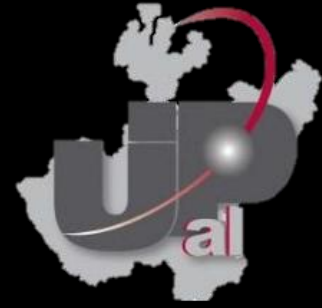


16/agosto/2019



Universidad Politécnica de la Zona Metropolitana
de Guadalajara --- **Ingeniería mecatrónica**

9°B T/M

Asignaturas: – Dinámica y control de robots

- Control inteligente

Profesor: – Enrique Morán Garabito

- Rosa María Razo Cerda

Integrantes:

**Lozada Canizal Jessica*

**Lozano Ochoa Marco Antonio*

**Navarro Cervantes Jose*

**Ramírez Arenas Juan Alberto*

Brazo antropomórfico

Reporte final – Proyecto ANUAL y MATERIA



Contenido

1. Meta	3
2. Objetivos	3
3. Justificación	3
4. Marco Teórico	3
4.1 Introducción	3
4.2 Brazo robótico	3
4.2.1 Tipos:	3
4.2.2 Brazos robóticos notables	4
4.3 Morfología del robot antropomórfico	4
4.4 Herramientas matemáticas para la localización espacial	5
4.4.1 Cinemática directa	5
4.4.2 Matriz de transformación homogénea	5
5. Cronograma.....	8
5.1 Diagrama de Gantt	9
6. Lista de materiales y costos	10
7. Diseño y análisis del brazo antropomórfico	11
7.1 Diseño del brazo en SolidWorks.....	11
7.1.1 Planos del brazo	12
7.1.2 Longitud y carga	15
7.2 Análisis en Ansys	15
7.2.1 Solución (representación gráfica)	15
8. Programación	17
8.1 Programa de control del microcontrolador	17
8.1.1 Comandos en ROS para el control del brazo.....	18
8.2 Programa de control difuso.....	18
9. Resultados	20
10. Conclusiones.....	21
10.1 Sugerencias y aportes	22
11. Referencias.....	23
Anexos.....	24
Anexo 1: Diagrama de Gantt	24
Anexo 2: Reporte generado	26

Units	26
Geometry	26
Mesh.....	27
Static Structural	28
Solution	29
Anexo 3: Código de programación del microcontrolador	31
Anexo 4: Código de programación en Python de la raspberry	34

1. Meta

Diseñar, construir y programar un brazo robótico controlado por el software de ROS.

2. Objetivos

- Diseñar la estructura del brazo en CAD.
- Construir el mecanismo del brazo.
- Programar el sistema de control del brazo antropomórfico con ROS.

3. Justificación

Brindar una alternativa para el control de objetos que pueden lesionar al operador al manejarlas directamente, mediante el control a distancia de un brazo robótico.

4. Marco Teórico

4.1 Introducción

Un brazo robótico es un tipo de brazo mecánico, normalmente programable, con funciones parecidas a las de un brazo humano; este puede ser la suma total del mecanismo o puede ser parte de un robot más complejo. Las partes de estos manipuladores o brazos son interconectadas a través de articulaciones que permiten tanto un movimiento rotacional (tales como los de un robot articulado), como un movimiento traslacional o desplazamiento lineal.

4.2 Brazo robótico

El efector final, o mano robótica, se creó para efectuar cualquier tarea que se desee como puede ser soldar, sujetar, girar, etc., dependiendo de la aplicación. Por ejemplo, los brazos robóticos en las líneas de ensamblado de la industria automovilística realizan una variedad de tareas tales como soldar y colocar las distintas partes durante el ensamblaje. En algunas circunstancias, lo que se busca es una simulación de la mano humana, como en los robots usados en tareas de desactivación de explosivos.

4.2.1 Tipos:

Robot cartesiano: Usado para trabajos de “pick and place” (tomar y colocar), aplicación de impermeabilizantes, operaciones de ensamblado, manipulación de máquinas herramientas y soldadura por arco. Es un robot cuyo brazo tiene tres articulaciones prismáticas, cuyos ejes son coincidentes con los ejes cartesianos.

Robot cilíndrico: Empleado para operaciones de ensamblaje, manipulación de máquinas herramientas, soldadura por punto y manipulación en máquinas de fundición a presión. Es un robot cuyos ejes forman un sistema de coordenadas cilíndricas.

Robot esférico / Robot polar: Utilizado en la manipulación en máquinas herramientas, soldadura por punto, fundición a presión, máquinas de desbarbado, soldadura por gas y por arco. Es un robot cuyos ejes forman un sistema polar de coordenadas.

Robot SCARA: Usado para trabajos de “pick and place” (tomar y colocar), aplicación de impermeabilizantes, operaciones de ensamblado y manipulación de máquinas herramientas. Es un robot que tiene dos articulaciones rotatorias paralelas para proporcionar elasticidad en un plano.

Robot articulado: Utilizado para operaciones de ensamblaje, fundición a presión, máquinas de desbarbado, soldadura a gas, soldadura por arco y pintado por spray. Es un robot cuyo brazo tiene como mínimo tres articulaciones rotatorias.

Robot paralelo: Uno de los usos es la plataforma móvil que manipula las cabinas de los simuladores de vuelo. Es un robot cuyos brazos tienen articulaciones prismáticas o rotatorias concurrentes.

4.2.2 Brazos robóticos notables

En el espacio el Sistema de Manipulación Remota del Transbordador Espacial, también conocido como Canadarm, y su sucesor el Canadarm2, son ejemplos de brazos robóticos de múltiples grados de libertad que ha sido usado para realizar distintas tareas tales como inspección de los transbordadores espaciales y satélites a través de cámaras colocadas en su extremo o mano, y tareas de carga y descarga de la bodega de los transbordadores espaciales.

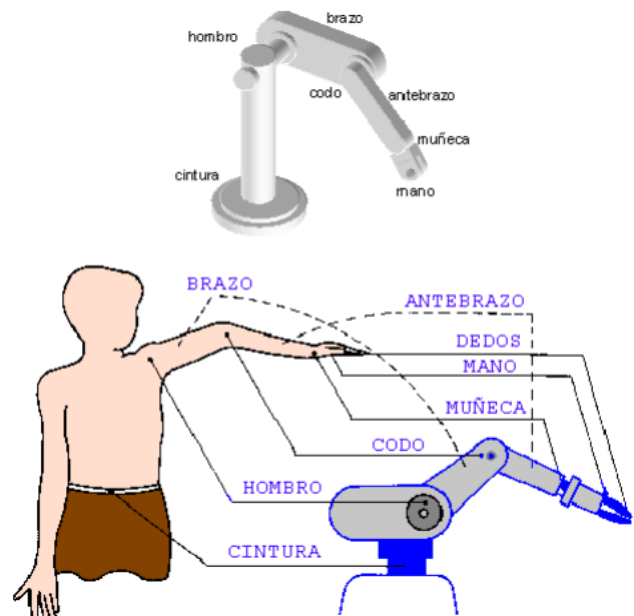


Ilustración 1 - Comparación brazo-robot

4.3 Morfología del robot antropomórfico

Dependiendo de tipo de articulaciones que posee un robot, se puede definir su clasificación, existen articulaciones rotacionales (que generan solamente movimiento de rotación) y prismáticas o lineales (que generan desplazamientos longitudinales) y el conjunto de estas puede definir el tipo de robot industrial entre Antropomórfico (con mínimo 3 articulaciones rotacionales).

El posicionamiento del robot en el espacio tridimensional requiere de 6 coordenadas (tres para la posición cartesiana y 3 para la orientación de la herramienta de trabajo), la relación establecida entre coordenadas cartesianas, articulares y su orientación se denomina cinemática directa.

4.4 Herramientas matemáticas para la localización espacial

4.4.1 Cinemática directa

Estudia el movimiento que realiza el robot con respecto a un sistema de referencia. Se interesa por la descripción analítica del movimiento espacial del robot como una función del tiempo. Y en especial, las relaciones de posición y orientación del extremo final del robot.

Hay dos problemas fundamentales en la cinemática:

Problemática cinemática directa

Determina cuál es la posición y orientación del extremo del robot con respecto a un sistema de coordenadas como referencia.

Métodos de solución de problema cinemática directa:

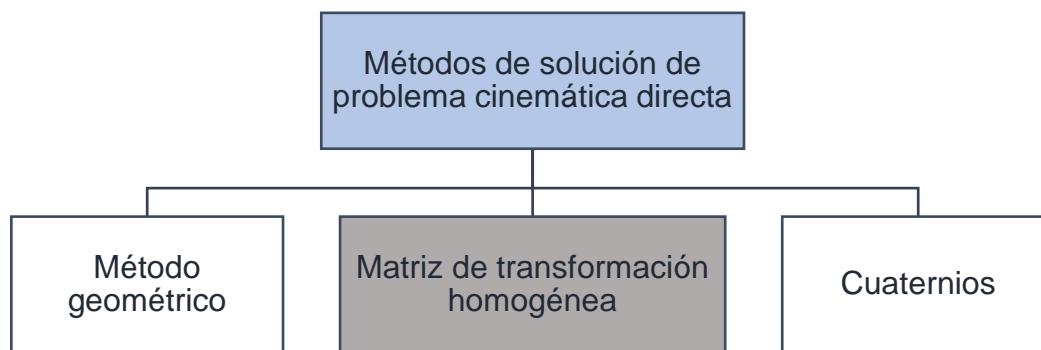


Ilustración 2 - Métodos de solución

4.4.2 Matriz de transformación homogénea

Álgebra vectorial y matricial.

Cada uno de los elementos que componen el brazo robótico es una cadena cinemática en la que cada eslabón se encuentra unido por articulación. Es suficiente con encontrar una matriz de transformación que calcule o transforme la posición del extremo del robot tomando como coordenadas de referencia la base.

Denavit Y Hartenberg propusieron un método cinemático para describir y representar la geometría espacial de los elementos de una cadena cinemática, y en particular de un robot, como referencia un sistema fijo.

Representación Denavit Hartenberg (D-H)

Jacques Denavit y Richard Hartenberg propusieron en 1955 un método matricial que permite establecer de manera sistemática un sistema de coordenadas $[s]$ ligado a cada eslabón i de una cadena articulada, pudiéndose determinar a continuación las ecuaciones cinemáticas de la cadena completa.

Según la representación D-H, escogiendo adecuadamente los sistemas de coordenadas asociadas a cada eslabón, será posible pasar de uno al siguiente mediante 4 transformaciones básicas que dependen exclusivamente de las características (fundamentales) geométricas de cada eslabón.

Principios básicos de la representación D-H

- 1 Rotación alrededor del eje Z_{i-1} un ángulo θ_i .
- 2 Traslación a lo largo de Z_{i-1} una distancia d_i : vector $d_i(0,0,d_i)$
- 3 Traslación a lo largo de X_i una distancia a_i : vector $a_i(a_i,0,0)$
- 4 Rotación alrededor del eje X_i del ángulo α_i ;

Algoritmo de Denavit-Hartenberg

DH 1. Numerar los eslabones comenzando con 1 (primer eslabón móvil de la cadena) y acabando con n (último eslabón móvil). Se numerará como eslabón 0 a la base fija del robot.

DH 2. Numerar cada articulación comenzando por 1 (la correspondiente al primer grado de libertad) y acabando en n.

DH 3. Localizar el eje de cada articulación. Si ésta es rotativa, el eje será su propio eje de giro. Si es prismática, será el eje a lo largo del cual se produce el desplazamiento.

DH 4. Para i de 0 a n-1 situar el eje z_i sobre el eje de la articulación i + 1.

DH 5. Situar el origen del sistema de la base $\{S_0\}$ en cualquier punto del eje z_0 . Los ejes x_0 y y_0 se situarán de modo que formen un sistema dextrógiro con z_0 .

DH 6. Para i de 1 a n-1, situar el origen del sistema $\{S_i\}$ (solidario al eslabón i) en la intersección del eje z_i con la línea normal común a z_{i-1} y z_i . Si ambos ejes se cortasen se situaría $\{S_i\}$ en el punto de corte. Si fuesen paralelos $\{S_i\}$ se situaría en la articulación i + 1.

DH 7. Situar x_i en la línea normal común a z_{i-1} y z_i .

DH 8. Situar y_i de modo que forme un sistema dextrógiro con x_i y z_i .

DH 9. Situar el sistema $\{S_n\}$ en el extremo del robot de modo que z_n coincida con la dirección de z_{n-1} y x_n sea normal a z_{n-1} y z_n .

DH 10. Obtener θ_i como el ángulo que hay que girar en torno a z_{i-1} para que x_{i-1} y x_i queden paralelos.

DH 11. Obtener d_i como la distancia, medida a lo largo de z_{i-1} , que habría que desplazar $\{S_{i-1}\}$ para que x_i y x_{i-1} quedasen alineados.

DH 12. Obtener a_i como la distancia medida a lo largo de x_i (que ahora coincidiría con x_{i-1}) que habría que desplazar el nuevo $\{S_{i-1}\}$ para que su origen coincidiese con $\{S_i\}$.

DH 13. Obtener α_i como el ángulo que habría que girar entorno a x_i , para que el nuevo $\{S_{i-1}\}$ coincidiese totalmente con $\{S_i\}$.

DH 14. Obtener las matrices de transformación ${}^{i-1}\mathbf{a}_i$ definidas en [4.10].

DH 15. Obtener la matriz de transformación que relaciona el sistema de la base con el del extremo del robot $\mathbf{T} = {}^0\mathbf{A}_1 \cdot {}^1\mathbf{A}_2 \cdot \dots \cdot {}^{n-1}\mathbf{A}_n$.

DH 16. La matriz \mathbf{T} define la orientación (submatriz de rotación) y posición (submatriz de traslación) del extremo referido a la base, en función de las n coordenadas articulares.

5. Cronograma

El cronograma a continuación muestra la planeación realizada para conseguir realizar el proyecto en la fecha de entre especificada por el profesor, con la excepción de que en algunas fechas no se siguió el plan debido a cuestiones externas para la construcción del mecanismo.

Actividad	Designado(s)	Fecha
Diseñar la estructura del brazo en CAD	Juan	22/01/19
Calcular esfuerzos máximos del brazo en ansys	Jose y Marco	24/01/19
Hacer los eslabones del brazo en MDF	Marisol y Jessica	05/02/19
Investigar y comprar motores y demás partes de las articulaciones	Juan, Marco y Jose	07/02/19
Ensamblar brazo	Todos	09/02/19
Comenzar con la programación y primeras pruebas	Todos	13/02/19
Correcciones mecánicas del brazo	Todos	20/03/19
Investigar parámetros para programación en ROS	Todos	27/03/19
Realizar programa de control del brazo en ROS	Todos	02/04/19
Segunda parte		
Corregir parte mecánica	Todos	27/05/19
Análisis de la parte mecánica corregida	Todos	06/06/19
Programar prototipo de comunicación serial	Todos	28/06/19
Programar sistema de control	Todos	14/05/19
Realizar prueba de control	Todos	23/05/19

Tabla 1 - Cronograma

5.1 Diagrama de Gantt

Para el seguimiento de las actividades establecidas en el diagrama de Gantt pudimos realizar la mayoría de las tareas en la fecha establecida, sin embargo, fueron algunas tareas las cuales no se realizaron en la fecha deseada las cuales son las siguientes:

- La tarea 15 que corresponde a la elaboración de los eslabones se retrasó debido a que el corte de la madera no estuvo listo a tiempo.
- La tarea 17 correspondiente al ensamble del brazo se prolongó debido a que el posicionamiento del motor de la segunda articulación no era la mejor, así que se optó por moverlo a la base.

El diagrama completo se puede visualizar en el anexo 1.

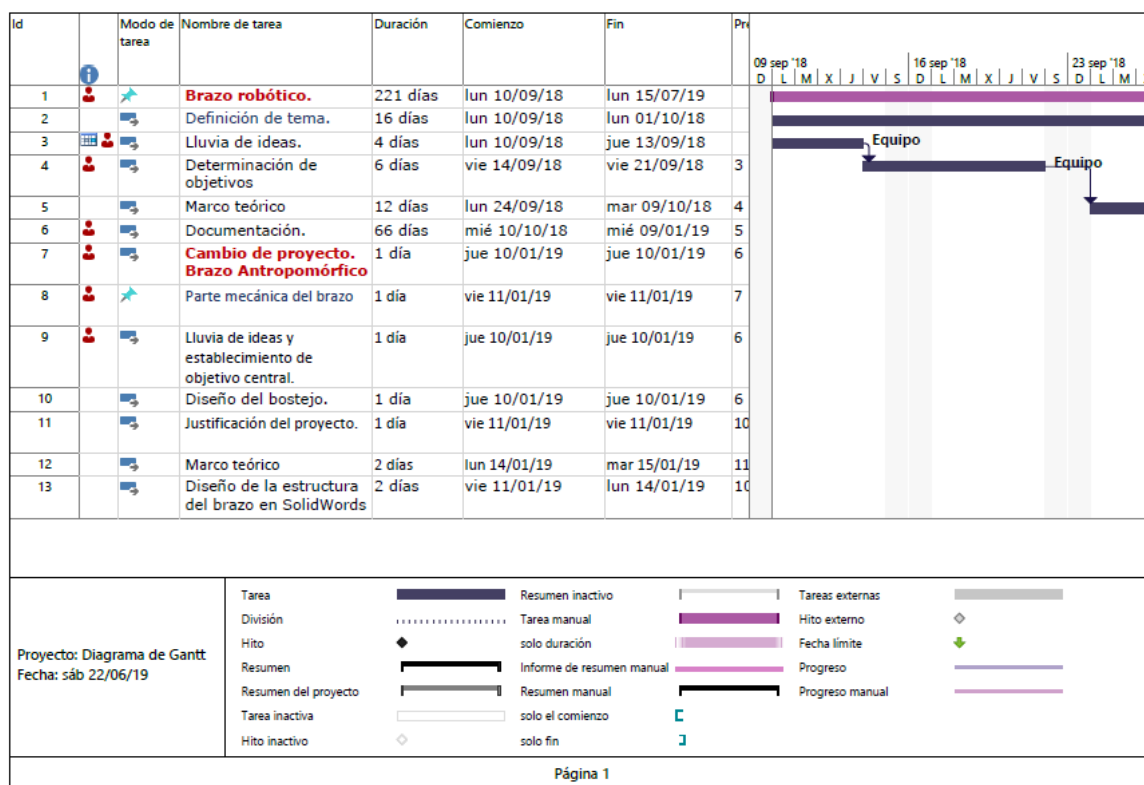


Ilustración 3 - Diagrama de Gantt

6. Lista de materiales y costos

Material	Costo por pieza	Cantidad	Subtotal
Motores Nema 23 (9kp)	\$150.00	4	\$600.00
Drivers A4988	\$45.00	3	\$135.00
Raspberry pi 3	\$1200.00	1	\$1200.00
Microcontrolador FRDM KL25Z	\$540.00	1	\$540.00
Alambre para prototipos	\$3.00	7 metros	\$21.00
Balero cónico de automóvil	\$40.00	2	\$80.00
Engranes de aluminio 20 dientes	\$30.00	3	\$90.00
Engranes de aluminio 60 dientes	\$40.00	2	\$80.00
Correas dentadas (200 dientes)	\$50.00	2	\$100.00
Correas dentadas (900 dientes)	\$100.00	1	\$100.00
MDF para eslabones y base	\$200.00	1	\$200.00
Tornillos, tuercas rondanas de varias medidas	\$2.00	30	\$60.00
Tornillo sin fin	\$17.00	1	\$17.00
Fuente de alimentación (Lanix)	\$250.00	1	\$250.00
TOTAL			\$3473.00

Tabla 2 - Materiales y costos

7. Diseño y análisis del brazo antropomórfico

7.1 Diseño del brazo en SolidWorks

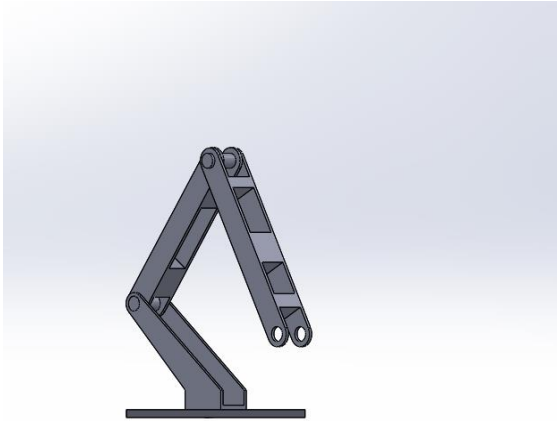


Ilustración 4 - Diseño en CAD

Es importante en el desarrollo saber los recursos necesarios para manejar las propiedades físicas del brazo.

El brazo antropomórfico consta de tres grados de libertad, los cuales se ha de mostrar en las siguientes imágenes.

1. Rotación en el eje de la base.
2. Movimiento de codo.
3. Movimiento de antebrazo.

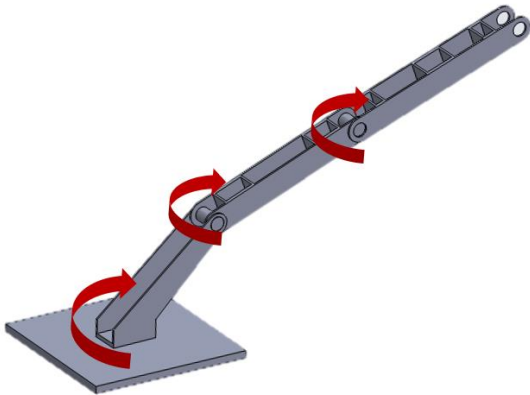


Ilustración 5 - Grados de libertad

7.1.1 Planos del brazo

A continuación, se muestran los planos para la elaboración de la estructura del brazo antropomórfico.

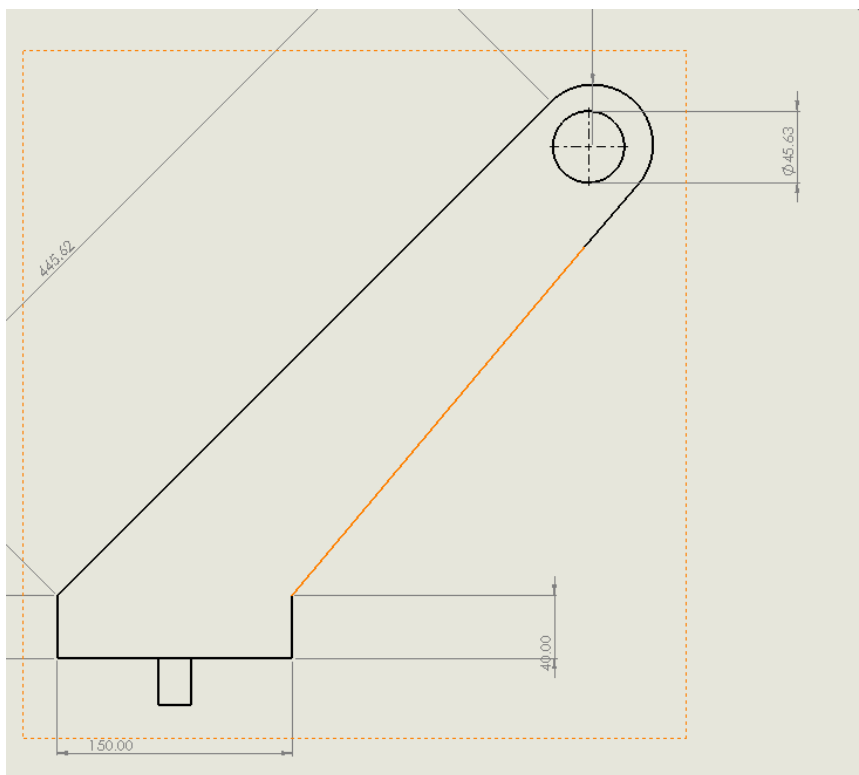


Ilustración 6 - Plano eslabón 1 vista lateral

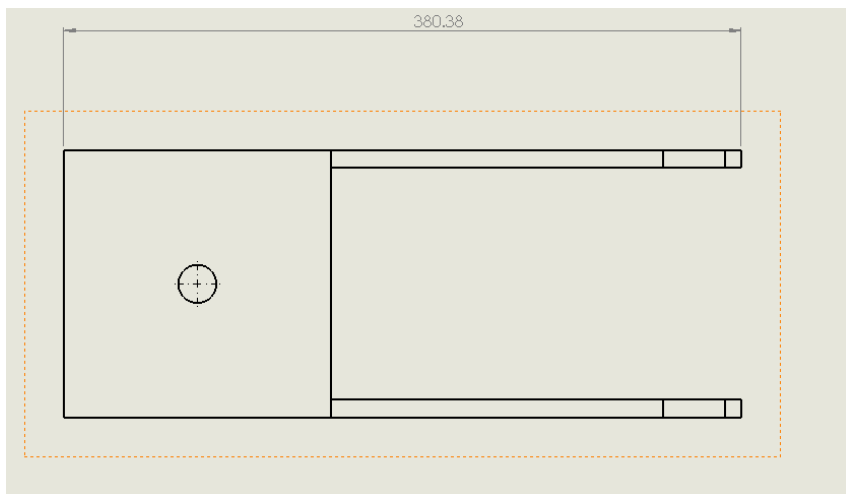


Ilustración 7 - Plano eslabón 1 vista superior

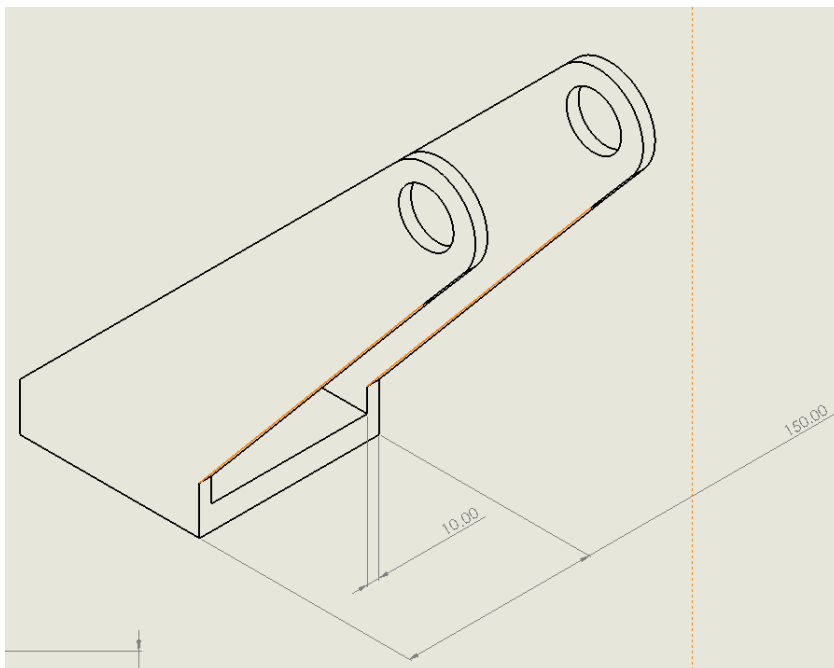


Ilustración 8 - Plano eslabón 1 vista isométrica

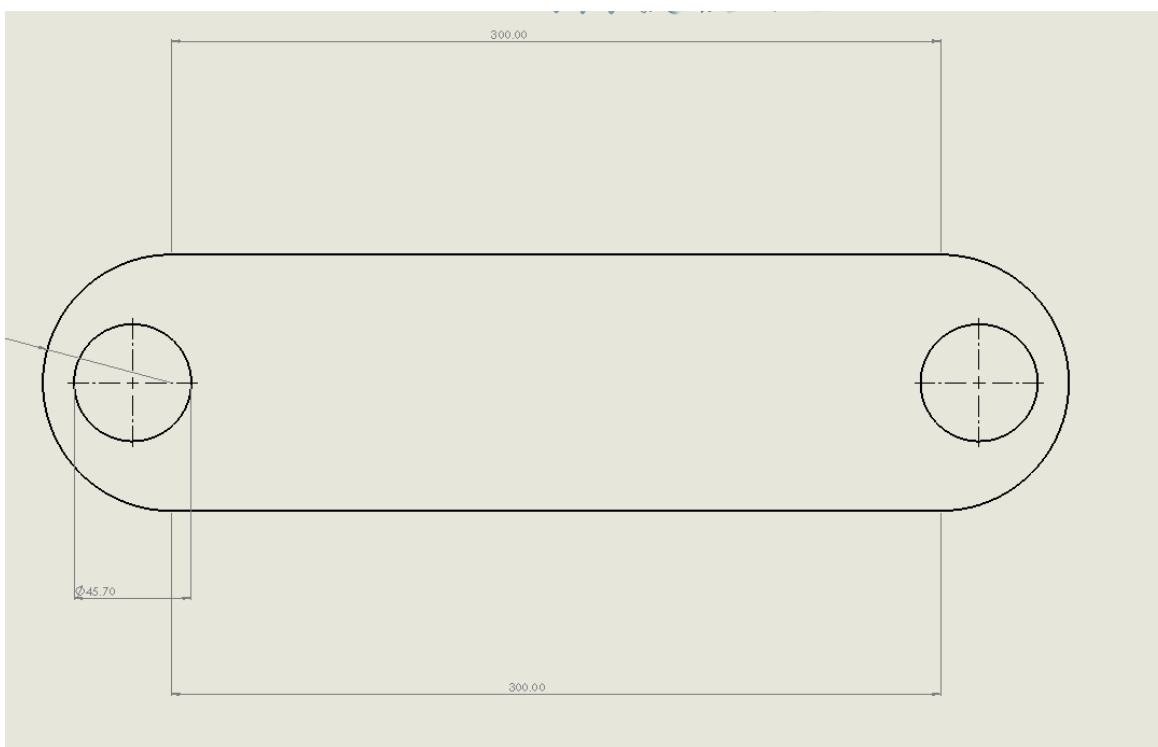


Ilustración 9 - Plano eslabón 2 vista lateral

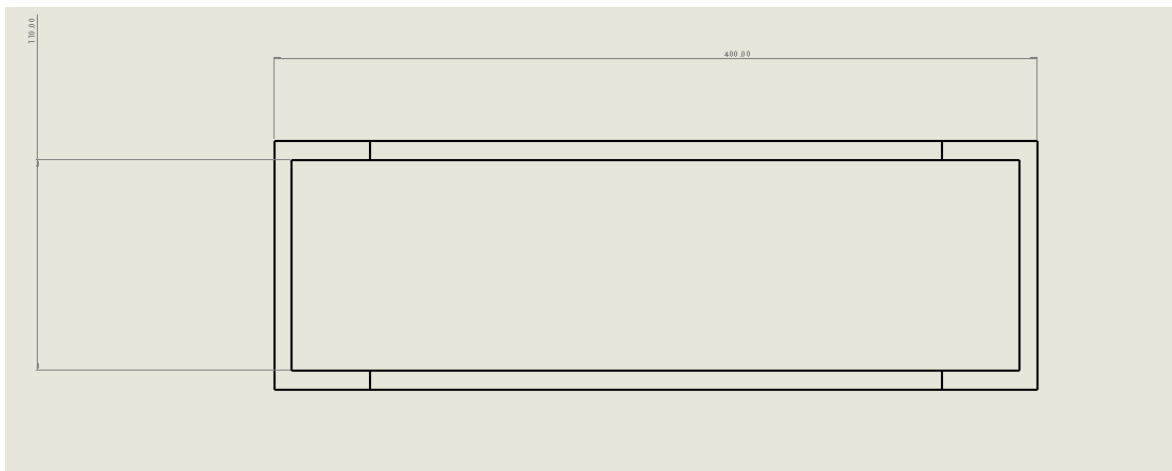


Ilustración 10 - Plano eslabón 2 vista superior

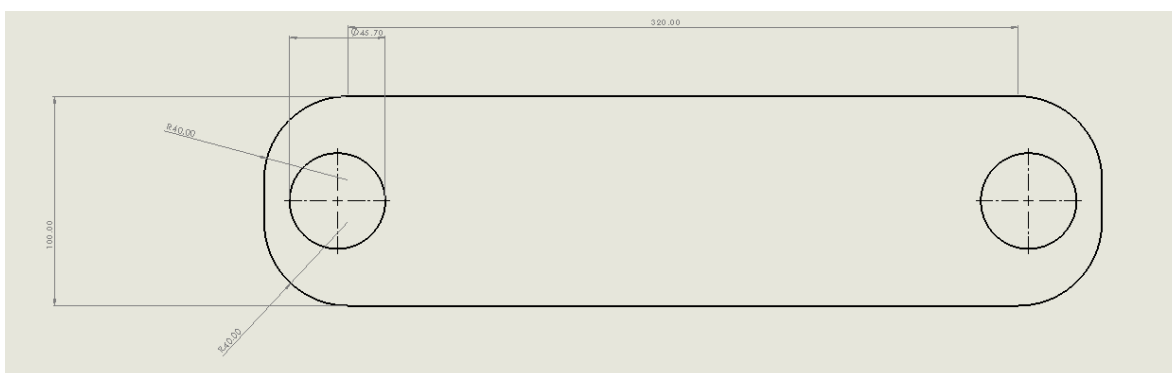


Ilustración 11 - Plano eslabón 3 vista lateral

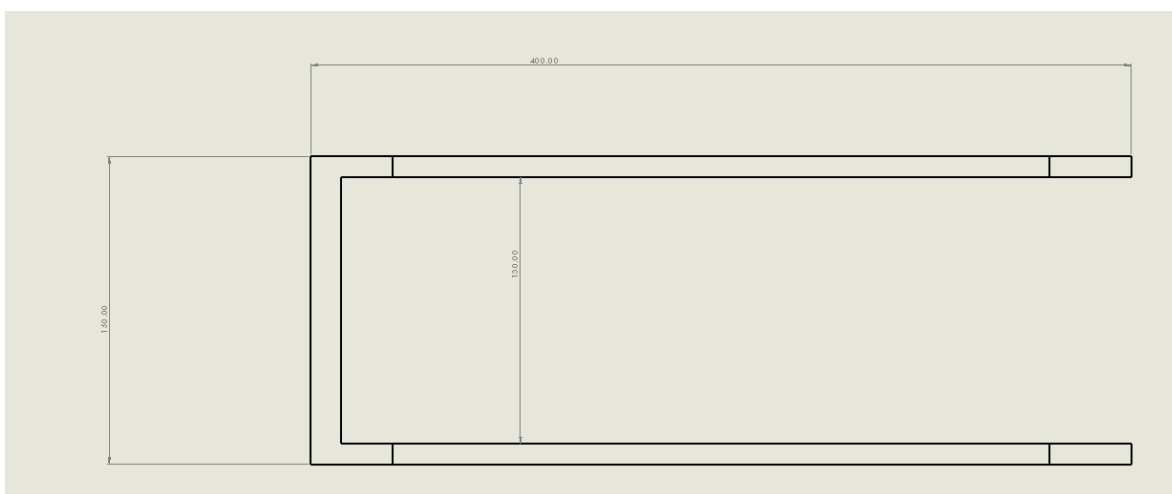


Ilustración 12 - Plano eslabón 3 vista superior

7.1.2 Longitud y carga

- La longitud de operación es de 1 metro de longitud, desde la base hasta el extremo del efector.
- La carga en el efector final del brazo que se requiere soportar es de 300gr.

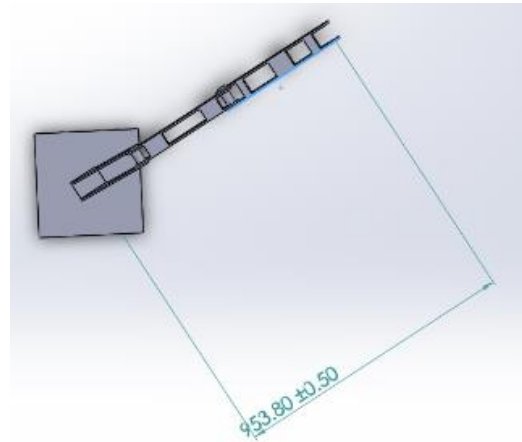


Ilustración 13 - Dimensiones

7.2 Análisis en Ansys

Previo al corte del MDF utilizado para la estructura del brazo se realizó el análisis de esfuerzos para determinar si la selección del material y la morfología eran aptas para soportar las cargas a la que estaría expuesto el brazo, en donde los resultados obtenidos mediante el software de ansys versión 18.1 fueron los siguientes con respecto a la carga de 300 gramos previamente planteada.

7.2.1 Solución (representación gráfica)

En la primera solución se obtiene la deformación total la cual representa que el material sufrirá un estiramiento que va desde los 0.000936 milímetros hasta los 0.00842 milímetros lo cual representa una deformación aceptable en la que la estructura será capaz de trabajar con la carga máxima de 300 gramos.

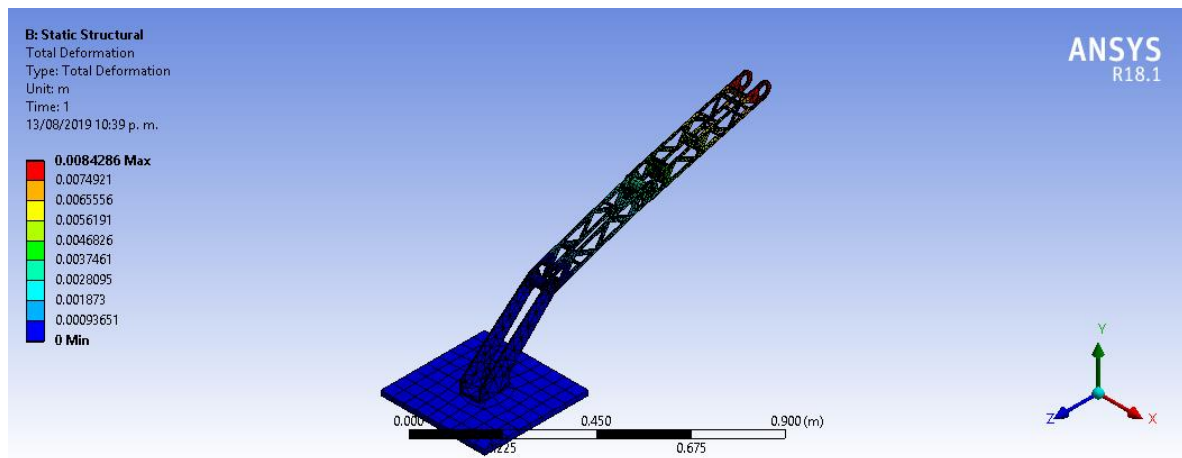


Ilustración 14 - Deformación total

Como segundo resultado se muestra el estrés equivalente en el cual se observa la presión que se ejerce en las distintas partes del brazo, donde la presión máxima esta dentro de las especificaciones que el material de la estructura del brazo tolera.

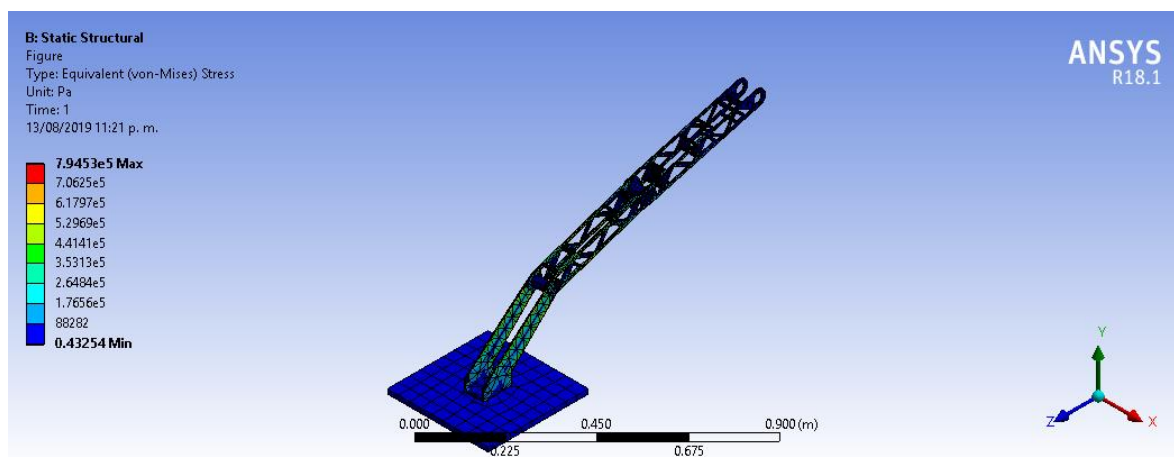


Ilustración 15 - Estrés equivalente

8. Programación

8.1 Programa de control del microcontrolador

El siguiente código se utilizó para programar el microcontrolador FRDM KL25Z, el cual llevaría incorporada la librería de ROS para poder comunicarse con el sistema de ROS en el portátil. A continuación, se explican las líneas más importantes del algoritmo en donde el código completo se puede encontrar en el anexo 3:

Al inicio del código se manda llamar a las librerías que se requerirán, la primera librería es mbed la cual nos permitirá el manejo de los pines de propósito general del microcontrolador y las funciones de programación básicas.

La segunda librería sirve para utilizar el sistema de ROS en el microcontrolador para de este modo poder suscribirse y publicar mensajes.

La tercera librería hace referencia al tipo de mensajes que se utilizaran, en este caso son de tipo twist debido que nos permite mandar máximo 6 datos por mensaje, en donde solo hemos utilizado 3 de ellos.

La cuarta librería también hace referencia a un tipo de mensaje que se utilizará en el código, el mensaje tipo String sirve para publicar la posición del brazo a través de la lectura de los potenciómetros.

```
#include "mbed.h"
#include <ros.h>
#include <geometry_msgs/Twist.h>
#include <std_msgs/String.h>
```

Luego, mediante la siguiente línea se nombra al nodo con el que vamos a trabajar para poder suscribirnos o publicar un mensaje.

```
ros::NodeHandle nh;
```

Con las siguientes líneas podemos especificar la función a realizar por el pin dentro de los paréntesis, ya sea como función de entrada o salida de una señal.

```
DigitalOut step(D2);
AnalogIn potbase(A0);
```

Este comando nos permite utilizar los mensajes tipo String y nombrar una variable para ser utilizada.

```
std_msgs::String str_msg_dat;
```

La siguiente línea crea un publicador con nombre pubmon en donde se utiliza el tipo de mensaje String.

```
ros::Publisher pubmon("pubmon", &str_msg_dat);
```

Con la siguiente función se crea la variable con mensaje de tipo twist llamada msg la cual recibirá los valores en grados desde la terminal del portátil para mover el robot.

```
void messageCb( const geometry_msgs::Twist& msg)
{ ...
```

La siguiente línea sirve para que el nodo pueda suscribirse a mensajes de tipo twist nombrados cmd_ang.

```
ros::Subscriber<geometry_msgs::Twist> sub("cmd_ang", messageCb);
```

Dentro del proceso principal se inicia el nodo y los modos de suscriptor y publicador de mensajes configurados anteriormente.

```
int main()
{
    nh.initNode();
    nh.subscribe(sub);
    nh.advertise(pubmon);
```

Y finalmente se publica el mensaje de tipo String al haber hecho la lectura de los potenciómetros

```
pubmon.publish( &str_msg_dat );
nh.spinOnce();
```

8.1.1 Comandos en ROS para el control del brazo

1. Como primer comando se abre el nodo maestro de ROS en una terminal nueva:

```
$ roscore
```

2. Al haber conectado el microcontrolador al equipo se procede a dar privilegios de escritura y lectura al mismo:

```
$ sudo chmod 666 /dev/ttyACM0
```

3. Después se prosigue a abrir el nodo de comunicación serial de Python para comenzar con la comunicación:

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

4. Al abrir el nodo ya es posible mandar mensajes de tipo twist al microcontrolador para controlarlo, en donde las letras a, b y c son el numero de grados que se desea mover a cada articulación, con un máximo de 220°.

```
$ rostopic pub /cmd_ang geometry_msgs/Twist '[0.0, 0.0, 0.0]' '[a, b, c]'
```

5. Se abre otra terminal para visualizar la posición en grados del brazo:

```
$ rostopic echo chatter
```

8.2 Programa de control difuso

Dentro del mismo sistema se incorporó un control difuso al brazo antropomórfico en donde se visualizaría mediante tres leds de colores rojo, amarillo y verde el nivel de esfuerzo que estaría soportando el brazo, debido a que conocemos que cuando el brazo esta completamente estirado se realiza el mayor esfuerzo posible. A continuación, se explican los aspectos mas importantes del algoritmo de control, el cual se puede encontrar completo en el anexo 4.

Como base se incluyen las librerías, la librería numpy sirve para realizar cálculos vectoriales, la librería skfuzzy para realizar el control difuso y la serial para recibir los datos del microcontrolador.

```
import numpy as np
import skfuzzy as fuzz
import time
import RPi.GPIO as GPIO
```

```
import serial
```

Se configura el puerto serial y la velocidad de transferencia.

```
frdm = serial.Serial('/dev/ttyACM0', baudrate=57600, timeout=1.0)
```

Se establecen la función de los pines GPIO.

```
GPIO.setup(13, GPIO.OUT)  
pwm2 = GPIO.PWM(13, 100)
```

Se realiza la lectura del puerto serial.

```
vals[0] = frdm.readline()
```

La siguiente línea sirve para crear un vector dentro del rango de 0 a 5 con intervalos de valor 0.1.

```
pot1_x = np.arange(0, 5.05, 0.1)
```

El siguiente comando es para fuzzificar las variables de entrada y salida

```
voltaje_pot1_lo = fuzz.trimf(pot1_x, [0,0,2.5])
```

Con la siguiente línea podemos configurar las funciones de pertenencia de las variables.

```
voltaje_pot1_nivel_lo = fuzz.interp_membership(pot1_x,voltaje_pot1_lo, pot1)
```

Después, se procede a calcular el valor mínimo recibido.

```
active_rule1 = np.fmin(voltaje_pot_nivel_lo,voltaje_foto_nivel_lo)
```

Para que con lo anterior se pueda obtener la salida al calcular el valor mínimo de nuevo.

```
control_activation_1 = np.fmin(active_rule1,intensidad_led_lo)
```

El siguiente comando es una función específica de la librería scikit-fuzzy para calcular el centroide.

```
control_value = fuzz.defuzz(intensidad_led_x,aggregated,'centroid')
```

9. Resultados

Al haber concluido con el cronograma de diseño, ensamble y programación del brazo antropomórfico hemos obtenido los resultados esperados y definidos inicialmente, los cuales comprenden la estructura del brazo antropomórfico acorde a los diseños realizados y el control por cinemática directa del robot utilizando el sistema de ROS, presentando como evidencia la siguiente ilustración:



Ilustración 16 - Producto final

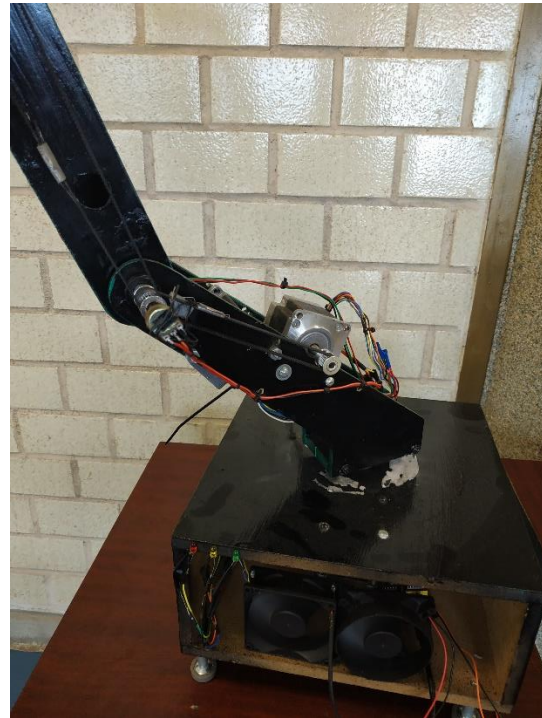


Ilustración 17 - Producto final (acercamiento)

10. Conclusiones

Juan Ramírez: El desarrollo de un robot antropomórfico es difícil para un equipo pequeño, sin presupuesto y cursando otras materias, pero el proyecto lo pudimos realizar y hacer funcionar a pesar de lo difícil que es para mi entender el sistema ROS aunque lo supe utilizar.

Es una sensación satisfactoria ver como resultado el robot moviéndose con comandos realizados por medio de ROS. Con esta experiencia siento que podre mejorar el diseño del robot y el sistema de control para futuras aplicaciones que tenga en los próximos proyectos o en algún trabajo futuro que tenga.

Jose Navarro: El desarrollo de este proyecto dio como experiencia que para que el producto o resultados sean los esperados se debe pasar por errores como pasaron en este, un error seria de diseño ya que al principio se puede pensar que el diseño es el correcto a como se pensó en el equipo, pero al momento de desarrollarlo en físico nos dimos cuenta de las ideas de diseño tenían errores y se tenían que modificar, otro de los errores que se presentaron fue en el posicionamiento de motores ya que influyen mucho para el movimiento de los eslabones eso era algo que no se tenia mucho en cuenta.

Referente a la programación el uso de ROS parece sencillo debido a que por medio de la comunicación serial y los comandos funcionaria el brazo, pero esto no es así ya que en esto intervenían librerías esenciales para la comunicación y si estas no se implementaban de manera correcta no funcionaria, así como también intervenía el puerto USB ya que se tenían que dar permisos para conectar el micro con ROS.

Marco Lozano: La realización del proyecto nos dejó experiencias y conocimientos en cuanto a la construcción y programación del mismo, ya que al iniciar un proyecto desde cero y al ir transcurriendo el tiempo fuimos identificando las limitaciones y formas en que pudiésemos optimizar el brazo robótico.

En cuanto a la programación realizada en ROS percibimos su utilidad al ser trabajado por nodos en donde el mal funcionamiento de un nodo no afecta a los demás, aspecto de gran ayuda para reconocer con mayor precisión la localización de un posible nodo que pueda fallar.

Jessica Lozada: La fabricación del brazo fue muy interesante y enriquecedora para el equipo. Además, fue el proyecto más completo.

En primer punto fue el área mecánica, existieron varios problemas con el peso de los motores. Por lo que nuestro compañero Juan, logró cambiar varios elementos y la posición de los motores para lograr un movimiento estable.

Me parece que cada persona del equipo, se desarrolló en su mejor área que es lo primordial de un equipo, y aprendimos a utilizar software que nos ayuda a posicionar nuestro robot como ROS.

Me hubiera gustado también, saber un poco sobre los cálculos para posicionar el brazo sin utilizar software como Ros, porque me agrada el modelado matemático, sin embargo, debo mencionar que estoy muy contenta con el resultado.

10.1 Sugerencias y aportes

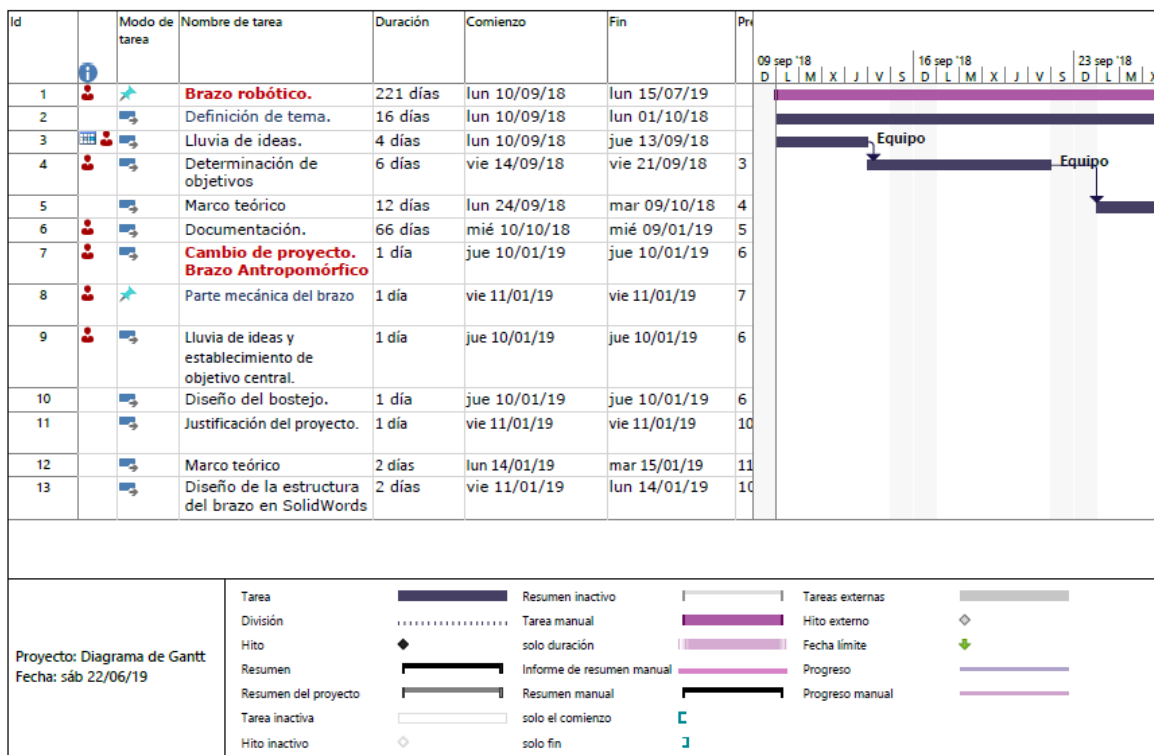
Como sugerencias en cuanto a la construcción mecánica de un brazo antropomórfico aconseja considerar las dimensiones del mismo y con esto elegir el mejor actuador o motor que pueda trabajar con el esfuerzo que se requiere en determinada articulación. Además, se aconseja, si es que se utilizan engranes y correas para la transmisión del movimiento, añadir un tensor para la correa para que al haber situaciones en que el brazo requiera gran esfuerzo los la correa no se salte los dientes del engrane.

11. Referencias







- R. González, Victor. (2003). Robots industriales. Recuperado 16 mayo, 2019, de http://platea.pntic.mec.es/vgonzale/cyr_0204/ctrl_rob/robotica/industrial.htm
- Brazo robótico. (2018, 7 noviembre). Recuperado 16 mayo, 2019, de https://es.wikipedia.org/wiki/Brazo_rob%C3%B3tico
- ROS.org. (2015, 20 noviembre). Recuperado 20 junio, 2019, de http://wiki.ros.org/rosterial_mbed/Tutorials

Anexos

Anexo 1: Diagrama de Gantt



Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
14		Justificación matemática de los esfuerzos del brazo, en Ansys	14 días	mar 15/01/19	vie 01/02/19
15		Fabricación de los eslabones del brazo	12 días	lun 04/02/19	mar 19/02/19
16		Compra de motores y materiales.	6 días	mié 20/02/19	mié 27/02/19
17		Ensamble de brazo	9 días	jue 28/02/19	mar 12/03/19
18		Primeras pruebas del brazo ensamblado con Arduino.	5 días	mié 13/03/19	mar 19/03/19
19		Correcciones mecánicas del brazo.	6 días	mié 20/03/19	mié 27/03/19
20		Elaboración de reporte final.	2 días	jue 28/03/19	vie 29/03/19
21		Evaluación del proyecto por el docente.	6 días	lun 01/04/19	lun 08/04/19
22		Receso de actividades.	32 días	mar 09/04/19	mié 22/05/19
23		Diseño y simulación del brazo, con correcciones hechas.	7 días	jue 23/05/19	vie 31/05/19
24		Sistema de control	30 días	mar 04/06/19	lun 15/07/19
25		Instalación de ROS	5 días	jue 28/03/19	mié 03/04/19

id		Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
26			Diagrama de Gantt	2 días	jue 04/04/19	vie 05/04/19
27			Instalación de Blender	2 días	lun 03/06/19	mar 04/06/19
28			Importación de archivos CAD a Blender	3 días	mié 05/06/19	vie 07/06/19
29			Movimiento del brazo en Blender	3 días	lun 10/06/19	mié 12/06/19
30			Importación CAD a Blender a Gazebo	6 días	jue 13/06/19	jue 20/06/19

Anexo 2: Reporte generado

Units

TABLE 1

Unit System	Metric (m, kg, N, s, V, A) Degrees rad/s Celsius
Angle	Degrees
Rotational Velocity	rad/s
Temperature	Celsius

Geometry

TABLE 2
Model (B4) > Geometry

Object Name	Geometry
State	Fully Defined
Definition	
Source	C:\Users\Marco\Desktop\Ensamblaje1.IGS
Type	Iges
Length Unit	Meters
Element Control	Program Controlled
Display Style	Body Color
Bounding Box	
Length X	0.35 m
Length Y	0.52473 m
Length Z	1.117 m
Properties	
Volume	2.5424e-003 m ³
Mass	1.9068 kg
Scale Factor Value	1.
Statistics	
Bodies	7
Active Bodies	7
Nodes	13720

Mesh

Elements	4644
Mesh Metric	None

TABLE 8
Model (B4) > Mesh

Object Name	<i>Mesh</i>
State	Solved
Display	
Display Style	Body Color
Defaults	
Physics Preference	Mechanical
Relevance	0
Element Order	Program Controlled
Sizing	
Size Function	Adaptive
Relevance Center	Coarse
Element Size	Default
Initial Size Seed	Assembly
Transition	Fast
Span Angle Center	Coarse
Automatic Mesh Based Defeaturing	On
Defeature Size	Default
Minimum Edge Length	1.7194e-003 m
Quality	
Check Mesh Quality	Yes, Errors
Error Limits	Standard Mechanical
Target Quality	Default (0.050000)
Smoothing	Medium
Mesh Metric	None

Inflation	
Use Automatic Inflation	None
Inflation Option	Smooth Transition
Transition Ratio	0.272
Maximum Layers	5
Growth Rate	1.2
Inflation Algorithm	Pre
View Advanced Options	No
Advanced	
Number of CPUs for Parallel Part Meshing	Program Controlled
Straight Sided Elements	No
Number of Retries	Default (4)
Rigid Body Behavior	Dimensionally Reduced
Mesh Morphing	Disabled
Triangle Surface Mesher	Program Controlled
Topology Checking	No
Pinch Tolerance	Please Define
Generate Pinch on Refresh	No
Statistics	
Nodes	13720
Elements	4644

Static Structural

TABLE 9
Model (B4) > Analysis

Object Name	<i>Static Structural (B5)</i>
State	Solved
Definition	
Physics Type	Structural
Analysis Type	Static Structural

Solver Target	Mechanical APDL
Options	
Environment Temperature	22. °C
Generate Input Only	No

TABLE 11
Model (B4) > Static Structural (B5) > Loads

Object Name	Fixed Support	Force
State	Fully Defined	
Scope		
Scoping Method	Geometry Selection	
Geometry	1 Face	
Definition		
Type	Fixed Support	Force
Suppressed	No	
Define By		Components
Coordinate System		Global Coordinate System
X Component		0. N (ramped)
Y Component		-3. N (ramped)
Z Component		0. N (ramped)

Solution

TABLE 14
Model (B4) > Static Structural (B5) > Solution (B6) > Results

Object Name	Total Deformation	Equivalent Elastic Strain	Equivalent Stress
State	Solved		
Scope			
Scoping Method	Geometry Selection		
Geometry	All Bodies		
Definition			
Type	Total Deformation	Equivalent Elastic Strain	Equivalent (von-Mises) Stress
By	Time		
Display Time	Last		
Calculate Time History	Yes		
Identifier			
Suppressed	No		
Results			
Minimum	0. m	2.248e-009 m/m	0.43254 Pa
Maximum	8.4286e-003 m	7.2466e-004 m/m	7.9453e+005 Pa
Minimum Occurs On	Part 7		
Maximum Occurs On	Part 1	Part 2	

Information		
Time	1. s	
Load Step	1	
Substep	1	
Iteration Number	1	
Integration Point Results		
Display Option		Averaged
Average Across Bodies		No

Anexo 3: Código de programación del microcontrolador

```
#include "mbed.h"           //Libreria de mbed para controlar el micro
#include <ros.h>             //Libreria para hacer funcionar ros en el micro
#include <geometry_msgs/Twist.h> //Libreria para mensajes de tipo twist
#include <std_msgs/String.h>  //Libreria para mensajes de tipo string

ros::NodeHandle nh;        //Se nombra nh al nodo a utilizar

DigitalOut step(D2);       //Declaracion de los pines
DigitalOut dir(D5);        //como salida digital
DigitalOut en(D8);         //que controlaran
DigitalOut step1(D3);      //los drivers de los motores
DigitalOut dir1(D6);
DigitalOut step2(D4);
DigitalOut dir2(D7);

AnalogIn potbase(A0);      //Declaracion de los pines
AnalogIn potl1(A1);        //como entrada analogica
AnalogIn potl2(A2);        //que haran las lecturas de los potenciómetros

std_msgs::String str_msg_dat; //Se nombra una variable de tipo string
ros::Publisher pubmon("pubmon", &str_msg_dat); //Se configura el nodo como publicador
char bufdatos[50]="";       //Cadena de caracteres para el publicador

float stepDelay = 0.0016;   //Se nombran las variables a utilizar
float angbase = 0;          //en el codigo y se inicializan o
float angl1 = 0;            //establecen los valores de ellas
float angl2 = 0;
float angbasegrado = 0;
float angl1grado = 0;
float angl2grado = 0;

void messageCb( const geometry_msgs::Twist& msg) //Funcion para el nodo suscriptor
{
    angbase = potbase.read(); //Lectura del potenciómetro de la base
    angl1 = potl1.read();
    angl2 = potl2.read();
    angbasegrado = angbase * 270; //El valor se convierte a grados
    angl1grado = angl1 * 270;
    angl2grado = angl2 * 270;

    int base = msg.angular.x; //Se guarda en una variable el
    int link1 = msg.angular.y; //valor recibido por el mensaje
    int link2 = msg.angular.z; //de tipo twist

    if(base > 0 && base < 170) { //Si el valor recibido esta dentro del rango
        base = base * 9.3; //Conversion debido a la relacion de engranes del brazo
        for (int x = 0; x < base; x++) { //Ciclo para mandar el numero de pasos
            step=1; //que se movera el motor base
            dir=0;
            en=0;
            wait(stepDelay); //Retraso para ajustar la velocidad del motor
            step=0;
        }
    }
    if(base < 0 && base > -170) { //Si el valor recibido es negativo
        base = base * -1; //el motor girara en sentido contrario
        base = base * 9.3;
        for (int x = 0; x < base; x++) {
            step=1;
            dir=1;
            en=0;
        }
    }
}
```



```

        wait(stepDelay);
        step=0;
    }
}
if(link1 > 0 && link1 < 170) {
    link1 = link1 * 34.87;
    for (int x = 0; x < link1; x++) {
        step1=1;
        dir1=0;
        en=0;
        wait(stepDelay);
        step1=0;
    }
}
if(link1 < 0 && link1 > -170) {
    link1=link1*-1;
    link1 = link1 * 34.87;
    for (int x = 0; x < link1; x++) {
        step1=1;
        dir1=1;
        en=0;
        wait(stepDelay);
        step1=0;
    }
}
if(link2 > 0 && link2 <170) {
    link2 = link2 * 27.9;
    for (int x = 0; x < link2; x++) {
        step2=1;
        dir2=0;
        en=0;
        wait(stepDelay);
        step2=0;
    }
}
if(link2 < 0 && link2 > -170) {
    link2 = link2*-1;
    link2 = link2 * 27.9;
    for (int x = 0; x < link2; x++) {
        step2=1;
        dir2=1;
        en=0;
        wait(stepDelay);
        step2=0;
    }
}
}

ros::Subscriber<geometry_msgs::Twist> sub("cmd_ang", messageCb);    //Declaracion del suscriptor

int main()
{
    nh.initNode();           //Inicializacion del nodo
    nh.subscribe(sub);       //Inicia la suscripcion de mensajes
    nh.advertise(pubmon);    //Inicia la publicacion de mensajes

    while (1) {
        angbase = potbase.read();    //Lectura de los potenciómetros
        angl1 = potl1.read();
        angl2 = potl2.read();
        angbasegrado = angbase *270;    //Conversion a grados
        angl1grado = angl1*270;
        angl2grado = angl2*270;
    }
}

```

```
printf(bufdatos,"Esl1: %0.2f -- Esl2: %0.2f -- Esl3: %0.2f",angbasegrado,angl1grado,angl2grado);
str_msg_dat.data = bufdatos;          //La conversion se guarda en la variable str_msg_dat
pubmon.publish( &str_msg_dat );      //La variable anterior se publica

nh.spinOnce();          //Instruccion para que el nodo repita el ciclo
wait_ms(1);             //Retraso de un segundo
}
}
```

Anexo 4: Código de programación en Python de la raspberry

```
import numpy as np                                #Libreria para utilizar funciones matematicas
import skfuzzy as fuzz                            #Libreria para el control difuso
import time                                       #Libreria para controlar el reloj del procesador
import RPi.GPIO as GPIO                         #Libreria para utilizar los pines gpio
import serial                                    #Libreria para utilizar puerto serial

frdm = serial.Serial('/dev/ttyACM0', baudrate=57600, timeout=1.0) #Configuracion del puerto serial en variable frdm

GPIO.setwarnings(False)                        #Desactivacion de las alertas
GPIO.setmode(GPIO.BCM)
GPIO.setup(12, GPIO.OUT)                       #Se asignan la funcion de los pines como salida
pwm1 = GPIO.PWM(12, 100)                       #De tipo pwm
GPIO.setup(13, GPIO.OUT)
pwm2 = GPIO.PWM(13, 100)
GPIO.setup(16, GPIO.OUT)
pwm3 = GPIO.PWM(16, 100)

while True:

    vals = [0]*3                                #Vector para guardar datos del serial
    for i in range(3):                          #Inicio del ciclo for
        vals[0] = frdm.readline()               #Lectura del primer potenciometro
        vals[1] = frdm.readline()               #Lectura del segundo potenciometro
        vpot1 = vals[0]                         #El vector se guarda en otra variable
        vpot2 = vals[1]
        spot1 = float(vpot)                     #El dato se convierte a flotante
        spot2 = float(vldr)
        pot1 = spot1*0.98                       #Se multiplica para no desbordar el valor desfuzzificado
        pot2 = spot2*0.98

    time.sleep(0.05)                            #Retraso de 50 ms
    pot1_x = np.arange(0, 5.05, 0.1)            #Se crea vector para especificar el rango en x
    pot2_x = np.arange(0, 5.05, 0.1)
    intensidad_led_x = np.arange(0, 5.05, 0.1)

    voltaje_pot1_lo = fuzz.trimf(pot1_x, [0,0,2.5]) #Se fuzzifican las variables de entrada y salida
    voltaje_pot1_md = fuzz.trimf(pot1_x, [0,2.5,5])
    voltaje_pot1_hi = fuzz.trimf(pot1_x, [2.5,5,5])
    voltaje_pot2_lo = fuzz.trimf(pot2_x, [0,0,2.5])
    voltaje_pot2_md = fuzz.trimf(pot2_x, [0,2.5,5])
    voltaje_pot2_hi = fuzz.trimf(pot2_x, [2.5,5,5])
    intensidad_led_lo = fuzz.trimf(intensidad_led_x, [0,0,2.5])
    intensidad_led_md = fuzz.trimf(intensidad_led_x, [0,2.5,5])
    intensidad_led_hi = fuzz.trimf(intensidad_led_x, [2.5,5,5])

    voltaje_pot1_nivel_lo = fuzz.interp_membership(pot1_x, voltaje_pot1_lo, pot1) #Se hace la lectura en las funciones
    voltaje_pot1_nivel_md = fuzz.interp_membership(pot1_x, voltaje_pot1_md, pot1) #de pertenencia
    voltaje_pot1_nivel_hi = fuzz.interp_membership(pot1_x, voltaje_pot1_hi, pot1)
    voltaje_pot2_nivel_lo = fuzz.interp_membership(pot2_x, voltaje_pot2_lo, pot2)
    voltaje_pot2_nivel_md = fuzz.interp_membership(pot2_x, voltaje_pot2_md, pot2)
    voltaje_pot2_nivel_hi = fuzz.interp_membership(pot2_x, voltaje_pot2_hi, pot2)

    active_rule1 = np.fmin(voltaje_pot_nivel_lo, voltaje_foto_nivel_lo) #Se crean las reglas de comportamiento
    control_activation_1 = np.fmin(active_rule1, intensidad_led_lo)      #Funcion de numpy para obtener el valor
    minimo

    active_rule2 = np.fmin(voltaje_pot_nivel_lo, voltaje_foto_nivel_md)
    control_activation_2 = np.fmin(active_rule2, intensidad_led_lo)

    active_rule3 = np.fmin(voltaje_pot_nivel_lo, voltaje_foto_nivel_hi)
    control_activation_3 = np.fmin(active_rule3, intensidad_led_md)
```

```

active_rule4 = np.fmin(voltaje_pot_nivel_md,voltaje_foto_nivel_lo)
control_activation_4 = np.fmin(active_rule4,intensidad_led_lo)

active_rule5 = np.fmin(voltaje_pot_nivel_md,voltaje_foto_nivel_md)
control_activation_5 = np.fmin(active_rule5,intensidad_led_md)

active_rule6 = np.fmin(voltaje_pot_nivel_md,voltaje_foto_nivel_hi)
control_activation_6 = np.fmin(active_rule6,intensidad_led_hi)

active_rule7 = np.fmin(voltaje_pot_nivel_hi,voltaje_foto_nivel_lo)
control_activation_7 = np.fmin(active_rule7,intensidad_led_md)

active_rule8 = np.fmin(voltaje_pot_nivel_hi,voltaje_foto_nivel_md)
control_activation_8 = np.fmin(active_rule8,intensidad_led_hi)

active_rule9 = np.fmin(voltaje_pot_nivel_hi,voltaje_foto_nivel_hi)
control_activation_9 = np.fmin(active_rule9,intensidad_led_hi)

c1 = np.fmax(control_activation_1, control_activation_2)    #Funcion de numpy para obtener el valor maximo
c2 = np.fmax(control_activation_3, control_activation_4)
c3 = np.fmax(control_activation_5, control_activation_6)
c4 = np.fmax(control_activation_7, control_activation_8)
c5 = control_activation_9
c6 = np.fmax(c2, c3)
c7 = np.fmax(c3, c4)
c8 = np.fmax(c4, c5)
c9 = np.fmax(c5, c6)
aggregated = np.fmax(c1,c9)

control_value = fuzz.defuzz(intensidad_led_x,aggregated,'centroid')    #Funcion de scikit para obtener el centroide
c = control_value-1
a=c*30                                #Multiplicacion para mandar al pwm
b=abs(a)                              #Se obtiene el valor absoluto para evitar
errores
if b>0 and b<30:                      #Si el valor desfuzzificado esta dentro del primer tercio
    pwm1.start(b)                     #se activa el led verde
    pwm2.start(0)
    pwm3.start(0)
if b>30 and b<60:                     #Si esta dentro del segundo se activa el led amarillo
    pwm2.start(b)
    pwm1.start(0)
    pwm3.start(0)
if b>60 and b<100:                    #Y dentro del tercero se activa el led rojo
    pwm3.start(b)
    pwm1.start(0)
    pwm2.start(0)

print 'Valor PWM LED: ',b             #Se imprime el valor del pwm

```