UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA

INGENIERÍA ELECTRÓNICA

Trabajo de titulación previo a la obtención del título de

INGENIERO ELECTRÓNICO

TEMA

IMPLEMENTACIÓN DE UN SISTEMA DE APRENDIZAJE PARA UN ROBOT HUMANOIDE BASADO EN LOS MOVIMIENTOS ARTICULARES DE UNA PERSONA USANDO CINEMÁTICA INVERSA

AUTOR

JOSÉ DAVID LOJA ROMERO

TUTOR

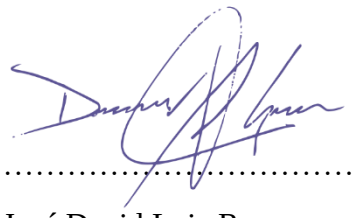
CARLOS GERMÁN PILLAJO ANGOS

Quito, noviembre del 2018

CESIÓN DE DERECHOS DE AUTOR

Yo José David Loja Romero con documento de identificación N° 030259074-0, manifiesto mi voluntad y cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del trabajo de grado/titulación intitulado "IMPLEMENTACIÓN DE UN SISTEMA DE APRENDIZAJE PARA UN ROBOT HUMANOIDE BASADO EN LOS MOVIMIENTOS ARTICULARES DE UNA PERSONA USANDO CINEMÁTICA INVERSA", mismo que ha sido desarrollado para optar por el título de: Ingeniero Electrónico, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en condición de autor me reservo los derechos morales de la obra antes citada. En concordancia, suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Politécnica Salesiana.



…….

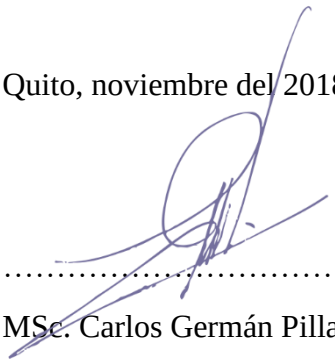
José David Loja Romero
C.I. 0302590740

Quito, noviembre del 2018.

DECLARATORIA DE COAUTORÍA DEL DOCENTE TUTOR

Yo declaro que bajo mi dirección y asesoría fue desarrollado el trabajo de titulación, "IMPLEMENTACIÓN DE UN SISTEMA DE APRENDIZAJE PARA UN ROBOT HUMANOIDE BASADO EN LOS MOVIMIENTOS ARTICULARES DE UNA PERSONA USANDO CINEMÁTICA INVERSA", realizado por José David Loja Romero, obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana para ser considerado como trabajo final de titulación.

Quito, noviembre del 2018



…….

MSc. Carlos Germán Pillajo Angos

C.I: 170925511-9

DEDICATORIA

Dedico este proyecto de titulación a mis padres Victoria y Romeo por haber sido los motores en vida y permitirme seguir adelante con mis sueños. Su perseverancia y sus consejos han sido cruciales en los momentos más difíciles de toda mi carrera. A mis hermanos Abraham e Israel por siempre estar a mi lado y brindarme todo su apoyo incondicional en cualquier actividad que he realizado. A Morelva que me acompañado y apoyado en cada momento.

AGRADECIMIENTO

Agradezco a la Universidad Politécnica Salesiana por el conocimiento y las oportunidades que me ha brindado para crecer como profesional y como persona. Agradezco al MSc. Carlos Pillajo por su tiempo y profesionalismo que me ha brindado en la realización de este proyecto de titulación.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente proyecto técnico está encaminado en la implementación de un sistema de aprendizaje para un robot humanoide el cual le permite replicar los movimientos de las extremidades inferiores de un ser humano. Dicho sistema está basado en sensores que permiten obtener las coordenadas de movimiento en un plano para posteriormente calcular los ángulos de los servomotores del robot en base a cinemática inversa. Los datos recogidos del ser humano son enviados hacia el robot de manera inalámbrica mediante una red local. Todo el sistema de base de datos y algoritmos se encuentra implementado mediante el programa Matlab en el cual se usa una interfaz gráfica para el muestro de los movimientos, presentación gráfica de los datos y ejecución del movimiento. Todo esto se encuentra montado sobre la placa de desarrollo LattePanda que controla en su totalidad al robot humanoide.

ABSTRACT

The present technical project is aimed at the implementation of a learning system for a humanoid robot which allows it to replicate the movements of the lower extremities of a human being. This system is based on sensors that allow to obtain the coordinates of movement in a plane to later calculate the angles of the servomotors of the robot based on inverse kinematics. The data collected from the human being is sent to the robot wirelessly through a local network. The entire database and algorithm system is implemented through the Matlab program in which a graphical interface is used for the sampling of the movements, graphic presentation of the data and execution of the movement. All this is mounted on the LattePanda development board that controls the humanoid robot in its entirety.

INTRODUCCIÓN

El movimiento de un robot humanoide por lo general está basado en programación secuencial y muy pocas veces utilizando un análisis basado en cinemática inversa por lo que sus acciones son limitadas. Se plantea implementar un sistema basado en la imitación de movimientos de un ser humano en tiempo real, para dotar a un robot humanoide de movimientos que no estén sujetos a una previa programación secuencial. Así el robot podrá aprender y guardar dichas acciones, basado en el análisis del menor error en ejecución obtenidos por el algoritmo de aprendizaje de una serie de muestras de coordenadas, permitiéndole caminar, tomar objetos, entre otras actividades que el usuario realice. Todas estas acciones las realizará, tomando en consideración las limitaciones mecánicas del robot, de una manera más natural y sin necesidad de calibración por código. Todo el sistema se va a implementar en una placa de desarrollo embebido.

Como se explicó, este documento se centra en la investigación de los movimientos de un ser humano como el caminar, para identificar factores que ayuden en la comprensión de los mismos y poder concluir que movimientos van a ser posibles, todo esto dependiendo de la mecánica del robot. De esta forma, implementar un sistema embebido que va a comandar todos los sistemas en el robot acoplarse la red I2C con la que se va a realizar la lectura de los datos provenientes de los sensores MPU6050.

En el Capítulo 1 se detalla el tipo de cinemática va a ser empleada en la resolución, el desempeño, y los objetivos en el desarrollo de este proyecto. En el Capítulo 2 se encuentra la parte teórica del proyecto que presentan la cinemática de movimiento de un humano, sistemas expertos, base de datos desde Matlab, cinemática directa e inversa, mismos que detallan el funcionamiento y comportamiento, explicando la temática del sistema de aprendizaje para un robot humanoide. En el Capítulo 3 se describe el modelamiento del hardware, así como también sus características. Se realiza el desarrollo del sistema de aprendizaje para un robot humanoide.

Por último, en el Capítulo 4 se realiza un análisis del sistema de aprendizaje con el fin de comprobar su funcionamiento y verificación de datos.

CAPÍTULO 1
ANTECEDENTES

1.1. Planteamiento del problema

Hoy en día el movimiento de un robot humanoide por lo general se realiza con el movimiento de sus articulaciones ya sea por juntura o coordenadas de un actuador final utilizando programación dedicada. Es decir, programación bajo sentencias ya establecidas por el programador, que se guían en el estudio del movimiento que se debe seguir o el que debe cumplir el robot. Esto se debe en gran parte a que los sistemas ya están establecidos, y por consiguiente deben seguir un proceso como en la industria.

El problema surge cuando los sistemas se implementan en un entorno inconstante en el que dichas acciones preprogramadas ya no tienen sentido. La opción de solución es reprogramar dicho actuador o robot para que cumpla con los requerimientos. Este proceso conlleva inversión de recursos como lo es el tiempo.

La opción alternativa es dotar a estos procesos de un sistema de aprendizaje basado en un humano experto, es decir, un control en que basará su ejecución en los conocimientos del humano, el cual ha obtenido estos en base a la experiencia o la capacitación previa.

1.2. Tema

Implementación de un sistema de aprendizaje para un robot humanoide basado en los movimientos articulares de una persona usando cinemática inversa.

1.3. Justificación

El movimiento de un robot humanoide por lo general está basado en programación secuencial y muy pocas veces utilizando un análisis cinemático por lo que sus acciones son limitadas. Con la implementación de un sistema basado en el aprendizaje de movimientos de un ser humano en tiempo real dotamos al robot para que aprenda nuevos movimientos y se adapte al entorno que se encuentre.

En la naturaleza los sistemas de imitación son indispensables para la supervivencia de los seres vivos. De igual manera el ser humano evolucionó imitando y mejorando los comportamientos de sus antepasados, aprendiendo nuevas funciones y actividades.

Por esta razón se requiere implementar la misma metodología a un sistema robótico para que pueda aprender de nosotros.

El sistema se puede extrapolar en prótesis inteligentes que aprendan de los requerimientos del usuario y muten conjuntamente. También se puede probar en exoesqueletos a nivel de salud que se basen en el movimiento de una persona normal y ayude en la rehabilitación de la misma. Otras aplicaciones podrían ser brazos robóticos que graben movimientos sin necesidad de programación, robots inteligentes que puedan asistir en actividades cotidianas como en la cocina aprendiendo de su usuario entre otras aplicaciones.

1.4. Objetivos

1.4.1. Objetivo general

Implementar un sistema de aprendizaje para un robot humanoide basado en los movimientos articulares de una persona usando cinemática inversa basada en el sensor MPU6050 con la ayuda de un sistema embebido.

1.4.2. Objetivos específicos

Realizar el análisis de cinemática inversa para obtener las ecuaciones que rigen a los grados de libertad y así poder trasladar los movimientos del usuario al robot en los ejes X,Y,Z.

Implementar un sistema embebido de control (CES) basado en el MPU6050 como sensor para obtener las coordenadas de los movimientos articulares, de las extremidades inferiores.

Realizar la comunicación inalámbrica entre CES y el robot utilizado para comprobar el funcionamiento de la red.

Programar un algoritmo de aprendizaje en Matlab utilizando sistemas expertos en base a los datos de coordenadas obtenidas mediante el CES para obtener el menor error en la ejecución.

Realizar las pruebas del sistema para evaluar la comunicación y funcionamiento del algoritmo de aprendizaje.

1.5. Alcance

El presente proyecto de titulación está dirigido a la investigación y desarrollo del movimiento de un robot humanoide que por lo general está basado en programación secuencial y muy pocas veces utilizando un análisis cinemático por lo que sus acciones son limitadas. Se implementará un sistema basado en la imitación de movimientos de un ser humano en tiempo real para dotar a un robot humanoide de movimientos que no estén sujetos a una previa programación secuencial siendo así que el robot pueda aprender y guardar dichas acciones permitiéndole caminar y que el usuario realice, tomando en consideración las limitaciones mecánicas del robot, de una manera más natural y sin necesidad de calibración por código. Todo el sistema se va a implementar en una placa de desarrollo embebido. Además, se establecerá el modelo matemático, de cada juntura del Robot Humanoide.

CAPÍTULO 2
MARCO CONCEPTUAL

1. Robot Humanoide

No es difícil imaginar que, en las próximas décadas, los robots humanoides vivirán entre nosotros en el entorno humano y nos ayudarán a vivir mejor. Para convertirse en manos reales de ayuda en lugar de simplemente herramientas o máquinas, los robots también requieren apariencias sociales como las personas. Teniendo esto en cuenta, hemos desarrollado un robot humanoide con movimiento en la parte superior del cuerpo y le permitimos interactuar con las personas para saber qué tipo de comportamiento esperan de un asistente humanoide cotidiano. (Sojib, 2017)

El concepto de "estructura humanoide" se refiere a la perspectiva de un robot para hacerlo social en el entorno humano. Asimo es el robot humanoide más popular, pero no tiene rostro expresivo. Un robot humanoide se asemeja o parece a un ser humano, con la capacidad de caminar en posición vertical, siendo ampliamente utilizado en la robótica. (Jin-Ling Lin, 2017)

Los robots humanoides aún se encuentran en estudio y solo pueden funcionar en entornos limitados en la práctica. Su rango de actividad debe cubrir la gama de humanos para apoyar a las personas y ser serviciales. También podrán rastrear o semejar el movimiento humano ya son humanoides. (Fukushima, Satoh, Doi, & Ohtsuka, 2017)

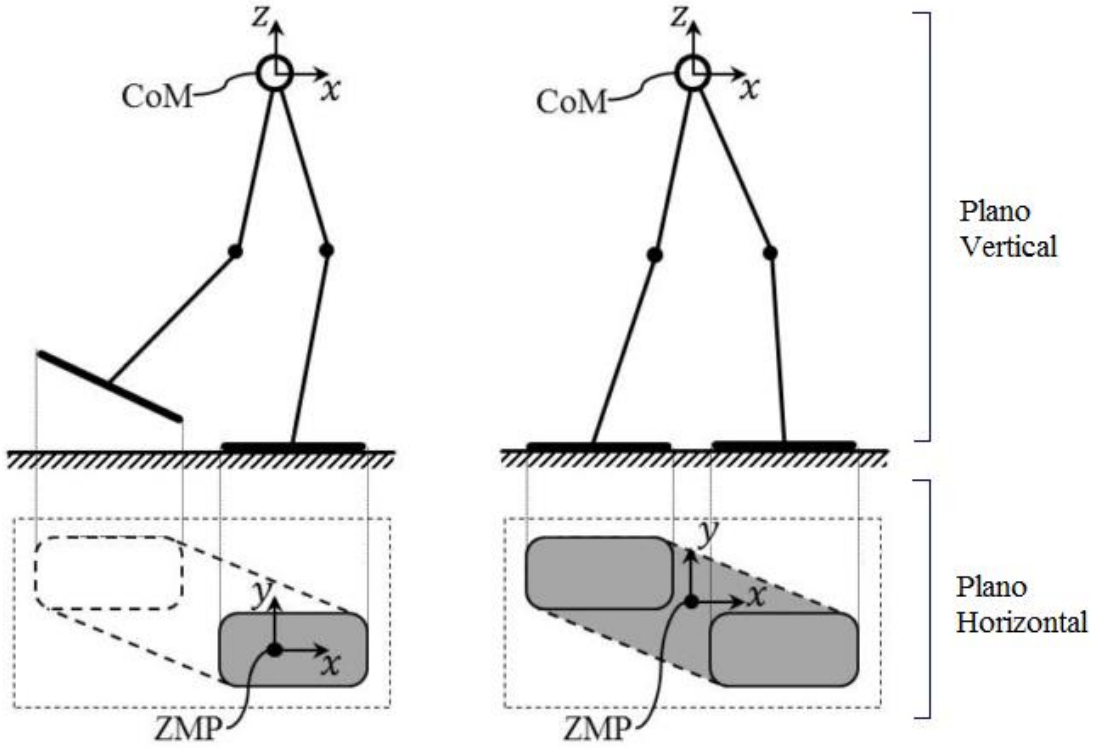
2.1.1. Movimiento de un Robot Humanoide

Existen varios algoritmos para realizar el movimiento de un robot humanoide que es más similar al humano. Por lo general, el caminar del robot humanoide se realiza mediante el uso de la generación de patrones de caminar en función de la cinemática inversa y ZMP (Zero Moment Point) (Rodríguez, 2017).

Estos métodos se pueden variar en diferentes resultados por diferencia entre los tamaños o las especificaciones de un robot humanoide. Las situaciones del robot se logran mediante una compleja ecuación matemática. (Lee, 2014)

El movimiento inestable del caminar del robot humanoide se puede detectar incluso en el caso de robots humanoides programados con la misma estructura y mecanismo. Esos pueden ser causados por la característica del actuador, los marcos que necesitan un proceso mecánico o un error en la medición del enlace. Además, es inevitable manejar el escenario con habilidad de programación competente para la realización del robot que se basa en cinemática inversa. Es por ello que se toma en consideración el centro de masa (CoM) y el Punto de momento cero ZMP tal como se aprecia en la Figura 2.1. (Lee, 2014)

Figura 2.1. Movimiento de un Robot Humanoide.



Polígonos de soporte de postura única y doble postura de soporte, (Lee, 2014)

2. Cinemática

Los métodos cinemáticos, que se originan en el campo de la robótica, es uno de los enfoques utilizados en la especificación y control del movimiento. La animación de un robot humanoide se puede realizar cambiando las orientaciones de las articulaciones por tiempo. El control de movimiento es la gestión de las transformaciones de las articulaciones por tiempo. La cinemática es el estudio del movimiento al considerar la posición, la velocidad y la aceleración. No enfatiza sobre las fuerzas subyacentes que

producen el movimiento. la cinemática directa e inversa son las dos categorías principales de la animación cinemática. (Memisoglu, 2003)

La cinemática describe los movimientos de los cuerpos sin tener en cuenta la causa. Las relaciones son fundamentales para todos los tipos de control de robots y al calcular las trayectorias de los mismos. Un control del robot más avanzado implica, por ejemplo, momentos de inercia y sus efectos sobre la aceleración de las articulaciones del robot individual y el movimiento de la herramienta. En los modelos cinemáticos, se estudian la posición, la velocidad, la aceleración y las derivadas más altas de las variables de posición de la herramienta de robot. La cinemática de robots estudia especialmente cómo se mueven varios enlaces entre sí y con el tiempo. Esto implica que la descripción cinemática es geométrica. (Wallén, 2008)

Las articulaciones giran y controlan la posición angular relativa de los enlaces del manipulador, notando que no todas las combinaciones de posiciones de articulaciones en la cadena son válidas, porque algunas combinaciones producen colisiones entre los eslabones de la cadena o con algún elemento fijo del entorno, como el piso o una pared. Todas las combinaciones válidas de valores conjuntos forman el espacio conjunto. El término grados de libertad (DOF) con su término en inglés (The term degrees of freedom) se refiere a la cantidad de uniones en una cadena cinemática; claramente, más DOF implica más flexibilidad en el movimiento de la cadena. La cinemática del robot es la aplicación de la geometría al estudio de cadenas cinemáticas con múltiples grados de libertad. Más específicamente, la cinemática del robot proporciona la transformación desde el espacio de la articulación, donde se definen las cadenas cinemáticas, al espacio cartesiano, donde se mueve el robot, y viceversa. La cinemática del robot es bastante útil, ya que se puede utilizar para planificar y ejecutar movimientos, así como para calcular las fuerzas y pares de los actuadores. (Kofinas, 2012)

2.2.1. Cinemática Directa

Existen muchos parámetros de movimiento que deben controlarse para tener el efecto deseado en un personaje simple. La cinemática de avance es la forma de establecer dichos parámetros de movimiento, como la posición y la orientación del esqueleto, en momentos específicos. Al establecer directamente las rotaciones en las articulaciones

y dar una transformación global a la articulación de la raíz, se pueden manejar diferentes poses. El movimiento del efector final (en el caso la pierna) está determinado por manipulaciones indirectas de transformaciones en la articulación de la raíz del esqueleto y rotaciones en las articulaciones de la pierna y de la rodilla. Matemáticamente, la cinemática directa se expresa como: (Memisoglu, 2003)

$$x = f(\theta)$$ Ec. (2.1)

Donde, dado θ, deriva de x. En el caso de la pierna, dados los ángulos de rotación de la rodilla y la cadera, la posición del pie se calcula utilizando las posiciones y las rotaciones de la cadera y la rodilla. Para trabajar en las figuras articuladas, debe existir una representación de matriz, que definirá la rotación y las orientaciones. Denavit y Hartenberg fueron los primeros que desarrollaron una notación, llamada notación DH, para representar la formulación de la matriz para la cinemática directa. La notación DH es una notación basada en enlaces donde cada enlace se representa con cuatro parámetros; θ, d, a y α.

Donde:

- θ es el ángulo de articulación, para el tipo rotacional.
- d es la distancia desde el origen, para prismática o lineal.
- a es la distancia de desplazamiento, longitud del enlace (también se puede encontrar como l).
- α es el ángulo de desplazamiento.

La cinemática directa define un mapeo desde el espacio conjunto hasta el espacio cartesiano tridimensional. Dada una cadena cinemática con *m* juntas y un conjunto de valores de unión (θ1, θ2, ..., θm), la cinemática directa puede encontrar la posición (px, py, pz) y la orientación (ax, ay, az) del efector final de la cadena cinemática en el espacio xyz tridimensional.

La cinemática directa es un problema independiente del dominio y se puede resolver para cualquier cadena cinemática simple o compleja que proporcione una solución analítica de forma cerrada. (Kofinas, 2012)

2.2.2. Cinemática Inversa

Los robots humanoides generalmente necesitan alcanzar puntos objetivo o seguir trayectorias en el espacio tridimensional. Para hacer que el efector final alcance un punto o siga una trayectoria, el diseñador debe especificar los valores apropiados para las uniones de la cadena cinemática. La cinemática inversa define formas de ir del espacio tridimensional al espacio conjunto. En particular, la cinemática inversa define una relación entre puntos en el espacio tridimensional (posición (px, py, pz) y orientación (ax, ay, az)) y valores / ángulos de unión (θ1, θ2, ..., θm) en el espacio conjunto de una cadena cinemática con *m* juntas. El problema de la cinemática inversa depende del dominio y cada cadena cinemática tiene una solución diferente. La solución al problema cinemático inverso puede conducir a una ecuación analítica de forma cerrada o a una aproximación numérica e iterativa (por ejemplo, con el método de aproximación jacobiano). A medida que aumenta el número de DOF, un punto en el espacio tridimensional puede tener más de un punto coincidente en el espacio conjunto. Esta multiplicidad de soluciones hace que la cinemática inversa sea una relación, no un mapeo. (Kofinas, 2012)

3. Transformaciones Afines

Una transformación afín es un mapeo que transforma puntos y vectores de un espacio a otro, de una manera que preserva las relaciones de distancias. Los espacios de origen y destino pueden ser n-dimensionales con $n \geq 2$. Las siguientes son transformaciones afines: contracción geométrica, expansión, dilatación, reflexión, rotación, cizalladura, transformaciones de similitud, similitudes de espiral y traducción.

Todas las combinaciones posibles de lo anterior producen una transformación afín también. La flexibilidad de las transformaciones afines con respecto a la representación de objetos en diferentes espacios, la convierte en una herramienta muy útil en gráficos por computadora. A los efectos de esta tesis, solo utilizamos la rotación y la traducción, por lo que nos centraremos solo en estos dos tipos de transformación afín. Además, estamos trabajando en un espacio de trabajo cartesiano tridimensional y, por lo tanto, todas las definiciones y ejemplos a partir de ahora se centrarán en este espacio. (Kofinas, 2012)

3.3.1. Matriz de transformación

Una matriz de transformación afín es a ((n + 1) × (n + 1)) matriz, donde ***n*** es el número de dimensiones en el espacio en el que se define la transformación. En general, una matriz de transformación es una matriz de bloques de la forma:

$$T = \begin{bmatrix} X & Y \\ [0 \dots 0] & 1 \end{bmatrix} \quad \text{Ec. (2.2)}$$

Donde X es una matriz (n × n), Y es un vector (n × 1) y la última línea de T contiene n - 1 ceros seguido de un 1. Si se quiere aplicar la transformación, a un punto dado $\mathrm{p} = (\mathrm{p}_1, \mathrm{p}_2, \dots, \mathrm{p}_\mathrm{n})$ en el espacio de n dimensiones, simplemente se multiplica la matriz de transformación afín con el vector de columna $\upsilon = (\mathrm{p}_1, \mathrm{p}_2, \dots, \mathrm{p}_\mathrm{n}, 1)^T$: (Kofinas, 2012)

$$\upsilon' = \begin{bmatrix} p_1' \\ \vdots \\ p_n' \\ 1 \end{bmatrix} = T\upsilon = \begin{bmatrix} X & Y \\ [0 \dots 0] & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ \vdots \\ p_n \\ 1 \end{bmatrix} \quad \text{Ec. (2.3)}$$

Para un punto $\mathrm{p} = (\mathrm{p}_\mathrm{x}, \mathrm{p}_\mathrm{y}, \mathrm{p}_\mathrm{z})$ en el espacio tridimensional, la transformación será:

$$\upsilon' = \begin{bmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{bmatrix} = T\upsilon = \begin{bmatrix} X_{xx} & X_{xy} & X_{xz} & Y_x \\ X_{yx} & X_{yy} & X_{yz} & Y_y \\ X_{zx} & X_{zy} & X_{zz} & Y_z \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \quad \text{Ec. (2.4)}$$

La matriz que resulta de la multiplicación de dos matrices de transformación afines T_1 y T_2 sigue siendo una transformación afín:

$$T = T_1 T_2 = \begin{bmatrix} X_1 & Y_1 \\ [0 \dots 0] & 1 \end{bmatrix} \begin{bmatrix} X_2 & Y_2 \\ [0 \dots 0] & 1 \end{bmatrix} = \begin{bmatrix} X_1 X_2 & X_1 X_2 + Y_1 \\ [0 \dots 0] & 1 \end{bmatrix} \quad \text{Ec. (2.5)}$$

Esta propiedad se generaliza al producto de cualquier número de matrices de transformación:

$$\hat{T} = T_1 T_2 T_3 \dots T_K = \begin{bmatrix} \hat{X} & \hat{Y} \\ [0 \dots 0] & 1 \end{bmatrix} \quad \text{Ec. (2.6)}$$

Una matriz de transformación afín es invertible, si y solo si X es invertible, y toma la forma:

$$T^{-1} = \begin{bmatrix} X^{-1} & -X^{-1}Y \\ [0 \dots 0] & 1 \end{bmatrix} \quad \text{Ec. (2.7)}$$

3.3.2. Matriz de traducción

La traducción en un espacio cartesiano es una función que mueve (traduce) cada punto en una distancia fija en una dirección específica. Podemos describir una traducción en el espacio tridimensional con una matriz (4 × 4) de la siguiente forma:

$$A = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Ec. (2.8)}$$

Donde d_x, d_y y d_z definen la distancia de traslación a lo largo de los ejes $x, y\ y\ z$, respectivamente. Aparentemente, la matriz de traducción es una matriz de transformación afín con $X = I$. Por lo tanto, para mover un punto $p = (p_x, p_y, p_z)$ en el espacio tridimensional por distancias (d_x, d_y, d_z), simplemente se aplica la transformación: (Kofinas, 2012)

$$\upsilon' = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = A\upsilon = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \quad \text{Ec. (2.9)}$$

3.3.3. Matriz de rotación

La rotación en un espacio cartesiano es una función que rota vectores en un ángulo fijo sobre una dirección especificada, una rotación en el espacio n-dimensional se describe como una matriz (n × n) ortogonal R con determinante 1:

$$\mathrm{R}^{\mathrm{T}} = R^{-1} \qquad RR^T = R^T R = I \qquad \det(R) = 1 \quad \text{Ec. (2.10)}$$

En el espacio cartesiano tridimensional hay tres matrices de rotación distintas, cada una de las cuales realiza una rotación de $\theta_x, \theta_y, \theta_z$ sobre el eje x, y, z respectivamente, asumiendo un sistema de coordenadas a la derecha:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \quad \text{Ec. (2.11)}$$

$$R_y = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \quad \text{Ec. (2.12)}$$

$$R_z = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Ec. (2.13)}$$

Para rotar un vector definido por el punto final $p = (p_x, p_y, p_z)$ sobre un eje específico, uno simplemente puede multiplicarse con la matriz de rotación correspondiente. Para rotar el vector primero sobre el eje xy luego sobre el eje y, se tiene que multiplicar con las matrices de rotación correspondientes en el siguiente orden:

$$\mathrm{p}' = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = R_y R_x p = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \quad \text{Ec. (2.14)}$$

Aparentemente, las tres matrices de rotación se pueden combinar para formar nuevas matrices de rotación para realizar rotaciones complejas sobre todas las dimensiones. Por ejemplo, la matriz de rotación que gira vectores primero sobre el eje x, luego sobre el eje y, y finalmente sobre el eje z es la siguiente:

$$R' = R_z R_y R_x \quad \text{Ec. (2.15)}$$

La forma analítica de la matriz de rotación anterior es la siguiente:

$$R' = \begin{bmatrix} \cos\theta_y \cos\theta_z & -\cos\theta_x \sin\theta_z + \sin\theta_x \sin\theta_y \cos\theta_z & \sin\theta_x \sin\theta_z + \cos\theta_x \sin\theta_y \cos\theta_z \\ \cos\theta_y \sin\theta_z & \cos\theta_x \cos\theta_z + \sin\theta_x \sin\theta_y \sin\theta_z & -\sin\theta_x \cos\theta_z + \cos\theta_x \sin\theta_y \sin\theta_z \\ -\sin\theta_y & \sin\theta_x \cos\theta_y & \cos\theta_x \cos\theta_y \end{bmatrix}$$

Ec. (2.16)

Se puede transformar fácilmente cualquier matriz de rotación Rb en una matriz de transformación afín R simplemente rellenando la última línea y la última columna con (0, ..., 0, 1):

$$R = \begin{bmatrix} & \hat{R} & & \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \\ [0 & \dots & 0] & 1 \end{bmatrix}$$

Ec. (2.17)

A partir de ahora, cualquier matriz de rotación será una matriz de transformación afín.

3.3.4. Matriz de transformación afín y cinemática

A los efectos de la cinemática, se utiliza matrices de rotación y traslación, de modo que se pueda transformar puntos en el espacio tridimensional. Considerando matrices de transformación afines que combinan rotación y traslación; el bloque X de la matriz define la rotación, mientras que el bloque Y de la matriz define la traslación: (Kofinas, 2012).

$$T = \begin{bmatrix} & \hat{R} & & \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} \\ [0 & 0 & 0] & 1 \end{bmatrix}$$

Ec. (2.18)

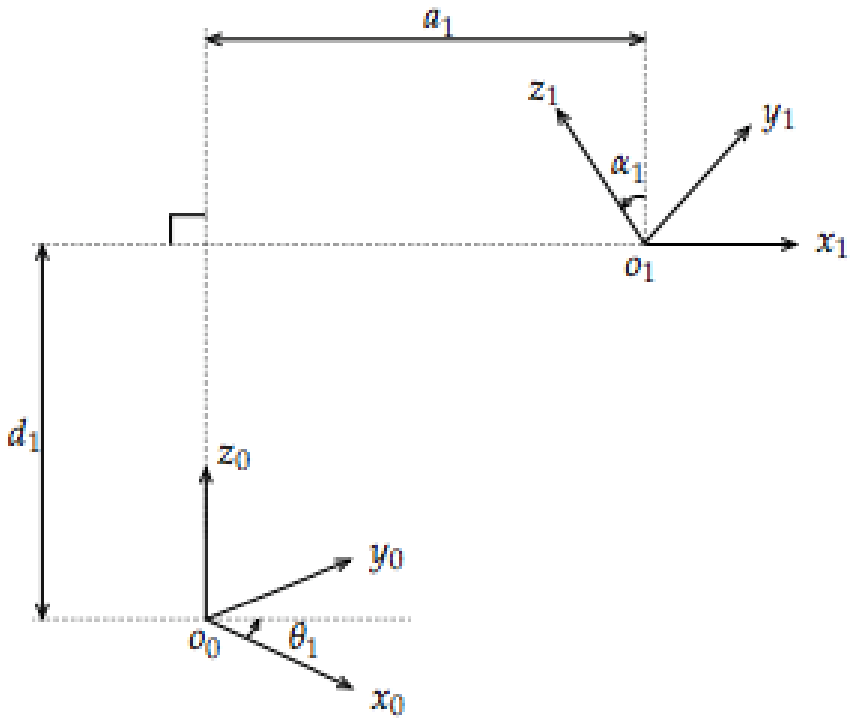
3.4. Parámetros de Denavit - Hartenberg (DH)

Denavit y Hartenberg han ideado una forma de crear una matriz de transformación que describe los puntos en un extremo de una articulación a un sistema de coordenadas que se fija en el otro extremo de la articulación, como una función del estado conjunto. Concluyeron que se puede describir completamente esta matriz de transformación usando solo cuatro parámetros, conocidos como parámetros Denavit Hartenberg (DH): a, α, d y θ. Antes de explicar estos parámetros, primero se establece el marco de

referencia de cada articulación i con respecto al marco de referencia de su unión anterior: (Kofinas, 2012)

- El eje zi se establece en la dirección del eje de la articulación (la dirección de rotación).
- El eje xi es paralelo a la normal común entre zi y zi-1 (producto exterior). La dirección de xi se deriva usando la regla de la mano derecha de zi-1 a zi.
- El eje yi se sigue de los ejes xi y zi para formar un sistema de coordenadas a la derecha. (Memisoglu, 2003)

Figura 2.2. Parámetros Denavit - Hartenberg (DH)



Parámetros de enlace D-H θ1, d1, a1, α1 y las transformaciones correspondientes entre el cuadro 0 y el cuadro 1, de acuerdo con la representación estándar D-H., (Wallén, 2008)

Ahora, se puede describir los parámetros DH (Ver Tabla 2.1.):

- a: longitud de la normal común.
- α: ángulo sobre la normal común, desde zi-1-axis a zi-axis.
- d: desplazamiento a lo largo del eje zi-1 a la normal común.
- θ: ángulo alrededor del eje zi-1, desde xi-1-axis hasta xi-axis.

Tabla 2.1. Matriz de Denavit - Hartenberg

Parámetro	Descripción
$\boldsymbol{\theta}$	Ángulo entre $X_i - 1$ y x_i referido a Z_i
$\boldsymbol{d}$	Distancia entre $X_i - 1$ y x_i medida a lo largo de Z_i
$\boldsymbol{a}$	Distancia entre $Z_i - 1$ y Z_i medida a lo largo de X_i
$\boldsymbol{\alpha}$	Ángulo entre Z_i y $Z_i - 1$ referido a X_i

Parámetros de la Matriz de Denavit – Hartenberg, Fuente: (Memisoglu, 2003)

Ahora, al pasar del marco de referencia base de alguna unión al marco de referencia transformado de esta unión usando la matriz de transformación TDH, que consiste en dos traducciones y dos rotaciones parametrizadas por los parámetros DH de la articulación se tiene:

$$\mathrm{T_{DH}} = \mathrm{R_x}(\alpha)\mathrm{T_x}(a)\mathrm{R_z}(\theta)\mathrm{T_z}(d) \qquad \text{Ec. (2.19)}$$

La forma analítica de la matriz resultante de la composición anterior es la siguiente:

$$T_{DH} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & a \\ \sin\theta\cos\alpha & \cos\theta\cos\alpha & -\sin\alpha & -d\sin\alpha \\ \sin\theta\cos\alpha & \cos\theta\sin\alpha & \cos\alpha & d\cos\alpha \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{Ec. (2.20)}$$

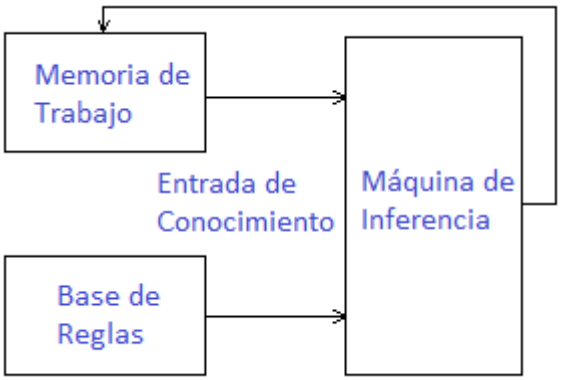
3.5. Sistema Experto

El sistema experto es una rama de la ciencia de la inteligencia artificial que el humano inspiró desde su propio ser. El ser humano siempre trata de heredar sus propias experiencias a las siguientes generaciones. Con la amplia difusión de la información del presente siglo, se impuso una nueva necesidad de emular la experiencia y el comportamiento humano de manera similar; a partir de este punto, se han inventado sistemas informáticos expertos. El sistema experto está transformando principalmente las experiencias humanas en formas de software. Para actuar de manera similar al comportamiento del ser humano. El sistema experto siempre recopila una gran cantidad de información de su dominio y los transforma en conocimiento, utilizando las reglas que el humano asignó según sus propias experiencias. (Lastra, 2013)

3.5.1. Sistema Experto basado en reglas

Hay tres partes en un sistema experto: una regla base, memoria de trabajo y un motor de inferencia. La primera parte en un sistema experto es la base de reglas. Esto se compone de dos partes: un conjunto de reglas y un diccionario. El conjunto de reglas define el conocimiento experto en forma de reglas que pueden considerarse como una colección de oraciones de causa y efecto. Además, las reglas también pueden llevar información de incertidumbre. La información de incertidumbre contiene información sobre cómo se obtuvieron las reglas y cuán precisas o plausibles son las afirmaciones de la regla. Si bien los sistemas expertos pueden diferir en la forma de incertidumbre que utilizan, esta investigación se centrará en el uso de la teoría de conjuntos aproximados, pero se implementará de manera que se adapte a los cambios futuros. (Cernik, 2009)

Figura 2.3. Arquitectura del Sistema Experto



Sistema experto basado en reglas, (Cernik, 2009)

3.5.2. Creación de un sistema experto

Para crear un sistema experto, el usuario o diseñador debe tener: una fuente de conocimiento experta, un motor de inferencia, un entendimiento sobre cómo construir una base de reglas y conocimiento sobre cómo ingresar y recuperar IO del sistema experto. La parte más difícil es obtener el conocimiento para crear la base de reglas. Estas fuentes de conocimiento pueden provenir de varios lugares, como expertos de dominio, minería de datos y otros dispositivos heredados. (Cernik, 2009)

Para crear actualmente un sistema experto, un programador debe tomar la fuente de conocimiento y traducirla a la forma de regla. Si bien esto puede sonar fácil, implica

que el programador posee una comprensión parcial sobre el conocimiento que está codificando y el lenguaje del sistema experto en el que está emulando la capacidad de tomar decisiones. Una vez que el conocimiento se ha transferido a una base de reglas, el usuario debe proporcionar información al sistema experto, en forma de memoria de trabajo. Esta entrada puede provenir de una GUI, consola o script dependiendo del tipo de aplicación. Una vez que esto se complete, el usuario puede ejecutar el sistema experto y traducir la respuesta de la memoria de trabajo. (Cernik, 2009)

3.5.3. Componentes del sistema experto basado en el conocimiento

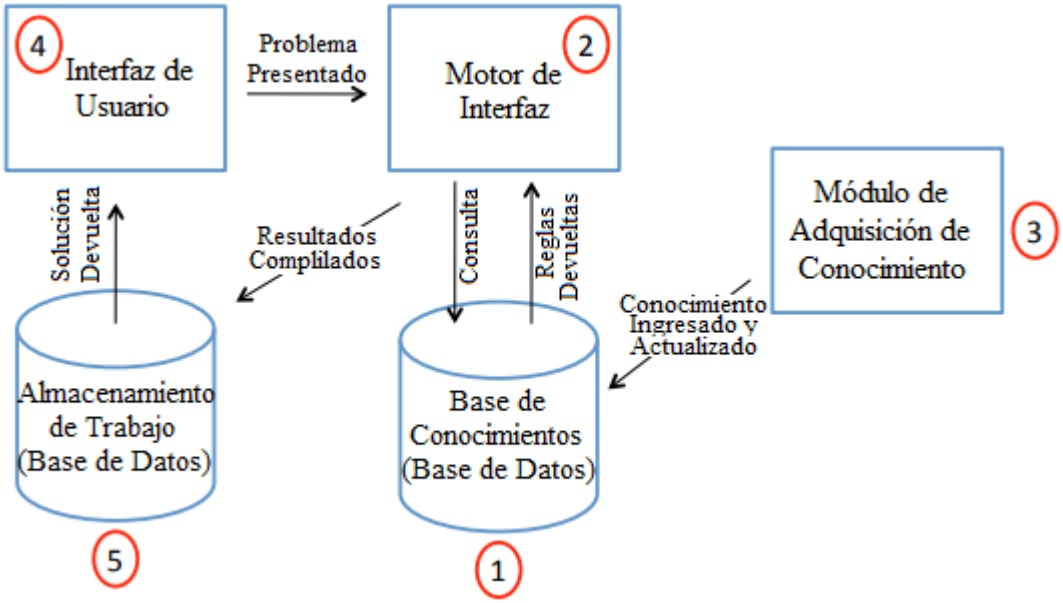
Los sistemas expertos basados en el conocimiento se componen de varios componentes independientes. Como se aprecia en la Figura 2.4, muestra los componentes independientes y cómo trabajan juntos para resolver un problema dentro del dominio del problema. Las flechas representadas en la Figura datan el flujo de información en todo el sistema. El primer componente es la base de conocimiento en la que se almacenan el conocimiento heurístico de los expertos del dominio, así como los hechos pertinentes sobre el problema (Grosan, C., 2011).

El segundo componente es el motor de inferencia que utiliza estrategias de los dominios de búsqueda y ordenación para probar las reglas contenidas en la base de conocimientos sobre un problema en particular. El motor de inferencia lo logra al consultar información de la base de conocimiento y aplicar los resultados devueltos.

El módulo de adquisición de conocimiento, el tercer componente, facilita la transferencia de conocimiento a la base de conocimiento para uso futuro (Grosan, C., 2011). El cuarto componente es la interfaz de usuario que permite a los usuarios interactuar con el sistema experto basado en el conocimiento presentando el problema al motor de inferencia y las soluciones de visualización.

El quinto componente es el almacenamiento de trabajo que el sistema experto basado en el conocimiento utiliza para almacenar información mientras se resuelve un problema específico y luego contiene la información correspondiente a la solución. (Lucien Millette, 2012)

Figura 2.4. Componentes del Sistema Experto



Componentes independientes del Sistema Experto basado en el Conocimiento, (Lucien Millette, 2012)

3.5.4. Obstáculos a la adquisición de conocimiento

Un problema relacionado con los sistemas expertos es la adquisición de información del mismo que se define como el proceso de extracción de datos de los expertos en el dominio del problema, para definir la funcionalidad requerida del sistema experto. La adquisición de conocimiento ha sido referida como el "cuello de botella en el proceso de construcción de sistemas expertos" (Golabchi, 2008)

Los dos grupos clave de partes interesadas durante la adquisición de la información son el ingeniero del conocimiento y los expertos en el dominio del problema. El ingeniero actúa como el conducto para extraer información específica del dominio de los expertos y del problema (Lucien Millette, 2012)

3.6. Base de Datos a Matlab

Database Toolbox conecta Matlab a una base de datos utilizando las funciones de MATLAB. La información se recupera como una cadena. En ese punto, utiliza el amplio conjunto de herramientas de Matlab para trabajar con los datos. Puede incluir las funciones de la “Caja de herramientas” en los archivos M de Matlab. Database Toolbox también viene con Visual Query Builder (VQB), una interfaz gráfica de usuario fácil de usar para recuperar datos. Con el VQB, crea consultas seleccionando

información de listas en lugar de ingresar funciones de Matlab. El VQB los recupera en una matriz de células Matlab para que luego pueda procesar utilizando el conjunto de funciones. Con el VQB, puede visualizar la información recuperada en tablas relacionales, informes y gráficos. (Math Works Inc. & Guide, 2000)

3.7. Placa de desarrollo LattePanda

LattePanda es una de las placas de desarrollo más poderosa del mercado. Se basa en el chip Intel Atom CherryTrail, que es capaz de ejecutar la versión completa de Windows 10. También es compatible con Linux y Android, y es la única placa de desarrollo que puede ejecutar windows / linux / android. Incluye todo lo que tiene una PC común y puede hacer cualquier cosa al igual que un ordenador . Es compatible con casi todos los dispositivos conocidos: impresoras, joysticks, cámaras y más. Cualquier periférico que funcione en tu PC funcionará en LattePanda. (Lattepanda, Studio, 2017)

LattePanda puede instalar la edición completa de Windows 10, incluidas potentes herramientas como Visual Studio, NodeJS, Java, Processing y más. Usando las API existentes, puede desarrollar sus propios proyectos de software y hardware en LattePanda como lo haría en una PC normal: C #, Javascript, Ruby, etc.

LattePanda también está diseñado con un coprocesador compatible con Arduino, lo que significa que puede usarse para controlar y sentir el mundo físico, ¡como una placa Arduino! (Lattepanda, Studio, 2017)

Figura 2.5. Placa de Desarrollo LattePanda

Tarjeta LattePanda con coprocesador Atmega, (Lattepanda, Studio, 2017)

3.7.1. Características

Tabla 2.2. Características de la Placa de Desarrollo LattePanda

Sistema	
Características	**LattePanda**
Procesador	Intel Cherry Trail Quad Core 1.8GHz
Sistema Operativo	Edición completa preinstalada de Windows 10
Caché	2 MB
RAM	2GB DDR3L - 4GB DDR3L (Lattepanda mejorado)
Capacidades Gráficas	Intel Gen 8
Video Compatible	Decodificación 1080 HE 60 HEVC, H264, VP8
Salida de video	Conector de pantalla HDMI
Capacidad de Almacenamiento	32 GB - 64 GB (Lattepanda mejorado)
Ranura MicroSD	
Coprocesador	Atmega32u4 Arduino Compatible

Especificaciones de LattePanda, Fuente: (Casserfelt & Einestam, 2017)

3.8. Interfaz de comunicación CES Robot Humanoide NodeMCU 1.0

NodeMCU es una tarjeta de desarrollo de hardware y software libre diseñana para la conexión de dispositos actuadores o sensores a internet ya que posee el módulo ESP12E (variante del ESP8266 módulo Wifi) el cual permite conexión inalámbrica en un red local. Además, incluye un microcontrolador o MCU lo que le permite conectar a el actuadores, sensores u otros dispositivos embebidos.

Posee compatibilidad con distintos lenguajes de programacion como:

- LUA (lenguaje de programación imperativo, estructurado y bastante ligero)
- Micropython (implementación del lenguaje de programación Phyton 3 escrito en C)
- Arduino.

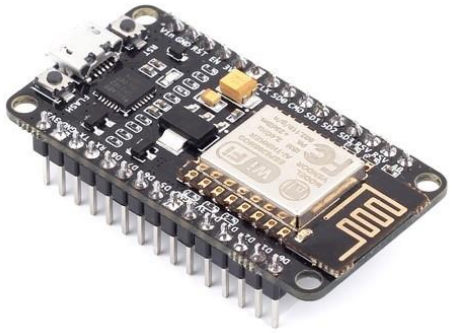
Tabla 2.3. Características placa NodeMCU 1.0

Características	NodeMCU
Voltaje de alimentación	5 V DC
Voltaje Entradas/Salidas	3.3V DC
CPU	Tensilica Xtensa LX3 (32 bit)
Pines Disgitales GPIO	17
Pin Análogo ADC	1 (0-1V)
UART	2
SoC	Módulo ESP12
Antena	En PCB
Módulos integrados	WEP, TKIP, AES y WAPI
Comunicación extra	I2C

Especificaciones de LattePanda, Fuente: (Casserfelt & Einestam, 2017)

La version 1.0 de la placa NodeMCU cuenta con el chip ESP12E que a diferencia de su versión anterior le permiter tener algunos pines a favor. Es caracterizado por su tamaño ya que facilmente encaja en un protoboard.

Figura 2.6. Placa NodeMCU 1.0



Acelerómetro y giroscopio MPU 6050, (Escudero, Maria, & Suarez, 2016)

3.9. IMU MPU-6050

IMU básicos (Unidad de medición de inercia) MPU-6050 es un sensor que contiene un acelerómetro (micro-electro-mechanical systems MEMS (sistemas microelectromecánicos)) y un giroscopio MEMS en un chip. (Fedorov, Ivoylov, Zhmud, & Trubin, 2015)

Figura 2.7. Sensor MPU-6050

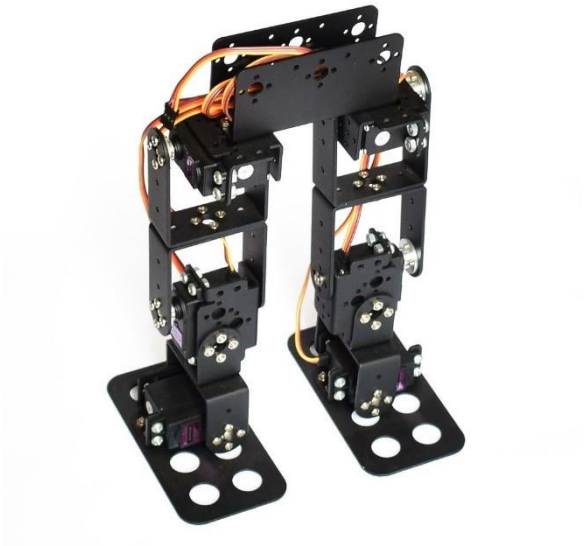
Acelerómetro y giroscopio MPU 6050, (InvenSense Inc., 2013)

Tanto el acelerómetro como el giroscopio contienen 3 ejes que pueden capturar x, y, z con hardware de conversión de analógico a digital de 16 bits para cada canal. El MPU-6050 usa I2C para la comunicación, que es un bus de computadora serie multi-maestro, multiescalar, de un solo extremo, con baja velocidad pero muy útil porque usa solo dos cables: líneas SCL (reloj) y SDA (datos). (Schwartz & Park, 2015).

3.10. Estructura de Robot Humanoide Bipedo 6 Dof

Bípedo 6 Dof es una estructura comercial de robot humanoide bípedo. Las piezas son maquinadas de aluminio de 2mm de grosor con protección a la oxidación. Se puede usar con 6 servo motores, 3 por cada pierna.

Figura 2.8. Robot Humanoide Bípedo 6 Dof.



Estructura de aluminio Robot Bípedo 6 Dof, (Hun-ok Lim, 2010)

Dada la naturaleza de su construcción sus movimientos son limitados, pero permite movimientos pierna libre en un plano horizontal, además de caminar hacia adelante y atrás. Posee en cada eje de sus articulaciones un rodamiento el cual le permite realizar un movimiento suave y prolijo a la vez que mantiene la articulación nivelada y con un eje concéntrico. Para su ensamblaje consta de 23 piezas en total, sin contar elementos de sujeción como tornillos y tuercas.

Tabla 2.4. Dimensiones de Estructura Robot Humanoide 6 Dof

Característica	**Dimensión**
Alto	19 cm
Ancho	12 cm
Largo	14 cm
Peso	100 gr

Características básicas del Robot Bípedo 6 Dof, (Hun-ok Lim, 2010)

3.11. ServoMotores HITEC 422

HITEC 422 es un servo motor compuesto por un tren de engranes de resina de alto impacto el cual brinda confiabilidad para torques no mayores a 4.10 kg/cm. Posee doble soporte para el eje central.

Figura 2.9. Servomotor Hitec 422

Servomotor Hitec 422, (DFRobot, 2017)

Su voltaje de trabajo se encuentra entre 4.8 – 6V DC lo que le permite tener velocidades de giro de 0.21 segundos/60° y 0.16 segundos/60° respectivamente.

Tabla 2.5. Características Servo Motor Hitec 422

Características	**Hitec 422**
Modulación	Análoga
Torque	4.8V: 3.31 kg/cm 6.0V: 4.10 kg/cm
Peso	45.4 g
Rango de Rotación	180°
Ciclo de Pulso	20 ms
Tamaño de Pulso	900 – 2100 us

Características básicas del servo motor Hitec 422, (DFRobot, 2017)

CAPÍTULO 3
DESCRIPCIÓN DEL ROBOT HUMANOIDE
DISEÑO E IMPLEMENTACIÓN

3.1. Introducción

En este capítulo se presenta la descripción de la planta, en este caso del Robot Humanoide, además se realiza el acoplamiento del Robot con la tarjeta, necesario para transferirlo al entorno de Matlab y desarrollar el sistema de aprendizaje empleando la cinemática inversa.

El sistema consta de dos partes, el CES o Sistema Embebido de Control y el Robot Humanoide. Ambos poseen distintos dispositivos que permiten la comunicación entre ellos y su control. En la Figura 3.1 se presenta una descripción gráfica del sistema.

Figura 3.1. Placa de Desarrollo LattePanda

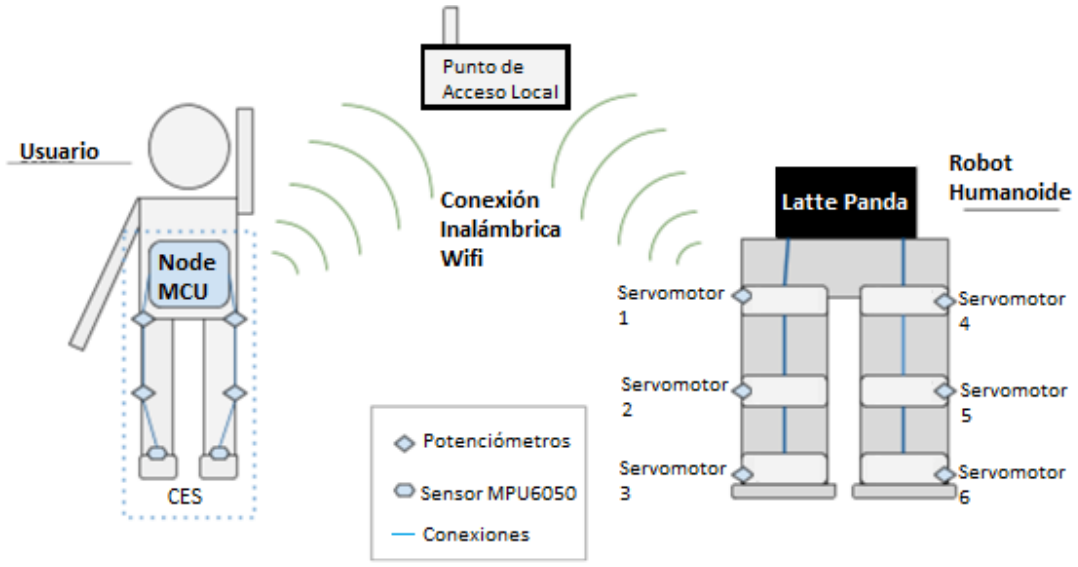


Diagrama general del funcionamiento del sistema CES - Robot Humanoide, (Elaborado por: David Loja)

3.2. Implementación del CES (Sistema Embebido de Control)

El CES permite al usuario adquirir los movimientos articulares de las extremidades inferiores y enviarlos inalámbricamente al Robot Humanoide mediante el protocolo de comunicación UDP (Protocolo de Datagramas de Usuario).

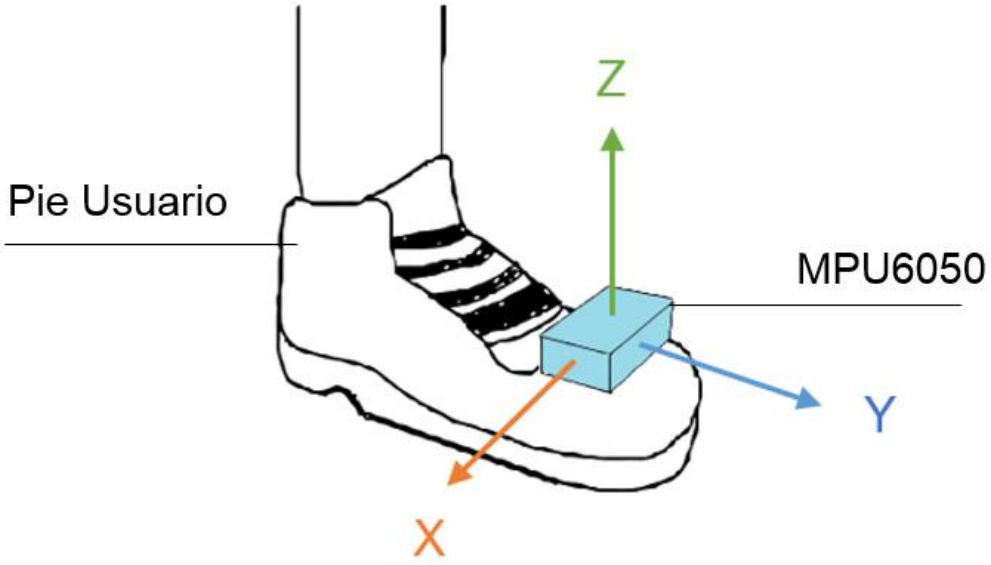
3.2.1. Ubicación y Conexión del CES

Para la adquisición de datos el CES cuenta con acelerómetros ubicados en la parte superior del pie y potenciómetros en cuatro articulaciones clave del usuario, como son cadera y rodilla. Los acelerómetros recogen los datos de aceleración lineal, de cada pie por separado, en los tres ejes X, Y, Z.

Por razones de ubicación del sensor, como se muestran en la Figura 3.3, el CES valida únicamente los datos de Z en la vertical, Y en la Horizontal; ya que el eje X se encuentra apuntando en dirección ortogonal al plano que forma ZY lo cual no es un movimiento válido para el robot humanoide mostrado en la Figura 3.2.

El sensor MPU6050 acelerómetro es sujeto al pie del usuario por una banda elástica de fácil uso.

Figura 3.2. Ubicación del sensor MPU6050

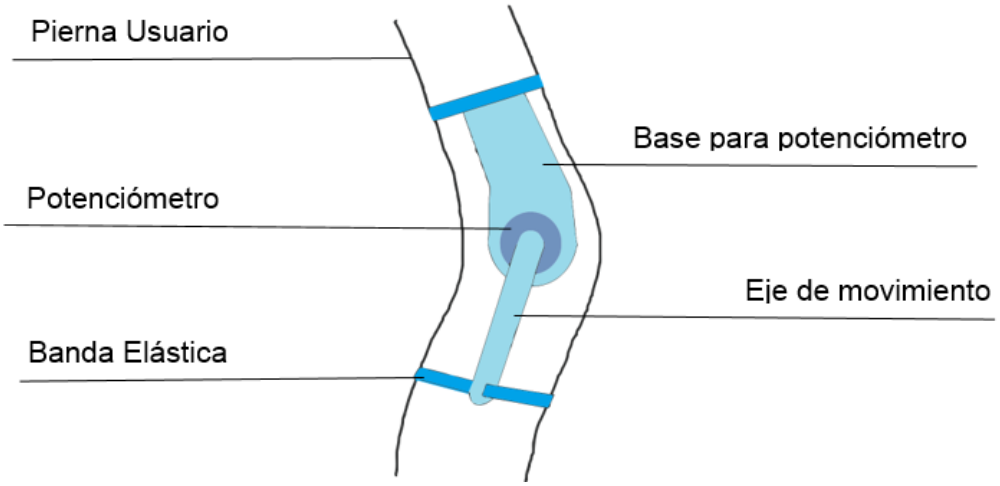


Representación gráfica MPU6050, (Elaborado por: David Loja)

Por otra parte, los potenciómetros ubicados en cada articulación se encuentran sujetos a las articulaciones del beneficiario mediante elásticos permitiendo que no existan restricciones ajenas al movimiento natural de las extremidades inferiores del usuario.

Además, esto permite que los sensores giren libremente y sin fuerzas que puedan causar al sistema a diferencia si estuviera fijamente sujeto a la articulación.

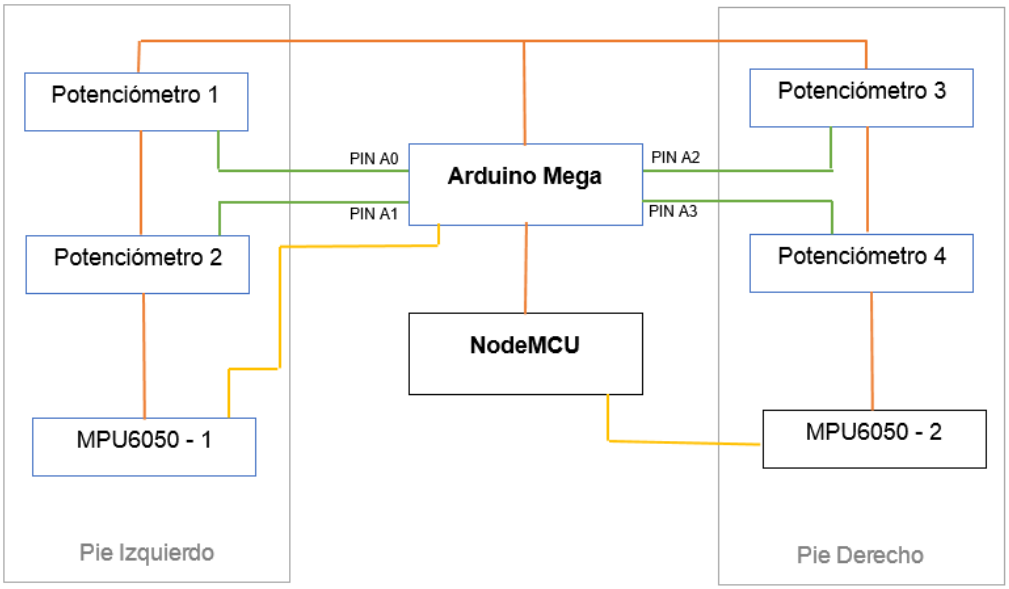
Figura 3.3. Ubicación del potenciómetro en rodilla derecha – vista lateral.



Representación gráfica potenciómetro, (Elaborado por: David Loja)

El CES consta en total de 4 potenciómetros, cada uno ubicado en las articulaciones de la cadera y rodilla respectivamente de cada pierna y 2 sensores MPU6050 uno por cada pie del usuario. Todos comparten la misma fuente de alimentación que alimenta al CES por medio del puerto USB del Arduino Mega de 5V DC. Por razones de optimización en la conexión física del sistema se implementó por separado la conexión de los 4 potenciómentros y el acelerómetro del pie izquierdo hacia el Arduino Mega mientras que el acelerómetro del pie derecho esta conectado a la placa NodeMCU.

Figura 3.4. Diagrama de Conexión del CES.



Esquema de conexión del CES, (Elaborado por: David Loja)

Cabe mencionar que los acelerómetros estan comunicados mediante I2C al Arduino Mega y a la placa NodeMCU respectivamente, siendo así dos redes de comunicación independientes en la adquisición de datos como se observa en la Figura 3.4. Al igual cada potenciómetro esta conectado a un pin de lectura análoga ADC del Arduino MEGA comenzando por A0 hasta A3. Todos los dispositivos comparten alimentación de 5V DC marcado en color anaranjado ademas de su negativo en comun GND.

3.2.2. Calibración de sensor MPU6050

Es necesario calibrar debido a que internamente el chip puede estar desnivelado. Previamente a la calibración se debe llamar a las librerías en el IDE de Arduino como se muestran:

```
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"
```

Para la comunicación I2C de Arduino con otros dispositivos se usa la librería "Wire.h". La librería empleada para la inspección de instrumentos I2C es "I2Cdev.h". La librería que permite extraer los valores de interpretación del acelerómetro y giroscopio es "MPU6050.h".

Como primer aspecto se lee los offsets actuales en el sensor, esto se realiza con el comando getXAccelOffset() en el IDE de Arduino y se asigna a las variables respectivas.

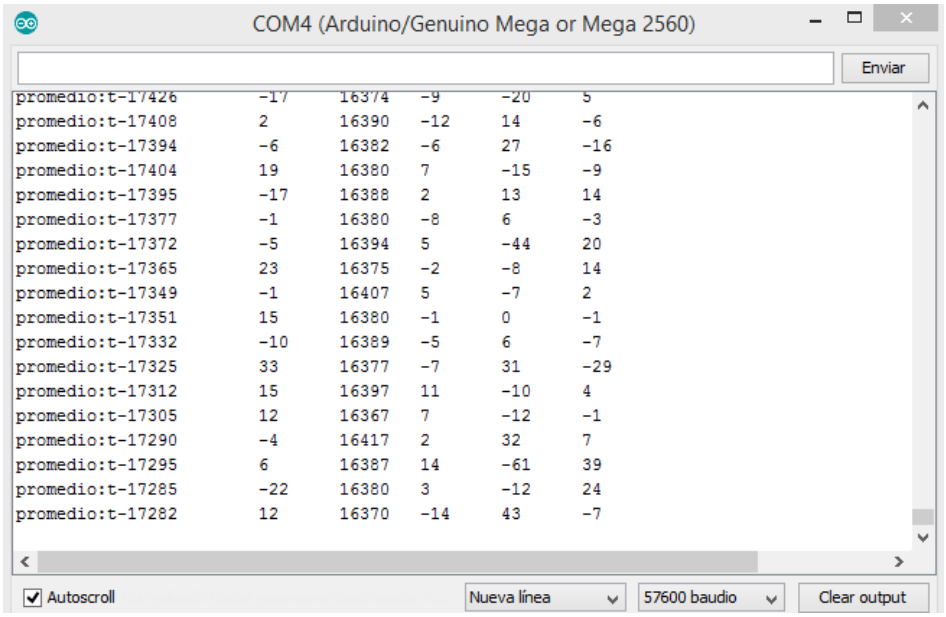
```
ax_o=sensor.getXAccelOffset();
ay_o=sensor.getYAccelOffset();
az_o=sensor.getZAccelOffset();
```

Las variables para almacenar los offsets provenientes de “sensor” que previamente fue declarado con MPU6050 sensor son “ax_o, ay_o, az_o”.

La dirección por defecto del sensor MPU6050 conectada en el Arduino es 0x68 ya que el pin físico AD0 del módulo se encuentra desconectado.

El siguiente proceso es tomar 100 lecturas para corregir los offset hasta que nuestros valores se aproximen a 0 o 16384 como se muestra en la siguiente captura de pantalla.

Figura 3.5. Lectura de offsets del sensor MPU6050



Calibración MPU6050, (Elaborado por: David Loja)

Para asignar los offsets calibrados al sensor se usa el comando setXAccelOffset() como se muestra:

```
sensor.setXAccelOffset(ax_o);
sensor.setYAccelOffset(ay_o);
sensor.setZAccelOffset(az_o);
```

3.2.3. Obtención de las coordenadas X, Y, Z

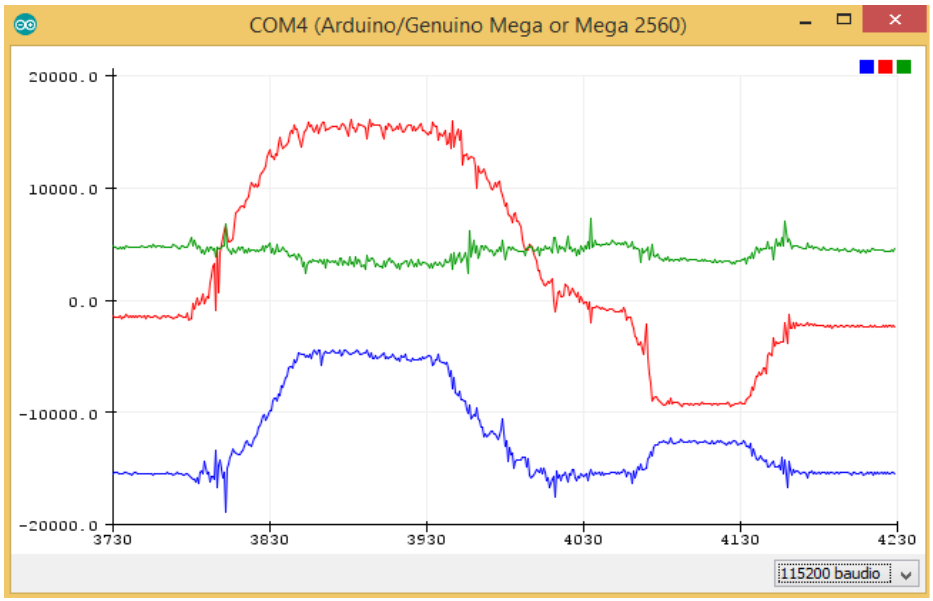
Para el sistema la coordenada Z siempre será igual a cero debido a que el robot humanoide no puede efectuar un movimiento fuera de su plano XY ya que físicamente en la cadera del mismo, únicamente pose un servomotor que le permite moverse en X+, X-, Y+, Y-. Por lo cual discriminaremos el dato de Z.

El robot humanoide debe recibir las coordenadas en X, Y para que posteriormente ejecute las acciones necesarias en los actuadores en base a los ángulos obtenidos por la cinemática inversa.

3.2.3.1. Adquisición y Calibración de X, Y mediante el sensor MPU6050

Para la obtención de las coordenadas X, Y mediante el sensor MPU6050 se tomó en cuenta la reacción de la aceleración lineal que tiene cada eje con los movimientos y con la gravedad. En la Figura 3.11 se observa la respuesta que tiene el sensor MPU6050 al desplazamiento vertical en Z y el horizontal en Y donde, la respuesta en azul es Z, rojo Y, verde X.

Figura 3.6. Respuesta del sensor MPU6050 al movimiento.



Gráfica de respuesta a movimiento vertical y horizontal, (Elaborado por: David Loja)

A partir de aquí en adelante se tomará como eje Z del usuario como eje Y del robot y el eje Y del usuario como eje X del robot debido a que por ubicación del sensor en el pie se debe hacer esta conversión para que sea más fácil de visualizar en los cálculos posteriores y en la programación.

Como se observa en la Figura 3.6 en la vertical mientras suba o baje la posición del pie va a aumentar o disminuir la cantidad de bits en la lectura por lo que se puede tomar la altura física con la que se eleva el pie y luego realizar un escalamiento para obtener la coordenada en Y que posteriormente se enviará al robot humanoide.

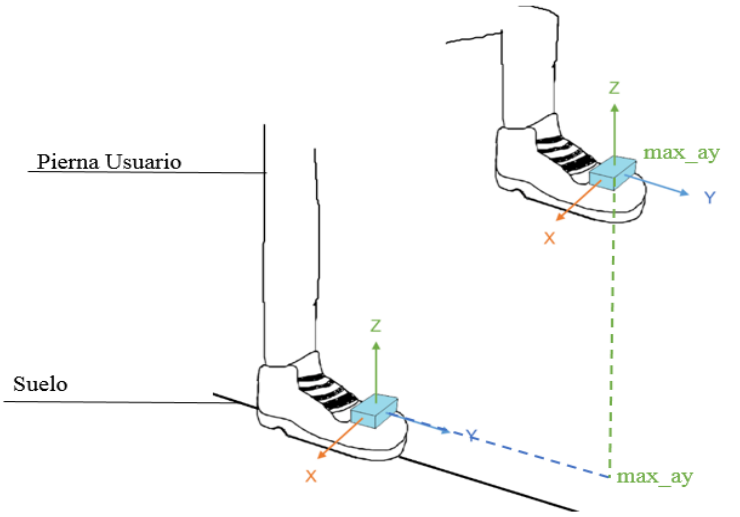
De igual manera se obtiene la coordenada en X, pero en este caso con la reacción en el eje horizontal.

Para la coordenada en X del robot humanoide se usa la aceleración lineal en Y del CES anteriormente explicada como se observa:

xai=map(ay, max_ay,min_ay,88,0);

La variable que recoge el escalamiento del sensor MPU6050 en el eje Y del mismo es “xai”. A partir de un proceso de calibración del CES el cual permite calibrar los desplazamientos máximos y mínimos que va a realizar el usuario con el CES surge “max_ay” y “min_ay”. En la Figura 3.7 se observa el proceso de calibración para la pierna derecha.

Figura 3.7. Desplazamiento en Z del CES para calibración.



Representación levantamiento pierna derecha, (Elaborado por: David Loja)

La altura máxima predeterminada es de 88 cm y la mínima de 0 cm. Estas medidas son estandar en el caso del usuario de muestras, es decir, van a variar dependiendo de cada usuario que vaya a colocarse el CES. El mismo proceso es realizado para el eje Y.

Para que el usuario conozca cual es el procedimento y que eje se encuentra calibrando, el CES imprime mediante comunicación serial de 115200 baudios el estado de la calibración respondiendo al siguiente código:

```
Serial.println("--- MAX AZ ---");
digitalWrite(alerta,HIGH);
```

```
delay(600);
digitalWrite(alerta,LOW);
delay(3000);
for(int i=0;i<=20;i++){
Wire.beginTransmission(MPU_addr);
Wire.write(0x3B);
Wire.endTransmission(false);
Wire.requestFrom(MPU_addr,14,true);
ax=Wire.read()<<8|Wire.read();
ay=Wire.read()<<8|Wire.read();
max_az=Wire.read()<<8|Wire.read();
Serial.println(max_az);}
Serial.println("--- MIN AZ ---");
digitalWrite(alerta,HIGH);
delay(600);
digitalWrite(alerta,LOW);
delay(3000);
for(int i=0;i<=20;i++){
Wire.beginTransmission(MPU_addr);
Wire.write(0x3B);
Wire.endTransmission(false);
Wire.requestFrom(MPU_addr,14,true);
ax=Wire.read()<<8|Wire.read();
ay=Wire.read()<<8|Wire.read();
min_az=Wire.read()<<8|Wire.read();
Serial.println(min_az);}
```

Además, el CES cuenta con un buzzer de 5V DC el cual le permite emitir un “pitido” para que el usuario sin necesidad de visualizar el estado de calibración le permita realizarlo correctamente. La variable “alerta” es la encargada de portar la señal de activación del buzzer, esta se encuentra conectada al pin 11 del Arduino Mega.

Para obtener datos más estables y para que el usuario pueda realizar los movimientos de calibración correctamente se empleó un tiempo de 3 segundos de espera para que mantenga la posición correcta y además un bucle de ejecución “for” que va a obtener 20 lecturas en ese instante asegurando de esta manera una lectura correcta del momento en el que se inició el programa.

El sensor MPU6050 al ser sensible frente a cualquier perturbación como un movimiento brusco, varía su medición en cualquier eje de manera espontánea. Por esta razón se empleó un complemento de medición siendo los potenciómetros en cada articulación los que mediante cinemática directa brinden estabilidad a la lectura de datos.

3.2.3.2. Adquisición y Calibración de X, Y mediante potenciómetros

Los potenciómetros en cada articulación al igual que los sensores MPU6050 necesitan ser calibrados por el usuario previamente para obtener el valor máximo y mínimo desplazamiento de cada articulación y posteriormente ser escalados a ángulos de giro, lo cual permite mediante cinemática directa encontrar los valores de X, Y que serán correlacionados con los X, Y de los acelerómetros y de esta manera estabilizar y discriminar datos erróneos por perturbaciones como movimientos bruscos.

Además, permitirá seguir obteniendo coordenadas en movimientos que el usuario no necesariamente tenga que mover el pie, es decir el punto final, como por ejemplo hacer el movimiento de sentadilla.

Para la lectura de los datos del ADC respectivo a cada potenciómetro se implementó el siguiente código en la pierna izquierda:

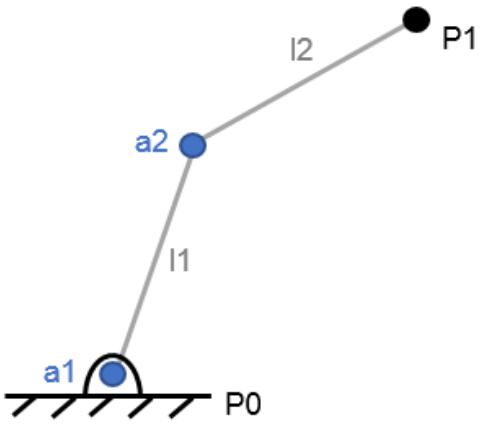
```
a_ci=map(sci,max_sci,min_sci,0,90);
a_pi=map(spi,max_spi,min_spi,180,90);
a_ci=(3.14159*a_ci)/180.0;
a_pi=(3.14159*a_pi)/180.0;
```

“a_ci, a_pi” son variables donde se almacenan los datos del ángulo de cada articulación proveniente de la lectura de “sci, spi” que corresponden al sensor de cadera izquierda

y sensor de pierna izquierda respectivamente. Luego se transforma de grados a radianes mediante un regla de tres simple.

Para optener los valores de X, Y mediante el sistema de potenciómetros se aplicó cinemática directa haciendo relación de un manipulador RR el cual se caracteriza por tener una base fija y dos eslabones rectos como se observa en la Figura 3.8.

Figura 3.8. Manipulador RR.

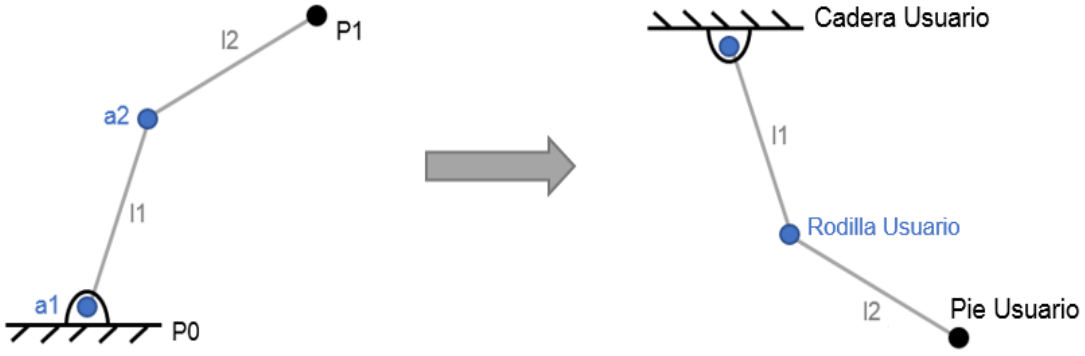


Representación gráfica manipulador RR, (Elaborado por: David Loja)

“a1, a2” corresponden a las articulaciones 1 y 2 respectivamente. Las longitudes de los dos brazos rectos llevan las variables “l1, l2”. El punto final con referencia al punto fijo “P0” es el punto “P1”.

Para el robot humanoide se toma en cuenta que el punto “P0” va a ser la cadera y el punto “P1” el punto superior del pie como se observa en la Figura 3.9.

Figura 3.9. Manipulador RR a Extremidad inferior de usuario.



Relación gráfica entre manipulado y extremidad, (Elaborado por: David Loja)

Mediante el algortimo de Denavit Hartenberg se obtuvo las ecuaciones que rigen al "manipulador" e implementas en el IDE de Arduino se tiene lo siquiente:

xi=l1_i*cos(a_ci)+l2_i*sin(a_ci+a_pi); Ec. (3.21)

yi=l1_i*sin(a_ci)+l2_i*cos(a_ci-a_pi); Ec. (3.22)

Las variables que almacenan las coordenadas en centímetros generadas por las ecuaciones de cinemática directa son "xi, yi" y las variables de los ángulos antes ya mencionados en radianes están descritas por "a_ci, a_pi". El mismo procedimiento se aplica a la pierna derecha.

3.2.4. Comunicación Serial Arduino Mega – NodeMCU

NodeMCU y Arduino Mega permite trabajar a velocidades de transferencia de 115200 baudios lo que asegura una rápida transferencia en el sistema. Para ello los datos del Arduino Mega fueron almacenados en una variable de tipo "string" como ejemplo se observa el código para la pierna izquierda:

```
String data;
data+=String(xi);
data+=",";
data+=String(yi);
data+="\n";
Serial.println(data);
delay(50);
```

La variable que almacena los datos a ser transmitidos es "data". Con "data+=" se agrega el dato en formato string con "String(xi)". Para generar una trama string en la que sea posible separar un dato de otro se agrega una "," entre cada dato para que posteriormente en la placa NodeMCU se pueda discriminar los datos. Por último, se envía la trama de datos colocando al final el delimitador o identificador de fin de trama "\n" y enviar con "Serial.println()". El tiempo de retardo para la ejecución del código es de 50 ms lo que permite al Arduino Mega tener una ejecución correcta de código.

En la placa NodeMCU se lee el puerto serial configurado con la misma velocidad de 115200 baudios del Arduino Mega y siempre que el puerto este habilitado como se observa:

```
while (Serial.available() > 0) {
float pix = Serial.parseFloat();
float piy = Serial.parseFloat();
float pdx = Serial.parseFloat();
float pdy = Serial.parseFloat();
```

Para retornar el estado del puerto serial se usa el comando “Serial.available()”. “Serial.parseFloat()” permite leer los datos secuencialmente como se encuentran ubicados en la trama de datos como se observa en la Figura 3.10.

Figura 3.10. Trama enviada por puerto serial.

Trama de datos, (Elaborado por: David Loja)

3.2.5. Configuración de NodeMCU

Para la configuración y manejo de la tarjeta NodeMCU es necesario agregar en el apartado de Preferencias del IDE de Arduino en la sección “Gestor de URLs Adicionales de Tarjetas” el siguiente link:

http://arduino.esp8266.com/versions/2.4.0/package_esp8266com_index.json

El cual contiene el repositorio donde se encuentra los paquetes de dispositivos que contienen el módulo ESP8266 y relacionados.

Luego en el apartado de “Herramientas>Gestor de Tarjetas” se escribe en el buscador

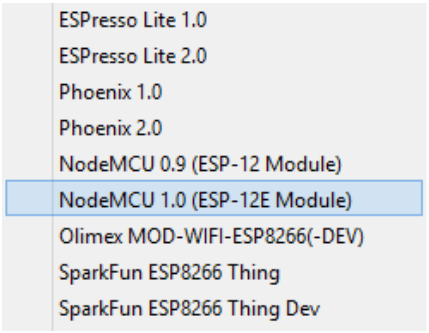
de dicha ventana: esp8266 como se muestra en la Figura 3.11 y se procede a instalar la librería.

Figura 3.11. Captura de ventana Gestor de tarjetas del IDE de Arduino.

Ventana de Instalación de librería esp8266, (Elaborado por: David Loja)

Una vez instalado se podrá verificar la instalación al tener nuevas tarjetas agregadas en el apartado “Herramientas>Placas:” donde se puede observar la placa “NodeMCU 1.0 (ESP-12E Module)”. Para cargar cualquier programa en la nueva tarjeta se realiza el mismo procedimiento como al cargar una tarjeta de Arduino normal.

Figura 3.12. Captura de ventana Gestor de tarjetas del IDE de Arduino.



Tarjetas disponibles con módulo esp8266, (Elaborado por: David Loja)

Dentro del código para el correcto funcionamiento de la tarjeta NodeMCU como primer aspecto se debe llamar a las siguientes librerías:

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

La librería empleada para el manejo del módulo ESP de la tarjeta NodeMCU es “ESP8266WiFi.h”. y para la utilización del protocolo UDP dentro del IDE de Arduino se dispone de la librería “WiFiUdp.h”.

La tarjeta NodeMCU entre sus funcionalidades nos permite colocar la red Wifi a la que nos vamos a conectar con su clave de acceso mediante el siguiente comando:

```
WiFi.begin(ssid, password);
```

Además, permite configurar una IP fija que será asignada a la tarjeta NodeMCU dentro de la red que en este caso es “192.168.1.20” y también su puerta de acceso y subred como se observa en el siguiente código:

```
IPAddress Ip(192,168,1,20);
IPAddress Gateway(192,168,1,1);
IPAddress Subnet(255,255,255,192);
WiFi.config(Ip, Gateway, Subnet);
```

La función “WiFi.config()” permite agregar las nuevas configuraciones a la tarjeta NodeMCU.

3.2.6. Envío de datos mediante protocolo UDP desde NodeMCU

Los datos son enviados mediante el protocolo UDP ya que permite el envío de los datos a través de una red sin necesidad de haber establecido conexión o sesión. Además, el protocolo no verifica si los paquetes fueron entregados únicamente los envía y debido a esto es muy adecuado para aplicaciones que se quiera enviar y recibir datos en tiempo real sin restricciones por verificación de paquetes entregados.

El programa implementado para el envío de datos mediante UDP es el siguiente:

```
String dato;
dato+= String(pix);
dato+= ",";
dato+= String(piy);
```

```
dato+= ",";
dato+= String(pdx);
dato+= ",";
dato+= String(pdy);
dato+= "\n";
Udp.beginPacket("192.168.1.6", 45678);
Udp.print(dato);
Udp.endPacket();
delay(50);
```

La trama por enviar debe ser compuesta como en el protocolo serial entre el Arduino Mega y la tarjeta NodeMCU mencionada anteriormente.

Cada dato debe estar separado por “,” y se debe terminar la trama con el identificador “\n”. Posteriormente con el comando “Udp.beginPacket()” apuntamos a la dirección IP a la que se va a enviar los datos acompañado de la puerta de enlace que en este caso es “45678” y la IP es “162.168.1.6”. Este último cambiará dependiendo en la red que se encuentre conectado.

Con la función “Udp.print()” se envía los datos mediante el protocolo UDP y posteriormente la terminación del paquete enviado con “Udp.endPacket()”.

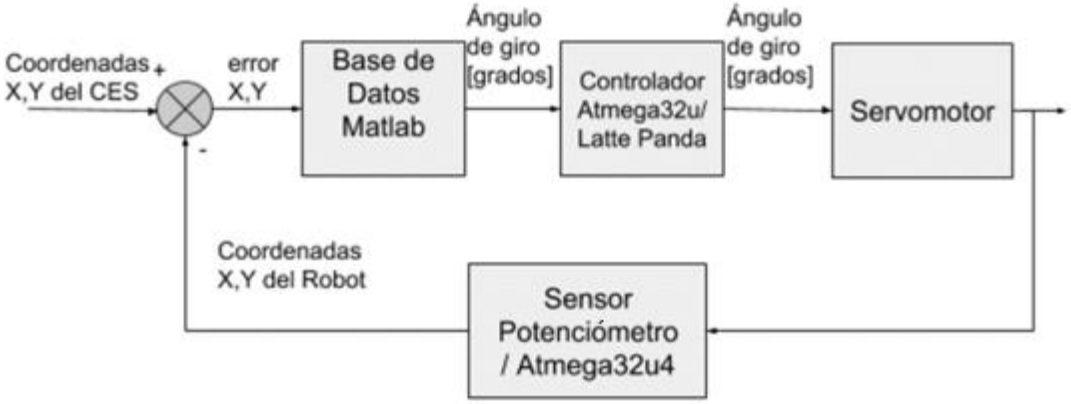
Además, se agrega un tiempo de espera de 50 ms para evitar conflictos en la comunicación y permitir una ejecución de código correcta.

3.3. Implementación del Sistema de Control de Robot Humanoide

El sistema de control está basado en el error obtenido en las muestras que toma el usuario con el CES y posteriormente son ejecutados en el robot humanoide.

En la Figura 3.13 se describe el lazo de control aplicado para cada articulación del robot humanoide, siguiendo asi el ángulo de giro al cuál debe colocarse el eje del servomotor.

Figura 3.13. Diagrama de control del robot humanoide.



Acción de control del robot humanoide, (Elaborado por: David Loja)

El Robot Humanoide mostrado en la Figura 2.8 es controlado por la tarjeta LattePanda la cual contiene un microcontrolador ATmega32u4 compatible con el IDE de Arduino.

La interfaz con la que se controla y adquiere los datos para el robot humanoide están creadas en una interfaz gráfica de Matlab 2015a ya que LattePanda al poseer como sistema operativo Windows 10 permite instalar programas como Matlab 2015a y Arduino.

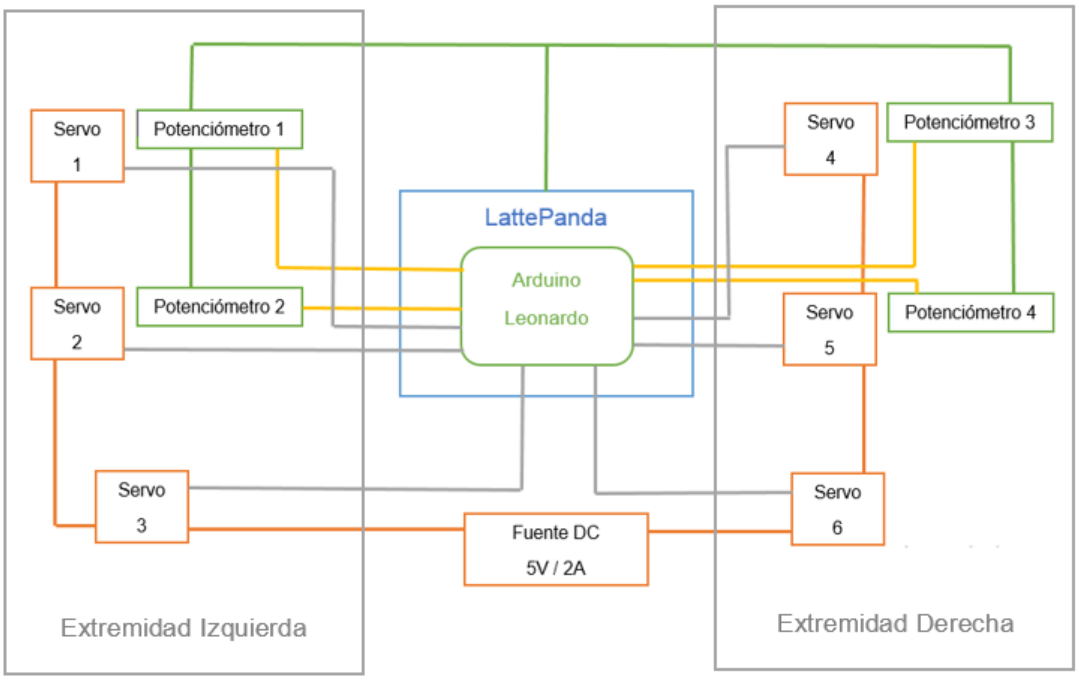
El robot humanoide consta de cuatro potenciómetros que al igual que en el CES medirán el ángulo al que se flexionan las articulaciones del robot humanoide.

Además, cuenta con seis servomotores marca Hitec 422 que permiten al robot accionar sus articulaciones y proporcionar su movimiento locomotor en las extremidades inferiores.

La conexión eléctrica para la alimentación del sistema se encuentra separada en la etapa de control diferenciada en color verde como se muestra en la Figura 3.14 y la etapa de potencia diferenciada de color anaranjado.

Esta última es alimentada con una fuente de 5 voltios en corriente continua a 2 A mientras que la etapa de control es alimentada por una fuente de 5 voltios a 2.1 A que esta conectada a la tarjeta LattePanda.

Figura 3.14. Diagrama de conexión de Robot Humanoide.



Esquema de conexión de robot Humanoide, (Elaborado por: David Loja)

La placa mediante reguladores de voltaje estabilizan el voltaje de entrada y proporciona alimentación al microcontrolador ATmega32u4 integrado y además tiene pines libres en la tarjeta que permiten alimentar los potenciómetros. Los pines de la tarjeta LattePanda se puede observar en la Figura 2.5.

3.3.1. Recepción de datos mediante el protocolo UDP en Matlab 2015a

Los datos son enviados desde el CES hasta el Robot Humanoide mediante el protocolo UDP como se indica en la sección 3.2.6. Estos son recibidos y almacenados en Matlab 2015a en tiempo real. Para validar la recepción se indica con que IP se está enviando los datos como se muestra:

```
UDPComIn=udp('192.168.1.20','LocalPort',45678);
set(UDPComIn,'DatagramTerminateMode','off');
```

La función que permite configurar el puerto UDP es “udp()”, teniendo así la IP de donde se envían los datos que en este caso es “192.168.1.20” dentro de la red local “LocalPort” ya que tanto como el CES y el Robot Humanoide se encuentran enlazados bajo la misma red mediante Wifi. Por último, se indica la puerta de acceso “45678”

que es determinada por el usuario permitiendo tener una red segura ya que cualquier otro dispositivo que quiera enviar datos a la IP de recepción serán rechazados ya que no cuentan con el código de la puerta de acceso. Con el comando “set()” colocamos como configuración default el modo de terminación de datagrama del protocolo UDP en “off”.

Dentro de un bucle determinado por el número de muestras a tomar que puede definir el usuario, se recepta los datos provenientes del CES como se indica:

```
fopen(UDPComIn);
data_udp=fscanf(UDPComIn);
sdata_udp=textscan(data_udp,'%f %f %f %f','Delimiter',',');
xi(i)=sdata_udp{1};
yi(i)=sdata_udp{2};
xd(i)=sdata_udp{3};
yd(i)=sdata_udp{4};
```

“fopen()” es la función que permite abrir el protocolo UDP antes ya configurado. “fscanf()” recepta los datos provenientes del protocolo y los almacena en “data_udp”. Para convertir los datos a texto guardándolos como un vector de tipo string se emplea “textscan()”.

El formato de tipo flotante en que fueron enviados los datos delimitados por “,” es “%f”. Con “sdata_udp{ }” se extrae los datos del vector string, de esta manera la posición 1 corresponde al desplazamiento en X de la pierna izquierda del CES, posición 2 corresponde al desplazamiento en Y de la pierna izquierda, posición 3 corresponde al desplazamiento en X de la pierna derecha y posición 4 corresponde al desplazamiento en Y de la pierna derecha que son almacenados en los vectores de tamaño igual al número de muestras “xi, yi, xd, yd” respectivamente.

En cada bucle es necesario cerrar el protocolo UDP para evitar conflictos de comunicación con el siguiente comando:

```
fclose(UDPComIn);
```

3.3.2. Cinemática Inversa en Robot Humanoide

Una vez obtenida las posiciones en el plano XY mediante la aplicación de las Ecuaciones de Cinemática Inversa aplicada a un manipulador RR mostrado en la Figura 3.8. se pueden obtener los ángulos a los que se debe colocar cada servo motor del Robot Humanoide para que pueda replicar los movimientos del usuario que son enviados mediante el CES.

Las ecuaciones que rigen al Robot Humanoide en su pierna izquierda son las siguientes:

```
theta2_i=acos((xi(i)^2+yi(i)^2-l1^2-l2^2)/(2*l1*l2));
```

Ec. (3.23)

```
theta1_i=asin(((yi(i)*(l1+l2*cos(theta2_i)))-
(xi(i)*l1*sin(theta2_i)))/(xi(i)^2+yi(i)^2));
```

Ec. (3.24)

Donde, “theta1_i, theta2_i” son los ángulos de los servomotores de la cadera y rodilla respectivamente de la rodilla izquierda. Las longitudes del manipulador como se muestra en la Figura 3.8 son “l1, l2”. Los vectores que almacenan los desplazamientos provenientes del protocolo UDP son “xi, yi”. Las funciones arco-coseno y arco-seno “acos(), asin()” respectivamente y “cos(), sin()” son las funciones coseno y seno dentro de Matlab 2015a.

Debido a que las operaciones trigonométricas devuelven valores en radianes se debe transformar a grados ya que en estos serán enviados mediante puerto serial al microcontrolador ATmega32u4 y dentro del IDE de Arduino, la librería para controlar los servos admite valores de ángulos en grados. La conversión de radianes a grados se muestra en el siguiente código:

```
the1_i=abs(theta1_i*180/pi);
the2_i=abs(theta2_i*180/pi);
```

La posición física de los servomotores en el robot humanoide tiene ligeros giros con respecto a su centro por lo que en base a prueba y error se determinó los grados a corregir para que los servo motores giren su eje al ángulo correcto. Teniendo así que

mover el eje de giro a - 5 grados del servo motor correspondiente a la cadera izquierda “the1_i” y +10 grados de la rodilla izquierda “the2_i”. Estos valores dependerán de cada manipulador y deben ser configurados por el usuario en base a observaciones.

```
if the1_i<=0
the1_i=0;
else
the1_i=the1_i-5;
end
if the2_i>=180
the2_i=180;
else
the2_i=the2_i+10;
end
the1_i=round(the1_i);
the2_i=round(the2_i);
```

La función “round” permite redondear los valores de decimales a enteros. Para los servomotores 3 y 6 representados en la Figura 3.14 correspondientes a los servos de los “tobillos” del robot humanoide “thep_1, thep_2”, se aplica un escalamiento para que tenga un movimiento equilibrado al moverse a partir del ángulo en cada rodilla “the2_d, the_i”, girando así desde su eje determinado en 135 grados. Al igual estos datos dependerán de cada manipulador. El mismo procedimiento se aplica para la pierna derecha.

```
thep_i=135+round(the2_d/7);
thep_d=135-round(the2_i/7);
```

3.3.3. Comunicación serial Matlab 2015a – microcontrolador ATmega32u4

El microcontrolador ATmega32u4 es programado mediante el IDE de Arduino ya que tiene cargado un subprograma que le permite ser compatible con esta interfaz de usuario. El IDE de arduino lo reconoce como un Arduino Leonardo conectado en el puerto serial COM2.

Al iniciar una lectura en un puerto serial en Matlab 2015a es recomendable eliminar cualquier otro llamado al puerto que se desea conectar, esto al iniciar y terminar el programa a ejecutar con el siguiente código:

```
delete(instrfind({'Port'},{'COM2'}));
```

Dentro de Matlab2015a se aplicó el siguiente código para configurar de manera correcta el puerto serial:

```
portName = 'COM2';
s = serial(portName,'BaudRate',115200,'Terminator','LF');
s.timeout = 0.5;
try
try
fopen(s);
catch
delete(s);
fopen(s);
end
catch
disp('Unable to open the port ');
end
```

“portName” contiene el puerto COM a conectar. Para configurar el puerto serial se usa el comando “serial()” que en este caso se emplea una velocidad de 115200 baudios. Para ejecutar el código de una manera más controlada se usa “try, catch” ya que si se produce un problema en la comunicación serial no lanzará un error que pueda detener la ejecución.

El envío de datos desde Matlab 2015a hacia el microcontrolador ATmega32u4 o Arduino Leonardo se realiza medianta el comando “fprintf” con el tipo de dato. Para

el ángulo el tipo de dato es flotante “%f” mientras que para el separador “,” es de tipo string “%s”.

```
fprintf (s, '%f', the1_i);
fprintf (s, '%s', ',');
fprintf (s, '%f', the2_i);
fprintf (s, '%s', ',');
fprintf (s, '%f', the1_d);
fprintf (s, '%s', ',');
fprintf (s, '%f', the2_d);
fprintf (s, '%s', ',');
fprintf (s, '%f', thep_i);
fprintf (s, '%s', ',');
fprintf (s, '%f', thep_d);
fprintf(s,'%s','\n');
```

La transmisión desde Matlab 2015a termina con el identificador “\n”.

Este identificador nos permite identificar en el código cargado en el Arduino Leonardo para que comience a transmitir los datos de los potenciómetros desde el Arduino a Matlab y al igual que en la sección 3.9.1 se utilizan las mismas funciones para la recepción de datos en Matlab2015a:

```
data_serial=fscanf(s);
sdata_serial=textscan(data_serial,'%f %f %f %f','Delimiter',',');
spi(i)=sdata_serial{1};
sci(i)=sdata_serial{2};
spd(i)=sdata_serial{3};
scd(i)=sdata_serial{4};
```

3.3.4. Base de datos en Matlab 2015a

Los datos de las muestras tomadas son guardados en los vectores “xi, yi, xd, yd” del tamaño de muestras a tomar definidas por el usuario. Estos datos son agregados como una matriz de vectores en la variable “sensores” como se indica en el siguiente código:

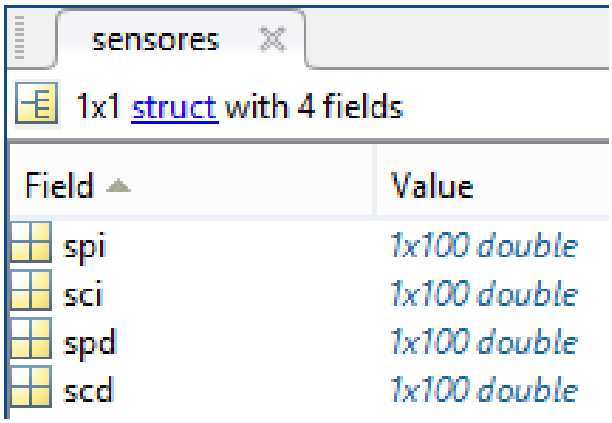
```
sensores.spi=xi;
sensores.sci=yi;
sensores.spd=xd;
sensores.scd=yd;
```

Posteriormente se guarda la matriz de vectores con el comando “save()” el cual contiene el nombre de la variable “sensores” como se indica en el siguiente código:

```
save(['sensores' num2str(cont) '.mat'],'sensores');
```

Mendiante la variable “cont” se almacena el número de muestras que vaya tomandose la cual va a ser mostrada para el nombre de archivo a guardar en la base de dato formando el vestor de tipo string “['sensores' num2str(cont) '.mat']”. Como ejemplo se muestra la base de datos formada en la Figura 3.15.

Figura 3.15. Base de Datos de Potenciómetros de Robot Humanoide.



Base de datos en Matlab 2015a, (Elaborado por: David Loja)

Para extraer los datos de las bases de datos utilizamos el comando “load()” que como entrada tiene el nombre de archivo de tipo “.mat”. El código para la lectura de la base de datos es el siguiente:

```
data_sens=load(nom);
v_xi=data_sens.sensores.spi;
v_yi=data_sens.sensores.sci;
v_xd=data_sens.sensores.spd;
v_yd=data_sens.sensores.scd;
```

La dirección para acceder al vector dentro de la base de datos sigue el siquiente formato: “data_sens.sensores.spi” siendo “data_sens” la base de datos llamada.

3.3.5. Cálculo de la muestra Óptima

Las muestras son comparadas en base al error en cada instante de las muestras, es decir, por cada posición de la muestra, es comparada con un set point o un valor establecido para de esta manera obtener la mejor muestra o la de menor error a comparar con la muestra establecida.

El código se aplica en un bucle repetitivo del tamaño del número de muestras que por defecto en este caso fue de 100 muestras y son almacenadas en una matriz que guarda tanto las muestras tomadas como los errores calculados con el siguiente código:

```
xizquierda(cont+i+1,j+2)={abs((abs(real_xi{1,1}-
actual_xi{1,1})/real_xi{1,1})*100)};
```

“real_xi” es el valor establecido como set point con el que será comparado el valor “actual_xi” de cada muestra, teniendo así el porcentaje de error de la muestra con el set point.

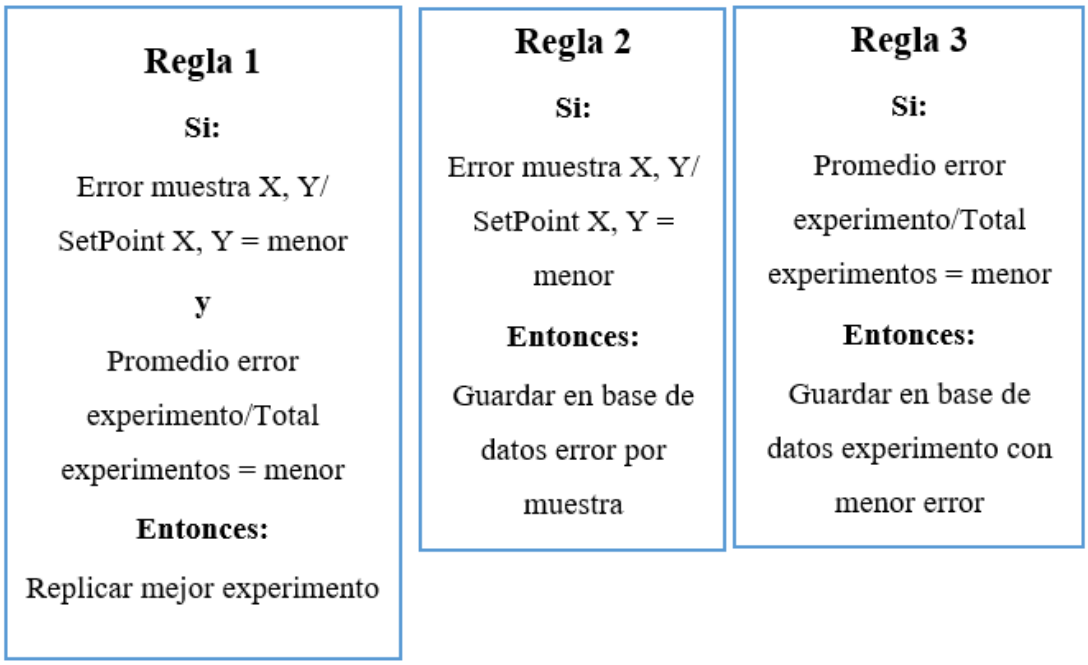
Todo el sistema esta basado en un experto humano que proporciona las muestras de las cuales el software se encarga de evaluar la mejor o la óptima ejecución de movimiento para replicar en el robot humanoide.

De esta manera es un sistema de aprendizaje basado en un sistema experto con base de datos ya que los datos son almacenados en tiempo real y el robot humanoide ejecutará el mejor movimiento en su ejecución.

3.3.5.1. Reglas Lógicas

Las reglas lógicasestán basadas en los siguientes hechos: coordenadas en el plano X, Y del humano experto, error entre los valores de X, Y muestreados y el SetPoint, muestra o experimento con menor error.

Figura 3.16. Reglas lógicas.



Reglas lógicas para obtener el mejor experimento, (Elaborado por: David Loja)

La Regla 2 indica que posteriormente a ejecutar el muestreo, estas son comparadas cada coordenada en X, Y del CES con las coordenadas X,Y del SetPoint de manera individual determinando así el error por muestra que es guardada en la base de datos para ser comparada posteriormente.

La Regla 3 indica el promedio calculado luego de tomar los errores almacenados por la Regla 2, para determinar el experimento que posea el menor error en muestreo para posteriormente ser almacenado en la base de datos.

La Regla 1 contiene las Reglas 2 y 3 las cuales permiten identificar la mejor muestra a ser replicada por el robot humanoide.

3.4. Interfáz gráfica para el control del sistema de aprendizaje

La interfaz gráfica permite manejar el sistema de una manera más sencilla para el usuario permitendo configurar aspectos como el número de muestras y las dimensiones de las extremidades inferiores.

En la ventana “muestras” se puede observar información acerca del muestreo realizado como el nombre del exprimento con el cual será guardado en la base de datos, el tiempo de captura de la muestra, capturas por segundo y el número de muestras a realizar como se observa en la Figura 3.17. Con el botón “Muestrear” se activa la captura de datos provenientes mediante UDP desde el CES hasta el robot humanoide, siendo estos datos las coordenada X, Y de cada pierna. Cuando es seleccionado “Capturar SP” se toman los datos de X, Y pero son guardados en la base de datos como los valores de set point para el sistema.

Figura 3.17. Ventana “muestras” de la interfaz de usuario.

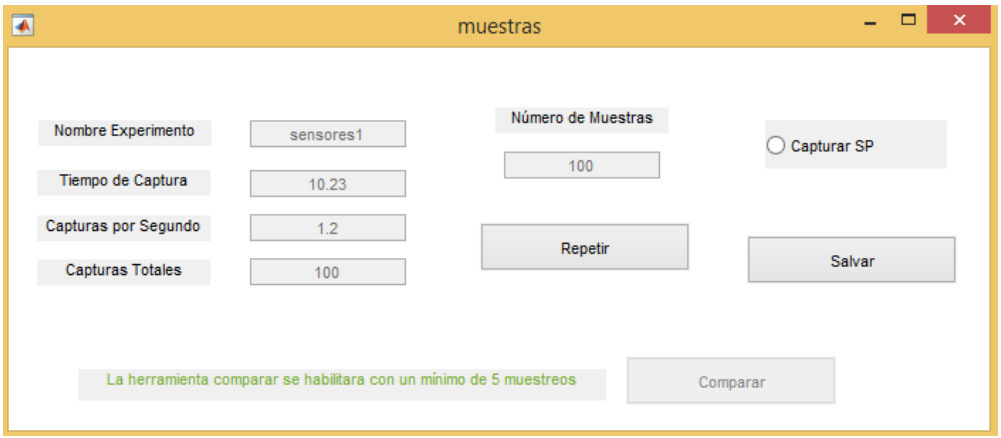
Ventana de la interfaz gráfica del sistema, (Elaborado por: David Loja)

La ventana “muestras” nos permite 3 funcionalidades como capturar los experimentos a comparar, capturar el set point con el que se van a comparar todos los experimentos y acceder a la ventana “calcular” una vez se haya completado un total de muestreo mayor a 5 experimentos.

Cabe recalcar que la ventana nos permite guardar los datos que se han muestreado con el boton “Salvar” como se muestra en la Figura 3.17 o a su vez repetir el experimento como se observa en la Figura 3.18 presionando el botón “Repetir”.

En la ventana “calcular” al presionar el boton “Cargar” se presentará de forma gráfica y en las tablas todos los experemientos realizados anteriormente lo que permitirá analizarlos. Además, se puede observar cada experimento de forma individual con el boton “Ver Detalles Muestras” en la cual despliega la ventana “sensores_poten”.

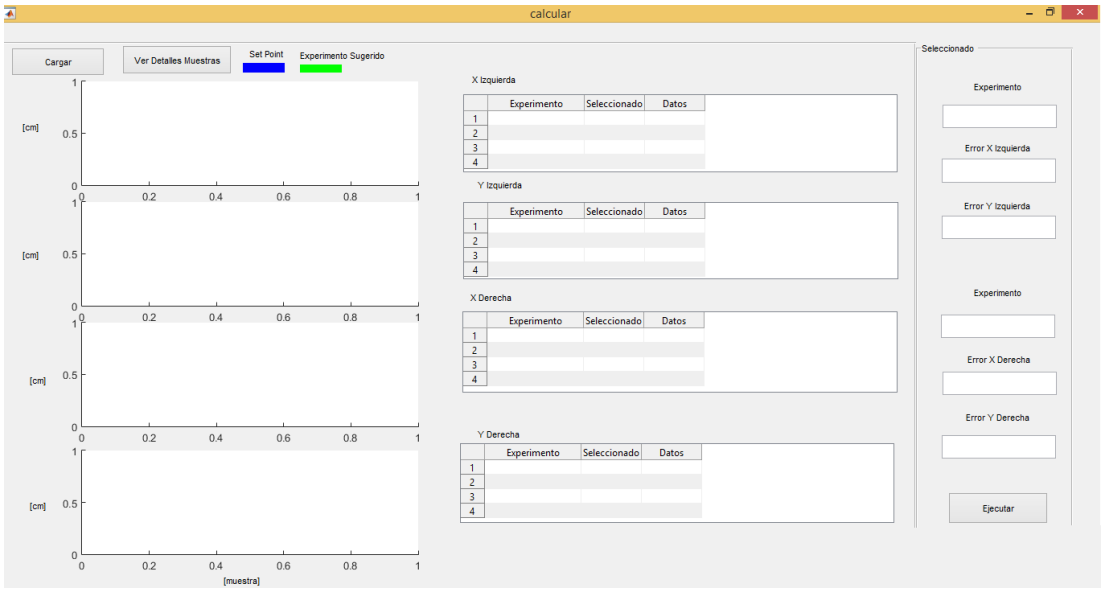
Figura 3.18. Ventana “muestras” de la interfaz de usuario, opción repetir.



Ventana “muestras” posterior a toma de datos, (Elaborado por: David Loja)

En las tablas de cada coordenada correspondiente a los valores de X, Y para cada pierna se muestra todos los valores numéricos de los experimentos como sus errores individuales calculados. En el apartado de seleccionado se encuentran los campos donde serán presentados los mejores experimentos en base a sus errores promedios para posteriormente ser replicados con el boton “Ejecutar”. La ventana se muestra en la Figura 3.19.

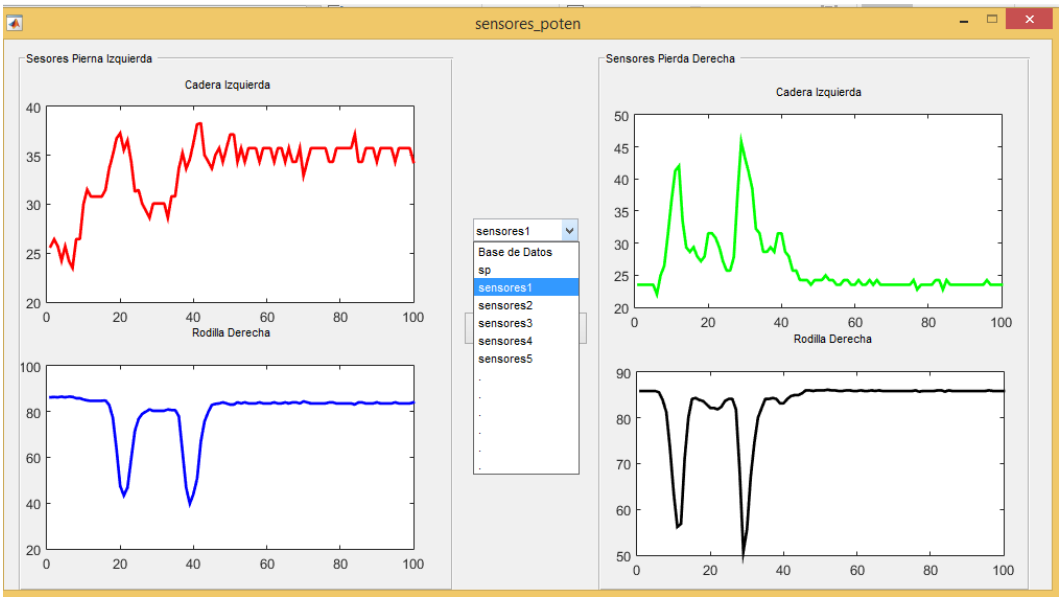
Figura 3.19. Ventana “calcular” de la interfaz de usuario.



Ventana para comparar experimentos, (Elaborado por: David Loja)

En la ventana “sensores_poten” se puede observar todos los experimentos de manera gráfica identificando los valores de cada muestra como se observa en la Figura 3.20.

Figura 3.20. Ventana “sensores_poten” de la interfaz de usuario.



Gráficas de experimento “sensores1”, (Elaborado por: David Loja)

CAPÍTULO 4
ANÁLISIS DE RESULTADOS

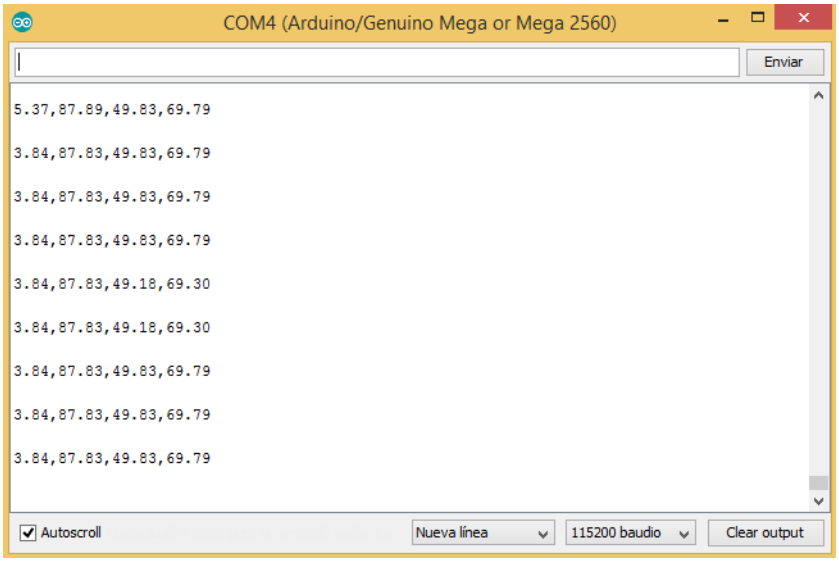
4.1. Generalidades

En este capítulo se presentarán las pruebas realizadas del sistema embebido de control CES y del robot humanoide, además de los resultados obtenidos en el mismo. Las pruebas fueron realizadas con un total de 5 experimentos como muestras y 1 set point para realizar las comparaciones respectivas.

4.2. Resultados obtenidos en el CES

Los datos son recogidos por el CES como se indican en la sección 3.8.3 obteniendo las coordenadas en de X, Y del sistema como se observa en la Figura 4.1.

Figura 4.1. Datos de X, Y del CES visualizados por puerto serial.



Datos de coordenadas visualizadas por puerto serial, (Elaborado por: David Loja)

Los datos presentados tienen un retardo de 50 milisegundos en la trasmisión, este valor fue establecido bajo observación ya que a un mayor retardo el movimiento replicado en el robot humanoide es lento y los datos son disparejos.

La velocidad de transmisión entre el Arduino Mega y la placa NodeMCU que conforman el CES influye en el tiempo de respuesta del Robot Humanoide al momento de replicar los movimientos. Los tiempos de respuesta a distintas velocidades de transmisión se muestra en la Tabla 4.1.

Tabla 4.1. Tiempo de respuesta entre CES y Robot Humanoide en base a la velocidad de transmisión.

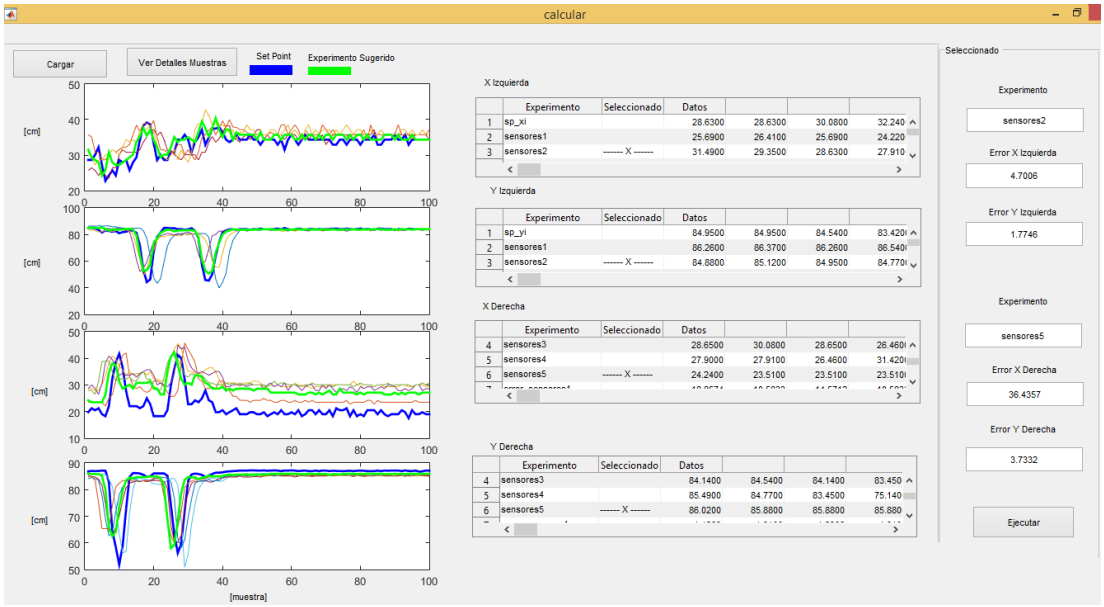
Velocidad de Transmisión Arduino Mega – NodeMCU [baudios]	**Tiempo de respuesta en ejecución [ms]**
9600	~1000
38400	~500
115200	50

Tiempos de respuesta obtenidos, (Elaborado por: David Loja)

4.3. Resultados obtenidos en Robot Humanoide

Los datos mostrados en la Figura 4.2 son el resultado de 5 experimentos con un total de 100 muestras por cada experimento. El movimiento realizado fue una caminata de dos pasos consecutivos mediante el CES.

Figura 4.2. Resultado experimentos de caminata.



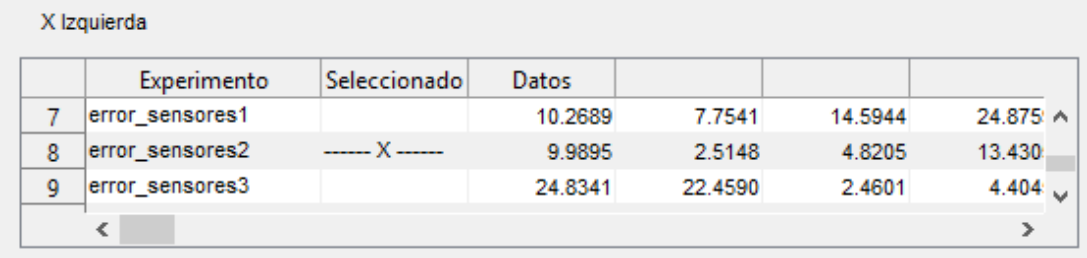
Ventana para la obtención del mejor experimento, (Elaborado por: David Loja)

El usuario por su naturaleza no puede replicar un movimiento igual a otro, como lo es el movimiento tomado como set point y los demás experimentos. Tratar de replicar el mismo movimiento varias veces conlleva ciertos errores como se ven reflejados en las

gráficas de la Figura 4.2 donde se observan de color azul el set point establecido y de color verde el experimento con menor error en comparación a los demás experimentos.

Los otros cuatro experimentos son representados con colores variados en las mismas gráficas. Además, en las tablas adyacentes se observa los datos individuales de cada experimento y los errores calculados para cada coordenada. En la Figura 4.3 se observa el ejemplo de los errores en X de la pierna izquierda.

Figura 4.3. Resultado errores individuales por muestra en X de pierna izquierda.



X Izquierda

	Experimento	Seleccionado	Datos			
7	error_sensores1		10.2689	7.7541	14.5944	24.875
8	error_sensores2	------ X ------	9.9895	2.5148	4.8205	13.430
9	error_sensores3		24.8341	22.4590	2.4601	4.404

Tabla de errores de X Izquierda, (Elaborado por: David Loja)

Como resultado del algoritmo para el cálculo del mejor experimento basado en su menos ejecución se obtuvieron los siguientes resultados presentados en la Tabla 4.2.

Tabla 4.2. Resultado experimento seleccionado por pierna.

Experimento Seleccionado	**Error en X [%]**	**Error en Y [%]**	**Pierna**
sensores2	4.7	1.77	Izquierda
sensores5	36.44	3.73	Derecha

Errores obtenidos en experimentos, (Elaborado por: David Loja)

El mayor error en el muestreo se da en X para la pierna derecha, esto se debe a que el usuario no pudo replicar, en el eje X, el mismo movimiento en comparación al set point; esto se puede observar en la gráfica correspondiente a “X Derecha” de la Figura 4.2 donde la muestra correspondiente a “sensores5” tiene diferencias notables en sus datos con respecto al set point. El mejor experimento dependerá de muchos factores como la calibración inicial, la ubicación de los sensores en el CES y los experimentos que haga el usuario.

Los resultados de reproducir el mejor experimento para cada pierna son satisfactorios debido a que se obtuvo un bajo error en ejecución como se muestra en la Tabla 4.3.

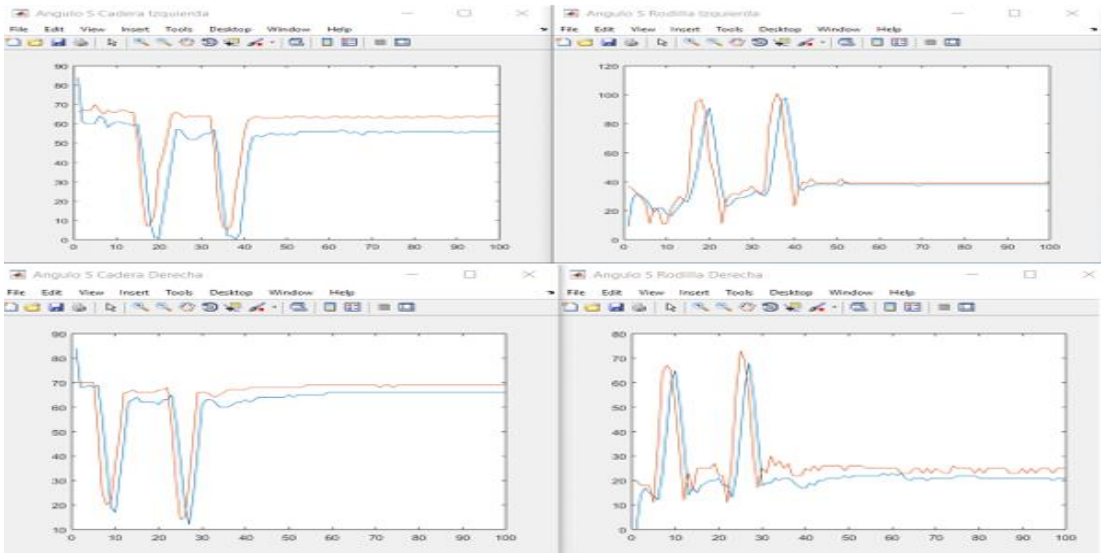
Tabla 4.3. Resultado experimento seleccionado en ejecución en base al ángulo de giro en cada articulación de Robot Humanoide.

Experimento Seleccionado	**Error en ángulo de cadera [%]**	**Error en ángulo de rodilla [%]**	**Pierna**
sensores2	0.26	0.18	Izquierda
sensores5	0.13	0.24	Derecha

Errores obtenidos en ejecución, (Elaborado por: David Loja)

Los errores obtenidos en la ejecución con respecto a las señales de los potenciómetros ubicados en las articulaciones de cadera y rodilla del robot humanoide son bajas debido a que el robot fue debidamente calibrado y ajustado los escalamientos de cada sensor, además se puede observar el resultado en la Figura 4.4 de forma gráfica, donde el valor ejecutado en color azul y la respuesta de los sensores en color anaranjado.

Figura 4.4. Respuesta de los sensores articulares del robot humanoide en la pierna izquierda.



Respuesta gráfica de sensores articulares, (Elaborado por: David Loja)

CONCLUSIONES

Para obtener las ecuaciones de cinemática inversa se usó únicamente las coordenadas X, Y ya que el robot utilizado tiene movimientos sólo en dichos ejes como se indica en la Figura 3.8. Identificando dos grados de libertad (cadera, rodilla). Las ecuaciones de la cinemática inversa indicado en la sección 3.3.2. limitan longitud máxima a la que puede llegar el punto final del manipulador debido a que uno de sus parámetros son las longitudes de cada extremidad.

Las coordenadas de los movimientos articulares se obtuvieron mediante el sistema embebido CES o sistema embebido de control, que proporciona los desplazamientos en los ejes X, Y que el usuario realice basándose en la reacción que tiene el sensor MPU6050 a la gravedad y a los movimientos que se realicen conjuntamente con los potenciómetros ubicados en cada articulación lo que permite la estabilidad en la obtención de las coordenadas.

Para comprobar el funcionamiento de la red se implementó un punto de acceso local provisto con tecnología inalámbrica Wifi a la cual se conectó el robot humanoide y el CES. La IP configurada al CES es 192.168.1.20 y la IP del robot es definida por el punto de acceso mediante DHCP. Todos los datos fueron transmitidos mediante el protocolo UDP logrando de esta manera que las acciones realizadas sean en tiempo real.

Para obtener el menor error en la ejecución del movimiento del robot humanoide se implementó un algoritmo el cual compara todas las muestras realizadas independientemente con el set point determinado por el usuario. Fundamentalmente está basado en la relación entre los valores establecidos y los obtenidos en tiempo real, obteniendo un error relativo en cada iteración. Para lo cual el algoritmo accede a cada base de datos y selecciona el experimento con menor error en base al promedio de todos sus errores a lo largo de las muestras. Los errores en ejecución obtenidos son menores al 1% como se muestra en la Tabla 4.2.

Para evaluar la comunicación y funcionamiento del algoritmo de aprendizaje se realizaron pruebas de comunicación con distintos movimientos articulares sin obtener

inconvenientes en relación a la estabilidad y fluidez de movimiento ya que la trasmisión de datos se realizó mediante el protocolo UDP acompañado de un identificador para validar la IP que va a enviar los datos. El algoritmo de aprendizaje va a depender mucho del usuario ya que debe replicar de la misma manera cada movimiento para que los errores en muestreo no resulten muy altos, para lo cual el algoritmo siempre selecciona al experimento con menor error sin presentar inconvenientes.

RECOMENDACIONES

Para evitar errores en la toma de datos por experimento, se debe calibrar de forma correcta el CES verificando el giro de cada sensor y la ubicación correcta de los sensores MPU6050 ya que el cálculo de los ángulos a aplicar dependes de dichos sensores.

La comunicación serial entre las tarjetas de desarrollo deben ser establecidas por preferencia en la velocidad de transmisión más alta permitida por los dispositivos ya que, al ser una aplicación en tiempo real, los tiempos de respuesta deben ser pequeños.

Las bases de datos en Matlab 2015a permite flexibilidad al momento de guardar la información necesaria por lo que se debe conocer y tener un control sobre las direcciones donde van a ser guardadas para facilitar el proceso de llamado a dichas bases de datos.

LISTA DE REFERENCIAS

Casserfelt, K., & Einestam, R. (2017). PiEye in the Wild : Exploring Eye Contact Detection for Small Inexpensive Hardware Karl Casserfelt.

Cernik, J. (2009). Framework for an Expert System Generator. *Transformation*.

DFRobot. (2017). Hitec HS422 Servo (SKU:SER0002), 2–4. Recuperado a partir de https://www.dfrobot.com/wiki/index.php/Hitec_HS422_Servo_(SKU:SER0002)

Escudero, I. H., Maria, L., & Suarez, B. (2016). Grau en Enginyeria en Tecnologies Industrials Diseño e implementación de un laboratorio para el estudio del envejecimiento en acelerómetros basado en LabVIEW Autor : Director : Escola Tècnica Superior d ' Enginyeria Industrial de Barcelona Resumen.

Fedorov, D. S., Ivoylov, A. Y., Zhmud, V. A., & Trubin, V. G. (2015). Using of Measuring System MPU6050 for the Determination of the Angular Velocities and Linear Accelerations, *1*(11), 517744.

Fukushima, K., Satoh, Y., Doi, M., & Ohtsuka, T. (2017). Optimization of Limb Reaching Movement for Climbing Humanoid Robots *, *2*, 1292–1297.

Golabchi, M. (2008). A knowledge-based expert system for selection of appropriate structural systems for large spans. *Asian journal of civil engineering*, *9*(2), 179–191. Recuperado a partir de http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.129.7272

Grosan, C., and A. A. (2011). *Intelligent Systems: A Modern Approach* (Vol. 220). Springer-Verlag, Berlin. Recuperado a partir de http://www.grundfos.com/grundfos-for-engineers/intelligent-systems.html#Featuredtheme2

Hun-ok Lim, A. T. (2010). 6 DOF Robot Arm Installation Diagram.

InvenSense Inc. (2013). MPU-6000 and MPU-6050 Product Specification, *1*(408).

Jin-Ling Lin, C.-W. H. (2017). Motion Replaying by Humanoid Robots Based on Semented and Stable Postures.

Kofinas, N. (2012). Forward and Inverse Kinematics for the NAO Humanoid Robot, (July), 1–78.

Lastra, J. L. M. (2013). Ahmed Rabee Sadik Design and Implementation of an Expert System for Monitoring and Management of Web-Based Industrial Applications Master of Science Thesis, (December).

Lattepanda, Studio, V. (2017). LattePanda (2G / 32GB / Without Win10 License) - The Most Powerful Win10 Dev Board.

Lee, K. (2014). ZMP Based Walking Pattern Generation of Child- sized Humanoid Robot : CHARLES (Cognitive Humanoid Autonomous Robot with Learning and Evolutionary System), 427–430.

Lucien Millette. (2012). Improving the knowledge-based expert system lifecycle. *Tesis Master*, 93.

Math Works Inc., & Guide, U. (2000). GARCH Toolbox For Use with M ATLAB.

Memisoglu, A. (2003). Human Motion Control Using Inverse Kinematics.

Rodríguez, R. (2017). Análisis Cinemático de un robot metamórfico tipo hexápodo, 12–16.

Schwartz, M., & Park, J. (2015). Arduino Guide using MPU-6050 and nRF24L01.

Sojib, N. (2017). Design and Development of the social Humanoid Robot named Ribo, 21–23.

Wallén, J. (2008). *On Kinematic Modelling and Iterative Learning Control of Industrial Robots*.

ANEXOS

Anexo 1. Recepción de datos mediante protocolo UDP.

```
UDPComIn=udp('192.168.1.20','LocalPort',45678);
set(UDPComIn,'DatagramTerminateMode','off');

delete(instrfind({'Port'},{'COM2'}));
portName = 'COM2';
s = serial(portName,'BaudRate',115200,'Terminator','LF');
s.timeout = 0.5;
try
    try
        fopen(s);
    catch
        delete(s);
        fopen(s);
    end
catch
    disp('Unable to open the port ');
end
```

Anexo 2. Lectura de datos mediante protocolo UDP.

```
fopen(UDPComIn);
data_udp=fscanf(UDPComIn);
sdata_udp=textscan(data_udp,'%f %f %f %f','Delimiter',',');

xi(i)=sdata_udp{1};
yi(i)=sdata_udp{2};
xd(i)=sdata_udp{3};
yd(i)=sdata_udp{4};
```

Anexo 3. Ecuaciones de cinemática inversa.

```
theta2_i=acos((xi(i)^2+yi(i)^2-l1^2-l2^2)/(2*l1*l2));
theta1_i=asin(((yi(i)*(l1+l2*cos(theta2_i)))-(xi(i)*l1*sin(theta2_i)))/(xi(i)^2+yi(i)^2));
```

Anexo 4. Lectura base de datos.

```
sp=load('sp.mat');
v_xi=sp.sensores.spi;
v_yi=sp.sensores.sci;
v_xd=sp.sensores.spd;
v_yd=sp.sensores.scd;
```

Anexo 5. Obtención error por cada experimento.

```
xizquierda(1,j+2)={sp.sensores.spi(1,j)};
yizquierda(1,j+2)={sp.sensores.sci(1,j)};
xderecha(1,j+2)={sp.sensores.spd(1,j)};
yderecha(1,j+2)={sp.sensores.scd(1,j)};

xizquierda(i+1,j+2)={v_xi(1,j)};
yizquierda(i+1,j+2)={v_yi(1,j)};
xderecha(i+1,j+2)={v_xd(1,j)};
yderecha(i+1,j+2)={v_yd(1,j)};

real_xi=xizquierda(1,j+2);
if real_xi{1,1}==0
    real_xi{1,1}=1;
end
actual_xi=xizquierda(i+1,j+2);
xizquierda(cont+i+1,j+2)={abs((abs(real_xi{1,1}-actual_xi{1,1})/real_xi{1,1})*100)};
e_xi(1,j)=abs((abs(real_xi{1,1}-actual_xi{1,1})/real_xi{1,1}));
prom_xi(i,1)=sum(e_xi);
```

Anexo 6. Obtención mejor experimento.

```
[min_xi,pos_xi]=min(prom_xi);
[min_yi,pos_yi]=min(prom_yi);
[min_xd,pos_xd]=min(prom_xd);
[min_yd,pos_yd]=min(prom_yd);

if pos_xi==pos_yi
    set(handles.edit1,'string',['sensores' num2str(pos_xi)]);
    set(handles.edit2,'string',num2str(min_xi));
    set(handles.edit3,'string',num2str(min_yi));
    selec_i=load(['sensores' num2str(pos_xi)]);
    plot(handles.axes1,selec_i.sensores.spi,'Color','g','LineWidth',2);
    plot(handles.axes2,selec_i.sensores.sci,'Color','g','LineWidth',2);
    xizquierda(pos_xi+1,2)={'------ X ------'};
    xizquierda(cont+pos_xi+1,2)={'------ X ------'};
    yizquierda(pos_xi+1,2)={'------ X ------'};
    yizquierda(cont+pos_xi+1,2)={'------ X ------'};
```

Anexo 7. Usuario con el CES vista lateral.

Anexo 8. Usuario con el CES vista frontal.

Anexo 9. Usuario con el CES flexionando rodilla.

Anexo 10. Robot Humanoide vista lateral.

Anexo 11. Robot Humanoide vista frontal.

Anexo 12. Robot Humanoide flexionando rodilla.