Diseño e implementación de dispositivo de inspección microscópica de amplias superficies

Proyecto de Investigación

Por

Carlos Alfredo Catalán Catalán

En cumplimiento a los requerimientos para la obtención de la Especialidad de Tecnólogo en Mecatrónica

Revisor académico: Dr Horacio Canales Siller

Santiago de Querétaro, Qro., México, Agosto 2020.

ESPECIALIDAD TECNÓLOGO EN MECATRÓNICA

Coordinador Académico
CIDESI

Los abajo firmantes, miembros del Comité Tutorial del estudiante **Carlos Alfredo Catalán Catalán**, una vez leído y revisado el informe de la práctica de entrenamiento Industrial, titulado **"Diseño e implementación de dispositivo de inspección microscópica de amplias superficies"**, aceptamos que el referido informe revisado y corregido sea presentado por el estudiante para concluir su plan de estudios como **Especialista Tecnólogo en Mecatrónica**.

Y para que así conste firmo la presente a los trece días del mes de Agosto del año dos mil veinte.

Dr. Horacio Canales Siller

Tutor Académico

Índice de contenido

Índice de figuras

Introducción

La mecatrónica es una rama de la ingeniería que busca unificar conocimientos de las áreas de mecánica, electrónica, informática y sistema de control con la finalidad de mejorar y simplificar los procesos industriales, o ciertas actividades cotidianas.

Una de esas actividades es el objetivo de este proyecto, la cual es la inspección microscópica, muy usada en la industrial para verificar la calidad de los productos, o en las ciencias biológicas para analizar microorganismo presentes en ciertos lugares, también muy usada en aspectos académicos. El microscopio se utiliza en toda actividad donde es necesario hacer visible los objetos que el ojo humano no es capaz por sí solo de visualizar.

La inspección de un objeto con microscopio conlleva algunas actividades manuales según el dispositivo que se cuente. Pero si hablamos de un dispositivo convencional es necesario realizar un ajuste manual del enfoque para observar claramente y con detalle el objeto, o bien si deseamos observar un área mayor a lo que el lente puede abarcar sería necesario tener que recorrer manualmente el lente para observar el área extra que deseamos. ¿Y qué tal si quisiéramos conservar la evidencia de esta área inspeccionada para un posterior análisis? Si contáramos con un microscopio digital podríamos tomar fotografías sobre el área pero aun así tendríamos que realizar ciertos ajustes manuales para poder cubrir toda el área deseada.

El objetivo de este proyecto es mapear microscópicamente determinadas áreas de forma automática y añadiendo la posibilidad de controlarlo vía remota; pero siendo un dispositivo de bajo costo y fácil de producir en comparación con los dispositivos más sofisticados de este tipo. ¿Pero de qué forma logramos esto? A través de componentes que no son muy especializados y que son de fácil acceso en para una persona ordinaria tales como motores a pasos 28BYJ-48, un micro ordenador Raspberry Pi 3B+, una impresora 3D y un microscopio digital económico es posible lograrlo.

La Raspberry nos ofrece un entorno de desarrollo muy amigable en lenguaje Python, siendo este de código abierto y contando con mucho desarrollo por parte de una comunidad extensa. Es

posible controlar el movimiento del microscopio en el espacio montándolo sobre un robot cartesiano, con ejes impulsados por motores a paso; también la Raspberry y Python nos ofrece la posibilidad de controlar el microscopio con librerías ya existentes como es el caso de OpenCv, siendo también de código abierto y gratuito para desarrollos. Las imágenes recolectadas podemos montarlas en un servidor web para que personas vía remota puedan visualizar las áreas que son inspeccionados gracias a la librería Flask de Python. Y finalmente podemos reconstruir la superficie a partir de las fotos adquiridas.

Planteamiento del problema

Se deberá diseñar y construir un dispositivo mecánico portable que permita mover un microscopio digital en los 3 ejes del espacio, programar el control para dicho mecanismo e implementar un programa que permita reconstruir en una solo imagen el área inspeccionada.

Ya existen en el mercado dispositivos capaces de ofrecer estas funcionalidades sin embargo suelen ser equipos muy sofisticados que requieren estar en un lugar con condiciones muy controladas, y que tienen limitaciones en cuanto al tipo y áreas que pueden inspeccionar.

Hoy en día con el desarrollo de la tecnología de manufactura aditiva se abren muchas posibles aplicaciones para este campo, muchas de ellas seguramente fuera de laboratorios donde no será factible transportar un equipo de análisis sofisticado, es por eso que esta tecnología está pensada para ser portable y montada in-situ, dando la posibilidad de analizar la calidad de aplicaciones de manufactura aditiva justo después de llevarse a cabo.

Sabemos que existe un rezago tecnológico en nuestro país, la manufactura aditiva desarrollada es un esfuerzo por cerrar esta brecha, pero es necesario ofrecer productos de calidad que sean verificables, este proyecto acompaña a la manufactura aditiva en el viaje por ofrecer servicios tecnológicos de primer nivel.

Objetivos

Objetivo general

- Desarrollar un sistema mecatrónico para el montaje de un microscopio digital que permita escanear y mapear amplias superficies.

Objetivos particulares

- Diseño, impresión y ensamblaje de un mecanismo de posicionamiento en ejes X, Y y Z para el microscopio.
- Desarrollar un programa de control para el posicionamiento del microscopio.
- Integrar el mecanismo con el control electrónico.
- Implementar un programa de control del microscopio digital.
- Implementar un servidor web sobre la Raspberry que permita interactuar con el control del microscopio y transmita el video obtenido.
- Integrar el control del dispositivo con una interfaz gráfica remota.

Justificación

La brecha tecnológica de nuestro país es bien conocida debido a la falta de investigación en general de la industria, México es más bien un consumidor tecnológico; y es por esta razón que los productos de un desarrollo tecnológico más avanzado suelen ser bastantes caros e inasequibles para la industria. Con este proyecto se pretende cerrar un poco esta brecha, ya que al realizarse con componentes de fácil acceso y que son comunes en el mercado Méxicano, hacen que el proyecto sea más a asequible, también al desarrollarse con código abierto evitamos el tener que pagar licencias de software disminuyendo aún más los costos. Por estas razones es que se considera dicho proyecto como factible de realizarse, pues no existen inconvenientes que pueda afectar su funcionalidad.

Una vez obtenido el proyecto, tendremos el beneficio de contar con un equipo portable y adaptable a diversas superficies donde se requiera realizar reconstrucción microscopia de superficies.

Cualquier persona o institución que requiera un microscopio será un potencial usuario, pues en muchas ocasiones se requieren mayores prestaciones que las que un microscopio convencional puede aportar y al ser portable le confiere un uso más general. Puede ser en el departamento de calidad de cualquier empresa productiva, las escuelas o instituciones educativas, laboratorios privados, etc.

Antecedentes

Con los recientes avances tecnológicos solo la imaginación de cómo y dónde usar un microscopio son el límite, bueno casi. Pero podemos imaginarnos cientos de usos prácticos y de qué modo hacerlo, ahora ¿qué tal que al microscopio le integramos funcionalidades de otras áreas tecnológicas? Como la robótica para poder mover el microscopio a posiciones deseadas rápidamente y con precisión, y es obvio que este robot tendría que tener un controlador, por lo que ahora estaríamos utilizando conocimientos de las ciencias computacionales, pero que tal si no solo controla el movimiento, si no que controla automáticamente el enfoque gracias a una red neuronal, ahora ya hablamos de Inteligencia artificial, o más exacto de Deep Learning.

Microscopia

Según la Wikipedia la microscopía es,

"el conjunto de técnicas y métodos destinados a hacer visible los objetos de estudio que por su pequeñez están fuera del rango de resolución del ojo normal."

La historia nos dice que la microscopia inicia con Antonie van Leeuwenhoek (Holanda, 1632-1723) quien era un aficionado a pulir lentes, con lo cual se obtenía un cierto grado de aumento que le permitió por primera vez observar algunas bacterias. Posterior mente Zacharias Janssen desarrollo el microscopio compuesto, lo cual con los avances en la ciencia desemboco en los actuales y diversos microscopios. Con el desarrollo de la electrónica y la computación surgieron los microscopios digitales, estos a diferencia de los oculares, es una cámara la que visualiza la muestra en ves del ojo humano. Esto permitió que las imágenes capturas sean transmitidas a ordenadores o pantallas, lo cual conlleva a grandes posibilidades de procesamiento de la información.

Una cosa es clara, el microscopio llego para quedarse debido a las limitaciones del ojo humano, y estaría de más describir el por qué es importante para las ciencias.

Enfoque

El hacer enfocar un objeto significa hacer que los rayos de luz reflejados por dicho objeto coincidan con el foco de la lente, aunque para dispositivos digitales el foco sería el sensor. Y básicamente es hacer que el objeto que deseamos aparezca nítidamente en nuestra captura, es decir que sus bordes y superficies pueden distinguirse claramente.

Cuando se enfoca, no es como tal que se enfoque hacia nuestro objeto, sino más bien que se enfocará una cierta distancia, y todos los objetos que estén a esta distancia se verán nítidos, perdiendo definición aquellos que varía más su distancia respecto al enfoque.

Procesamiento de imágenes

El procesamiento de imágenes se realiza sobre ordenadores debido a la cantidad y complejidad de los cálculos necesarios para extraer información de ellas. Gracias a los modernos equipos de cómputo que poseen gran capacidad de procesamiento es que esta área ha visto un despertar de sí.

Todos los objetos absorben, reflejan o emiten cuantos de energía dependiendo de su longitud de onda, intensidad y tipo de radiación. Este tipo de radiación se define a partir de sus propiedades físicas dentro del espectro electromagnético. El ojo humano, por su parte, solo es capaz de detectar energía electromagnética en el espectro de luz visible, mientras que para los rayos X, la radiación ultravioleta, infrarroja o de microondas, es necesaria la construcción de detectores que puedan recabar esta información, ya sea de forma digital o analógica, para poder ser cuantificada y analizada (Pérez y Valente, 2018).

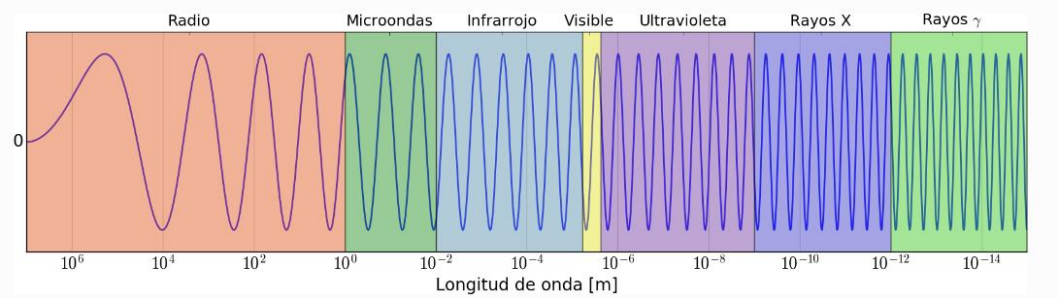

Figura 1 Esquema cualitativo del espectro electromagnético.

La representación digital de una imagen se hace a través de un *Bitmap*, que son básicamente arreglos de elementos (vectores o matrices) ordenados de un modo específico. Para el caso de

una imagen en 2 dimensiones, un *Bitmap* es una matriz de n * m elementos, donde cada elemento recibe el nombre de Pixel. Cada Pixel tendrá un valor numérico que determinara el color en esa posición.

Ahora bien, para el caso de imagen en escala de grises, cada pixel tiene un solo valor, la cual denota la intensidad del blanco o negro, pero si hablamos de imágenes a color en el espacio de color RGB cada pixel será un vector de 3 elementos donde cada elemento denota la tonalidad del rojo, verde y azul. Existen otros tipos de representaciones de color como por ejemplo el *CMYK* (cian, magenta, amarillo y negro), pero el más común de usar es el RGB. Todos los colores del rango visible pueden representarse como combinaciones RGB variando desde el [0, 0,0] que es el negro hasta [255, 255,255] que es el blanco.

Para analizar la información contenida en una imagen digital, es necesario aplicar operaciones sobre sus pixeles, cuestión que puede ser bastante complicada de entender a simple vista pues se requiere conocimientos de algebra vectorial, cálculo entre otras ramas de las matemáticas. Dado que este es un tema bastante estudiado, existen ya distribuciones de software capaces de realizar cálculos, uno de ellos es el conocido *OpenCv* que soporta una multitud de algoritmos relacionado a la visión por computadora y aprendizaje máquina. Una de las grandes ventajas es que es soportado en una amplia variedad de lenguajes de programación y es una distribución de *Open Source* para desarrollos académicos o comerciales. Es por esta razón que se utilizará las librerías de *OpenCV* para el procesamiento de imágenes captadas por el microscopio.

Raspberry Pi

Raspberry Pi (RPi) es un ordenador de tamaño compacto que soporta de forma muy parecida a una PC convencional las aplicaciones comunes que alguien podría darle a un ordenador de escritorio o laptop. Aparte de los puertos comunes que podemos hallar en una PC como los HDMI, Ethernet, USB, etc. encontramos una serie de pines para realizar conexiones a periféricos de bajo nivel, lo que la convierte en una poderosa herramienta para ejercer control sobre actuadores y sensores o para comunicarse con otros dispositivos mediante protocolos como I2C o SPI.

Existen diferentes modelos de *RPi* y estos varían en costo y capacidad de procesamiento, almacenamiento, etc.. También existen en el mercado otro tipo de microordenadores pero la *RPi* tiene una comunidad bastante grande de desarrolladores, y esta puede ser programada en *Python*

que a su vez, su comunidad de desarrollo es bastante grande. Por esta razón se ha optado por utilizar una *Raspberry Pi 3B+* para el control y procesamiento de este proyecto.

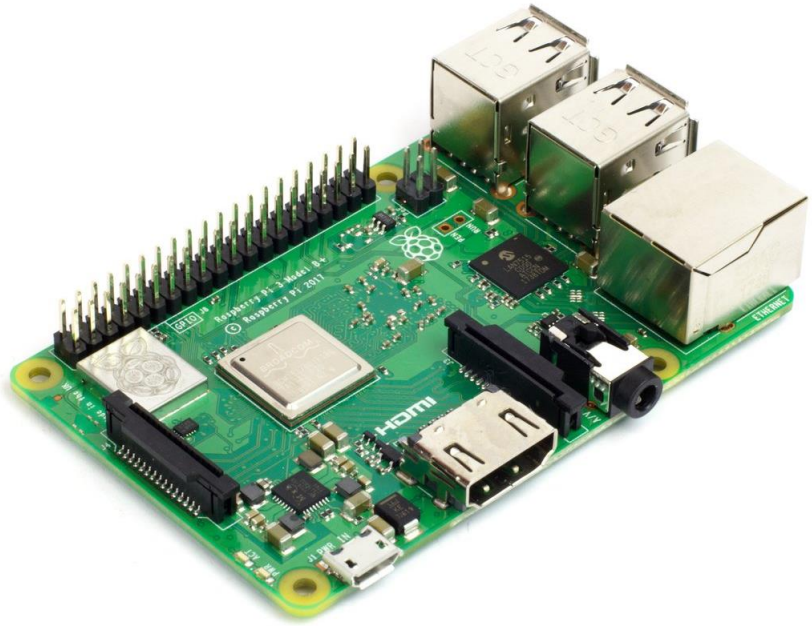

Figura 2 Raspberry Pi 3B+.

Robótica

Un robot es un dispositivo mecatrónico que puede ser programado para cumplir con cierta tarea sin necesidad de un operador humano.

Existen diversas arquitecturas de robots hoy en día, según la aplicación el tipo de robot podrá ser muy específico o puede darse el caso de que para cierta aplicación más de una arquitectura sea viable implementar. Sin embargo habría que considerar otros factores que afectan la selección de alguna arquitectura, como por ejemplo la complicidad de la dinámica y el control, el costo de construcción y mantenimiento, o el refaccionamiento por mencionar algunas.

Algunos ejemplos de arquitectura son los siguientes mostrados en la Figura 1 (Barrientos, Peñin, Balagues & Aracil, 2da Edición).

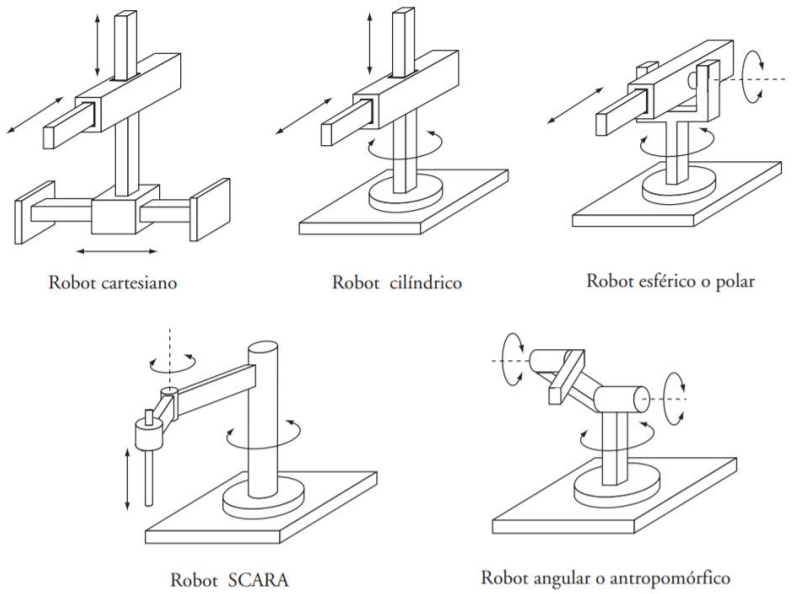

Figura 3 Arquitecturas más comunes de Robots.

Para nuestro proyecto, no necesitamos alcanzar ángulos de inclinación o rotativos, sino solo que el microscopio sea capaz de posicionarse en los 3 ejes del espacio de forma simple y precisa, es por eso que hemos optado por utilizar una arquitectura del robot cartesiano que proporciona movimientos lineales sobre los 3 ejes. Un robot cartesiano es un dispositivo sobre el cual los ejes están montado uno sobre otro. Es decir, el eje Z se monta sobre el eje Y, y a su vez este eje es montado sobre el X. Por esta razón es obvio que el eje X será el que mayor carga deberá mover, incluido el peso del microscopio propio. Esta arquitectura nos permite un control simple de manera lineal para cada eje, ahora habrá que seleccionar el actuador que impulse el movimiento.

Un actuador es un dispositivo capaz de transformar la energía de un tipo en energía de otro tipo con el fin de activar un proceso. Puede ser energía eléctrica, neumática, hidráulica, etc.

Lo más sencillo es utilizar algún actuador eléctrico, ya que la energía eléctrica es más fácil de obtener que cualquier otra y los actuadores eléctricos son más baratos y fáciles de conseguir. Los actuadores eléctricos son mejor conocidos como motores eléctricos.

Los motores eléctricos son máquinas que transforman la energía eléctrica en energía mecánica rotativa. Tiene múltiples ventajas respecto a otros tipos de motores, las cuales vienen siendo su economía, limpieza, comodidad y seguridad de funcionamiento (Carlos, 2011-2019).

Existen distintos tipos de motores eléctricos, cada uno con sus particularidades que pueden ser aprovechadas según la aplicación, pero de nuestro interés serán los motores a pasos, pues estos pueden ser controlados con precisión sin requerir un sistema de lazo cerrado.

Los motores de velocidad gradual o de pasos son motores especiales que se utilizan cuando el movimiento y la posición se tienen que controlar con precisión. Estos motores giran en pasos discretos, y cada paso corresponde a un pulso suministrado a uno de sus devanados de estator.

Dependiendo de su diseño un motor de pasos puede avanzar 90°, 45°, 18° o incluso una fracción de grado por pulso. Variando la velocidad de los pulsos se puede hacer que el motor avance muy lentamente, un paso a la vez o, que gire gradualmente a velocidades de 4000 rpm.

Los motores de pasos pueden girar en el sentido o contra el sentido de las manecillas del reloj, esto dependiendo de la secuencia de los pulsos que se apliquen a los devanados.

Un motor de este tipo usualmente es controlado por un microcontrolador. Este dispositivo puede configurarse para que cuente el número de pulsos suministrados en cada dirección, lo que permite conocer con precisión la posición del motor.

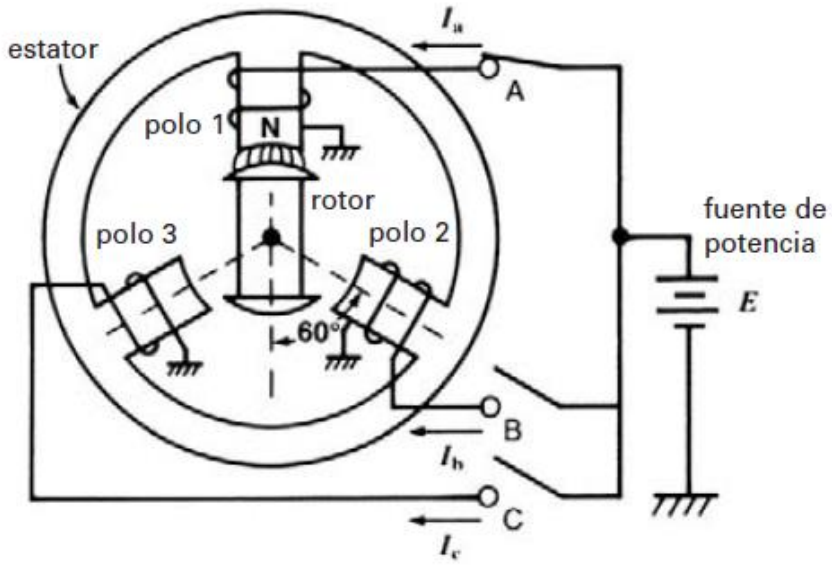

Figura 4 Motor a pasos elemental.

En la figura 4 se presenta el motor a pasos elemental. Consta de un estator con tres polos salientes y un rotor de 2 polos. Los devanados se controlan por los interruptores A, B, C.

Si los interruptores están abiertos, el rotor puede girar a cualquier posición. Pero si el interruptor A se cierra, se generará un campo magnético en el polo 1 el cual atraerá al rotor y este se alineará como en la figura 4. Y después, se abre el interruptor A y se cierra el B, el rotor se alineará con el polo 2. El rotor habrá girado 60° en sentido contrario a las manecillas del reloj. Y esta operación se repite pero ahora abriendo el interruptor B y cerrando el C. El rotor girará otros 60° para alinearse con el polo 3. Si se quisiera que el motor girara en sentido contrario habría que realizar esa misma secuencia pero en el otro sentido. Básicamente así es como un motor a pasos funciona, claro que los motores de aplicaciones son más complejos, contando con más bobinas en el estator y más polos magnéticos en el rotor. Y un motor a pasos debe ser controlador con un microcontrolador que permita activar y desactivar las bobinas lo suficientemente rápido como se quiera que gire el motor.

La secuencia que hemos descrito arriba, hacía que cada polo del rotor coincidiera perfectamente con un devanado, y a esto le llamamos *paso* y cuando se activa la siguiente bobina el rotor avanza un paso más. Pero ahora supongamos el estado inicial de la figura 4, la bobina A esta energizada atrayendo un polo del rotor, si con la bobina A encendida, encendemos la bobina B el rotor girará en medio de las bobinas A y B logrando así *medio paso.* Es decir la mitad de avance de un paso normal, una secuencia de este tipo genera una mayor precisión del movimiento.

Se ha optado para este proyecto el uso de motores a pasos 28BYJ-48 ya que sus prestaciones son suficientes para el objetivo. A continuación se describen sus características principales:

- Tensión nominal de 5V DC.
- 4 Fases.
- Par motor 0.34 Kg/cm.
- Consumo de 55mA.
- 32 pasos por vuelta.
- Caja reductora de 1/64.

Debido a la caja reductora ya montada sobre el motor, para lograr una vuelta de la flecha final necesitamos 32 * 64 = 2048 pasos y si dividimos los 2048 pasos por revolución entre 360° obtenemos el avance en grados por cada paso, siendo este 5.69° por cada paso. Si utilizamos

una secuencia *medio paso* necesitamos el doble de pasos para alcanzar una vuelta completa, es decir 2 * 2048 = 4096 medios pasos, sin embargo por cuestiones mecánicas el fabricante indica que se requieren en realidad 4076 medios pasos para una vuelta completa. Entonces 4076 / 360° = 0.088° por cada medio paso, es decir por cada medio paso el motor girará 0.088° que es un precisión alta. Es por eso que utilizaremos secuencias *Halfstep* o *medios pasos* para mover nuestros motores.

Ahora solo falta determinar la manera de cómo transformar el movimiento rotatorio del motor en un movimiento lineal del motor. Existen muchas posibilidades, como husillos, poleas y bandas, entre otras. Cualquiera es buena, pero hemos optado por utilizar un sistema de piñón-cremallera ya que sobre la misma cremallera podemos diseñar las guías para los carros de los ejes. Un piñón es una rueda solida con un borde dentado que sirve para entrelazarse con otros piñones o cremalleras. Y una cremallera es una barra solida con alguno de sus bordes igualmente dentados, como se muestra en la figura siguiente:

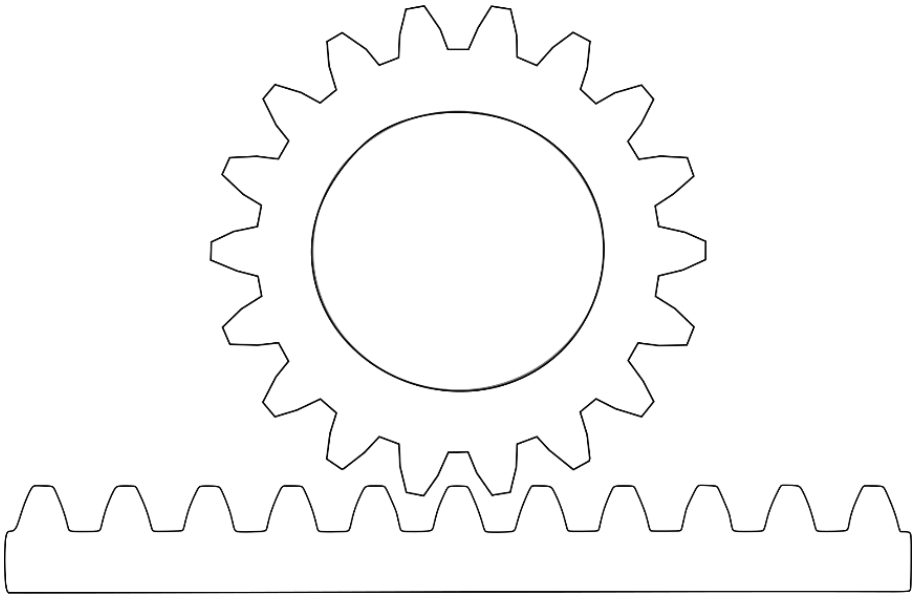

Figura 5 Esquema piñón-cremallera.

Para utilizar este sistema, hay que tener sobre todo una consideración:

- El modulo del piñón y la cremallera debe ser iguales para que embonen.

La siguiente información ha sido tomada del Blog “Metalmecánica Fácil”, para describir los parámetros de una cremallera.

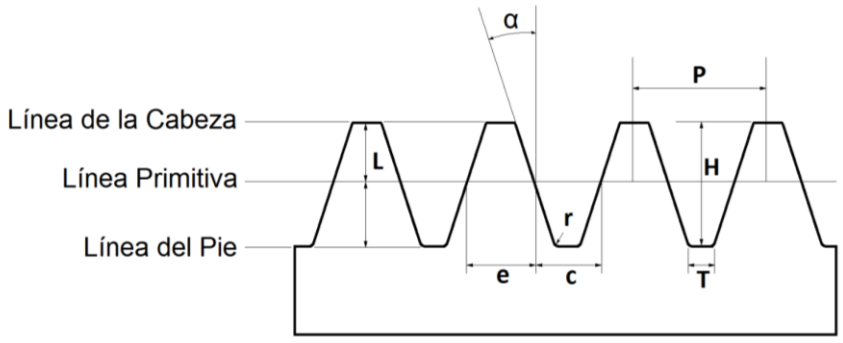

Figura 6 Parámetros de una cremallera métrica.

Donde:

P = Paso.

H = Altura total del diente.

e = Espesor del diente.

L = Altura de la cabeza del diente.

α = Semi ángulo del diente.

R = Radio del Pie del diente.

T = Ancho del fondo del diente.

Las fórmulas son las siguientes:

1. $P = \pi * M$

2. $H = 2.167 * M$

3. $e = 0.5 * P$

5. $L = M$

6. $\alpha = 20°$ (Usualmente)

7. $r = 0.3 * M$

8. $T = (P-(4 * L * \tan \alpha) / 2$

Donde M es el módulo.

Ahora para un piñón recto es necesario conocer el número de dientes (N) y el módulo (M), donde el módulo es M=N/Dp , Dp es el diámetro primitivo del piñón.

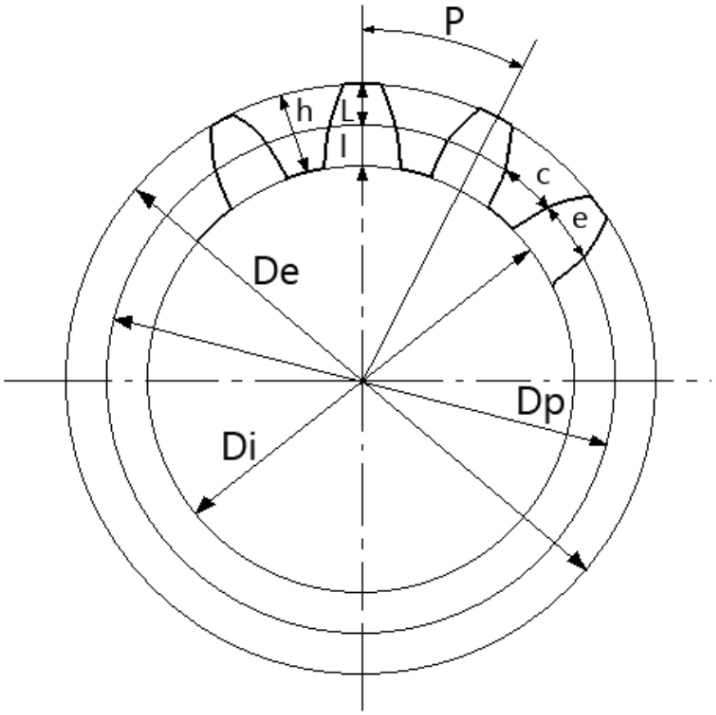

Figura 7 Parámetros de un engranaje recto.

Donde:

P = Paso diametral.

H = Altura total del diente.

L = Altura de la cabeza del diente.

I = Altura del pie del diente.

e = Espesor del diente.

c = Espacio entre dientes.

De = Diámetro exterior.

Di = Diámetro interno.

Dp = Diámetro primitivo.

Las fórmulas son las siguientes:

$Dp = M*N$

$De = Dp + (2*M)$

$H = 2.167*M$

$P = 3.1416*M$

$Di = De – (2*H)$

$c = P/2$

$e = P/2$

$L = M$

$I = 1.167*M$

La elección del módulo es un proceso iterativo de cálculo que acaba una vez que se encuentra el tamaño óptimo para la carga que ha de soportar el diente, pero para nuestro caso por ser un prototipo de bajo Par fijaremos $M = 1.02$ y $N = 12$.

Ya que tenemos una idea clara del proyecto, toca comenzar con el desarrollo de cada componente para su posterior integración.

Metodología

Selección del microscopio

Para este paso realizamos una búsqueda en la web para conocer las diferentes opciones disponibles en el mercado. Se analizó las prestaciones de los microscopios digitales disponibles y decidimos adquirir un microscopio de bajo costo, con las siguientes características:

- Conexión USB.
- Aumento máx. de 1000X.
- Compatible con Windows :2000,2003,XP,Vista,7,8,10;Linux/MAC OS/Android

El factor determinante fue que es capaz de correr en diferentes sistemas operativos así no nos limitamos a ciertos sistemas de cómputo. En la siguiente figura presentamos el microscopio adquirido.

Figura 8 Microscopio Digital adquirido.

Diseño del robot cartesiano

Comenzaremos estableciendo algunas condiciones para nuestro robot, primero debe ser de bajo costo por lo que deberemos optimizar en el material requerido, eso quiere decir que sean piezas

pequeñas pero capaces de aguantar las cargas, así como que el microscopio debe ser capaz de escanear un área de 20 cm x 20 cm.

Ahora iniciaremos a partir del microscopio, esta será nuestra referencia. El microscopio no indica sus dimensiones, por lo que deberemos obtenerlas manualmente mediante mediciones, esto se logrará con un Vernier o pie de rey. Procedemos ahora a dibujar en software el microscopio. Se decidió utilizar el software de diseño Fusion 360 pues es un software libre.

Existen muchas maneras de dibujar un sólido, pero para este caso se decidió hacerlo mediante una *revolución*. Para esto se obtuvo las medidas del microscopio y se dibujó su perfil. Como se muestra a continuación.

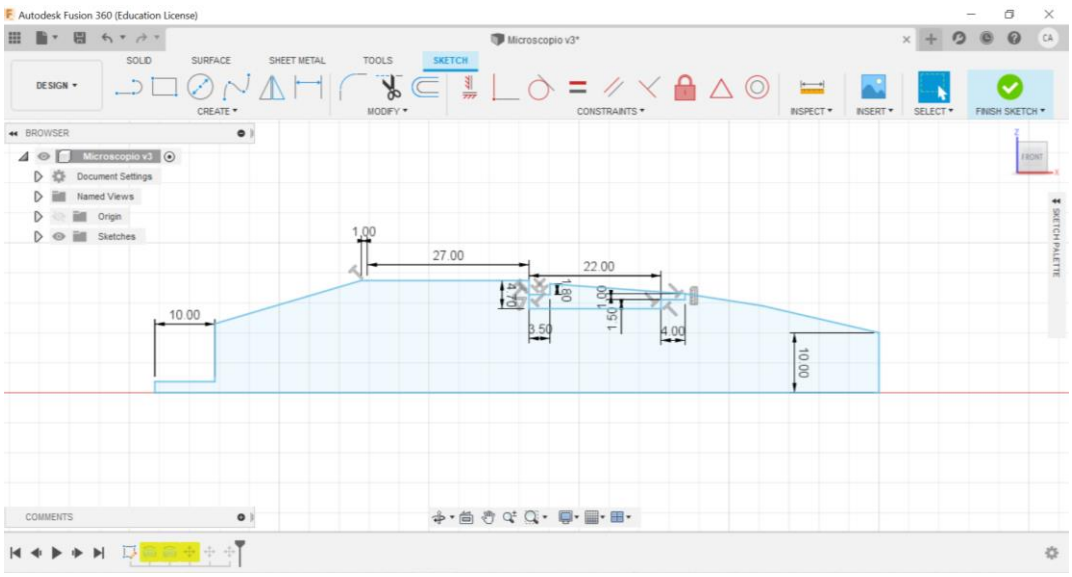

Figura 9 Perfil del microscopio

Una vez con el perfil, solo hará falta utilizar la función de *revolución* muy famosa en los softwares de diseño, obteniendo así el siguiente sólido.

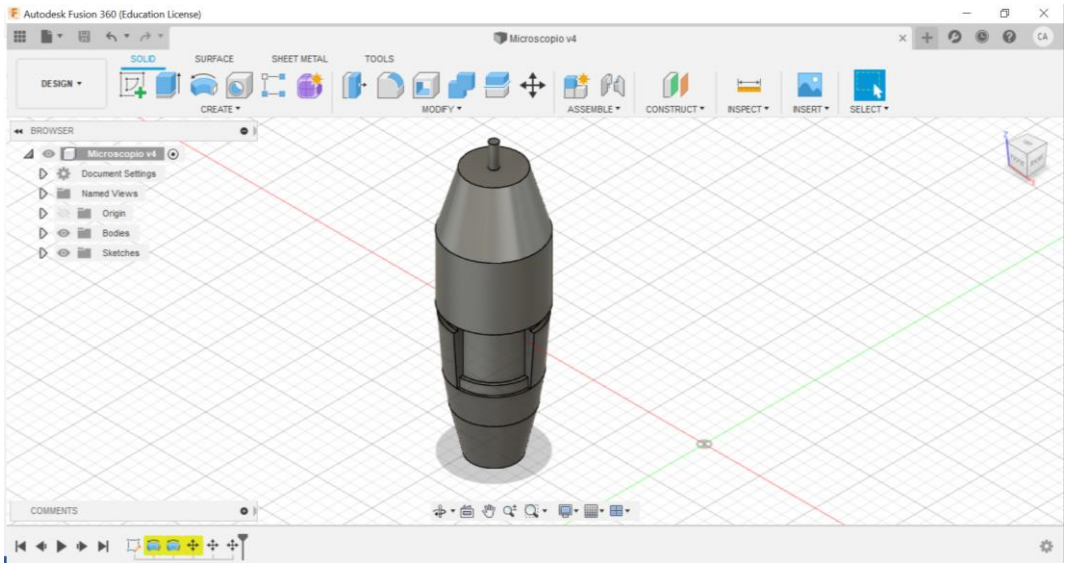

Figura 10 Representación del microscopio en software Fusion 360.

Debemos diseñar la base que sostendrá al microscopio, pero como esta base se desplazará sobre el eje Z requiere contar ya con el diseño del piñón y la cremallera. Será el siguiente paso en nuestro diseño.

El sistema piñón-cremallera que transformara el movimiento rotativo de los motores a un movimiento lineal. Se decidió que M = 1 y N = 12. Teniendo ya definido los principales parámetros es cuestión de realizar los cálculos y proceder a dibujar el piñón y cremallera, como se muestran a continuación.

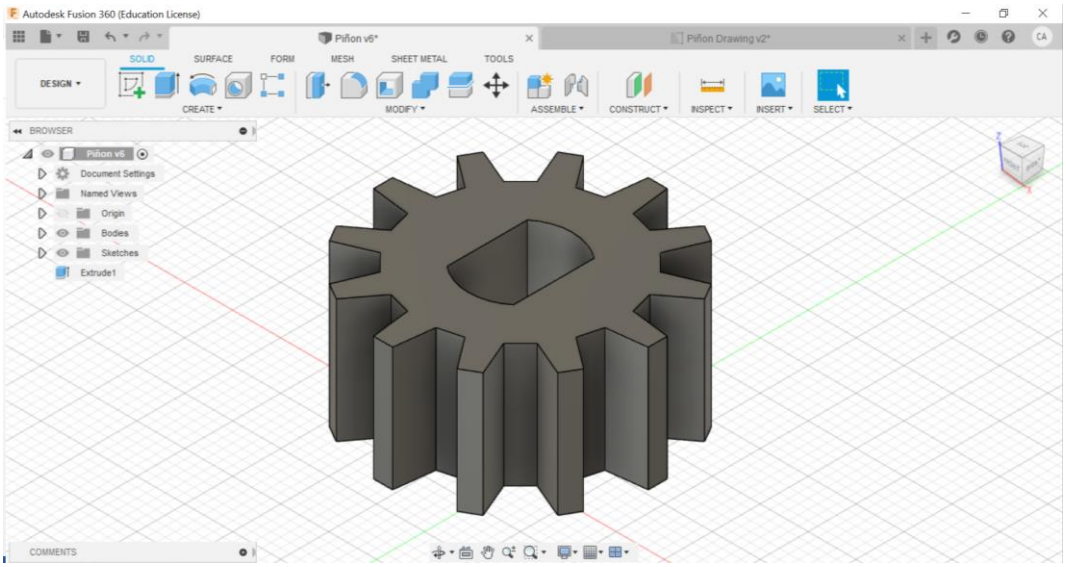

Figura 11 Piñón

Ya que las cremallera serán los eje por donde el reste de eje correrá, esta serán dibujadas como guías a su vez. Obteniendo así el siguiente sólido.

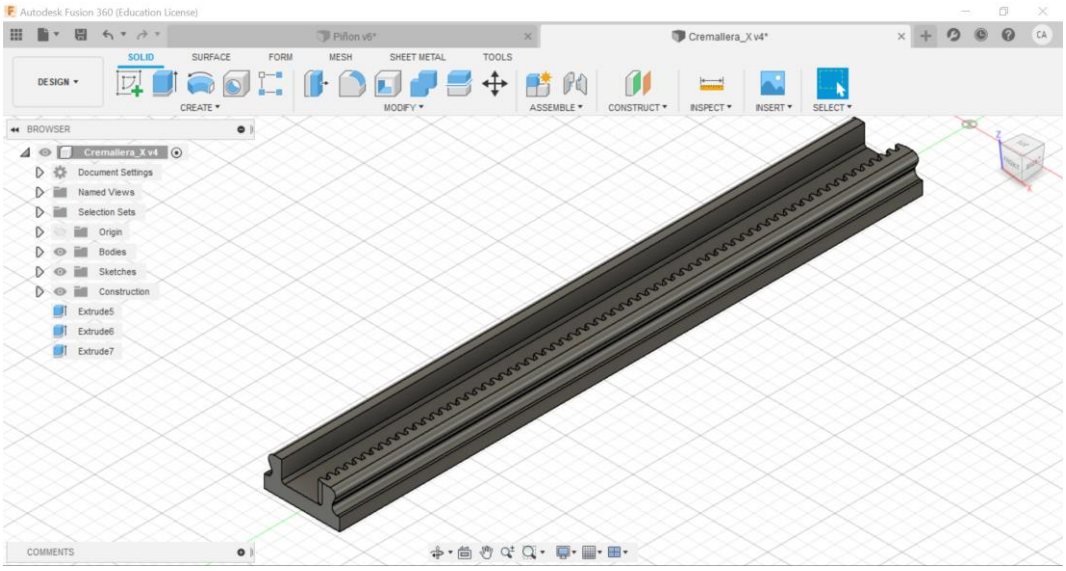

Figura 12 Guía-cremallera para ejes.

Cuando se habla de guías siempre existen dos elementos interactuando, uno fijo que es la guía y otro móvil donde se montaran los demás ejes. Entonces habrá que diseñar este elemento faltante al cual posteriormente le llamaríamos carro, ver la siguiente figura.

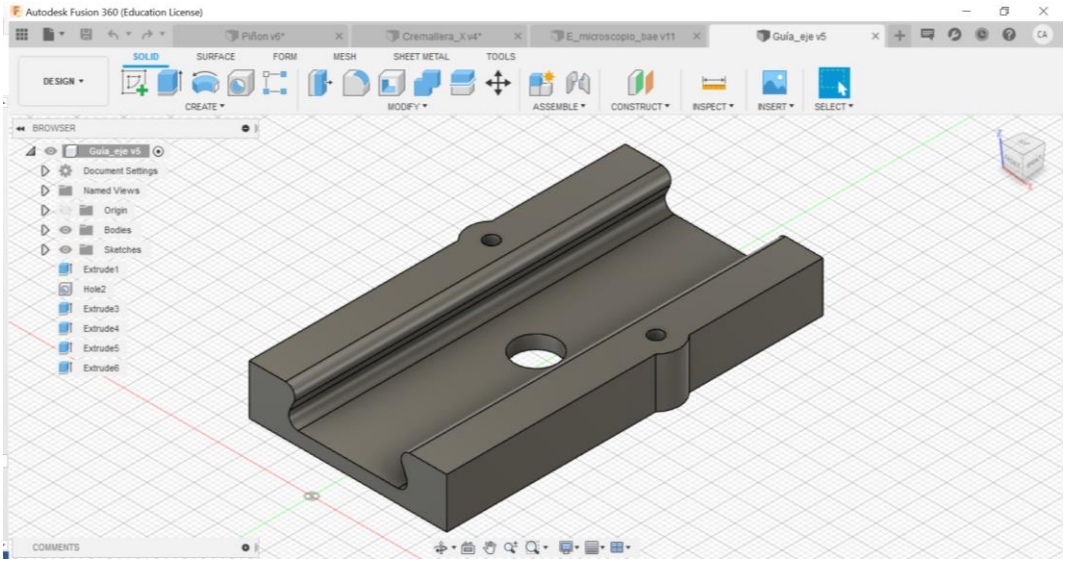

Figura 13 Carro.

Se puede observar que este carro tiene un orificio en el centro por donde el eje final del motor pasará para acoplarse con el piñón y la cremallera.

Ahora ya disponemos de las medidas base para comenzar el diseño de nuestro robot cartesiano. Comenzaremos por diseñar un soporte para montar el microscopio. Este ya viene con una ranura para su montura con su propia base por lo que la aprovecharemos para nuestra propia base. Lo más fácil será abrazar el microscopio para asegurar que este esté fijo y este deberá ser capaz de subir y bajar por lo que se le acoplara una cremallera para este efecto, resultando en la siguiente figura.

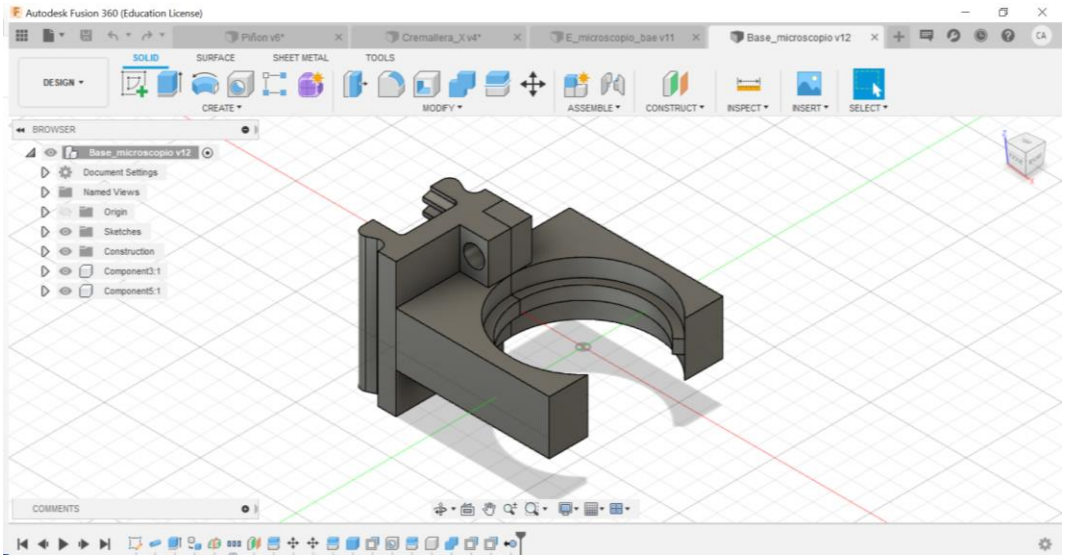

Figura 14 Base para microscopio.

Esta base consta de dos piezas, las cuales son fijadas por un tornillo para permitir montar el microscopio. También puede observarse que una de ellas tiene acoplado una guía-cremallera para poder subir o bajar el microscopio.

Sabemos que montaremos el microscopio sobre un robot cartesiano, para poder movernos en los 3 ejes del espacio. Sin embargo existen diferentes configuraciones para esta arquitectura, por lo que iniciamos proponiendo el primer concepto. Debido a que nuestro motor seleccionado es de prestaciones limitadas, hemos optado por montar el *Y* sobre dos ejes *X*, cada uno con un motor para así asegurar un movimiento constante sobre el eje *X*. Aunque es posible usar solo un eje *X* si se cuenta con un motor de prestaciones mayores.

Ya con estas piezas se procede a realizar una primera proyección para reconocer posibles defectos de diseño. Resultando en el primer concepto.

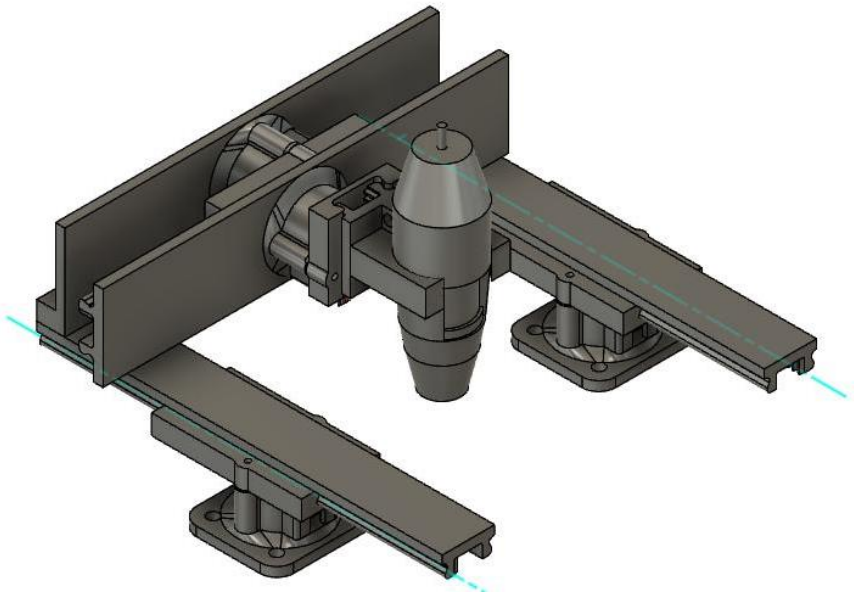

Figura 15 Concepto 1 Robot cartesiano.

Es fácil apreciar que cuenta con muchos defectos y una imagen un poco extraña. Al principio se propuso que los ejes se montaran sobre las cremalleras y los motores fueran fijos sobre unas bases, pero esto podría traer problemas de que la estructura se colgara una vez que el eje saliera del carro. Así mismo se puede observar que el microscopio ejercería bastante torsión al eje *Y* por la distancia a la que se hallaba. Debido a esto se buscó un nuevo diseño. Se considera que las piezas por separado son las adecuadas, solo que la configuración no es la correcta. Por lo que se propone un segundo concepto.

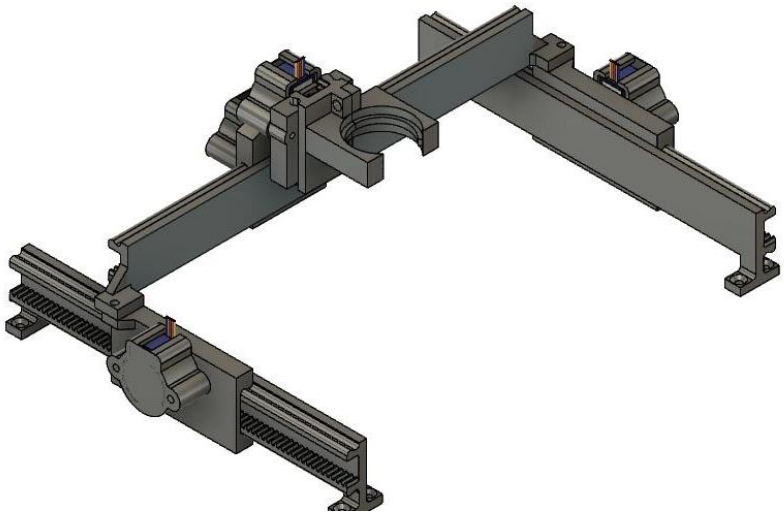

Figura 16 Concepto 2 Robot cartesiano.

En este segundo concepto se ha cambiado la configuración hacia que las cremalleras, por ser más largas sean fijas y los carros se vuelvan los dispositivos móviles. También se ha reducido la distancia del soporte del microscopio al eje *Y* para reducir la torsión. Sin embargo algunas uniones como la de los eje *X*-*Y* o el soporte del motor *Z* con el eje *Y* lucen bastante débiles, por lo que ahora se buscará mejorar esta parte con un nuevo concepto. Dando como resultado el concepto 3.

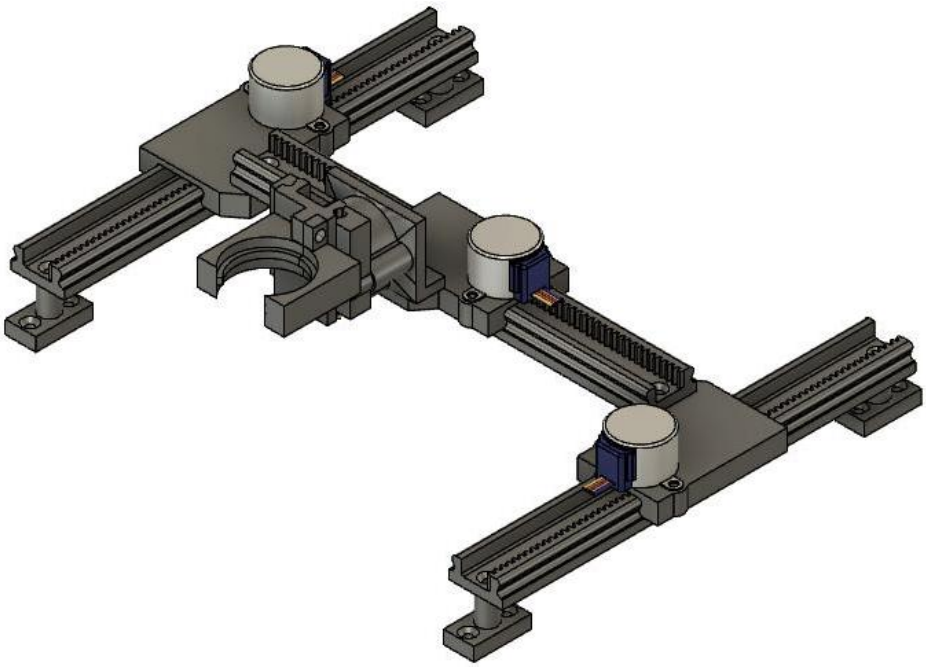

Figura 17 Concepto 3 Robot cartesiano.

En este concepto se han mejorado las uniones entre los ejes, se han estandarizado los carros por un solo diseño, pero esto a la vez ha añadido otros inconvenientes, como que los 2 ejes *X* se encuentran contrarios, lo que podría significar una complicación más a la hora de fijar las secuencias de los motores, así mismo el soporte del microscopio ha aumentado su distancia del eje *Y* produciendo una torsión sobre el carro del eje *Y*. Y las dos cremalleras del eje *X* deberán ser más largas de lo planeado para contrarrestar la contraposición de sus carros.

Pareciera ser un diseño ya aceptable, pero aún es posible corregir algunos defectos y evitar inconvenientes en fases más adelantadas del desarrollo, esto nos lleva a buscar un 4 concepto.

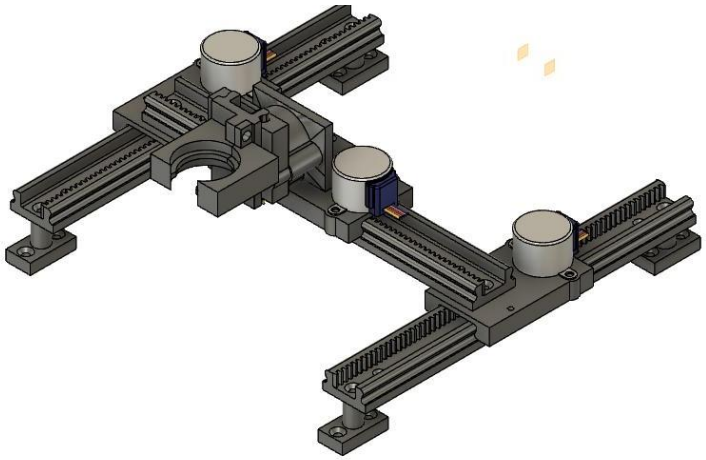

Figura 18 Concepto 4 Robot cartesiano.

Con las mejoras del Concepto 4 hemos estandarizado las cremalleras *2*X-Y* a un solo tipo, los carros son de un solo tipo lo que le otorga al dispositivo una mayor versatilidad al no disponer de diferentes piezas para su funcionamiento. Se redujo la distancia del soporte al eje *Y* reduciendo la torsión sobre este; así como los carros del eje *X* ahora se mueven en la misma dirección, facilitando el control de los motores. Ahora que ya no se cuentan con desperfectos críticos, se ha optado por seleccionar el concepto 4 del robot cartesiano como el diseño final. El siguiente paso será generar los archivos .STL para imprimir las piezas en 3D los cuales son fácilmente generados en el software Fusion 360.

Una vez las piezas impresas se les proporcionó un lijado manual para remover las rebabas de la impresión y alisar las superficies de contacto para evitar atascos. Como paso final se montan las piezas y se generan las uniones entre ellas con tornillería prevista.

Programación del control robótico y microscopio digital

Ya tenemos definido como es que el robot cartesiano se moverá, ahora procederemos a programar las rutinas de movimientos que este deberá ejecutar, así como el control del microscopio digital.

Comenzaremos con el control de posicionamiento del robot. El dispositivo que controlará el posicionamiento será la Raspberry, la cual es posible programarla en el lenguaje Python, existe ya una librería para poder utilizar los pines de manera fácil. Las librerías a utilizr son *RPi.GPIO* para el control sobre los pines y *time* para poder crear retardos. Lo primero será mandar a llamar las librerías. Así mismo añadiré una ruta para poder cargar los módulos de *OpenCv* ya que se encuentran en un ambiente virtual. Posteriormente definiremos como variables los pines a usar para cada motor y switches que nos ayudaran en el control de posicionamiento. También defino ciertas variables para hacer más fácil el control así como la secuencia de medio paso para los motores.

```
import sys #Importar modulo 'sys'.
sys.path.append('/usr/local/lib/python3.7/dist-packages')
sys.path.append('/usr/local/lib/python3.7/site-packages')
import RPi.GPIO as GPIO #Librería para controlar pines
import time #Librería para crear retaros
import cv2 #Librería para procesamiento de imágenes

control_pins_z = [8,10,12,16] #Pines para control del motor Z
control_pins_y = [29,31,33,35] #Pines para control del motor Z
control_pins_x = [(3,13),(5,15),(7,19),(11,21)] #Pines para control de los
motores X's
control_pins_x1 = [3,5,7,11] #Pines para control del motor X1
control_pins_x2 = [13,15,19,21] #Pines para control del motor X2
switches = [18,22,24,26,32,36,38,40] #Pines para switches

pasos_x_rev = 4076 #Numero de pasos para una vuelta de motor(externo)
avance = 38.4 #Avance lienal del piñon por revolucion. A = pi() * Dp = p *
z (paso * # de dientes)

dimension_x = 1.1 #Dimensiones de la fotografía tomada por el microscopio
dimension_y = 1.4
```

```
paso_micro = 0.05 #mm igual a 50 micras
paso_chico = 1 #1mm de avance
paso_mediano = 5 #5mm de avance
paso_grande = 10 #10mm de avance

micro = cv2.VideoCapture(0)

halfstep_seq= [ #Secuencia HALFSTEP
    [1,0,0,0],
    [1,1,0,0],
    [0,1,0,0],
    [0,1,1,0],
    [0,0,1,0],
    [0,0,1,1],
    [0,0,0,1],
    [1,0,0,1]
]
```

Con el objeto de ahorrar código, se definen funciones para poder utilizarlas a lo largo del desarrollo, iniciaremos definiendo dos funciones de inicialización de pines ya que, cada que la Raspberry ejecuta una función es necesario inicializarlos, de lo contrario la ejecución nos marcará un error.

```
def init_in(switches):
    '''
    Función para inicializar pines de entrada.

    Parameters
    ----------
    switches : List
        Lista de pines a usar como entradas.

    Returns
    -------
    None.

    '''
```

```
    GPIO.setmode(GPIO.BOARD)
    for pin in switches:
        GPIO.setup(pin,GPIO.IN)

def init_out(control_pins):
    '''
    Función para inicializar salidas para el control de bobinas.

    Parameters
    ----------
    control_pins : List
        Lista de pines a configurar como salidas.

    Returns
    -------
    None.

    '''
    GPIO.setmode(GPIO.BOARD)
    for pin in control_pins:
        GPIO.setup(pin,GPIO.OUT) #Declarar los pines como salidas.
        GPIO.output(pin,0) #Poner en 0V cada pin.
```

Ahora generaremos una función simple para mover un motor. Esta función será capaz de efectuar la secuencia de pulsos según un parámetro de dirección dado, así como calculará el número de medios pasos requeridos para alcanzar el recorrido deseado. Esto se logra sabiendo que el avance de un piñon por revolución es A = π * Dp = p * z = 3.2 * 12 = 38.4 mm/rev. Si queremos conocer la cantidad de pulsos que debemos mandar para avanzar una distancia *g* cualquiera solo habría que calcular pulsos = (g * medios_pasos_rev) / A ; y esto se calcula dentro de nuestra función.

```
def mov(control_pins, dist, direccion):
    '''
    Controla los pulsos de salida para las bobinas definiendo el sentido de
giro.

    Parameters
    ----------
```

```
    control_pins : List
        Lista de pines que reciben la secuencia.
    dist : Float
        Distancia a avanzar en mm.
    direccion : Bool
        Sentido de giro del motor, Horario o antihorario.

    Returns
    -------
    bool
        Bandera para saber su culminación.

    '''
    init_out(control_pins)
    pulsos = (dist * pasos_x_rev) / avance
    cont = 0
    if direccion == True:
        while cont <= pulsos:
            for seq in range(8):
                if cont <= pulsos:
                    for pin in range(4):
                        GPIO.output(control_pins[pin],halfstep_seq[seq][pin
])
                    cont += 1
                    time.sleep(0.015) #Movimientos rapido 0.001
    else:
        while cont <= pulsos:
            for seq in reversed(range(8)):
                if cont <= pulsos:
                    for pin in range(4):
                        GPIO.output(control_pins[pin],halfstep_seq[seq][pin
])
                    cont += 1
                    time.sleep(0.015)
    GPIO.cleanup()
    return True
```

Ahora, con esta función podemos controlar el movimiento de cada motor, según los parámetros entregados. Podemos entonces pasar a programar una secuencia de escaneo para nuestro microscopio.

Supongamos que el área de enfoque de nuestros microscopio es de 5mm x 5 mm con una imagen detallada, si quisiéramos nosotros inspeccionar un área de 20 mm x 20 mm deberíamos hacer un recorrido en *X* y en *Y* observando toda esta área. El funcionamiento es el siguiente, el microscopio se posicionara en la esquina superior derecha del área a inspeccionar como se muestra a continuación:

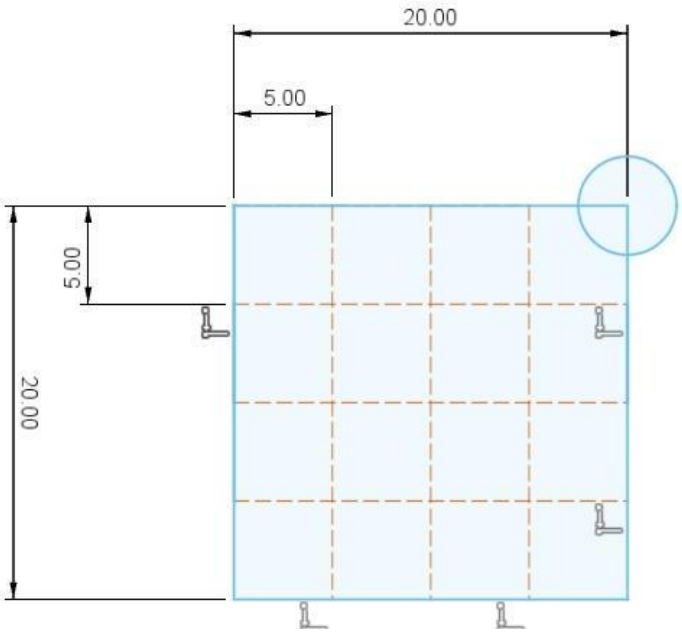

Figura 19 Posición cero del microscopio.

Posteriormente, se deberá posicionar en el centro de la primera captura de la siguiente manera y realizar un barrido sobre el área total que marca la línea azul:

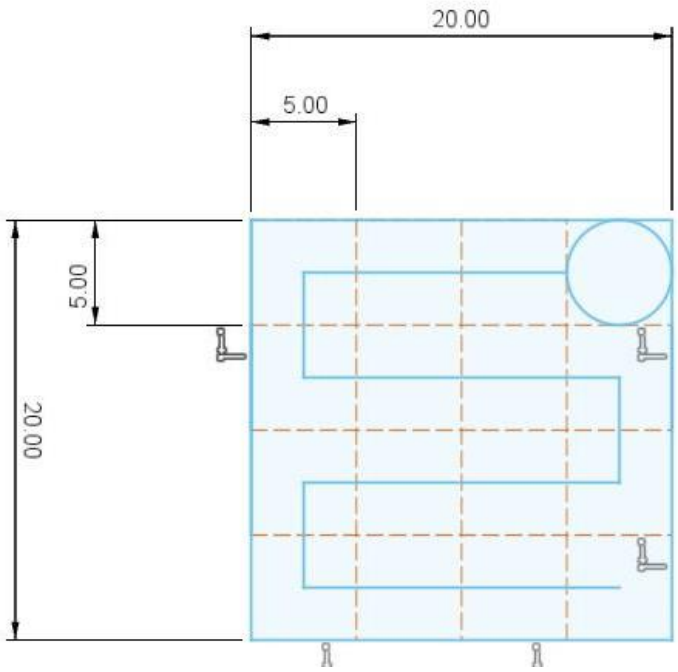

Figura 20 Primer posición del microscopio.

Se puede observar que el área se dividió en secciones más pequeñas que el microscopio si alcanza a capturar, esto se logra dividiendo la distancia a inspeccionar en *X* entre la dimensión de captura del microscopio igual en el eje *X*, así es como conocemos cuantos saltos debemos hacer sobre *X*. Y aplica de la misma manera para el eje *Y*. Pero esto tiene un inconveniente ya que si las distancia en *X* y *Y* no son múltiplos de las dimensiones de nuestra fotografía, queda un remanente a inspeccionar; por ahora omitiremos este restante espacio y nuestro microscopio solo realizara el múltiplo de las dimensiones de la fotografía.

Ahora, tenemos cuatro posibilidades para la rutina, primero es que el microscopio deba hacer recorridos en Y haciendo recorridos en X o bien sin hacer recorrido en X, y también puede ser que no deba hacer recorridos en Y pero si en X o tampoco en X. Estas posibilidades las analizaremos con condicionales *if* dentro de nuestra función. Nuestra función queda de la siguiente manera:

```
def scan(X,Y):
    init_in(switches) #Inicializos switches.
    capturas_x = int(X / dimension_x) #Numero entero de fotos a tomar en X.
    capturas_y = int(Y / dimension_y) #Numero entero de fotos a tomar en Y.
    #Movimientos para posicionar el micro en la 1er captura.
    mov(control_pins_x, (dimension_x/2), True)
    mov(control_pins_y,(dimension_y/2),True)
```

```
    print("Posicion 0 alcanzada")
    time.sleep(1.5)
    control_direccion = True
    if capturas_y > 1: #Se necesita hacer recorridos en Y.
        for foto_y in range(capturas_y): #Ciclo para recorrer el eje Y.
            id1=foto_y #1er identificador para foto.
            if capturas_x > 1: #Se necesita hacer recorridos en X.
                for foto in range(capturas_x): #Movimiento y captura para
cada foto en X.
                    id2 = foto #Segundo identificador para foto.
                    cadena = "{}.{}".format(id1,id2) #Generamos
identificador
                    #Aqui va función que analiza enfoque y ajusta.
                    time.sleep(1.5)
                    extract_image(cadena)
                    mov(control_pins_x, dimension_x, control_direccion) #Av
anzamos a la siguiente posición en X.
                    time.sleep(1) #Retraso para estabilizar ejecución.
            else: #No se necesita hacer recorridos en X
                id2 = 0 #Segundo identificador para foto.
                cadena = "{}.{}".format(id1,id2) #Generamos identificador
                #Aqui va función que analiza enfoque y ajusta.
                time.sleep(1.5)
                extract_image(cadena)
                time.sleep(1) #Retraso para estabilizar ejecución.
            mov(control_pins_y, dimension_y, True) #Sube en Y para realizar
el siguiente recorrido.
            control_direccion = not control_direccion #Cambia el sentido de
giro del motor.
    else: #No se necesita hacer recorridos en Y
        id1=0 #1er identificador para foto.
        if capturas_x > 1: #Se necesita hacer recorridos en X
            for foto in range(capturas_x): #Movimiento y captura para cada
foto en X.
                id2 = foto #Segundo identificador para foto.
                cadena = "{}.{}".format(id1,id2)
                #Aqui va función que analiza enfoque y ajusta.
                time.sleep(1.5)
                extract_image(cadena)
                mov(control_pins_x, dimension_x, control_direccion) #Avanza
mos a la siguiente posición en X.
                time.sleep(1) #Retraso para estabilizar ejecución.
        else: #No se necesita hacer recorridos en X
            id2=0 #Segundo identificador para foto.
            cadena = "{}.{}".format(id1,id2) #Generamos identificador
            #Aqui va función que analiza enfoque y ajusta.
            time.sleep(1.5)
            extract_image(cadena)
    print("Ciclo acabado")
    return True
```

Se dejó espacios con comentarios donde se colocará una función de autoenfoque pero que por ahora no pertenece a este proyecto.

La función *extract_image()* se encarga de abrir la cámara del microscopio, y guardar la foto con el identificador asignado como parámetro, como se muestra a continuación:

```
def extract_image(cadena):
    '''
    Almacena la imagen mostrada en el servidor asignándole un
identificador.

    Parameters
    ----------
    cadena : Str
       Identificador para cada imagen.

    Returns
    -------
    bool
        Indicador de ejecución.

    '''
    img = cv2.imread("fotos/last_image.jpg",cv2.IMREAD_COLOR)
    cv2.imwrite("fotos/images/{}.jpg".format(cadena),img)
    print("Foto Hecha")
    return True
```

Una vez que hemos obtenido cada imagen es posible realizar la reconstrucción de la superficie, esto lo ejecutaremos con una nueva función llamada *maping()*. Básicamente esta función recogerá la lista de imágenes guardadas durante el escaneo, y a través de la librería de *Stitcher* de *OpenCv* se buscarán puntos clave en cada imagen, para posteriormente compararlos con las demás imágenes y donde se halle coincidencia sobrepondrá cada imagen.

```
def maping():
    print("[INFO] loading images...") #Indicador de inicio de carga.
    imagePaths = sorted(list(paths.list_images("fotos/images"))) #Obtenemos
la lista de imagenes en el directorio marcado.
    images = [] #Creamos una lista para almacenar las imagenes.
    for imagePath in imagePaths:
        image = cv2.imread(imagePath) #Cargamos cada imagen de la lista.
        images.append(image) #Agregamos cada imagen a la lista "images"
```

```
    print("[INFO] stitching images...") #Indicador de inicio de
procesamiento.
    stitcher = cv2.Stitcher_create() #Creamos el objeto Stitcher.
    (status, stitched) = stitcher.stitch(images) #Se eejcuta todo el
procesamiento con esta función.

    if status == 0: #Si el procesamiento funciono guardamos el resultado.
        cv2.imwrite("fotos/output.png",stitched)

    else: #Si el procesamiento no funciono desplegamos el tipo de error.
        print("[INFO] image stitching failed ({})".format(status))
```

Y por último se diseñó una función de *home()* la cual envía al microscopia a una posición inicial siempre al inicio de la ejecución del programa, colocando interruptores de inicio de carrera en cada eje es fácil lograrlo, solo debemos avanzar los motores de poco en poco hasta recibir la señal del interruptor.

```
def home():
    '''
   Función para que el micro haga HOME, debera alcansar interruptores de
inicio de carrera para cada eje.

   Returns
   -------
   bool
       Indicador de Terminado.

   '''
    #Inicia HOME de Z
    print("Inicia Home Z")
    init_in(switches) #Inicializamos Switches.
    if GPIO.input(switches[0]): #Esta activo el inicio de carrera.
        time.sleep(0.2)
        if GPIO.input(switches[0]): #Validamos si está activo.
            print("1er condicion")
            mov(control_pins_z, 3, True) #Movimiento hacia adelante para
desactivar inicio de carrera.
```

```
            init_in(switches)
            while GPIO.input(switches[0]) == False: #Movimiento hacia
atrás. hasta activar el inicio de carrera.
                mov(control_pins_z, 0.5, False)
                init_in(switches)
    else: #No está activo el inicio de carrera
        print("2da condicion")
        while GPIO.input(switches[0]) == False: #Avanzar hacia atrás.
mientras no este activo el inicio de carrera.
            mov(control_pins_z, 0.5, False)
            init_in(switches)
    #Inicia HOME de Y
    print("Inicia Home Y")
    init_in(switches)
    if GPIO.input(switches[1]):
        time.sleep(0.2)
        if GPIO.input(switches[1]):
            print("1er condicion")
            mov(control_pins_y, 3, True)
            init_in(switches)
            while GPIO.input(switches[1]) == False:
                mov(control_pins_y, 0.5, False)
                init_in(switches)
    else:
        print("2da condicion")
        while GPIO.input(switches[1]) == False:
            mov(control_pins_y, 0.5, False)
            init_in(switches)
    #Inicia HOME de X
    print("Inicia Home X")
    init_in(switches)
    if GPIO.input(switches[2]):
        time.sleep(0.2)
        if GPIO.input(switches[2]):
            mov(control_pins_x,2,True)
            init_in(switches)
            while GPIO.input(switches[2]) == False:
                mov(control_pins_x, 0.5, False)
```

```
                init_in(switches)
    else:
        print("2da condicion")
        while GPIO.input(switches[2]) == False:
            mov(control_pins_x, 0.5, False)
            init_in(switches)

    return True
```

Servidor Web

Uno de los objetivos es poder controlar el robot remotamente y a la vez poder visualizar lo que ocurre en tiempo real, esto se le denomina *Streaming*. Para lograr esto requerimos mandar la información y recibirla a través de un servidor web, que por las cualidades de las *Raspberry* puede ser montado en ella misma. También existen ya librerías desarrolladas para este propósito lo que nos reduce el trabajo a solo implementar para nuestros requisitos; la librería que utilizaremos es *Flask.*

Iniciaremos importando los módulos requeridos, así como agregando una nueva ruta de búsqueda para el módulo de *OpenCv* que se halla almacenada en un ambiente virtual.

```
import sys
sys.path.append('/usr/local/lib/python3.7/dist-packages')
from flask import Flask, render_template, Response, request
from flask_cors import CORS
from camera_opencv import Camera
import control_micro #Modulo de control movimiento.
```

Después definiremos algunas variables que nos ayudaran al control del programa de forma más general, para que no debamos escribir su valor numérico cada que las requiramos y sean controlables desde un solo lugar en caso que deseemos modificar su funcionamiento. Estas son los pines de salida para las bobinas de los motores, el largo y ancho de la foto que definen el avance en la rutina de escaneo y por último una distancia de avance largo y corto. Estas dos distancias se utilizarán para el control manual desde la interfaz de usuario, por ahora solo definiremos 2 posibles movimientos (largo y corto) por cuestión de simplicidad pero podrían agregarse más.

```
control_pins_z = [8,10,12,16]
control_pins_y = [29,31,33,35]
control_pins_x = [(3,13),(5,15),(7,19),(11,21)]

largo_foto = 1.1
ancho_foto = 1.4

m_largo = 5
m_corto = 0.05
```

Ahora creamos el objeto *Flask* que generará nuestro servidor web e inicializamos *CORS* para permitir la comunicación sin restricciones desde la interfaz de usuario vía el servidor.

```
app = Flask(__name__)
CORS(app)
```

Ahora generamos la ruta principal del servidor con la sentencia *@app.route('/')*, la ruta está definida dentro de los paracentesis y como sobre la principal se desplegara el video, solo se añade un ' / '.

```
@app.route('/')
def index():
    return render_template('index.html')

def gen(camera):
    """Video streaming generator function."""
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

Una vez que el usuario ingresa a esta ruta, las funciones definidas dentro se ejecutarán. Primeramente la función *index()* devuelve una página *HTML* con un encabezado y una etiqueta para la imagen. El archivo *HTML* es el siguiente, pero nótese que la etiqueta de la imagen apunta a la segunda ruta de nuestra aplicación *(@app.route('/video_feed'))*.

```
<html>
  <head>
    <title>Video Streaming Demonstration</title>
```

```
  </head>
  <body>
    <h1>Video Streaming Demonstration</h1>
    <img src="{{ url_for('video_feed') }}">
  </body>
</html>
```

Esta función (*def video_feed()*) regresa las imágenes que serán desplegadas en el servidor. Vemos que para logarlo utiliza una función generadora definida como *gen()*. Esta función entra en un ciclo que continuamente regresa las imágenes de la cámara.

```
@app.route('/video_feed')
def video_feed():
    return Response(gen(Camera()),
                    mimetype='multipart/x-mixed-replace; boundary=frame')
```

Ya solucionado el envío del video al servidor web, podemos pasar a definir otras rutas para la activación de los mandos manuales. De momento se definieron 4 tipos de movimientos para cada eje más un mando para el escaneo. Los ejes pueden moverse de la siguiente manera: Movimiento largo positivo, movimiento largo negativo, movimiento corto positivo o movimiento cortó negativo. Ahora pues supongamos que para el eje X denotaremos cada uno de estos movimientos de la forma X+,X-,x+ y x- respectivamente; y así mismo para los eje Y y Z. por lo que las rutas se definieron de la siguiente manera:

```
#Rutas para accionar movimientos independientes de los ejes.
@app.route('/X+')
def movimiento_X():
    control_micro.mov(control_pins_x,m_largo,True)
    return "Ejecutado"
@app.route('/X-')
def movimiento_Xneg():
    control_micro.mov(control_pins_x,m_largo,False)
    return "Ejecutado"
@app.route('/x+')
def movimiento_x():
    control_micro.mov(control_pins_x,m_corto,True)
    return "Ejecutado"
@app.route('/x-')
```

```
def movimiento_xneg():
    control_micro.mov(control_pins_x,m_corto,False)
    return "Ejecutado"
#MOVIMIENTOS EN Y
@app.route('/Y+')
def movimiento_Y():
    control_micro.mov(control_pins_y,m_largo,True)
    return "Ejecutado"
@app.route('/Y-')
def movimiento_Yneg():
    control_micro.mov(control_pins_y,m_largo,False)
    return "Ejecutado"
@app.route('/y+')
def movimiento_y():
    control_micro.mov(control_pins_y,m_corto,True)
    return "Ejecutado"
@app.route('/y-')
def movimiento_yneg():
    control_micro.mov(control_pins_y,m_corto,False)
    return "Ejecutado"
#MOVIMIENTOS EN Z
@app.route('/Z+')
def movimiento_Z():
    control_micro.mov(control_pins_z,m_largo,True)
    return "Ejecutado"
@app.route('/Z-')
def movimiento_Zneg():
    control_micro.mov(control_pins_z,m_largo,False)
    return "Ejecutado"
@app.route('/z+')
def movimiento_z():
    control_micro.mov(control_pins_z,m_corto,True)
    return "Ejecutado"
@app.route('/z-')
def movimiento_zneg():
    control_micro.mov(control_pins_z,m_corto,False)
    return "Ejecutado"
```

Podemos observar que dentro de cada ruta, se ejecuta la función antes desarrollada para el control del robot *mov()* donde se le otorga de parámetros los pines a controlar, la distancia a avanzar y la dirección, es por eso que estas variables se definieron al inicio del script para poder ser manipuladas desde este módulo.

Definiremos una nueva para la aplicación del escaneo, esta será */scan* pero el método debe ser de recepción de datos, ya que el usuario podrá definir el área que desea inspeccionar, por esa razón definiremos la ruta con un parámetro extra como se muestra a continuación:

```
@app.route('/scan', methods=['POST'])
def datos():
    '''
    Recibe un JSON con las distancias en X y Y a inspeccionar.

    Returns
    -------
    str
        Mensaje de finalización.

    '''
    content = request.get_json()
    print(content)
    X = int(content['X'])
    Y = int(content['Y'])
    control_micro.scan(X, Y,largo_foto,ancho_foto)
    return "Hecho"
```

El método *POST* indica a la página que debe ser capaz de recibir datos, y esto lo haremos a través de la función *get_json()* del módulo *request* de *Flask*. Un dato tipo JSON es una estructura con un nombre de parámetro y un valor definido, en nuestro caso estará conformado por una variable *X* y una variable *Y* y se les entregara un valor el cuál convertimos a un entero con la función *int()*, ya que tenemos el dato lo pasamos a la función *scan()* de nuestro control de motores. Al final retornamos una bandera para indicar su finalización.

Y por último definiremos un sentencia if para que la aplicación solo corra si es ejecutada desde la línea de comandos.

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port =80, debug=True, threaded=True)
```

Si el script se ejecuta de esta manera, la aplicación correrá con el método *run()* de forma pública por el tipo de *host*, escuchando en el puerto 80 y reportara errores.

Con esto tenemos todo listo para correr la aplicación.

Resultados

Una vez teniendo el diseño de cada pieza, se procede a generar los archivos .STL para poder imprimir las piezas en 3D. Debido al proceso de impresión las piezas requieren un proceso de desbaste de exceso de material, el cual se da a mano.

Figura 21 Piezas después de impresión 3D.

Una vez que nos aseguramos de remover el exceso de material de cada pieza pasamos a la montura y fijación sobre una base. Quedando finalmente el dispositivo montado de la siguiente manera.

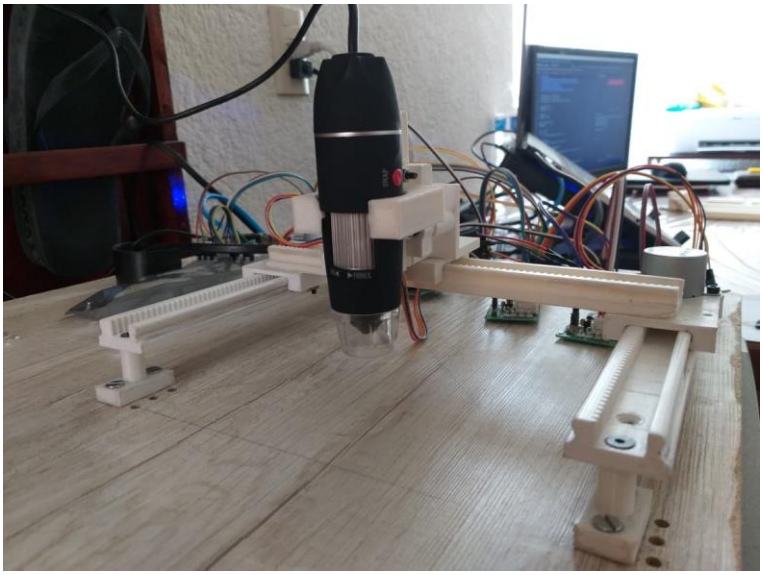

Figura 22 Microscopio montado sobre robot cartesiano. Vista frontal.

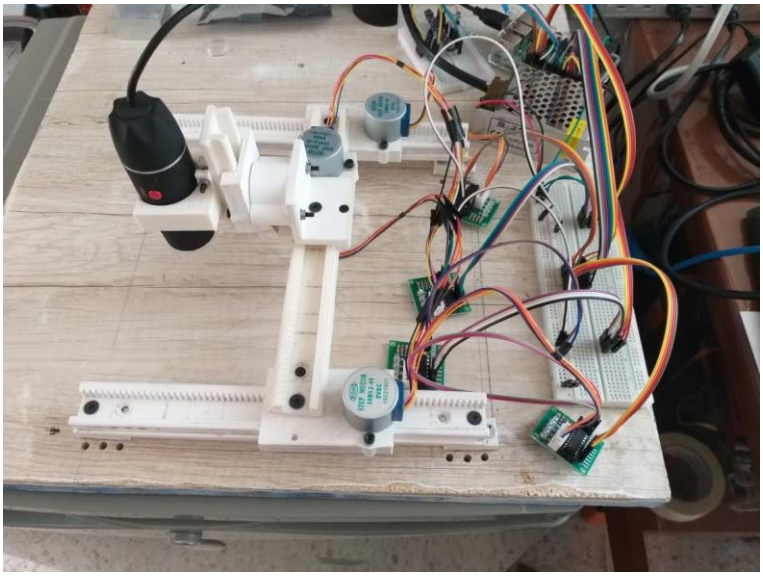

Figura 23 Microscopio montado sobre robot cartesiano. Vista lateral superior.

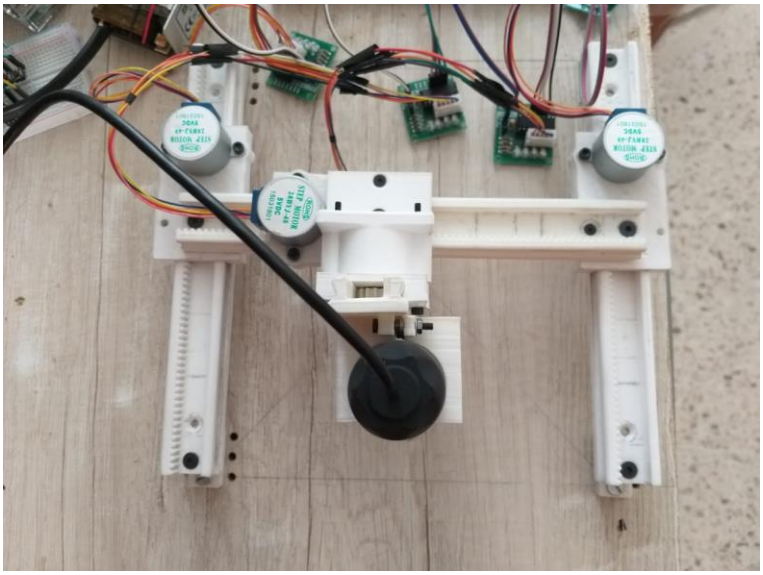

Figura 24 Microscopio montado sobre robot cartesiano. Vista superior.

Y se añade la Raspberry con una fuente de poder para poder mover los motores.

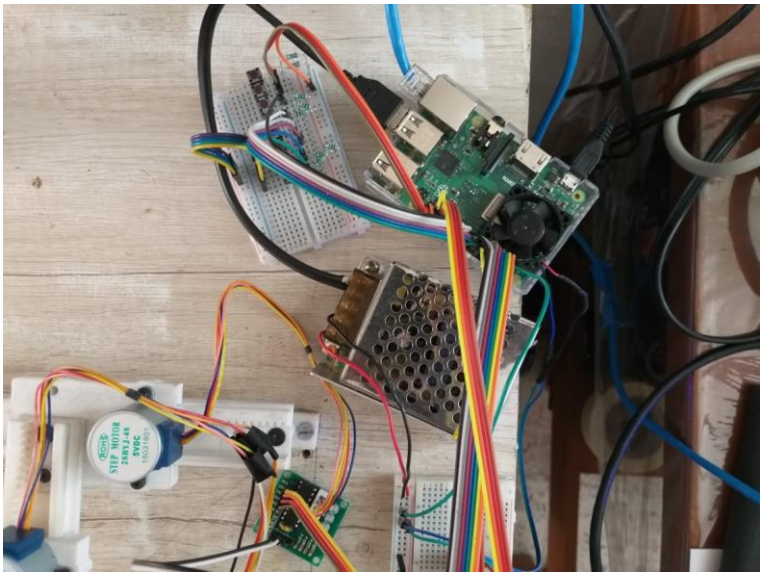

Figura 25 Raspberry y fuente de 5v.

Ahora es posible probar las funciones de control, en las siguientes dos imágenes observamos inicialmente un motor en reposo, y posteriormente un motor en movimiento, lo cual puede apreciarse gracias a los leds rojos que se encienden del driver. Estas acciones son ejecutada por la función *mov()* del módulo *control_micro*.

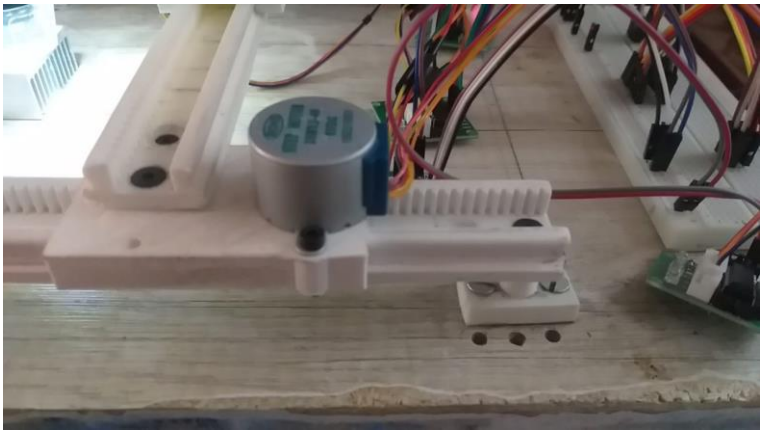

Figura 26 Motor en reposo, leds del driver apagados.

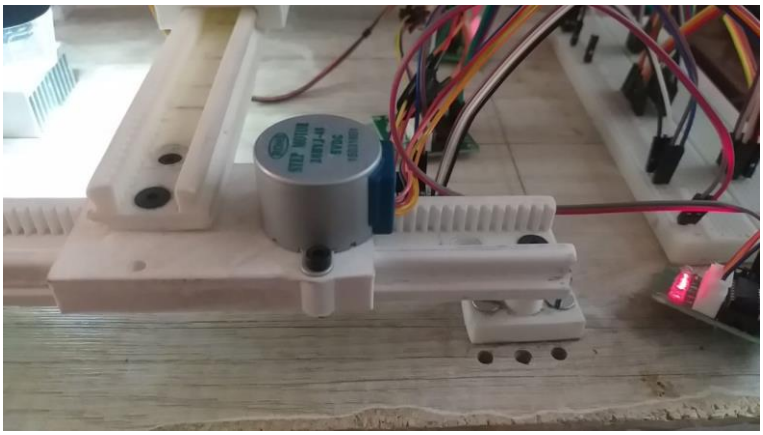

Figura 27 Motor activado, leds del driver encendidos.

Ya que sabemos que podemos ejecutar movimientos para cada eje gracias al control, podemos probar la función de escaneo del área, que es fundamental en nuestra aplicación, esta función debe hacer un barrido del área e ir tomando fotos de ella. En la siguiente imagen se presentan las capturas realizadas para un área de 3 mm x 3 mm.

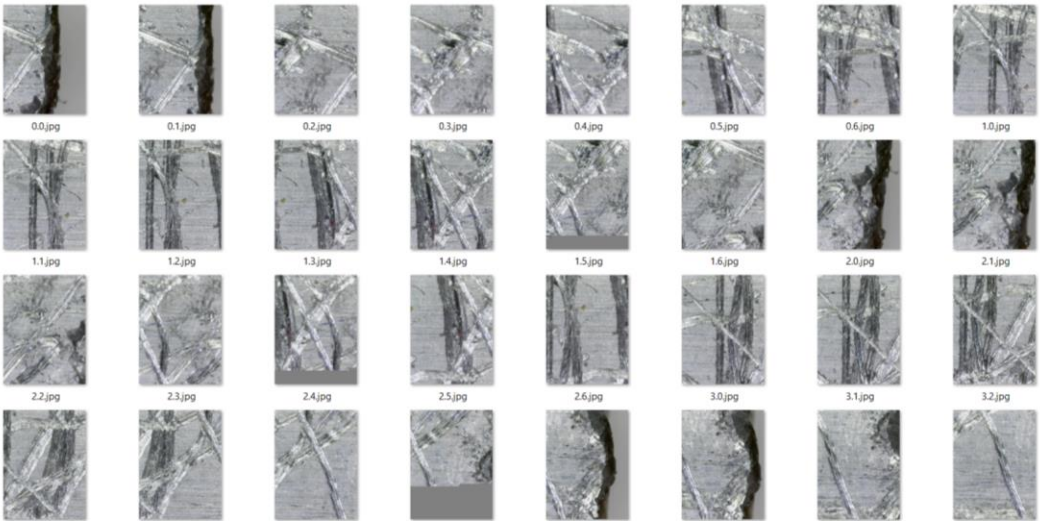

Figura 28 Imágenes recolectadas durante un escaneo.

Posteriormente proporcionamos estas imágenes a nuestra función *maping()* para reconstruir la superficie examinada, obteniendo como resultado la siguiente figura.

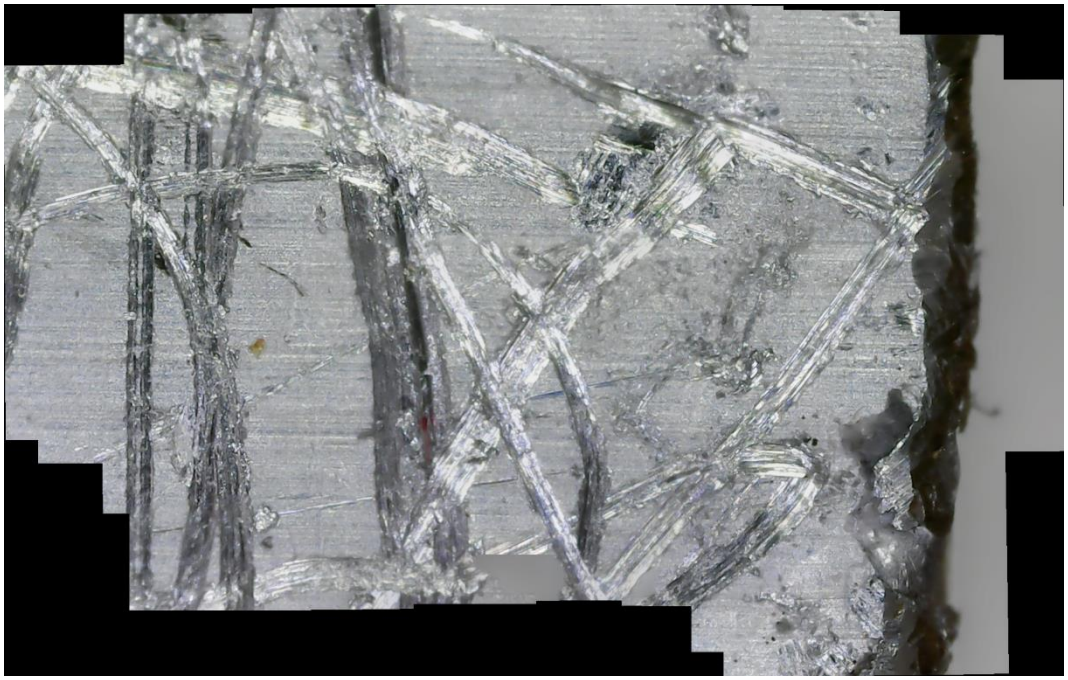

Figura 29 Reconstrucción de superficie escaneada.

Es importante destacar, que la Raspberry no fue capaz de realizar el procesamiento de imágenes, ya que esta se quedó en un estado de 'Bucle' de donde por lo menos en 40 min no salió. Sin embargo el mismo algoritmo se ejecutó en una PC con mayores capacidades dando el resultado antes presentado.

Uno de los objetivos de este proyecto, es poder controlar y monitorear el trabajo del microscopio remotamente. Esto se logra activando la aplicación appCam.py. Esta aplicación habilita nuestro servidos *Flask* y podemos saber que está en funcionamiento por los mensajes desplegados como se muestra en la siguiente figura.

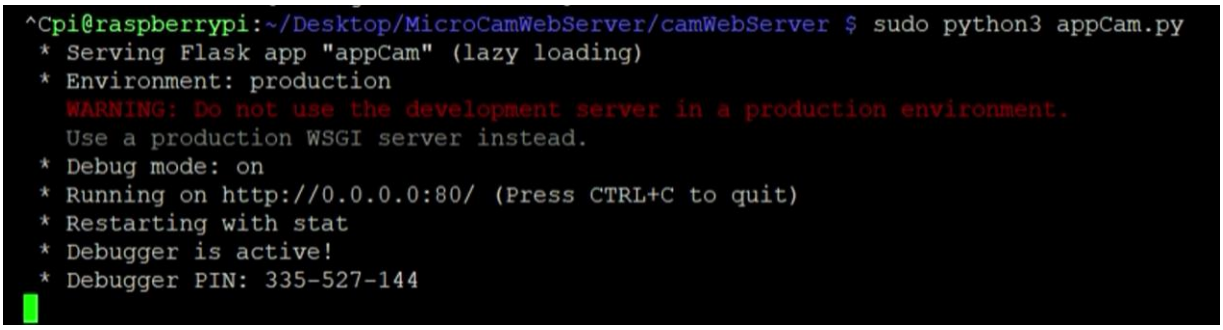

Figura 30 Arranque de servidor para transmisión y control.

Una vez que el usuario ingresa al servidor se generan las llamadas a los métodos de transmisión de imagen.

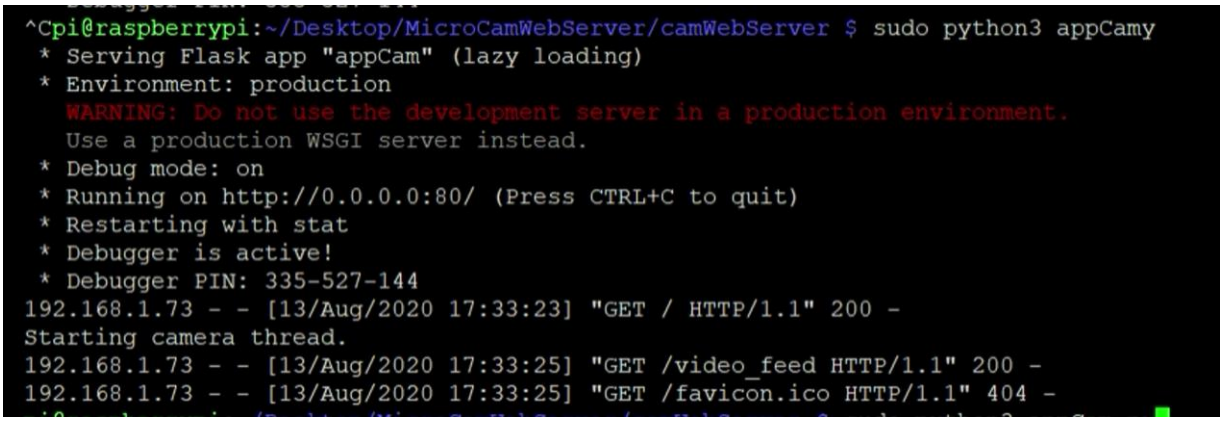

Figura 31 Solicitud del usuario al servidor.

A partir de este momento el servidor está transmitiendo de forma constante el video y a la escucha de los comandos. Los comandos recibidos a través del servidor web se visualizan de la siguiente manera.

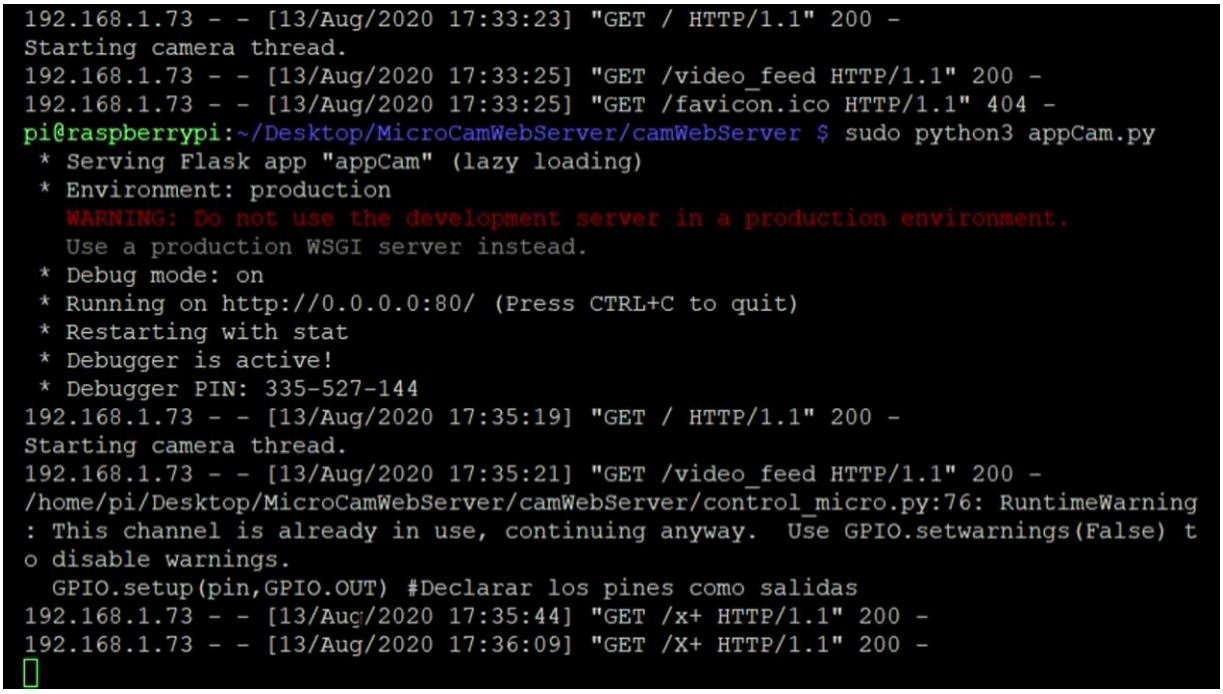

Figura 32 Solicitudes tipo GET para movimientos lineales simples.

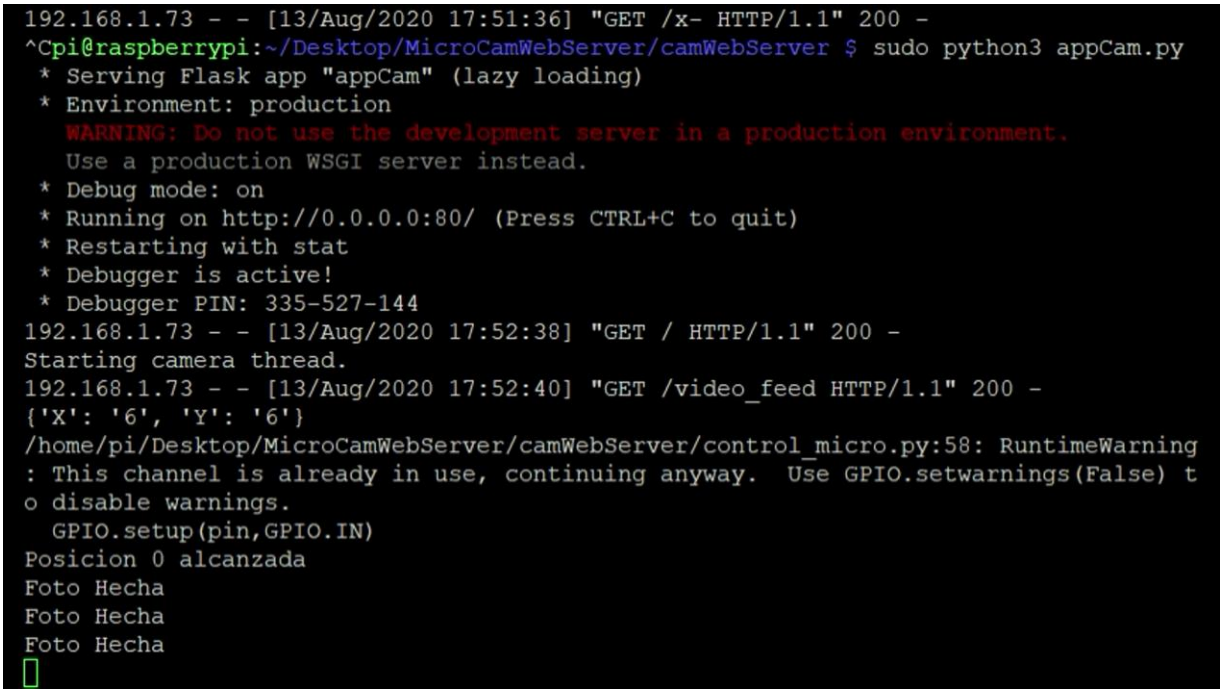

Figura 33 Solicitud POST para escaneo de área.

Nótese que en la figura 33 el servidor recibe una solicitud tipo *POST* la cual implica recibir datos del tipo JSON, este dato contiene las dimensiones en milímetros a inspeccionar. Después de recibido comienza la rutina de posicionamiento y a tomar las fotografías.

El servidor por internet se mira de la siguiente manera.

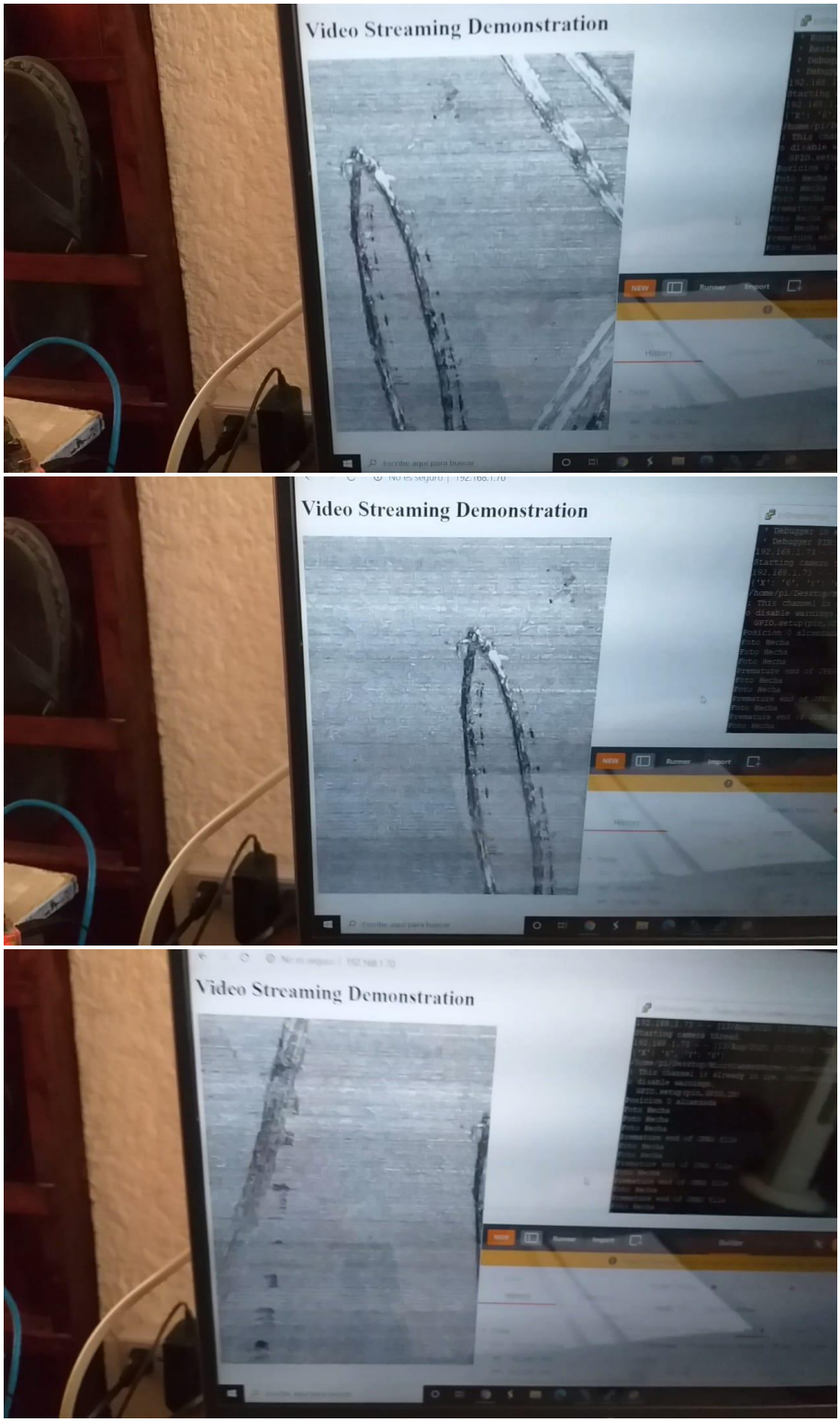

Figura 34 Secuencia de streaming vía servidor web.

La conexión vía remota con otro ordenador se logró estableciendo un túnel a través de la red con el framework NGROK.

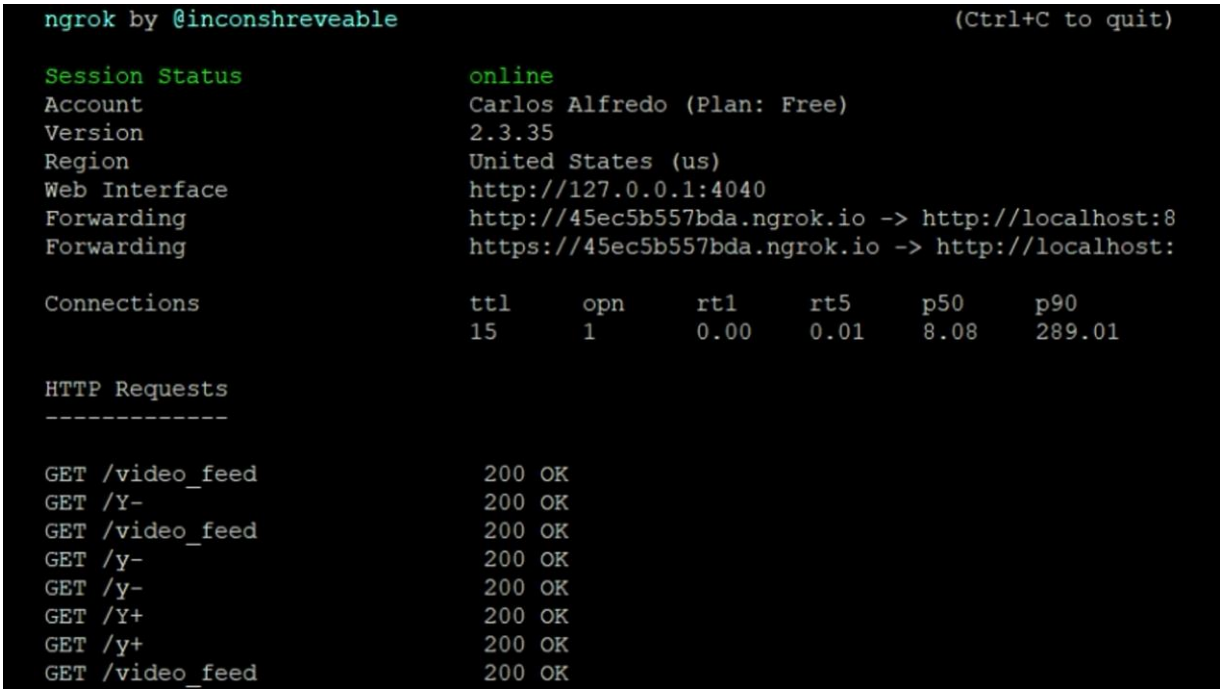

Figura 35 Túnel ngrok y solicitudes a través de él.

Conclusiones

Se ha logrado a lo largo del proyecto cubrir los requerimientos planteados a su inicio, sin embargo nos hemos encontrado con algunas dificultades en los aspectos de desarrollo e implementaciones. Los cuales se describen a continuación.

El diseño final del robot cartesiano es muy probablemente la mejor opción de entre todas por lo discutido antes, aunque es posible que se pueda mejorar el diseño para evitar que el microscopio se cuelgue tanto por su peso. Esto genera que no esté exactamente paralelo a las superficies planas sobre la cual se monta el dispositivo, aunque durante las pruebas pareciera que esto no se ve reflejado. Las piezas fueron impresas con una calidad media, generando cierto error en las dimensiones de estas, y posteriormente fueron tratadas manualmente lo que ocasiona desperfectos en la mecánica planeada. Estos desperfectos se manifestaron en ciertos puntos de los movimientos, ya que a veces no eran continuos o parejos los avances; lo que conllevaba a visualizar avances diagonales y por ende recolectar fotografías no planeadas.

Inicialmente se planeó un avance controlado del ancho y largo de la foto que era capaz de tomar el microscopio, pero al implementar el algoritmo de reconstrucción nos percatamos que no se lograba obtener los suficientes puntos clave para hacer coincidir las imágenes, lo que llevo a corregir las distancias de avance durante el escaneo, buscando incrementar los puntos de coincidencia. Esto desemboco en un incremento del número de fotografías a tomar con lo cual nuestro modelo de Raspberry demostró ser incapaz de soportar, ya que se quedaba atorada. Por lo que se requiere un modelo de mayores prestaciones para realizar la tarea.

Los servidores Flask transmiten adecuadamente los datos, pero tienen la limitante que solo un cliente puede visualizar la transmisión, por lo que si se requieren más usuarios será necesario buscar otras opciones de transmisión.

La interfaz gráfica y el servidor fueron capaces de comunicarse sin problema alguno, salvo que en ciertas ocasiones la imagen parecía congelarse por instantes posiblemente por la velocidad de transmisión de datos.

En general los objetivos particulares y general han sido alcanzados durante la elaboración de este proyecto, y ya que se trata de un prototipo nos hemos dado cuenta que su realización es viable pero es necesario afinar detalles para poder ofrecer un producto de calidad.

Bibliografía

Referencias

P. Pérez & M. Valente (2018). Fundamentos básicos del procesamiento de imágenes. Curso de Introducción al procesamiento de imágenes radiológicas en ámbito clínico.

A. Barrientos, L.F. Peñin, C. Balaguer, R. Aracil(2da Edición). Capítulo 2. Morfología del Robot. Fundamentos de Robótica, 31-62.

Carlos. (2011-2019). Motor Eléctrico. EcuRed [versión Electrónica]. Cuba, https://www.ecured.cu/Motor_el%C3%A9ctrico

Libros

Bolton, William,(2013),*Mecatrónica, sistemas de control electrónico en la ingeniería mecánica y eléctrica.* Essex, Inglaterra. Pearson education limited.

Pérez Castellanos, Jóse(Ed.), Teoría de máquinas y mecanismos. E.U.A. McGraw Hill.

McKinney, Wes (2012). Python for data analysis. E.U.A. O'Reilly Media Inc.

Wilde, Theodore (2006). Máquinas eléctricas y sistemas de potencia. Prentice Hall.

Sánchez Gonzales, Carmelo (Ed.). Fundamentos de Robótica. España, McGraw Hill.