IMPLEMENTACIÓN DE UN SISTEMA DE PROGRAMACIÓN DE MÁQUINAS RECTILÍNEAS PARA LA FABRICACIÓN DE CUELLOS TEJIDOS.

PRESENTADO POR:

DANIEL ESCAMILLA GUTIÉRREZ

SEBASTIAN MOSQUERA MUÑOZ

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

INGENIERÍA ELECTRÓNICA

Pereira- Risaralda

2016

IMPLEMENTACIÓN DE UN SISTEMA DE PROGRAMACIÓN DE MÁQUINAS RECTILÍNEAS PARA LA FABRICACIÓN DE CUELLOS TEJIDOS.

PRESENTADO POR:
DANIEL ESCAMILLA GUTIÉRREZ
SEBASTIAN MOSQUERA MUÑOZ

Trabajo presentado como requisito para optar al título de
Ingeniero Electrónico

DIRECTOR:

M. Sc. MAURICIO HOLGUIN LONDOÑO

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERÍAS
INGENIERÍA ELECTRÓNICA
Pereira- Risaralda
2016

Pereira 24 de Febrero 2016

“Aunque nos cueste llegar a la cima, llegaremos. No importa cuánto tardemos, el tiempo será seguro”

RESUMEN

Este documento pretende explicar las metodologías implementadas para reemplazar el sistema actual de programación de las máquinas SCOMAR A80, el cual consiste en un conjunto de partes mecánicas que mueven una cadena formada por eslabones y que en ella se distribuyen una cantidad de pines que activan simultáneamente un conjunto de bobinas de acuerdo con el sentido que la máquina lleve según el patrón de diseño que se requiera, estos pines pasan por un arreglo de finales de carrera fijos que al accionarse permite el paso de corriente a cada una de las bobinas conectadas al carro de la máquina. Para eso, se diseña una tarjeta electrónica que permita reemplazar el accionar mecánico que tienen los finales de carrera por uno electrónico reduciendo así el desgaste y las fallas, aumentando la velocidad de respuesta y la producción.

Para realizar el cambio de la cadena y los pines en la programación de la máquina se diseña una aplicación usando el software QT Creator el cual permite programar en lenguaje C++, con la gran ventaja de que la programación es orientada a objetos y con entorno gráfico, de esta manera poder tener una aplicación intuitiva que facilite el diseño de cada prenda y la programación de la máquina.

En la interacción entre el operario y la máquina, se dispone de una tarjeta de desarrollo (Raspberry) conectada a un conjunto de periféricos, como es una pantalla, teclado y conexión Wi-Fi en la cual se ejecuta la aplicación y se controla la tarjeta electrónica que acciona las bobinas. Para poder conocer el sentido que el carro de la máquina lleva se instalan 2 sensores inductivos ubicados en el eje del motor que permiten detectar cuando la máquina llega a cada extremo, de esta manera, realizar el cambio de la secuencia de salidas de la Raspberry a la tarjeta electrónica y así, hacer las diferentes variaciones en la activación de las bobinas.

AGRADECIMIENTOS

Agradezco la paciencia, empuje y esfuerzo de mis padres FRAY MARTIN ESCAMILLA, BEBI GUTIERREZ que siempre han querido que sobresalga en todos los retos a los cuales me he enfrentado y me han brindado las herramientas para poderlo lograr, a DIOS por guiar mi camino y poner a mi alcance todo para ser cada día una mejor persona y poder alcanzar mis sueños y mis metas. También a mi hijo ALEJANDRO ESCAMILLA que ha tenido que aguantar mi ausencia en ciertos momentos en los cuales los he dedicado a mi estudio y es la persona que sigue mis pasos por tal motivo es mi principal inspiración, a mi compañera, amiga y esposa ALEXANDRA COLORADO quien es la persona que ha estado acompañándome en todo momento y es quien me ha animado a poder terminar con este proyecto, mi hermano DAVID ESCAMILLA por su apoyo incondicional. Al M. Sc. MAURICIO HOLGUIN LONDOÑO por ayudarnos y creer en este proyecto además de guiarnos como director de nuestra tesis, al M. Sc. EDWIN ANDRES QUINTERO SALAZAR por su excelente gestión como director del programa de Ingeniería Electrónica y brindarnos todas las herramientas para poder hacer realidad este proyecto. Mis amigos, familiares, profesores y demás compañero de la Universidad que de una y otra manera han aportado a mi formación personal, técnica y profesional que sin ellos hoy no podría alcanzar esta meta.

DANIEL ESCAMILLA GUTIERREZ

Agradezco a Dios ya que solo él traza nuestro camino. A mis padres GABRIEL MOSQUERA y CONSUELO MUÑOZ, por darme el apoyo incondicional, porque han dado todo su esfuerzo para qué pueda cumplir mis metas, por la excelencia y sabiduría inculcadas siempre en mí. A mis hermanos, TOMAS MOSQUERA MUÑOZ y DAVID MOSQUERA MUÑOZ, por siempre estar ahí cuando lo requiero, por demostrarme día a día que siempre se puede ser mejor. Mis Abuelos JUAN BAUTISTA MUÑOZ y MARIA CENELIA JIMENEZ, por enseñarme, que a pesar de los años, se puede seguir soñando, creciendo y trabajando, MARIA OLEGARIA MOSQUERA y JOSÉ ALVARO MOSQUERA, que desde el cielo me brindan su fuerza. A las personas que me apoyaron durante mi estadía en la ciudad de Pereira, mi familia, mis amigos que se convirtieron en familia, mis compañeros de Universidad, profesores y demás personas que conocí durante este tiempo, ya que gracias a ellos disfrute cada día en esta ciudad y pase los mejores años de mi vida, hasta ahora. AL M. Sc. MAURICIO HOLGUÍN LONDOÑO por enfocarnos hacia el mejor camino durante casi toda nuestra carrera, por apoyarnos en este proyecto y compartir sus conocimientos y experiencia con nosotros. Al director del Programa de Ingeniería Electrónica, M. Sc EDWIN ANDRES QUINTERO SALAZAR, porque siempre se preocupó por brindarnos todas las herramientas a su alcance para lograr esta meta. A DANIEL ESCAMILLA GUTIERREZ, mi compañero de proyectos, mi amigo, agradezco su compromiso y su entrega y ante todo, su paciencia, aunque este camino no ha sido fácil siempre he contado con su apoyo. Esté es solo uno de los tantos proyectos que hemos realizado y sé que con la ayuda de Dios realizaremos muchos más.

SEBASTIAN MOSQUERA MUÑOZ

Tabla de contenido

Índice de figuras

Índice de tablas

CAPITULO I: INTRODUCCIÓN

1. PLANTEAMIENTO DEL PROBLEMA

La implementación de estrategias comerciales en la industria textil nacional, ha permitido que a través de las últimas dos décadas nuestro país se convierta en uno de los principales competidores de la región, siendo la industria textil-moda la que ocupa la segunda plaza en trayectoria del desarrollo industrial y económico del país [1]. Lastimosamente la lenta reconversión tecnológica en el sector, ha creado una serie de altibajos económicos que afectan y llevan a desparecer a los pequeños productores.

Datos obtenidos por el observatorio de la economía latinoamericana, demuestran que en la industria textil del país un porcentaje cercano al 80% son pequeños productores, 16% representa las medianas empresas y tan solo un 4% a las empresas grandes. A raíz de los diferentes acuerdos de cooperación internacional (TLC) firmados por el gobierno nacional, se han implementado varias estrategias empresariales para mitigar los efectos de los tratados, entre ellas se encuentran las *Misiones Tecnológicas Empresariales*, lastimosamente solo las grandes empresas como lo son Enka, Fabricato y Coltejer, por nombrar algunas, han logrado establecerlas, ya que estas estrategias tienen como principal fomento la inversión en maquinaria de alta tecnología. Estas mismas cifras demuestran que las pequeñas empresas, tienen un nivel de endeudamiento por encima del 70% de su capacidad y presentan grandes problemas de obsolescencia en su maquinaria [1].

Las máquinas de tejido de punto se dividen en diferentes clases, dependiendo del tipo de aguja que manejen, en este caso se encuentra la máquina rectilínea que genera la línea de tejido de punto por trama. La máquina rectilínea de punto genera el proceso característico que lleva su nombre, esta permite modificar la trayectoria del hilo de forma que produzca el enlace o tejido de punto, gracias al acoplamiento de varios elementos mecánicos [2].

El diseño actual de la máquina rectilínea, está compuesto por un sistema mecánico a través de unos eslabones y unos pines que pasan por un segmento de interruptores o switches electromecánicos, a medida que la máquina llega a un extremo el sistema mecánico mueve una cadena conformada por un conjunto unido de eslabones encargada de accionar los switches, según la configuración de pines que tenga cada eslabón de acuerdo al patrón del diseño. Este

sistema al ser tan mecánico está expuesto a derrame de sustancias como aceite, polvo, hilos que pueden afectar el accionar de cada interruptor, haciendo que la prenda que se está tejiendo salga con imperfecciones o en el peor de los casos ruptura en las agujas.

Adicional a esto para cambiar de diseño se necesita detener completamente la máquina para retirar la cadena y modificar los pines en cada eslabón, según el diseño que se requiera, esto hace que la máquina pierda productividad por el tiempo que dura detenida, así como también la manipulación directa del operario es frecuente que se presente daños en el diseño, como también en el sistema mecánico por mala ubicación de la cadena en el sistema.

Con el desarrollo de este proyecto de grado se quiere dar respuesta a la pregunta de si es posible diseñar un sistema de programación en máquinas rectilínea que permita mejorar la manera en cómo opera la máquina y facilitar la programación de la misma, ya que es una necesidad para la empresa Tejidos del Risaralda LTDA el contar con un dispositivo que reemplace completamente todo el sistema mecánico en la programación de las prendas en la máquina y facilitar la manipulación por parte del operario, permitiendo mejorar la producción y disminuir las fallas.

2. JUSTIFICACIÓN

La industria textil maneja una gran gama de procesos que se destinan para dar a los hilos y a los tejidos diferentes propiedades específicas. Uno de los procesos más importantes que se encuentran en la industria, es el tejido de punto, esta técnica se caracteriza por ser un enlace de un hilo, consigo mismo, mediante técnicas de bucles o mallas, estas técnicas conceden elasticidad y extensibilidad al tejido.

La automatización en los procesos textiles, se genera en esta industria desde la era de la revolución industrial, la máquina a vapor y el telar industrial, representaron el cambio principal en la automatización de los procesos de confección que aún sigue en expansión. Una integración de variables de automatización en entornos de desarrollo de textil, constituyen una alternativa que facilita la implementación de nuevos desarrollos tecnológicos, por esto, su uso podría ayudar a la creación proyectos enfocados a la regeneración de la industria textil a bajo costo.

El desarrollo de este proyecto permitirá a la empresa Tejidos del Risaralda LTDA contar un sistema de programación para sus máquinas rectilíneas modelo SCOMAR A80 completamente electrónico y programable a través de un teclado y una pantalla haciendo uso de un software

diseñado de acuerdo a las necesidades que se requieren, para monitorear y crear los patrones de diseños de una manera más sencilla e intuitiva que mejoren la manipulación por parte de los operarios y reduzcan las fallas presentadas por un mal diseño, este desarrollo permitirá reducir las labores de mantenimiento al no tener que reemplazar piezas mecánicas, ni adquirir elementos que ya son de difícil acceso en el mercado por ende su costo es amplio, permitiendo de esta manera en un largo plazo cubrir el costo del desarrollo de acuerdo al ahorro en dinero que se obtiene por la disminución en la compra de elementos que se dañan en el sistema actual de programación.

Con este proyecto se pretende desarrollar una aplicación que permita reemplazar todo el sistema de programación mecánico de la máquina a través del diseño de una tarjeta electrónica que sea la encargada de sustituir todos los switches electromecánicos y mediante una tarjeta de desarrollo conectada a una pantalla táctil y un teclado ejecute un software con la capacidad de crear los diferentes patrones de diseños y almacenarlos en el dispositivo, de igual forma cargarlos para que se ejecuten en la máquina y poder visualizar a través de la pantalla la cantidad de prendas y pasadas que lleva la máquina según el patrón de diseño que se esté ejecutando. Este software contiene además la lógica necesaria para controlar a través de los puertos digitales de la tarjeta de desarrollo la tarjeta de control diseñada, habilitando o deshabilitando secuencialmente cada una de las bobinas conectadas en ella de acuerdo al patrón de diseño. Como el carro de la máquina solo puede hacer cambios de estado de las bobinas en los extremos se ubican 2 sensores inductivos en un lugar estratégico que permita con exactitud enviar una señal a la tarjeta de desarrollo cada vez que la máquina llegue a un extremo.

3. OBJETIVOS

3.1 OBJETIVO GENERAL

Diseño e implementación de un sistema automatizado capaz de controlar y cambiar el accionar mecánico que posee la máquina rectilínea SCOMAR A80 para la programación de prendas, con base en la plataforma de desarrollo Raspberry Pi, incluyendo la respectiva interfaz de usuario para la creación de los patrones de diseño y para el control y monitoreo de la programación.

3.2 OBJETIVOS ESPECÍFICOS

- Estudiar el marco de antecedentes, requerimientos y restricciones de los componentes que integran el proyecto.

- Diseño e implementación del sistema de control electrónico que permita, teniendo en cuenta los requerimientos y restricciones, reemplazar el accionar mecánico en la programación de la máquina por uno electrónico.

- Diseño e implementación del sistema de programación e interfaz de usuario para el control y creación de los patrones de diseño de la máquina rectilínea SCOMAR A80.

- Integración de la Raspberry Pi con el sistema de programación y el sistema de control, puesta a punto de la máquina y pruebas de funcionamiento.

CAPITULO II: METODOLOGÍA

4. MÁQUINA SCOMAR A80

La máquina rectilínea SCOMAR A80 expuesta en la figura 1, es una máquina italiana diseñada especialmente para realizar cuellos y puños para camiseta, polos y buzos, que cuenta con una sola fontura, 4 guía Hilos, sensores de nudo, caída de tejido, desprendimiento de carro, ruptura de hilo, algunos modelos poseen un contador electrónico de piezas otros poseen un tacómetro, la máquina maneja una aguja galga 12. Las primeras máquinas fueron importadas a Colombia por FUKUTEX S.A.S empresa de Medellín, actualmente es uno de los principales distribuidores de maquinaria e insumos textil y de tejido de punto.

Figura 1. SCOMAR A80

Por su velocidad, rendimiento, dimensiones, peso y fácil mantenimiento, son muy utilizadas en la industria del tejido de puntos en Colombia

5. RASPBERRY PI

El módulo Raspberry Pi es una placa de desarrollo computacional de bajo costo (SBC), desarrollada en el Reino Unido por la Raspberry Pi Foundation. El desarrollo del proyecto inició en el año 2008 como un mecanismo para facilitar la enseñanza de la informática a la población más vulnerable, su aspecto físico se puede ver en la figura 2.

Figura 2. Raspberry PI

ARQUITECTURA

La placa base de este "minicomputador" en su modelo RASPBERRY PI B+ es del tamaño de una tarjeta de crédito, sus dimensiones son de 85x53 mm, está conformada por un chip Broadcom que contiene un CPU ARM, un GPU VideoCore IV y una memoria RAM de hasta 1 GB (en la última versión). Posee un puerto HDMI, un puerto de video RCA y un plug (minijack) de audio, además de los puertos USB cuenta también con una conexión LAN y puertos GPIO para vincular distintas interfaces. El sistema operativo se almacena en una tarjeta micro SD y su alimentación es de 5V y de 2000 mA (en sus últimos modelos). En las figuras 3 y 4 se puede observar el mapa general de componentes y las dimensiones del modelo B+.

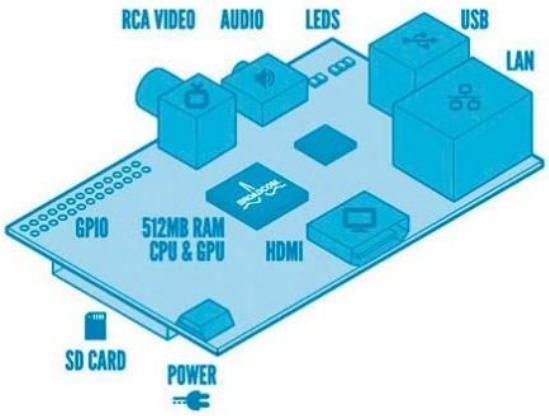

Figura 3. Mapa general de componentes de la Raspberry PI

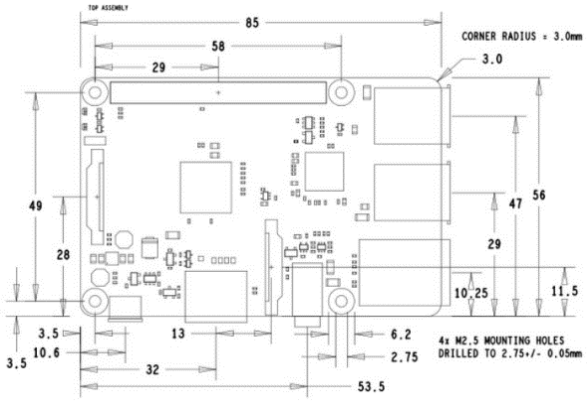

Figura 4. Dimensiones de la Raspberry PI B+

Para conocer los diferentes Modelos de la Raspberry, descripción de componentes y configuración e instalación del sistema operativo, ver ***Anexo 1****.*

6. PANTALLA TÁCTIL

La "Raspberry Pi 7" Touch Screen Display" es una pantalla táctil diseñada especialmente para su funcionamiento con la tarjeta de desarrollo computacional Raspberry Pi. En la figura 5 se puede apreciar la pantalla y sus accesorios

Figura 5. Raspberry Pi 7" Touch Screen Display y accesorios

Características generales

- Pantalla táctil de 7".
- Dimensiones de pantalla de: 194mm x 110 mm x 20 mm (tamaño Completo).

- Tamaño de pantalla visible: 155 mm x 86 mm.
- Resolución de pantalla: 800 x 480 pixeles.
- Conexión al Raspberry Pi utilizando un cable plano conectado al puerto DSI.
- Placa de adaptación: se utiliza para alimentar la pantalla y convertir las señales paralelas de la pantalla al puerto serie (DSI) en la Raspberry Pi.

7. RELÉ DE ESTADO SÓLIDO (SSR)

Los Relés de Estado Sólido (SSR. Solid State Relay) son unos componentes electrónicos formados por un conjunto de elementos independientes dentro de un circuito impreso. El relé permite el aislamiento eléctrico entre los circuitos de entrada y el circuito de salida, entre sus funciones se encuentran, los detectores de paso por cero, la protección frente a transitorios, como circuito de salida CA y como aislamiento asegurado generalmente a un acoplamiento óptico.

Características generales

- Conexión con o sin función de paso por cero
- Desconexión a I=0
- No ocasionan arcos ni rebotes al no existir partes móviles.
- Vida de trabajo óptima
- Frecuencia de conmutación elevada
- Facilidad de mantenimiento y Funcionamiento silencioso
- Control a baja tensión, compatible TTL/CMOS
- Circuito de entrada sensible a perturbaciones
- Requiere elementos de protección externos
- Sensibilidad a altas temperaturas y sobretensiones

MODELO HHG1-0/032F-20

El relé de estado sólido modelo HHG1-0/032F-20 cuyas características se indican en la Tabla 1 cuenta con una entrada DC y una salida del mismo tipo como se observa en la figura 6.

Figura 6. Relé de estado sólido (SSR) / HHG!0/032F-20

Nombre	Relé de estado sólido (SSR)
Modelo	HHG1-0/032F-20
Voltaje de Entrada	3 – 32VDC
Voltaje de Salida	12 – 200VDC
Corriente de carga	4A
Caída de Tensión	< 1.2VDC
Ruptura de Voltaje	1VDC
Temperatura	-30°~ 80°C
Resistencia de Aislamiento	500MΩ
Dimensiones	43x25.5x12(mm)
Tiempo de respuesta	≤ 5mS

Tabla 1. Características Relé de estado sólido (SSR) /HHG1-0/032F-20

8. DIODO ZENER

Un diodo Zener es básicamente un diodo de unión, pero construido especialmente para trabajar en la zona de ruptura de la tensión de polarización inversa. En la figura 7 se encuentran descritos su símbolo y aspecto físico.

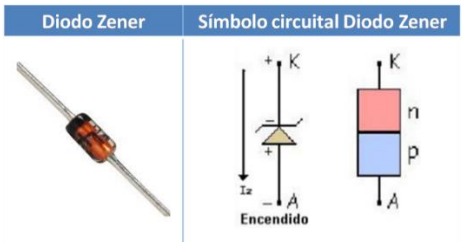

Figura 7. Símbolo y aspecto físico del diodo Zener

Su principal aplicación es como regulador de tensión, es decir, como circuito que mantiene la tensión de salida casi constante, independiente de las variaciones que se presenten en la línea de entrada o del consumo de corriente de las cargas conectadas en la salida del circuito. La relación voltaje – corriente, característica el diodo zener, se puede observar en la figura 8.

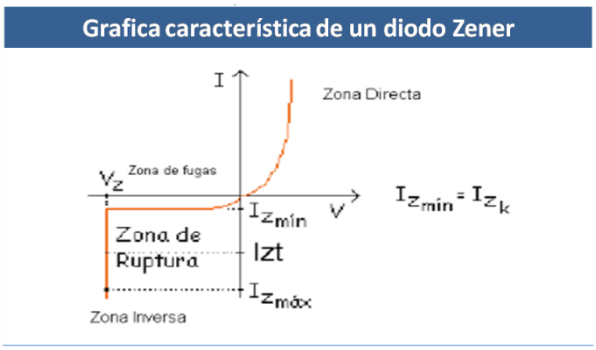

Figura 8. Curva característica del diodo Zener

Si a un diodo Zener se le aplica una corriente eléctrica del ánodo al cátodo (polarización directa) toma las características de un diodo rectificador básico, pero si se le suministra corriente eléctrica de cátodo a ánodo (polarización inversa), el diodo sólo dejara pasar una tensión constante.

9. OPTOACOPLADOR

Los optoacopladores son elementos electrónicos que permiten el aislamiento eléctrico establecido entre circuitos de entrada y salida, estos basan su funcionamiento en el empleo de un haz de luz de radiación luminosa para pasar señales de un circuito a otro sin conexión eléctrica.
Existen diferentes tipos de optoacopladores como lo son:

Fototransistor: Se compone de un optoacoplador con una etapa de salida formada por un transistor BJT.
Fototriac: Se compone de un optoacoplador con una etapa de salida formada por un triac
Fototriac de paso por cero: Optoacoplador en cuya etapa de salida se encuentra un triac de cruce por cero. El circuito interno de cruce por cero conmuta al triac sólo en los cruces por cero de la corriente alterna.

OPTOACOPLADOR 4N25

El optoacoplador 4N25, de tipo fototransistor, es formado por un diodo emisor de luz y una etapa de salida formada por un transistor de tipo BJT, la configuración de pines de este elemento es mostrado en la figura 9, así como sus características generales en la Tabla 2.

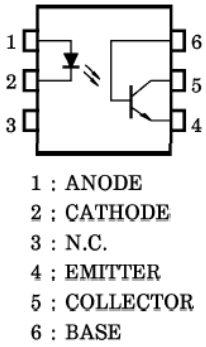

Figura 9. Configuración de pines Optoacoplador 4N25

Nombre	Optoacoplador fototransistor de propósito general, 4N25
Encapsulado	DIP de 6 pines
Voltaje de aislamiento pico	7500V AC
Resistencia de aislamiento	10Ω
Corriente continua de entrada del fotodiodo	60mA
Potencia de disipación del fotodiodo	120mW
Voltaje de Colector a Emisor	30V
Voltaje de Colector a Base	70V
Potencia de disipación del detector	150mW
Relación de transferencia de corriente (CTR)	20%
Tiempo de encendido y apagado no saturado	2μs

Tabla 2. Características generales del optoacoplador 4N25

10. SENSORES INDUCTIVOS

Los sensores inductivos o sensores de proximidad inductiva, incorporan una bobina electromagnética la cual es utilizada para detectar la presencia de un objeto metálico, estos sensores utilizan inducción electromagnética para generar y detectar las corrientes de pérdidas que se generan. Existen sensores blindados que son utilizados para posicionamiento y no blindados que son más utilizados en la detección de presencia de objetos.

MODELO PSN17-5DP

El PSN17-5DP es un sensor inductivo blindado del tipo PNP. En las figuras 10 y 11 se puede ver su forma y sus dimensiones, de igual manera sus características en la Tabla 3.

Figura 10. Sensor inductivo PSN17-5DP

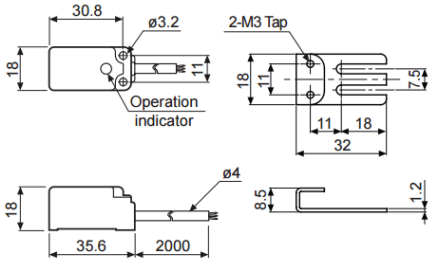

Figura 11. Dimensiones sensor inductivo PSN17-5DP

Nombre	Sensor de Proximidad Inductiva
Modelo	PSN17-5DP
Distancia de Detección	5mm ± 10%
Distancia de Ajuste	0 – 3.5mm
Alimentación	12 – 24 VDC
Corriente	Max. 10 mA
Respuesta en Frecuencia	700 Hz
Peso	71 g
Temperatura	-25°C ~ 70°C

Tabla 3. Características del PSN17-5DP

11. ENTORNO DE DESARROLLO

Las diferentes herramientas y librerías usadas en el desarrollo de este proyecto son descritas en este apartado.

11.1 DISEÑO DE CIRCUITOS IMPRESOS (PCB)

Los PCB (Printed Circuid Board) o circuitos impresos, son medios para la conexión de componentes electrónicos por medio de pistas o rutas de material conductor.

Los PCB fueron inventados en Inglaterra alrededor de 1936 por el ingeniero australiano Paul Eisler, para su uso en la detección de proyectiles durante la segunda guerra mundial. A mediados de 1950, la armada estadounidense desarrolló un proceso de auto-ensamblaje de circuitos impresos, en donde las patas de los componentes eran insertadas en una lámina de cobre con el patrón de interconexión y luego eran soldadas, aunque no eran realmente usados, ya que el método *"wire wrap"* o punto a punto era el más usado, ya que los componentes electrónicos que existían en la época contaban con patas de alambre [3].

Por lo general, la composición de los PCB's está dada por sustrato rígido, pistas impresas, resina de cubrimiento y pad's de plata (material conductor). Con el desarrollo de la laminación de las tarjetas y de la evolución tecnológica de los sistemas de grabado, se tiene la ventaja de contar con menor sensibilidad al ruido, mayor compatibilidad electromagnética, mayor robustez mecánica y la posibilidad de producción en serie. Los PCB pueden clasificarse de acuerdo a su material de confección, al tipo de montaje o su nivel de capas.

Para conocer más sobre diseño de circuitos impresos y el diseño de PCB con la herramienta de desarrollo EAGLE, ver ***Anexo 2****.*

11.2 LIBRERÍA QT

Lo que diferencia a Qt de una librería C++ cualquiera es que añade muchísimas funcionalidades a C++, cambiándolo de tal forma, que prácticamente crea un nuevo lenguaje de programación. Además, facilita la tarea de programar en C++ que en casos como en la programación de entornos gráficos, puede ser bastante ardua.

No obstante, es importante destacar que Qt no deja de ser C++, es decir, siempre se pueden usar librerías estándar o cualquier otra librería y la sintaxis de C++ común y corriente, por lo cual, es muy versátil. Por otra parte, existen bindings (asociaciones) de Qt para que programadores de otros lenguajes de programación puedan utilizar las librerías sin tener que dejar de usar su lenguaje habitual. Ejemplos de estos bindings son Qt Jambi (Java), PyQt (Python), PHP-Qt (PHP) o Qyoto (C#), entre otros muchos.

Otro punto clave de Qt es que se considera una biblioteca "multiplataforma", ya que permite programar el mismo código y utilizarse para crear programas para Linux, Windows, Mac OS,

etc., permitiendo realizar lo mismo que Java, pero siendo mucho más eficiente al no haber "máquina virtual" de por medio, sino código compilado expresamente para cada máquina.
Puesto que Qt se creó y creció como software libre (aunque en la actualidad hay disponible una versión comercial alternativa), existen muchas comunidades de programadores que crean librerías bastante útiles, como son el caso de Qextserialport o Qwt, y que al ser de código abierto, se pueden utilizar libremente y de manera gratuita.
Qt fue desarrollada inicialmente por la empresa noruega Trolltech (fundada por Haavard Nord y Eirik Chambe-Eng, dos estudiantes del instituto noruego de Tecnología) y posteriormente comprada por Nokia en 2008. Está disponible en tres tipos de licencia, GNU LGPL, GNU GPL y Propietaria (Nokia) y se ha utilizado para desarrollar software de todos los ámbitos como por ejemplo: Skype, Mathematica, Google Earth, Adobe Photoshop Album, etc.

Para conocer más acerca de la descripción de componentes de la librería qt, instalación e integración con la Raspberry, ver ***Anexo 3***.

11.3 LIBRERÍA WIRINGPI

WiringPi es una librería escrita en C y liberada baja licencia GNU LGPLv3, que puede ser empleada en varios lenguajes de programación, además de C y C++, con alguna pequeña modificación en forma de adaptación.
Su principal uso es en la programación de periféricos a través de los 28 pines de entradas y salida de propósito general de o General Purpose Input Output (GPIO) de la Raspberry Pi Modelo B+.
WiringPi, además, ofrece un comando que permite programar y configurar los pines de la GPIO, pudiendo efectuar la lectura y escritura de los pines desde la línea de comandos o incluso, incorporándola en un Shell script.

Para conocer acerca de la instalación y funcionalidades de la librería wiringpi, ver ***Anexo 4.***

CAPITULO III: DESARROLLO

12. ANÁLISIS Y REQUERIMIENTOS

Como se ha descrito en los apartados anteriores de este documento lo que se necesita es reemplazar el sistema mecánico de programación de la máquina SCOMAR A80 para ello es necesario reemplazar los 15 micro switches o finales de carrera que posee la máquina por un sistema electrónico, 12 de estos activan bobinas de 24V a 0.5A y 3 restantes envían una combinación de señales a 0V al variador conectado al motor para los diferentes cambios de velocidad parametrizados.

El diseño del esquemático y el PCB de cada una de las tarjetas electrónicas se desarrollaron en el software EAGLE 7.3.0 y la fabricación de los circuitos impresos fue desarrollada por la empresa Micro Led PCB en la ciudad de Bogotá.

SISTEMA MECÁNICO DE PROGRAMACIÓN DE LA MÁQUINA

La cadena de programación, mostrada en la figura 12, está formada por un conjunto de eslabones entrelazados entre sí con una distribución de pines que definen el patrón de diseño. Cada vez que se cambie el patrón de diseño se debe detener la máquina, retirar la cadena y manualmente distribuir los pines, en este proceso se pueden presentar errores al momento de ubicar los pines en posiciones incorrectas, hacer una distribución no adecuada de los pines haciendo que se activen al mismo tiempo bobinas que cumplen funciones de forma individual como por ejemplo los guiahilos, ocasionando daños físicos en la máquina o afectando el diseño de las prendas.

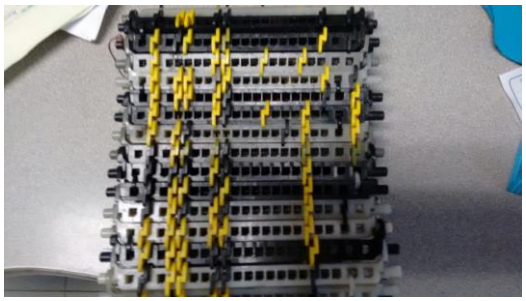

Figura 12. Cadena de programación máquina SCOMAR A80

El sistema que hace mover la cadena está compuesto por un conjunto de piezas mecánicas conectadas mediante una polea al eje del motor, haciendo que cada vez que la máquina llegue a un extremo la cadena mueva una posición, la distribución de pines de la cadena pasa por una fila de accionadores que al ser tocado por un pin sube y acciona cada uno de los switches electromecánicos para que conmute una señal DC que va a las bobinas o a las líneas del variador del motor. Como se puede apreciar en las figuras 13 y 14.

De igual manera se puede apreciar el desgaste que tiene los accionadores debido a mucho de los casos a la mala ubicación de los pines dentro de la cadena, en otros casos por elementos que se enredan en la cadena y ejercen una presión anormal sobre los accionadores haciendo que el resorte que traen se reviente o se dañe.

Figura 13. Conjunto de accionadores

Figura 14. Distribución y muestra de desgaste de los switches

13. SISTEMA DE CONTROL

El procedimiento de control del sistema de programación de máquinas rectilíneas, está compuesto por una etapa de potencia, una etapa de acondicionamiento de sensores y otra de distribución de conexiones.

13.1 ETAPA DE POTENCIA

La función de esta etapa es recibir las señales provenientes de la Raspberry de acuerdo a la secuencia de activación o desactivación que lleve el patrón de diseño, esta tarjeta debe soportar la conmutación de una señal de 24V y un consumo de corriente por línea de control de 0.5A, en total son 15 conmutadores electrónicos los que se deben implementar en esta etapa.

13.1.1 Tarjeta de relés

Para el diseño de la tarjeta se optó por usar relés de estados sólidos en vez de diseñarlo con componentes electrónicos, puesto que estos últimos generarían más puntos de fallos al requerir más componentes en la tarjeta de igual forma el tamaño de la misma sería aun mayor y más dificultoso el diseño del PCB. En las figuras 15 y 16, se observa el diseño de esta tarjeta.

Los requerimientos tenidos en cuenta para el diseño de la tarjeta se enumeran a continuación:

- Disponer de más módulos para la conexión de los relés de estado sólido para interconectar futuros elementos a la tarjeta.
- Aislar totalmente los niveles de tensión de la parte lógica (baja potencia) emanada por los puertos GPIO de la Raspberry Pi, de los niveles de tensión (alta potencia) provenientes de la máquina.
- La tensión máxima que entrega las salidas del puerto GPIO son 3.3V y corriente máxima que debe fluir por cada pin del puerto GPIO es de 50mA.
- Los niveles de tensión de las salidas del puerto GPIO son HIGH (un voltaje entre 2 y 3.3 V) o LOW (entre 0 y 0.8 V).

- Los relés son activados por un optoacoplador, teniendo en cuenta eso cada salida del puerto GPIO debe alimentar el led bajo una corriente de 10mA aprox.

Según la ley de Ohm la intensidad de corriente eléctrica (I) es igual a la tensión (V) divida entre la resistencia (R). Si se desprecia la resistencia del circuito, se tendría que R = 0 y teniendo en cuenta que el resultado de una división entre cero tiende a infinito, sin la resistencia provocaríamos un cortocircuito que dañaría el LED o incluso la propia Raspberry.

$$I = \frac{V}{R}$$

A la hora de escoger la resistencia se debe tener en cuenta la intensidad de polarización de un led típico que está entre los 10 y los 40mA. Considerando que la eficiencia de un LED es superior cuanto menor es la tensión que circula a través de él, la ecuación que define el valor de la resistencia está dada por:

$$R = \frac{V}{I} = \frac{3.3\ V}{0.010\ A} = 330\Omega$$

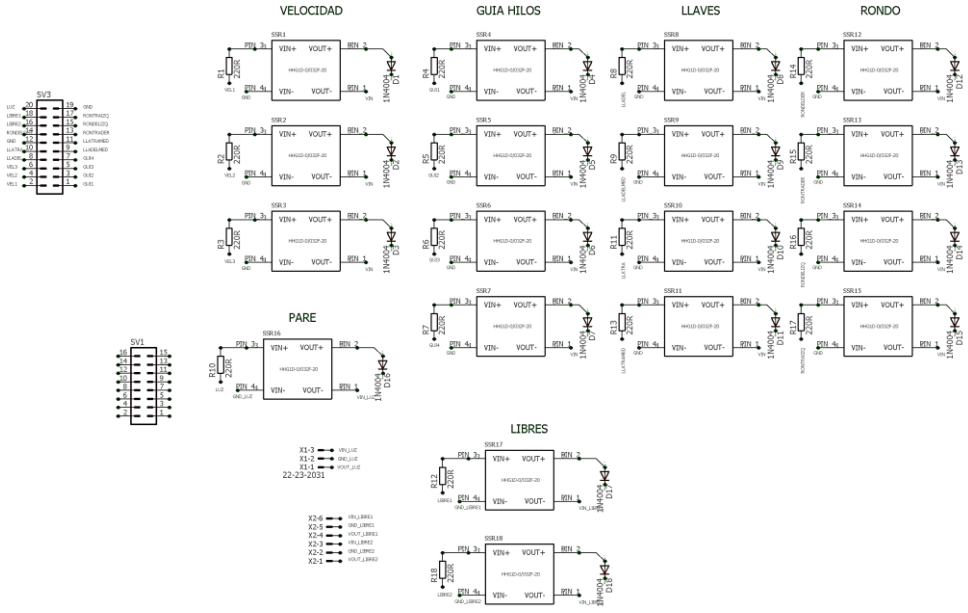

Figura 15. Esquemático de la tarjeta de relés

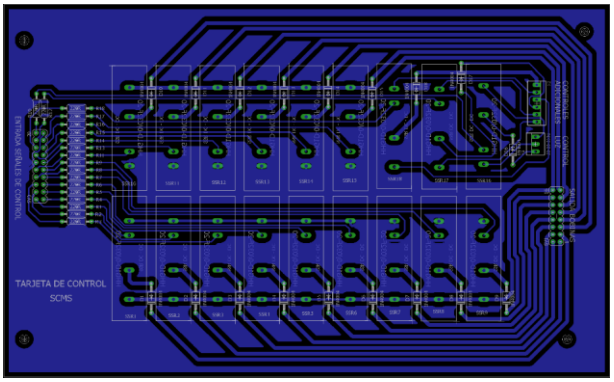

Figura 16. PCB de la tarjeta de relés

13.1.2 Tarjeta de distribución de conexiones

Para separar la etapa de potencia y la de adquisición de la señal de los sensores inductivos se diseñó una tarjeta donde llega todas las líneas del puerto GPIO de la Raspberry a través de un cable ribbon de 40 hilos y las distribuye en diferentes puertos de salida a las entradas de la tarjeta de potencia conector ribbon de 20 hilos y la de los sensores conector ribbon de 4 hilos. En la figura 17, se ve el diseño del PCB de la tarjeta

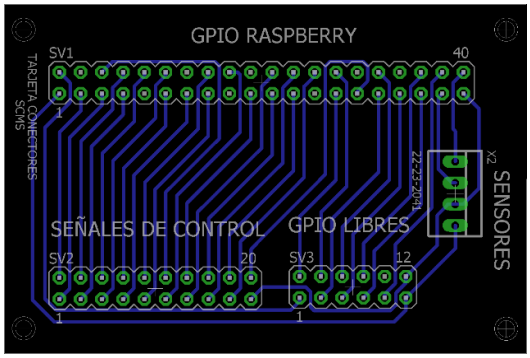

Figura 17. PCB de la tarjeta de distribución de conexiones

13.1.3 Tarjeta de acondicionamiento de sensores

Los sensores entregan una Señal de 24V que se debe acondicionar puesto que los puertos GPIO de las Raspberry manejan un voltaje de 3.3V para ello se debe introducir un circuito que baje el voltaje de los sensores a ese rango, para poder hacerlo se utilizó un par de diodos Zener 1N4728

los cuales tienen un voltaje de corte de 3.3V voltios, para aislar la parte de potencia y no tener que acoplar la tierra de la Raspberry con la alimentación de los sensores, se conectó un optoacoplador 4N25, cuando el sensor emite una señal el optoacoplador conmuta cero voltios al puerto de la Raspberry proveniente de la alimentación de la tarjeta, cuando el sensor no envía señal este permanece en 3.3V. El esquemático del circuito se muestra en la figura 18 y el PCB en la figura 19.

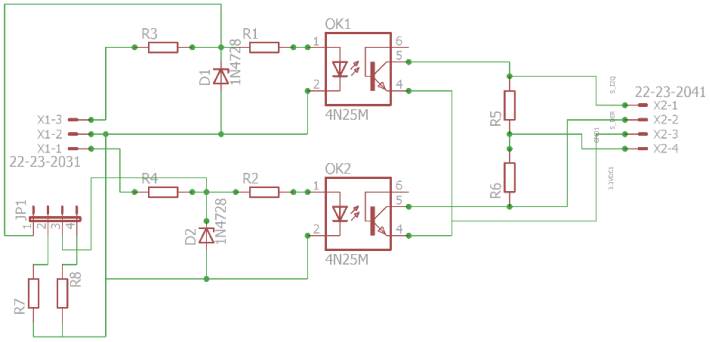

Figura 18. Esquemático circuito de acondicionamiento de sensores inductivos

Las resistencias R7 y R8 no están conectadas directamente en el circuito sino que se hace mediante jumper, se tuvieron en cuenta en el circuito por si en algún caso se requería agregar resistencias de Pull-Up a la entrada del opto acoplador.

El valor de las resistencias R3 y R4 que son las resistencias limitadoras de corriente para el diodo Zener se obtienen a través del siguiente cálculo:

$$Rs = \frac{Vin - Vz}{I_{Lmax} + I_{Zmin}}$$

Los datos de I_{Zmin} son obtenidos de la hoja de datos del diodo Zener, la corriente I_{Lmax} viene dada por el led que está conectado internamente en el opto acoplador que su consumo varía entre 10mA y 50mA.

$$Rs = \frac{24V - 3.3V}{0.05A + 0.0303A} = 257.78\Omega$$

Las resistencias R1 y R2 que son las que limitan la corriente que pasa al diodo led interno del optoacoplador se calcula de la siguiente manera:

$$R = \frac{V}{I} = \frac{3.3\,V}{0.010\,A} = 330\Omega$$

Las resistencias R5 y R6 son resistencia de Pull-Down que pueden tomar un valor entre 1K y 10K, un alto valor de resistencia de pull-down puede limitar la velocidad a la que el pin puede

cambiar de estado y como la máquina tiene una alta velocidad entonces la reacción de la señal al detectar cada sensor debe ser lo más rápido posible por eso se escoge el valor mínimo en la resistencia de pull-down.

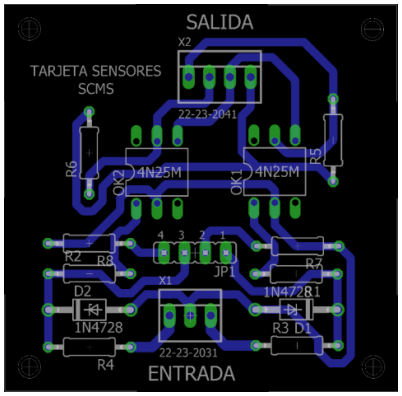

Figura 19. PCB circuito de acondicionamiento de sensores inductivos

14. SISTEMA DE PROGRAMACIÓN

Para la creación de un nuevo proyecto en Qt Creator, ver ***Anexo 5****.*

14.1 REQUERIMIENTOS Y RESTRICCIONES

Estos son los requerimientos mínimos que se debe tener en cuenta en el desarrollo de la interfaz de control.

- La aplicación debe permitir almacenar los archivos de programación, para ello el sistema de almacenamiento debe ser amplio.
- La interfaz debe manejar eventos y mensajes de alerta ante posibles errores como doble lectura de sensor, producción terminada, mensajes de alerta cuando se desee eliminar algún elemento como archivos o líneas en el patrón de diseño.
- Permitir cambiar el valor de los ciclos del patrón de diseño sin necesidad de cargar nuevamente un programa y tener que detener la máquina.
- Tener la capacidad de continuar con la programación aun cuando la máquina se apague y se encienda si el contador de piezas aún no ha finalizado.

- Permitir ir creando patrones de diseño sin que se afecte el funcionamiento de la máquina ni se vea alterado el rendimiento de la aplicación.
- Mostrar en pantalla el sentido que lleva el carro de la máquina, de igual manera las líneas del patrón de diseño que va efectuando en cada recorrido, así mismo mostrar el contador de piezas y los ciclos.

14.2 ARQUITECTURA

La arquitectura de funcionamiento del sistema se muestra en la figura 20.

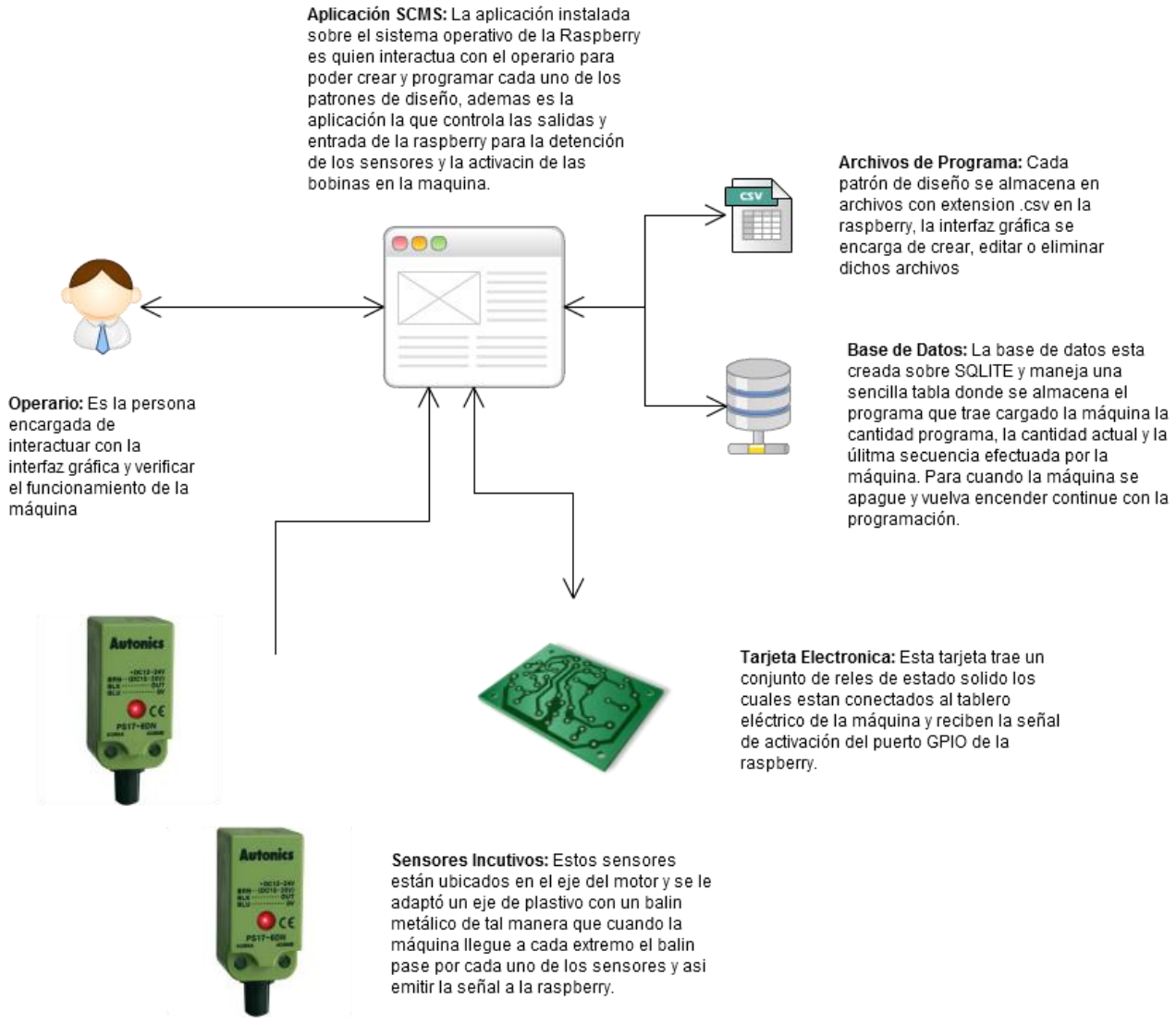

Figura 20. Arquitectura del sistema

14.3 LÓGICA DE CONTROL

En este apartado se describen las clases utilizadas para el control lógico del sistema.

14.3.1 Cantidad

Esta clase se encarga de ajustar la cantidad de prendas que se desee realice la máquina. En la figura 21 se muestra el diagrama de flujo de la clase cantidad

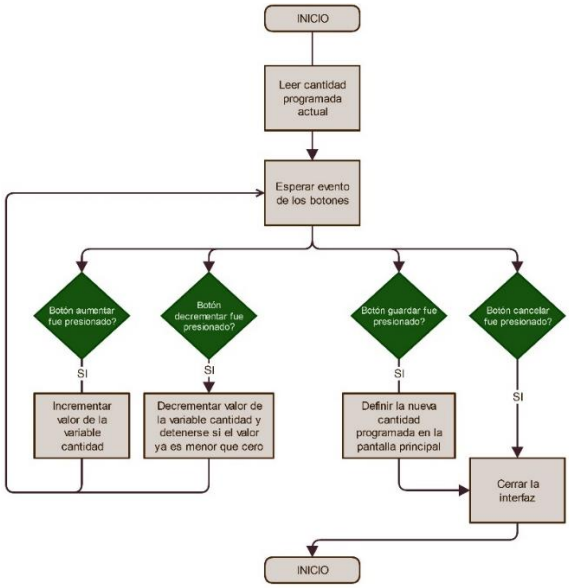

Figura 21. Diagrama de flujo de la clase cantidad

14.3.2 Thread Sensores

Esta clase es un hilo o Thread que constantemente está leyendo el estado de los sensores, como este proceso se ejecuta en segundo plano, no afecta el funcionamiento del programa principal. El diagrama de flujo de la clase, se muestra en la figura 22.

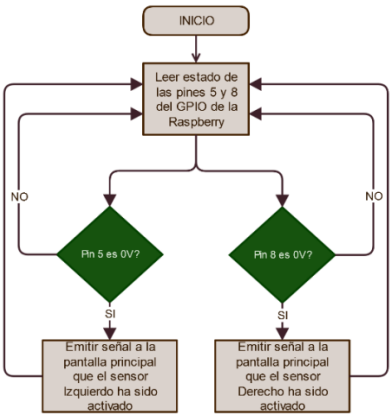

Figura 22. Diagrama de flujo de la clase thread sensores

14.3.3 Cargar programa

Esta clase crea y edita los diferentes programas y patrones de diseño, que se cargan en la máquina. Como se observa en el diagrama expuesto en la figura 23

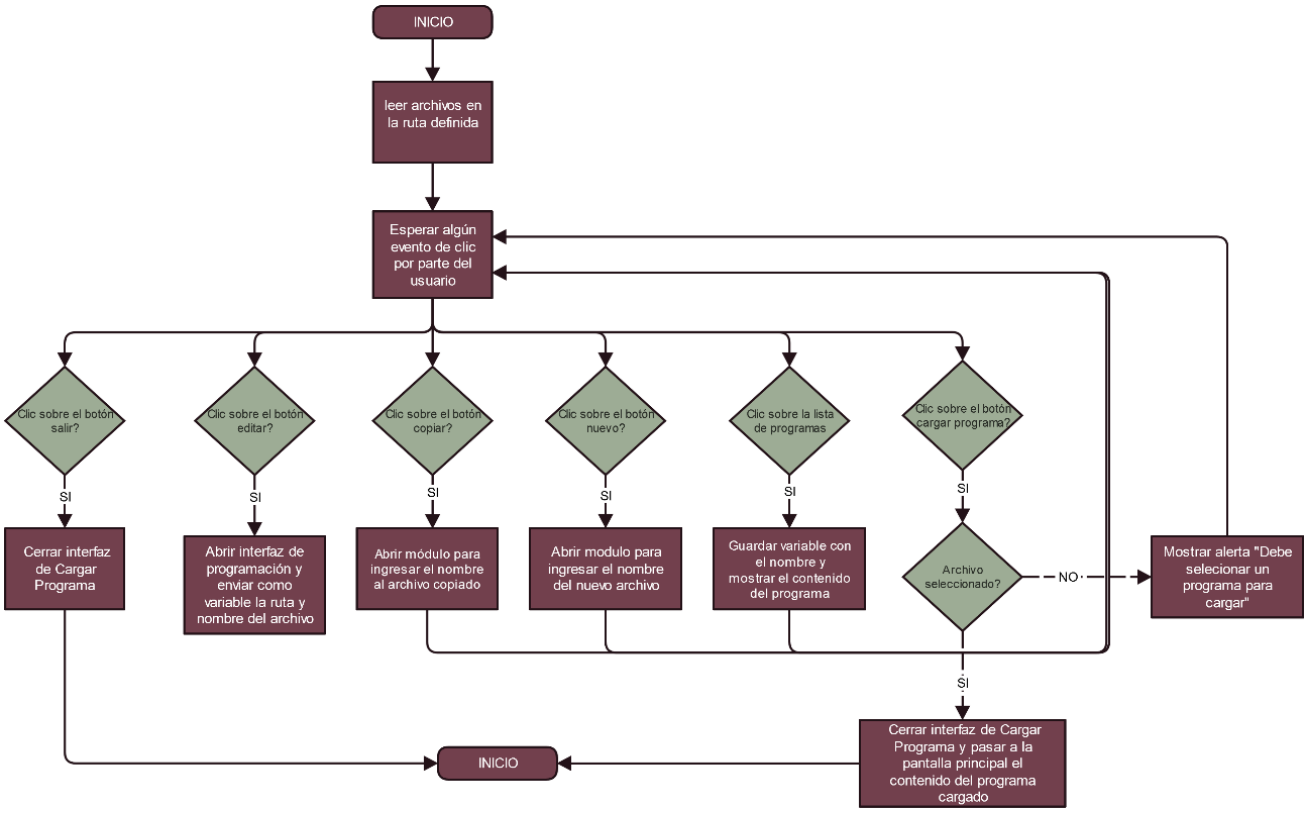

Figura 23. Diagrama de flujo de la clase cargar programa

14.3.4 Copiar archivo

Esta clase que se describe en la figura 24, copia el contenido de un archivo con el patrón de diseño ya creado, en uno nuevo según el nombre de archivo que el usuario ingrese.

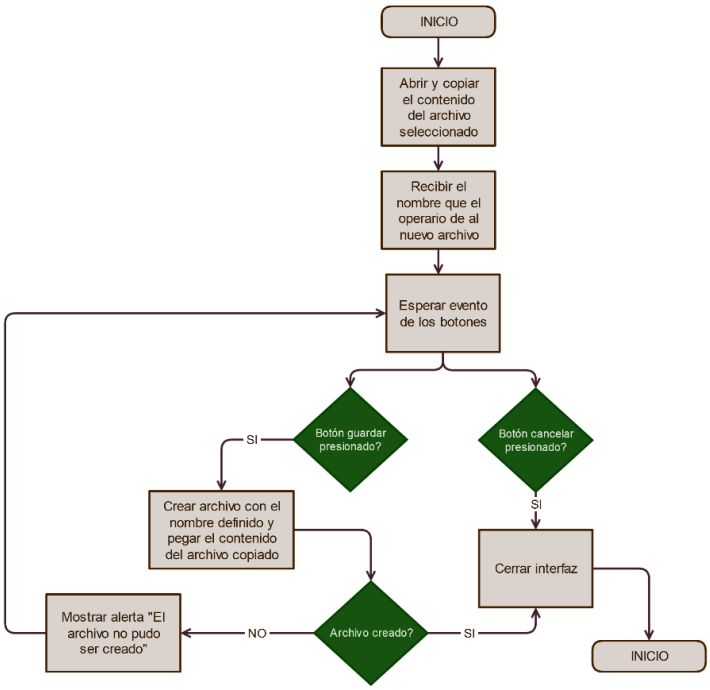

Figura 24. Diagrama de flujo de la clase copiar

14.3.5 Guardar archivo

Esta clase se encarga de crear nuevos archivos de programación sin contenido en su interior, para luego ser editados con el patrón de diseño que se requiera a través del módulo de programación. En la figura 25 se muestra el diagrama de flujo de esta clase.

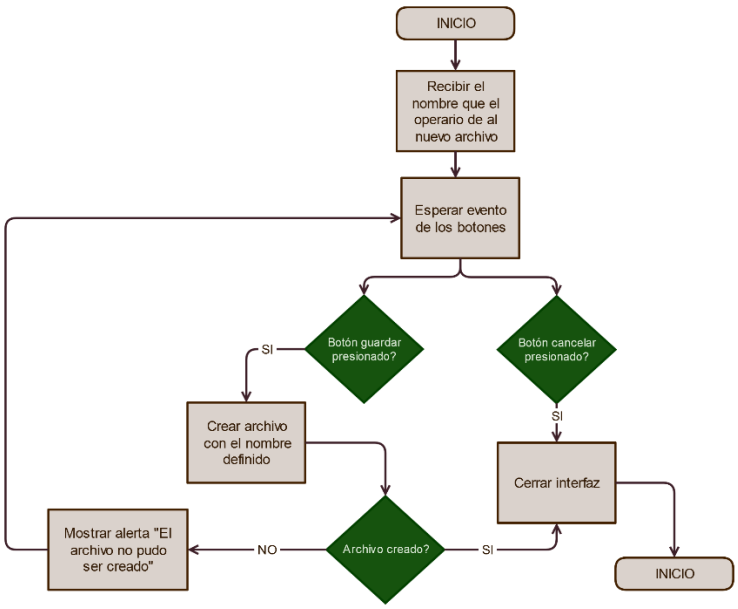

Figura 25. Diagrama de flujo de la clase guardar archivo

14.3.6 Modificar ciclos

Esta clase lee los ciclos definidos en el patrón de diseño y permite modificar su valor guardando los cambios directamente en el archivo sin necesidad de editarlo en el módulo de programación. El diagrama de flujo de la clase, es mostrado en la figura 26.

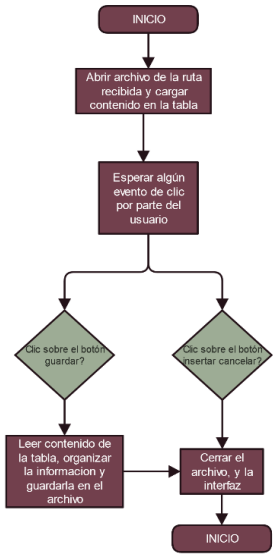

Figura 26. Diagrama de flujo de la clase modificar ciclos

14.3.7 Módulo programación

Esta clase permite crear y editar los patrones de diseño para luego guardarlos en un archivo de programación. En la figura 27 se muestra el diagrama de flujo.

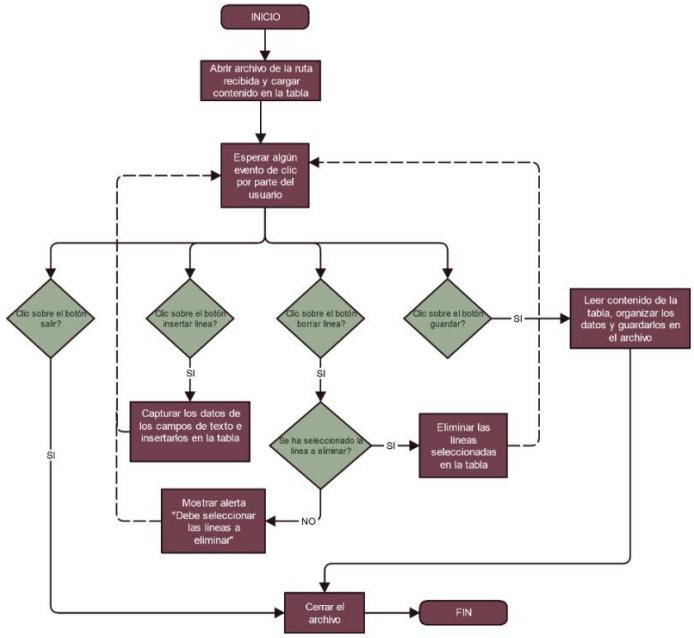

Figura 27. Diagrama de flujo de la clase programación

14.4 INTERFAZ GRÁFICA

Este apartado describe el diseño de toda la interfaz gráfica que interactúa con el usuario y ejecuta las operaciones lógicas de control.

14.4.1 Pantalla principal

Esta interfaz que se muestra al iniciar la aplicación están disponibles las diferentes opciones básicas que posee, basta con presionar cada uno de los botones para acceder a los diferentes módulos tal como se ven en la figura 28 y se describen más adelante.

Figura 28. Pantalla principal de la interfaz de programación

Los demás botones que se muestran desactivados son para trabajos futuros de nuevas funcionalidades que se le requieran dar a la aplicación.

1. Cargar Programa

Este módulo permite crear, editar y cargar los diferentes programas con los patrones de diseño que se requieran en la elaboración de cada tejido.

2. Ciclos

Este módulo permite modificar los diferentes ciclos que se hayan creado en el patrón de diseño del programa cargado en la máquina.

3. Cuenta Prendas

Este módulo permite definir o editar la cantidad de prendas programadas en la producción del programa cargado en la máquina.

4. Apagar

Este botón envía una señal de apagado a la Raspberry haciendo que la aplicación se cierre y el sistema operativo se apague.

5. Enviar a Corte

Este botón envía una señal al programa para que inicie en la posición 1, en los casos que se requiera iniciar nuevamente una prenda.

6. Área De Monitoreo

Esta sección indica la dirección que lleva el carro y la secuencia que va llevando el programa según el patrón de diseño que se haya generado, además indica la cantidad de piezas programadas y las que va completando la máquina, de igual forma muestra el conteo de los ciclos que se hayan creado en el diseño.

14.4.2 Interfaz cargar programa

Este módulo mostrado en la figura 29 permite crear, eliminar, copiar, editar los diferentes patrones de diseños guardados localmente en la Raspberry para luego ser cargados en la máquina para que esta lo ejecute.

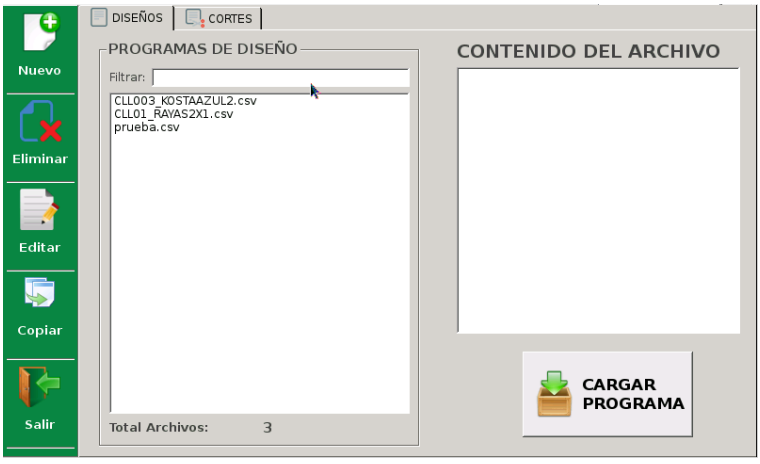

Figura 29. Pantalla de creación y cargue de programas de la interfaz de programación

14.4.2.1 Crear un nuevo programa

Esta interfaz mostrada en la figura 30 permite guardar un patrón de diseño en el archivo de programa.

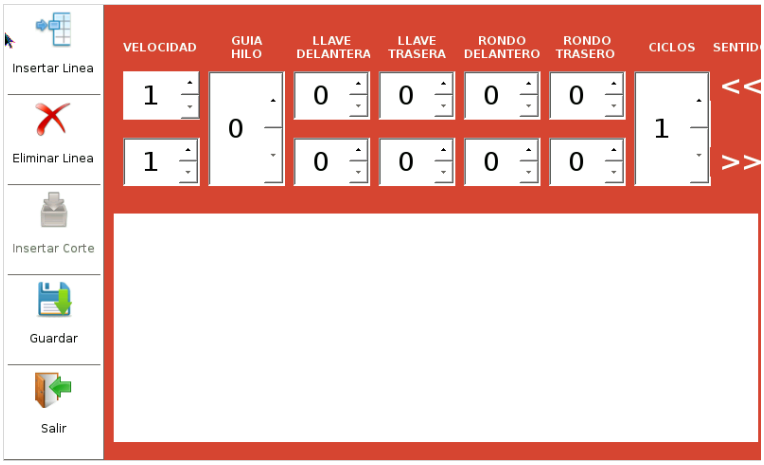

Figura 30. Módulo de edición

Para crear el patrón de diseño primero se debe configurar las funciones que la máquina va ejecutar en cada pasada o sentido de ida y vuelta, definiendo la velocidad, guía hilo, estado de las llaves así mismo como los rondós que traducido a lo que ejecuta la máquina son los diferentes bobinas que deben ser activadas según el sentido que lleve la máquina, la opción ciclos permite definir cuantas veces se repetirá la misma pasada una vez definida la configuración para esa pasada se presiona el botón “Insertar Línea” para ir creando el patrón de diseño así mismo hacerlo con los diferentes cambios o configuraciones que tendrá ese programa.

14.4.2.2 Editar programa

Para editar un programa se debe seleccionar de la lista y luego presionar el botón “Editar” para que se abra el módulo donde se puede cambiar el patrón de diseño de dicho programa, cuando se selecciona o se hace clic sobre cada uno de los programas de diseño de la lista al lado derecho se puede observar el contenido que trae dicho programa de esta manera sin necesidad de abrirlo saber cómo esta creado el patrón de diseño y así saber cuál que se debe modificar antes de editarlo.

14.4.2.3 Eliminar programa

Para eliminar un archivo se debe seleccionar de la lista y presionar luego el botón “Eliminar” esto abre un diálogo confirmando si se desea eliminar el archivo, al ser confirmado el archivo se elimina totalmente.

14.4.2.4 Copiar programa

Este módulo permite de igual forma copiar el contenido de cualquier archivo dando la opción de escoger el nuevo nombre que tendrá, para ello se debe seleccionar de la lista el archivo y luego presionar el botón “Copiar” esto abre un diálogo donde se pide el nuevo nombre para el archivo.

14.4.2.5 Cargar el programa

Para cargar un programa en la máquina se debe seleccionar de la lista y luego presionar el botón “CARGAR PROGRAMA” al hacerlo se cierra el MÓDULO actual y pasa a la pantalla principal, mostrada en la figura 31, donde se abre un diálogo solicitando la cantidad de piezas a programar.

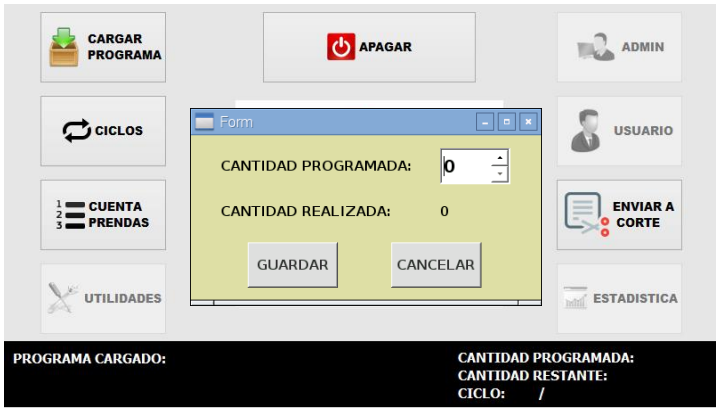

Figura 31. Definir cantidad

Una vez el programa es cargado se activa el sistema que verifica el estado de los sensores y la máquina puede ser puesta en funcionamiento. A medida que esta opere y active cada uno de los sensores en el área de monitoreo se podrá observar el sentido que lleva, la secuencia que se

encuentra realizando según el patrón de diseño, el incremento en los ciclos y en el conteo de las prendas realizadas, como se ve en la figura 32.

Figura 32. Pantalla principal en funcionamiento con un programa cargado

Cuando el conteo de las prendas realizadas es igual a la cantidad programada, el sistema mostrara un mensaje avisando que la producción ha sido terminada y además da la opción de volver a realizar o no la misma producción.

14.5 INICIO DE LA APLICACIÓN

La aplicación se inicia automáticamente cuando se enciende la Raspberry para ello se genera un script que lo lance cada vez que el sistema operativo arranque. El algoritmo de funcionamiento de la aplicación cada vez que se ejecuta se describe en el diagrama de flujo de la figura 33.

Figura 33. Diagrama de flujo inicio básico del sistema

Una vez la máquina se encuentre en posición con el carro en el extremo derecho se da inicio a la máquina presionando el botón que arranca el motor, los sensores inductivos le indican a la Raspberry cuando llegue a cada extremo para que aumente la secuencia en el programa del diseño. En el siguiente diagrama de flujo de la figura 34 se puede apreciar mejor el funcionamiento.

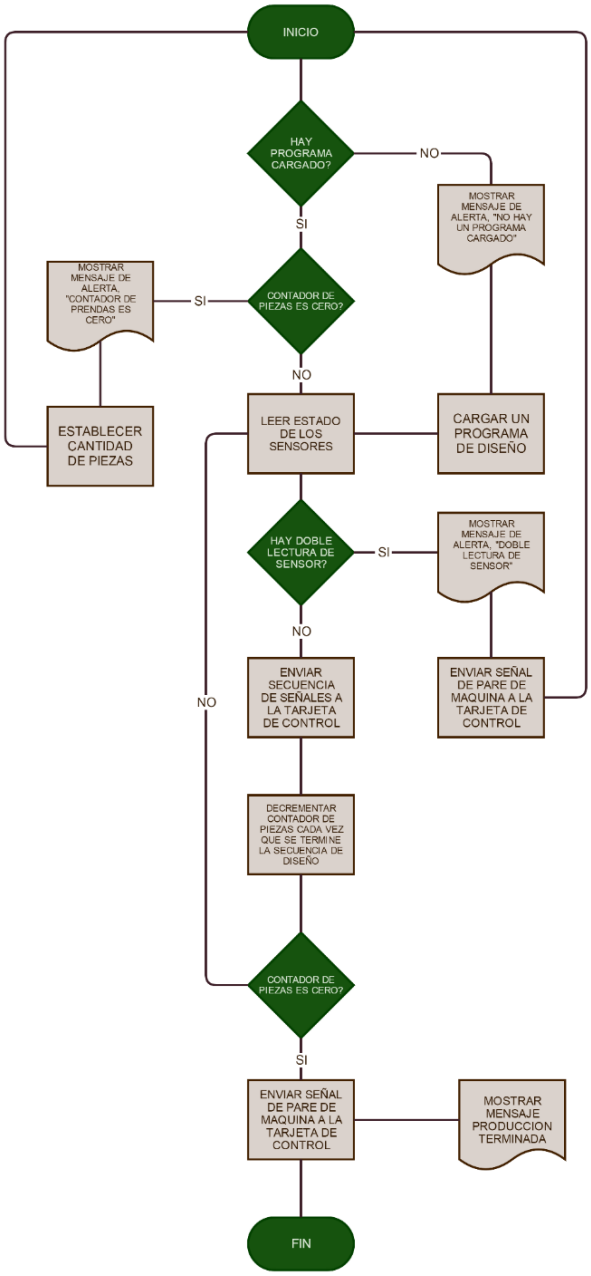

Figura 34. Diagrama de flujo del funcionamiento del sistema

CAPITULO III: ANÁLISIS DE RESULTADOS

Para efectos de resaltar y de mostrar de manera ordenada los resultados del proyecto, se muestra a continuación las diferentes etapas de montaje, pruebas y puesta a punto del sistema y las conclusiones finales del mismo.

15. MONTAJE

15.1 MÓDULO DE OPERACIÓN

El módulo de operación está compuesto por la tarjeta Raspberry Pi y su pantalla táctil Raspberry Pi 7” Touch Screen Display. Estas dos se conectan a través de una placa de adaptación en donde se utiliza un cable USB - micro USB para alimentar la pantalla y un cable plano para convertir las señales paralelas de la pantalla al puerto serie (DSI) en la Raspberry Pi como se ve en la figura 35. Para la protección de los elementos, se realiza la creación de una caja o carcaza plástica con medidas de 20x15x5 cm, en la cual se inserta el módulo de operación.

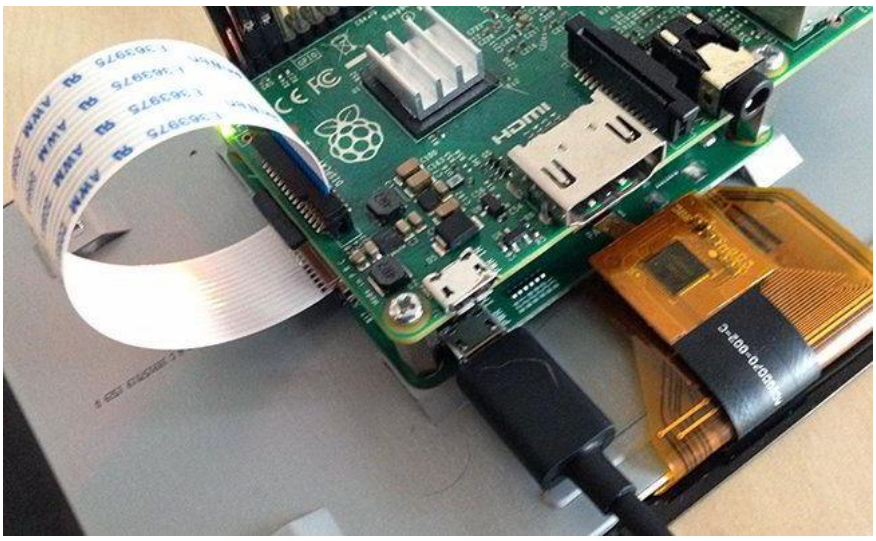

Figura 35. Conexión pantalla y Raspberry Pi

Finalmente el módulo se instala en una base plástica, la cual esta sujetada a la máquina, las medidas de la base son mostradas en la figura 36, en las figuras 37 y 38, se puede observar el módulo de operación, instalado en la máquina.

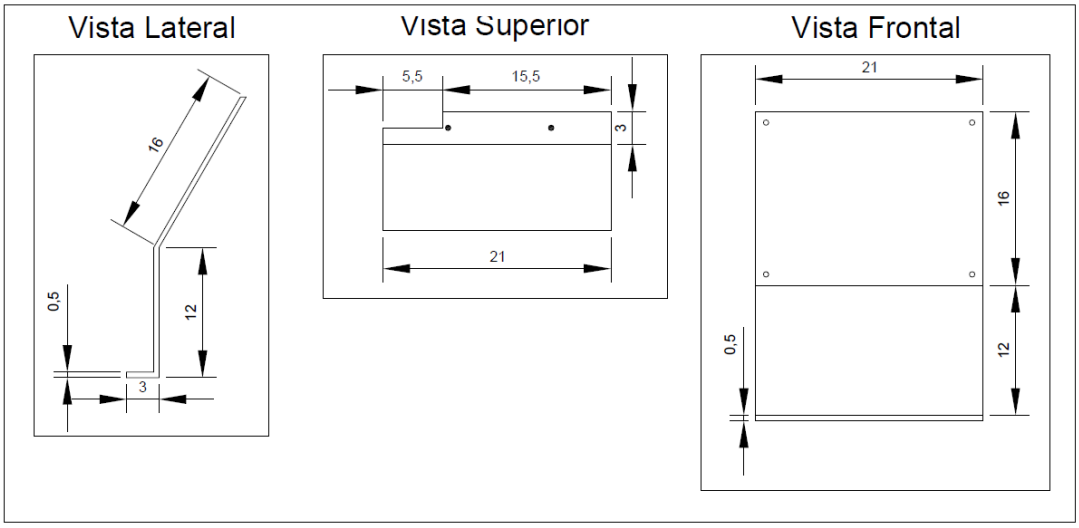

Figura 36. Medidas base de instalación

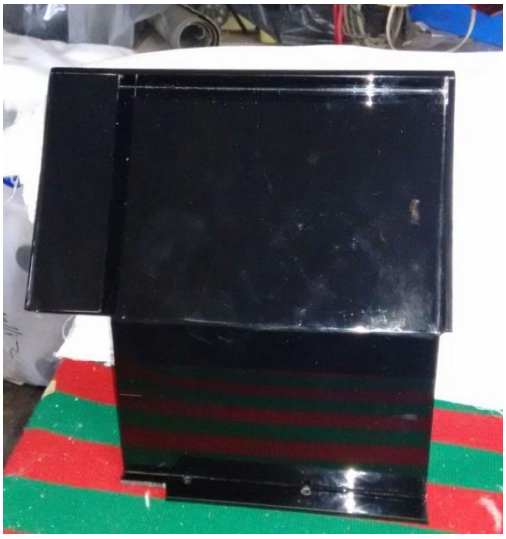

Figura 37. Pantalla en base

Figura 38. Módulo de operación

15.2 MÓDULO DE SENSADO

Para la instalación de los sensores inductivos, los cuales detectan el sentido del carro tejedor, fue necesaria la creación de una base metálica, en la cual se instalan los dos sensores en el extremo superior e inferior. Las dimensiones de la base son mostradas en la figura 39.

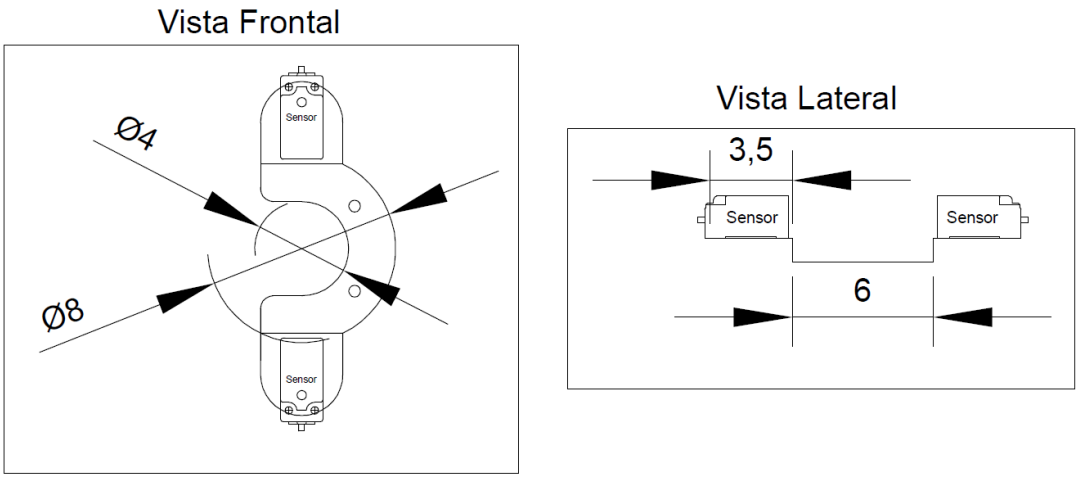

Figura 39. Medidas base sensores

La base es instalada alrededor del extremo saliente del eje de la caja de velocidad de la máquina SCOMAR A80. En este mismo eje, se instala un anillo plástico el cual posee un punto metálico como se muestra en la figura 40.

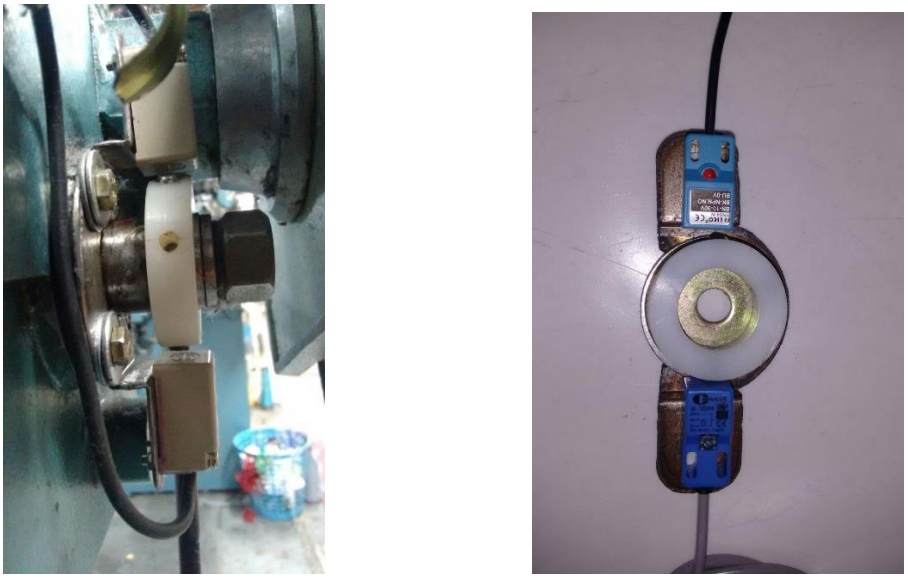

Figura 40. Base y soporte de los sensores

El posicionamiento de este punto metálico al ser detectado por los sensores inductivos, determina la dirección en la cual está el carro tejedor. Si el sensor superior detecta el punto metálico, indica que el carro llegó hasta el extremo izquierdo de la máquina, así mismo si el sensor inferior es el que detecta el punto metálico, realiza la indicación de que el carro tejedor

terminó su desplazamiento de izquierda a derecha. Las señales de los sensores, están conectadas a la tarjeta de acondicionamiento de sensores. (**Ver numeral 13.1.3**)

15.3 MÓDULO DE CONTROL

Después de la creación de las PCB definidas en la etapa de control y sensado (**ver 13.1**) estas son instaladas en la parte posterior de la máquina ASCOMAR A80 ya que esta cuenta con una caja de distribución de conexiones, mostradas en las figuras 41, 42 y 43 en donde están conectadas todas las señales, borneras, contactores y demás elementos eléctricos que permiten el funcionamiento de la máquina.

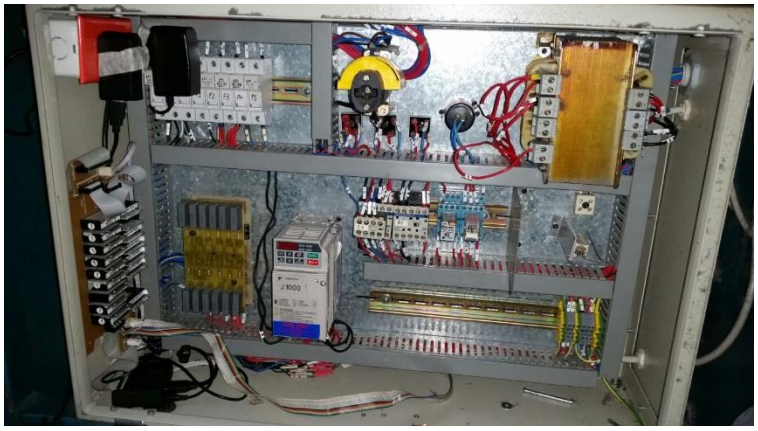

Figura 41. Caja de distribución de conexiones SCOMAR A80

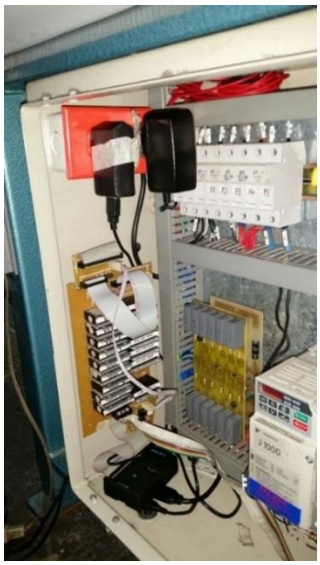

Figura 42. Posición Tarjetas de Control

Figura 43. Tarjeta de Relés

16. PRUEBAS Y PUESTA A PUNTO DEL SISTEMA

Una vez conectados todos los elementos que conforman el sistema electrónico de programación para la máquina SCOMAR A80, se procede a realizar el encendido del mismo, en la pantalla de inicio de la aplicación que se observa en la figura 44, se cargar un programa prediseñado, con el cual se realizan las pruebas de las conexiones y las señales emitidas y percibidas por todos los elementos que conforman el sistema.

Figura 44. Pantalla principal de la aplicación

PROGRAMA DE TEJIDO PARA PRUEBAS

Para la ejecución de la prueba, se determinó generar un programa llamado "CUCUTA_NEGRO.csv" para la fabricación de cuellos para tallas S y M con medidas de 39 x 8.5 cm para cada uno. Para este proceso de elaboración, la máquina SCOMAR A80 utiliza 410 agujas.

Explicación del programa:

Este programa tiene los patrones de diseño para crear cuellos en acrílico con 3 colores, 2 colores de ½ cm y un color de fondo. El programa se divide en 5 partes diferentes, dentro de una sola línea de programación:

5;4;1;1;0;0;1;C 3;4;1;0;0;0;X;C 3;4;1;0;0;0;1;C 3;4;1;0;0;0;X;C 3;4;1;1;0;0;1;C 2;4;1;1;0;0;X;C 2;0;1;0;0;0;1;C	Esta parte del programa hace el corte o separación de cada cuello
2;0;0;0;0;0;X;C 1;3;1;1;0;0;1 2;3;1;0;1;0;X 5;3;0;1;0;1;3 5;3;1;0;1;0;X 5;0;0;0;0;0;1 5;0;0;0;0;0;X 3;3;1;1;0;0;1 5;3;1;1;0;0;X 3;3;0;1;0;1;1	Esta parte del programa hace el rondo o el principio de cada cuello y hace uso de un color de hilo enhebrado en el guiahilo 3
5;3;1;0;1;0;X 5;1;1;1;0;0;2	Esta parte del programa hace un cambio de color usando el guiahilo 1
5;1;1;1;0;0;X 5;3;1;1;0;0;2	Nuevamente cambia de color usando el guiahilo 3
5;3;1;1;0;0;X 5;2;0;1;0;1;1 5;2;1;0;1;0;X 3;2;1;1;0;0;1 5;2;1;1;0;0;X 7;2;1;1;0;0;36 7;2;1;1;0;0;X	Esta parte del programa hace ya el fondo del cuello y posee un ciclo de 36 vueltas definido para el ancho del cuello

Los cuellos diseñados por el programa descrito, son mostrados en las figuras 45 y 46.

Figura 45. Tejido sin desprender

Figura 46. Cuello terminado

17. DATOS DE COMPARACIÓN

Con la máquina a pleno funcionamiento, se realizan comparaciones con otra máquina de las mismas características sin el sistema instalado. Se seleccionan 5 diseños de cuellos diferentes, precargados en el módulo (ver figura 47) y se realiza cierta cantidad de cada uno paralelamente, de acuerdo a la producción establecida por la compañía, para no incurrir en sobre costos innecesarios:

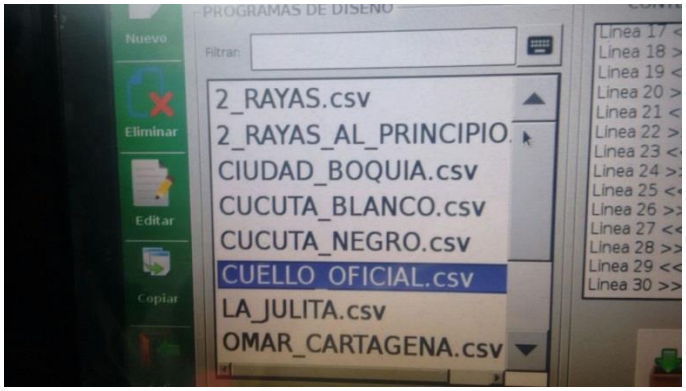

Figura 47. Lista de diseños

Los diseños escogidos para la toma de datos son:

- CUCUTA_NEGRO.csv 1500 unidades
- CUCUTA_BLANCO.csv 1200 unidades
- CUELLO_OFICIAL.csv 920 unidades.
- 2_RAYAS.csv 1100 unidades.
- OMAR_CARTAGENA.csv 2500 unidades

En los tiempos de estas pruebas se obvian las detenciones comunes de la máquina como lo son, por terminación de un hilo, falta de fluido eléctrico, por verificación en las medidas de los cuellos, entre otras que no comprometen el sistema que se está evaluando. En las tablas 4 y 5 se muestran, los datos de comparación entre los dos tipos de máquinas.

MÁQUINA 1: Con el sistema instalado

PROGRAMA	TIEMPO EN EL DISEÑO DEL PROGRAMA	CANTIDAD CUELLOS	DURACIÓN DE LA PRODUCCIÓN	PROBLEMAS PRESENTADOS
CUCUTA_NEGRO	10 MINUTOS	750	38 horas	Ninguno
CUCUTA_BLANCO	5 MINUTOS	600	30 horas	Ninguno
CUELLO_OFICIAL	8 MINUTOS	460	23 Horas	Se presentaron problemas con un nudo que afecto las agujas haciendo que se iniciara el cuello desde el principio. Dicha tarea demoró 1 min cada vez que se presentaba
2_RAYAS	6 MINUTOS	550	26 Horas	Ninguno
OMAR_CARTAGENA	15 minutos	1250	60 Horas	Se presentaron problemas al realizar el cambio de hilaje, el cual afectaba las muestras.

Tabla 4. Datos de los diferentes programas ejecutados en la máquina 1

MÁQUINA 2: Sin el sistema instalado

PROGRAMA	TIEMPO EN EL DISEÑO DEL PROGRAMA	CANTIDAD CUELLOS	DURACIÓN DE LA PRODUCCIÓN	PROBLEMAS PRESENTADOS
CUCUTA_NEGRO	45 MINUTOS	750	43 Horas	Falla en la programación del diseño, se tuvo que corregir el diseño del programa en la cadena.
CUCUTA_BLANCO	48 MINUTOS	600	33 Horas	La máquina tuvo problemas con unas agujas y el operario debía poner a que la máquina iniciara en el corte, tarea que le demora 8 min o más cada vez que se presentaba
CUELLO_OFICIAL	40 MINUTOS	460	23 Horas	Ninguno
2_RAYAS	45 MINUTOS	550	30 Horas	Ninguno
OMAR_CARTAGENA	65 MINUTOS	1250	75 Horas	Falla en la programación del diseño, se tuvo que corregir el diseño del programa en la cadena.

Tabla 5. Datos de los diferentes programas ejecutados en la máquina 2

Nota: Los tiempo son tomados teniendo en cuenta que un cuello demoraba en promedio 2.8 minutos y se daba unos minutos de más por detenciones normales de la máquina.

Para poder crear un nuevo diseño en la **Máquina 2** se debe detener para retirar la cadena y hacer el patrón de diseño, luego colocarla teniendo la precaución de ubicarla bien en el tambor, en la **Máquina 1** se puede ir creando un nuevo diseño mientras está ejecutando otro diferente, además se puede copiar un diseño ya creado y hacer los ajustes necesarios para el nuevo.

Cuando se presentan problemas con el hilo o las agujas que terminen dañando el cuello que actualmente se está haciendo y para no generar desperdicio lo que se hace es volver a iniciar un nuevo cuello, en el caso de la **Máquina 2** se debe detener levantar el sistema de interruptores, mover el tambor hasta la primera posición de la cadena, luego bajar el sistema de interruptores y encender la máquina, para el caso de la **Máquina 1** basta solo con presionar un botón, el sistema solo vuelve al inicio sin necesidad de detenerla.

En las tablas 6, 7, 8, 9 10 y 11 se relacionan los datos comparativos entre ambas máquinas.

PROGRAMA	CUCUTA_NEGRO		
	Diseño del Programa (minutos)	Cantidad	Horas
MÁQUINA 1	10	750	38
MÁQUINA 2	45	750	43
DISMINUCIÓN EN LOS TIEMPOS PRODUCCIÓN	77.77%	11.62%	

Tabla 6. Porcentajes de comparación en tiempos de producción 1

PROGRAMA	CUCUTA_BLANCO		
	Diseño del Programa (minutos)	Cantidad	Horas
MÁQUINA 1	5	600	30
MÁQUINA 2	48	600	33
DISMINUCIÓN EN LOS TIEMPOS PRODUCCIÓN	89.58%	9.09%	

Tabla 7. Porcentajes de comparación en tiempos de producción 2

PROGRAMA	CUELLO_OFICIAL		
	Diseño del Programa (minutos)	Cantidad	Horas
MÁQUINA 1	8	460	23
MÁQUINA 2	40	460	23
DISMINUCIÓN EN LOS TIEMPOS PRODUCCIÓN	80%	0%	

Tabla 8. Porcentajes de comparación en tiempos de producción 3

PROGRAMA	2_RAYAS		
	Diseño del Programa (minutos)	Cantidad	Horas
MÁQUINA 1	6	550	26
MÁQUINA 2	45	550	30
DISMINUCIÓN EN LOS TIEMPOS PRODUCCIÓN	86.66%	13.33%	

Tabla 9. Porcentajes de comparación en tiempos de producción 4

PROGRAMA	OMAR_CARTAGENA.csv		
	Diseño del Programa (minutos)	Cantidad	Horas
MÁQUINA 1	15	1250	60
MÁQUINA 2	75	1250	75
DISMINUCIÓN EN LOS TIEMPOS PRODUCCIÓN	80%	20%	

Tabla 10. Porcentajes de comparación en tiempos de producción 5

	Enviar a Corte	Programar un Diseño	Número de fallas por diseño al día	Número de fallas en la cadena de programación o en el sistema por semana
MÁQUINA 1	2 Seg.	5 – 8 Min aprox.	0	0
MÁQUINA 2	6 Min.	40 Min < aprox.	5 Veces	10 Veces

Tabla 11. Tiempos comparativos entre ambas máquinas para ciertos eventos

18. CONCLUSIONES

El proyecto fue ejecutado a satisfacción a través del desarrollo de los diferentes objetivos específicos, en los cuales se obtuvieron logros y dificultades. Estos ayudaron a la consecución del objetivo general que fue planteado como el diseño e implementación de un sistema automatizado capaz de controlar y cambiar el accionar mecánico por uno electrónico en la programación de máquinas rectilíneas SCOMAR A80 para la fabricación de cuellos tejidos. El propósito de desarrollar una solución integral que permitiera el reemplazo del sistema mecánico de las máquinas rectilíneas, se obtuvo, facilitando la programación de los patrones de diseño por medio de una interfaz gráfica la cual controla un sistema electrónico que remplazo el sistema mecánico de la máquina.

Estudiar el marco de antecedentes, requerimientos y restricciones de los componentes que integran el proyecto.

Logros:

- La obtención de diferentes recursos computacionales, para la creación de las librerías necesarias para la configuración de los elementos que integran el sistema de automatización de la máquina rectilínea.
- El aprendizaje de las diferentes características creadas para la conexión de la tarjeta con diferentes periféricos, corroborando así el correcto funcionamiento del módulo computacional Raspberry Pi y los objetivos de su creación.
- El desarrollo de procesos tecnológicos perfeccionados en la ingeniería, como la evolución de diferentes técnicas para la creación de PCB´s, son aplicados a la automatización en virtud de elevar la calidad y los procesos de la compañía, haciendo que la producción sea mayor.

Dificultades:

- Es poca la información que se logró obtener acerca del modelo de la máquina seleccionada para este proyecto, ya que son modelos de muchos años de fabricación. Para el remplazo de

los elementos mecánicos que se automatizaron, fue necesario el aprendizaje del proceso de tejido, así como el estudio del sistema de configuración que conlleva el uso de los switches electromecánicos, la cadena de programación, el sistema de piñones y demás elementos que componen el sistema mecánico que posee la máquina rectilínea, esto a través del conocimiento empírico de los operarios.

Diseño e implementación del sistema de control electrónico que permita, teniendo en cuenta los requerimientos y restricciones, reemplazar el accionar mecánico en la programación de la máquina por uno electrónico.

Logros:

- El no disponer de piezas mecánicas en la programación de la máquina facilita labores de mantenimiento, producción y manipulación de la misma, además de ahorrar en la compra de elementos que ya no se fabrican sino que son conseguidos como repuestos de máquinas no funcionales.
- La rápida respuesta de los sensores inductivos así como el amplio rango de consumo de corriente que suministran, permite que los componentes electrónicos usados duren largo tiempo.
- Al reemplazar todo el sistema mecánico de programación se reduce en 20CM el ancho que ocupa de espacio la máquina.

Dificultades:

- Los sensores inicialmente utilizados tenían metal en su estructura y hacia masa con la estructura de la máquina, provocando errores en el funcionamiento de los sensores ya que la máquina al detener el motor genera una contracorriente muy alta alterando las señales que el sensor entregaba. Por tal motivo se cambian por sensores aislados.
- Los relés de estado sólidos son una gran solución cuando se requiere reemplazar un accionar mecánico por uno electrónico, ya que dichos relés al no contar con piezas mecánicas en su funcionamiento su respuesta es muy rápida.

- Para no acoplar la tierra de la máquina con el sistema de control con tal de hacerlo inmune al ruido e interferencias se usaron optoacopladores que aislaran la etapa lógica de control de la potencia.
- El soporte que se usó para instalar la pantalla en la máquina se mandó a fabricar sobre acrílico esto con el fin de que la estructura de la máquina no interfiriera sobre la pantalla, al usar un elemento metálico.

Diseño e implementación del sistema de programación e interfaz de usuario para el control y creación de los patrones de diseño de la máquina rectilínea SCOMAR A80.

Logros:

- El desarrollo de una interfaz de usuario de fácil manejo, permite introducir los requerimientos básicos para el control implementado, además muestra en tiempo real lo que pasa durante ejecución del proceso. La memoria de número de prendas realizadas, permite que la detención del ciclo de tejido no afecte el tiempo de configuración de la máquina, ya que no es necesaria la re-configuración mecánica o manual de la misma por parte del operario, para la terminación del tejido.
- La facilidad que ofrece guardar los programas de diseño ahorra tiempo en la programación y permite reutilizar programas para el diseño de nuevos tejidos.
- El desarrollar la interfaz de usuario y la lógica de control sobre el software Qt creator haciendo uso de la librería Qt y demás, da al producto final un aspecto amigable a la vista y permite organizar muy bien los elementos de tal manera que sea fácil de usar por el usuario final, así como dotar al programa de múltiples procesos y opciones que mejoren el rendimiento y la ejecución de varias tareas al mismo tiempo.

Dificultades:

- La integración de los programas de desarrollo para ejecutar el programa sobre la Raspberry nos dio algo de dificultad puesto que se requieren de ciertas librerías y cabeceras, además de software de terceros para que se permita la compilación sin problemas.

- Para que la librería WiringPi funcione correctamente se debe ejecutar el programa con privilegios “sudo” ya que para el acceso al puerto GPIO el sistema operativo Debian sobre el cual corre la Raspberry lo tiene restringido sólo al súper usuario.

Integración de la Raspberry Pi con el sistema de programación y el sistema de control, puesta a punto de la máquina y pruebas de funcionamiento.

Logros:

- La reducción estimada del 75% del tiempo del ciclo de programación de diseño, se da gracias a la eliminación del 100% de los problemas que se presentaban en esta etapa de configuración de los eslabones y las cadenas para cada programa de las prendas con el sistema mecánico. Esto representa un incremento en la eficiencia de la producción de la empresa Tejidos del Risaralda LTDA.
- La máquina logro acumular un ciclo de trabajo de 312 horas durante la etapa de pruebas, sin presentar ningún tipo de fallas en el sistema.

Dificultades:

- El ruido eléctrico que genera el motor afecta significativamente las señales lógicas que provienen de la Raspberry ya que son señales de muy bajo voltaje 3.3V, se debió aislar el cableado con un apantallamiento para que la incidencia del ruido no afecte las señales.

19. BIBLIOGRAFIA

[1] BONILLA O., E. Y MOLANO APONTE, L.: "LA DINÁMICA DE LA PRODUCTIVIDAD EN LA INDUSTRIA TEXTIL DE COLOMBIA 2000- 2010", en Observatorio de la Economía Latinoamericana, Nº 199, 2014. Disponible en <http://www.eumed.net/cursecon/ecolat/co/14/industria-textil.html> [Consulta Mayo de 2015]

[2] TEJIDO DE PUNTO. Disponible en, <http://repositorio.utn.edu.ec/bitstream/123456789/1964/2/Resumen%20T%C3%A9cnico.pdf> [Consulta Septiembre 2015]

[3] GÓMEZ P. JORGE I.: (Marzo de 2011) "DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA AUTOMATIZADO PARA UNA MÁQUINA TEXTIL RECTILÍNEA PARA LA MICROEMPRESA TEXTIL -TECMATEJ -" Trabajo de Grado, en Escuela Politécnico Nacional, Quito – Ecuador.

[4] VARGAS JAIRO, OCHOA L. HECTOR G. (2012) "DISEÑO Y CONSTRUCCIÓN DE UN SISTEMA DE AVANCE Y CORTE PARA LA AUTOMATIZACIÓN DEL PROCESO DE MANUFACTURA DE PASADORES PARA LA CONFECCIÓN DE ROPA APLICADO A LA MÁQUINA RECUBRIDORA KANSAI". Trabajo de grado, en Universidad De San Buenaventura, Bogotá, Colombia. [Consulta Septiembre 2015]

[5] INSTALLING OPERATING SYSTEM IMAGES USING WINDOWS, disponible en < https://www.raspberrypi.org/documentation/installation/installing-images/windows.md> [consulta Noviembre de 2015]

[6] NOOBS SETUP disponible en <https://www.raspberrypi.org/help/noobs-setup/> [consulta Noviembre de 2015]

[7] ALFONSO GOMEZ A. (Marzo de 2013). El Protoboard. MODMEX PC, Publicación Trimestral, Volumen 1, Pág. 16.

[8] GONZÁLES GALLEGO S. (1990) "LA ERGONOMÍA Y EL ORDENADOR" Prologo, Barcelona, España. Editorial Marcombo. S.

[9] NAGEL LAWRENCE W. PEDERSON DANIEL O. (Abril 1973) SPICE (Simulation Program with Integrated Circuits Emphasis), 16th Midwest Symposium, Circuit Theory, Waterloo, Canada.

[10] TUINENGA P.W. (1992). “A Guide to Circuit Simulation and Analysis Using PSpice”. New Jersey. EE.UU. Editorial Prentice Hall.

[11] JOHN KARL-HEINZ. TIEGELKAMP MICHAEL. (2001) “IEC 61131-3: PROGRAMMING INDUSTRIAL AUTOMATION SYSTEMS: Concepts and Programing Languages, Requirements for Programming Systems, Aids to Decision-Making Tools”. New York. EE.UU. Editorial Springer. [Consulta Septiembre 2015]

[12] CadSoft Computer Newark Corporation. “What is EAGLE”, Disponible en <http://www.cadsoftusa.com/> [consulta en Noviembre de 2015]

[13] MORENO C, ALEJANDRA, BLOG: “TIPOS DE TEJIDO, TEXTILES Y TIPOS DE TEJIDOS”, Disponible en <http://textilesytiposdetejidos.blogspot.com/2008/02/textiles-y-tipos-de-tejidos.html> [consulta Junio de 2015]

[14] BARRERA T. FRANCISCO A. (1995), “TÉCNOLOGIA DEL TEJIDO DE PUNTO POR TRAMA A DOS CARAS”, México DF. – México. Editorial, Universidad iberoamericana.

[15] DOUTEL, FERNANDO, BLOG: “CONOCE A LA PLACA QUE QUIERE REVOLUCIONAR TU MUNDO DIGITAL: RASPBERRY PI A FONDO”, Disponible en <http://www.xataka.com/componentes/conoce-a-la-placa-que-quiere-revolucionar-tu-mundo-digital-raspberry-pi-a-fondo> [Consulta Junio de 2015]

[16] PASCUAL, JUAN A. BLOG: “RASPBERRY PI: ¿QUÉ MODELO ME COMPRO?”, Disponible en <http://computerhoy.com/noticias/hardware/raspberry-pi-que-modelo-me-compro-23811> [Consulta Junio de 2015]

[17] GUÍA RÁPIDA. RELÉS DE ESTADO SÓLIDO SSR’s. Disponible en, <http://www.reitec.es> [consulta en Agosto de 2015]

[18] MARCOS A. JORGE, FERNANDEZ S. CELSO, ARNESTO Q. JOSÉ I. (Segunda Edición 2009): “AUTÓMATAS PROGRAMABLES Y SISTEMAS DE AUTOMATIZACIÓN”. Barcelona, España. Editorial Marcombo. S.A [Consulta Septiembre 2015]

[19] OROZCO G. ÁLVARO A., GUARNIZO L. CRISTIAN HOLGUÍN L. MAURICIO, (2008) “AUTOMATISMOS INDUSTRIALES”. Pereira, Colombia. Editorial Universidad Tecnológica de Pereira. [Consulta Septiembre 2015]

ANEXOS

ANEXO 1: MODELOS DE LA RASPBERRY, DESCRIPCIÓN DE COMPONENTES Y CONFIGURACIÓN E INSTALACIÓN DEL SISTEMA OPERATIVO.

MODELOS

Desde la creación en 2008 de la primera versión de la RASPBERY PI, se han lanzado 5 versiones de las cuales 4 han salido al mercado:

RASPBERRY PI A: Es la versión más básica de este micro-computador, tiene soporte para tarjetas SD, posee un puerto USB, contiene 8 puertos GPIO.

RASPBERRY PI B: Fue el primer modelo comercial de la marca, cuenta con el doble de memoria RAM (512 MB) que su modelo anterior, además de esto suma un puerto USB más y adhiere una conexión ETHERNET 10 /100.

RASPBERRY PI A+: Esta ofrece la misma estructura de hardware que el modelo A, pero aumenta los puertos GPIO de 8 a 17, incluye mejores en su sistema de audio, tiene soporte para tarjetas micro SD.

RASPBERRY PI B+: Esta versión cuenta con 4 puertos USB, tarjetas micro SD, 17 puertos GPIO y un consumo menor a 3W.

RASPBERRY PI 2 B: Esta nueva versión tiene como grandes cambios, el remplazo del chip BCM2835 al nuevo BCM2836, que tiene la misma arquitectura del anterior, pero su nueva CPU es una ARM CORTX-A7 de cuatro núcleos a 900 MHz, y dobla su memoria RAM a 1 GB.

DESCRIPCIÓN GENERAL DE LOS COMPONENTES DE LA RASPBERRY PI

FUENTE DE PODER

La fuente de poder de 5V cuenta con una entrada micro USB y un fusible de 2A, también cuenta con un transistor de efecto de campo, P-Channel MOSFET (DMG2305UX) para protección al cambio de polaridad.

PUERTOS GPIO

Los puertos GPIO, son un sistema de propósito general de entradas y salidas, GPIO (General Purpose Input/Output). Los puertos incluidos en la RASPBERRY PI, son del tipo “unbuffered”,

esto quiere decir que no tienen protección contra altas magnitudes. Los 40 pines disponibles en el modelo 2B cumplen diferentes funciones, como se puede observar en la Tabla 13.

	FUNCIÓN	PIN	FUNCIÓN	PIN	FUNCIÓN	PIN	FUNCIÓN
1	3.3 V	11	GPIO 17 *(GPIO_GEN0)*	21	GPIO 9 *(SPI_MIS0)*	31	GPIO 6
2	5V	12	GPIO 18 *(GPIO_GEN1)*	22	GPIO 25 *(GPIO_GEN6)*	32	GPIO 12
3	GPIO 2 *(SDA1)*	13	GPIO 27 *(GPIO_GEN2)*	23	GPIO 11 *(SPI_SCLK)*	33	GPIO 13
4	5V	14	GND	24	GPIO 8 *(SPI_CE0_N)*	34	GND
5	GPIO 3 *(SCL1)*	15	GPIO 22 *(GPIO_GEN3)*	25	GND	35	GPIO 19
6	GND	16	GPIO 23 *(GPIO_GEN4)*	26	GPIO 7 *(GPIO_CE1_N)*	36	GPIO 16
7	GPIO 4 *(GPIO_CCLK)*	17	3.3V	27	ID_SD	37	GPIO 26
8	GPIO 14 *(TXD0)*	18	GPIO 24 *(GPIO_GEN5)*	28	ID_SC	38	GPIO 20
9	GND	19	GPIO 10 *(SPI_MOS1)*	29	GPIO 5	39	GND
10	GPIO 15 *(RXD0)*	20	GND	30	GND	40	GPIO 21

Tabla 12. Distribución de pines

Pines de alimentación: Pines de salida de 3, 3V y 5V, limitados a 50 mA y Pines de conexión a tierra (GND), estos pueden aportar alimentación a los circuitos que se deseen conectar a los puertos GPIO.

Pines ID_SD & ID_SC: Estos pines están reservados para EEPROM a través de la interfaz I2C.

Pines especiales: Son pines especiales destinados a interfaz UART, dentro de ellos se puede encontrar pines TXD y RXD especiales para comunicación en serie. Los pines sobrantes, es decir los que no están diseñados para una función especial, son conexiones programables para el desarrollo de los proyectos.

PUERTOS USB & ETHERNET

El módulo computacional cuenta con 4 puertos USB 2.0 de alta velocidad, además de un puerto Ethernet (10/100).

PUERTOS DE AUDIO Y VIDEO

El módulo RASPBERRY PI B+ cuenta con dos puertos de salida de audio y de video:

Puerto TRRS: es un Jack de 4 polos, para conexión de Plug de 3.5mm, funciona como salida de audio o como salida de Audio/Video.
Puerto HDMI: El puerto HDMI (High Definition Multimedia Interface) es un puerto especial de 19 o 29 terminales, capaz de transmitir de manera simultánea videos de alta definición, así como varios canales de audio y otros datos de apoyo.

RANURA PARA MICRO TARJETA (SD CARD)

Para la operación y funcionamiento, la RASPBERRY PI B+ / PI 2 B requiere una tarjeta flash en la cual almacenar el sistema operativo y los archivos. Esta tarjeta es del tipo micro SD

CONFIGURACIÓN

En el momento de la adquisición de un módulo computacional RASPBERRY PI, es necesario realizar algunas configuraciones para su uso. Estas configuraciones incluyen la descarga del sistema operativo y la activación de los diferentes puertos para su uso.

PREPARACIÓN DE LA TARJETA SD

El primer paso para operar el MÓDULO es la descarga del sistema operativo, para ello se debe contar con una tarjeta micro SD de mínimo 8GB. Esta memoria debe ser formateada con programas tales como SD Association's Formating Tool, SD Formater, Flash Format, o algún otro que nos asegure la limpieza completa de la misma, otra forma de realizar la liberación total de la memoria micro SD es siendo inicializada con un formato FAT32, con esto el sistema asegura que la memoria este totalmente vacía y así proceder a la instalación del sistema operativo.

INSTALACIÓN DEL SISTEMA OPERATIVO

El sitio oficial de RASPBERRY PI ofrece diferentes opciones de sistemas operativos para el funcionamiento del módulo. Entre ellas Raspberry ORG, recomienda como primera medida utilizar **RASPBIAN,** que es el sistema operativo oficialmente soportado para Raspberry Pi, basado en Debian adaptado a las capacidades de la placa computarizada. La segunda opción sugerida por Raspberry ORG, es **NOOBS** (New Out Of Box Software), este es un instalador de sistema operativo, el más recomendado en caso de que sea la primera vez que se opera un módulo computacional Raspberry PI.

Raspberry ORG, también ofrece la posibilidad de descargar sistemas operativos de terceros, como lo son, **UBUNTO MATE, SNAPPY UBUNTU CORE, WINDOWS 10 IOT CORE, OSMC, OPENLEC, PINET** y **RISC OS.** Algunos de estos sistemas operativos están especialmente diseñados para crear centros de entretenimiento basados en contenido multimedia. [5]

Pasos para la instalación de RASPBIAN

La organización Raspberry ofrece 3 opciones de imágenes de RASPBIAN, cada una de ellas basadas en tres tipos diferentes de Debian. La escogencia de la versión de RASPBIAN que se desee descargar, queda sujeto a las necesidades de cada usuario y al tipo de MÓDULO con el que se cuente.

Pasos:

- Descargar el archivo de RASPBIAN deseado.
- Realizar la preparación de la tarjeta Micro SD
- Descargar la aplicación Win32 Disk Imager (Esta aplicación está diseñada para escribir una imagen de disco sin procesar) y ejecutarla.
- Extraer el archivo de imagen del documento ZIP descargado para la versión de RASPBIAN que se desee instalar
- Con Win32 Disk Imager seleccionar el archivo de imagen extraído anteriormente.
- En la opción “Device” de la aplicación seleccionar la tarjeta Micro SD.

- Click en la opción “write” de Win32 Disk Imager y esperar a que se complete la escritura en la tarjeta.
- Con los archivos copiados en su totalidad en la tarjeta Micro SD, se puede proceder a retirar la tarjeta de la ranura del ordenador.

Guía de inicio Raspberry PI

Después de tener preparadas las memorias Micro SD, se requieren de algunos elementos para realizar la preparación del sistema operativo en el módulo computacional. [6]

Elementos:

- Teclado de conexión USB.
- Mouse de conexión USB.
- Monitor con conexión HDMI/DVI
- Cables de conexión HDMI/DVI
- Fuente de poder 5V - USB.

Pasos para inicio con NOOBS

- Realizar la conexión del teclado, mouse y el monitor.
- Conectar la fuente de poder de 5V al módulo computacional.
- Al realizar el paso anterior, inmediatamente se encenderá la Raspberry Pi, algunos adaptadores de poder incluyen la opción de un interruptor de encendido.
- En el monitor aparecerá una lista de diferentes sistemas operativos que pueden ser descargados e instalados, si la versión de NOOBS descargada fue la completa, no es necesario descargar el RASPBIAN ya que viene incluido.
- Seleccionar el sistema operativo y hacer click en “Install”
- RASPBIAN ejecutara el proceso de instalación (Puede tomar algunos minutos)
- Cuando el proceso de instalación termine, el menú de configuración de Raspberry PI aparecerá en pantalla automáticamente.

Dentro del menú de configuración se puede acceder al cambio de región, así como también al idioma, entre otros.

RASPBIAN genera un usuario y contraseña por defecto.
Usuario: **pi**
Contraseña: **Raspberry**

Después de cambiar la configuración en el menú como se desee, es necesario desplazarse hasta la opción de selección "Finish" y seleccionarla. Para cargar la interfaz gráfica es necesario teclear la palabra **"startx"** unos cuantos segundos después, se podrá navegar dentro del escritorio del sistema operativo, como se observa en la figura 48.

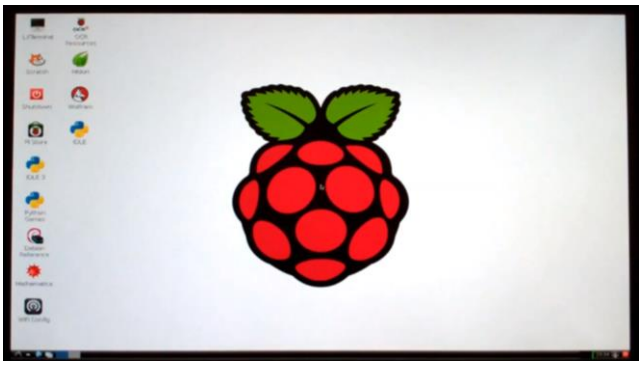

Figura 48. Sistema Operativo RASPBIAN

ANEXO 2: DISEÑO PCB CON EAGLE

CLASIFICACIÓN DE LOS PCB

- **Material de confección:** Generalmente están compuestas de fibra de vidrio, sobre estas se dibujan las pistas de material conductor, comúnmente se realizan en cobre. los PCB pueden construirse de diferentes sustratos, dependiendo de la funcionalidad del mismo:
  - En los circuitos utilizados en electrónica de bajo consumo, se suele recurrir a papeles impregnados de resinas fenólicas.
  - Los circuitos utilizados en electrónica de alto consumo e industrial, se fabrican en su mayoría de fibra de vidrio impregnado con resina epóxica resistente a las llamas.
  - Circuitos de radio frecuencia, generalmente se usan plásticos con constante dieléctrica baja, en su mayoría poliestirenos y poliamidas.
  - Para los circuitos que son usados en naves espaciales, se utilizan núcleos de cobre o aluminio, los suficientemente gruesos para lograr disipar el calor de los componentes electrónicos, ya que en el vacío o en gravedad cero, estos son incapaces de generar enfriamiento por convección.

- **Tipo de montaje:** Estas están determinadas por el tipo de encapsulado del componente que desea instalarse:
  - THT, Tecnología de agujeros pasantes, denominados también como componentes de inserción, ya que se requieren orificios en el PCB, para su montaje.
  - SMT, Tecnología de montaje superficial, este tipo de encapsulado no requiere de agujeros para su montaje en las PCB, no tienen alambres de conexión, pero pueden venir con diferentes tipos de pines como lo son, los pines metalizados, pines en forma de J, pin de gaviota, pin forma de cuña, pin doblado, o pin en forma de I.

- **Número de capas:** las PBC´s más sencillas tienen solo una capa de pistas, dispuestas sobre una de sus caras, los componentes de estas generalmente son del tipo THT y suelen ser circuitos de bajo coste y baja complejidad. Las tarjetas bicapa o doble cara, son utilizadas en circuitos de mediana complejidad debido a que para ser montados en placas de una sola capa requerirían de un gran tamaño o de un gran número de puentes para lograr las conexiones entre los elementos.

CAD – DISEÑO ASISTIDO POR COMPUTADORA

El diseño asistido por computadora o CAD (Computers Aided Design), se describe como la utilización de recursos computacionales para la creación de representaciones gráficas de objetos físicos en 2D o 3D. Los CAD se han convertido en una herramienta necesaria para la creación de los PCB, ya que gracias a ellas se han podido elaborar circuitos electrónicos de menor costo de producción, simular el funcionamiento sin la necesidad de ser armado.

Las primeras herramientas de diseño de circuitos electrónicos fueron producidas alrededor de los años 60 por la compañía IBM, la cual desarrollo un software para el análisis de circuitos electrónicos, llamado ECAP (Electric Circuit Analysis Program), este programa fue el punto de partida para que a mediados de los años 70, un grupo de investigación de la Universidad de Berkeley en California, desarrollara SPICE (Simulation Program with Integrated Circuits Emphasis) el cual permitía el análisis de circuitos analógicos sin necesidad de su ensamble, esto en grandes ordenadores y workstations, donde en un fichero de texto se describía el circuito y el tipo de análisis requerido, igualmente se permitía dejar parámetros sin especificar para lograr que tomaran valores por defecto, SPICE realizaba la lectura del fichero y comprobaba que no existieran conexiones y sintaxis declaradas para realizar la simulación del funcionamiento.[8-11].

Con el avance en los sistemas de cómputo en los años 80 y 90, en donde la comercialización a computadoras personales de bajo costo y alto rendimiento tuvo un crecimiento inesperado, sumando la evolución de los sistemas microcontrolados, los microchips y los componentes electrónicos en general, los programas de diseño de circuito han evolucionado al punto de convertirse en herramientas de trabajo cotidiano, creando herramientas para sistemas electrónicos de potencia, sistemas de alta frecuencia, sistemas electromagnéticos, etc.

Definiciones básicas para el diseño de circuitos impresos (PCB)

Para entender el diseño de circuitos impresos, se deben de conocer las diferentes definiciones conceptos y terminologías utilizadas en los procesos que aplican hacia ellos.

- **Esquemático:** el diagrama esquemático es la visualización primaria de la conexión de los elementos del circuito que se desea realizar.

- **Layout:** se refiere al diseño de la tarjeta, específicamente al trazado de las líneas de la misma. El layout hace parte del Boardfile, que es el archivo general donde se realiza el diseño físico de la tarjeta.
- **Pads:** es una superficie de cobre que permite la fijación de un componente a la tarjeta a través del proceso de soldado. Los pads suelen ser rectangulares para elementos de montaje superficial (smd) es decir, soldar el componente por el mismo lado de la placa en donde se ubica, y redondos para los componentes through-hole pensados para introducir el pin del componente para luego soldarla por el lado opuesto al cual se introdujo.
- **Soldermask:** la máscara de soldado es una capa física en barniz que se utiliza para evitar la oxidación del cobre del PCB, esta capa cubre las caras de la tarjeta a excepción de los pads.
- **SMD:** Los SMD (Surface Mount Devices), dispositivos de montaje superficial son elementos que no requieren agujeros para ser instalados en el circuito impreso. No se cuenta con alambres de conexión, por el contrario el propio encapsulado posee sus extremos metalizados o terminales cortos y rígidos de diversas formas
- **Track:** es un segmento de línea de conexión generalmente de cobre, son requeridos para la conexión entre pads, tienen distinto ancho, esto depende de las corrientes que fluyen a través de ellos.

DISEÑO DE CIRCUITOS IMPRESOS CON CAD

En la actualidad existen una gran cantidad de programas a los cuales se puede acceder para realizar la simulación de los circuitos, ya sean analógicos o digitales, dibujar esquemas o diseñar placas de circuito impreso.

Una de estas herramientas de diseño asistido por computadora es EAGLE (Easily Applicable Graphical Layout Editor), es un software para la elaboración de circuitos impresos de baja/mediana complejidad, diseñado por la empresa CadSoft Computer Newark Corporation a principios de la década de los 90. El programa presenta una serie de características generales, así como el MÓDULO esquemático, el editor de capas y el módulo de autoruter. [12]

Características generales:

- Editor esquemático
  - Hasta 999 hojas por esquema.
  - Vista previa de los iconos en las hojas.
  - Referencias cruzadas para redes.
  - Generación automática de contactos para referencias cruzadas.
  - Remplazo de las funciones de las piezas sin pérdida de coherencia entre el esquema y el diseño (layout)
  - Generación automática de las conexiones de alimentación.
  - Generación automática de la placa.
  - Verificación de conexiones eléctricas entre los esquemas eléctricos y de líneas de conexión

- Editor de diseño
  - Área máxima de dibujo de 4 x 4 m (aprox. 150 x 150 pulgadas)
  - Soporte completo en SMD
  - Rotación de objetos en pasos de ángulos arbitrarios de 0.1°
  - Bloqueo de componentes después de ser movidos.
  - Locación de texto en cualquier orientación.
  - Cálculo dinámico de las rutas de señal.
  - Función de Pads magnéticos.
  - Extracción de las esquinas de los Tracks, en cualquier radio.
  - Comprobación de las reglas de diseño para PBC (p. e. traslapas, medias de pistas o líneas de conexión).
  - Soporte para gran variedad de encapsulados.

- Editor de autoruta
  - Cambio del modo manual al automático en cualquier instante.
  - Rejilla de trazado desde 0.02 mm
  - Uso de reglas de diseño de líneas de conexión
  - Sin restricciones de posicionamiento.

- Hasta 16 capas de señales (con direcciones preferidas definidas por el usuario)

Diseño de PCB con EAGLE.

Para realizar el diseño de un PCB, primero se debe hacer el plano esquemático, ya que se deben verificar las conexiones de los componentes del circuito, para así proceder a realizar el diseño de la placa de circuito impreso.

- **Pasos para la creación del plano esquemático**

Lo primero que se debe realizar es la creación de un nuevo proyecto, ya que dentro del mismo se realizara la creación de varios tipos de tarjeta y de librerías como se observa en la figura 49

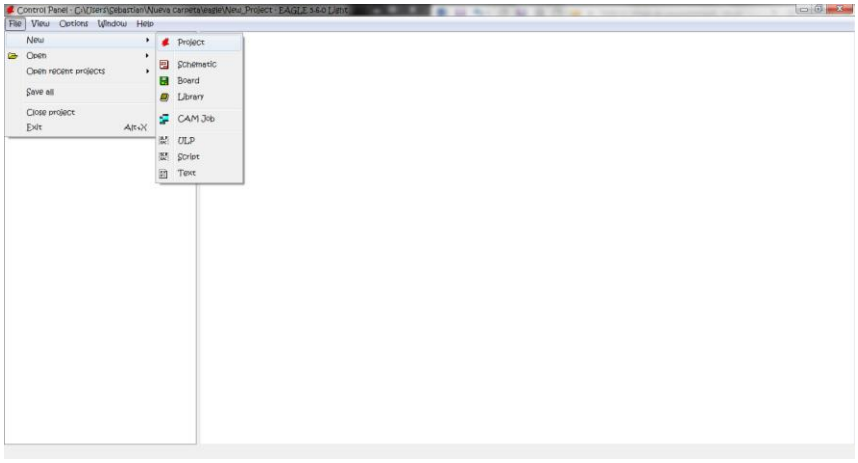

Figura 49. Creación de nuevo proyecto en EAGLE

Ahora que se realizó la creación de proyecto, se pasa a crear un nuevo plano esquemático teniendo seleccionada la carpeta establecida para que el plano quede creado dentro de ella como se muestra en la figura 50:

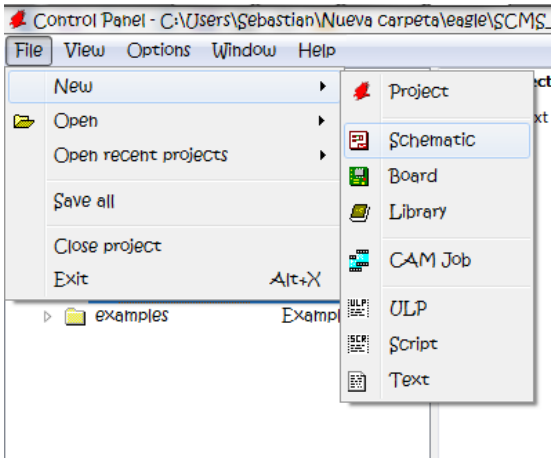

Figura 50. Creación de plano esquemático

En el panel de navegación ubicado al lado izquierdo de la ventana del plano esquemático, se pueden encontrar una lista de botones con los cuales se podrá editar el plano, estos se muestran en la figura 51. Haciendo click sobre la tecla de selección "add" como se ve en la figura 52 se encuentra una lista detallada con todas las librerías de componentes presentes en EAGLE, esto se ve en la figura 53.

Figura 51. Panel de selección

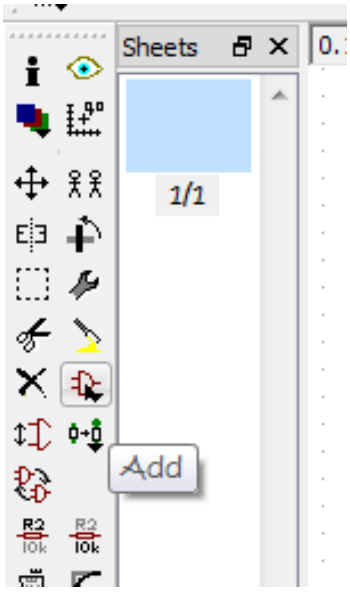

Figura 52. Tecla de selección "Add"

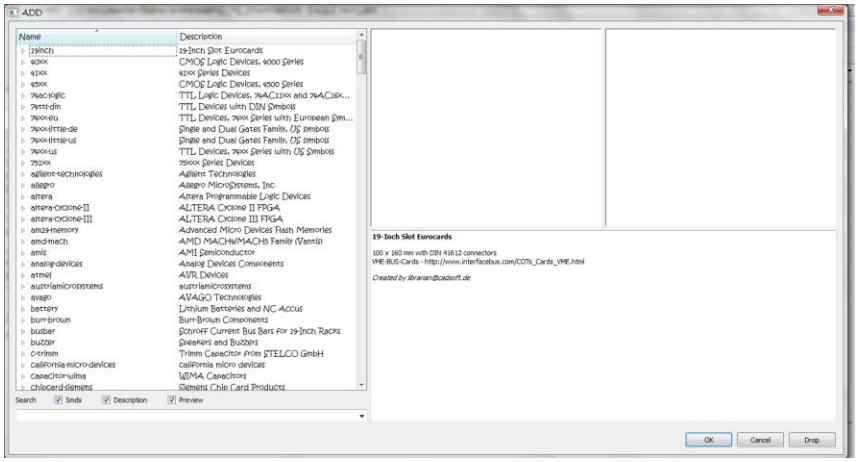

Figura 53. Ventana de selección de elementos

Al costado izquierdo de la figura 53 se puede observar la lista de las librerías presentes en EAGLE, de donde se seleccionan los componentes necesarios para realizar el circuito deseado.
El plano a realizar como ejemplo en este apartado, es el correspondiente al circuito de acondicionamiento de sensores inductivos, el cual se explica en el ítem 13.1.3 del documento principal.

La lista de elementos necesarios para realizar el circuito es descrita en la tabla 13:

ELEMENTO	LIBRERÍA	DESCRIPCIÓN EAGLE	DENOMINACIÓN
Optoacoplador	Optocoupler	General Purpose 6-Pin Phototransistor Optocouplers	OK1 – OK2 4N25M
Diodos	Diode	Z DIODE 3.3 V, 1W, 5 percent	D1 – D2 1N4728
Resistencias	Resistor	RESISTOR, European symbol	R1 – R2 – R3 – R4 R5 – R6 – R7 – R8
Terminaciones	Con – molex	Molex conectors .100" (2.54mm) Center Header - 3 Pin .100" (2.54mm) Center Header - 4 Pin	22 – 23 – 2031 22 – 23 – 2041
Terminaciones	Jumper	Jumper , JP4E	JP1

Tabla 13 Elementos de circuito

Con la Tecla de selección “add” se adicionan cada uno de los componentes de la tabla anterior, en la figura 54 se observa la selección del elemento opto acoplador 4N25M y en la figura 55 se ve el componente en el plano esquemático junto con los demás elementos necesarios para la creación del circuito.

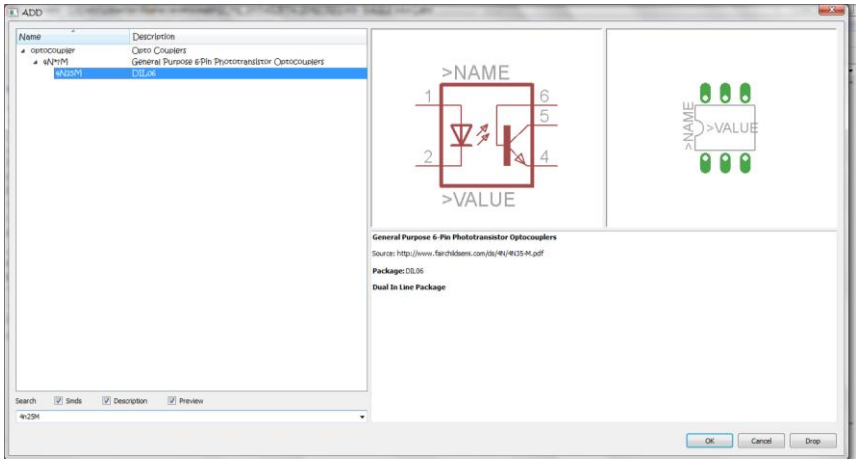

Figura 54. Selección de un elemento

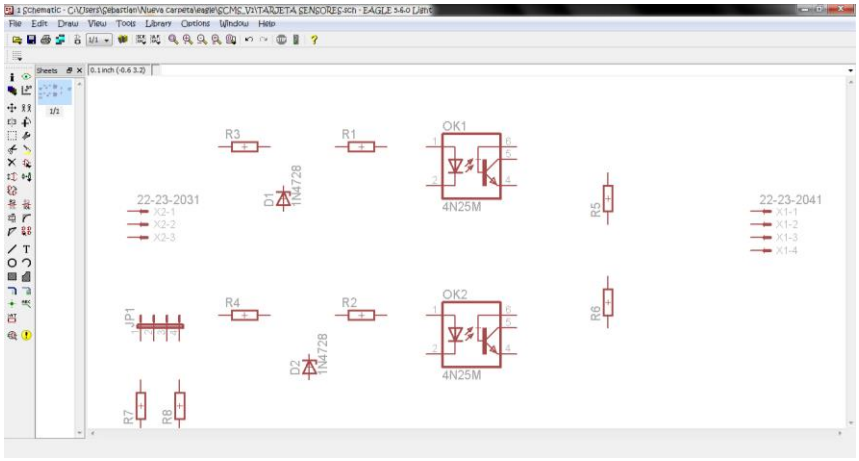

Figura 55. Elementos en plano esquemático

En la figura 56 se puede observar el circuito con todas sus conexiones:

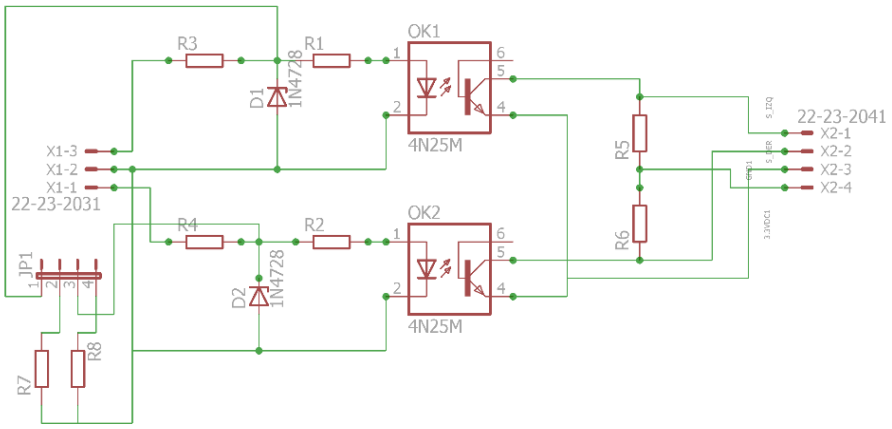

Figura 56. Conexión del circuito

Después de tener el diseño del esquema del circuito, se puede pasar a la creación de la placa, para esto es necesario hacer click en el botón board, ubicado en la parte superior del plano esquemático. Una nueva ventana aparecerá, esto se denomina un archivo “brd”, se verá la conexión de los elementos con hilos delgados de color dorado, pero estarán por fuera de un recuadro que se encuentra ubicado en la parte derecha del archivo, como se muestra en la figura 57:

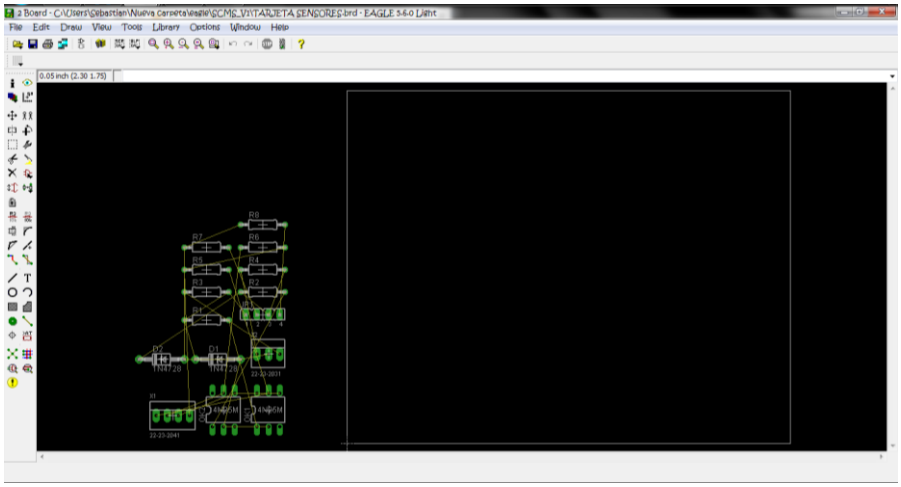

Figura 57. Ventana del plano BRD

Ahora es necesario ubicar cada uno de los elementos dentro de la placa que aparece en la parte derecha, con la tecla “move” se puede realizar el movimiento de los elementos hacia la placa.

Una vez se tengan ubicados los componentes dentro de la placa, se pueden ubicar de la mejor manera, para reducir el tamaño de la tarjeta, o para tener la placa con la forma deseada como se ve en la figura 58.

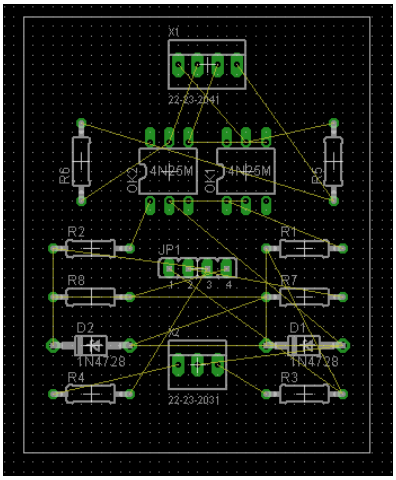

Figura 58. Ubicación de componentes

Una vez se obtenga la figura deseada, EAGLE presenta la opción de crear la autoruta, en donde diseña el trazado de las pistas de la placa automáticamente, esto se realiza con el botón "auto" mostrado en la figura 59

Figura 59. Tecla de selección "Auto"

Después de que el autoruteador entregue las pistas trazadas, se pueden realizar algunos ajustes en la placa, como son agregar texto para dar indicaciones en la placa y la posición de los agujeros de fijado en la base, el resultado final se puede observar en la figura 60.

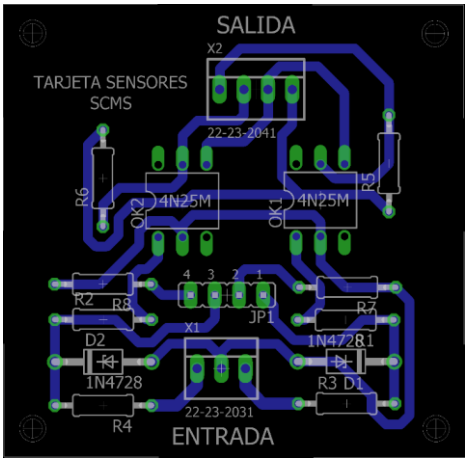

Figura 60. Placa final

Cuando no sea posible la traza de todas las pistas de la placa con la opción del autoruteo, EAGLE genera la posibilidad de realizarlas individualmente mediante la opción “route” como se ve en la figura 61

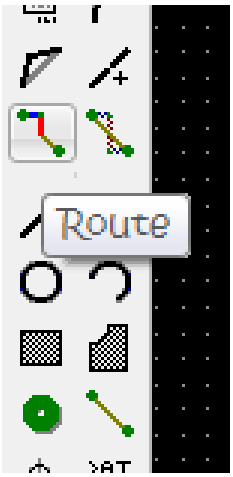

Figura 61. Tecla de selección "Route"

ANEXO 3: DESCRIPCIÓN DE COMPONENTES DE LA LIBRERÍA QT, INSTALACIÓN E INTEGRACIÓN CON LA RASPBERRY.

COMPONENTES DE LA LIBRERIA QT

El modelo de objetos Qt.

Qt se basa en el modelo de objetos Qt. esta arquitectura es la que hace que Qt sea tan potente y fácil de usar. Los dos pilares de Qt son la clase QObject (objeto Qt) y la herramienta MOC (compilador de metaobjetos).
La programación mediante el modelo de objetos Qt se fundamente en derivar las clases nuevas que se creen de objetos QObject. Al hacer esto, se heredan una serie de propiedades que caracterizan a Qt (y lo hacen especial frente al C++ estándar):

- ❖ Gestión simple de la memoria.
- ❖ Signals y slots.
- ❖ Propiedades.
- ❖ Auto-conocimiento.

No hay que olvidar, sin embargo, que Qt no deja de ser C++ estándar con macros, por lo que cualquier aplicación "programada en Qt", puede tener código en C++ estándar o incluso en C.

Gestión simple de la memoria

Cuando se crea una instancia de una clase derivada de un QObject es posible pasarle al constructor un puntero al objeto padre. Esta es la base de la gestión simple de memoria.
Al borrar un objeto padre, se borran todos sus objetos hijos asociados. Esto quiere decir que una clase derivada de QObject puede crear instancias de objetos "hijos-de-QObject" pasándole el puntero this como padre sin preocuparse de la destrucción de estos hijos.
Esto es una gran ventaja frente al C++ estándar, ya que en C++ estándar cuando se crea un objeto siempre hay que tener cuidado de destruir los objetos uno a uno y de liberar memoria con la sentencia delete.

Figura 62. Ejemplo de QWidget padre con hijos

Un ejemplo de aplicación podría ser crear un QWidget ventana (objeto padre) con dos botones QPushButton y una línea de texto editable QLineEdit (hijos de esta ventana). Esto se puede ver en la Figura 62. QWidget es una clase derivada de QObject que permite representar objetos gráficos como ventanas, botones, etc. Por tanto, a su vez, QPushButton y QLineEdit derivan de QWidget.

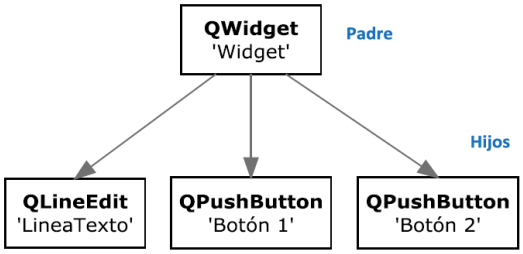

Figura 63. Jerarquía de memoria del QWidget de la figura 62

Al destruir la ventana se destruirían los hijos liberando la memoria correspondiente de manera transparente para el programador. La jerarquía de memoria se puede ver en la Figura 63. Cabe destacar que de cada QObject también podrían crearse objetos hijos, aumentando el árbol de memoria. Estos objetos "nietos" del padre original se eliminarían al eliminarse su objeto padre.

Signals y slots

Las signals y los slots (señales y ranuras, en castellano) son lo que hace que los diferentes componentes de Qt sean tan reutilizables.

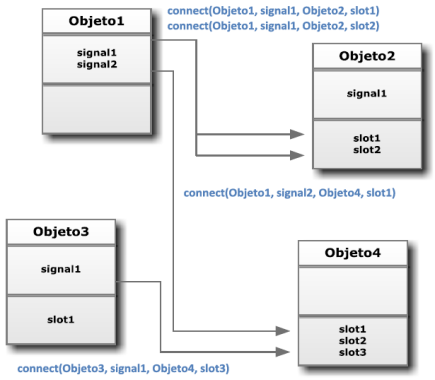

Figura 64. Conexiones entre signals y slots de diferentes objetos

Proporcionan un mecanismo a través del cual es posible exponer interfaces que pueden ser interconectadas libremente (figura 64). Para realizar la interconexión se usa el método "connect". Un ejemplo de una signal podría ser pulsar un botón QPushButton y la signal que se emite cuando se pulsa, clicked() y un slot podría ser por ejemplo, escribir un texto en un QLineEdit.
La ventaja principal de las signals y de los slots es que el QObject que envía la signal no tiene que saber nada del QObject receptor. Este tipo de programación se llama en inglés "loose coupling" y permite integrar muchos componentes en el programa de manera sencilla y minimizando la probabilidad de fallo.

Para poder usar las signals y los slots cada clase tiene que declararse en un fichero de cabecera y la implementación se sitúa en un archivo ".cpp" por separado. Este archivo de cabecera se pasa entonces a través de una herramienta conocida como el MOC (compilador de metaobjetos). El MOC produce un ".cpp" que contiene el código que permite que funcionen las signals y los slots y otras muchas características de Qt. Para realizar este proceso de compilación que puede parecer complejo, se utiliza la herramienta de qmake de Qt que lo realiza de manera automática. De estas herramientas se hablará más adelante. En la Figura 65 se puede ver el flujo desde que se crea una clase derivada de QObject hasta que se compila:

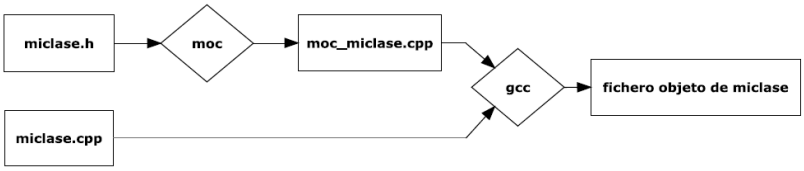

Figura 65. El flujo del MOC

Como ya se mencionó anteriormente, una aplicación Qt sigue siendo C++ al 100%. Por tanto, las signals y los slots son simplemente palabras reservadas o keywords que reemplaza el preprocesador convirtiéndolas en código C++ real.

Los slots se implementan como cualquier método miembro de la clase mientras que las signals las implementa el MOC. Cada objeto tiene una lista de sus conexiones (qué slots se activan con qué signals) y sus slots (cuáles se usan para construir la tabla de conexiones en el método connect). Las declaraciones de estas tablas están escondidas en la macro Q_OBJECT. Esto se puede ver en el código de la figura 66.

```
class MiObjeto : public QObject
{
  Q_OBJECT
public:

  MiObjeto( QObject *parent=0) : QObject( parent)
  {
    // . . .
  }

public slots:
  void miSlot( void )
  {
    // . . .
  }
signals:
  void miSignal();
};
```

Figura 66. Código fuente genérico de un QObject (objeto Qt)

Las herramientas de Qt

Las herramientas principales que permiten funcionar a Qt son las siguientes:

- ❖ Variables de entorno.
- ❖ Qt Creator.
- ❖ Qmake.
- ❖ UIC.
- ❖ MOC.

Variables de entorno

Para poder usar Qt es necesario que se configuren tres variables: QTDIR, PATH y QMAKESPEC.

- QTDIR. Variable que indica el directorio que contiene la distribución de Qt para permitir la compilación de Qt.
- PATH. Variable que se usa para poder invocar a las herramientas Qt desde la línea de comandos.
- QMAKESPEC. Variable que contiene la ruta del directorio mkspecs, necesario para compilar nuestro programa.

Si se usa Qt Creator, estas variables se configuran de manera automática en la instalación, sin embargo es posible configurarlas manualmente si se desea compilar externamente al IDE.

QT Creator

Es un IDE (Entorno Integrado de Desarrollo) multiplataforma creado inicialmente por Trolltech y mantenido en la actualidad por Nokia. Requiere la versión de las bibliotecas Qt 4.x en adelante. Integra un editor de texto, un depurador y Qt Designer.
Qt Designer es una herramienta muy potente (antes era un programa independiente), utilizada en varias partes de este proyecto, que permite crear "widgets" o elementos gráficos de manera visual, facilitando en gran medida la creación de código para la parte de la GUI. Tras diseñar de manera gráfica un widget, se genera un archivo con extensión '.ui', que no es más que un archivo XML que posteriormente se traduce a una clase en C++ mediante el UIC.

Qmake

Puesto que las aplicaciones hechas con Qt dependen de las librerías Qt, sus propias librerías y en algunos casos, librerías de terceros (como es el caso de Qwt o Qextserialport) y por otra parte, intenta ser totalmente portable, los "makefiles" pueden convertirse en ficheros tremendamente

complejos. Por si fuera poco, no sólo se compila código fuente en C++, sino que el MOC introduce un paso intermedio adicional de pre compilación y encima, los archivos ".ui" (Interfaces de Usuario o widgets) pueden compilarse a partir de archivos XML a clases C++.
Qmake tiene el objetivo de simplificar todo este proceso de generación de código, que con la herramienta "make" clásica, sería tremendamente complejo. Afortunadamente, si se utiliza el IDE Qt Creator, se generan unos archivos ".pro" (archivos de descripción de proyecto) que Qmake interpreta y permite generar todo el código para compilarlo posteriormente con gcc con pulsar sólo un botón.

UIC

UIC (User Interface Compiler) o compilador de interfaces de usuario traduce los ficheros ".ui" generados por Qt Designer (dentro de Qt Creator) a clases C++. Qmake configura esta herramienta de manera automática.
Para poder utilizar una clase proveniente de un archivo ".ui" en el código Qt, los desarrolladores recomiendan especialmente un método denominado "método de herencia múltiple", que consiste en crear una clase que herede de QWidget (lo habitual) y además herede de Ui::NombreForm (donde NombreForm será el nombre del archivo ".ui" también llamado form). Con esto se consiguen dos cosas, por una parte se puede acceder a todos los componentes del form desde el ámbito de la subclase y por otra parte, se pueden usar signals y slots de forma habitual.
Para que todo esto funcione hay que añadir con un #include "ui_nombreform.h" (el cual será el archivo de cabecera correspondiente al fichero ".ui" generado por el UIC). Este método tan simple permite usar código hecho "a mano" y código generado por el UIC a partir de Qt Designer.

MOC

MOC (Meta Object Compiler) o compilador de metaobjetos crea metaobjetos que describen las clases haciendo uso de las signals y los slots.

Clases de Qt.

El número de clases de Qt es muy amplio, por tanto, en este apartado tan sólo se van a describir algunas de las utilizadas en el proyecto.

- **QObject:** Es la clase más importante de todas, ya que describe el objeto básico Qt, que permite usar la gestión simple de memorias o las signals y slots.
- **QWidget:** Es la clase que permite crear el elemento básico de interfaz de usuario.
  - **QMainWindow:** Widget de tipo MainWindow (ventana principal de programa).
  - **QDialogBox**: Widget de tipo cuadro de diálogo estándar.
  - **QLabel:** Widget que permite mostrar etiquetas de texto o imágenes.
  - **QLineEdit:** Widget que muestra una línea de texto.
  - **QTextEdit:** Similar a QLineEdit pero permitiendo mostrar texto, incluso en formato HTML.
  - **QPushButton:** Widget que genera un botón.
  - **QFrame:** Widget de tipo marco.
  - **QMessageBox:** Widget de tipo Message Box.
- **QLayout:** Es la clase que gestiona la geometría de los widgets.
  - **QBoxLayout:** Alinea los widgets de manera vertical u horizontal.
- **QThread:** Clase que gestiona hilos de ejecución independientes del SO que se esté usando.
- **QTimer:** Clase que proporciona una interfaz de alto nivel para los timers.

INSTALACIÓN

La versión de Qt creator que mejor funciona en la Raspberry es la 2.8.1 para ello se debe descargar desde la página http://download.qt.io/official_releases/qtcreator/2.8/2.8.1/ buscar la versión para Windows (qt-creator-windows-opensource-2.8.1.exe) e instalarlo.

INTEGRACIÓN CON LA RASPBERRY PI

Para poder desarrollar sobre esta aplicación, desde un sistema operativo Windows, esto para mayor facilidad, teniendo en cuenta que la mayoría de computadores o portátiles usan este sistema, es necesario hacer un par de configuraciones, tanto en el QT Creator, como en el equipo portátil para que estos permitan hacer una compilación cruzada, es decir qué el código se compile desde un sistema operativo Windows pero para un sistema Basado en Linux.

INSTALACIÓN DE PROGRAMAS ESENCIALES

Se requieren ciertos programas y librerías para poder Integrar el QT Creator con la Raspberry Pi, a continuación se indican cuales son y desde donde se pueden descargar:

- **Smartty:** Este es un software que mediante conexión SSH permite la transferencia de archivos a través del protocolo de conexión segura SCP, además de editar remotamente archivos sobre el equipo al cual se está conectado. Este software puede ser descargado directamente desde la página http://sysprogs.com/files/SmarTTY/SmarTTY-2.2.msi.

- **MinGW:** Es una serie de librerías que permiten dar propiedades al sistema operativo Windows de compilar bajo los compiladores GCC (C, C++, Fortran, Ada, etc.) que se usan sobre sistemas operativos Unix. El archivo comprimido con todas las librerías se puede descargar desde la página: https://docs.google.com/uc?export=download&confirm=acIC&id=0B4D8x6CJEmtuczdiQklwMEs4RUU.

  Luego de descargar el archivo se crea una carpeta llamada MinGW (ver figura 67) en la unidad (C:) del equipo, posteriormente se abre el archivo comprimido descargado y se extrae el contenido de la carpeta mingw del archivo comprimido a la carpeta creada en el PC.

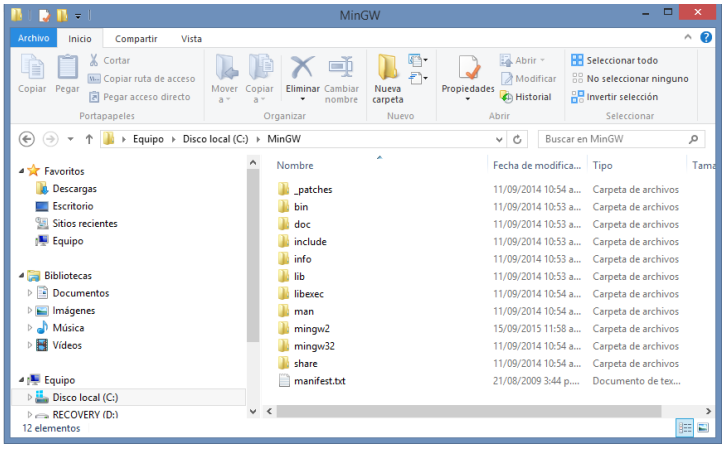

Figura 67. Distribución de carpetas librerías MinGW

- **librerías Qt 4.8.5:** Estas librerías se descargan directamente en la Raspberry y son las que permiten que se ejecute los archivos que son compilados en el QT Creator. La Raspberry debe tener conexión a internet para poder ejecutar los comandos y descargar los archivos.

- **Toolchain RPI:** El toolchain incluye el compilador C++ para poder hacer compilación cruzada, en este caso desde Windows a Linux/ARM. Este programa se puede descargar desde la página [http://sysprogs.com/files/gnutoolchains/raspberry/raspberry-gcc4.6.3.exe.](http://sysprogs.com/files/gnutoolchains/raspberry/raspberry-gcc4.6.3.exe)

  Ejecutar el archivo descargado para iniciar el instalador, aceptar los términos de la licencia, dejar por defecto las opciones que tiene el instalador habilitadas y presionar el botón Install para dar inicio a la instalación como se muestra en la figura 68.

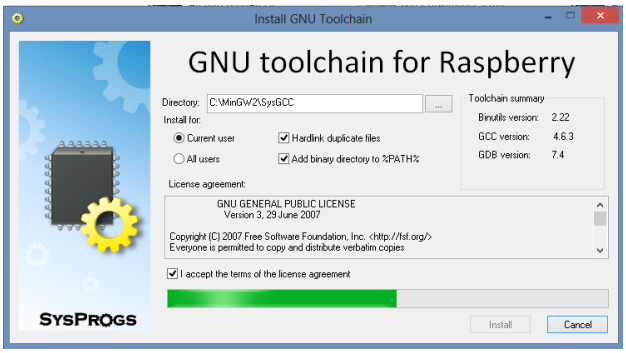

Figura 68. Progreso de instalación GNU toolchain

- **QtCrossTool:** Utilidad que sirve para sincronizar todas las librerías y cabeceras entre la Raspberry y el PC. El archivo comprimido que contiene esta utilidad puede descargarse desde la página http://visualgdb.com/tools/QtCrossTool/QtCrossTool.zip

Configuración del SmarTTY

Iniciar el SmarTTY previamente instalado desde el acceso directo creado en el PC.
Hacer click sobre el icono "Setup a New connection.." como se ve en la figura 69

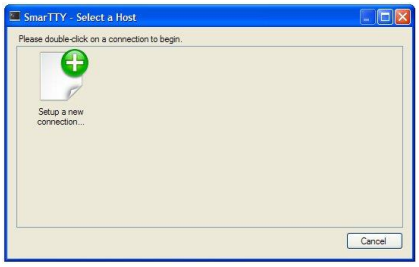

Figura 69 Nueva conexión SmarTTY

Esto hará que se abra una nueva ventana donde se debe escribir los datos básicos de conexión con la Raspberry. Según la red a la cual esté conectada la Raspberry se escribe la dirección IP asignada y como usuario se puede poner root si se tiene habilitado el usuario, o se puede usar el usuario pi y contraseña Raspberry que vienen cargados por defecto en la placa o alguna otra contraseña que haya sido modificada para el usuario pi como se muestra en la figura 70:

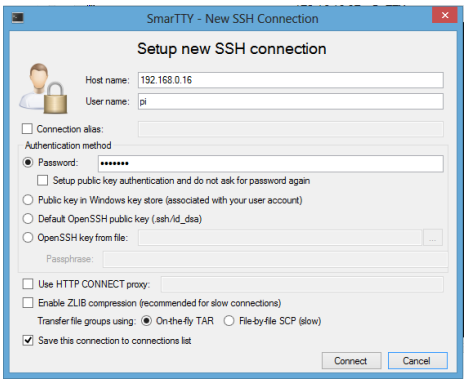

Figura 70. Datos de conexión Raspberry Pi con SmarTTY

Al presionar el botón Connect, los datos de conexión serán guardados para poder utilizarla posteriormente sin tener que introducir de nuevo los datos y se conecta a la Raspberry como se muestra en la figura 71.

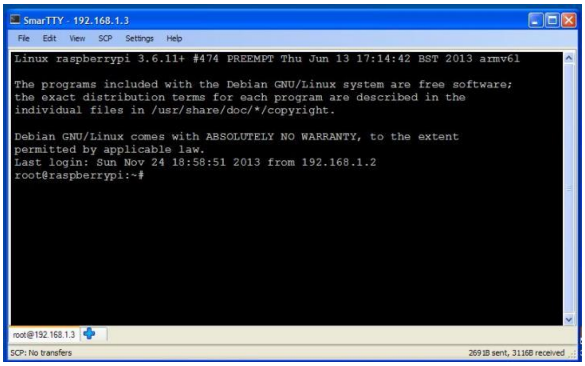

Figura 71. Terminal remota Raspberry Pi con SmarTTY

Instalación librerías QT4 en RPI

Se puede hacer desde la ventana abierta con SmarTTY o desde la propia Raspberry a través de una terminal, por comodidad es mejor hacerlo todo desde el PC con SmarTTY.

Los comandos para ejecutar que permite instalar estas librerías son:
sudo apt-get update
sudo apt-get install libqt4-dev
sudo apt-get install libqt4-sql-sqlite

Sincronización librerías/cabeceras RPI

Se abre el archivo comprimido QtCrossTool.zip descargado previamente y se ejecuta la aplicación QtCrossTool.exe contenida en el haciendo doble clic. Se debe ingresar la ruta de las librerías de acuerdo a como se indica en la figura 72.

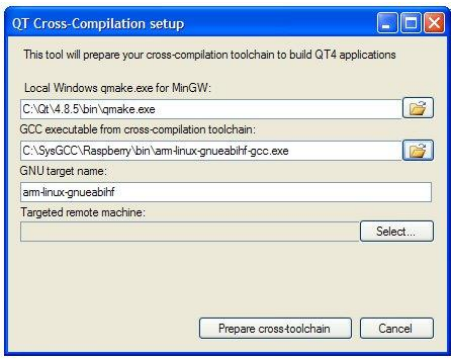

Figura 72. Datos ingresados de ubicación de librerías y cabeceras

La Raspberry debe estar encendida y conectada a la misma red del equipo PC para poder sincronizar las librerías y cabeceras entre ambos, para ello se debe presionar el botón Select el cual abre una instancia del programa SmarTTY donde aparecen los datos de conexión establecidos previamente con la Raspberry (ver figura 73).

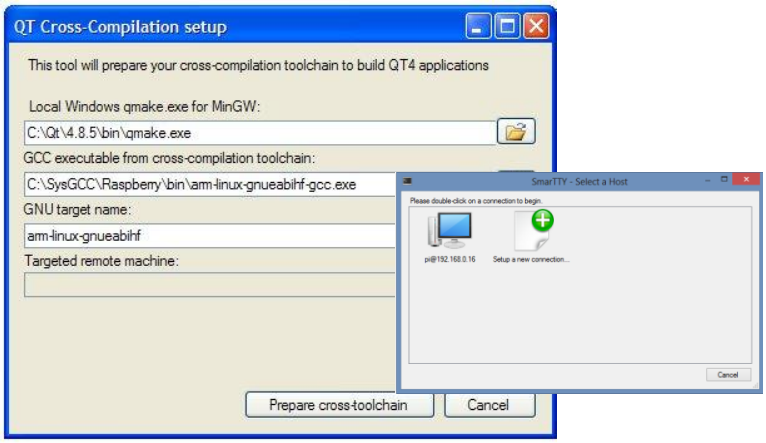

Figura 73. Selección de conexión con Raspberry Pi guardada en el SmarTTY

Se da doble sobre el icono de conexión  y el programa se conectara con la Raspberry, Una vez se haya conectado, empezará a descargar desde la RPI al PC, todas las librerías y cabeceras, para sincronizar ambos con las mismas versiones como se muestra en la figura 74, imprescindible para que la compilación cruzada funcione bien.

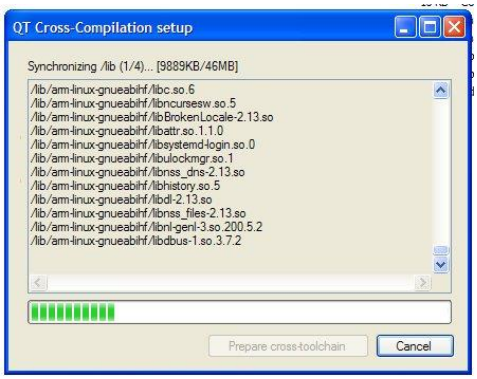

Figura 74. Progreso de sincronización de librerías y cabeceras Raspberry - PC

Configuración QT-Creator

Iniciar el programa QT Creator en el PC, y seleccionar **Tools -> Options**. Seleccionar **Devices -> Add -> Generic Linux Device**, y pulsar Start Wizard.

Rellenar los datos para la conexión con tarjeta RPI como se muestra en la figura 75, pulsar Next y en la siguiente pantalla Finish:

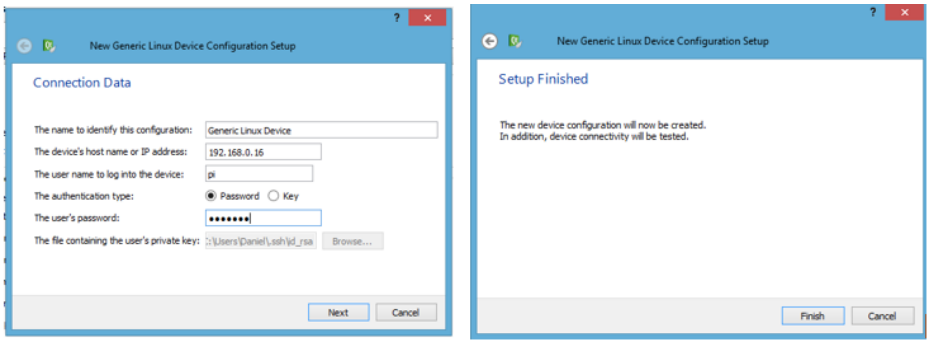

Figura 75. Configuración de conexión Qt Creator con Raspberry Pi

Qt Creator se intentará conectar con la tarjeta, para verificar la conexión con la Raspberry, si la comunicación se lleva a cabo, una ventana indica que el test de conexión finalizo correctamente.

Ahora se selecciona **Build & Run - > Compilers**, para añadir el compilador cruzado C++, y pulsar **Add -> GCC**. Rellenar los datos de acuerdo a como se indica en la imagen 76.

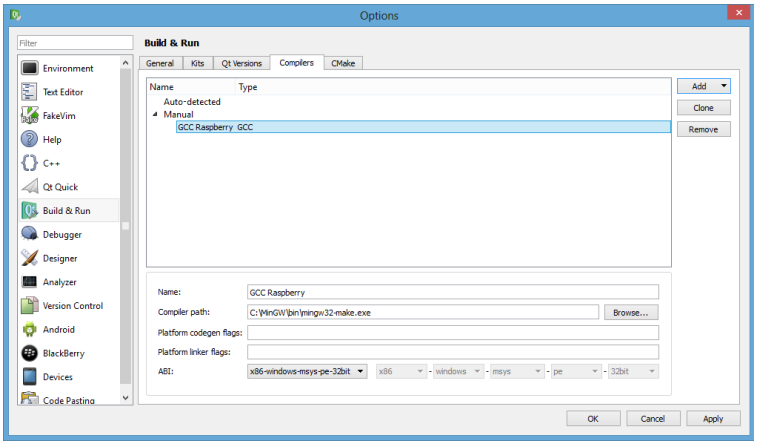

Figura 76. Datos de configuración compilador GCC

Luego se selecciona **QT Versions**, y se pulsa **Add** se abrirá una ventana con el directorio raíz del sistema donde se debe seleccionar la ubicación del archivo **qmake.exe**, el cual se encuentra en la ruta **C:\SysGCC\Raspberry\bin.**

Ahora se debe seleccionar la opción **Kits** de las pestañas superiores y se pulsa el botón **Add**. Se ingresan los datos de acuerdo a como se indica en la siguiente 77.

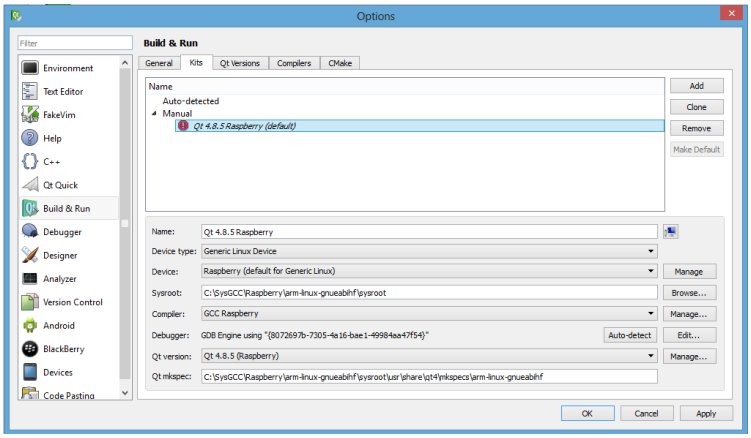

Figura 77. Configuración de Kits en el Qt Creator

Para finalizar se presiona los botones Apply y OK para guardar todos los cambios.

ANEXO 4: INSTALACIÓN Y FUNCIONALIDADES LIBRERÍA WIRINGPI

INSTALACIÓN

Existen 2 maneras para instalar la librería, una es descargando directamente el archivo desde la página principal de la librería (https://git.drogon.net/?p=wiringPi;a=summary), buscar la versión más reciente y presionar el enlace al lado derecho que dice snapshot como se indica en la figura 78, esto hará que se descargue un archivo comprimido.

Figura 78. Directorio de descargas librería WiringPi

Ubicar la carpeta donde fue descargado el archivo, luego por línea de comandos descomprimir el archivo y ejecutarlo.

```
tar xfz wiringPi-XXXXX.tar.gz
cd wiringPi-XXXXX
./build
```

Otra opción es utilizando Git:

Git es un software de control de versiones, libre y de código abierto, diseñado para manejar todo tipo de proyectos, desde pequeños a grandes con rapidez y eficiencia.
En caso de no estar instalado GIT se debería instalar. Para ello en primer lugar hay que actualizar los repositorios y posteriormente instalarlo. Estos comandos se pueden ejecutar desde una conexión con la Raspberry a través del SmarTTY.

```
sudo apt-get update
sudo apt-get install git-core
```

Posteriormente hay que obtener la librería mediante GIT

```
git clone git://git.drogon.net/wiringPi
```

Una vez descargado, hay que ir al directorio donde se encuentra y obtener una versión actualizada.

```
cd wiringPi
git pull origin
```

A continuación, se debe ejecutar el script de creación e instalación en el directorio de WiringPi.

```
./build
```

El nuevo script se encarga de compilar y ejecutar todo, sin que sea necesaria la intervención del usuario.

FUNCIONALIDADES

Para poder utilizar esta librería es necesario indicar que se va a emplear, en la primera parte del programa se incorpora la cabecera de la librería (#include<wiringPi.h>) y en la compilación se especifica que utilice la misma (gcc –o pruebas -l wiringPi pruebas.c).
Hay tres funciones constructoras que permiten trabajar con la librería Wiring Pi, todas ellas devuelven -1 en caso de que se produzca un error:

- **int wiringPiSetup(void)** : Inicializa WiringPi y emplea el convenio de numeración de WiringPi.

- **int wiringPiSetupGpio(void)** : Idéntica a la anterior, pero utiliza la numeración de pines de Broadcom (BCM GPIO).
- **int wiringPiSetupSys(void)** : En este caso utiliza la interfaz /sys/class/gpio, en vez de trabajar directamente sobre el Hardware.

A continuación en la figura 79, se muestra la tabla resumen de los pines GPIO para el Modelo B+/PI2 B (40 Pines) de la Raspberry.

Raspberry Pi GPIO Header

BCM	WiringPi	Name	Physical		Name	WiringPi	BCM
		3.3v	1	2	5v		
2	8	SDA.1	3	4	5V		
3	9	SCL.1	5	6	0v		
4	7	1-Wire	7	8	TxD	15	14
		0v	9	10	RxD	16	15
17	0	GPIO. 0	11	12	GPIO. 1	1	18
27	2	GPIO. 2	13	14	0v		
22	3	GPIO. 3	15	16	GPIO. 4	4	23
		3.3v	17	18	GPIO. 5	5	24
10	12	MOSI	19	20	0v		
9	13	MISO	21	22	GPIO. 6	6	25
11	14	SCLK	23	24	CE0	10	8
		0v	25	26	CE1	11	7
0	30	SDA.0	27	28	SCL.0	31	1
5	21	GPIO.21	29	30	0v		
6	22	GPIO.22	31	32	GPIO.26	26	12
13	23	GPIO.23	33	34	0v		
19	24	GPIO.24	35	36	GPIO.27	27	16
26	25	GPIO.25	37	38	GPIO.28	28	20
		0v	39	40	GPIO.29	29	21
BCM	WiringPi	Name	Physical		Name	WiringPi	BCM

Figura 79. Numeración Puertos GPIO Raspberry pi y su equivalente con la librería WiringPi

FUNCIONES GENERALES

- **void pinMode(int pin, int mode)** : Especifica el pin (primer argumento) y modo (segundo argumento), que puede ser entrada (INPUT), salida (OUTPUT) o salida PWM (PWM_OUTPUT).

- **void digitalWrite(int pin, int value) :** Se utiliza para poner un pin, que previamente ha sido configurado como OUTPUT a dos posibles valores 1 (HIGH) o 0 (LOW).

- **void digitalWriteByte(int value) :** Permite escribir en los 8 pines de la GPIO un valor.

- **void pwmWrite(int pin, int value) :** Escribe el valor del registro PWM (segundo argumento) al pin indicado (primer argumento). El valor suministrado debe estar comprendido entre 0 y 1024, además, hay que tener en cuenta que sólo soporta PWM el pin BCM_GPIO 18.

- **int digitalRead(int pin) :** devuelve el valor leído en el pin indicado (único argumento), que puede ser 1 (HIGH) o 0 (LOW).

- **void pullUpDnControl(int pin, int pud) :** establece sobre el pin indicado (primer argumento) el modo de tensión o resistencia, elevar a 3v3 (PUD_UP), tirar a tierra (PUD_DOWN) o ni elevar ni disminuir (PUD_OFF).

FUNCIONES DE CONTROL PWM

Las siglas PWM corresponden a **pulse-width modulation**, en castellano, Modulación por ancho de pulsos, consistente en cambiar el ciclo de una señal periódica. No se pueden utilizar cuando se trabaja en modo sistema (mode sys).

- **void pwmSetMode(int mode) :** El generador puede trabajar en dos modos balanceado (PWM_MODE_BAL), que es el que utiliza por defecto Raspberry Pi o marca espacio (PWM_MODE_MS).

- **void pwmSetRange(unsigned int range) :** Establece el valor máximo del rango del registro PWM, establecido por defecto a 1024.

- **void pwmSetClock(int divisor) :** Sirve para establecer el divisor del reloj PWM.

FUNCIONES DE TIEMPO

- **unsigned int millis(void) :** Devuelve el número de milisegundos que han transcurrido desde que se ha invocado a una función Wiring Pi.

- **void delay(unsigned int howLong) :** Provoca la pausa del programa durante al menos howLong milisegundos.

- **void delayMicroseconds(unsigned int howLong) :** Análogo al anterior, pero en este caso se estabecen microsegundos.

PROGRAM/THREAD PRIORITY

- **int piHiPri(int priority) :** sirve para establecer la prioridad del programa o thread, que oscila entre 0, valor establecida por defecto y 99 valor máximo y permite la planificación en tiempo real. El valor devuelto es 0 en caso de éxito y -1 en caso de error. Sólo los programas ejecutados como root pueden cambiar su prioridad.

FUNCIONES DE INTERRUPCIONES

- **int waitForInterrupt(int pin, int timeOut) :** permite establecer un tiempo de espera en milisegundos (segundo argumento) para una interrupción definida sobre un pin (primer argumento), en el caso de definir como timeOut -1 esperará indefinidamente. El valor devuelto es 0 en caso de éxito y -1 en caso de error. Esta función está obsoleta y se recomienda utilizar la que se presenta a continuación.

- **int wiringPiISR(int pin, int edgeType, void (*function)(void)) :** Mediante esta function se puede establecer un manejador sobre un pin, además se debe especificar si se detecta mediante un flanco de bajada (INT_EDGE_FALLING) , un flanco de subida (INT_EDGE_RISING), en ambos flancos (INT_EDGE_BOTH) o el pin no ha sido inicializado (INT_EDGE_SETUP). El valor devuelto es 0 en caso de éxito y -1 en caso de error.

FUNCIONES DE PROGRAMACIÓN CONCURRENTE

WiringPi permite la utilización de hilos POSIX, así como mecanismos de exclusión mutua (mutex).

- **int piThreadCreate(void * (* fn) (void *) :** Crea un Thread de una función que ha sido declarada previamente mediante PI_THREAD. La declaración sería similar a PI_TREAD(miThread){ código } y posteriormente, esta función será el argumento de la función. El valor devuelto es 0 en caso de éxito y -1 en caso de error.

- **void piLock(int keyNum) :** Bloquea una clave (keyNum) cuyo valor va de 0 a 3, cuando otro thread intenta bloquear a la clave, queda a la espera de que sea liberado.

- **void piUnlock(int keyNum) :** Permite desbloquear una clave definida como en la función anterior.

OTRAS FUNCIONES

- **int piBoardRev(void) :** Indica la versión de la placa sobre la que se está trabajando.

- **int wpiPinToGpio(int wPiPin) :** Transforma el número de pin según el convenio de WiringPi al convenio de BCM GPIO.

- **void setPadDrive(int group, int value) :** Establece la "fuerza" de los drivers para un grupo de pins.

ANEXO 5: CREACIÓN DE UN PROYECTO NUEVO EN QT CREATOR

CREACIÓN DEL PROYECTO EN QT CREATOR

Para crear un proyecto se debe hacer clic sobre la opción File en el menú superior, esto hará que se desplegué varias opciones y en ella seleccionar la primera “New File or Project” o seguir la secuencia de comando Ctrl+N.

Al hacerlo se abre una ventana con los diferentes tipos de proyectos que se pueden crear, se selecciona la opción “Applications” y se escoge la opción “Qt Gui Application”, luego se presiona el botón Choose.. para continuar como se ve en la figura 80.

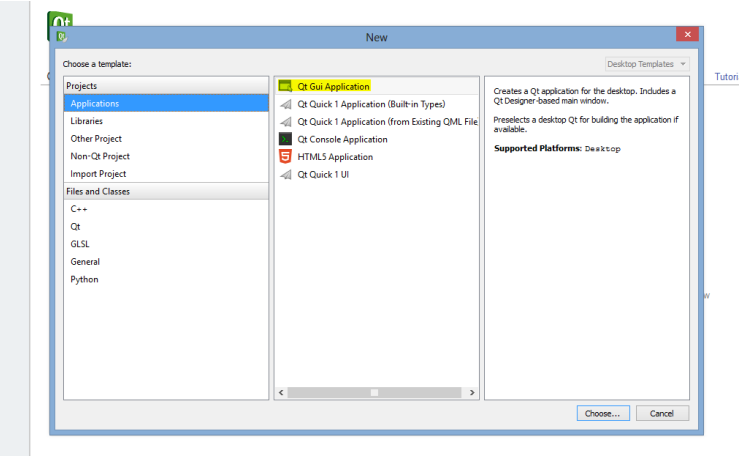

Figura 80. Selección tipo de proyecto

En la ventana siguiente se define el nombre del proyecto y la ruta donde será guardado, la ruta puede ser indicada manualmente o a través del botón “Browser” puede buscar directamente en las carpetas del sistema y seleccionarla. Se presiona el botón Next para continuar.

Por defecto el asistente precarga el Kits antes configurado en el QT Creator, el cual se compone de un conjunto de valores que definen un único entorno, como el dispositivo, compilador, la versión Qt y el comando depurador a usar y algunos metadatos. Cómo solo se configuro un Kit, se deja por defecto los Valores que aparecen y se presiona el botón Next, esto se puede apreciar en la figura 81.

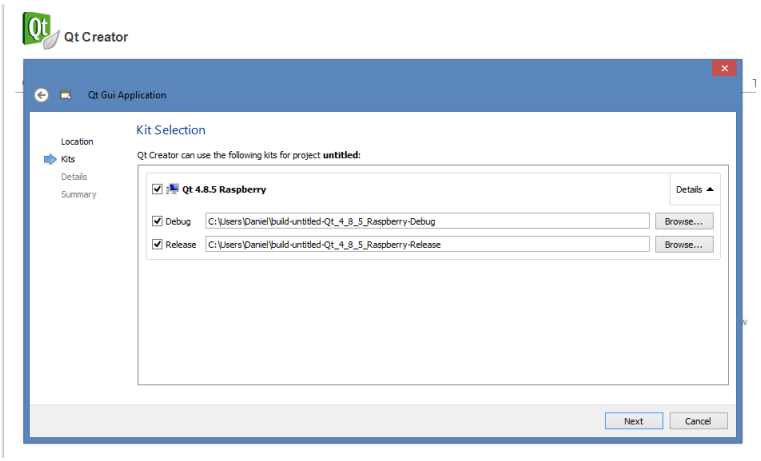

Figura 81. Selección de Compilador y Depurador para el Proyecto

En la ventana siguiente se ingresa el nombre que se le desee dar al método principal Main y se presiona el botón Next para continuar.

La siguiente ventana en aparecer, figura 82, indica un resumen de los archivos que se generaran al crear el proyecto y la ruta donde serán creados, de igual forma permite adicionar un control de versión a los cambios que se hagan en el proyecto. Para finalizar se presiona el botón Finish.

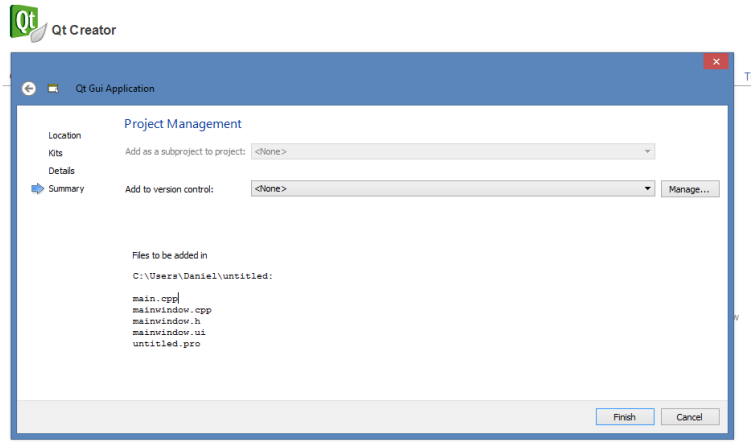

Figura 82. Resumen de configuración del proyecto

Se habilita entonces el entorno de desarrollo donde se muestra el directorio raíz del proyecto, donde está el archivo principal de configuración del proyecto, los cabeceros o archivos .h, los

archivos fuentes .cpp, los form o ventanas de diseño gráfico .ui, el editor de código fuente y demás herramientas de compilación y depuración, como se ve en las figuras 83, 84 y 85.

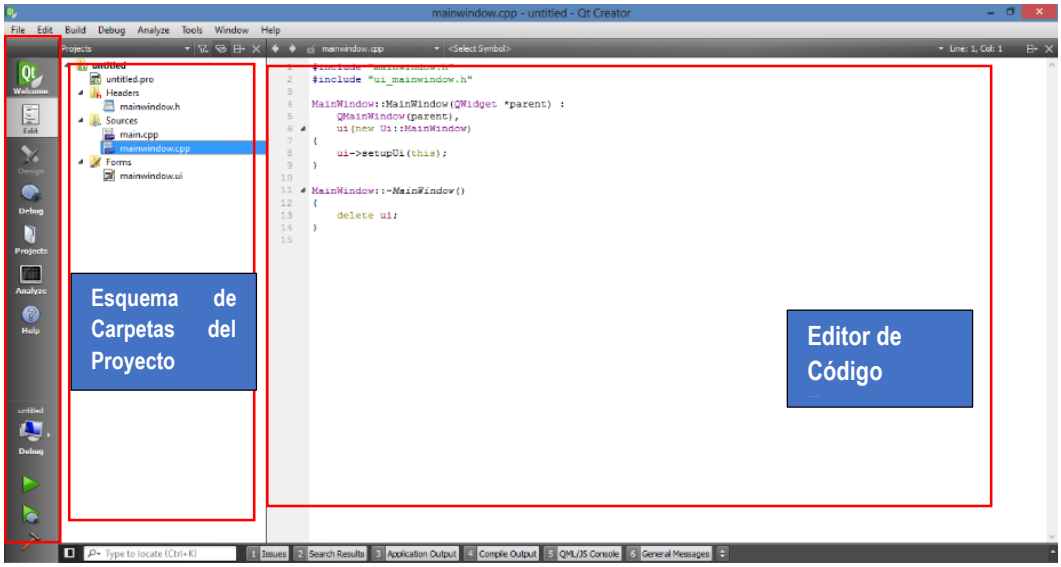

Figura 83. Entorno de escritura del código fuente

Menú de herramientas

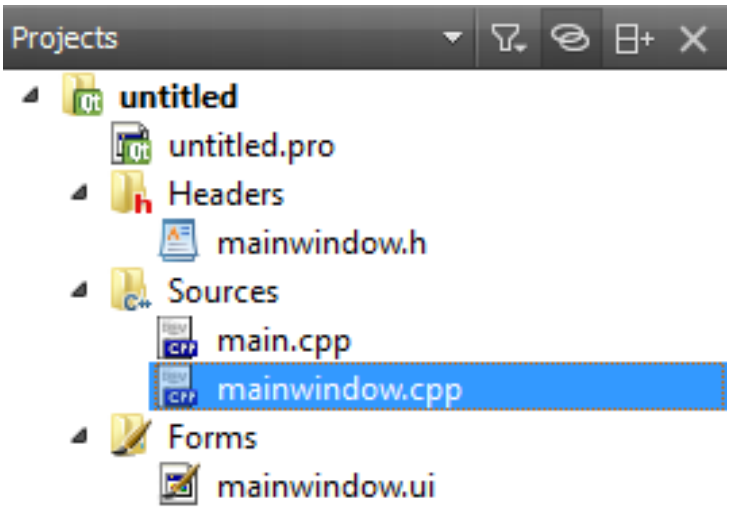

Figura 84. Esquema de carpetas proyecto en el Qt Creator

Figura 85. Editor de código fuente

CREAR NUEVOS MÓDULOS

Para adicionar nuevos archivos de cabecera, código fuente, diseño gráfico entre otros, basta con hacer clic sobre la carpeta principal en el esquema de carpetas del proyecto y seleccionar la opción "Add New". Al hacerlo este abrirá un asistente como se ve en la figura 86, donde permite escoger los diferentes tipos de archivos que se desee adicionar al proyecto.

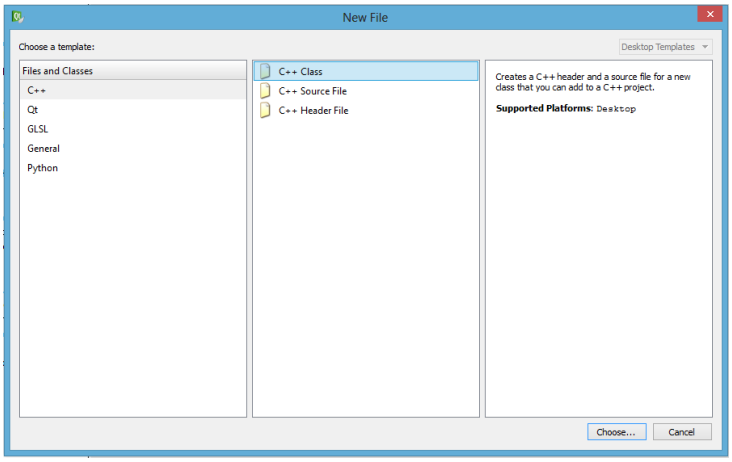

Figura 86. Ventana de selección de archivos de tipo C++

Para adicionar una nueva clase, cabecera o archivo de código fuente en blanco independiente de los objetos y estructuras del framework del Qt Creator, se debe escoger el tipo de archivo C++ y se presiona el botón "Choose.." para continuar con el asistente, el cual permite definir el nombre

de la clase o del archivo según la opción que se haya seleccionado, así mismo la ubicación del archivo, se debe presionar luego el botón “Next” para continuar, en el paso siguiente se muestra un resumen de los archivos que se van a crear y el proyecto al cual pertenecerán, para finalizar se presiona el botón “Finish”.

Para adicionar nuevos archivos al proyecto con la estructura del framework del Qt creator, se debe escoger la opción Qt al lado izquierdo y luego, seleccionar al lado derecho el tipo de estructura de archivos que se deseen crear, ya sea una clase, un formulario o ventana de diseño o un archivo de recursos, esta ventana de selección se muestra en la figura 87. Se presiona el botón “Choose..” para continuar con el asistente

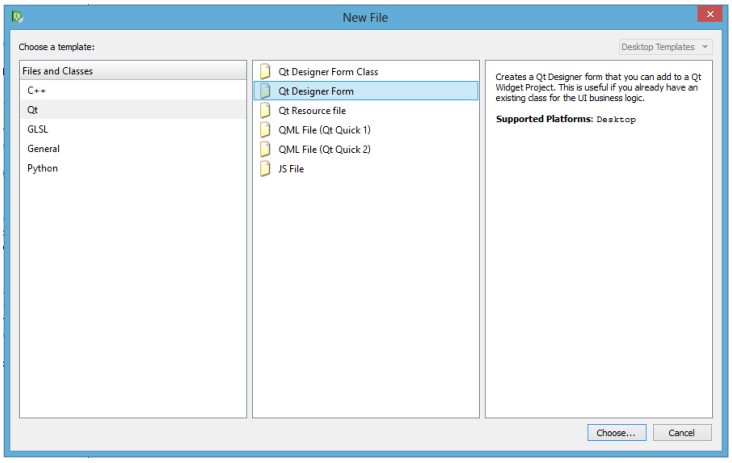

Figura 87. Ventana de selección de archivos de tipo Qt

Existen diferentes tipos de formularios o template prediseñados que pueden ser seleccionados al momento de crearlos o simplemente seleccionar una ventana principal o una sub ventana como se ve en la figura 88. Para continuar se presiona el botón “Next”.

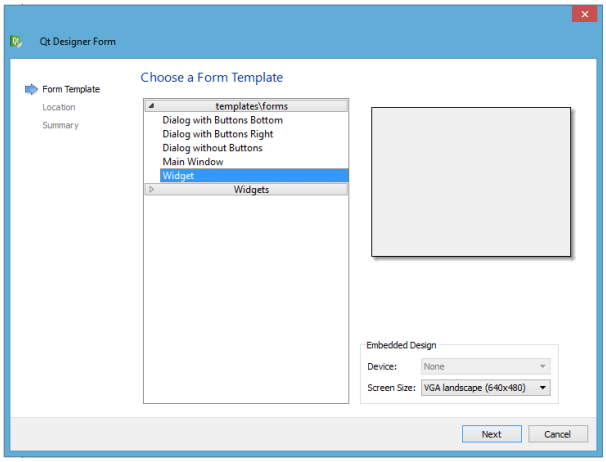

Figura 88. Selección de los diferentes template

En la siguiente ventana se escoge el nombre de la clase o del archivo según la opción que se haya escogido y se presiona el botón “Next” para continuar.

En una nueva ventana, figura 89, se observa un resumen de los archivos a ser creados y el proyecto al cual pertenecerán, para finalizar se presiona el botón “Finish”.

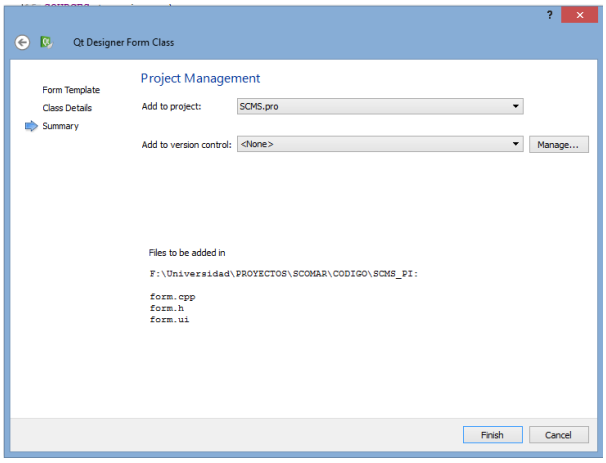

Figura 89. Resumen de configuración y creación del nuevo archivo

ANEXO 6: CÓDIGO FUENTE

CLASE CANTIDAD

cantidad.h

```
#ifndef CANTIDAD_H
#define CANTIDAD_H

#include <QWidget>
#include "ingreso.h"

namespace Ui {
class cantidad;
}

class cantidad : public QWidget
{
    Q_OBJECT

public:
    explicit cantidad(QWidget *parent = 0);
    ~cantidad();
    void definirCantidad(int, int, QString archivo);

signals:
    void cantidadChanged(int);

private slots:
    void on_pushButton_clicked();
    void on_pushButton_2_clicked();

private:
    Ui::cantidad *ui;
    int cantidadDefinida;
    QString NombreArchivo;
};

#endif // CANTIDAD_H
```

cantidad.cpp

```
#include "cantidad.h"
#include "ui_cantidad.h"
#include <QSqlQuery>

cantidad::cantidad(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::cantidad)
{
    ui->setupUi(this);
    cantidadDefinida = 0;
}

cantidad::~cantidad()
{
    delete ui;
}

void cantidad::on_pushButton_clicked()
{
    this->hide();
    emit cantidadChanged(ui->spinBox->text().toInt());
    ingreso conn;
    conn.connOpen();
    QString qry = "UPDATE PROGRAMA_CARGADO SET CANTIDAD_FALTANTE = ? WHERE NOMBRE_PROGRAMA = ?";
```

```
    QSqlQuery query(qry);
    query.addBindValue(ui->spinBox->text().toInt());
    query.addBindValue(NombreArchivo);
    query.exec();
    conn.connClose();
}

void cantidad::on_pushButton_2_clicked()
{
    this->close();
}

void cantidad::definirCantidad(int valorCantidad, int valorProg, QString archivo){
    this->NombreArchivo = archivo;
    ui->spinBox->setValue(valorProg);
    ui->label_cantidad->setText(QString::number(valorCantidad));
}
```

CLASE CARGAR PROGRAMA

cargarprograma.h

```
#ifndef CARGARPROGRAMA_H
#define CARGARPROGRAMA_H

#include <QMainWindow>
#include <QDir>
#include <QDebug>
#include <QRegExp>
#include <QFileSystemModel>
#include <QSplitter>
//#include <qfilesystemmodel.h>
#include "MÓDULOprogramacion.h"
#include "guardararchivo.h"
#include "copiararchivo.h"
#include "ingreso.h"

namespace Ui {
class cargarPrograma;
}

class cargarPrograma : public QMainWindow
{
    Q_OBJECT

public:
    explicit cargarPrograma(QWidget *parent = 0);
    ~cargarPrograma();

private slots:
    void on_actionNuevo_triggered();
    void on_actionEditar_triggered();
    void on_actionSalir_triggered();
    void on_actionCopiar_triggered();
    void on_actionEliminar_triggered();
    void mostrarArchivo(QString);
    void on_listView_clicked(const QModelIndex &index);
    void on_pushButton_clicked();

signals:
    void archivoChanged(QString);

private:
    Ui::cargarPrograma *ui;
    QStringList List;
    QString rutaDisenos,rutaCortes;
```

```
    //QFileSystemModel *model;
    QString archivoLista;
};

#endif // CARGARPROGRAMA_H
```

cargarprograma.cpp

```
#include "cargarprograma.h"
#include "ui_cargarprograma.h"

#include <qsortfilterproxymodel.h>

cargarPrograma::cargarPrograma(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::cargarPrograma)
{
    ui->setupUi(this);
    //rutaProgramas = "F:/Universidad/PROYECTOS/SCOMAR/PROGRAMAS/";
    rutaDisenos ="/home/pi/Desktop/PRUEBAS/PROGRAMAS/DISEÑOS";
    rutaCortes ="/home/pi/Desktop/PRUEBAS/PROGRAMAS/CORTES";

    //model = new QFileSystemModel;
    QFileSystemModel *modelDiseno = new QFileSystemModel;
    modelDiseno->setRootPath(rutaDisenos);

    ui->listView->setModel(modelDiseno);
    ui->listView->setRootIndex(modelDiseno->index(rutaDisenos));
    ui->listView->show();
    ui->label_Total->setText(QString("%1").arg(ui->listView->model()->rowCount()));

    QFileSystemModel *modelCortes = new QFileSystemModel;
    modelCortes->setRootPath(rutaCortes);
    ui->listCorte->setModel(modelCortes);
    ui->listCorte->setRootIndex(modelCortes->index(rutaCortes));
    ui->listCorte->show();
}

cargarPrograma::~cargarPrograma()
{
    delete ui;
}

void cargarPrograma::on_actionNuevo_triggered()
{
    guardarArchivo *guardar = new guardarArchivo;
    guardar->show();
}

void cargarPrograma::on_actionEditar_triggered()
{
    if(!archivoLista.isNull()){
        MÓDULOProgramacion *MÓDULOP = new MÓDULOProgramacion;
        MÓDULOP->rutaArchivo = rutaDisenos + archivoLista;
        MÓDULOP->CargarArchivo();
        MÓDULOP->show();
    }else{
        QMessageBox msgBox;
        msgBox.setWindowTitle("MENSAJE");
        msgBox.setText("SE DEBE SELECCIONAR UN ARCHIVO");
        msgBox.exec();
    }

}

void cargarPrograma::on_actionSalir_triggered()
```

```
{
    this->close();
}

void cargarPrograma::on_actionCopiar_triggered()
{
    if(!archivoLista.isNull()){
        copiarArchivo *copiar = new copiarArchivo;
        copiar->setRuta(rutaDisenos + archivoLista);
        copiar->show();
    }else{
        QMessageBox msgBox;
        msgBox.setWindowTitle("MENSAJE");
        msgBox.setText("SE DEBE SELECCIONAR UN ARCHIVO");
        msgBox.exec();
    }
}

void cargarPrograma::on_actionEliminar_triggered()
{
    if(!archivoLista.isNull()){
        QFile file(rutaDisenos + archivoLista);
        if(!file.remove()){
            QMessageBox msgBox;
            msgBox.setWindowTitle("MENSAJE");
            msgBox.setText("ERROR AL ELIMINAR EL ARCHIVO");
            msgBox.exec();
        }
    }else{
        QMessageBox msgBox;
        msgBox.setWindowTitle("MENSAJE");
        msgBox.setText("SE DEBE SELECCIONAR UN ARCHIVO");
        msgBox.exec();
    }
}

void cargarPrograma::mostrarArchivo(QString ruta){
    ui->textEdit->clear();
    QString fileName = ruta + archivoLista;
    if ( fileName.isEmpty() )
      return;

    QFile file( fileName );
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
       return;

    int cont = 0;
    QTextStream in( &file );
    QString line;
    while (!in.atEnd()) {
      if(cont == 0){
        in.readLine();
      }else{
          if ((cont % 2) == 0) {
               line = "Linea " + QString::number(cont) + " >> " + in.readLine();
               ui->textEdit->append( line );
          }else{
               line = "Linea " + QString::number(cont) + " << " + in.readLine();
               ui->textEdit->append( line );
          }

      }
      cont++;
    }
}

void cargarPrograma::on_listView_clicked(const QModelIndex &index)
{
    if(ui->listView->isActiveWindow() || ui->listView->isTopLevel()){
        archivoLista = ui->listView->model()->data(index).toString();
        mostrarArchivo(rutaDisenos);
```

```
    }else if(ui->listCorte->isActiveWindow() || ui->listCorte->isTopLevel()){
        archivoLista = ui->listCorte->model()->data(index).toString();
        mostrarArchivo(rutaCortes);
    }
}

void cargarPrograma::on_pushButton_clicked()
{
    this->hide();
    emit archivoChanged(archivoLista);
    ingreso conn;
    QString qry;
    conn.connOpen();

    qry = "DELETE FROM PROGRAMA_CARGADO";
    QSqlQuery h(qry);
    h.exec();

    qry = "INSERT INTO PROGRAMA_CARGADO (NOMBRE_PROGRAMA, PERFIL_USUARIO, CANTIDAD_FALTANTE)
VALUES (?,?,?)";
    QSqlQuery query(qry);
    query.addBindValue(archivoLista);
    query.addBindValue("ADM");
    query.addBindValue(0);
    query.exec();
    conn.connClose();
}
```

CLASE COPIAR ARCHIVO

copiararchivo.h

```
#ifndef COPIARARCHIVO_H
#define COPIARARCHIVO_H

#include <QWidget>
#include <QFile>
#include <QMessageBox>

namespace Ui {
class copiarArchivo;
}

class copiarArchivo : public QWidget
{
    Q_OBJECT

public:
    explicit copiarArchivo(QWidget *parent = 0);
    ~copiarArchivo();
    void setRuta(QString ruta);

private slots:
    void on_buttonBox_accepted();
    void on_buttonBox_rejected();

private:
    Ui::copiarArchivo *ui;
    QString rutaOrigen, rutaCopia;
};

#endif // COPIARARCHIVO_H
```

copiararchivo.cpp

```
#include "copiararchivo.h"
#include "ui_copiararchivo.h"

copiarArchivo::copiarArchivo(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::copiarArchivo)
{
    ui->setupUi(this);
}

copiarArchivo::~copiarArchivo()
{
    delete ui;
}

void copiarArchivo::setRuta(QString ruta){
    rutaOrigen = ruta;
}

void copiarArchivo::on_buttonBox_accepted()
{
    QFile myFile (rutaOrigen);
    QString nombreArchivo = ui->lineEdit->text();
    if (!myFile.copy("/home/pi/Desktop/PRUEBAS/PROGRAMAS/"+ nombreArchivo + ".csv")){
    //if (!myFile.copy("F:/Universidad/PROYECTOS/SCOMAR/PROGRAMAS/"+ nombreArchivo + ".csv")){
        QMessageBox msgBox;
        msgBox.setWindowTitle("MENSAJE");
        msgBox.setText("ERROR COPIANDO ARCHIVO");
        msgBox.exec();
        this->close();
    }else{
        this->close();
    }
}

void copiarArchivo::on_buttonBox_rejected()
{
    this->close();
}
```

CLASE GUARDAR ARCHIVO

guardararchivo.h

```
#ifndef GUARDARARCHIVO_H
#define GUARDARARCHIVO_H

#include <QWidget>
#include <QFile>
#include <QMessageBox>

namespace Ui {
class guardarArchivo;
}

class guardarArchivo : public QWidget
{
    Q_OBJECT

public:
    explicit guardarArchivo(QWidget *parent = 0);
    ~guardarArchivo();
```

```
private slots:
    void on_buttonBox_accepted();

private:
    Ui::guardarArchivo *ui;
};

#endif // GUARDARARCHIVO_H
```

guardararchivo.cpp

```
#include "guardararchivo.h"
#include "ui_guardararchivo.h"

guardarArchivo::guardarArchivo(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::guardarArchivo)
{
    ui->setupUi(this);
}

guardarArchivo::~guardarArchivo()
{
    delete ui;
}

void guardarArchivo::on_buttonBox_accepted()
{
    QString nombreArchivo = ui->lineEdit->text();
    //QFile file("F:/Universidad/PROYECTOS/SCOMAR/PROGRAMAS/" + nombreArchivo + ".csv");
    QFile file("/home/pi/Desktop/PRUEBAS/PROGRAMAS/" + nombreArchivo + ".csv");

    if(!file.open(QFile::WriteOnly | QFile::Text )){
        QMessageBox msgBox;
         msgBox.setText("ERROR CREANDO EL ARCHIVO");
         msgBox.exec();
    }
    file.close();
}
```

CLASE MODIFICAR CICLOS

modificarciclos.h

```
#ifndef MODIFICARCICLOS_H
#define MODIFICARCICLOS_H

#include <QWidget>
#include <qspinbox.h>

namespace Ui {
class modificarCiclos;
}

class modificarCiclos : public QWidget
{
    Q_OBJECT

public:
    explicit modificarCiclos(QWidget *parent = 0);
    ~modificarCiclos();
    void cargarCiclos(QStringList);
```

```
    int fila, columna;

public slots:
    void EditarCelda();

signals:
    void ciclosChanged(QStringList);

private slots:
    void spinBoxChange(int value);
    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

private:
    Ui::modificarCiclos *ui;
    QStringList data;
    QStringList rowData;
    QStringList datosLista;
};

#endif // MODIFICARCICLOS_H
```

modificarciclos.cpp

```
#include "modificarciclos.h"
#include "ui_modificarciclos.h"

modificarCiclos::modificarCiclos(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::modificarCiclos)
{
    ui->setupUi(this);
}

modificarCiclos::~modificarCiclos()
{
    delete ui;
}

void modificarCiclos::cargarCiclos(QStringList lista){
    datosLista = lista;
    int x = 0;
    for (int var = 0; var < datosLista.count(); ++var) {
        rowData = datosLista.at(var).split(";");
        data << rowData[6];

        if (data[var] != "X" && data[var] != "1"){
            ui->tableWidget->setRowCount(x+1);
            ui->tableWidget->setItem(x, 0, new QTableWidgetItem(QString::number(var+1)));
            ui->tableWidget->setItem(x, 1, new QTableWidgetItem(data[var]));
            x++;
        }

    }
}

void modificarCiclos::EditarCelda(){
    //ui->tableWidget->removeCellWidget(fila,columna);

    fila = ui->tableWidget->currentItem()->row();
    columna = ui->tableWidget->currentItem()->column();

    QSpinBox *widget = new QSpinBox();
    widget->setValue(ui->tableWidget->item(fila,columna)->text().toInt());
    ui->tableWidget->setCellWidget(fila, columna, widget);
```

```
    connect(widget, SIGNAL(valueChanged(int)),this, SLOT(spinBoxChange(int)));
}

void modificarCiclos::spinBoxChange(int value){
    ui->tableWidget->item(fila,columna)->setText(QString::number(value));
}

void modificarCiclos::on_pushButton_clicked()
{
    QString datos;
    for (int var = 0; var < datosLista.count(); ++var) {
        rowData = datosLista.at(var).split(";");

        rowData[6] = ui->tableWidget->item( var, 1 )->text();
        datos = rowData[0] +";"+ rowData[1] +";"+ rowData[2] +";"+ rowData[3] +";"+ rowData[4]
+";"+ rowData[5] +";"+ rowData[6];
        datosLista.removeAt(var);
        datosLista.insert(var, datos);
    }
    this->hide();
    emit ciclosChanged(datosLista);
}

void modificarCiclos::on_pushButton_2_clicked()
{
    this->hide();
}
```

CLASE MÓDULO DE PROGRAMACIÓN

MÓDULOprogramacion.h

```
#ifndef MÓDULOPROGRAMACION_H
#define MÓDULOPROGRAMACION_H

#include <QMainWindow>
#include <QFile>
#include <QMessageBox>
#include <QFileDialog>
#include <QStandardItemModel>
#include <QDebug>

namespace Ui {
class MÓDULOProgramacion;
}

class MÓDULOProgramacion : public QMainWindow
{
    Q_OBJECT

public:
    explicit MÓDULOProgramacion(QWidget *parent = 0);
    ~MÓDULOProgramacion();
    QString rutaArchivo;
    void CargarArchivo();
    int fila, columna;

private slots:
    void on_actionGuardar_Programa_triggered();
    void on_actionInsertar_Linea_triggered();
    void spinBoxChange(int value);

    void on_actionSalir_triggered();

    void on_actionEliminar_Linea_triggered();
```

```
public slots:
    void EditarCelda();

private:
    Ui::MÓDULOProgramacion *ui;
};

#endif // MÓDULOPROGRAMACION_H
```

MÓDULOprogramacion.cpp

```
#include "MÓDULOprogramacion.h"
#include "ui_MÓDULOprogramacion.h"

MÓDULOProgramacion::MÓDULOProgramacion(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MÓDULOProgramacion)
{
    ui->setupUi(this);
}

MÓDULOProgramacion::~MÓDULOProgramacion()
{
    delete ui;
}

void MÓDULOProgramacion::CargarArchivo(){

    QString data;
    QFile importedCSV(rutaArchivo);
    QStringList rowOfData;
    QStringList rowData;
    data.clear();
    rowOfData.clear();
    rowData.clear();

    if (importedCSV.open(QFile::ReadOnly))
    {
        data = importedCSV.readAll();
        rowOfData = data.split("\n");           //Value on each row
        importedCSV.close();
    }

    for (int x = 1; x < rowOfData.size()-1; x++)  //rowOfData.size() gives the number of row
    {
        rowData = rowOfData.at(x).split(";");   //Number of collumn

        for (int y = 0; y < rowData.size(); y++)
        {
            ui->tableWidget->setRowCount(x);
            ui->tableWidget->setItem(x-1, y, new QTableWidgetItem(rowData[y]));

        }
    }
    ui->statusBar->showMessage(tr("File successfully loaded."), 3000);
}

void MÓDULOProgramacion::on_actionGuardar_Programa_triggered()
{
    QFile file(rutaArchivo);

    if (file.open(QFile::WriteOnly | QFile::Truncate))
    {
        QTextStream data(&file);
        QStringList strList;
        for( int c = 0; c < ui->tableWidget->columnCount(); ++c )
        {
```

```
            strList << ui->tableWidget->horizontalHeaderItem(c)-
>data(Qt::DisplayRole).toString();
        }

        data << strList.join(";") << "\n";
        for( int r = 0; r < ui->tableWidget->rowCount(); ++r )
        {
            strList.clear();
            for( int c = 0; c < ui->tableWidget->columnCount(); ++c )
            {   QTableWidgetItem* item = ui->tableWidget->item(r,c);        //Load items
                if (!item || item->text().isEmpty())                        //Test if there is
something at item(r,c)
                {
                    ui->tableWidget->setItem(r,c,new QTableWidgetItem("0"));//IF there is nothing
write 0
                }
                strList << ui->tableWidget->item( r, c )->text();

            }
            data << strList.join( ";" )+"\n";
        }
        ui->statusBar->showMessage(tr("File saved successfully."), 3000);
        file.close();
    }
}

void MÓDULOProgramacion::on_actionInsertar_Linea_triggered()
{
    int currentRow;
    currentRow =  ui->tableWidget->rowCount();
    ui->tableWidget->setRowCount(currentRow + 1);

    ui->tableWidget->setItem(currentRow, 0, new QTableWidgetItem(ui->spinBox->text()));
    ui->tableWidget->setItem(currentRow, 1, new QTableWidgetItem(ui->spinBox_3->text()));
    ui->tableWidget->setItem(currentRow, 2, new QTableWidgetItem(ui->spinBox_5->text()));
    ui->tableWidget->setItem(currentRow, 3, new QTableWidgetItem(ui->spinBox_7->text()));
    ui->tableWidget->setItem(currentRow, 4, new QTableWidgetItem(ui->spinBox_9->text()));
    ui->tableWidget->setItem(currentRow, 5, new QTableWidgetItem(ui->spinBox_11->text()));
    //ui->tableWidget->setItem(currentRow, 6, new QTableWidgetItem(ui->spinBox_13->text()));
    //ui->tableWidget->setItem(currentRow, 7, new QTableWidgetItem(ui->spinBox_15->text()));
    ui->tableWidget->setItem(currentRow, 6, new QTableWidgetItem(ui->spinBox_17->text()));

    currentRow =  ui->tableWidget->rowCount();
    ui->tableWidget->setRowCount(currentRow + 1);

    ui->tableWidget->setItem(currentRow, 0, new QTableWidgetItem(ui->spinBox_2->text()));
    ui->tableWidget->setItem(currentRow, 1, new QTableWidgetItem(ui->spinBox_3->text()));
    ui->tableWidget->setItem(currentRow, 2, new QTableWidgetItem(ui->spinBox_6->text()));
    ui->tableWidget->setItem(currentRow, 3, new QTableWidgetItem(ui->spinBox_8->text()));
    ui->tableWidget->setItem(currentRow, 4, new QTableWidgetItem(ui->spinBox_10->text()));
    ui->tableWidget->setItem(currentRow, 5, new QTableWidgetItem(ui->spinBox_12->text()));
    //ui->tableWidget->setItem(currentRow, 6, new QTableWidgetItem(ui->spinBox_14->text()));
    //ui->tableWidget->setItem(currentRow, 7, new QTableWidgetItem(ui->spinBox_16->text()));
    ui->tableWidget->setItem(currentRow, 6, new QTableWidgetItem("X"));
}

void MÓDULOProgramacion::EditarCelda(){
    ui->tableWidget->removeCellWidget(fila,columna);

    fila = ui->tableWidget->currentItem()->row();
    columna = ui->tableWidget->currentItem()->column();

    QSpinBox *widget = new QSpinBox();
    widget->setValue(ui->tableWidget->item(fila,columna)->text().toInt());
    ui->tableWidget->setCellWidget(fila, columna, widget);

    QObject::connect(widget, SIGNAL(valueChanged(int)),this, SLOT(spinBoxChange(int)));
}

void MÓDULOProgramacion::spinBoxChange(int value){
    ui->tableWidget->item(fila,columna)->setText(QString::number(value));
```

```
}

void MÓDULOProgramacion::on_actionSalir_triggered()
{
    this->close();
}

void MÓDULOProgramacion::on_actionEliminar_Linea_triggered()
{
    if(ui->tableWidget->isItemSelected(ui->tableWidget->currentItem())){
        QMessageBox msgBox;
        msgBox.setWindowTitle("MENSAJE");
        msgBox.setText("Realmente desea Eliminar las Lineas?");
        msgBox.setStandardButtons(QMessageBox::Yes| QMessageBox::No);

        if (msgBox.exec() == QMessageBox::Yes) {
            ui->tableWidget->removeRow(ui->tableWidget->currentRow());
            ui->tableWidget->removeRow(ui->tableWidget->currentRow()+1);
        }
    }else{
        QMessageBox msgBox;
        msgBox.setWindowTitle("MENSAJE");
        msgBox.setText("SE DEBE SELECCIONAR UNA LINEA");
        msgBox.exec();
    }
}
```

CLASE PANTALLA PRINCIPAL

pantallaprincipal.h

```
#ifndef PANTALLAPRINCIPAL_H
#define PANTALLAPRINCIPAL_H

#include <QMainWindow>
#include "ingreso.h"
#include "cantidad.h"
#include "modificarciclos.h"
#include "cargarprograma.h"
#include "threadsensores.h"
#include "mantenimiento.h"
#include <QDebug>
#include <QMessageBox>
#include "wiringPi.h"

namespace Ui {
class pantallaPrincipal;
}

class pantallaPrincipal : public QMainWindow
{
    Q_OBJECT

public:
    explicit pantallaPrincipal(QWidget *parent = 0);
    ~pantallaPrincipal();
    int cantidadActual;
    int cantidadProg;
    void configGPIO();
    void definirSalidas(QString data, QString sentido);
    QString Sensor1, Sensor2;
    ThreadSensores *Sensores;
    QMessageBox msgBox;
    void lecturaSensores();

public slots:
```

```
    void abrirCargarPrograma();
    void VerPrograma(QString);
    void progCantidad(int);
    void leerPrograma(QString);
    void progCiclos(QStringList list){
        strList.clear();
        strList << list;
    }
    void cambiarContLineas(int linea){
        contLineas = linea;
    }

private slots:
    void on_pushButton_9_clicked();
    void on_pushButton_clicked();
    void on_pushButton_10_clicked(bool checked);
    void on_pushButton_11_clicked();
    void on_pushButton_3_clicked();

signals:
    void sensorIzqChanged(QString);
    void sensorDerChanged(QString);

private:
    Ui::pantallaPrincipal *ui;
    QString rutaProgramas;
    QString NombreArchivo;
    int contLineas;
    QStringList strList;
    QStringList rowData, salidas;
    int sensorDer, sensorIzq, error, contaCiclos;
    int
vel1,vel2,vel3,gui1,gui2,gui3,gui4,llaDel,llaDelMed,llaTra,llaTraMed,ronDelDer,ronTraDer,ronDelIz
q,ronTraIzq;
    QString inputstate;
};

#endif // PANTALLAPRINCIPAL_H
```

pantallaprincipal.cpp

```
#include "pantallaprincipal.h"
#include "ui_pantallaprincipal.h"

pantallaPrincipal::pantallaPrincipal(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::pantallaPrincipal)
{
    ui->setupUi(this);

    ingreso conn;
    cantidadActual = 0;
    cantidadProg = 0;
    rutaProgramas = "/home/pi/Desktop/PRUEBAS/PROGRAMAS/";
    sensorDer = 0;
    sensorIzq = 0;
    error = 0;
    contLineas = 0;
    contaCiclos = 0;

    wiringPiSetup();
    configGPIO();


  conn.connOpen();

    QString nombrePrograma;
```

```
    int cantidad;
    QSqlQuery qry;
    qry.prepare("SELECT NOMBRE_PROGRAMA, CANTIDAD_FALTANTE FROM PROGRAMA_CARGADO");
    if(!qry.exec() )
      qDebug() << qry.lastError();
    else
        while(qry.next()){
            nombrePrograma = qry.value(0).toString();
            cantidad = qry.value(1).toInt();
        }


    if(cantidad > 0){
        QMessageBox msgBox;
        msgBox.setWindowTitle("MENSAJE");
        msgBox.setText("Desea continuar con la programacion?");
        msgBox.setStandardButtons(QMessageBox::Yes| QMessageBox::No);

        if (msgBox.exec() == QMessageBox::Yes) {
            this->VerPrograma(nombrePrograma);
            cantidadProg = cantidad;
            ui->cantidadprogEdit->setText(QString::number(cantidadProg));
            ui->cantidadresEdit->setText(QString::number(cantidadActual));
            lecturaSensores();
        }else{
            this->VerPrograma("");
        }
    }else{
       this->VerPrograma("");
    }
    Sensores->start();
}

pantallaPrincipal::~pantallaPrincipal()
{
    delete ui;
}

void pantallaPrincipal::abrirCargarPrograma(){
    cargarPrograma *CargarPrograma = new cargarPrograma;
    connect(CargarPrograma,SIGNAL(archivoChanged(QString)),this,SLOT(VerPrograma(QString)));
    CargarPrograma->show();
}

void pantallaPrincipal::VerPrograma(QString archivo){
    ui->programaEdit->clear();
    NombreArchivo = rutaProgramas + archivo;
    strList.clear();

    QFile file( NombreArchivo );

    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
       return;

    QTextStream in( &file );

    while (!in.atEnd()) {
        strList << in.readLine();
    }
    strList.removeFirst();

    ui->programaEdit->setText(archivo);
    ui->lineasEdit->clear();
    contLineas = 0;
    cantidadActual = 0;
    lecturaSensores();

}

void pantallaPrincipal::leerPrograma(QString Sen){
    QString line;
```

```
    qDebug() << "Sensor activo: " << Sen;
    error = 0;

    if(Sen == "DER"){
    //if (sender()->objectName() == "pushButton_2"){
        if(sensorIzq == 0 && contLineas == 0){
            sensorDer = 1 ;
            rowData = strList.at(contLineas).split(";");
            ui->cicloProgLabel->setText(QString::number(rowData[6].toInt()));
            ui->cicloReaLabel->setText("0");
        }else if(sensorIzq == 1){
            sensorDer = 1;
            sensorIzq = 0;
        }else if (sensorDer == 1 && sensorIzq == 0){
            error = 1;
            if(msgBox.isHidden()){
                msgBox.setWindowTitle(":: MENSAJE ::");
                msgBox.setText("DOBLE LECTURA DE SENSOR DERECHO");
                msgBox.exec();
            }
        }
    }
    if(Sen == "IZQ"){
    //if(sender()->objectName() == "pushButton_8"){
        if(sensorDer == 1){
            sensorIzq = 1 ;
            sensorDer = 0 ;
        }else if(sensorIzq == 1 && sensorDer == 0){
            error = 1;
            if(msgBox.isHidden()){
                msgBox.setWindowTitle(":: MENSAJE ::");
                msgBox.setText("DOBLE LECTURA DE SENSOR IZQUIERDO");
                msgBox.exec();
            }
        }
    }

    if(ui->programaEdit->text() != ""){
        if(((sensorDer == 1 || sensorIzq == 1) && error == 0)){
            if (cantidadActual < cantidadProg){
                if (sensorDer == 1) {
                     if(rowData[6].toInt() == contaCiclos){
                         contLineas++;
                         rowData = strList.at(contLineas).split(";");
                         ui->cicloProgLabel->setText(QString::number(rowData[6].toInt()));
                         contaCiclos = 0;
                         ui->cicloReaLabel->setText(QString::number(contaCiclos));
                         line = "Linea " + QString::number(contLineas+1) + " << " +
strList[contLineas];
                         definirSalidas(strList[contLineas], "<<");
                         ui->lineasEdit->setText(line);
                     }else{
                         line = "Linea " + QString::number(contLineas+1) + " << " +
strList[contLineas];
                         definirSalidas(strList[contLineas], "<<");
                         ui->lineasEdit->setText(line);
                     }
                }
                if(sensorIzq == 1){
                    contaCiclos++;
                    if(rowData[6].toInt() == contaCiclos){
                        contLineas++;
                        line = "Linea " + QString::number(contLineas+1) + " >> " +
strList[contLineas];
                        definirSalidas(strList[contLineas], ">>");
                        ui->lineasEdit->setText(line);
                        ui->cicloReaLabel->setText(QString::number(contaCiclos));
                    }else{
                        contLineas++;
                        ui->cicloReaLabel->setText(QString::number(contaCiclos));
```

```
                        line = "Linea " + QString::number(contLineas+1) + " >> " +
strList[contLineas];
                        definirSalidas(strList[contLineas], ">>");
                        ui->lineasEdit->setText(line);
                        contLineas--;
                    }
                }

                if(contLineas == strList.count() - 1){
                    contLineas = 0;
                    sensorIzq = 0;
                    contaCiclos = 0;
                    cantidadActual++;
                }

                ui->cantidadresEdit->setText(QString::number(cantidadActual));
            }else{
                QMessageBox msgBox;
                msgBox.setWindowTitle(":: MENSAJE ::");
                msgBox.setText("PRODUCCION TERMINADA");
                msgBox.exec();
                cantidadActual = 0;
                //ui->programaEdit->clear();
            }

        }else{
            qDebug() << "ERROR DE DOBLE LECTURA DE SENSOR";
            error = 0;
            return;
        }

    }else{
        QMessageBox msgBox;
        msgBox.setWindowTitle(":: MENSAJE ::");
        msgBox.setText("NO HAY UN PROGRAMA CARGADO");
        msgBox.exec();
    }

}


void pantallaPrincipal::on_pushButton_9_clicked()
{
    cantidad *cant = new cantidad;
    cant->definirCantidad(cantidadActual, cantidadProg, ui->programaEdit->text());
    connect(cant,SIGNAL(cantidadChanged(int)),this,SLOT(progCantidad(int)));
    cant->show();
}

void pantallaPrincipal::progCantidad(int valor){
    cantidadProg = valor;
    ui->cantidadprogEdit->setText(QString::number(cantidadProg));
    ui->cantidadresEdit->setText(QString::number(cantidadActual));
}

void pantallaPrincipal::on_pushButton_clicked()
{
    if(ui->programaEdit->text() != ""){
        modificarCiclos *ciclo = new modificarCiclos;
        ciclo->cargarCiclos(strList);
        connect(ciclo,SIGNAL(ciclosChanged(QStringList)),this,SLOT(progCiclos(QStringList)));
        ciclo->show();
    }else{
        QMessageBox msgBox;
        msgBox.setWindowTitle("MENSAJE");
        msgBox.setText("NO HAY UN PROGRAMA CARGADO");
        msgBox.exec();
    }
}

void pantallaPrincipal::configGPIO(){
```

```
    //CONFIGURACION PINES DE ENTRADA
    pinMode(5,INPUT); //SENSOR IZQUIERDO
    pinMode(8,INPUT); //SENSOR DERECHO

    //CONFIGURACION PINES DE SALIDA

    pinMode(7,OUTPUT); //LUZ

    pinMode(0,OUTPUT); //VEL1
    pinMode(2,OUTPUT); //VEL2
    pinMode(3,OUTPUT); //VEL3

    pinMode(21,OUTPUT); //GUI1
    pinMode(22,OUTPUT); //GUI2
    pinMode(23,OUTPUT); //GUI3
    pinMode(24,OUTPUT); //GUI4

    pinMode(25,OUTPUT);

    pinMode(1,OUTPUT); //llaDel
    pinMode(4,OUTPUT); //llaDelMed
    pinMode(5,OUTPUT); //llaTra
    pinMode(6,OUTPUT); //llaTraMed

    pinMode(26,OUTPUT); //ronDelDer
    pinMode(27,OUTPUT); //ronTraDer
    pinMode(28,OUTPUT); //ronDelIzq
    pinMode(29,OUTPUT); //ronTraIzq

}

void pantallaPrincipal::definirSalidas(QString data, QString sentido){
    qDebug() << data;
    salidas = data.split(";");

    //VELOCIDAD
    switch (salidas[0].toInt()) {
    case 1:
        vel1 = 0;
        vel2 = 0;
        vel3 = 0;
        break;
    case 2:
        vel1 = 1;
        vel2 = 0;
        vel3 = 0;
        break;
    case 3:
        vel1 = 1;
        vel2 = 1;
        vel3 = 0;
        break;
    case 4:
        vel1 = 0;
        vel2 = 1;
        vel3 = 0;
        break;
    case 5:
        vel1 = 0;
        vel2 = 1;
        vel3 = 1;
        break;
    case 6:
        vel1 = 1;
        vel2 = 1;
        vel3 = 1;
        break;
    case 7:
        vel1 = 0;
        vel2 = 0;
```

```
    vel3 = 1;
    break;
default:
    vel1 = 0;
    vel2 = 0;
    vel3 = 0;
    break;
}
//GUIA HILOS
switch (salidas[1].toInt()) {
case 1:
    gui1 = 1;
    gui2 = 0;
    gui3 = 0;
    gui4 = 0;
    break;
case 2:
    gui1 = 0;
    gui2 = 1;
    gui3 = 0;
    gui4 = 0;
    break;
case 3:
    gui1 = 0;
    gui2 = 0;
    gui3 = 1;
    gui4 = 0;
    break;
case 4:
    gui1 = 0;
    gui2 = 0;
    gui3 = 0;
    gui4 = 1;
    break;
default:
    gui1 = 0;
    gui2 = 0;
    gui3 = 0;
    gui4 = 0;
    break;
}

//LLAVE DELANTERA
switch (salidas[2].toInt()) {
case 0:
    llaDel = 0;
    llaDelMed = 0;
    break;
case 1:
    llaDel = 1;
    llaDelMed = 1;
    break;
case 6:
    llaDel = 0;
    llaDelMed = 1;
    break;
default:
    llaDel = 0;
    llaDelMed = 0;
    break;
}

//LLAVE TRASERA
switch (salidas[3].toInt()) {
case 0:
    llaTra = 0;
    llaTraMed = 0;
    break;
case 1:
    llaTra = 1;
    llaTraMed = 1;
```

```
            break;
        case 6:
            llaTra = 0;
            llaTraMed = 1;
            break;
        default:
            llaTra = 0;
            llaTraMed = 0;
            break;
        }
    if (sentido == "<<"){
        //RONDO DELANTERO DERECHO
        switch (salidas[4].toInt()) {
        case 0:
            ronDelDer = 0;
            break;
        case 1:
            ronDelDer = 1;
            break;
        default:
            ronDelDer = 0;
            break;
        }

        //RONDO TRASERO DERECHO
        switch (salidas[5].toInt()) {
        case 0:
            ronTraDer = 0;
            break;
        case 1:
            ronTraDer = 1;
            break;
        default:
            ronTraDer = 0;
            break;
        }
    }else{
        //RONDO DERECHO
        switch (salidas[4].toInt()) {
        case 0:
            ronDelIzq = 0;
            break;
        case 1:
            ronDelIzq = 1;
            break;
        default:
            ronDelIzq = 0;
            break;
        }

        switch (salidas[5].toInt()) {
        case 0:
            ronTraIzq = 0;
            break;
        case 1:
            ronTraIzq = 1;
            break;
        default:
            ronTraIzq = 0;
            break;
        }

    }

        digitalWrite(0,vel1);
        digitalWrite(2,vel2);
        digitalWrite(3,vel3);

        digitalWrite(21,gui1);
        digitalWrite(22,gui2);
        digitalWrite(23,gui3);
```

```
    digitalWrite(24,gui4);

    digitalWrite(1,llaDel);
    digitalWrite(4,llaDelMed);
    digitalWrite(5,llaTra);
    digitalWrite(6,llaTraMed);


    if (sentido == "<<"){
        digitalWrite(26,ronDelDer);
        digitalWrite(27,ronTraDer);
        digitalWrite(28,0);
        digitalWrite(29,0);
    }else{
        digitalWrite(28,ronDelIzq);
        digitalWrite(29,ronTraIzq);
        digitalWrite(26,0);
        digitalWrite(27,0);
    }
}


void pantallaPrincipal::on_pushButton_10_clicked(bool checked)
{
    if (checked){
        digitalWrite(7,1);
    }else{
        digitalWrite(7,0);
    }
}

void pantallaPrincipal::on_pushButton_11_clicked()
{
    //system("sudo poweroff");
    system("sudo halt");
}
void pantallaPrincipal::on_pushButton_3_clicked()
{
    Mantenimiento *m = new Mantenimiento;
    m->show();
}

void pantallaPrincipal::lecturaSensores(){
    //if(Sensores->isFinished()){
        Sensores = new ThreadSensores(this);
        connect(Sensores,SIGNAL(sensorChanged(QString)),this,SLOT(leerPrograma(QString)));
    //}
}
```

CLASE THREAD SENSORES

threadsensores.h

```
#ifndef THREADSENSORES_H
#define THREADSENSORES_H

#include <QThread>

class ThreadSensores : public QThread
{
    Q_OBJECT
public:
    explicit ThreadSensores(QObject *parent = 0);
    explicit ThreadSensores(QString *h);
    void run();
    bool Stop;
```

```
    void sleep(int valor);

signals:
    void sensorChanged(QString);

public slots:

private:
int S1,S2;
};

#endif // THREADSENSORES_H
```

threadsensores.cpp

```
#include "threadsensores.h"
#include <QtCore>
#include "qdebug.h"
#include "wiringPi.h"

ThreadSensores::ThreadSensores(QObject *parent) :
    QThread(parent)
{
}

void ThreadSensores::run(){
    S1 = 0;
    S2 = 0;

    for (;;){

        if(digitalRead(5) == 0 && S1==0){
            emit sensorChanged("IZQ");
            S1 = 1;
            S2 = 0;
        }else if(digitalRead(8) == 0 && S2==0){
            emit sensorChanged("DER");
            S2 = 1;
            S1 = 0;
        }else{
           msleep(10);
        }

    }
}
void ThreadSensores::sleep(int valor){
    msleep(valor);
}
```