Reporte proyecto final

Sebastian Barrio Bejarano - A01753734
Abraham Trejo Arvizu - A01754195
Enrique Fest Flores - A01754194
Gabriela Marina Sánchez Ravelo - A01747124

15 de Marzo del 2024

Diseño y desarrollo de robots MR 3001 B.102
Escuela de Ingeniería y Ciencias
Departamento de Mecatrónica

Investigación Previa

El sistema mecatrónico a desarrollar es un brazo robótico con una pinza, el cual debe priorizar la eficiencia y velocidad en el traslado de 10 dados, completamente iguales. El desarrollo de este sistema tendrá limitaciones con los materiales, los cuales fueron especificados por el socio formador. Sin embargo, los pormenores del proyecto son a determinar por nosotros. Para ello daremos una revisión a la evolución en el área y observar qué es lo que necesita nuestro proyecto.

El primer modelo de brazo robótico fue un acercamiento analógico dado por da Vinci, el cual estaba potenciado por la fuerza humana, después de ellos los avances no se materializarían hasta el siglo XX, donde el primer brazo robótico potenciado con motor nacería por un estudiante británico, Bill Taylor, y años después la industria tomaría manos en la obra y utilizará los brazos para tareas repetitivas y pesadas, pero en los años consiguientes esta tecnología necesitaba de resultados precisos y programables para múltiples situaciones, así que se empezó a utilizar técnicas de cinemática en ellos, con el objetivo de que cada empresa pudiera obtener una de estas soluciones y aplicarlas a problemas específicos sin la necesidad de diseñar un producto único.

Un robot con tales funcionalidades, deberá de contar con diferentes aspectos Ingenieriles, estos serán, la aplicación de procesamiento de imágenes, para la segmentación de objetos cúbicos, Comunicación vía internet, para el control de un microprocesador, Modelación 3D sobre un brazo robótico, el estudio de su cinemática directa e inversa y la aplicación de la electrónica; Este documento se hablará de dos de ellas, Modelación 3D y cinemática directa e inversa.

Cinemática directa e inversa.

El estudio de la cinemática de manipuladores robóticos, está dividido en dos áreas, la cinemática directa y la cinemática inversa. La cinemática directa calculará la posición de la pinza basándose en los ángulos creados en las articulaciones del brazo, mientras que la cinemática inversa permitirá determinar los ángulos necesarios en las articulaciones para alcanzar posiciones específicas. Esto nos permitirá mover el brazo robótico a cualquier posición deseada usando un mapa de coordenadas, junto a la implementación de una cámara para determinar dónde se encuentran los cubos en nuestro espacio.

La cinemática directa pide determinar un marco de referencia para cada una de las uniones de nuestro robot, las cuales pueden ser rotacionales o lineales, ya definido ellos, debemos de obtener las diferentes medidas de nuestros eslabones fijos y variables. Terminado ello, deberemos de aplicar nuestros resultados a una matriz de rotación y traslación, usando multiplicación de matrices, iremos desde cada eslabón a cada articulación multiplicando las matrices entre ellas en orden, y así obtendremos la matriz homogénea de nuestro robot.

Por otra parte, para resolver la cinemática directa se puede aplicar la convención de Denavit-Hartenberg, la cual utiliza matrices de transformación homogénea como una parte integral de su metodología. La principal diferencia entre este método y el de matrices de transformación homogénea es que en la convención D-H se asignan sistemas de referencia a cada uno de los eslabones y articulaciones, definiendo parámetros específicos para describir las relaciones espaciales entre ellos. Entre estos parámetros se incluyen longitudes, ángulos y desplazamientos, y son esenciales para caracterizar la configuración geométrica del robot.

A partir de estos parámetros se construyen matrices que describen la relación espacial entre cada uno de los sistemas de referencia, que se encuentran en los eslabones.

Existe una serie de pasos que se deben seguir para obtener los parámetros, los cuales son:

1. Numerar eslabones y articulaciones

- Los eslabones se numeran desde 0 hasta n.
- Articulaciones se numeran desde 1 hasta n.

2. Localizar ejes de articulaciones
    - Para revoluciones, el eje es el de giro.
    - Para prismáticos, el eje es el de desplazamiento.
3. Colocar ejes Z
    - Z_i se coloca sobre el eje de la i-ésima articulación.
4. Colocar sistema de coordenadas 0
    - Se coloca el origen en algún punto a lo largo de Z_0.
5. Colocar resto de sistemas de coordenadas
    - Se colocan en las intersecciones de Z_i y la normal común a Z_i y Z_i+1.
6. Colocar ejes X
    - Cada X_i va en la dirección de la normal común a Z_i−1 y Z_i.
7. Colocar ejes Y
    - Determinados por XYZ dextrógiro
8. Sistema del extremo del robot
    - El sistema XYZ se coloca en el extremo del robot.
9. Ángulos θ
    - Cada θ_i es el ángulo de rotación alrededor de Z_i.
10. Distancias *d*
    - Cada d_i es la distancia a lo largo de Z_i−1 desde XYZ_i−1 hasta la intersección de las normales comunes.
11. Distancias *a*
    - Cada a_i es la longitud de la normal común.

12. Ángulos α
    - Ángulo de rotación alrededor de X_i para llegar de Z_{i-1} a Z_i.
13. Matrices de transformación de cada eslabón individuales
    - Cada eslabón define una matriz de transformación utilizando los parámetros θ_i, d_i, a_i, y α_i.
14. Matriz de transformación final
    - La matriz de transformación total que relaciona la base del robot con su herramienta es la encadenación (multiplicación) de todas esas matrices.

Para resolver el problema de cinemática inversa, se puede realizar a partir de diferentes métodos.

Método geométrico

El primer método es el geométrico se basa en relaciones geométricas y trigonométricas entre las articulaciones y el extremo del robot. A partir de las ecuaciones obtenidas en la cinemática directa, se invierten para expresar las variables de articulación en función de la posición y orientación deseadas del extremo y se resuelven las ecuaciones, utilizando relaciones geométricas. Es importante mencionar que este método se limita a usarse en robots con pocos grados de libertad.

Método numérico

Se utilizan métodos numéricos para que el error entre la posición del efector final de referencia y la posición actual sea lo más mínimo posible, por lo que se consigue una aproximación. La inversa de la Jacobiana se puede utilizar para ésto, teniendo como desventaja que la inversa tiende a provocar saltos grandes haciendo el método numéricamente inestable, para que el método se pueda aplicar, la matriz jacobiana debe ser cuadrada. Para resolver los problemas de la inversa de la Jacobiana, se puede hacer un cálculo utilizando la Pseudo-Inversa Moore - Penrose, éste cálculo es utilizado cuando la jacobiana no sea cuadrada y tenga más columnas que filas.

Un método alternativo sería el método de mínimos cuadrados amortiguados (o regularizados), en ésta, se utiliza un λ, que es un término de regularización, éste permite evitar la singularidad de la matriz jacobiana, cuando haya singularidad, la matriz será invertible y sea estable.

Método de matrices inversas

Otro método para obtener la cinemática inversa, es el uso de matrices de transformación, el procedimiento nos pide que debamos de igualar nuestras matrices homogéneas a una matriz "normal, orientación, aproximación, posición", dada la igualdad con álgebra eliminaremos una de las matrices con su inversa en ambos lados de la ecuación, dejando nuestra multiplicación de matrices con un inciso menos y en el lado de la matriz "n,o,a,p" junto a la inversa de la matriz eliminada, multiplicaremos estas matrices y las igualamos a nuestras matrices homogéneas, esto con la idea de despejar nuestras incógnitas y variables para obtener cómo se definirán las rotaciones

Método por desacoplo cinemático

Éste método es aplicable en robots con más de 3 grados de libertad, el objetivo va a ser encontrar el valor de las variables articulares que permiten que el robot alcance una posición y orientación específica en su efector final.

El método se basa en dividir en dos subproblemas, el posicionamiento y la orientación.

El subproblema de posicionamiento se resuelve utilizando técnicas geométricas o algebráicas para determinar las variables articulares que ubican la muñeca en la posición deseada, las coordenadas articulares q_4, q_5, $y\ q_6$, se pueden calcular únicamente con la información de la rotación deseada ([noa]),

La orientación (q_1, $q_2\ y\ q_3$) se obtiene de la información de la posición deseada (p_x, $p_y\ y\ p_z$), utilizando las mismas técnicas de posicionamiento, ésta etapa se realiza una vez se haya fijado la posición del efector final.

Planeación de trayectorias

Una vez que se han obtenido los modelos cinemáticos y dinámicos del robot, se aborda el desafío del control de los mismos. Otra área que desempeña un papel importante en la robótica es la planeación de trayectorias, la cual estudia la forma en la un sistema puede pasar de un estado inicial a un estado final, en un espacio dado. El objetivo es establecer las trayectorias que cada articulación del robot debe seguir a lo largo del tiempo para alcanzar los objetivos establecidos, cumpliendo simultáneamente una serie de restricciones físicas impuestas por los actuadores y de calidad de la trayectoria, tales como suavidad y precisión. Este proceso es esencial para garantizar la eficiencia en la ejecución de tareas, minimizando el tiempo y los recursos requeridos, mientras se sortean obstáculos y se respetan las limitaciones físicas del robot. Para implementar esto, se puede usar el método de interpolación polinomial. En primer lugar se presenta un esquema general del problema de la planificación, y se distingue entre el espacio cartesiano y el espacio de las articulaciones del robot. Después de esto, es necesario obtener un polinomio en coordenadas cartesianas que describa la trayectoria deseada en función del tiempo, asegurándose de que esté libre de colisiones. Posteriormente, se procede a muestrear esta trayectoria en una serie finita de puntos de control, que se utilizarán como puntos inicial y final de cada segmento de trayectoria. Cada uno de estos puntos de control se especifica en coordenadas cartesianas de posición y orientación. Es importante conocer las diferencias entre el espacio cartesiano y el artículo para poder definirlos. El espacio cartesiano describe la posición y orientación de un robot en coordenadas tridimensionales (x, y, z), basado en puntos fijos en el entorno. En cambio, el espacio articular utiliza las posiciones angulares de las articulaciones del robot para definir su estado. Mientras que el espacio cartesiano es más intuitivo para los humanos, el espacio articular ofrece un control directo sobre las articulaciones del robot. La elección entre ambos depende de las necesidades específicas de planificación y control del movimiento del robot.

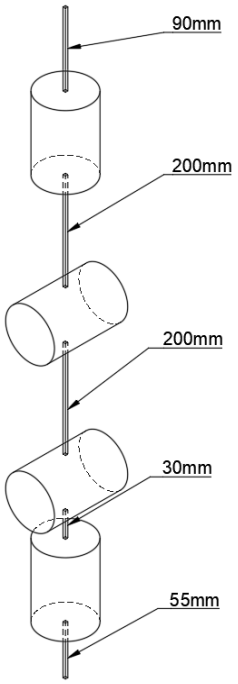
Visión computacional

La visión computacional es un campo de investigación que busca que las máquinas sean capaces de comprender el contenido presente en imágenes digitales y videos. Su objetivo es automatizar tareas realizadas por el sistema visual humano. En el contexto del desarrollo de un robot, la visión computacional desempeña un papel crucial al permitir que el robot perciba su entorno y tome decisiones basadas en la información visual capturada. Para lograr esto, se utilizan técnicas como el procesamiento de imágenes, la detección de objetos, la segmentación y la reconstrucción 3D. Además, la visión computacional se combina con otros aspectos de la robótica, como la planificación de trayectorias y el control cinemático, para crear sistemas robóticos más inteligentes y autónomos.

Descripción del sistema

Diseño mecánico

En este reto se nos permitió utilizar un total de 5 servomotores, 3 de ellos siendo del modelo mg995 y los 2 restantes modelo sg90, utilizando un servo sg90 para abrir y cerrar el efector final, nos quedan 4 actuadores para el resto del robot, lo que significa que podemos tener hasta 4 grados de libertad. Para lograr un robot que pudiera posicionar el efector en cualquier lugar de su zona de trabajo y rotarlo en el eje de Roll del efector final se utilizó una configuración hombro codo muñeca como la que se detalla a continuación.



El robot cuenta con 4 articulaciones revolutas:

La base (θ1) rota en el eje Z y está elevada 55mm, tiene una distancia desde el eje de rotación hasta el siguiente actuador de 30 mm.

El hombro (θ2) rota en el eje Y, se encuentra a 30 mm de la base

El codo (θ3) se encuentra a 200 mm del hombro, al final de lo que sería el primer eslabón, o el brazo, y también rota en el eje Y, en la misma orientación que el hombro.

Por último, **la muñeca (θ4)** se encuentra a 200 mm del codo, estando al final del antebrazo, o eslabón 2 y este rota en el eje Z, permitiendo modificar el ángulo Roll del efector final.

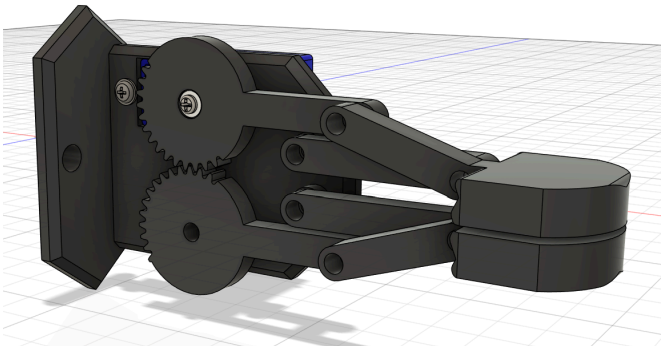


FIGURA 2. Efector final del robot

El efector final (Fig.2) es un gripper común que utiliza engranajes para transmitir el movimiento rotacional del servo a un movimiento de abrir y cerrar en las pinzas del gripper, el lugar donde las pinzas se encuentran cerradas se encuentra a 90 mm de la articulación de la muñeca.

Diseño (CAD)

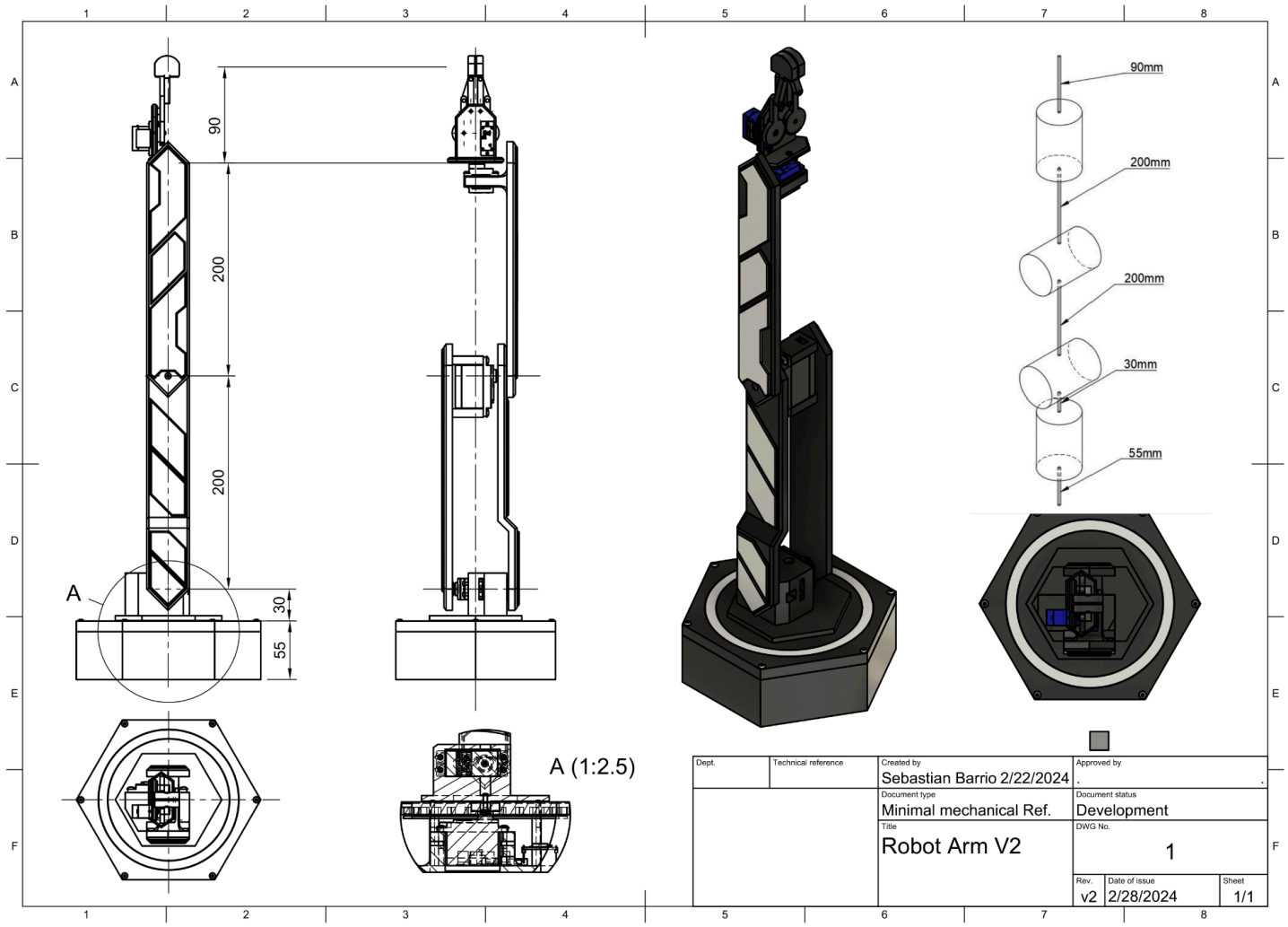


FIGURA 3. Dibujo técnico del cad, vista frontal, lateral, superior e isométrica, sección interna del servo de la base (A) y configuración cinemática (link al cad)

Las especificaciones descritas se implementaron en el diseño del robot, contando con una base hexagonal donde se encuentran los microcontroladores y circuitos necesarios para la operación del robot y el servomotor de la base. Todas las uniones cuentan con un rodamiento para disminuir fricción entre partes y evitar que los eslabones se trasladen de manera no deseada o se generen

esfuerzos de más en los motores, se usan también tornillos M2, M3 y M4 para asegurar todas las partes que lo requieren. El material de construcción es PLA como material de impresión en 3D.

Cinemática directa

Para conseguir la cinemática directa del robot, se utilizaron las medidas y la configuración descritas anteriormente en la Figura 1, y se utilizó el método de las matrices de transformación homogéneas por su facilidad de uso y la simplicidad del modelo de robot que propusimos. Para nuestro robot definimos primero los 4 “eslabones” cada uno con su traslacion y rotacion correspondiente al actuador al que pertenecen, siendo entonces:

$$^{0}T_{1} = Trans(0,0,a_1) * Rot(z,\Theta_1) * Trans(0,0,a_2)$$

$$^{1}T_{2} = Rot(y,\Theta_2) * Trans(0,0,a_3)$$

$$^{2}T_{3} = Rot(y,\Theta_3) * Trans(0,0,a_4)$$

$$^{3}T_{4} = Rot(z,\Theta_4) * Trans(0,0,a_5)$$

Donde el sistema de referencia 0 se encuentra en el punto más bajo del robot, teniendo en cuenta incluso la caja de electrónicos, y el sistema de referencia número 4 siendo el sistema de referencia final, que se encuentra apuntando en el mismo eje que el effector y que el eje Z universal, por lo que no hace falta ninguna rotación del sistema, esto nos da como matriz final de transformación:

$$T_F = {}^{0}T_4 = {}^{0}T_1 * {}^{1}T_2 * {}^{2}T_3 * {}^{3}T_4$$

Que al sustituir por las matrices correspondientes a las transformaciones, los valores de a1-a5 y hacer las multiplicaciones necesarias nos queda (el código utilizado se encuentra en el anexo):

$$T_F = \begin{bmatrix} -S(\theta_1)S(\theta_4) - C(\theta_4)(C(\theta_1)S(\theta_2)S(\theta_3) - C(\theta_1)C(\theta_2)C(\theta_3)) & S(\theta_4)(C(\theta_1)S(\theta_2)S(\theta_3) - C(\theta_1)C(\theta_2)C(\theta_3)) - C(\theta_4)S(\theta_1) & S(\theta_2+\theta_3)C(\theta_1) & 10C(\theta_1)(29S(\theta_2+\theta_3) + 20S(\theta_2)) \\ C(\theta_1)S(\theta_4) - C(\theta_4)(S(\theta_1)S(\theta_2)S(\theta_3) - C(\theta_2)C(\theta_3)S(\theta_1)) & C(\theta_1)C(\theta_4) + S(\theta_4)(S(\theta_1)S(\theta_2)S(\theta_3) - C(\theta_2)C(\theta_3)S(\theta_1)) & S(\theta_2+\theta_3)S(\theta_1) & 10S(\theta_1)(29S(\theta_2+\theta_3) + 20S(\theta_2)) \\ -S(\theta_2+\theta_3)C(\theta_4) & S(\theta_2+\theta_3)S(\theta_4) & C(\theta_2+\theta_3) & 290C(\theta_2+\theta_3) + 200C(\theta_2) + 85 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

FIGURA 4. Matriz de transformación final, obtenida con MATLAB.

Con esta matriz encontrada podemos encontrar el punto en el espacio en el que se encuentra el efector final si le damos los ángulos t1 a t4, que al comprobar con el modelo 3D y los ángulos igualados a 0 podemos observar que son correctos.

```
TFNum =

     1     0     0     0
     0     1     0     0
     0     0     1   575
     0     0     0     1
```

```
TF = simplify(T1*T2*T3*T4)

a1=55;
a2=30;
a3=200;
a4=200;
a5=90;
t1 = 0;
t2 = 0;
t3 = 0;
t4 = 0;

TFNum = eval(subs(TF)) %Matriz de transformacion
```

FIGURA 5. Comprobación de posición del efector final en MATLAB..

Cinemática inversa

Una vez que encontramos la cinemática directa ahora nos disponemos a encontrar la cinemática inversa del sistema, para esto decidimos utilizar el método geométrico por su simplicidad a la hora de calcularlo y el hecho de que llegamos a una solucion analitica y simbólica que luego podemos implementar en el controlador una vez construido el robot, para esto primero definimos 2 planos, uno superior y uno lateral para poder resolver el problema por partes, para el plano superior xy proponemos lo siguiente para encontrar theta 1.

$$\frac{y}{x} = tan\Theta_1$$

$$\Theta_1 = tan^{-1}(\frac{y}{x})$$

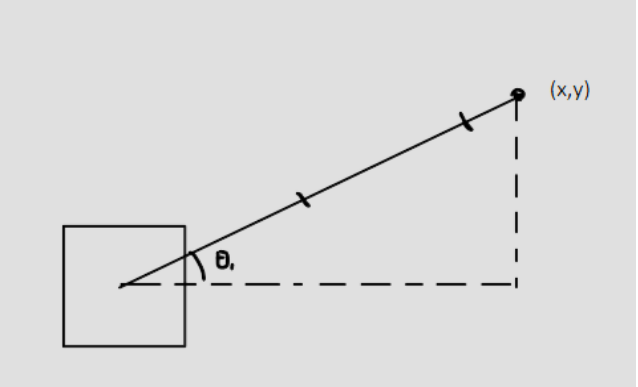


FIGURA 6. Plano superior XY, con la incógnita θ_1.

Para encontrar theta 2 y theta 3 se utilizaron unos ángulos auxiliares phi1-phi3 y se utilizó la ley de cosenos para resolver para los ángulos theta2 y theta 3 en configuración de codo superior, como se muestra en la figura siguiente

$$L = \sqrt{(x^2 + z^2)}$$

$$\phi_2 = cos^{-1}\left(\frac{a_3^2 + (a_4 + a_5)^2 - L^2}{2(a_3)(L)}\right)$$

$$\phi_1 = cos^{-1}\left(\frac{a_3^2 + L^2 - (a_4 + a_5)^2}{2(a_3)(L)}\right)$$

$$\phi_3 = tan^{-1}(z/x)$$

$$\Theta_2 = \phi_3 + \phi_1$$

$$\Theta_3 = \phi_2 - \pi$$

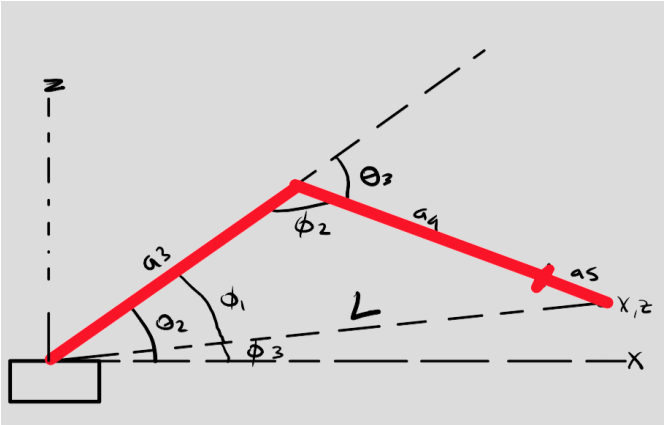


FIGURA 7. Plano ZX con 2 eslabones representados.

Y reemplazando obtenemos las siguientes ecuaciones, que junto con la ecuación encontrada en el plano superior nos permiten encontrar theta 2 y theta 3 en términos de las entradas x, y z

$$\Theta_1 = tan^{-1}\left(\frac{y}{x}\right)$$

$$\Theta_2 = tan^{-1}\left(\frac{2}{x}\right) + cos^{-1}\left(\frac{a_3^2 + (x^2 + z^2) - (a_4 + a_5)^2}{2(a_3)(\sqrt{(x^2 + z^2)})}\right)$$

$$\Theta_3 = cos^{-1}\left(\frac{a_3^2 + (a_4 + a_5)^2 - \sqrt{(x^2 + z^2)^2}}{2(a_3)(a_4 + a_5)}\right)$$

Una vez realizado esto podemos poner estas ecuaciones en matlab y de nuevo corroborar con el modelo 3D y podemos ver que son correctas.

```
%CINEMATICA INVERSA (por metodo geometrico)(resultado en radianes)
syms px py pz
px = 100;
py = 100;
pz = 100;
%ecuaciones de cinematica inversa
theta1 = atan(py/px);
theta2 = atan(pz/px) + acos((a3^2+(px^2+pz^2)-(a4+a5)^2)/(2*a3*(sqrt(px^2+pz^2))));
theta3 = acos((a3^2+(a4+a5)^2-(sqrt(px^2+pz^2))^2)/(2*a3*(a4+a5))) - pi;

%angulos conseguidos para un px py pz
rad2deg(theta1)
rad2deg(theta2)
rad2deg(theta3)
```

```
ans =

    45


ans =

  160.2160


ans =

 -153.8202
```

FIGURA 8. Código cinemática inversa en MATLAB.

Para este modelo de cinemática inversa no se calculo la theta 4, ya que como se puede apreciar en el diagrama y en la cinemática directa, la rotación de theta 4 no afecta la posición del efector final, solo la rotación del Roll del mismo.

Una vez conseguidas las expresiones para las articulaciones se hizo un último cambio para poder implementarlas, específicamente las theta2 y theta3 se modificaron para utilizar arcotangentes en lugar de arcosenos, ya que al utilizar senos los valores de las articulaciones se indefinen por que los valores de coseno de un valor positivo y el mismo valor en negativo son iguales, mientras que al usar arcotangente, específicamente ATAN2, se mantiene la información de los valores originales al hacer el cálculo final, al hacer esto obtenemos las ecuaciones finales para la implementación:

$$cos(\Theta_3) = \frac{(x^2 + y^2 + z^2 - a_3^2 - (a_4 + a_5)^2)}{(2a_3(a_4 + a_5))}$$

$$sen(\Theta_3) = -\sqrt{(1 - cos(\Theta_3^2))}$$

$$\Theta_1 = ATAN2(y, x)$$

$$\Theta_2 = ATAN2(z, \sqrt{x^2 + y^2}) - ATAN2((a_4 + a_5)sen(\Theta_3), a_3 + (a_4 + a_5)cos(\Theta_3))$$

$$\Theta_3 = -(ATAN2(sen(\Theta_3), cos(\Theta_3)))$$

Espacio de trabajo

El espacio de trabajo se define por el conjunto de puntos que puede alcanzar el robot alrededor de sí mismo y sobre el cual va a realizar las tareas establecidas. Es por esto que se realizó un plano, sobre el cual se situó el robot, y todos los elementos necesarios, como las canastas y el soporte para la cámara, además del punto de referencia para la programación de visión computacional, y de esta forma tener mucho mayor control en las posiciones de cada uno de los elementos y delimitar los movimientos del robot, para únicamente usar el espació necesario para realizar la tarea.

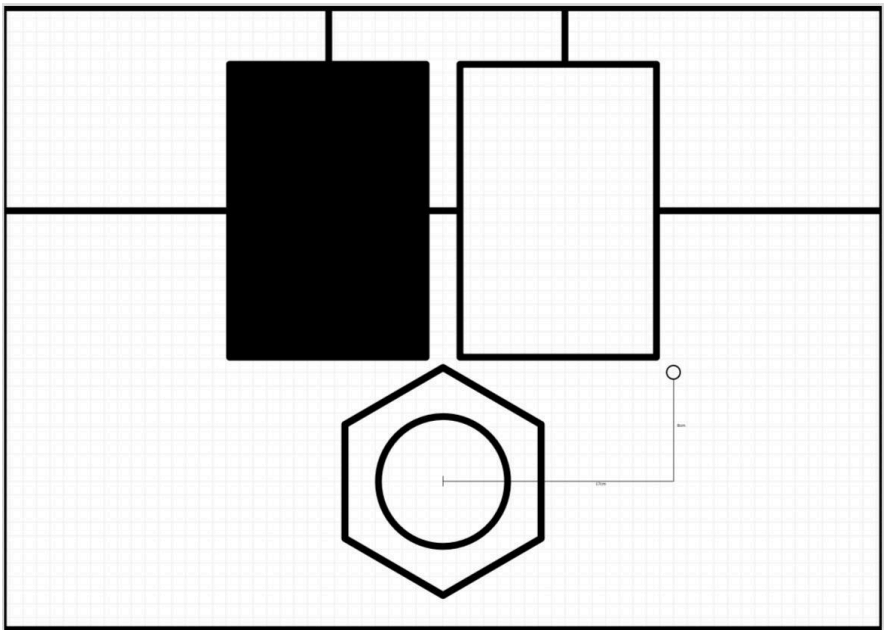


Figura 8. Plano del espacio de trabajo del robot.

A pesar de que el robot, realmente puede alcanzar otros puntos, no es necesario llevarlo a estos, pues la configuración que utilizamos para las canastas, requiere movimientos dentro de un rango de giro de la base, que se encuentra sobre el eje Z de 180^0, además en el eje Z, que representa la elevación sobre la base, el robot puede alcanzar una altura máxima de 500 mm y en los ejes X y Y, que representan los desplazamientos horizontal y vertical, respectivamente,el robot puede moverse a lo largo de todo el rango disponible en el espacio de trabajo, con la capacidad de alcanzar una distancia máxima de 500 unidades en cada uno de estos ejes.

Tabla 1. Alcance mínimo y máximo de la plataforma móvil.

Coordenadas	**Alcance mínimo**	**Alcance máximo**
X	-500	500
Y	-500	500
Z	0	500

Restricciones del robot

Ningún servo se puede mover más de 180^0, el robot no es flexible ni sus articulaciones extensibles, por lo que su longitud absoluta no se podrá aumentar de otra manera que no sea estirando todos los servomotores, en caso de que el robot esté montado en una mesa que no sea del tamaño específico de la base o menor, éste no podrá alcanzar debajo de sí mismo, ya que debería atravesar la mesa, cosa que no sería posible, además, está fijado a la mesa.

El robot está en una orientación específica previamente delimitada de tal forma que pueda realizar sus operaciones de la manera más satisfactoria posible sin molestar la visión de la cámara que muestra la posición de los dados.

Además, el robot no puede consumir un total mayor a 8 amps, y en sus segmentos electrónicos individuales, una parte del circuito no puede usar una cantidad mayor a 3 amps, la de los motores, y la segunda parte no puede consumir más de 5 amps, el arreglo de leds.

Planeación de trayectoria

En este proyecto de desarrollo de robot, no se incorporó la planificación de trayectorias. Esta decisión se debió principalmente a restricciones de tiempo que nos impidieron llevar a cabo esta implementación. Aunque observamos que el robot lograba alcanzar las posiciones deseadas sin inconvenientes, reconocemos que la inclusión de la planificación de trayectorias habría sido sumamente beneficiosa. Esta herramienta nos habría permitido suavizar los movimientos de los servomotores y mejorar el control en las tareas de pick & place. La capacidad de planificar rutas óptimas habría permitido minimizar el tiempo de ejecución de los movimientos, reducir el desgaste de los componentes mecánicos y mejorar la calidad de los productos finales o la ejecución de las tareas asignadas al robot. A pesar de no haber implementado la planeación de trayectoria a nuestro robot, sí se pensó en una forma de implementarlo, en primer lugar se definió el espacio de trabajo y las restricciones, considerando las posiciones y orientaciones que el último eslabón es capaz de alcanzar física y mecánicamente, además se tienen definidas las posiciones iniciales y finales, las

cuales son los puntos a donde se quiere llevar el robot, estos representan las coordenadas en donde se encuentra cada uno de los dados. Tomando esto en cuenta, se puede realizar una planificación de trayectoria mediante polinomios de 3er orden, en cada una de las dimensiones (x,y,z). Estos polinomios podrían ser ajustados para suavizar y conectar los puntos de inicio y fin de la trayectoria, así como para satisfacer las restricciones de velocidad y aceleración del robot. La elección específica de los coeficientes depende de los detalles de la trayectoria deseada y las restricciones de nuestro sistema.

Circuito electrónico

Para el diseño de nuestro circuito eléctrico tuvimos que tener cuenta la alimentación de nuestros actuadores, nuestros arreglos de leds y la alimentación de nuestros microprocesadores, por ello decidimos que la alimentación para nuestro circuito sería una fuente capaz de manejar el consumo de corriente de todos los componentes, y por ello elegimos una fuente de 12V 30A para después regular esos 12V a 5V usando reguladores step down. En nuestro diseño alimentamos 2 circuitos diferentes, uno para el funcionamiento de un arreglo de leds RGB y la alimentación de un ESP 01 para el control de los mismos, y el otro para el arreglo de Servos MG995, junto a su controlador correspondiente, el ESP32. Conectamos dos reguladores en paralelo, y con uno alimentamos a nuestro ESP32 junto a todos los servomotores que se van a utilizar, el segundo le da energía al array de leds, a otro regulador de voltaje 3.3 y a mediante estos 3.3v al ESP 01.

Nuestros cálculos para la cantidad de amperaje necesario por parte de los dos circuitos nos dice que mientras no usemos todos los servos al mismo tiempo, los reguladores no tendrán ningún problema, además que el circuito de mayor consumo es el encargado de los leds, por eso la elección que el regulador con mayor capacidad de amperaje fuera usado por los leds, mientras que el otro, supone una suficiente autonomía para el uso que le daremos, que no es el máximo.

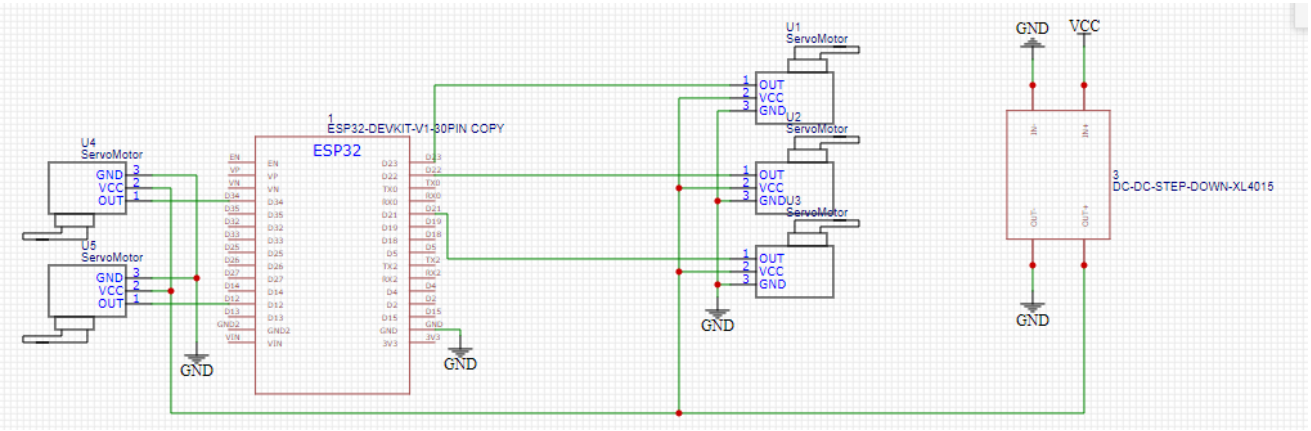


Figura 9. Primera parte del circuito electrónico.

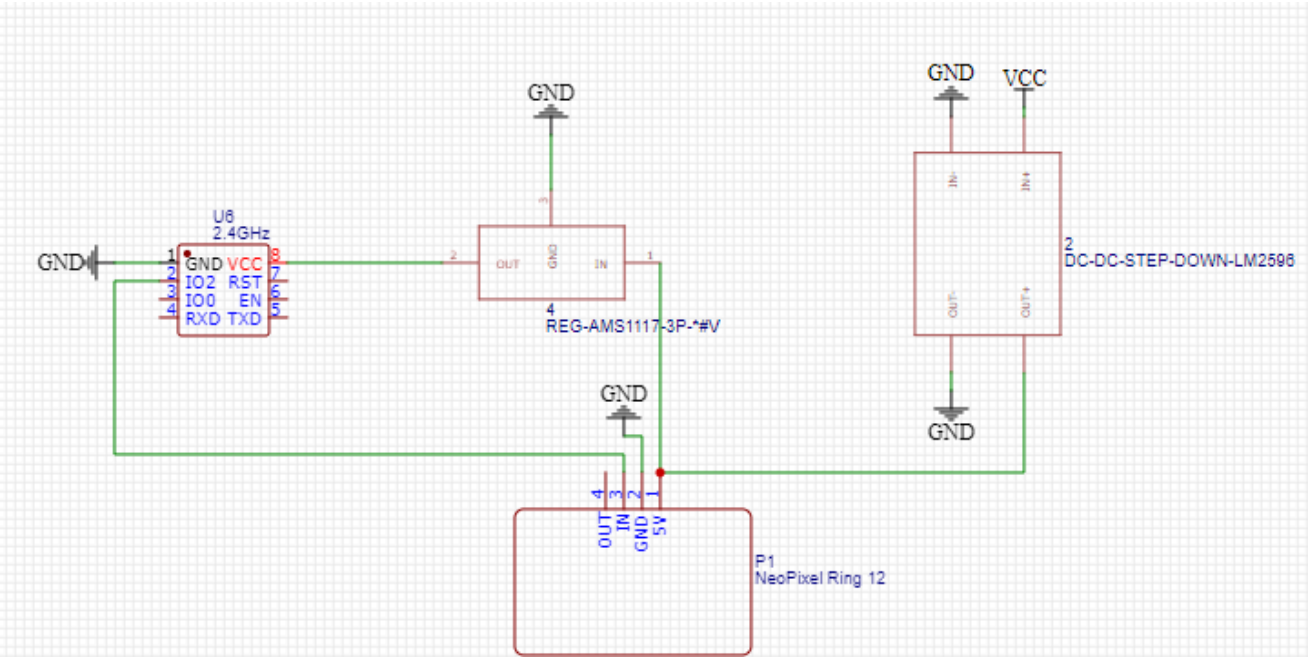


Figura 10. Segunda parte del circuito electrónico.

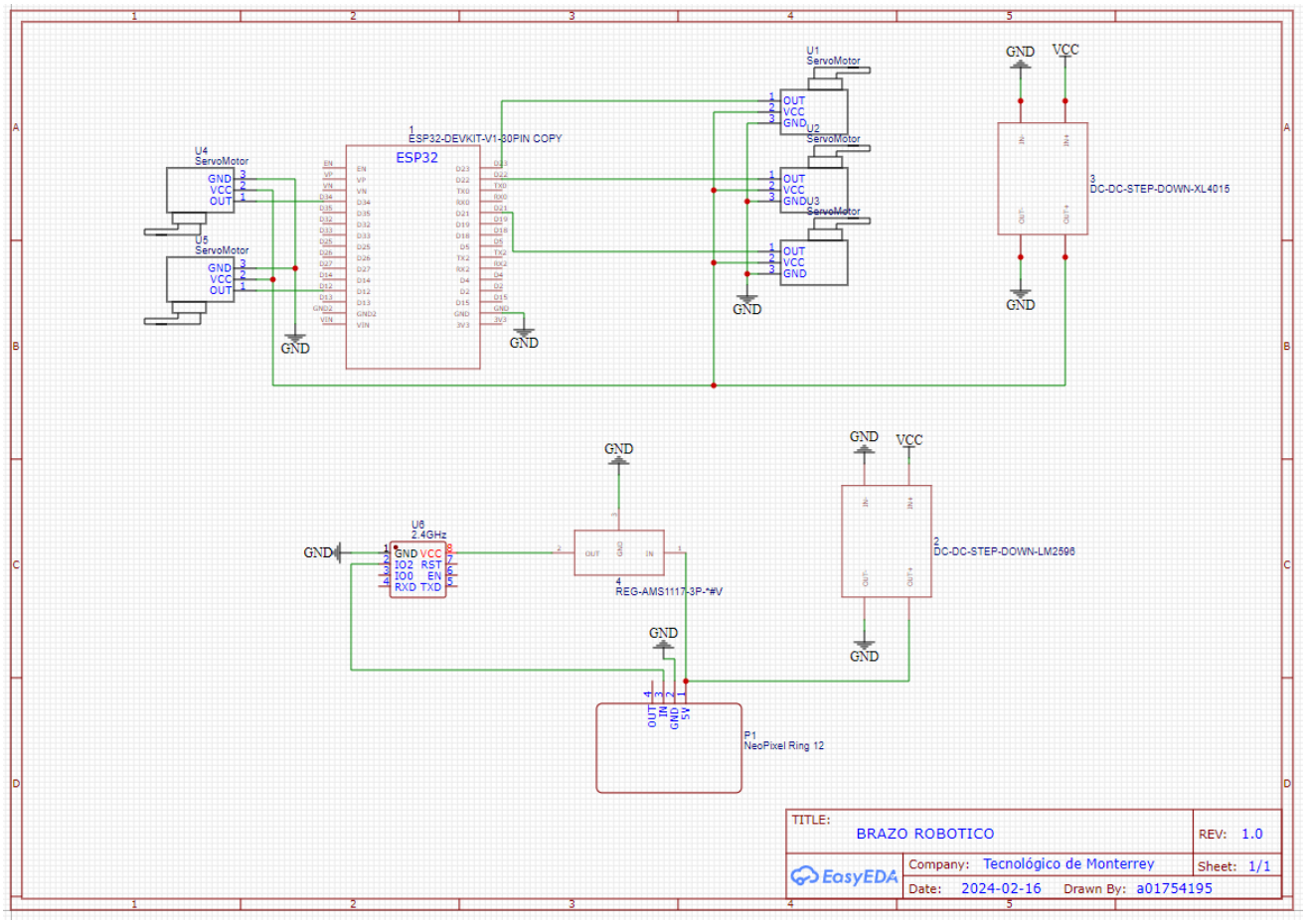


Figura 11. Esquemático completo.

Administración del proyecto

El proyecto se desarrolló a lo largo de 5 semanas, se realizó un calendario con la distribución de actividades necesarias, con la finalidad de presentar un robot competente y eficiente para la competencia. Cada etapa del proyecto se planificó, desde la definición de objetivos del proyecto, hasta la participación en la competencia y la posterior documentación de resultados. La asignación de responsabilidades se definió de manera que la carga de trabajo se distribuyera de manera equitativa, para lograr una colaboración efectiva y productiva entre todos los integrantes del equipo.

El objetivo planteado al inicio del proyecto fue realizar un brazo robótico manipulador de 4 grados, que recogiera 10 dados dentro de una canasta y los trasladara a otra canasta, con separaciones de máximo 2 cm entre cada uno de ellos.

Calendario de actividades

Tabla 2. Calendario de actividades realizadas.

Semana	Actividad	Responsable
1	Revisión de reglamento de competencia y requisitos de construcción.	
	Investigación de tecnologías para desarrollar el proyecto.	

	Investigación de los componentes y materiales.	Abrham Trejo Sebastian Barrio Enrique Fest Gabriela Sánchez
	Definición del prototipo objetivo.	
	Planificación de actividades.	
	Establecimiento de roles y responsabilidades.	
2	Diseño mecánico	Sebastian Barrio
	Diseño en CAD	Sebastian Barrio y Gabriela Sánchez
	Cinemática directa	Enrique Fest
3	Cinemática inversa	Sebastian Barrio
	Impresión 3D de las piezas.	Sebastian Barrio y Enrique Fest
	Diseño del circuito electrónico.	Sebastian Barrio Abraham Trejo
	Construcción del circuito electrónico.	Sebastian Barrio Abraham Trejo Enrique Fest
	Adaptación de contenedores y dados	Abraham Trejo y Enrique Fest
4	Ensamblaje de prototipo	Sebastian Barrio Gabriela Sánchez
	Integración de electrónica con el prototipo	Gabriela Sánchez
	Programación del controlador y actuadores	Sebastian Barrio
	Pruebas de interacción hardware-software	Sebastian Barrio Abrham Trejo Enrique Fest Gabriela Sánchez

	Programación de visión por computadora	Sebastian Barrio Gabriela Sánchez Abraham Trejo
5	Prueba y Calibración del programa de visión por computadora	Sebastian Barrio Abraham Trejo Enrique Fest Gabriela Sánchez
	Pruebas y ajustes finales	
	Simulación del entorno de la competencia	
	Documentación del proyecto	

De acuerdo al calendario establecido donde las actividades en verde son las realizadas, todas las actividades se terminaron en el tiempo establecido. Algo importante es que las actividades planteadas para realizarse en la semana 5, sí se realizaron en esa semana, de esta manera podemos concluir que la administración del proyecto fue la adecuada, y se comprueba que sí se siguió el calendario de administración de actividades, cumpliendo con él en tiempo y forma. Es importante mencionar que todos los integrantes cumplieron con las actividades de las que eran responsables, esto fue un punto clave para desarrollar con éxito el proyecto.

Resultados

Implementación Mecánica

Para la implementación del prototipo se utilizó toda la información previamente mencionada, se realizó la construccion del prototipo usando impresion 3D FDM con las piezas realizadas mediante cad, se utilizó para todas las piezas mecanicas PLA+ de color negro y se intergaron detalles esteticos fabricados mediante impresion 3D SLA utilizando resina tipo ABS de color blanco, todas las piezas estan unidas mediante tornillos, o insertadas a presion como en el caso de los rodamientos que sostienen las articulaciones. Los servos están unidos a los eslabones usando presión al plástico así como también un tornillo en el eje de rotación, que lo mantiene asegurado.

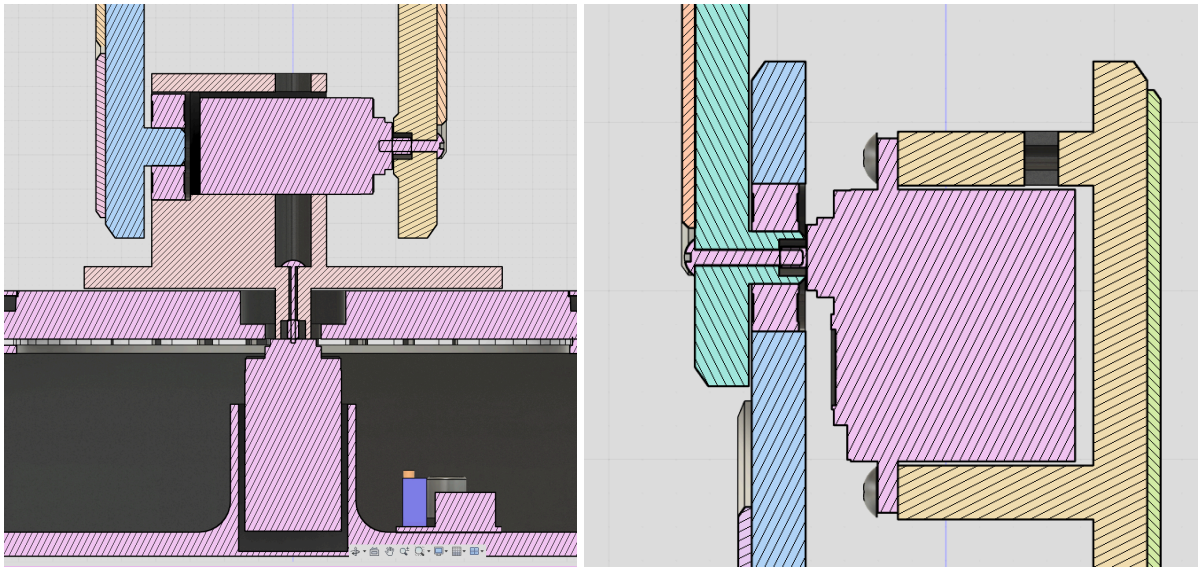


Figura 12. Sección transversal de las uniones de los servos

Esto junto con el uso de rodamientos permitió una fácil implementación del diseño, que a pesar de tener una cantidad considerable de tornillos, permite que todas las partes del sistema se mantengan rígidas y gracias a los rodamientos tengan un movimiento suave y sin forzar al motor. La construcción final del robot integrando todas las piezas fue la siguiente.

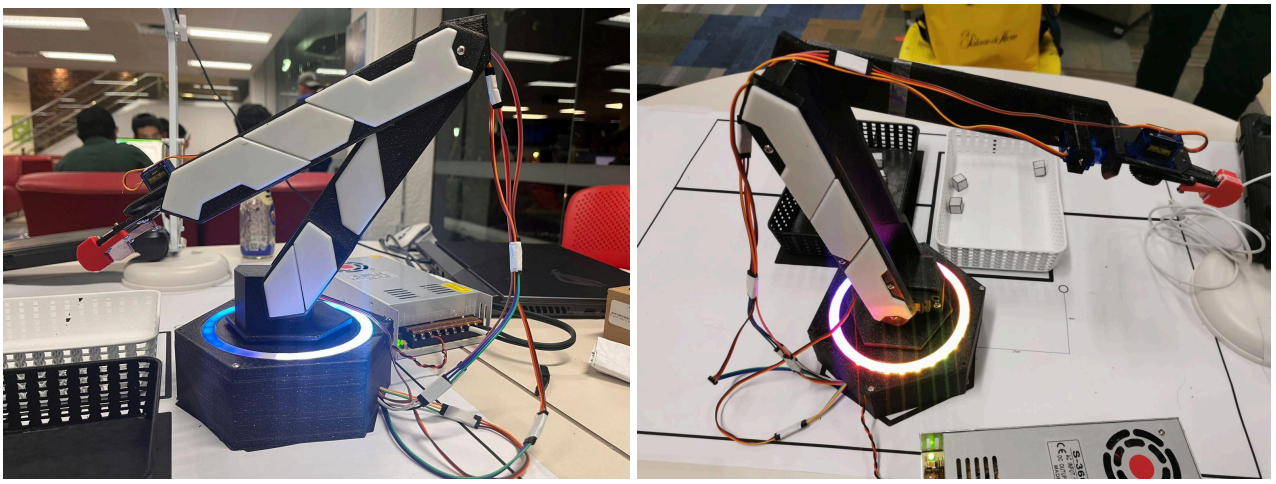


Figura 13. Construcción completa del robot

Implementación electrónica

Construimos el circuito descrito en la sección de electrónica de la manera más compacta posible usando un PCB protoboard en el que soldamos headers para conectar los componentes como los servos, leds, microcontroladores y soldamos los reguladores, después hicimos todas las conexiones e insertamos el circuito completado en la base del robot.

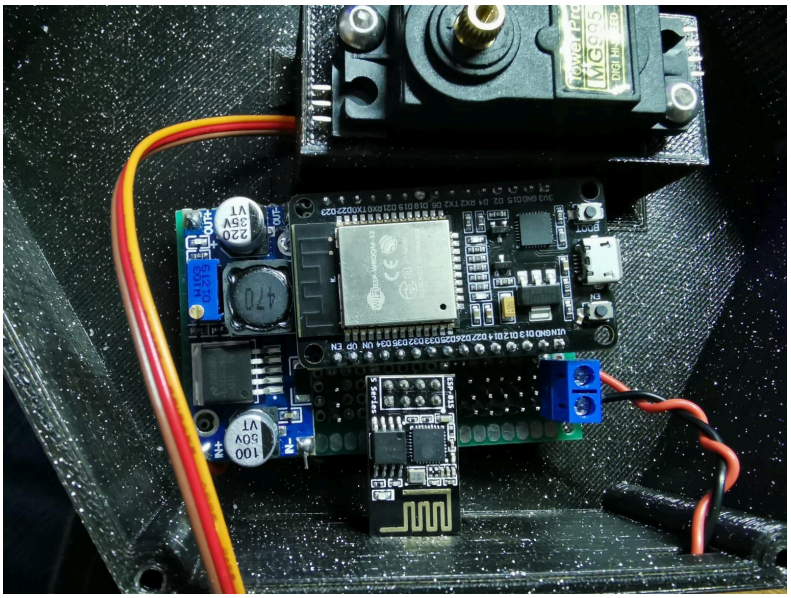


Figura 14. Construcción del circuito electrónico del robot

Implementación del software

El software del sistema del manipulador se compone de 2 programas, uno es un programa de python que se ejecuta en una PC y el otro es el programa en C++ de arduino que se ejecuta en el ESP32, estos dos se comunican mediante una conexión WiFi por la cual el programa de python envía datos de los ángulos que se deben ejecutar en el brazo robótico y el ESP32 contesta con el estado actual para evitar errores durante la comunicación.

El programa del ESP32 lleva a cabo solamente la escritura de los ángulos a los servos, pero incluye funcionalidad para la actualización del código de manera inalámbrica o programming Over The Air utilizando la librería “ArduinoOTA” y evidentemente también incluye funcionalidad para conectarse a la red WiFi por la cual se comunica con el programa de python. Cabe destacar que el orden de

escritura de los servos lleva un orden el cual consideramos es el orden óptimo para este robot y la función que debe realizar. El código completo del ESP32 se puede encontrar en el Anexo

Por otro lado, el código de python se encarga de el manejo de vision para la detección de cubos, los cálculos de conversión de coordenadas para pasar de las coordenadas de la cámara a las del robot y también de calcular los ángulos necesarios para cada servo utilizando la cinemática inversa del robot.
Para la visión computacional se usan las siguientes funciones, primero process frame que se encarga de buscar contornos en la imagen para posteriormente encontrar los cubos, sus coordenadas y su rotación, para esto primero le aplicamos los filtros o transformaciones de convertir a escala de grises, blur y luego sharpen, que nos da una imagen más fácil de trabajar para la binarización de la imagen, usamos el método de otsu para binarizarla y posteriormente ejecutamos un filtro más, llamado close, que se encarga de eliminar el ruido y las partículas pequeñas de la imagen. Ese es todo el procesamiento, ahora solo analizamos la imagen restante en busca de contornos y luego de esos contornos buscamos rectángulos, los dibujamos en la imagen, obtenemos sus coordenadas y las regresamos en la función. El código es el siguiente:

```
def processImage(frame):
    coordList = []
    #image process
    image = frame
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blur = cv2.medianBlur(gray, 5)
    sharpen_kernel = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
    sharpen = cv2.filter2D(blur, -1, sharpen_kernel)

    ret, thresh = cv2.threshold(sharpen, 0, 255, cv2.THRESH_OTSU)
    print("Used threshold value: ", ret)

    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
    close = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=1)

    contours = cv2.findContours(close, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours = contours[0] if len(contours) == 2 else contours[1]
    cv2.drawContours(image, contours, -1, (0, 0, 255), 3)
    cv2.circle(frame, pixelZero, 5, (255, 0, 255), -1)
    min_area = 400
    max_area = 2500
    image_number = 0

    #square detection
```

```
    for cnt in contours:
        area = cv2.contourArea(cnt)
        rect = cv2.minAreaRect(cnt)
        box = cv2.boxPoints(rect)
        box = np.intp(box)
        width = rect[1][0]
        height = rect[1][1]
         if area > min_area and area < max_area and (width/height > 0.6 and width/height < 1.4):  # Check if
shape is almost square
            cv2.drawContours(image, [box], 0, (36,255,12), 2)
            coordList.append(rect[0])
            cv2.circle(frame, (int(rect[0][0]), int(rect[0][1])), 5, (255, 0, 0), -1)
    cv2.imshow('Video Stream', frame)

    return frame, coordList
```

La otra función de visión computacional se encarga de encontrar el sistema de referencia de la cámara y del espacio de trabajo, que luego posteriormente convertiremos al espacio coordenado del robot para llevar a cabo los cálculos de cinemática inversa, en la alfombra de nuestro espacio de trabajo tenemos un circulo, con esta función determinamos tanto la posición del círculo en la cámara como la escala de la cámara en pixeles para conocer cuántos píxeles equivalen a cuantos mm y poder mover el robot en el espacio real, para esto llevamos a cabo el mismo procedimiento para obtener la imagen binaria pero en este caso usamos la función HoughCircles para detectar círculos en la imagen, obtenemos sus coordenadas y su diámetro en pixeles, en el caso de nuestra área de trabajo impresa, sabemos que este círculo debe tener un diámetro de 10mm por lo que podemos usar este dato para encontrar la escala de la cámara y hacer los cálculos previamente mencionados. El código de esta función es:

```
def findZero(frame):
    # Convert image to HSV color space
    image = frame
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blur = cv2.medianBlur(gray, 5)
    sharpen_kernel = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
    sharpen = cv2.filter2D(blur, -1, sharpen_kernel)

    ret, thresh = cv2.threshold(sharpen, 0, 255, cv2.THRESH_OTSU)
    print("Used threshold value: ", ret)

    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
    close = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=1)
    #cv2.imshow('close', close)
      circles = cv2.HoughCircles(thresh, cv2.HOUGH_GRADIENT, 1, 20, param1=50, param2=20, minRadius=10,
maxRadius=30)
```

```
    # If circles are found, draw them and set pixelDiameter to the diameter of the first found circle
    if circles is not None:
        circles = np.uint16(np.around(circles))
        pixelDiameter = circles[0][0][2] * 2  # Diameter is twice the radius
        print("Pixel Diameter: ", pixelDiameter)
        cv2.circle(frame, (circles[0][0][0], circles[0][0][1]), 5, (255, 0, 255), -1)
        cv2.circle(frame, (circles[0][0][0], circles[0][0][1]), 2, (0, 0, 255), 3)
        for i in circles[0, :]:
            # draw the outer circle
            cv2.circle(frame, (i[0], i[1]), i[2], (0, 255, 0), 2)
            # draw the center of the circle
            cv2.circle(frame, (i[0], i[1]), 2, (0, 0, 255), 3)
        zero = (circles[0][0][0], circles[0][0][1])
        #cv2.imshow('circles', frame)
    else:
        zero = None
        pixelDiameter = None  # Ensure pixelDiameter is None if no circles are found
    return zero, pixelDiameter
```

Por último nos quedan solo 3 funciones ayudantes, sendAngles, pixelToMM y calculateJoints. Las primeras 2 como su nombre indica solo se encargan de enviar los datos por WiFi al robot y de hacer la conversión de pixeles a milímetros usando la información que ya encontramos, y la otra función, calculateJoints, se encarga de convertir una entrada de coordenadas (x, y, z) y 2 parametros mas para la rotación del cubo y si la garra debe estar abierta o cerrada, y usando la cinemática inversa regresa el set de los 5 ángulos necesarios para cada servo, la función es la siguiente:

```
def calculateJoints(x,y,z,cubeRot=0,bOpen=False):
    A1=55.0  # Z offset from the 0 coord to the first movable articulation theta1
    A2=30.0  # Z offset between theta1 and theta2
    A3=200.0
    A4=200.0
    A5=100.0
    # Adjust z by A1 and A2 offsets
    z_adjusted = z - A1 - A2
    # Calculate the radius of the semi-sphere domain of the robot
    max_radius = A3 + A4 + A5
    # Calculate the actual distance of the point from the origin considering z adjustments
    actual_distance = math.sqrt(x**2 + y**2 + z_adjusted**2)
    # Clamp the x, y, z values if they are outside the semi-sphere domain
    if actual_distance > max_radius:
        print("Point outside range, clamping x, y, z values")

    cosTheta3 = max(min(((x**2+y**2+z_adjusted**2-A3**2-(A4+A5)**2)/(2*A3*(A4+A5))), 1), -1)
    senTheta3 = -math.sqrt(1-cosTheta3**2)
    theta1 = math.degrees(math.atan2(y,x))
theta2=
math.degrees(math.atan2(z_adjusted,math.sqrt(x**2+y**2))-math.atan2((A4+A5)*senTheta3,A3+(A4+A5)*cosTheta3))
    theta3 = -(math.degrees((math.atan2(senTheta3,cosTheta3))))
    openVal = 70 if bOpen else 0
```

```
    joints = [theta1, theta2-2, theta3-2, cubeRot, openVal]
    return joints
```

Con todas estas funciones podemos llevar a cabo el bucle principal del programa, que consiste en primero tomar una captura para encontrar el sistema de referencia y de escala, y luego encontrar todos los dados en la imagen, moverse hacia uno de ellos, tomarlo y llevarlo a su posición en la otra canasta, las posiciones finales se guardan en una lista, regresar a su posición 0, y por último tomar otra captura para repetir el proceso con todos los dados hasta que no se detecte ninguno. El código de la rutina principal es el siguiente:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('192.168.43.181', 12345))

cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

#10 coordinates for dice positions
finalPosList = []

sendAngles(calculateJoints(200,0,280,48,True))

ret, frame1 = cap.read()
if ret:
    pixelZero, pixelDiameternt = findZero(frame1)
    pixelDiameter = 24.5
    global RealMMZeroOffset
    RealMMZeroOffset = (260, -20)  # in mm
    posIndex = 0

while True:
    sendAngles(calculateJoints(200,0,280,48,True))
    time.sleep(2)
    cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

    ret, frame = cap.read()
    image, pixelList = processImage(frame)
    coordList = pixelToMM(pixelList, pixelDiameter)

    print(coordList)
    if coordList:
        #go to the cube coordinates
        sendAngles(calculateJoints(coordList[-1][0], coordList[-1][1], 40,48,True))
        time.sleep(2)
        #close the gripper
        sendAngles(calculateJoints(coordList[-1][0], coordList[-1][1], 40,48,False))
        time.sleep(2)
        #elevate the cube
        sendAngles(calculateJoints(coordList[-1][0], coordList[-1][1], 150,48,False))
        time.sleep(2)
```

```
        #go to the final elevated position
        sendAngles(calculateJoints(finalPosList[posIndex][0], finalPosList[posIndex][1], 150,48,False))
        time.sleep(2)
        #go to the final position
        sendAngles(calculateJoints(finalPosList[posIndex][0], finalPosList[posIndex][1], 40,48,False))
        time.sleep(2)
        #open the gripper
        sendAngles(calculateJoints(finalPosList[posIndex][0], finalPosList[posIndex][1],40,48,True))
        time.sleep(2)

        sendAngles(calculateJoints(finalPosList[posIndex][0], finalPosList[posIndex][1],150,48,True))
        time.sleep(2)
        posIndex += 1
    else:
        print("No object found")
    cv2.waitKey(0)
    cap.release()
    cv2.destroyAllWindows()
```

De igual manera, el código completo de todos los sistemas y programas complementarios se pueden encontrar en el Anexo.

Análisis de resultados

El robot realiza de manera eficiente la tarea de ordenar los cubos, al momento de recoger los cubos, llega de manera precisa a la posición de cada uno de ellos, y los traslada a la siguiente canasta, posicionándolo en las coordenadas deseadas, sin dejarlo caer, es importante mencionar que se respeta el requisito de dejar acomodados los dados con máximo 2 cm de separación. En el caso de la visión computacional, la programación y métodos de segmentación utilizados, son capaces de identificar los 10 cubos de manera correcta, reconociendo los bordes de los cubos y calculando su centro de área, logra obtener las coordenadas a partir de los pixeles, para llevar al robot a estas mismas. Un elemento clave para el funcionamiento de la visión computacional, es el plano del espacio de trabajo, pues de esta manera se delimitó mejor la posición de cada uno de los elementos y las pruebas de funcionamiento se pudieron realizar de manera que no hubo alteraciones entre ellas, es así como se logró simular siempre el mismo escenario haciendo esta fase más eficiente. Además el uso del plano, y al modificar las canastas y los dados, pintando una canasta de blanco y otra de negro, y los dados

de blanco, con contornos delimitados por líneas negras, mejoró significativamente la identificación de objetos en nuestro sistema al introducir un fondo completamente blanco en el entorno, lo que redujo el ruido visual. Esta diferenciación de colores y la claridad de los contornos permitieron una detección más precisa por parte del sistema de visión, mejorando la eficiencia y la confiabilidad del proceso de manipulación de objetos por el robot. En cuanto al desempeño del robot, realiza los movimientos necesarios y de manera precisa, llega a las posiciones deseadas sin comprometer de forma grave sus componentes, sin embargo, sí llega a tener movimiento bruscos, los cuales pueden llegar a mover algunos dados al momento de recolectar o dejar los dados. Para arreglar esto sería necesario implementar planeación de trayectorias, donde se establezcan restricciones de velocidad y tiempo. El tiempo en que el robot realiza la tarea no es extenso, pero, podría disminuirse, pues la forma en la que se trato de controlar los movimientos del robot y comprometer el área de trabajo de la menor forma posible fue dejando tiempo entre los cambios de posición, y al controlar la velocidad, se evitaría usar esta técnica.

Conclusiones individuales

Sebastian Barrio Bejarano:

En este reto se integraron diversas tecnologías y técnicas que tuvimos que llevar a cabo para que el robot funcionara de manera correcta, desde un diseño robusto que integrará suficiente soporte para los eslabones del robot y los servos, hasta un algoritmo de visión que detectara de manera confiable los cubos y su sistema de referencia, el objetivo del reto de mover los dados de una canasta a otra se logró de manera satisfactoria, aunque todavía hay espacio para mejorar el diseño, el resultado al que logramos llegar cumple con nuestras expectativas y el reto en general nos permitió integrar todo lo aprendido durante el periodo así como también habilidades conseguidas anteriormente, este prototipo muestra la viabilidad de un diseño como este de un manipulador de 4 DOF 4 en aplicaciones de pick and place.

Enrique Fest Flores:

El objetivo del reto, que fue la construcción y programación de un robot capaz de trasladar cubos de una caja a otra en un tiempo menor a 7 minutos sí se logró, ya fuimos capaces de crear un robot capaz de cumplir con esta tarea, además de apegarnos a nuestro diseño del CAD.

La visión computacional que fue parte esencial de nuestro código fue complejo, ya que lo visto en clase en matlab no se ha podido incorporar por completo y se tuvo que hacer más investigación para saber cómo incorporar correctamente en nuestro código final, al contrario de los cálculos de cinemática directa e inversa vistos en clase, que en este caso fueron útiles para obtener las configuraciones de las articulaciones del robot para que logre llegar a la posición de nuestro efector final deseada y con eso poder recoger los cubos para dejarlos trasladarlos de una caja a otra.

Gabriela Marina Sánchez Ravelo:

Durante el desarrollo de este reto se lograron implementar distintas técnicas, e integrarlas en un sistema mecatrónico, que resultó en un brazo robótico manipulador eficiente y preciso. Es importante mencionar que el contenido teórico visto durante el bloque Diseño y Desarrollo de robots, fue fundamental para lograr calcular la cinemática directa e inversa, e implementarla en la programación para definir los movimientos de los eslabones del robot, además de esto el módulo de visión, nos brindó las herramientas necesarias para integrar las visión computacional a nuestro proyecto y lograr el que el robot logre ir a cada uno de los dados sin importar que estén en diferentes lugares cada vez que se realice la tarea. Sin embargo, al implementar la cinemática inversa sí hubieron algunos inconvenientes, pero al final se lograron resolver. Algo que faltó integrar y pudo haber mejorado el resultado del proyecto fue la planeación de trayectorias, pues aunque nuestro robot logra realizar todos las trayectorias de manera satisfactoria, podría desempeñarlo de una mejor manera, realizando movimientos suaves y controlados, incluso al aplicar los conocimientos que obtuvimos en el módulo de control, se obtendrían mejores resultados. Sin embargo el objetivo principal se cumplió, el robot realiza lo establecido.

Abraham Trejo Arvizu :

El reto fue una buena manera de integrar los diferentes módulos aprendidos, como otros conceptos mecatrónicos pasados, en un robot funcional con gran valor industrial, ya que es algo muy trabajado allí, pero que por la mínima cantidad de tiempo, y las otras responsabilidades, no se pudo explorar al máximo y darle retoques para obtener resultados más congruentes; pero no falla en dejar en claro la necesidad de planeación como de implementación de aprendizajes. Y viendo las partes más negativas, los controladores de visión fueron muy endebles y muy susceptibles a fallos, esto por la falta de documentación en el tema, lo cual será muy frustrante para cualquiera que desee realizar lo visto en el documento, como además el control de los servos es complicado y llega a dar resultados no esperados. Y con el enfoque de ver los éxitos, tendremos en la mesa, la detección de cubos y círculos en las imágenes tomadas, la comunicación wifi con el esp 32, la buena implementación de la

cinemática inversa, usando arcotangentes, para evitar problemas con los signos y por último el funcionamiento electrónico, que permite un funcionamiento libre sin problemas de alimentación.

Trabajo futuro

El robot diseñado en este documento se puede aplicar en una gran variedad de trabajos, ya que se considera un brazo robótico de uso general donde podremos ver su necesidad en diferentes partes de la industria, y aplicando mejoras, para tener un movimiento más natural, podríamos imaginar el uso de este robot en:

- Pick and place: es un trabajo estándar en la industria para poder mover cada elemento de una caja a otra en orden, útil en situaciones donde el producto final no termina ordenado en la línea de producción y debe de ser empacada con gran velocidad.
- Rutinas de grabación: es una herramienta perfecta en situaciones donde es necesario hacer paneos precisos y repetitivos en sets pequeños, reduciendo tiempos en producción e incluso en escaneo 3d en objetos, ya que este necesita rutinas repetitivas para el escaneo de 360 grados sobre un objeto de interés.
- Manipulación de herramientas: la aplicación de ciertos instrumentos pequeños que se puedan intercambiar por la garra superior, abriría una gran posibilidad de uso en múltiples áreas de oportunidad para estos robots, generalmente usadas para puntos de soldadura.

Sugerencia de mejoras

Para obtener mejores resultados que los encontrados en este documento, proponemos la integración de la planeación de trayectoria, ya que al implementarla en el robot nos daría muchas más ventajas al momento de hacer su algoritmo pick & place.

Otra propuesta sería la implementación de mejores servomotores, ya que los SG90 utilizados para el efector final y la muñeca del robot tuvieron problemas, como la fuerza con la que pueden recoger los dados o la calidad de los mismos, ya que pueden sufrir averías y dañarse fácilmente.

Por último, nosotros elegimos un diseño para un robot manipulador de 4 DOF de uso general, pero para un reto específico como este puede haber sido mejor usar un diseño especializado para la tarea en lugar de hacer un robot manipulador.

Si tuviese que generar un robot mediante un prompt en una IA ¿Cuál sería ese prompt?

Sebastian Barrio Bejarano:

Eres una inteligencia artificial especializada en diseño de robots, genera un diseño para un robot manipulador de máximo 4 grados de libertad usando servomotores para realizar una tarea de pick and place, el robot debe de contar con un efector final capaz de recoger dados de 15 milímetros y un área de trabajo de alrededor de 400 milímetros, el robot debe de integrar visión computacional para la detección de cubos y ajustar el sistema de coordenadas de la cámara al sistema del robot.

Enrique Fest Flores:

En este caso consideré que le pido un robot como el del reto con las características que se nos pidieron: Ayúdame a generar un robot colaborativo de 4 grados de libertad, debe poder tomar dados de 1x1x1cm de una canasta y dejarlos en otra, para esto hay que usar visión computacional y tomar en cuenta el uso únicamente de servos para las articulaciones y el efector final, aparte de la cinemática directa e inversa del robot, también necesitaré que me apoyes con un programa de python para visión computacional, usaré una cámara de 720x10800 píxeles y como microprocesador un ESP32 por su conexión a internet.

Gabriela Marina Sánchez Ravelo:

Genera un diseño de un robot de 4 grados de libertad, el cual en el primer grado pueda rotar en su base, en el segundo y tercero que sería el primer eslabón y segundo eslabón, puedan hacer movimientos de flexión y extensión de manera que pueda manipular objetos en diferentes posiciones y alturas. El efector final, debe ser un pinza capaz de sostener objetos de al menos 2 cm. Además realiza la programación necesaria para incorporar la visión computacional, y sea capaz de identificar objetos cuadrados, para llevar al robot a estas posiciones y el robot pueda tomar los objetos y trasladarlos a otra posición específica.

Abraham Trejo Arvizu :

Podrías ayudarme a crear un brazo robótico de uso general con 4 grados de libertad, en un trabajo de pick and place, para dados de un tamaño pequeño, con un espacio de trabajo semiesférico usando servomotores, y un microprocesador esp-32, el cual debe de recibir la información ya calculada de una computadora, además de implementar visión computacional para los cubos.

Referencias

A. Barrientos, L.F. Peñín, C. Balaguer, R. Aracil, «Fundamentos de robótica», McGraw Hill, 1997.

Rodolfo Ponce R., Emmanuel A. Merchán C. Luis Héctor Hernández G. , Celeste Salgado P., Arturo E. Flores Pa. (septiembre, 2022). *"Solución a la cinemática directa e inversa de manipuladores robóticos, empleando álgebra de cuaterniones duales"*.https://somim.org.mx/memorias/memorias2022/articulos/A3_33.pdf

ESD/Robotics. "Historia y Evolución de la Robótica Industrial." *EDS Robotics*, 2 February 2021, https://www.edsrobotics.com/blog/evolucion-robotica-industrial/. Accessed 28 February 2024.

Moran, Michael E. "Evolution of robotic arms - PMC." *NCBI*, 2007, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4247431/. Accessed 28 February 2024.

Fikrul Akbar Alamsyah 2019 IOP Conf. Ser.: Mater. Sci. Eng. 494 012100

Steven M. Lavalle. Planning Algorithms. Cambridge University Press, 2006.

Anexos

Código del ESP32, Programa de python, código de matlab y todos los archivos del robot: **https://github.com/papalino456/RobotArm**

CAD (disponible en variedad de formatos) : **https://a360.co/49vovwI**