo del 2007 dentro del *Standford Artificial Intelligence Laboratory*. Se podría decir que el predecesor de ROS es un proyecto llamado *SwitchYard*, el cual es un software creado para el desarrollo de inteligencia artificial de robots.

En noviembre del 2007 la compañía estadounidense *Willow Garage* empezó el desarrollo de ROS dentro del campo de los robots de servicio. De esta manera ROS vino al mundo oficialmente el 22 de Enero del 2010 con la versión llamada *ROS 1.0*, sin embargo la versión más conocida fue *Box Turtle* lanzada en Marzo del mismo año.

La plataforma ROS es actualizada cada dos años (entre el lapso de Abril-Octubre), lo que significa que cada versión tiene soporte durante aproximadamente cinco años. Para propósitos de esta tesis se optó por utilizar la versión *ROS-Melodic* instalado en Ubuntu 18.04 *Bionic Beaver (LTS)*. Más información en: *https* : *//www.ros.org/history/*.

## 5.4. Integración de los diferentes programas

Para hacer uso del humanoide se requiere clonar un repositorio que se encuentra en la siguiente liga: https://github.com/mnegretev/Humanoids . Este repositorio contiene distintos programas, unos para que el hardware del humanoide se pueda comunicar con un ordenador, otros para el procesamiento de datos, concernientes a las entradas de sensores y control de actuadores e incluso para hacer pruebas remotas con el simulador.

En esta tesis se optó por seguir el desarrollo de software de la parte simulada. Haciéndolo lo suficientemente robusto para que sea capaz de usarse tanto en el robot real como en el simulado. Aunque actualmente no se pueden hacer pruebas con el robot en físico, trabajar con la parte simulada del humanoide representa claras ventajas, las cuales son:

- Complementar el simulador con plug-ins y herramientas extra para la simulación de una cámara virtual dentro del simulador Gazebo.

- Control del ruido inherente a la medición de los sistemas, ya que se puede probar la robustez del algoritmo de filtrado con diferentes niveles de ruido.

- Desarrollo del código para el simulador, para su aprendizaje y complementación de la documentación.

- Hacer el software más disponible para desarrolladores que no tengan la posibilidad de interactuar con el robot o que se encuentren en lugares distantes.

- Evitar retrasos por fallas del hardware.

## Descripción de los principales nodos usados en este trabajo

El repositorio anteriormente mencionado tiene instrucciones para la instalación del entorno de trabajo y ejecución de los programas. Para probar el simulador, el comando de inicio es:

*roslaunch surge_et_ambula humanoid_simul.launch*

Éste es un comando que levanta distintos nodos que se comunican entre sí para simular las variables físicas simuladas del robot. Cuando se levanta se puede observar que se abren algunas ventanas, las cuales son del visualizador del robot *Rviz*, el simulador con variables físicas *Gazebo*, una GUI (Graphical User Interface) para manipular las articulaciones del robot y una ventana que simula la imagen obtenida de la cámara simulada. Tal y como se observa en la Figura 5.3.

### Nodo segmentador de color

Ejecutando el comando *rosrun ball_tracker ball_tracker_simul* se alza el nodo que se subscribe a los tópicos que contienen la información de la imagen RGB obtenida por la cámara. Con esa información se procesa la imagen para segmentar un color deseado guardando los parámetro HSV en un archivo .xml.

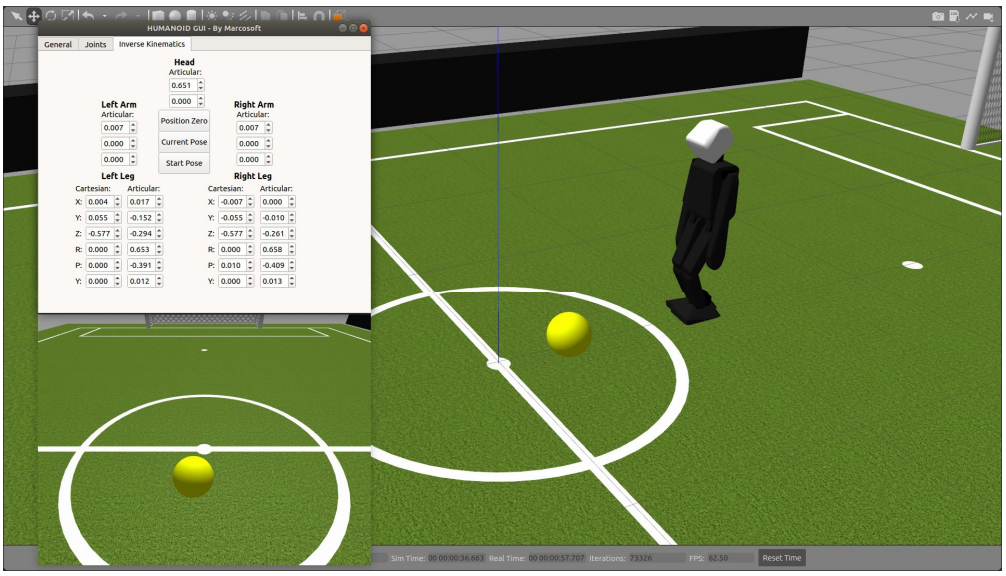


Figura 5.3: Simulador Gazebo para el humanoide.

### Nodo que calcula la posición relativa del balón

El nodo más importante de esta tesis es el que calcula la posición relativa del balón, ya que de éste depende la precisión y confiabilidad de las mediciones. Con el comando *rosrun ball_position ball_position_simul* se levanta, se subscribe a los tópicos de la imagen y hace la segmentación del objeto de interés con base en la teoría vista en el Capítulo 3.

Teniendo ya procesada la segmentación, el punto de interés de la figura segmentada es el centroide, con el cual se hacen cálculos geométricos para el posicionamiento del balón, haciendo uso de las herramientas *tf* para conocer la posición y orientación relativa de la cámara con respecto a los pies del robot. Como se ilustra en la Figura 5.4.

Al procesar la información de posición, el nodo *ball_position_simul* publica un tópico que contiene la posición $x$ y $y$ del balón, por lo que otros nodos pueden hacer uso de esa información.

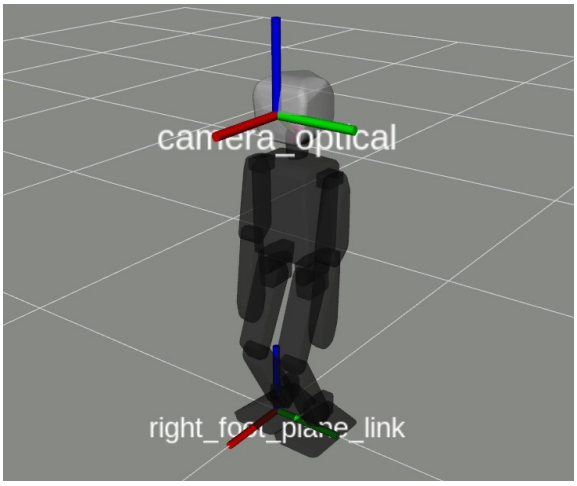


Figura 5.4: tf, herramienta de ROS para obtener la información relativa de las articulaciones de un robot (posición y orientación)

### Nodo estimador de posición y velocidad utilizando el Filtro de Kalman Extendido

El nodo llamado *kalman_estimator* es un *script* escrito en el lenguaje Python, el cual se subscribe a los tópicos de posición del balón para saber la posición instantánea del balón en los ejes $x$ y $y$. Con esto, el programa usa los parámetros de Kalman (véase la Sección 4.3) para obtener una corrección del ruido y hacer la debida predicción de posiciones y velocidades tomando una determinada cantidad de muestras con una frecuecia aproximada de 30 Hz. Para entender este nodo de una manera más visual la Figura 5.6 muestra el diagrama de flujo de los principales procesos.

Se puede revisar el código de este nodo en el **Apéndice A**. Adicionalmente a este programa, hay un código escrito en python para hacer la comparación de la posición real, la medida y la estimación mediante una gráfica, éste no es un nodo sino simplemente un *script* llamado *graph_kalman_estimator.py*.

### Nodo para movimiento del balón

Ejecutando en una nueva terminal el comando *rosrun ball_position move_ball_node* se inicializa un nodo que mueve el balón simulado a una dirección y velocidad inicial determinada, con base el lo que se vio en la Sección 4.2.

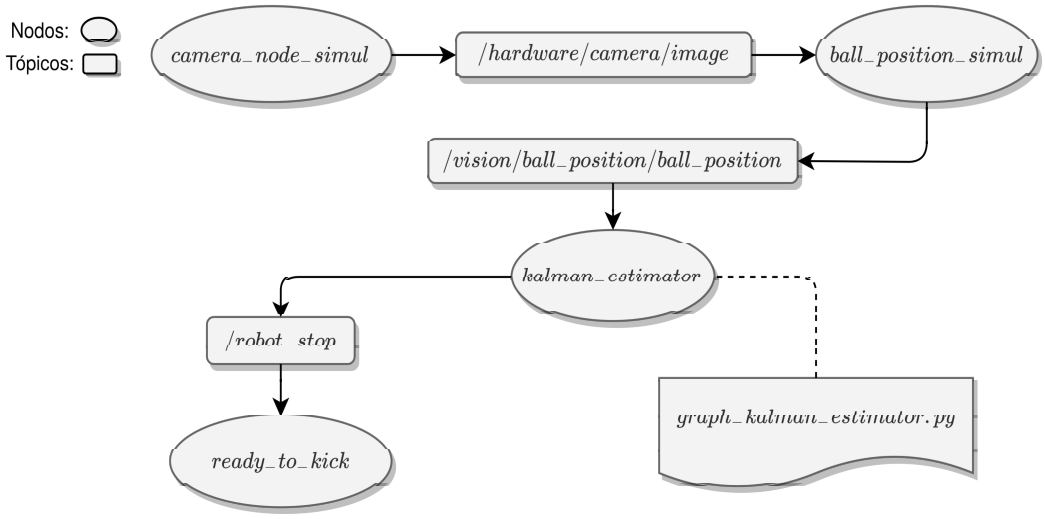


Figura 5.5: Diagrama simplificado de los principales nodos empleados

### Nodo para la secuencia de pateo

El programa que ejecuta el nodo *ready_2_kick* utiliza la clase Humanoid para hacer movimientos con posiciones predefinidas, en este caso pausandolas para mantenerse parado sobre un pie esperando a que el nodo *kalman_estimator* le dé la señal para que patee el balón.

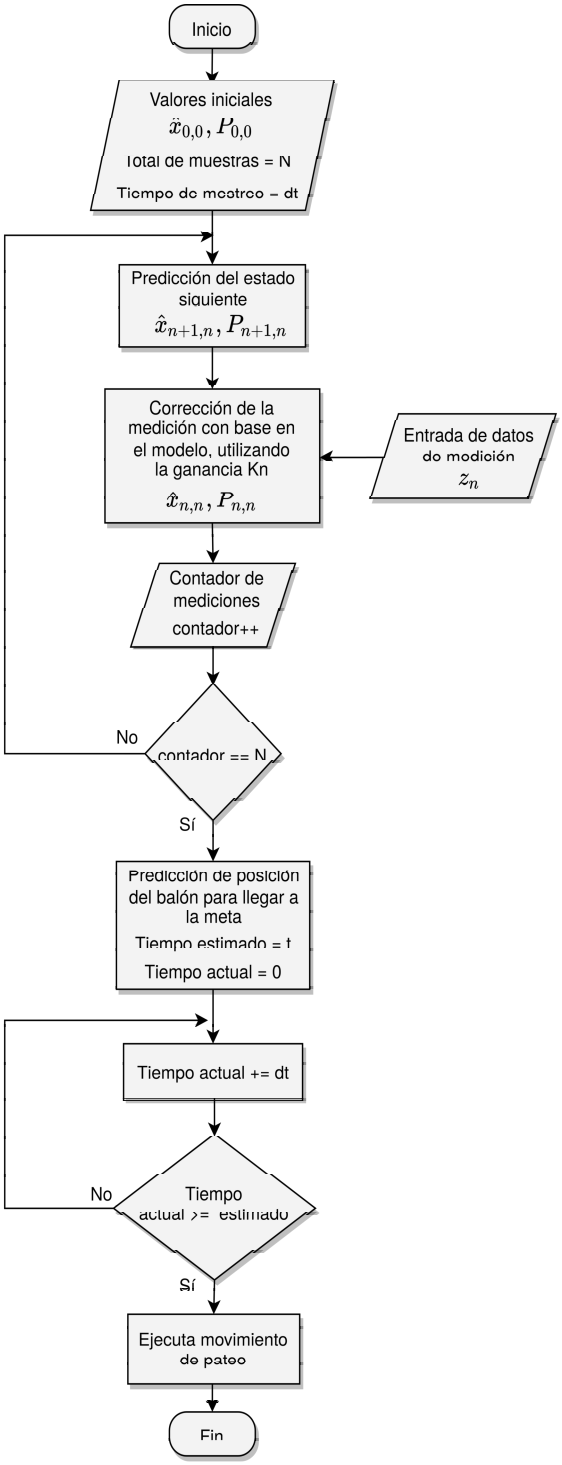


Figura 5.6: Diagrama de flujo del nodo estimador de posición y velocidad

# Capítulo 6

# Resultados

## 6.1. Descripción del experimento

Debido a la naturaleza experimental de esta tesis, las validaciones de la hipótesis deben obtenerse de carácter cuantitativo con una prueba empírica. Una forma de hacerlo es implementando el análisis de los datos de las mediciones de posición dentro del sistema simulado, en este caso el robot humanoide. De esta misma forma, se puede hacer dicha implementación desarrollando la prueba del *RoboCup* llamada *kick from a moving ball*, la cual consiste en patear un balón en movimiento.

En primera instancia, se hizo un desarrollo del sistema de visión, para el robot en físico se tuvo que hacer un nodo que se conecte con el sistema a bajo nivel del hardware de la cámara y publique la imagen 30 veces por segundo en un tópico, para el robot simulado se requirió instalar un *plug-in* dentro del archivo de configuración del robot (.xml) para que sea capaz de simular una cámara y visualice lo que se encuentra dentro del ambiente de Gazebo, véase la Figura 5.3.

Teniendo ya el sistema de visión con su respectivo procesamiento de imágenes para el posicionamiento del balón, se procedió a alimentar al estimador de Kalman con los valores calculados de posición haciendo una doble finalidad, la primera de filtrar el ruido inherente a la medición y la segunda hacer una estimación a *posteriori* de la posición del balón con un limitado número de muestras, dado que si se toman muchas mediciones, podría no dar tiempo para que el robot alcance a patear el balón.

Una vez dejando que el estimador de Kalman logre estimar la posición

más probable del balón en un tiempo $t$ el programa espera a que se cumpla cierto umbral de tiempo para comenzar a realizar el movimiento de pateo. Teniendo como objetivo que el pie llegue a una posición determinada por donde va a pasar el balón en movimiento, lo patee y el regrese a su posición inicial.

## 6.2. Parámetros del experimento

La cámara simulada dentro de Gazebo tendrá un rango de visión horizontal estándar que viene por defecto en el *plug-in* de 1.396 *rads* u $80^o$ con una resolución de imagen de 640x480 que es más que suficiente para este experimento. Se consideró suficiente que cada imagen se publique a una frecuencia de 30Hz (un rango cercano al sistema óptico del ser humano), de esta manera le da tiempo al estimador de tomar las muestras necesarias para una oportuna predicción.

Para que se tenga el mayor rango posible de velocidades del balón en este experimento se requiere hacer que el movimiento de pateo se concluya en el menor tiempo posible, sin embargo, si se le deja poco tiempo para patear, la estabilidad del robot puede verse afectada e incluso peligrosa (más evidentemente para el robot en físico) por lo que se decidió que un tiempo óptimo para patear el balón en un pequeño rango de movimiento es de 0.1 segundos. Con este tiempo se pueden hacer pruebas con una ventana de velocidades del balón de entre 2.0 a 2.8 $m/s$ ya que si es menor el valor, el balón se detiene antes de llegar a la meta esperada y si es mayor, el estimador no tendrá tiempo de hacer el proceso de estimación antes de que el balón sobrepase la meta.

Al principio del experimento el balón tiene una distancia aproximada del robot de 1.2 metros en $y$, se decidió que estuviera a esa distancia debido a que en esa posición está fuera del rango de visión del robot que está esperando recibir la instrucción de patear. Es importante que se mantenga fuera del rango de visión para que cuando detecte una posición del balón, comience a tomar un determinado número de muestras para la predicción de posición. La determinación del número de muestras se hizo de manera empírica con base en los resultados de predicción y corrección. En las Figuras 6.1, 6.2 y 6.3 se puede observar que a distintas velocidades (2.5, 3.2 y 3.5 m/s respectivamente) los datos de estimación de Kalman empiezan a converger alrededor de las ocho primeras muestras, por lo que se consideró que son suficientes para que se

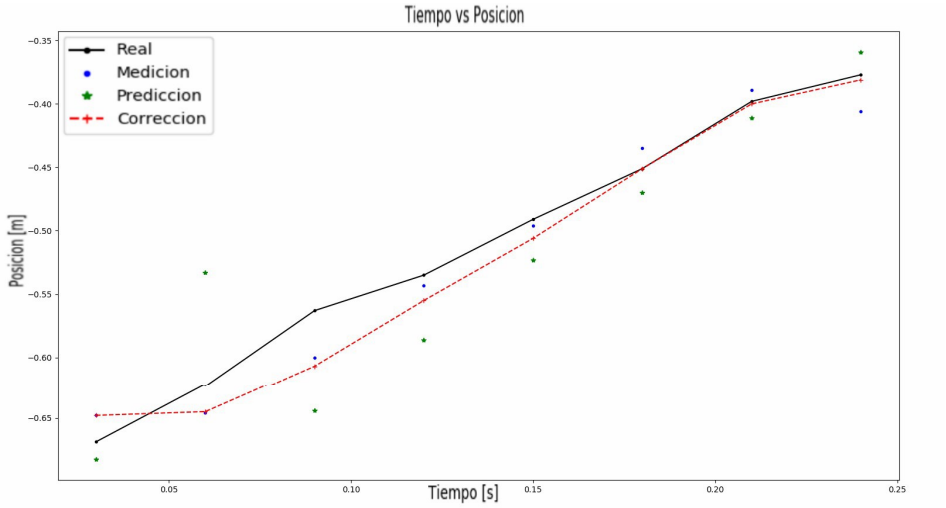


Figura 6.1: Gráfica de tiempo contra posición del balón con una velocidad inicial de 2.5 m/s.

pueda realizar una predicción lo suficientemente rápida antes de que el balón pase en frente del robot (véase la Figura 6.1).

## 6.3. Pruebas de estimación de posición y velocidad

En la Sección 4.3 se obtuvo de manera experimental el coeficiente de fricción dinámica del balón sobre una superficie de fieltro la cual se planea usar para hacer pruebas con el robot real, sin embargo en las pruebas simuladas se fueron observando los diversos comportamientos con coeficientes de ficción menores (desde 0.1 hasta 0.3636) para tener un movimiento más parecido al rectilíneo uniforme. No obstante las pruebas con el coeficiente de fricción obtenido experimentalmente no representaron problemas a la hora de hacer la estimación y se concluyeron los mismos resultados de la prueba de pateo.

Se hicieron diversas pruebas en la simulación, variando diversos parámetros para verificar la robustez del sistema de visión y estimación, entre los principales fueron las velocidades iniciales del balón, los ángulos de posición de la cabeza del robot y las distancias de recorrido del balón. A continuación se presentan ejemplos de las pruebas y cómo se desempeñan los algoritmos

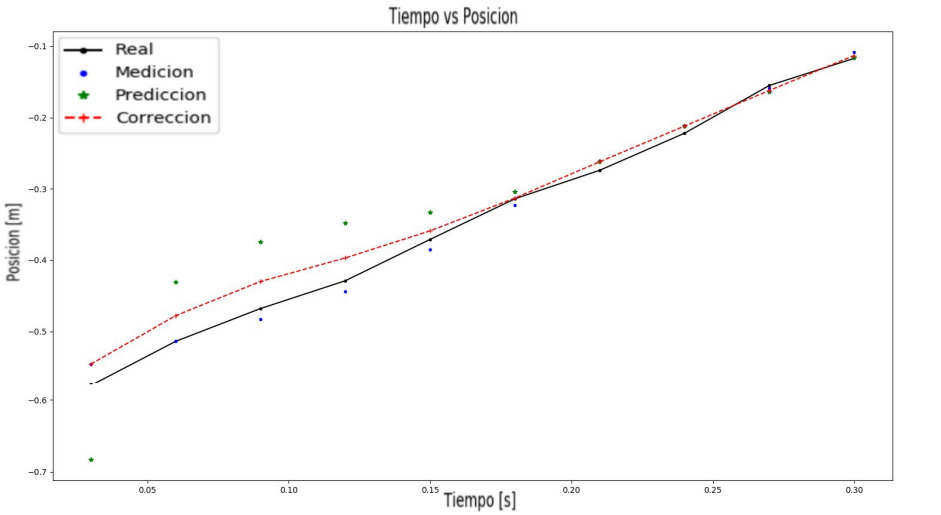


Figura 6.2: Gráfica de tiempo contra posición del balón con una velocidad inicial de 3.2 m/s.

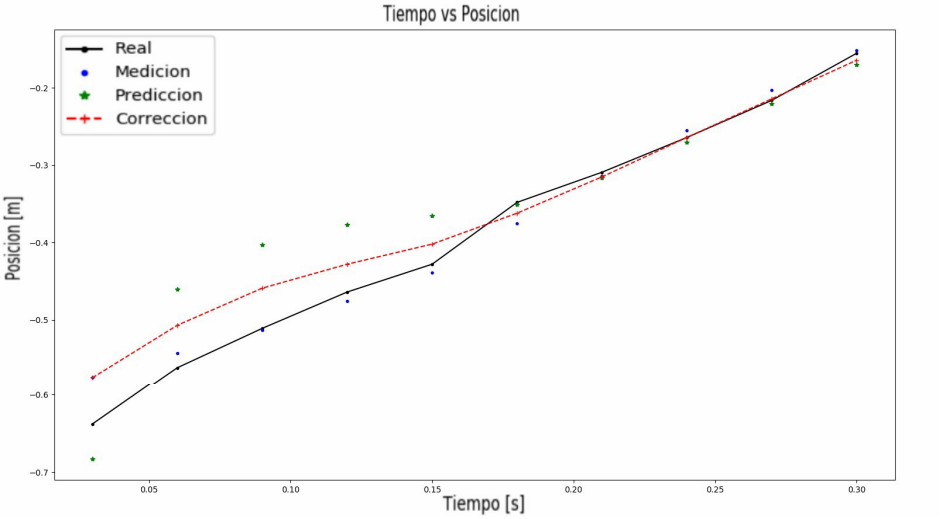


Figura 6.3: Gráfica de tiempo contra posición del balón con una velocidad inicial de 3.5 m/s.

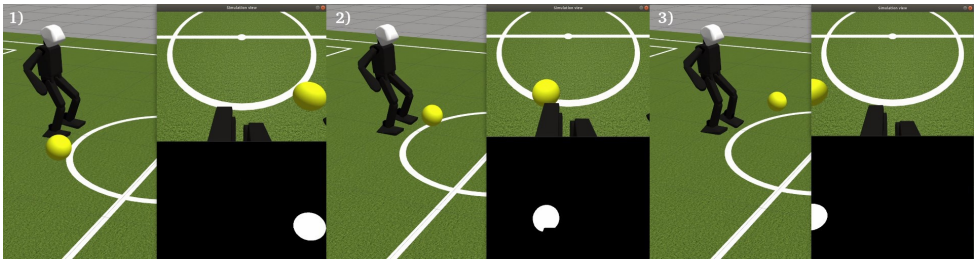


Figura 6.4: Prueba con una velocidad inicial del balón de 3.0 m/s. En este caso el balón tiene una energía cinética más elevada de lo que el sistema puede admitir para una oportuna estimación de posición. Puede verse que el balón viaja de derecha a izquierda sin que el robot intente patear.

si se varían las velocidades del balón.

**Prueba 1: Movimiento horizontal con una velocidad inicial de 3.0 m/s**

En la Figura 6.4 se observa la secuencia de una prueba del estimador del balón con un movimiento horizontal de derecha a izquierda en donde la velocidad inicial está fuera la ventana de velocidades. Esta secuencia consta de tres elementos los cuales se puede observar cómo se desempeña el robot en una prueba final, en cada uno está el estado del robot (latente para patear) y las entradas de imagen simulada(la primaria y la segmentada). Debido a que el balón tiene bastante ímpetu, al sistema de visión no le da el tiempo suficiente para hacer las mediciones mínimas para estimar una posición antes de que el balón llegue a cierto umbral. Otra situación en la que no le daría tiempo para patear es si se toman demasiadas muestras antes de realizar el movimiento de pateo.

**Prueba 2: Movimiento horizontal con una velocidad inicial de 1.8 m/s**

De manera similar a la Prueba 1, en esta prueba se hizo mover al balón con la misma trayectoria pero con una velocidad menor a la recomendada para realizar la prueba (alrededor de 2.0 m/s). En esta secuencia (Figura 6.5) se visualiza que aunque el estimador tenga el tiempo suficiente para hacer una estimación, el balón no tiene suficiente impulso para llegar hasta

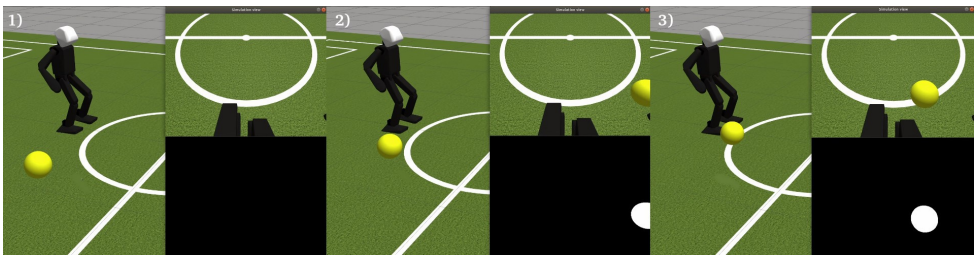


Figura 6.5: Prueba con una velocidad inicial del balón de 1.8 m/s. Con una velocidad inicial relativamente baja como en este caso, el estimador calcula de manera inmediata la extrapolación de la trayectoria, en la cual es posible saber si la velocidad del balón será de cero antes de llegar al punto donde el robot pueda patearlo.

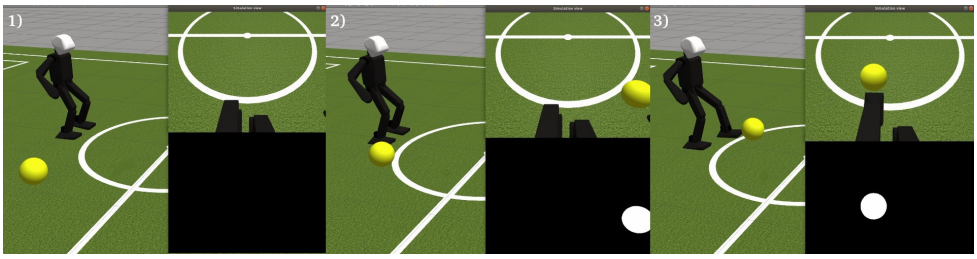


Figura 6.6: Prueba con una velocidad inicial del balón de 1.8 m/s. En esta prueba final, se ve cómo al ir en una velocidad óptima al estimador le da tiempo de calcular el tiempo de llegada del balón al punto de pateo y posteriormente patearlo con un movimiento muy similar al que hacen los seres humanos.

el pie que se supone va a patear cuando pase justo enfrente. En estos casos, el programa solamente indica que no dará el tiempo suficiente y no intenta patear.

### Prueba 3: Movimiento horizontal con una velocidad inicial de 2.2 m/s

Finalmente cuando se realizó una prueba con una velocidad que está dentro del rango aceptado se observa (Figura 6.6) que el balón es pateado por el robot, indicando que el estimador cumplió exitosamente su proceso.

La Figura 6.7 muestra la secuencia de una prueba similar, con la misma

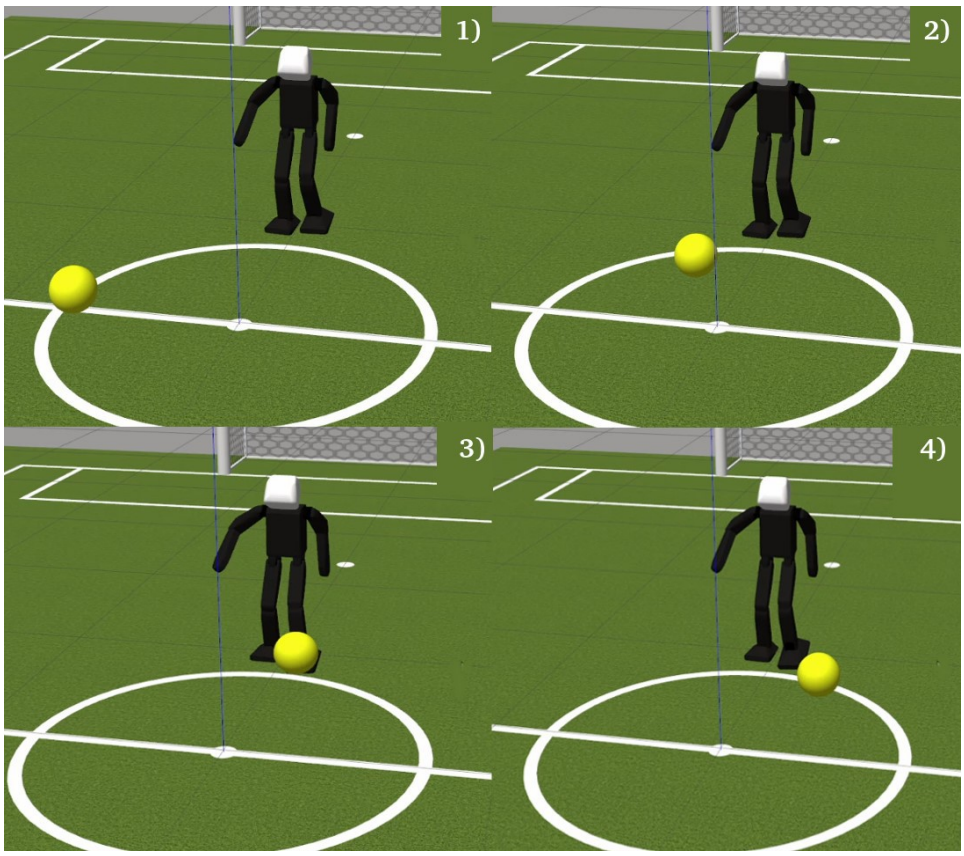


Figura 6.7: Secuencia que ilustra el pateo del balón con una distinta dirección en movimiento, mostrando la robustez del sistema hacia distintas trayectorias.

velocidad inicial pero con una trayectoria diferente con movimiento en los ejes $x$ y $y$ (utilizando el sistema de referencia que se vio en el Capítulo 4). En este caso el balón pasa hacia un punto cercano al pie que patea.

# Capítulo 7

# Discusión

## 7.1. Conclusiones

Con respecto a la parte de resultados, se concluye que los objetivos planteados en esta tesis se cumplieron satisfactoriamente en cuanto al desarrollo del sistema de visión del humanoide, el algoritmo de pateo con un control a lazo abierto basado en posiciones predefinidas(tanto el real como el simulado) y en la aplicación del Filtro de Kalman Extendido para la filtración del ruido inherente a las mediciones (en el robot simulado).

Al momento de unir los distintos programas en el sistema simulado se presentaron algunos retos a cumplir en cuanto a la optimización de los procesos y su debida interconexión, para lo cual las herramientas de la plataforma ROS fueron de vital importancia y ayudaron para integrar nodos escritos en diferentes lenguajes de programación.

Se logró crear la clase *Humanoid* cuya funcionalidad es hacer que el robot se mueva a distintas posiciones, utilizando archivos de configuración con formato *.yaml*, que contiene los ángulos de cada actuador para cada movimiento. Para esta tesis solamente se utilizó una serie de movimientos perfilizados para el pateo. No obstante es fácil agregar más archivos *.yaml* para crear una infinidad de posiciones haciendo que el humanoide tenga un mayor grado de biomímesis.

En cuanto al objetivo general, se logró cumplir que el robot simulado pateara el balón con una previa predicción de posición. Sin embargo, quedará en espera el hacer pruebas con el robot real, con la expectativa de que se requieran sólo cambios mínimos concernientes al hardware y al ruido de

medición con variables reales.

Una de las ventajas de hacer las pruebas del robot simulado, que al mismo tiempo es una desventaja para el robot real, es el del tiempo de latencia que hay entre la comunicación con la computadora con el hardware del humanoide, es importante de mencionar debido a que representaría un retardo de tiempo desde que se manda un mensaje hasta que el hardware lo reciba.

## 7.2. Trabajo Futuro

Como trabajo futuro, se deja como prioritario migrar todos los programas y parámetros al sistema embebido, para así dejarle completamente listo para las competencias internacionales y para el seguimiento del desarrollo de software que otros alumnos de la Facultad de Ingeniería estén dispuestos a hacer.

En la Sección 4.2 el modelo matemático se obtuvo como una deducción basada en la segunda ley de Newton y la posición del balón está descrita de manera parabólica respecto al tiempo y aunque, es intuitivamente viable, sería útil implementar un modelo matemático distinto con una descripción hiperbólica, igual que la ecuación de la fuerza de fricción viscosa.

Tal y como se vio en la parte de resultados el rango de velocidades del balón tuvo un umbral de 2.0 a 2.8 m/s aproximadamente, es decir, que es un rango estrecho, y cuando se implemente en el mundo real puede disminuir dependiendo las variaciones del coeficiente de fricción. Una manera en que se mejoraría lo anterior, es adaptando una cámara con mayor rango de visión horizontalmente con las ya conocidas cámaras de *ojo de pescado*, así se podrían tomar las ocho muestras de información para que el estimador pueda hacer una estimación (valga la redundancia) con mayor tiempo y distancia. Para hacer más robusto el sistema de segmentación de objetos (ya que con la actual puede verse afectada por las tonalidades de luz que hay en el ambiente), una forma conveniente para probar, es con la ayuda de una cámara estereoscópica a fin de obtener la profundidad con una nube de puntos.

Para mejorar algoritmo de pateo en cuanto a su estabilidad en cada movimiento, es conveniente agregar un sistema de control de velocidad (ya que sólo cuenta control de posición en cada articulación). A parte de esto, que se hiciera un control de posiciones con realimentación, ya que es fácil que el robot caiga con cada prueba. Con un sistema a lazo cerrado se obtendría la estabilidad necesaria e incluso se podrían realizar movimientos más rápidos,

ampliando sin duda el rango de velocidades a los que el robot pueda patear un balón en movimiento.

En este trabajo se desarrolló un sistema de visión de predicción utilizando un método basado en la esperanza de una serie de datos y un previo modelo matemático. Sin embargo, esta no es la única forma de hacer este tipo de funciones, para futuras propuestas de proyectos, se podría implementar un sistema de estimación basado el redes neuronales (más parecido al sistema del ser humano) o algoritmos genéticos, los cuales no tendrían las mismas limitantes que el Filtro de Kalman Extendido, como por ejemplo que se tenga que conocer el coeficiente de fricción dinámica, que se requiera una superficie completamente plana u horizontal y que el balón no tenga rebotes o elevaciones de ningún tipo en cuanto a su trayectoria.

# Referencias


Agoston, M. K., y Agoston, M. K. (2005). *Computer graphics and geometric modeling* (Vol. 1). Springer.

Bradski, G., y Kaehler, A. (2008). *Learning opencv: Computer vision with the opencv library.* " O'Reilly Media, Inc.".

Culjak, I., Abram, D., Pribanic, T., Dzapo, H., y Cifrek, M. (2012). A brief introduction to opencv. En *2012 proceedings of the 35th international convention mipro* (pp. 1725–1730).

Farazi, H., Allgeuer, P., Ficht, G., y Behnke, S. (2016). Nimbro teensize team description 2016. *RoboCup Humanoid League Team Descriptions.*

Forsyth, D., y Ponce, J. (2011). *Computer vision: A modern approach.* Prentice hall.

Gerndt, R., Seifert, D., Baltes, J. H., Sadeghnejad, S., y Behnke, S. (2015). Humanoid robots in soccer: Robots versus humans in robocup 2050. *IEEE Robotics & Automation Magazine*, *22*(3), 147–154.

Gonzalez, R. C., Woods, R. E., y Eddins, S. L. (2004). *Digital image processing using matlab.* Pearson Education India.

Haralick, R. M., Sternberg, S. R., y Zhuang, X. (1987, July). Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-9*(4), 532-550. doi: 10.1109/TPAMI.1987.4767941

Heijmans, H. J. (1999). Connected morphological operators for binary images. *Computer Vision and Image Understanding*, *73*(1), 99–120.

Herculano, I. S. L. C. L. A. M. M. M. A., Daniela Vacarini, y Samuel Pinto, e. a. (2017). Itandroids humanoidteam description paper for robocup 2017. , 5.

Kitano, H., y Asada, M. (1998). The robocup humanoid challenge as the millennium challenge for advanced robotics. *Advanced Robotics*, *13*(8), 723–736.

Pineda Cortés, L. (2017). *La computación en méxico por especialidades acádemicas.* México: Acádemia Mexicana de Computación AC.

Prince, S. J. (2012). *Computer vision: models, learning, and inference.* Cambridge University Press.

Pyo, Y., Cho, H., Jung, L., y Lim, D. (2015). Ros robot programming. *Seoul, ROBOTIS Co.*

Razi, M. R. A., Arifin, M., Muhtadin, D. S. W., Setiawan, E. D. S., Fahmy, M. N., Asshakina, S. Q., y Dzaka, M. A. (s.f.). Ichiro team-team description paper humanoid teensize league of robocup 2018.

Robocup Humanoid League Rulebook, R. F. (2019). Robocup soccer humanoid league, laws of the game. *RoboCup competition*, *73*(1), 14–22.

Sakagami, Y., Watanabe, R., Aoyama, C., Matsunaga, S., Higaki, N., y Fujimura, K. (2002). The intelligent asimo: System overview and integration. En *Ieee/rsj international conference on intelligent robots and systems* (Vol. 3, pp. 2478–2483).

Schwarz, M., Pastrana, J., Allgeuer, P., Schreiber, M., Schueller, S., Missura, M., y Behnke, S. (2013). Humanoid teensize open platform nimbro-op. En *Robot soccer world cup* (pp. 568–575).

Siciliano, B., y Khatib, O. (2016). *Springer handbook of robotics.* Springer.

Suryanto, A., y Dewi, Y. M. (s.f.). T-flow team.

Szeliski, R. (2010). *Computer vision: algorithms and applications.* Springer Science & Business Media.

Welch, G., Bishop, G., y cols. (1995). *An introduction to the kalman filter.* Citeseer.

# Apéndices

# Apéndice A

# Código del estimador de Kalman

```
#!/usr/bin/env python
import os
import json
import rospy
import numpy
import rospkg

from random import gauss
from std_msgs.msg import Bool
from std_msgs.msg import Float32MultiArray


#VARIABLES FISICAS
g  = 9.81       #ACELERACION DE LA GRAVEDAD
dt = 0.033333   #TIEMPO DE MUESTREO
mu_d = 0.15     #COEFICIENTE DE FRICCION DINAMICA

#CONTADOR DE MUESTRAS
measurements = 0

#NUMERO MINIMO DE MUESTRAS PARA EMPEZAR LA ESTIMACION
data_samples = 15

#MUESTRAS TOTALES
x_positions = []
y_positions = []

```

```
28 #BANDERA PARA DEJAR DE REGISTRAR DATOS
29 log_out = False
30
31 #TIEMPO ESTIMADO PARA LLEGAR A LA META
32 time_to_kick = 0
33
34 #PUBLICADOR PARA PATEAR EL BALON
35 kick = rospy.Publisher('/robot_stop', Bool, queue_size
       =1000)
36
37 #POSICION INICIAL
38 #print "-------------- ESTADO INICIAL ----------------"
39 #print "Numero de muestras por registar:", data_samples
40
41 Xn = numpy.array([[-1.2],
42             [-1.2],
43             [ 3.5],
44         [ 3.5]])
45 print "X0"
46 print  Xn
47
48 #MATRIZ DE COVARIANZA DEL PROCESO
49 Pn  = 5 * numpy.identity(4)
50
51 print "P0"
52 print  Pn
53
54 #MATRIZ DE PREDICCION
55 F1 = [ 1, 0, dt,  0]
56 F2 = [ 0, 1,  0, dt]
57 F3 = [ 0, 0,  1,  0]
58 F4 = [ 0, 0,  0,  1]
59
60 F = numpy.array([F1, F2, F3, F4])
61
62
63 #MATRIZ DE COVARIANZA DEL RUIDO DE LA ESTIMACION
64 Q  = 0.000001 * numpy.identity(4)
65 #Q  = 0 * numpy.identity(4)
66
67 print "Q"
68 print  Q
69
70 #MATRIZ DE OBSERVACION
71 H1 = [ 1, 0, 0, 0]
```

```
72 H2 = [ 0, 1, 0, 0]
73
74
75 H = numpy.array([H1, H2])
76
77 print "H"
78 print  H
79
80 #MATRIZ DE COVARIANZA DE RUIDO EN LA MEDICION
81 R = 0.02 * numpy.identity(2)
82
83 print "Rn"
84 print  R
85
86 #FUERZA DE FRICCION
87 Fr = numpy.array([[      0     ],
88        [      0     ],
89        [-dt*mu_d*g],
90        [-dt*mu_d*g]])
91
92 #print "Fr"
93 #print  Fr
94
95 #FUNCION DE PREDICCION ESTADO SIGUIENTE
96 def prediction_state():
97     global Xn_, Pn_, Xn1
98     #print ""
99     #print "--------------- ESTADO DE PREDICCION
    ----------------"
100
101     #PREDICCION DEL ESTADO SIGUIENTE
102     Xn1 = numpy.dot(F, Xn) + Fr
103     #print "Xn1"
104     #print  Xn1
105
106     #PREDICCION DE LA COVARIANZA
107     Pn1 = numpy.dot(numpy.dot(F, Pn), F.transpose()) + Q
108     #Pn1 = numpy.diag(numpy.diag(Pn1)) + Q
109     #print "Pn1"
110     #print  Pn1
111
112     # n -> n+1
113     Xn_ = Xn1
114     Pn_ = Pn1
115
```

```
116 #FUNCION PARA GUARDAR EL ESTADO DEL BALON
117 def catch_x_position(Rx, Z1, x_, xn):
118     position_list = list()
119     position_list.append(round(Rx,3)) #VALOR EXACTO
120     position_list.append(round(Z1,3)) #VALOR MEDIDO
121     position_list.append(round(x_,3)) #PREDICCION
122     position_list.append(round(xn,3)) #VALOR CORREGIDO
123 
124     x_positions.append(position_list)
125 
126 def catch_y_position(Ry, Z2, y_, yn):
127     position_list = list()
128     position_list.append(round(Ry,3)) #VALOR EXACTO
129     position_list.append(round(Z2,3)) #VALOR MEDIDO
130     position_list.append(round(y_,3)) #PREDICCION
131     position_list.append(round(yn,3)) #VALOR CORREGIDO
132 
133     y_positions.append(position_list)
134 
135 #FUNCION DE PATEO
136 def wait_for_kick():
137     rate = rospy.Rate(30)
138 
139     #CONTADOR DE TIEMPO
140     current_time = 0
141     while current_time < time_to_kick:
142       current_time += dt
143   rate.sleep()
144 
145     #print "Pateando balon..."
146     kick.publish(Bool(True))
147 
148 
149 #FUNCION QUE CALCULA EL TIEMPO RESTANTE PARA EL PATEO
150 def estimator():
151     global Xn, Xn1, time_to_kick
152 
153     #print "\nMuestreo completado."
154 
155     if float(Xn[1]) < 0:
156   #print "Estimando posiciones..."
157   while float(Xn[1]) < 0:
158             prediction_state()
159       Xn = Xn1
160       time_to_kick += dt
```

```
161
162         if Xn[3] < 0:
163             #print "\nEl balon no tiene suficiente impulso"
164     break
165
166                 if Xn[1] >= 0:
167     if time_to_kick < 0.1:
168               else:
169     wait_for_kick()
170
171     else: #print "\nEl balon esta fuera de rango."
172
173 #FUNCION DE RESPUESTA AL RECIBIR LOS DATOS DE ENTRADA
174 def measurement_input(data):
175     global Xn, Pn, measurements, log_out
176     measurements += 1
177     #print ""
178     #print "-------------- VALORES DE ENTRADA
        ----------------"
179     #print "Medicion numero:", measurements
180
181     #VALOR EXACTO DE POSICION
182     Rx = data.data[0]
183     Ry = data.data[1]
184
185     #VECTOR DE MEDICION
186     Z1 = data.data[2]
187     Z2 = data.data[3]
188
189     Z = numpy.array([[Z1],
190           [Z2]])
191     #print "Z"
192     #print  Z
193
194      if measurements - 1 < data_samples:
195   #print ""
196   #print "-------------- ESTADO DE CORRECCION
       ----------------"
197
198
199   #CALCULO DE LA GANANCIA DE KALMAN
200   K1 = numpy.dot(Pn_, H.transpose())
201   K2 = numpy.dot(numpy.dot(H,Pn_), H.transpose()) + R
202
203
```

```
204     K = numpy.dot(K1, numpy.linalg.inv(K2))
205 
206     #print "K"
207     #print  K
208 
209     #CORRECCION DE LA POSICION
210     Xn = Xn_ + numpy.dot(K, Z - numpy.dot(H, Xn_))
211 
212     #print "Xn"
213     #print  Xn
214 
215 
216     #CORRECCION DE LA COVARIANZA DEL PROCESO
217     Pn = numpy.dot(numpy.identity(4)- numpy.dot(K, H), Pn_)
218 
219     #print "Pn"
220     #print  Pn
221 
222     catch_x_position(Rx, Z1, float(Xn_[0]), float(Xn[1]))
223     catch_y_position(Ry, Z2, float(Xn_[1]), float(Xn[1]))
224     prediction_state()
225 
226       if measurements == data_samples:
227           log_out = True
228 
229 
230 #FUNCION PRINCIPAL
231 def kalman_estimator():
232       #INICIALIZA EL NODO
233       rospy.init_node('kalman_estimator', anonymous=True)
234       #print ""
235       #print "Inicializando nodo kalman_estimator por Luis
        Nava"
236 
237       #TOPICO PARA OBTENER LOS DATOS DE POSICION
238       rospy.Subscriber("/vision/ball_position/ball_position
        ",
239                        Float32MultiArray,
        measurement_input)
240 
241       #METODOS PARA GUARDAR INFORMACION EN UN ARCHIVO DE
        TEXTO
242       rospack = rospkg.RosPack()
243       rospack.list()
244 
```

```
245     abs_path = rospack.get_path('ball_position')
246 
247     if os.path.exists(abs_path + '/scripts/kalman_data_x.
      txt'):
248       os.remove(abs_path + '/scripts/kalman_data_x.txt')
249 
250     if os.path.exists(abs_path + '/scripts/kalman_data_y.
      txt'):
251   os.remove(abs_path + '/scripts/kalman_data_y.txt')
252 
253     #print "Esperando datos de entrada..."
254 
255     #BUCLE PARA LA RECEPCION DE MENSAJES
256     while not rospy.core.is_shutdown():
257   if log_out: break
258 
259     #FUNCION PARA ESTIMAR EL TIEMPO PARA PATEAR
260     estimator()
261 
262     #print "\nFinalizando nodo..."
263 
264     #SALVANDO POSICIONES EN UN ARCHIVO DE TEXTO
265     with open(abs_path + '/scripts/kalman_data_x.txt', 'a
      ')
266                                                 as
      filehandle:
267       for i in range(data_samples):
268             json.dump(x_positions[i], filehandle)
269       filehandle.write("\n")
270 
271     with open(abs_path + '/scripts/kalman_data_y.txt', 'a
      ')
272                                                 as
      filehandle:
273   for i in range(data_samples):
274             json.dump(y_positions[i], filehandle)
275       filehandle.write("\n")
276 
277 if __name__ == '__main__':
278     prediction_state()
279     kalman_estimator()
```